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

User's Manual

μPD77210 Family

Digital Signal Processor

Architecture

μPD77210
μPD77213

Document No. U15807EJ2V0UM00 (2nd edition)
Date Published May 2002 N CP(K)

[MEMO]

NOTES FOR CMOS DEVICES

① PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

Note:

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

② HANDLING OF UNUSED INPUT PINS FOR CMOS

Note:

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

③ STATUS BEFORE INITIALIZATION OF MOS DEVICES

Note:

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

The export of this product from Japan is regulated by the Japanese government. To export this product may be prohibited without governmental license, the need for which must be judged by the customer. The export or re-export of this product from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

License not needed:	μPD77210F1-DA2, μPD77210GJ-8EN
The customer must judge the need for license:	μPD77213F1-×××-DA2, μPD77213GJ-×××-8EN

- **The information in this document is current as of May, 2002. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC's data sheets or data books, etc., for the most up-to-date specifications of NEC semiconductor products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC. NEC assumes no responsibility for any errors that may appear in this document.
- NEC does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC semiconductor products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC endeavours to enhance the quality, reliability and safety of NEC semiconductor products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC semiconductor products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment, and anti-failure features.
- NEC semiconductor products are classified into the following three quality grades:
  "Standard", "Special" and "Specific". The "Specific" quality grade applies only to semiconductor products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of a semiconductor product depend on its quality grade, as indicated below. Customers must check the quality grade of each semiconductor product before using it in a particular application.
  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots
  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support)
  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.
  The quality grade of NEC semiconductor products is "Standard" unless otherwise expressly specified in NEC's data sheets or data books, etc. If customers wish to use NEC semiconductor products in applications not intended by NEC, they must contact an NEC sales representative in advance to determine NEC's willingness to support a given application.
  (Note)
  (1) "NEC" as used in this statement means NEC Corporation and also includes its majority-owned subsidiaries.
  (2) "NEC semiconductor products" means any semiconductor product developed or manufactured by or for NEC (as defined above).

M8E 00.4

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

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

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

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

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

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

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

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

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

- **United Kingdom Branch**
  Milton Keynes, UK
  Tel: 01908-691-133
  Fax: 01908-670-290

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

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

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

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

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

J02.4

INTRODUCTION

Target Readers

This manual is intended for users who understand the functions of μPD77210 Family devices and who design application systems (hardware and software) using these products.

The μPD77210 Family is the general name for the μPD77210 and μPD77213 devices. Unless functional differences are otherwise specified, the descriptions in this manual apply to all μPD77210 Family devices. Particular product names are specified when functional differences exist.

Purpose

This manual's purpose is to help the user understand the hardware and software functions of the μPD77210 Family devices, as listed below, and to serve as a reference for development of hardware and software in systems using these devices.

Organization

This manual consists of the following chapters:

- CHAPTER 1 GENERAL
- CHAPTER 2 PIN FUNCTIONS
- CHAPTER 3 USE METHODS
- CHAPTER 4 ARCHITECTURE
- CHAPTER 5 PERIPHERALS
- CHAPTER 6 BOOT FUNCTIONS
- APPENDIX A INDEX

How to Read This Manual

This manual assumes that the reader has general knowledge of electricity, logic circuits, and microcomputers.

- For a general understanding of μPD77210 Family basic functions:
  → Start reading this manual from **CHAPTER 1 GENERAL**.

- For hardware engineers and software engineers:
  → Read this manual from **CHAPTER 1 GENERAL** to **CHAPTER 6 BOOT FUNCTIONS**. The device's various internal function blocks are described in **CHAPTER 4 ARCHITECTURE**. The functions related to peripherals such as interfaces, timers, and interrupts are described in **CHAPTER 5 PERIPHERALS**.

- For others who will use this as a reference manual:
  → Refer to the index at the end of the manual. Use the terms in the index as keywords for searching.

- **CHAPTER 3 USE METHODS** describes the overall use of μPD77210 Family devices, from power-on to execution of user programs. Be sure to read this chapter before using a μPD77210 Family device.

- For description of the instruction language used in μPD77210 Family devices, see the ***μPD77016 Family Architecture User's Manual***.

Conventions	Data significance:	Higher digits on the left and lower digits on the right
	Active low representation:	$\overline{\times\times\times}$ (pin or signal name is overlined)
	Note:	Footnote for item marked with **Note** in the text
	Caution:	Information requiring particular attention
	Remark:	Supplemental information
	Numeric notation:	Binary … ×××× or 0b××××
		Decimal … ××××
		Hexadecimal … 0x××××

Related Documents: The related documents listed below may include preliminary versions. However, preliminary versions are not marked as such.

Documents Related to μPD77210

Document Name / Part Number	Pamphlet	Data Sheet	User's Manual		Application Note
			Architecture	Instructions	Basic Software
μ PD77210	U12395E	U15203E	This manual	U13116E	U11958E
μ PD77213					

Documents Related to Development Tools

Document Name		Document No.
RX77016 User's Manual	Function	U14397E
	Configuration Tool	U14404E
RX77016 Application Note	HOST API	U14371E

Documents Related to Middleware

Document Name	Document No.
μSAP77016-B01 User's Manual	U13130E
μSAP77016-B03 User's Manual	U13373E
μSAP77016-B04 User's Manual	U13955E
μSAP77016-B05 User's Manual	U14497E
μSAP77016-B06 User's Manual	U15165E
μSAP77016-B07 User's Manual	U15134E
μSAP77016-B08 User's Manual	U15152E
μSAP77016-B11 User's Manual	U15683E

Caution The related documents listed above are subject to change without notice. Be sure to use the latest version of each document for designing.

CONTENTS

LIST OF FIGURES (1/3)

LIST OF FIGURES (2/3)

LIST OF FIGURES (3/3)

LIST OF TABLES (1/2)

LIST OF TABLES (2/2)

CHAPTER 1 GENERAL

The μPD77210 Family, which is the successor to the μPD77111 Family, is the general name used for the 16-bit fixed decimal digital signal processors (DSPs) μPD77210 and μPD77213.

Features such as high speed and low power consumption make these devices suitable not only for voice processing in various mobile application sets but also for processing of signals from all types of media including music and video.

Remark The μPD77111 Family is the general name for the μPD77110, 77111, 77112, 77113A, 77114, and 77115.

1.1 Comparison with μPD77111 Family

Because μPD77210 Family devices employ a 0.13-μm process, they achieve twice the execution speed of μPD77111 Family devices while consuming only half the power. The μPD77210 Family also feature expanded peripheral functions.

1.1.1 High-speed operation

The maximum operating frequency has been raised to the range of 120 MHz (in the μPD77213) to 160 MHz (in the μPD77210). This makes it possible to implement a multi-channel speech codec or speech/video codec on a single chip.

1.1.2 Low power consumption

The current consumption during execution of MAC operations (parallel load/store) and NOP operations is 0.35 mA/MHz. The operating voltage is 1.5 V (1.6 V when the μPD77210 operates between 120 and 160 MHz).

1.1.3 Expanded peripheral functions

The range of peripheral functions provided in μPD77111 Family devices has been expanded in μPD77210 Family devices. The expanded functions and additional peripherals found in μPD77210 Family devices include the following.

- 8-/16-bit host interface (expanded function: supports 16-bit bus)
- General-purpose I/O ports (expanded function: up to 16 ports can be used)
- On-chip 8-channel DMA function (new)
- Time division multiplexing (TDM) serial interface (new)
- 32-/64-bit audio serial interface (new)
- 16-bit timer (new)
- SD (Secure Digital) card interface[Note]

Note μPD77213 only

1.2 Features

The μPD77210 Family is compatible with the μPD77111 Family's operations and instruction set. The μPD77210 Family also maintains binary level compatibility with the μPD77111 Family's software and middleware (except for parts that depend on the memory configuration and peripheral configuration).

1.2.1 DSP core kernel

Functions

- Parallel processing using dual load/store
- Hardware loop
- Execution of conditional instructions
- Execution of product sum operation in one instruction cycle
- On-chip JTAG-compliant functions

Programming

- 16 bits × 16 bits + 40 bits = 40-bit multiply accumulator
- Eight 40-bit general-purpose registers
- Eight data memory pointer registers (four each for the X and Y memory spaces)
- 12 levels of interrupts (four factors are assigned to each level for expanded interrupt factors)
- 3-operand instructions (trinomial operations)
- No pipeline at execution stage

Memory space (space from perspective DSP core kernel) [Note]

- Instruction memory 32-bit width × 16-bit words
- X data memory 16-bit width × 16-bit words
- Y data memory 16-bit width × 16-bit words

Note Not all of the logically available space has been implemented.

1.2.2 Peripheral block

Functions

- 8-channel DMA function
  Peripheral memory transfer (PMT) function
- 1 Mword × 16 bits SRAM interface
- Time division multiplexing (TDM) serial interface (TSIO)
- Audio serial interface (ASIO)
- 8-/16-bit host interface (HIO)
- General-purpose I/O port (PIO)
- 16-bit timer
- Expanded interrupt functions
- Peripheral circuit standby function
- Instruction correction function
- Memory space expansion function (paging function)
- PLL multiplication and division control functions
- SD card interface (μPD77213 only)

Memory space (space from paging function) [Note]

- Instruction memory 32-bit width × 16-bit words × 64 pages
- X data memory 16-bit width × 16-bit words × 64 pages
- Y data memory 16-bit width × 16-bit words × 64 pages
- DMA data memory 16-bit width × 16-bit words
- External data memory 16-bit width × 20-bit words

Note Not all of the logically available space has been implemented.

Table 1-1. Features of μPD77210 Family Devices

		μPD77210	μPD77213
Instruction cycle		6.25 ns	8.33 ns
Operating clock frequency (maximum)		160 MHz	120 MHz
Clock circuit	PLL multiplier circuit	× 10 to 32 (×2 steps), × 40 to 64 (×8 steps)	
	Divider output	÷ integer of 1 to 16	
Parallel instruction execution		Trinomial operations and parallel load/store instructions, binomial operations and parallel load/store instructions, monomial operations and parallel load/store instructions, register-to-register transfers and conditional instructions, branch and conditional instructions	
Hardware loop		Nesting is possible up to four levels.	
Conditional instructions		Conditional operations, conditional transfers, and conditional branching are enabled by combining independent instructions with other instructions.	
Multiply accumulator		16 bits × 16 bits + 40 bits → 40 bits	
Accumulator		40-bit I/O (binomial operations and monomial operations)	
General-purpose registers		Eight 40-bit registers	
Data memory pointers		Eight: four each for X memory space and Y memory space	
Interrupts		12 vectors (up to four levels of factors per vector)	
Three-stage pipeline control		Instruction fetch, instruction decode, instruction execution	
Instruction memory	System ROM	512 words (for boot function)	
	Internal RAM	31.5 Kwords	15.5 Kwords
	Internal ROM	None	64 Kwords
	External area	None	
Data memory	X internal RAM	30 Kwords	18 Kwords
	X internal ROM	None	32 Kwords
	Y internal RAM	30 Kwords	18 Kwords
	Y internal ROM	None	32 Kwords
	External area (X and Y)	1 Mword	1 Mword (8 Kwords when using SD card)
Serial interface		Audio serial (× 1)	
		TDM serial (× 1)	
		Standard serial (× 2) (shared as audio/TDM)	
Host interface		8-bit/16-bit parallel	
General-purpose I/O ports		Up to 16	
Timer		Two-channel (16-bit resolution)	
Memory card interface		None	SD card (1 bit)
DMA transfer function		8 channels, between peripherals and internal data RAM	
Power supply	DSP core	1.425 to 1.65 V, 1.55 to 1.65 V (when the μPD77210 operates between 120 and 160 MHz)	
	External I/O	2.7 to 3.6 V	
Standby modes		By HALT instruction or STOP instruction	
Package		144-pin LQFP, 161-pin FBGA	
Other		Debug function (JTAG)	

1.2.3 Ordering information

Part Number	Package
μPD77210F1-DA2	161-pin plastic fine pitch BGA (10 × 10)
μPD77210GJ-8EN	144-pin plastic LQFP (fine pitch) (20 × 20)
μPD77213F1-×××-DA2	161-pin plastic fine pitch BGA (10 × 10)
μPD77213GJ-×××-8EN	144-pin plastic LQFP (fine pitch) (20 × 20)

Remark ××× indicates ROM code number.

CHAPTER 2 PIN FUNCTIONS

This chapter describes the pin connections and pin functions of the μPD77210 Family. The pin names are shown below.

ASCK:	Audio serial clock input/output
ASI:	Audio serial data input
ASIEN:	Audio serial input enable
ASO:	Audio serial data output
ASOEN:	Audio serial output enable
BCLK:	Bit Clock input/output
CLKIN:	Clock input
CLKOUT:	Clock output
CSTOP:	Clear stop mode
EV_{DD}:	Power supply for I/O pins
GND:	Ground
HALTS:	Halt status signal output
HD0 to HD15:	Host data bus
$\overline{HCS}$:	Host chip select
HA0, HA1:	Host data access
$\overline{HRD}$:	Host read
$\overline{HRE}$:	Host read enable
$\overline{HWE}$:	Host write enable
$\overline{HWR}$:	Host write
I.C.:	Internal connection
IV_{DD}:	Power supply for DSP core
$\overline{INTmn}$:	Interrupt (m, n: 0 to 3)
LRCLK:	Left right clock input/output
MA0 to MA19:	External data memory address bus
$\overline{MBSTB}$:	External data memory bus strobe
MCLK:	Master clock input
MD0 to MD15:	External data memory bus
$\overline{MHOLDAK}$:	External data memory bus hold acknowledge
$\overline{MHOLDRQ}$:	External data memory bus hold request
$\overline{MRD}$:	External data memory read output
$\overline{MWR}$:	External data memory write output
$\overline{MWAIT}$:	External data memory access wait input
NC:	Non-connection
P0 to P15:	Port
PLL0 to PLL3:	PLL multiple rate set
Reserved:	Reserved
$\overline{RESET}$:	Reset
SDCLK:	SD card clock output
SDCR:	SD card command output/response input
SDDAT0:	SD card data input
SDMON:	SD card access monitor output
STOPS:	Stop status signal output
TCK:	Test clock input
TDI:	Test data input
TDO:	Test data output
TICE:	Test in-circuit emulator
TIMOUT:	Timer time out monitor output
TMS:	Test mode select
$\overline{TRST}$:	Test reset
TSCK:	Time division multiplex serial clock input
TSI:	Time division multiplex serial data input
TSIAK:	Time division multiplex serial input acknowledge
TSIEN:	Time division multiplex serial input enable
TSO:	Time division multiplex serial data output
TSOEN:	Time division multiplex serial data output enable
TSORQ:	Time division multiplex serial output request

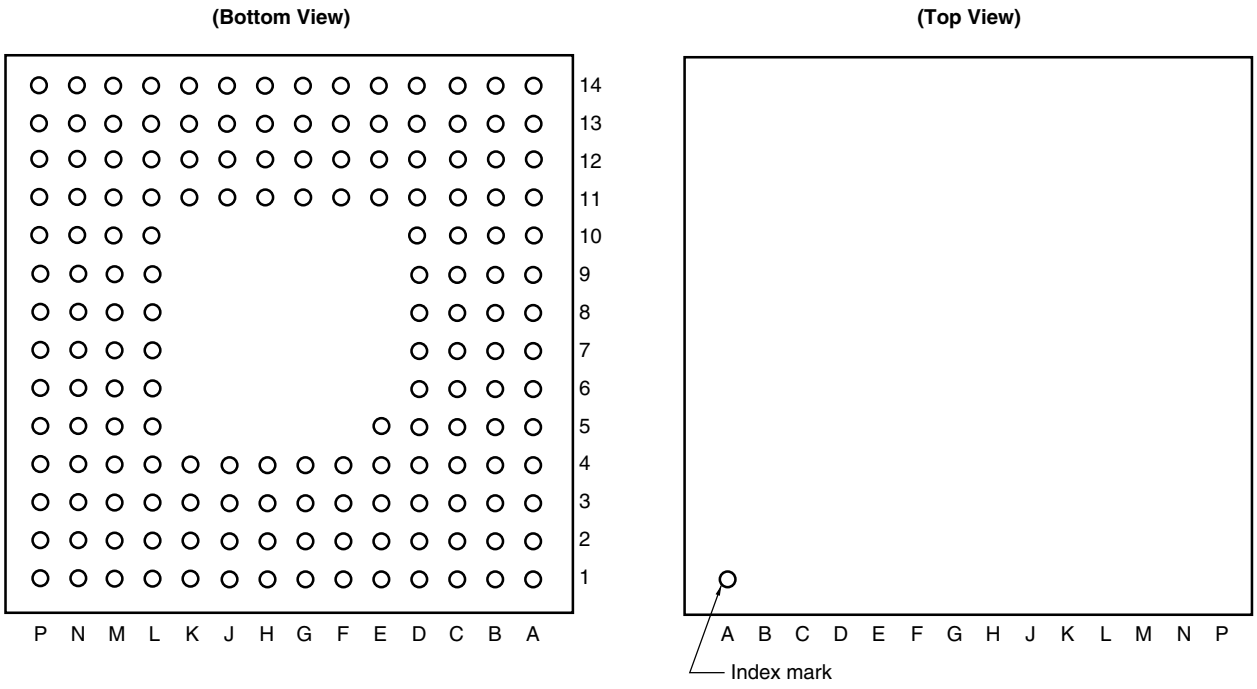
2.1 Pin Connection

The μPD77213 pins are connected in the same way as the μPD77210 except that some external memory pins are shared with the SD card interface.

2.1.1 161-pin plastic fine pitch BGA (10 × 10)

- μPD77210F1-DA2
- μPD77213F1-×××-DA2

Figure 2-1. 161-Pin Plastic Fine Pitch BGA



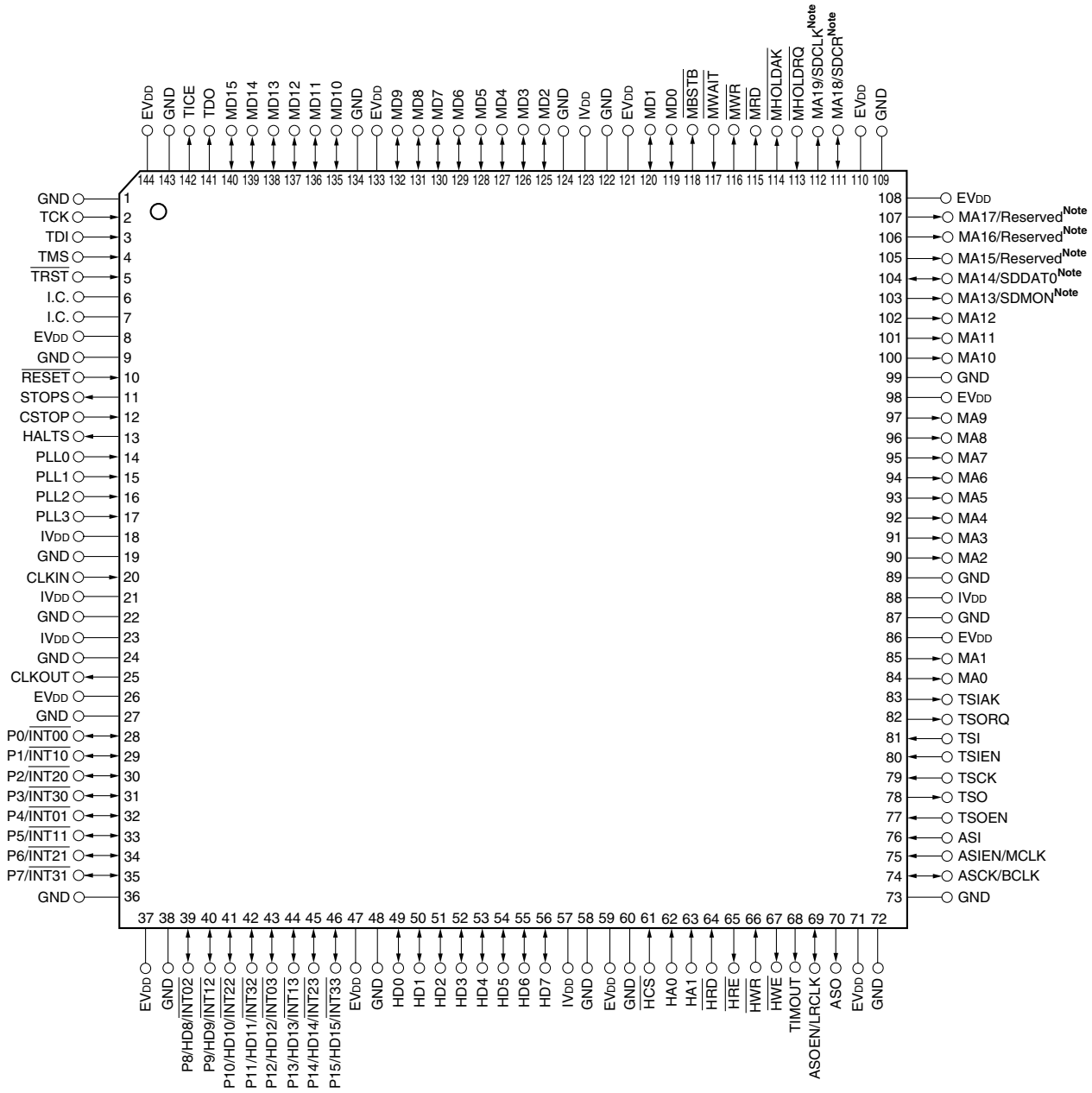
Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name
A1	NC	C14	EV_{DD}	H2	HD7	M5	TSORQ
A2	NC	D1	P10/HD10/$\overline{INT22}$	H3	HD6	M6	MA0
A3	P5/$\overline{INT11}$	D2	P11/HD11/$\overline{INT32}$	H4	GND	M7	MA4
A4	P2/$\overline{INT20}$	D3	P12/HD12/$\overline{INT03}$	H11	MD5	M8	MA5
A5	GND	D4	GND	H12	MD4	M9	MA10
A6	EV_{DD}	D5	GND	H13	MD1	M10	MA12
A7	IV_{DD}	D6	P1/$\overline{INT10}$	H14	MD3	M11	MA15/Reserved[Note]
A8	IV_{DD}	D7	GND	J1	EV_{DD}	M12	MA19/SDCLK[Note]
A9	PLL0	D8	GND	J2	$\overline{HCS}$	M13	MA18/SDCR[Note]
A10	STOPS	D9	GND	J3	HA1	M14	EV_{DD}
A11	EV_{DD}	D10	GND	J4	$\overline{HWR}$	N1	NC
A12	$\overline{TRST}$	D11	TMS	J11	GND	N2	NC
A13	NC	D12	TICE	J12	MD0	N3	ASIEN/MCLK
A14	NC	D13	MD12	J13	$\overline{MBSTB}$	N4	TSCK
B1	NC	D14	MD15	J14	IV_{DD}	N5	TSIAK
B2	NC	E1	P14/HD14/$\overline{INT23}$	K1	HA0	N6	MA1
B3	P7/$\overline{INT31}$	E2	P15/HD15/$\overline{INT33}$	K2	$\overline{HRD}$	N7	MA2
B4	P6/$\overline{INT21}$	E3	P13/HD13/$\overline{INT13}$	K3	TIMOUT	N8	MA7
B5	P3/$\overline{INT30}$	E4	GND	K4	ASO	N9	MA9
B6	CLKOUT	E5	NC	K11	GND	N10	MA11
B7	IV_{DD}	E11	GND	K12	$\overline{MWR}$	N11	MA16/Reserved[Note]
B8	PLL3	E12	MD14	K13	$\overline{MWAIT}$	N12	MA17Reserved[Note]
B9	PLL1	E13	MD9	K14	EV_{DD}	N13	NC
B10	CSTOP	E14	MD11	L1	$\overline{HWE}$	N14	NC
B11	I.C.	F1	EV_{DD}	L2	$\overline{HRE}$	P1	NC
B12	TCK	F2	HD1	L3	GND	P2	NC
B13	NC	F3	HD2	L4	GND	P3	ASI
B14	NC	F4	HD0	L5	TSIEN	P4	TSO
C1	EV_{DD}	F11	MD10	L6	GND	P5	TSI
C2	P8/HD8/$\overline{INT02}$	F12	MD13	L7	GND	P6	EV_{DD}
C3	P9/HD9/$\overline{INT12}$	F13	MD7	L8	MA8	P7	IV_{DD}
C4	P4/$\overline{INT01}$	F14	EV_{DD}	L9	GND	P8	MA3
C5	P0/$\overline{INT00}$	G1	HD3	L10	MA14/SDDAT0[Note]	P9	MA6
C6	CLKIN	G2	HD5	L11	GND	P10	EV_{DD}
C7	PLL2	G3	HD4	L12	$\overline{MHOLDRQ}$	P11	MA13/SDMON[Note]
C8	HALTS	G4	GND	L13	$\overline{MRD}$	P12	EV_{DD}
C9	$\overline{RESET}$	G11	GND	L14	$\overline{MHOLDAK}$	P13	NC
C10	I.C.	G12	MD8	M1	EV_{DD}	P14	NC
C11	TDI	G13	MD2	M2	ASCK/BCLK		
C12	TDO	G14	MD6	M3	ASOEN/LRCLK		
C13	GND	H1	IV_{DD}	M4	TSOEN		

Note MA13 to MA19 pins of the μPD77213 are alternate function pins.

2.1.2 144-pin plastic LQFP (fine pitch) (20 × 20)

- μPD77210GJ-8EN
- μPD77213GJ-×××-8EN

Figure 2-2. 144-Pin Plastic LQFP



Note MA13 to MA19 pins of the μPD77213 are alternate function pins.

Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name
1	GND	37	EV_{DD}	73	GND	109	GND
2	TCK	38	GND	74	ASCK/BCLK	110	EV_{DD}
3	TDI	39	P8/HD8/$\overline{INT02}$	75	ASIEN/MCLK	111	MA18/SDCR[Note]
4	TMS	40	P9/HD9/$\overline{INT12}$	76	ASI	112	MA19/SDCLK[Note]
5	$\overline{TRST}$	41	P10/HD10/$\overline{INT22}$	77	TSOEN	113	$\overline{MHOLDRQ}$
6	I.C.	42	P11/HD11/$\overline{INT32}$	78	TSO	114	$\overline{MHOLDAK}$
7	I.C.	43	P12/HD12/$\overline{INT03}$	79	TSCK	115	$\overline{MRD}$
8	EV_{DD}	44	P13/HD13/$\overline{INT13}$	80	TSIEN	116	$\overline{MWR}$
9	GND	45	P14/HD14/$\overline{INT23}$	81	TSI	117	$\overline{MWAIT}$
10	$\overline{RESET}$	46	P15/HD15/$\overline{INT33}$	82	TSORQ	118	$\overline{MBSTB}$
11	STOPS	47	EV_{DD}	83	TSIAK	119	MD0
12	CSTOP	48	GND	84	MA0	120	MD1
13	HALTS	49	HD0	85	MA1	121	EV_{DD}
14	PLL0	50	HD1	86	EV_{DD}	122	GND
15	PLL1	51	HD2	87	GND	123	IV_{DD}
16	PLL2	52	HD3	88	IV_{DD}	124	GND
17	PLL3	53	HD4	89	GND	125	MD2
18	IV_{DD}	54	HD5	90	MA2	126	MD3
19	GND	55	HD6	91	MA3	127	MD4
20	CLKIN	56	HD7	92	MA4	128	MD5
21	IV_{DD}	57	IV_{DD}	93	MA5	129	MD6
22	GND	58	GND	94	MA6	130	MD7
23	IV_{DD}	59	EV_{DD}	95	MA7	131	MD8
24	GND	60	GND	96	MA8	132	MD9
25	CLKOUT	61	$\overline{HCS}$	97	MA9	133	EV_{DD}
26	EV_{DD}	62	HA0	98	EV_{DD}	134	GND
27	GND	63	HA1	99	GND	135	MD10
28	P0/$\overline{INT00}$	64	$\overline{HRD}$	100	MA10	136	MD11
29	P1/$\overline{INT10}$	65	$\overline{HRE}$	101	MA11	137	MD12
30	P2/$\overline{INT20}$	66	$\overline{HWR}$	102	MA12	138	MD13
31	P3/$\overline{INT30}$	67	$\overline{HWE}$	103	MA13/SDMON[Note]	139	MD14
32	P4/$\overline{INT01}$	68	TIMOUT	104	MA14/SDDAT0[Note]	140	MD15
33	P5/$\overline{INT11}$	69	ASOEN/LRCLK	105	MA15/Reserved[Note]	141	TDO
34	P6/$\overline{INT21}$	70	ASO	106	MA16/Reserved[Note]	142	TICE
35	P7/$\overline{INT31}$	71	EV_{DD}	107	MA17/Reserved[Note]	143	GND
36	GND	72	GND	108	EV_{DD}	144	EV_{DD}

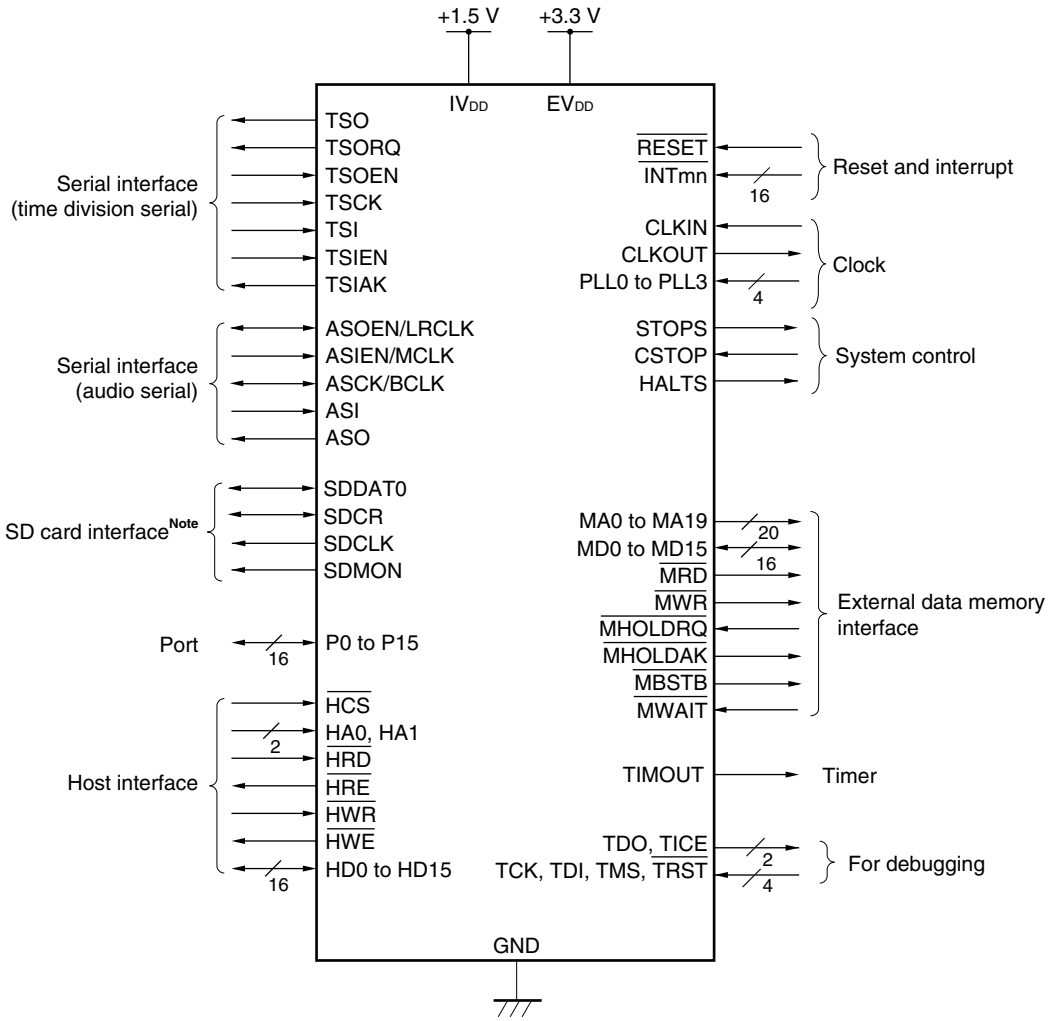
Note MA13 to MA19 pins of the μPD77213 are alternate function pins.

2.2 Pin Configuration

Figure 2-3 shows the pin connections of the μPD77210 family, classifying the pins by function.

The pin configuration of the μPD77213 is the same as the μPD77210 except that an SD card interface has been added.

Figure 2-3. Pin Configuration



Note μPD77213 only

Caution Some port pins, host interface pins, serial interface pins, interrupt pins, and SD card interface pins are alternate function pins.

Remark m, n = 0 to 3

2.3 Pin Functions

(1) Power supply pins

Pin Name	Pin No.		I/O	Function	Alternate-Function Pin
	144-Pin LQFP	161-Pin FBGA			
IV_{DD}	18, 21, 23, 57, 88, 123	A7, A8, B7, H1, J14, P7	–	Power supply for DSP core (+1.5 V) These pins supply power to the DSP core.	–
EV_{DD}	8, 26, 37, 47, 59, 71, 86, 98, 108, 110, 121, 133, 144	A6, A11, C1, C14, F1, F14, J1, K14, M1, M14, P6, P10, P12	–	Power supply for I/O (+3.3 V) These pins supply power to the external interface pins.	–
GND	1, 9, 19, 22, 24, 27, 36, 38, 48, 58, 60, 72, 73, 87, 89, 99, 109, 122, 124, 134, 143	A5, C13, D4, D5, D7, D8, D9, D10, E4, E11, G4, G11, H4, J11, K11, L3, L4, L6, L7, L9, L11	–	Ground These are ground pins.	–

Remark Please supply voltage to the IV_{DD} and EV_{DD} pins simultaneously.

(2) Clock and system control pins

Pin Name	Pin No.		I/O	Function	Alternate-Function Pin
	144-Pin LQFP	161-Pin FBGA			
CLKIN	20	C6	Input	Clock input This pin inputs a clock to operate the μPD77210 Family.	–
CLKOUT	25	B6	Output	Internal system clock output This pin outputs the internal system clock that is the clock input from CLKIN and which is multiplied by the PLL circuit.	–
PLL0 to PLL3	14 to 17	A9, B9, C7, B8	Input	PLL multiple setting input These pins set a clock multiple of the PLL circuit. • PLL3: PLL2: PLL1: PLL0 0000: ×10 0001: ×12 0010: ×14 0011: ×16 0100: ×18 0101: ×20 0110: ×22 0111: ×24 1000: ×26 1001: ×28 1010: ×30 1011: ×32 1100: ×40 1101: ×48 1110: ×56 1111: ×64	–
HALTS	13	C8	Output	HALT mode status output This pin is asserted active in halt mode and stop mode.	–
STOPS	11	A10	Output	Stop mode status output This pin is asserted active in stop mode.	–
CSTOP	12	B10	Input	Stop mode clear signal input Stop mode is cleared when this pin is asserted active.	–

(3) Reset and interrupt pins

Pin Name	Pin No.		I/O	Function	Alternate-Function Pin
	144-Pin LQFP	161-Pin FBGA			
$\overline{\text{RESET}}$	10	C9	Input	Internal system reset signal input This pin initializes the μPD77210 Family.	–
$\overline{\text{INT00}}$	28	C6	Input	Maskable external interrupt input These pins input external interrupts.	P0
$\overline{\text{INT01}}$	32	C4	Input		P4
$\overline{\text{INT02}}$	39	C2	Input		P8/HD8
$\overline{\text{INT03}}$	43	D3	Input		P12/HD12
$\overline{\text{INT10}}$	29	D6	Input		P1
$\overline{\text{INT11}}$	33	A3	Input		P5
$\overline{\text{INT12}}$	40	C3	Input		P9/HD9
$\overline{\text{INT13}}$	44	E3	Input		P13/HD13
$\overline{\text{INT20}}$	30	A4	Input		P2
$\overline{\text{INT21}}$	34	B4	Input		P6
$\overline{\text{INT22}}$	41	D1	Input		P10/HD10
$\overline{\text{INT23}}$	45	E1	Input		P14/HD14
$\overline{\text{INT30}}$	31	B5	Input		P3
$\overline{\text{INT31}}$	35	B3	Input		P7
$\overline{\text{INT32}}$	42	D2	Input		P11/HD11
$\overline{\text{INT33}}$	46	E2	Input		P15/HD15

(4) External data memory interface

Pin Name	Pin No.		I/O	Function	Alternate-Function Pin
	144-Pin LQFP	161-Pin FBGA			
MA0 to MA19[Note]	84, 85, 90 to 97, 100 to 107, 111, 112	M6, N6, N7, P8, M7, M8, P9, N8, L8, N9, M9, N10, M10, P11, L10, M11, N11, N12, M13, M12	Output (3S)	Address bus of external data memory These pins output an address when the external data memory is accessed.	SDCLK, SDCR, SDDAT0, SDMON
MD0 to MD15	119,120, 125 to 132, 135 to 140	J12, H13, G13, H14, H12, H11, G14, F13, G12, E13, F11, E14, D13, F12, E12, D14	I/O (3S)	16-bit data bus These pins input/output data when the external data memory is accessed.	–
$\overline{\text{MWR}}$	116	K12	Output (3S)	Write output This pin outputs a write strobe signal for the external data memory.	–
$\overline{\text{MRD}}$	115	L13	Output (3S)	Read output This pin outputs a read strobe signal for the external data memory.	–
$\overline{\text{MHOLDAK}}$	114	L14	Output	Hold acknowledge signal This pin goes low when the external device is granted use of the external data memory bus of the μPD77210 Family.	–
$\overline{\text{MHOLDRQ}}$	113	L12	Input	Hold request signal The external device inputs a low level to this pin when it uses the external data memory bus of the μPD77210 Family.	–
$\overline{\text{MWAIT}}$	117	K13	Input	Wait signal input This pin inserts wait cycles when the μPD77210 Family accesses the external data memory. • 0: Inserts wait cycles. • 1: Does not insert wait cycles.	–
$\overline{\text{MBSTB}}$	118	J13	Output	Bus strobe signal This pin goes low while the μPD77210 Family uses the external data memory bus.	–

Note MA13 to MA19 pins of the μPD77213 are alternate function pins.

Remark Those pins marked “3S” in the above table enter the high-impedance state under the following conditions:

MA0 to MA19, $\overline{\text{MRD}}$, and $\overline{\text{MWR}}$: When the bus is released ($\overline{\text{MHOLDAK}}$ = low level)

MD0 to MD15: When the external data memory is not accessed and when the bus is released ($\overline{\text{MHOLDAK}}$ = low level)

(5) Timer

Pin Name	Pin No.		I/O	Function	Alternate-Function Pin
	144-Pin LQFP	161-Pin FBGA			
TIMOUT	68	K3	Output	Time out monitor This pin is asserted active when the timer times out.	–

(6) Serial interface

Pin Name	Pin No.		I/O	Function	Alternate-Function Pin
	144-Pin LQFP	161-Pin FBGA			
ASCK/ BCLK	74	M2	I/O	Audio serial clock input ASCK: Audio serial clock input BCLK: Serial clock I/O	–
ASO	70	K4	Output (3S)	Audio serial data output	–
ASI	76	P3	Input	Audio serial data input	–
ASOEN/ LRCLK	69	M3	I/O	Audio serial output enable/left right clock input ASOEN: Audio serial output enable input LRCLK: Left right clock I/O	–
ASIEN/ MCLK	75	N3	Input	Audio serial input enable/master clock input output ASIEN: Audio serial input enable input MCLK: Master clock input (in master mode)	–
TSCK	79	N4	Input	Clock input for time division serial	–
TSO	78	P4	Output (3S)	Time-division serial data output	–
TSI	81	P5	Input	Time-division serial data input	–
TSORQ	82	M5	Output	Time-division serial output request	–
TSOEN	77	M4	Input	Time-division serial output enable	–
TSIEN	80	L5	Input	Time-division serial input enable	–
TSIAK	83	N5	Output	Time-division serial input acknowledge	–

Remark Those pins marked “3S” in the above table enter the high-impedance state when data transmission is completed and when the hardware reset ($\overline{\text{RESET}}$) signal is input.

(7) Host interface

Pin Name	Pin No.		I/O	Function	Alternate-Function Pin
	144-Pin LQFP	161-Pin FBGA			
HA1	63	J3	Input	Host address 1 This pin specifies a register that is accessed by the host interface pins (HD7 to HD0, or HD15 to HD0). • 1: The host interface status register (HST) is accessed. • 0: The host transmit data register (HDT (out)) is accessed for read ($\overline{HRD}$ = 0) and the host receive data register (HDT (in)) is accessed for write ($\overline{HWR}$ = 0).	–
HA0	62	K1	Input	Host address 0 This pin specifies a register that is accessed by HD7 to HD0 in 8-bit mode. This pin is invalid in 16-bit mode. • 1: Bits 15 to 8 of HST, HDT (in), and HDT (out) are accessed. • 0: Bits 7 to 0 of HST, HDT (in), and HDT (out) are accessed.	–
$\overline{HCS}$	61	J2	Input	Chip select input	–
$\overline{HRD}$	64	K2	Input	Host read input	–
$\overline{HWR}$	66	J4	Input	Host write input	–
$\overline{HRE}$	65	L2	Output	Host read enable output	–
$\overline{HWE}$	67	L1	Output	Host write enable output	–
HD0 to HD7	49 to 56	F4, F2, F3, G1, G3, G2, H3, H2	I/O (3S)	8-bit host data bus These pins constitute a host data bus in 8-bit host mode. Access to 16-bit data for input/output is controlled by the HA0 pin, and the data is accessed two times such that it is divided into two blocks of 8-bit data. In 16-bit mode, the lower 8 bits of the data are input/output.	–
HD8 to HD15	39 to 46	C2, C3, D1, D2, D3, E3, E1, E2	I/O (3S)	Host data bus These pins constitute a host data bus in 16-bit host mode. They input/output 16-bit data with HD0 to HD7.	P8 to P15/ $\overline{INT02}$, $\overline{INT12}$, $\overline{INT22}$, $\overline{INT32}$, $\overline{INT03}$, $\overline{INT13}$, $\overline{INT23}$, $\overline{INT33}$

Remark Those pins marked “3S” in the above table enter the high-impedance state while the host interface is not being accessed.

(8) I/O port

Pin Name	Pin No.		I/O	Function	Alternate-Function Pin
	144-Pin LQFP	161-Pin FBGA			
P0	28	C5	I/O	General-purpose I/O port	$\overline{\text{INT00}}$
P1	29	D6	I/O		$\overline{\text{INT10}}$
P2	30	A4	I/O		$\overline{\text{INT20}}$
P3	31	B5	I/O		$\overline{\text{INT30}}$
P4	32	C4	I/O		$\overline{\text{INT01}}$
P5	33	A3	I/O		$\overline{\text{INT11}}$
P6	34	B4	I/O		$\overline{\text{INT21}}$
P7	35	B3	I/O		$\overline{\text{INT31}}$
P8	39	C2	I/O		$\overline{\text{INT02}}$/HD8
P9	40	C3	I/O		$\overline{\text{INT12}}$/HD9
P10	41	D1	I/O		$\overline{\text{INT22}}$/HD10
P11	42	D2	I/O		$\overline{\text{INT32}}$/HD11
P12	43	D3	I/O		$\overline{\text{INT03}}$/HD12
P13	44	E3	I/O		$\overline{\text{INT13}}$/HD13
P14	45	E1	I/O		$\overline{\text{INT23}}$/HD14
P15	46	E2	I/O		$\overline{\text{INT33}}$/HD15

(9) Debugging interface

Pin Name	Pin No.		I/O	Function	Alternate-Function Pin
	144-Pin LQFP	161-Pin FBGA			
TDO	141	C12	Output (3S)	For debugging These interface pins are used when a debugger is used.	–
TICE	142	D12	Output		–
TCK	2	B12	Input		–
TDI	3	C11	Input		–
TMS	4	D11	Input		–
$\overline{\text{TRST}}$	5	A12	Input		–

Remark Those pins marked “3S” in the above table enter the high-impedance state while the debugging interface is not being accessed.

(10) Others

Pin Name	Pin No.		I/O	Function	Alternate-Function Pin
	144-Pin LQFP	161-Pin FBGA			
I.C.	6, 7	B11, C10	–	Internally connected. Leave these pins open.	–
NC	–	A1, A2, A13, A14, B1, B2, B13, B14, E5, N1, N2, N13, N14, P1, P2, P13, P14	–	No connection. Leave these pins open.	–

Caution If any signal is input to these pins or if these pins are read, the correct operation of the μPD77210 Family is not guaranteed.

(11) SD card interface (μPD77213 only)

Pin Name	Pin No.		I/O	Function	Alternate-Function Pin
	144-Pin LQFP	161-Pin FBGA			
SDCLK	112	M12	Output	SD card clock output • Leave this pin open.	MA19
SDCR	111	M13	I/O (3S)	SD cord command/response Input: Response Output: Command • Leave pull-up.	MA18
SDDAT0	104	L10	I/O (3S)	SD card data input/output Input: Read data Output: Write data • Leave pull-up.	MA14
SDMON	103	P11	Output	SD card interface access monitor This pin outputs a high level when the SD card interface is being accessed. 1: SD card interface being accessed 0: SD card interface not being accessed	MA13
Reserved	105 to 107	M11, N11, N12	–	Reserved for future function expansion. This pin becomes high impedance when the SD card interface is being used.	MA15 to MA17

Remark Those pins marked “3S” in the above table enter the high-impedance state when the SD card interface is not being accessed.

2.4 Connection of Unused Pins

Connect the unused pins as shown in the table below.

Table 2-1. Connection of Functional Pins

Pin Name	I/O	Recommended Connection
STOPS, HALTS	Output	Leave open.
CSTOP	Input	Connect to GND via a pull-down resistor.
CLKOUT	Output	Leave open.
P0 to P15	I/O	Connect to EV_{DD} via a pull-up resistor or to GND via a pull-down resistor.
HD0 to HD7[Note 1]	I/O	Connect to EV_{DD} via a pull-up resistor or to GND via a pull-down resistor.
HA0, HA1	Input	Connect to EV_{DD} via a pull-up resistor or to GND via a pull-down resistor.
$\overline{HCS}$, $\overline{HRD}$, $\overline{HWR}$	Input	Connect to EV_{DD} via a pull-up resistor.
$\overline{HRE}$, $\overline{HWE}$	Output	Leave open.
TIMOUT	Output	Leave open.
ASCK, TSCK	Input	Connect to EV_{DD} via a pull-up resistor or to GND via a pull-down resistor.
ASI, TSI	Input	
ASIEN, TSIEN	Input	Connect to GND via a pull-down resistor.
ASOEN, TSOEN, LRCLK	Input	
ASO, TSO	Output	Leave open.
SORQ	Output	
TSIAK	Output	
MA0 to MA19	Output	Leave open.
MD0 to MD15[Note 2]	I/O	Connect to EV_{DD} via a pull-up resistor or to GND via a pull-down resistor.
$\overline{MRD}$, $\overline{MWR}$	Output	Leave open.
$\overline{MHOLDRQ}$	Input	Connect to EV_{DD} via a pull-up resistor.
$\overline{MBSTB}$, $\overline{MHOLDAK}$	Output	Leave open.
$\overline{MWAIT}$	Input	Connect to EV_{DD} via a pull-up resistor.
TCK	Input	Connect to GND via a pull-down resistor.
TDO, TICE	Output	Leave open.
TMS, TDI	Input	Leave open (this pin is internally pulled up).
$\overline{TRST}$	Input	Leave open (this pin is internally pulled down).

Notes 1. These pins may be left open if $\overline{HCS}$, $\overline{HRD}$, and $\overline{HWR}$ are fixed to the high level. However, connect these pins as recommended in the HALT and STOP modes when the power consumption must be lowered.

2. These pins may be left open if the external data memory is not accessed in the program. However, connect these pins as recommended in the HALT and STOP modes when the power consumption must be lowered.

Caution The alternate-function pins not shown in this table should be connected in the same manner as the other alternate-function pins. When selecting the function of an alternate-function pin by selecting a software mode, select the default function.

Table 2-2. Connection of Non-Functional pin

Pin Name	I/O	Recommended Connection
I.C.	–	Leave open.
NC	–	Leave open.

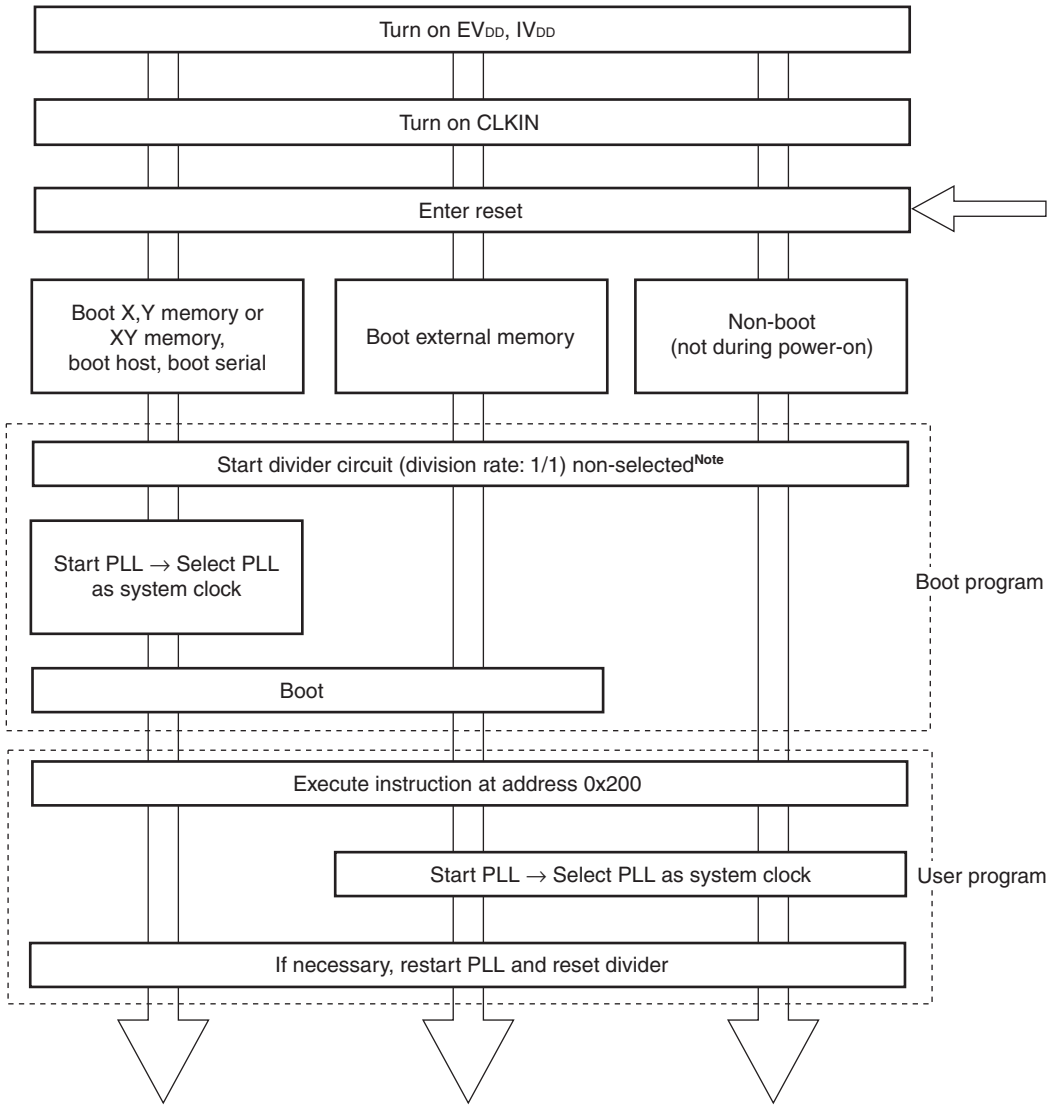
CHAPTER 3 USE METHODS

This chapter describes the flow of operations from power-on to actual execution of user program code in the μPD77210 Family device, and also covers the general use of this device. For details, see the various function descriptions.

3.1 From Power-on to Program Execution

The flow of operations from power-on to execution of a user program is shown below.

Figure 3-1. From Power-On to User Program Execution



Note The divider circuit is started but the divider circuit's output is not selected as the system clock.

3.1.1 Power-on

μPD77210 Family devices have two power sources: the DSP core's power supply (IV_{DD}) and the I/O power supply (EV_{DD}). IV_{DD} is a 1.5-V power supply while EV_{DD} is a 3-V power supply. We recommend that both should be started at the same time.

After the power is turned on, the clock is applied to the CLKIN pin, the PLL's multiply rate settings are applied to the PLL0 to PLL3 pins, the boot mode settings are applied to the P0 to P2 pins, and the PLL's lock range setting is applied to pin P3.

3.1.2 Reset

Once the rated power supply voltage has been reached, if input to the reset pin remains at low level for six consecutive clock cycles, the device is reset (initialized). Make sure that the clock input to the CLKIN pin is applied for at least six clock cycles. If a reset does not occur, be cautious about bus sharing with external resources since the internal registers' status has not been initialized and the status of pins has not been confirmed. Also, at this stage the PLL does not operate.

3.1.3 Boot

After the μPD77210 Family device has been reset, the user program is executed starting at address 0x200. However, since address 0x200 is a RAM address in μPD77210 Family devices, its value after power-on is undefined and it therefore cannot be used to execute instructions. Consequently, a boot operation must be executed to write instructions to RAM addresses starting with 0x200.

The boot operation is performed via a boot routine (stored in on-chip ROM) after the reset has been released. There are several boot modes, depending on the type of transfer destination for the instruction code (op code), and the mode is determined by the values at pins P0 to P2 during execution of the boot routine.

A boot operation following a reset (i.e., an initial reset boot operation) involves only the instruction area from address 0x200 to 0x7FFF. To boot other instruction areas, other reboot routines must be executed as part of a program. A user program must be employed to boot data areas (or a limited reboot routine can be used).

Before the boot operation occurs, the boot ROM starts the PLL and automatically switches the clock source to PLL output[Note]. A lockup time is required between starting the PLL and switching the clock source, and during this time there are approximately 5,000 cycles of the input clock. Although there is no reason for the user to be concerned with the lockup time, the lockup time is added to the boot time. Once the clock source has been switched, the μPD77210 Family device operates using the PLL clock and the boot operation is executed.

Note If external memory boot is selected, the PLL is not automatically started. Instead, the boot operation is processed using an externally supplied clock. For details, see **3.3 Clock Settings After Boot Operation**.

3.1.4 Program execution

After the boot routine has been completed, the μPD77210 Family device starts executing instructions starting at address 0x200.

3.2 Standby Mode

When the DSP program is idle, two standby modes are available as a means of reducing current consumption. An instruction is executed to switch to a standby mode.

Execute a HALT instruction to switch to HALT mode. This will automatically switch the clock source to the divider. If the clock source has already been switched to the divider, there is no change in the operating clock. Various types of interrupts (non-masked interrupts only) can return the operation mode to normal mode.

Execute a STOP instruction to switch to STOP mode. This masks the system clock and stops operation of the DSP. There is no way to automatically stop the PLL. To stop the PLL, switch the clock source to the input clock and stop the PLL before executing the STOP instruction. When a signal is applied to the CSTOP pin, the operation mode will be reset to normal mode.

3.3 Clock Settings After Boot Operation

In the following cases, the PLL is not started automatically when a boot operation is executed, so the PLL must be started by a user program after completing the boot operation.

- Non-boot
- External memory boot

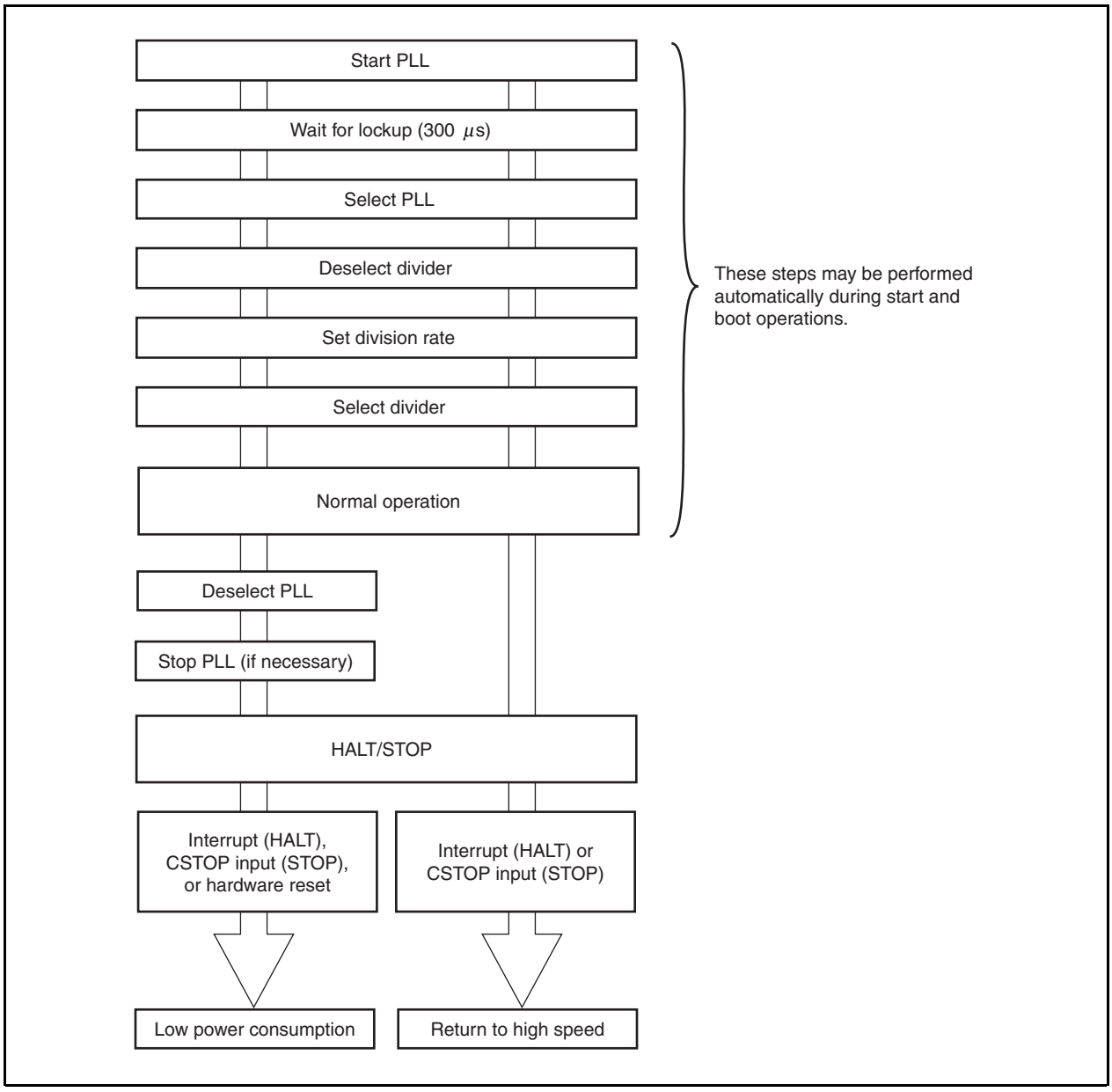
In these cases, during a boot operation following power-on the PLL is not automatically started by the boot ROM. Instead, the clock that is input from CLKIN operates as the direct system clock. To use the PLL multiplied clock as the clock source, the program must execute processing to start the PLL, wait for the PLL lockup, and then switch the clock source to PLL output.

The clock control register (CLKC) is used to start the PLL and switch the clock source. After the PLL has been started, a nop instruction (or other instruction) is used to calculate the timing for the PLL lockup time (300 μs), then the clock source is switched to PLL output.

In addition, CLKC can be used to set and select the divider. It can also be used to select a divided clock based on the clock input from the CLKIN pin or the PLL output clock.

The flow of operations from starting to stopping the PLL is shown below.

Figure 3-2. From Starting to Stopping the PLL



CHAPTER 4 ARCHITECTURE

This chapter describes the architecture of μPD77210 Family devices.

4.1 Overall Block Configuration

This section divides the physical structure of the μPD77210 Family into several functional blocks for explanation. The μPD77210 Family consists of the following internal units:

- Buses (main bus, X data bus, and Y data bus)
  Refer to **4.2 Buses**.
- System control units
  Refer to **4.3 System Control Units**.
- Program control unit
  Refer to **4.4 Program Control Unit**.
- Data addressing unit
  Refer to **4.5 Data Addressing Unit**.
- Operation unit
  Refer to **4.6 Operation Unit**.
- Peripheral unit
  Refer to **CHAPTER 5 PERIPHERALS**.

Figure 4-1 illustrates the block organization. Refer to the corresponding sections for the functions of the respective blocks.

Figure 4-1. Overall Block Configuration

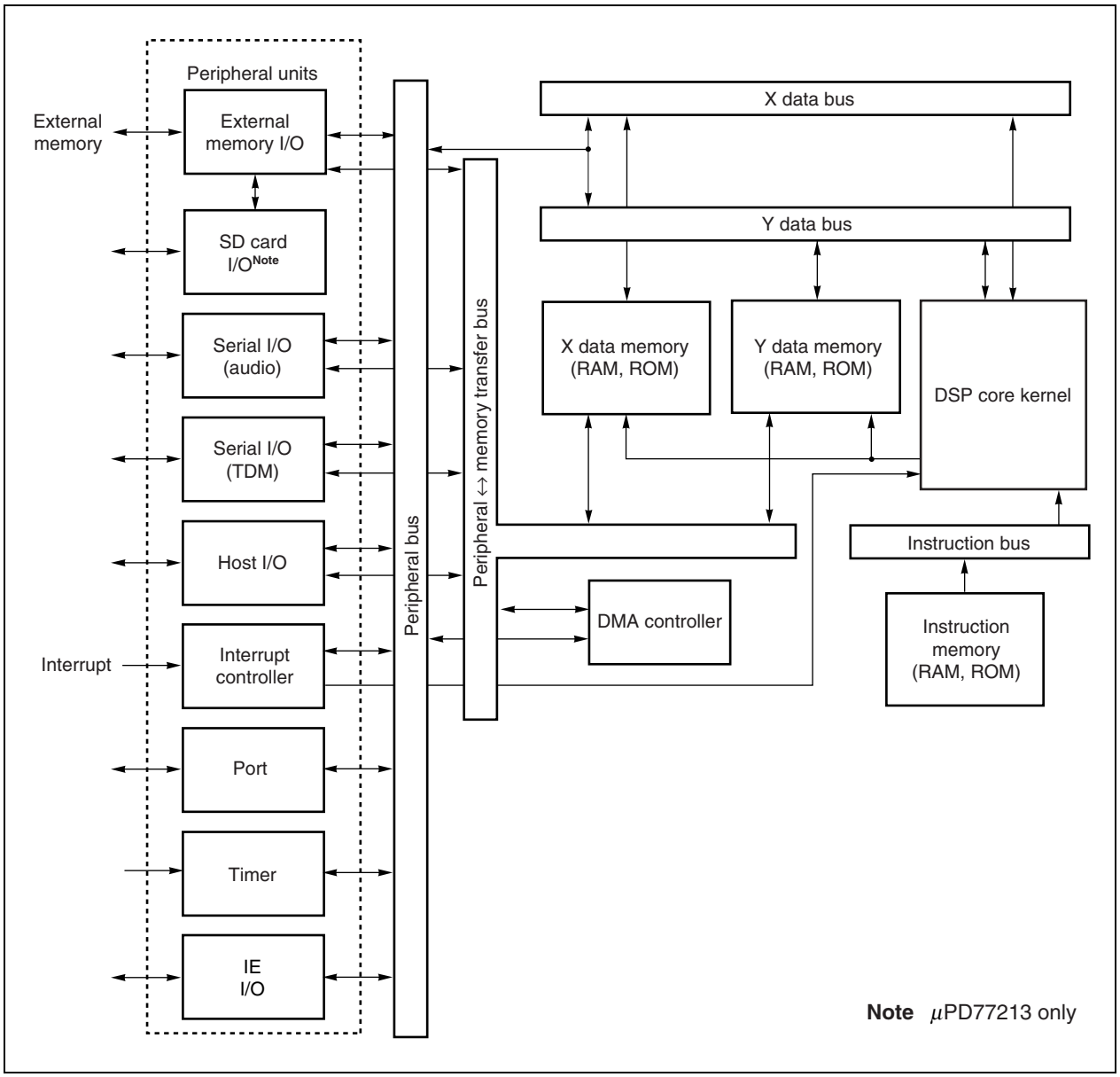
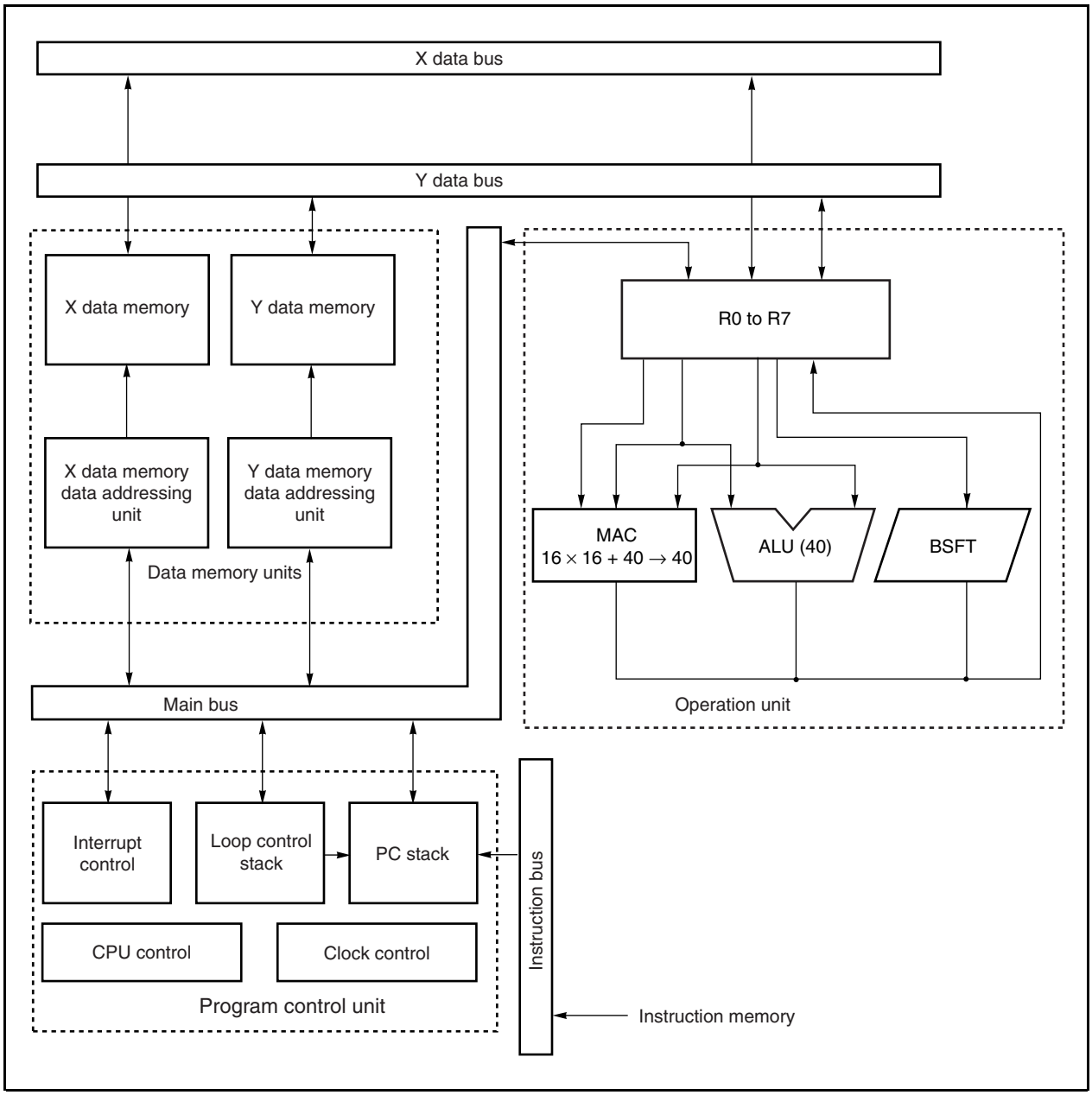


Figure 4-2. DSP Core Kernel



4.2 Buses

A bus transfers data between external devices and the processor. The μPD77210 Family is provided with the following four types of buses:

- Main bus
- X data bus
- Y data bus
- Peripheral ↔ memory transfer bus

4.2.1 Main bus

(1) Function

This 16-bit bus connects the general-purpose registers (R0 to R7) and control registers, etc. It transfers data when the following instructions are executed:

- Register-to-register transfer instruction
  This instruction transfers data between the L part of a general-purpose register and a non-general-purpose register. These registers are listed in Table 4-1. Note that this instruction transfers only the L part of a general-purpose register.
  For details, refer to ***μPD77016 Family Instructions User's Manual***.

  Caution A general-purpose register consists of 40 bits. These 40 bits are divided into three parts: L (lower 16 bits), H (16 bits in the middle), and E (higher 8 bits). For details, refer to 4.6.2 General-purpose registers and data formats.

- Immediate value setting instruction
  This instruction sets immediate data to a specified register. Of the registers listed in Table 4-1, the following can be specified.

  - General-purpose registers (L part (R0L to R7L) only)
  - Data pointers (DP0 to DP7)
  - Index registers (DN0 to DN7)
  - Modulo registers (DMX, DMY)

For the details of this instruction, refer to ***μPD77016 Family Instructions User's Manual***.

(2) Registers connected to main bus

The table shown below lists the registers connected to the main bus.

Table 4-1. Registers Connected to Main Bus

Register Name	Assembler-Reserved Name	Load (L)/Store (S)
General-purpose register	R0L to R7L (L part of R0 to R7)	L/S
Data pointer	DP0 to DP7	
Index register	DN0 to DN7	
Modulo register	DMX, DMY	
Stack	STK	
Stack pointer	SP	
Loop counter	LC	
Loop stack (LSTK)	LSR1, LSR2, LSR3	
Loop stack pointer	LSP	
Status register	SR	
Interrupt enable flag stack register	EIR	
Error status register	ESR	

4.2.2 Data bus

(1) Function

This 16-bit bus connects the general-purpose registers, X and Y data memories, and internal peripherals. It transfers data when the following instructions are executed.

- Parallel load/store instruction
- Partial load/store instruction
- Direct addressing load/store instruction
- Immediate value index load/store instruction

For the details of the load/store instruction, refer to **μPD77016 Family Instructions User's Manual**.

The data bus is classified into X data bus, Y data bus, and peripheral bus. The logical and physical relations among these buses are shown in Table 4-2.

Table 4-2. Functional Block and Bus

Functional Block	Relations Among X Data Bus, Y Data Bus, and Peripheral Bus
Internal memory peripherals	X and Y data buses are logically and physically separated. Therefore, both the X and Y data buses are validated for transfer by a single instruction.
Internal peripheral	X and Y data buses are logically and physically connected. Even when a peripheral-related register is accessed from X or Y memory space, therefore, the same peripheral register is accessed as long as the address is the same. At this time, however, the peripheral register cannot be accessed simultaneously from the X and Y data memory spaces with a single instruction.
External memory	Although the X and Y data buses are logically separated, they are physically common. Therefore, the X and Y external memories cannot be accessed simultaneously with a single instruction.

(2) X data bus

This 16-bit bus connects the general-purpose registers, X data memory, and the bus from the internal peripherals. This bus transfers data when the following instructions are executed:

- Parallel load/store instruction (for X data memory)
- Partial load/store instruction (for X data memory)
- Direct addressing load/store instruction (for X data memory)
- Immediate value index load/store instruction (for X data memory)

Cautions 1. Although the X and Y data buses are separated inside the device, a single data bus is commonly used externally. Thus, an instruction that accesses both external memories cannot be executed in the same instruction cycle.

2. The same peripheral register is accessed for internal peripheral regardless of whether the X or Y data memory is accessed, as long as the address is the same.

3. Even in the case of 2 above, a peripheral register cannot be accessed from both the X and Y memory spaces in the same instruction cycle.

The table shown below shows the registers and memories connected to the X data bus.

Table 4-3. Registers and Memories Connected to X Data Bus

Register/Memory Name	Assembler-Reserved Name	Load (L)/Store (S)
General-purpose register	R0 to R7 R0E to R7E R0H to R7H R0L to R7L R0EH to R7EH	L/S
X internal RAM	–	
X internal ROM		From ROM to bus only
External memory		L/S
Internal peripheral		

Caution A general-purpose register consists of 40 bits. These 40 bits are divided into three parts: L (lower 16 bits), H (16 bits in the middle), and E (higher 8 bits). Any of these parts can be specified for transfer. For details, refer to 4.6.2 General-purpose registers and data formats.

(3) Y data bus

This 16-bit bus connects the general-purpose registers, Y data memory, and the bus from the internal peripherals. This bus transfers data when the following instructions are executed:

- Parallel load/store instruction (for Y data memory)
- Partial load/store instruction (for Y data memory)
- Direct addressing load/store instruction (for Y data memory)
- Immediate value index load/store instruction (for Y data memory)

Cautions 1. Although the X and Y data buses are separated inside the device, a single data bus is commonly used externally. Thus, an instruction that accesses both external memories cannot be executed in the same instruction cycle.

2. The same peripheral register is accessed for internal peripheral units regardless of whether the X or Y data memory is accessed, as long as the address is the same.

3. Even in the case of 2 above, a peripheral register cannot be accessed from both the X and Y memory spaces in the same instruction cycle.

Table 4-4 shows the registers and memories connected to the Y data bus.

Table 4-4. Registers and Memories Connected to Y Data Bus

Register/Memory Name	Assembler-Reserved Name	Load (L)/Store (S)
General-purpose register	R0 to R7 R0E to R7E R0H to R7H R0L to R7L R0EH to R7EH	L/S
Y internal RAM	–	
Y internal ROM		From ROM to bus only
External memory		L/S
Internal peripheral		

Caution A general-purpose register consists of 40 bits. These 40 bits are divided into three parts: L (lower 16 bits), H (16 bits in the middle), and E (higher 8 bits). Any of these parts can be specified for transfer. For details, refer to 4.6.2 General-purpose registers and data formats.

4.2.3 Peripheral ↔ memory transfer bus

This bus is connected to internal peripherals and internal data memory and is used for DMA transfers between the two. Background transfers that are not routed through the DSP core kernel are supported.

For details, see **5.8 Peripheral Memory Transfer (PMT)**.

4.3 System Control Units

The following basic functions, which support the digital signal processor operations of the μPD77210 Family, are called system control units:

- Clock generator
- Reset function
- Pipeline architecture
- Standby function

4.3.1 Clock generator

The clock generator is a circuit that generates and controls the system clock that is supplied to the CPU. It consists of a PLL, divider, and clock controller.

The internal system clock is generated from an external clock that is input via the CLKIN pin. This system clock provides the reference timing within the device. At the same time, the internal system clock can be output via the CLKOUT pin for use in tracking the timing of synchronization with external devices (also, this function can be invalidated by register settings). A PLL and output divider are included to provide multiplied and divided clocks supplied to blocks within the device. The PLL and output divider can be started and stopped by a user program.

A reset sets the PLL and output divider to active mode. Immediately after a reset, the internal system clock becomes the input clock (input via the CLKIN pin). At the start of a boot operation, the PLL output clock is selected as the internal system clock (although some boot modes do not start the PLL or select the PLL output clock). For details of the relationship between boot modes and PLL activation, see **CHAPTER 6 BOOT FUNCTIONS**.

External pins (PLL0 to PLL3) are used to set the PLL multiplication rate m. The range of specifiable values is 10 (multiply by 10) to 64(multiply by 64), and the relationship between pin settings and multiplication rates is shown in Table 4-5. The range of values that can be specified via register settings for the output division rate n is 1/1 (divide by 1) to 1/16 (divide by 16).

Table 4-5. PLL Multiplication Rate Settings

Pin Setting				Multiplication Rate (m)
PLL3	PLL2	PLL1	PLL0	
0	0	0	0	10
0	0	0	1	12
0	0	1	0	14
0	0	1	1	16
0	1	0	0	18
0	1	0	1	20
0	1	1	0	22
0	1	1	1	24
1	0	0	0	26
1	0	0	1	28
1	0	1	0	30
1	0	1	1	32
1	1	0	0	40
1	1	0	1	48
1	1	1	0	56
1	1	1	1	64

The clock that is input to the PLL controller is multiplied from 10 to 64 times. Make sure that the multiplied clock frequency is specified within the PLL lock frequency range stipulated in the specifications. The PLL lock frequency ranges from 80 MHz to 160 MHz. As is shown in Table 4-6, there are two lock ranges. After reset, use pin P3 to specify one of these lock ranges.

The output divider divides the clock input to the divider by an integer ranging from 1 to 16 (i.e., 1/1 to 1/16). Make sure that the frequency that is ultimately supplied within the DSP (as the externally input clock times “m/n”) is within the operating frequency range stipulated in the specifications for μPD77210 Family devices.

Table 4-6. PLL Lock Range Settings

Pin Setting	PLL Lock Range
P3	
0	120 to 160 MHz
1	80 to 120 MHz

During a standby mode (HALT mode or STOP mode), the divider’s output clock is automatically selected as the internal system clock’s source. Therefore, the divider must be activated before executing standby mode. If the divider’s output clock has been selected as the internal system clock’s source for normal operation, there is no switching of clock sources when entering standby mode. The clock source used prior to entering standby mode is used again when recovering from standby mode.

Register settings can be made to specify either “Output” or “Do not output” for output of the internal system clock via the CLKOUT pin.

For details of clock control, see **5.12 Clock Controller (CLKC)**.

4.3.2 Reset function

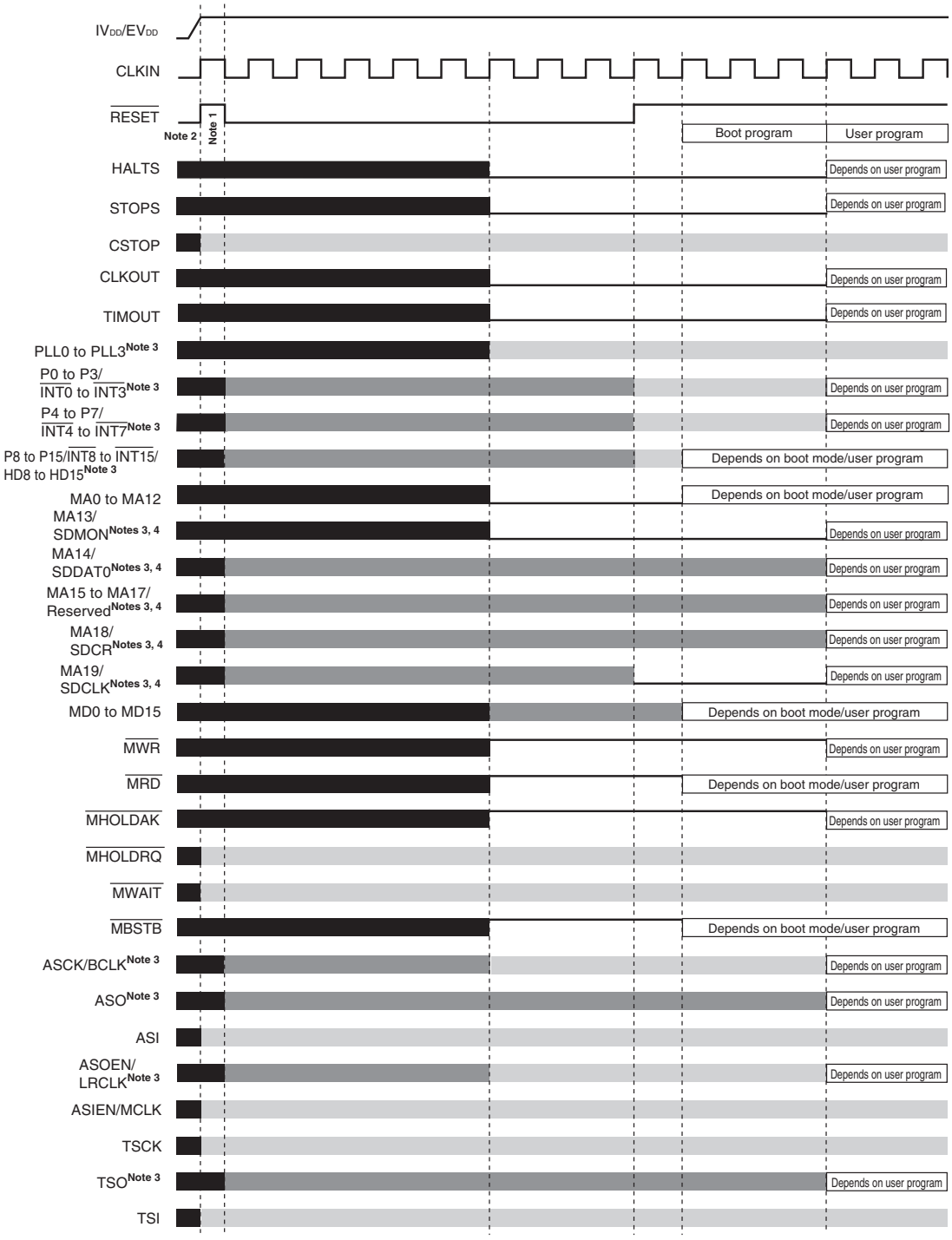
(1) Hardware initialization

The device can be given a hardware reset by holding the signal input to the $\overline{\text{RESET}}$ pin active (low level) for a certain period (at least six cycles of the clock input to the CLKIN pin). The purpose of this type of reset is to ensure that the device has been properly initialized before executing a program. A reset must be performed to initialize the status of pins and the device’s internal status. If a reset is not performed during power-on, the status of pins is uncertain and, in the worst case, this can affect the operation of other devices that are connected.

The registers and signal pins that are initialized and their initial values are listed in Tables 4-7 to 4-9, and the timing of the reset operation is shown in Figure 4-3.

After a reset, if the $\overline{\text{RESET}}$ pin is deasserted inactive (high level), a boot operation is performed according to the settings of pins P0 to P2. For details of how the values of various pins and registers change according to the type of boot operation, see **CHAPTER 6 BOOT FUNCTIONS**. At the same time, the PLL’s lock range is specified according to the value set to pin P3. The values of pins P0 to P3 must remain stable from three clocks prior to release of reset until the boot operation has been completed.

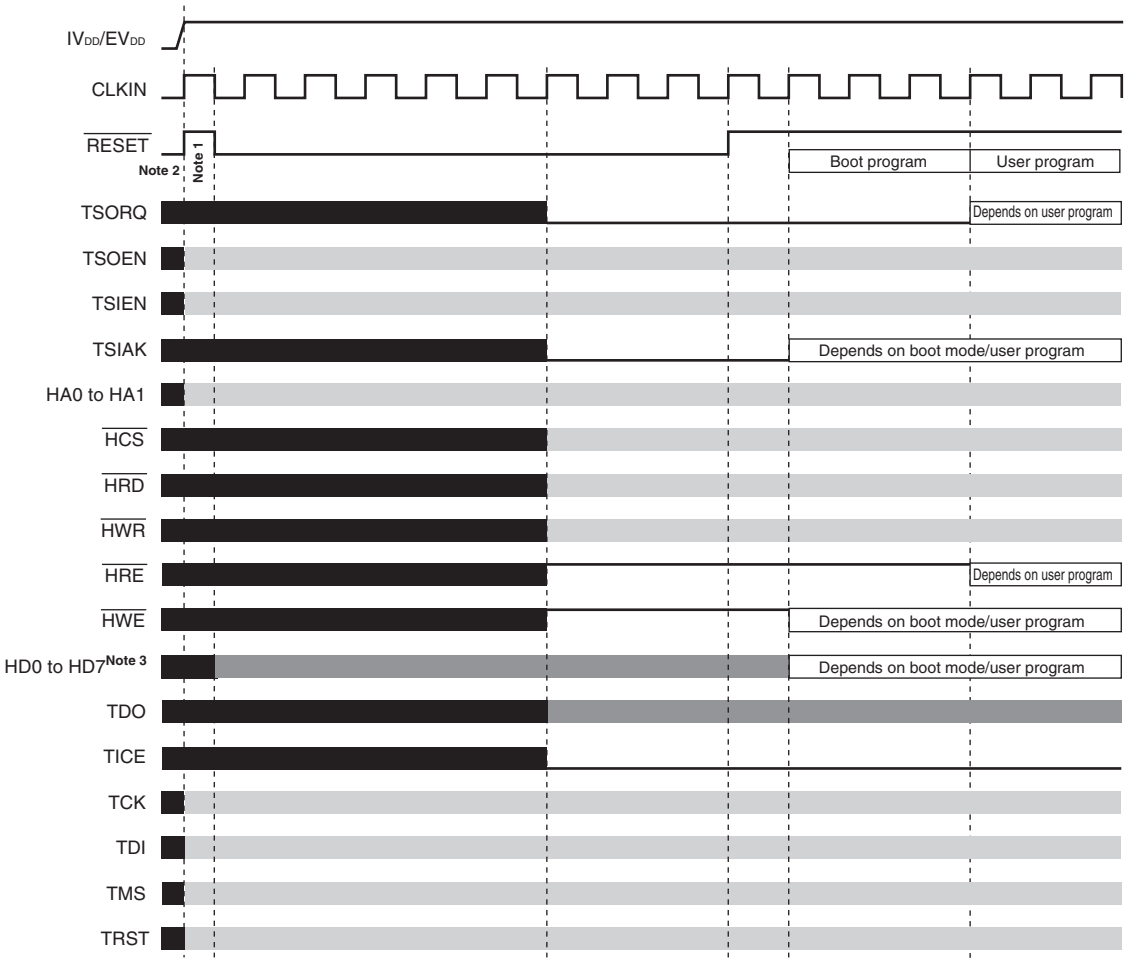
Figure 4-3. Reset Operation Timing (1/2)



Notes 1. RESET can remain at low level.
2. Make sure that the input voltage for each pin does not exceed EVDD.
3. In the μPD77210, the pin status is undefined until six clock cycles have been input to the CLKIN pin.
4. In the μPD77210, pins MA13 to MA19 are the same as pins MA0 to MA12.

Remark Undefined status High impedance Input status (high impedance)

Figure 4-3. Reset Operation Timing (2/2)



Notes 1. RESET can remain at low level.

2. Make sure that the input voltage for each pin does not exceed EV_{DD}.

3. In the μPD77210, the pin status is undefined until six clock cycles have been input to the CLKIN pin.

Remark Undefined status High impedance Input status (high impedance)

Table 4-7. CPU Registers to Be Initialized and Their Initial Values

Register Name	Initial Value
SR	0xF000
PC	0
SP	0
LC	0b1xxx xxxx xxxx xxxx
LSP	0
RC	0b1xxx xxxx xxxx xxxx
EIR	0xFFFF
ESR	0

Table 4-8. Peripheral Registers to Be Initialized and Their Initial Values

Register Name	Initial Value
SST1	0x00C2
TSST	0x0005
TFMT	0x0000
TTXL, TTXH, TRXL, TRXH	0xFFFF
SST2	0x0002
ASST	0x8012
HST	0x0301
MSHW	0x0000
MCST	0x0000
MWAIT	0xFFFF
PMC0 to PMC7	0x0000
PCD0 to PCD3	0x0000
ICR0 to ICR11	0x0000
TCSR0, TCSR1	0x0000
CFER	0x0000

Table 4-9. Pins to Be Initialized and Their Initial Values

Pin Name	Initial Value
MA0 to MA19	Low level output[Notes 1, 2]
MD0 to MD15	High impedance
$\overline{\text{MWR}}$, $\overline{\text{MRD}}$, $\overline{\text{MBSTB}}$	High level output[Note 1]
TSORQ, TSIAK	Low level output
ASO, TSO	High impedance[Note 2]
ASCK/BCLK, ASOEN/LRCLK	High impedance[Note 2]
HD0 to HD7	High impedance[Note 2]
$\overline{\text{HWE}}$, $\overline{\text{HRE}}$	High level output
STOPS, HALTS	Low level output
CLKOUT	Low level output
TIMOUT	Low level output
P0 to P15	High impedance (input status)[Note 2]
TICE	Low level output

Notes 1. High impedance mode is set when the bus is released ($\overline{\text{MHOLDAK}}$ = 0). When 0 is set for $\overline{\text{MHOLDRQ}}$, the bus can be released even during a reset.

2. In the μPD77213, high impedance mode is set when $\overline{\text{RESET}}$ = 0 (except for MA0 to MA13). Initial status is set when the reset is released after input of the specified number of clocks. In the μPD77210, the status is undefined until it is initialized by input of the clock.

4.3.3 Pipeline architecture

The μPD77210 Family employs pipeline architecture to enhance the execution speed. Generally, one instruction completes its processing via several machine cycles each of which performs elemental processing. The instructions of the μPD77210 Family have the following three machine cycles:

F: Instruction fetch cycle
Reads an op code from the instruction memory.
D: Decode cycle
Decodes the read op code.
E: Execution cycle
Executes the decoded result.

The part that executes each machine cycle is called a pipeline stage. Each stage independently completes processing with the same number of clocks (1 cycle). Therefore, an instruction under execution enters stages one after another without wait time. In addition, three instructions can exist in the respective three stages at the same time. In other words, it seems as if one instruction were processed with the execution time of one stage as long as the instruction passes through the pipelines without any instruction stream fault. The number of clock cycles in one stage is called one instruction cycle.

The minimum instruction cycle in the μPD77210 is 6.25 ns (when operating at 160 MHz), and in the μPD77213 it is 8.33 ns (when operating at 120 MHz).

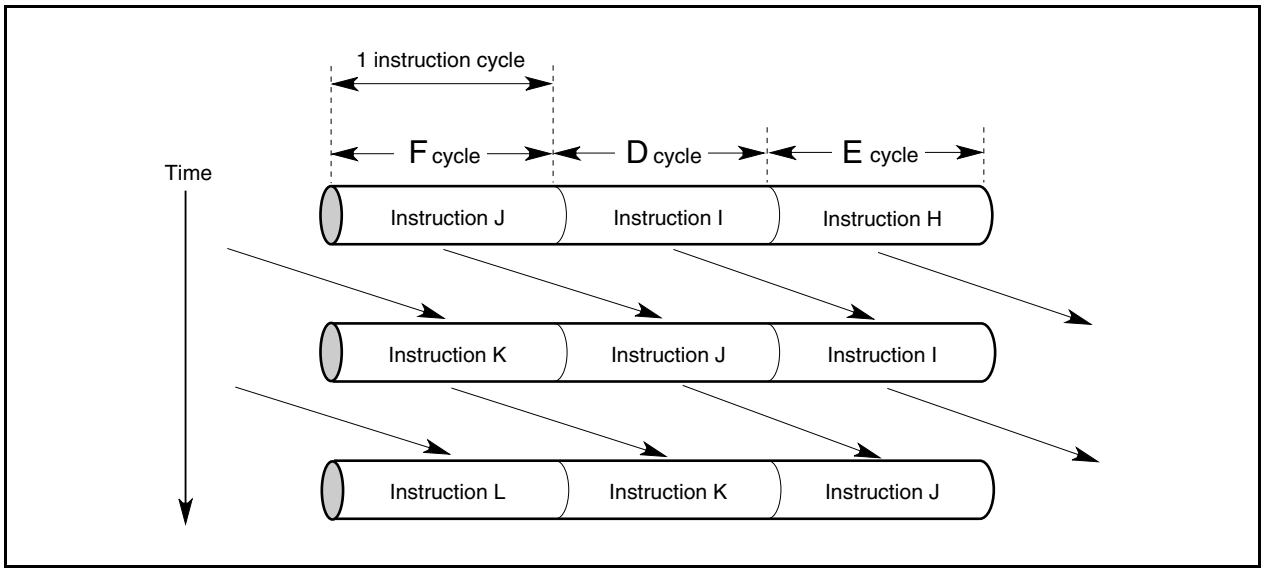
Figure 4-4 provides images of pipeline operation.

Figure 4-4 (a) is a conceptual illustration that shows the flow of executed instructions when viewed from each pipeline stage.

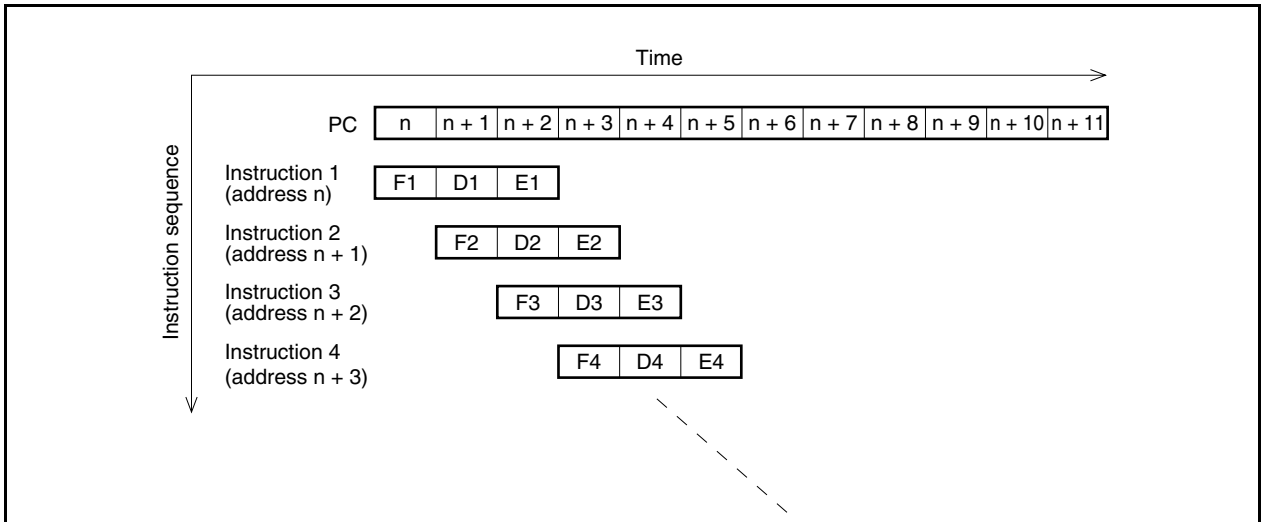
Figure 4-4 (b) shows the sequence in which instructions are executed in pipeline, from the viewpoint of each instruction.

Figure 4-4. Pipeline Image

(a) Pipeline image 1



(b) Pipeline image 2



Remark Fn: Fetch cycle of instruction n
Dn: Decode cycle of instruction n
En: Execution cycle of instruction n

(1) Successive MAC, ALU, Barrelshifter operations

When an instruction performing arithmetic/logic operations uses the result of the operation executed by the preceding instruction as an input operand, the result of the operation is written to a general-purpose register and, at the same time, input to the operation unit for the operation by the subsequent instruction. Consequently, programming can be done without having to be aware of the pipeline.

(2) Branch instruction

If a pipeline hazard occurs as a result of executing a branch instruction, the pipeline is replenished again with a NOP instruction inserted into the delay slot. Though the execution time is consequently extended, this does not cause erroneous application operation, and there is no need for users to consider the pipeline operation even in programming branch instructions.

For further details about pipeline timing with branch instructions refer to **4.4.2 Program execution control block**.

Caution The delay due to the processing of the pipeline must be taken into account in the following cases:

- **Instructions that control interrupts (by setting EIR, etc.) requires two instruction cycles to update the interrupt control information (refer to 4.4.4 Interrupts).**

- **When a value is set to DPn by using a Inter-register transfer instruction or immediate value setting instruction, the memory cannot be accessed by using the value set to DPn as an address until the instruction that follow the instruction that has set a value to DPn.**

  Example:

```
inst#1   DP0 = 0x0100    ;
inst#2   NOP             ; DP0 cannot be used here!
inst#3   R0L = *DP0      ;
```

- **The branch instruction cannot be written within three instructions before the loop end (refer to 4.4.3 Flow control block).**

4.3.4 Standby functions

μPD77210 Family devices include standby functions that stop operation of the devices to reduce current consumption. Instructions are used to set a standby status, and a device whose status is “standby” is said to be in standby mode. Specifically, there are two instructions that are used to set standby mode: the HALT instruction and the STOP instruction.

(1) Standby mode set by HALT instruction

When the HALT instruction is executed, the clock source is automatically switched to the divided clock set in the CLKC register. Therefore, before executing the HALT instruction, be sure to set a division rate and start operation of the divider.

The external interrupt signal that releases HALT mode requires four cycles of the divided clock. Be sure to enable the interrupt factor that is used for recovery.

When using a reset for recovery, the reset forcibly switches the clock source to an external clock, which can cause clock hazards. At that point, the contents of internal memory cannot be guaranteed, so be sure to perform a boot operation.

The status of pins during HALT mode is kept the same as before HALT mode except that the divided output is set for the CLKOUT pin and the HALT pin is set at high level.

Table 4-10. Status of Pins During HALT Mode

Pin Name	When $\overline{\text{HOLDRQ}}$ Is Active (Low Level)	When $\overline{\text{HOLDRQ}}$ Is Inactive (High Level)
CLKOUT[Note 1]	System clock output[Note 2]	
MA19 to MA0	High impedance	Previous status is maintained
MD15 to MD0	High impedance	
$\overline{\text{MRD}}$/$\overline{\text{MWR}}$	High impedance	High level
$\overline{\text{MHOLDAK}}$	High impedance	High level
$\overline{\text{MBSTB}}$	Low level	
TSORQ, TSIAK, TSO, ASO	High level	
$\overline{\text{HRE}}$, $\overline{\text{HWE}}$, HD15 to HD0	Previous status is maintained	
P15 to P0	Previous status is maintained	
TDO, TICE	Previous status is maintained	

Notes 1. Low level is maintained if the CLKC register is set to disable CLKOUT output.

2. During HALT mode, the divided clock has a high-level width of one cycle and is not a clock with 50% duty.

Caution Input pins and other pins that have high impedance during HALT mode should be fixed to either high level or low level.

(2) Standby mode set by STOP instruction

STOP mode is set by executing a STOP instruction. When this occurs, the device's current consumption is reduced by several hundred μA (when PLL has been stopped). The steps for setting STOP mode and recovering from STOP mode are as follows.

(a) Execute STOP instruction to set STOP mode.
(b) The internal clock output is masked and the device's current consumption is reduced by several hundred μA (when PLL has been stopped).
(c) Recover from STOP mode via a hardware reset or the CSTOP pin.

When a STOP instruction is executed, the clock source is automatically switched to the divided clock that was set to the CLKC register. Therefore, before executing the STOP instruction, be sure to set a division rate and start operation of the divider.

The PLL is not stopped during STOP mode. To further reduce current consumption, set the clock source directly to the CLKIN pin and stop the PLL before executing the STOP instruction.

Since the PLL's lock-up is not required when the PLL has not been stopped, it is possible to quickly recover from STOP mode. However, the clock supplied to the CLKIN pin should not be stopped, in order to maintain the lock status.

If a STOP instruction has been executed in instruction ROM, the current in ROM cannot be stopped, so the device's current consumption cannot be lowered. Therefore, it is recommended that the STOP instruction be executed in instruction RAM instead of instruction ROM.

To release STOP mode, set high level input to the CSTOP pin. Release requires 12 cycles of the divided clock at high level width.

When using a reset for recovery, the reset forcibly switches the clock source to an external clock, which can cause clock hazards. At that point, the contents of internal memory cannot be guaranteed, so be sure to perform a boot operation.

The status of pins during STOP mode is kept the same as before STOP mode except that the divided output is set for the CLKOUT pin and the HALTS and STOPS pins are set at high level. However, since no clock is supplied to the DSP core kernel and the peripherals, the DSP core kernel is unable to detect access request flags and interrupts that are output from the peripherals. When recovering, first set high level input to the CSTOP pin and low level input to the STOPS pin, then set low level input to the CSTOP pin to initiate the recovery processing.

4.4 Program Control Unit

This unit controls program execution. Data can be loaded from or stored to the registers in this unit via the main bus. This unit plays a role in execution of the following instructions:

- General instruction execution
- Branch instruction
- Hardware loop instruction
- Interrupt (Although an interrupt is not an instruction, PC, STK, SP, SR, and EIR are automatically controlled by INTC.)

Execution of these instructions is controlled by the following three blocks of the program control unit:

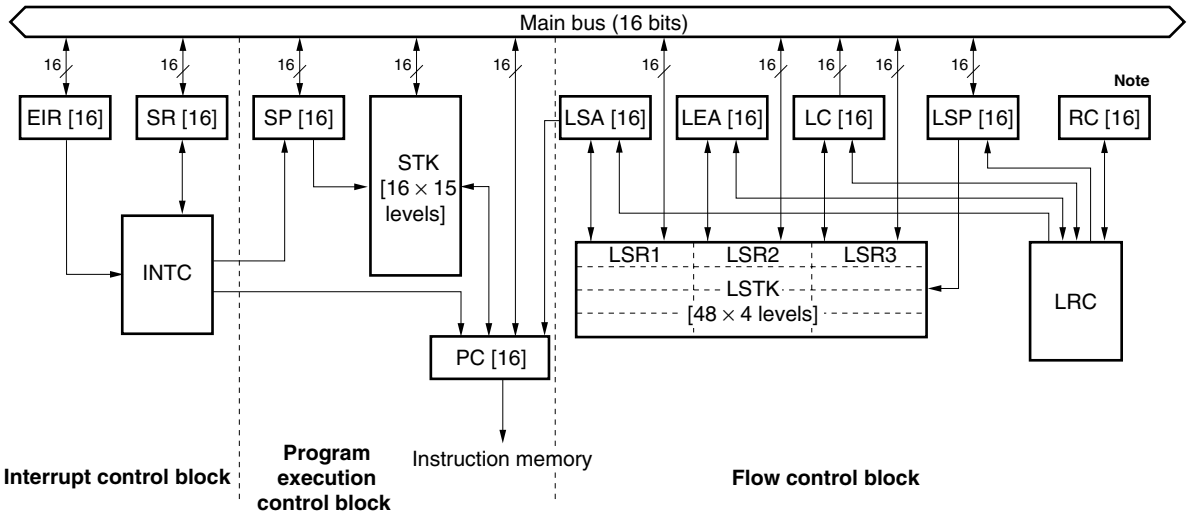
- Program execution control block
- Flow control block
- Interrupt control block

4.4.1 Block configuration shows a detailed block diagram of the program control unit. **4.4.2 Program execution control block** to **4.4.4 Interrupts** describe the details of the functions of the respective blocks.

4.4.1 Block configuration

Figure 4-5 shows the block configuration of the program control unit.

Figure 4-5. Program Control Unit



Note RC cannot directly transfer data via the main bus.

4.4.2 Program execution control block

Program execution is controlled by the following registers:

- Program counter (PC)
- Stack (STK)
- Stack pointer (SP)

(1) Program counter (PC)

This is a 16-bit register that holds the address of the instruction currently under execution when the program is executed. Therefore, the range of the value the PC can be set is the entire memory space.

Caution The PC can take any value as long as it is in the range of 16 bits, but the portion that is not defined as the instruction memory space or the portion that is reserved for the system must not be accessed.

(a) Internal instruction memory

μPD77210 Family devices include ROM or RAM that is used as internal instruction memory. The memory capacity differs from product to product in the μPD77210 Family. Table 4-11 lists instruction ROM and instruction RAM capacities for two μPD77210 Family products.

Table 4-11. Internal Instruction Memory Capacity

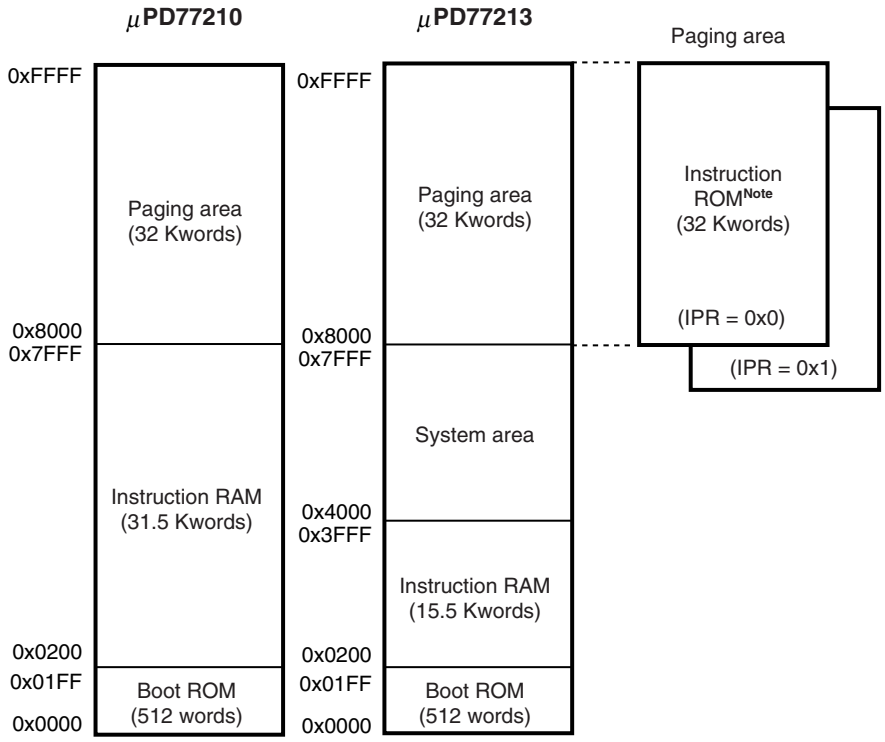
Item	Internal ROM Capacity	Internal RAM Capacity
μPD77210	None	31.5 Kwords
μPD77213	64 Kwords	15.5 Kwords

(b) Instruction memory

An instruction memory map of μPD77210 Family devices is shown below.

There is a method whereby instruction areas can be accessed as data areas. For details, see **4.5.3 Instruction memory alias**.

Figure 4-6. Instruction Memory Map



Note The MSB 8 words (0xFFF8 to 0xFFFF) of instruction ROM is the system area.

Caution Neither programs nor data can be allocated to the system area. The system area cannot be accessed, either. If system area addresses are accessed, the device's normal operation cannot be guaranteed. Also, it is not possible to access the paging area for an IPR page that does not exist. If access of such pages is attempted, the device's normal operation cannot be guaranteed.

(2) Stack (STK) and stack pointer (SP)

Stack (STK) is a register file dedicated to saving/restoring program counter (PC) and consists of 16 bits by 15 levels.

It is used to:

- Save return address when a subroutine is called
- Save the current address under execution when an interrupt occurs

For the details of the interrupt, refer to section **4.4.4 Interrupts**.

A pointer register that points to the stack level (called stack top) that is currently to be accessed is called stack pointer (SP). SP consists of 16 bits, but setting a value other than 0 to 15 to this pointer is prohibited. The stack top and SP are connected to the main bus; therefore, data can be exchanged with a general-purpose register via the main bus.

When the stack overflows or underflows, stack error flag (ste) of ESR is set to 1.

Remark Do not write the RET or RETI instruction just after the inter-register transfer instruction to load from/store to STK or SP.

(3) Related instructions

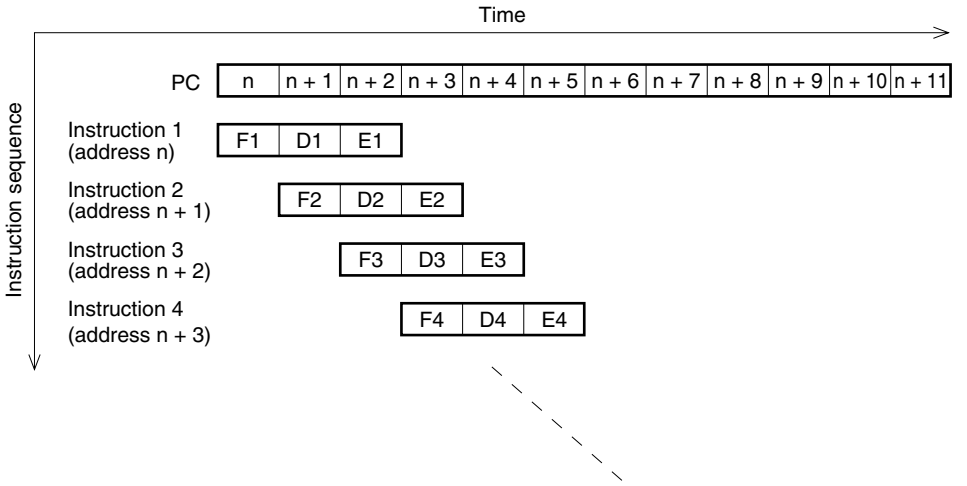
The operations of the program counter (PC), stack (STK), and stack pointer (SP) can be viewed from the following two points:

- Instruction execution and PC operation
- Branch instruction and operations of PC, SP, and STK

(a) Instruction execution and PC (normal operation)

The value of PC is incremented each time an instruction is fetched. Figure 4-7 shows the image when this PC operation is combined with pipeline execution.

Figure 4-7. Normal Operation of PC



Remark Fn: Fetch cycle of instruction n
Dn: Decode cycle of instruction n
En: Execution cycle of instruction n

(b) Branch instruction and operations of PC, SP, and STK

The branch instructions are classified into the following three types:

<1> Jump and subroutine call

The branch instructions are further subdivided into these two types of instructions, depending on whether the address of the instruction under opcode fetch (value of the PC) is saved to the stack or not.

- JMP instruction
  Does not save the address of the instruction under opcode fetch to the stack. Therefore, program flow cannot automatically return to the branch source address from the branch destination address.
- Subroutine call instruction
  Saves the address of the instruction under opcode fetch (address of the instruction next to the subroutine call instruction) to the stack. To return program flow from the branch destination address to the branch source address, the return instruction is used.

<2> Branch viewed from PC setting format

The branch instructions can be classified into the following two types when viewed from the format in which the branch destination address is set to the PC:

- Immediate jump/call

  This format is called immediate jump or immediate call. The JMP/CALL instructions for which a numeric value is coded as an operand execute branch in this format. At this time, the numeric value is added to or subtracted from the current PC value as 16-bit 2's complement. Therefore this is in fact a relative branch, relative to the current PC value. Program flow can be branched in the range of ±32 Kwords, i.e., in the entire 64-Kword space.

Caution When this instruction is written in assembler, write a direct branch destination address or label as the operand. The assembler calculates the correct relative branch distance to the current PC value automatically.

- Register indirect jump/call

  This format is called register indirect jump or register indirect call. The JMP/CALL instructions for which DPn register is described as an operand execute branch in this format. At this time, the value of the DPn register is directly set to the PC.

<3> Conditional or unconditional branch

The μPD77210 Family does not have dedicated conditional branch or conditional return instructions. Conditional branch is realized by combining conditional instructions and branch instructions, and conditional return is realized by combining conditional instructions and return instructions. These are classified into the following two types:

- Unconditional JMP/CALL/RET instructions

  These instructions always (unconditionally) branch (JMP/CALL/RET).
- Conditional JMP/CALL/RET instructions

  These instructions branch (JMP/CALL/RET) only when the condition of the combined conditional instructions is true.

Table 4-12 summarizes the above discussion. Note that, although the processing execution sequence when branch takes place does not differ depending on whether the instruction is conditional or unconditional, the actual execution time is 1 instruction cycle longer when a conditional instruction is used in combination. Although this is not indicated in the table, if the condition of a conditional branch instruction is not satisfied, delay due to pipeline hazard does not occur (refer to **Figures 4-8** to **4-11**).

Table 4-12. Classification of Branch Instructions

Instruction Name	Condition Judgment	Address Specification	Word Length	Instruction Cycles
Jump instruction	Unconditional	PC relative	1	2
	Conditional			3
	Unconditional	Register indirect absolute	1	3
	Conditional			
Subroutine call instruction	Unconditional	PC relative	1	2
	Conditional			3
Indirect subroutine call instruction	Unconditional	Register indirect absolute	1	3
	Conditional			
Return instruction	Unconditional	–	1	2
	Conditional			3
Interrupt return instruction	Unconditional	–	1	2
	Conditional			3

Caution Above number of instruction cycles are valid if the condition is satisfied and program flow branches. If the condition is not satisfied, even the conditional branch occupies one instruction cycle because the branch is not performed and no pipeline hazard occurs.

Figures 4-8 to 4-11 show the timing of the following instructions:

- Unconditional immediate jump
- Unconditional indirect jump
- Conditional immediate jump (condition satisfied: branch)
- Conditional immediate jump (condition not satisfied: pass)

The meanings of the symbols in each figure are as follows (n = 0, 1, 2, ..):

ifn:	instruction fetch	jifn:	jump destination instruction fetch
idn:	instruction decode	exn:	instruction execution
ia:	instruction address operation	addr:	address output
p:	purge	push:	stack push
pop:	stack pop	jdec:	jump destination decode
popi:	interrupt pop		

Figure 4-8. Timing of Unconditional Immediate Jump

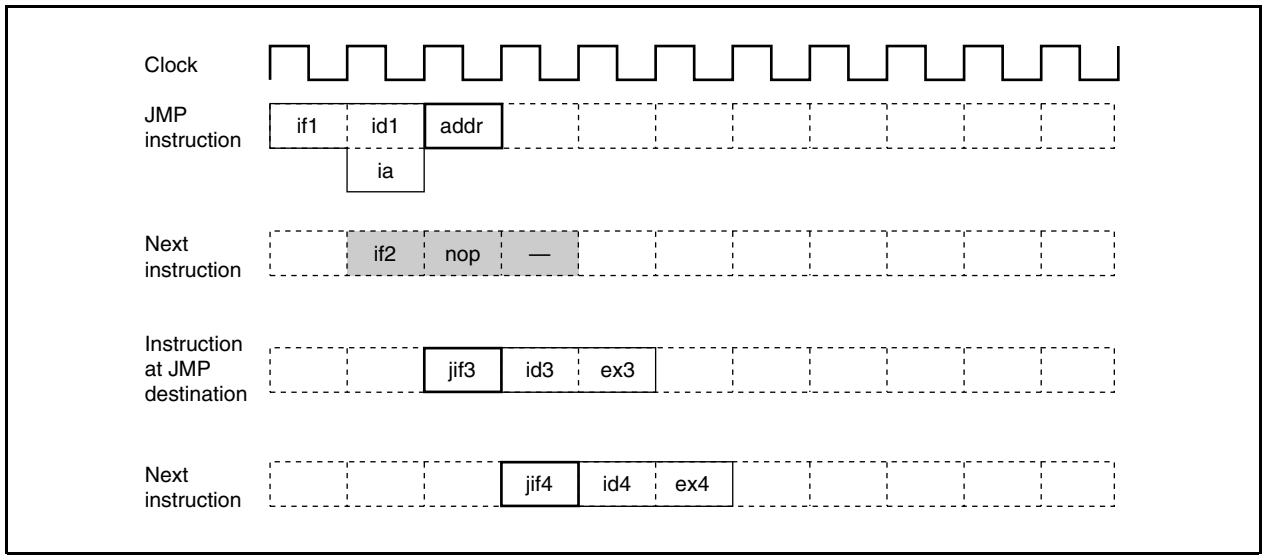


Figure 4-9. Timing of Unconditional Indirect Jump

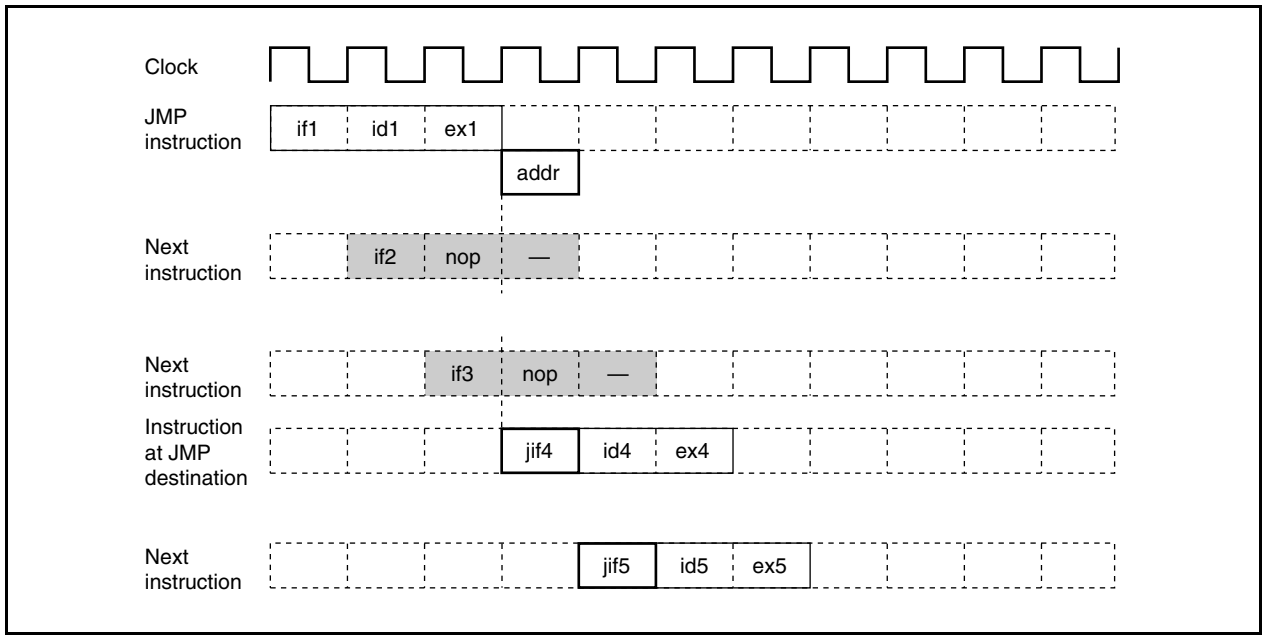


Figure 4-10. Timing of Conditional Immediate Jump (Condition Satisfied: Branch)

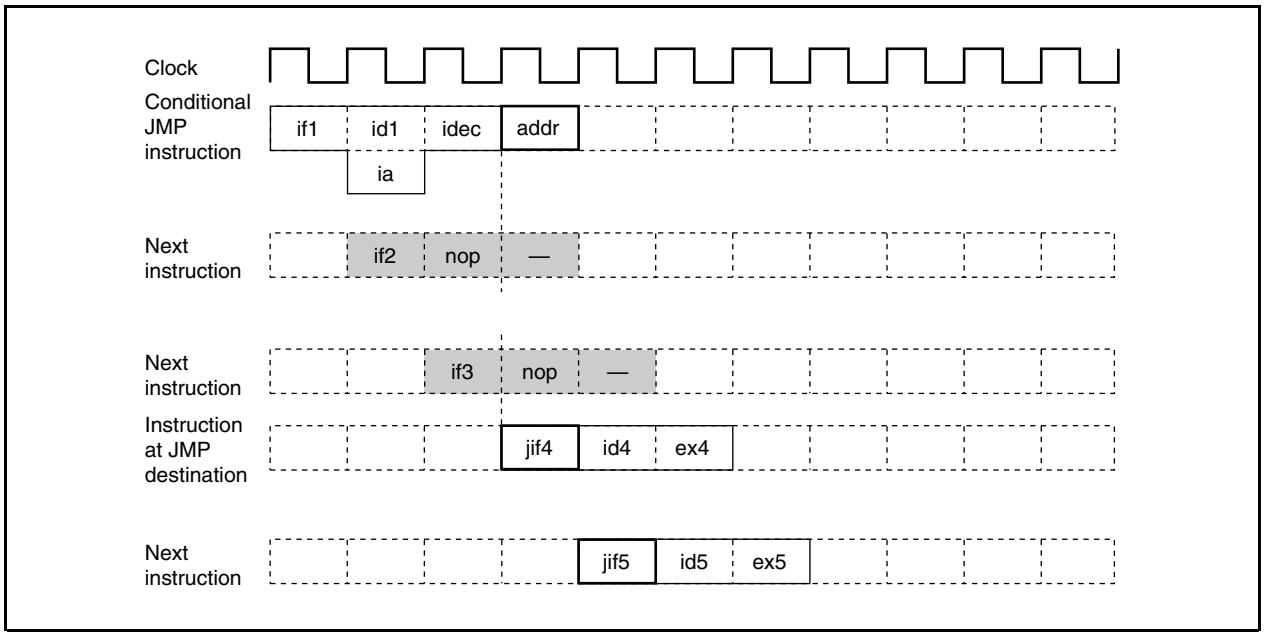
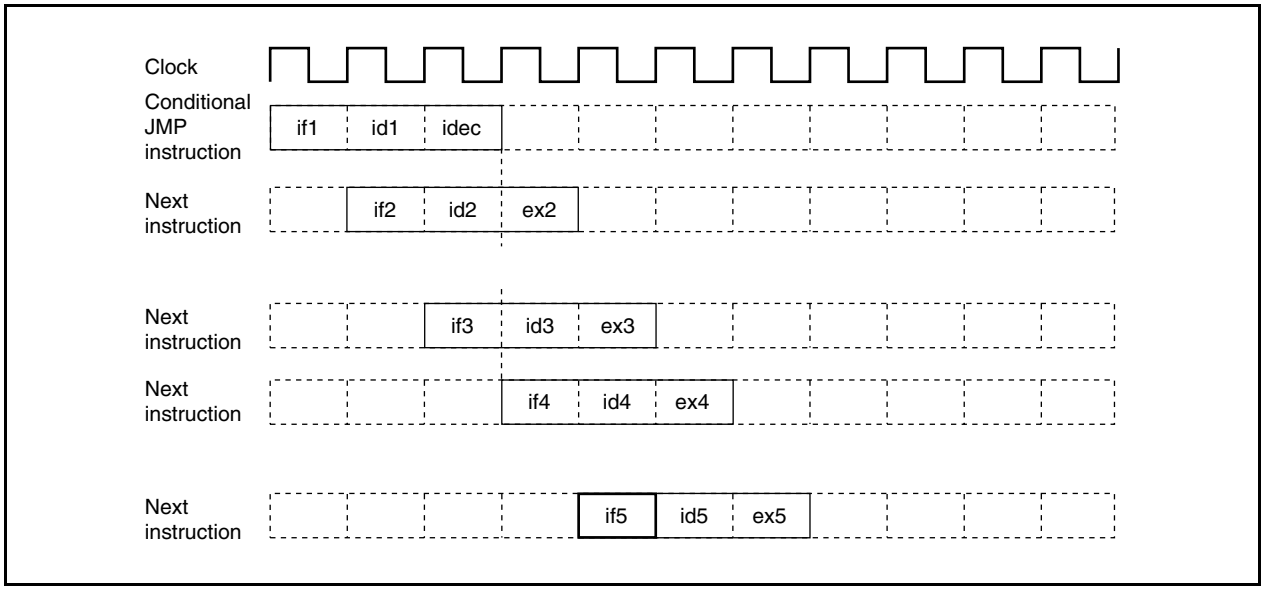


Figure 4-11. Timing of Conditional Immediate Jump (Condition Not Satisfied: Pass)



(c) Operation of subroutine call/return

Subroutine call is executed by the CALL instruction. When the CALL instruction is executed, execution branches in the following procedure:

<1> The value of SP is incremented (pre-increment).
<2> The value of PC (address next to the CALL instruction) is saved to the STK indicated by SP.
<3> The branch destination address is set to the PC. At this time, if the branch destination is given as a numeric value, the numeric value is added to or subtracted from the current PC value as 2's complement. If the branch destination is given by the DPn register, the value of the DPn register is directly set to the PC.

To return execution from a subroutine, the RET instruction is used. This instruction is executed in the following procedure:

<1> The value in the STK indicated by the SP is restored to the PC.
<2> The value of SP is decremented (post-decrement).

Remark For the timing of the CALL instruction, refer to the timing of the JMP instruction.
The timing of the CALL instruction is the same as that of the JMP instruction, except that the return address is saved to the stack. The timing of the return instruction is the same as that of the immediate jump instruction, i.e. it takes two instruction cycles.

(d) Operation when interrupt occurs

When an interrupt occurs, the address of the instruction under opcode fetch (address of the instruction when the interrupt is acknowledged) is saved to the stack, and the branch destination address is set to the PC. To return from the interrupt, the RETI (return from Interrupt) instruction is used.
For the operation of the interrupt, refer to **4.4.4 Interrupts**.

4.4.3 Flow control block

In general, a high-level language provides sophisticated flow control syntax (e.g., for loop and while loop of the C language). The μPD77210 Family is provided with hardware that allows this flow control to be directly described as assembly instructions, and performs loop/repeat operation without any timing overhead. The loop/repeat control circuit controls the loop/repeat operations.

Flow control is managed by the following registers and functional blocks:

- Repeat counter (RC)
  This 16-bit counter register holds the number of repetitions of a repeat instruction.

- Loop start address register (LSA)
  This 16-bit register holds the loop start address during loop execution.

- Loop end address register (LEA)
  This 16-bit register holds the loop end address during loop execution.

- Loop counter (LC)
  An initial value is set to this 16-bit register when execution of the LOOP instruction is started. Each time loop is executed once, the value of this register is decremented. When the current value of the register reaches 0, it indicates the end of the loop.

- Loop stack (LSTK)
  The LSTK is a register file with 3 × 16 bits × 4 levels to save and store the LSA, LEA and LC values. It saves the LSA, LEA and LC values by the loop instruction. The values are restored to the LSA, LEA and LC upon loop termination or by the loop pop instruction.
  This file serves as one of the following three 16-bit registers for input/output to/from the main bus with inter-register transfer instruction.
  - LSR1: Saves/restores loop start address (stack for LSA)
  - LSR2: Saves/restores loop end address (stack for LEA)
  - LSR3: Saves/restores loop counter (stack for LC)

  If LSR1 is specified for the inter-register transfer instruction source, the LSP is decremented after transfer. If LSR1 is specified for the inter-register transfer instruction destination, data is transferred after the LSP is incremented.

- Loop stack pointer (LSP)
  This pointer indicates the current position of LSTK. Although this is a 16-bit register, the value that can be set to it is 0 to 4.
  The LSP value can be input/output to/from the main bus with inter-register transfer instruction. The LSP value becomes 0 by reset.
  The LSP is incremented/decremented by 3 bits (bits 2 to 0). Bits 15 to 3 are fixed to 0.
  The LSP is incremented in the following cases:
  - When the LSA, LEA, and LC values are saved to the LSTK by the loop instruction
  - When the LSR1 is specified for the inter-register transfer instruction destination

  The LSP is decremented in the following cases:
  - When the LSTK value is returned to the LSA, LEA, and LC upon loop termination or by the loop pop instruction
  - When LSR1 is specified for the inter-register transfer instruction source

Cautions 1. When the value of LSP is not between 0 to 4, a stack overflow or underflow occurred indicating an error.

2. Do not set the LSP value between 5 and 0xFFFF.

- Loop/repeat controller (LRC)
  This circuit controls the loop and repeat instructions.

Caution Flow control registers, except repeat counter (RC), are connected to the main bus, so that data can be transferred between them and general-purpose registers.

Flow control has the following two functions:

- Repeat function (REP instruction)
- Loop function (LOOP instruction, LPOP instruction)

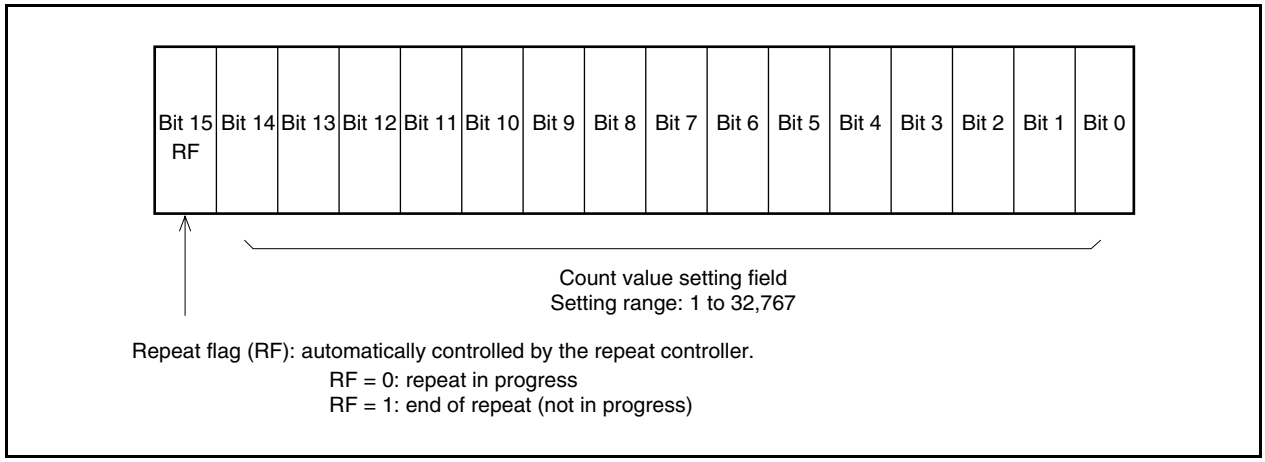
(1) Repeat function

The repeat function that is written by the REP instruction realizes repetition of one instruction on a count basis. The instruction to be repeated the repeat target instruction, follows immediately the REP instruction itself.

(a) Format of repeat counter (RC)

Figure 4-12 shows the format of the repeat counter (RC).

Figure 4-12. Format of RC



Caution During the entire repeat operation no interrupt will be acknowledged. For further details refer to 4.4.4 Interrupts.

(b) Summary of repeat function

The repeat function can be summarized as follows:

- A single instruction is repeated.
- The number of repetitions can be directly given as a numeric value or by using a general-purpose register (R0L to R7L).
- The number of repetitions ranges from 1 to 32,767.
- The value of PC is not incremented during repeat operation.
- RC is decremented each time the instruction is repeated, and repeat ends when the instruction has been repeated the specified number of times.
- The repeat function depends on RC only, and is not counted as nesting of loop instructions.

(c) Procedure of repeat function execution

When the REP instruction is executed, the repeat function is implemented in the following procedure.

<1> The number of repetitions given as the parameter of the REP instruction is set to RC.
<2> The value of PC is incremented, and the instruction immediately after the REP instruction is repeated. At this time, an invalid cycle of one instruction cycle is generated.
<3> During repeat operation, PC holds a next address of this instruction that has been repeated.
<4> The value of RC is decremented each time the instruction has been repeated once. After the instruction has been repeated the specified number of times, repeat ends.
<5> When repeat ends, the value of PC is incremented. When execution shifts from the instruction that has been repeated to the next instruction, the pipeline stages are successive. Therefore, no overhead occurs when repeat ends.

For the repeat instruction, refer to ***μ*PD77016 Family Instructions User's Manual**.

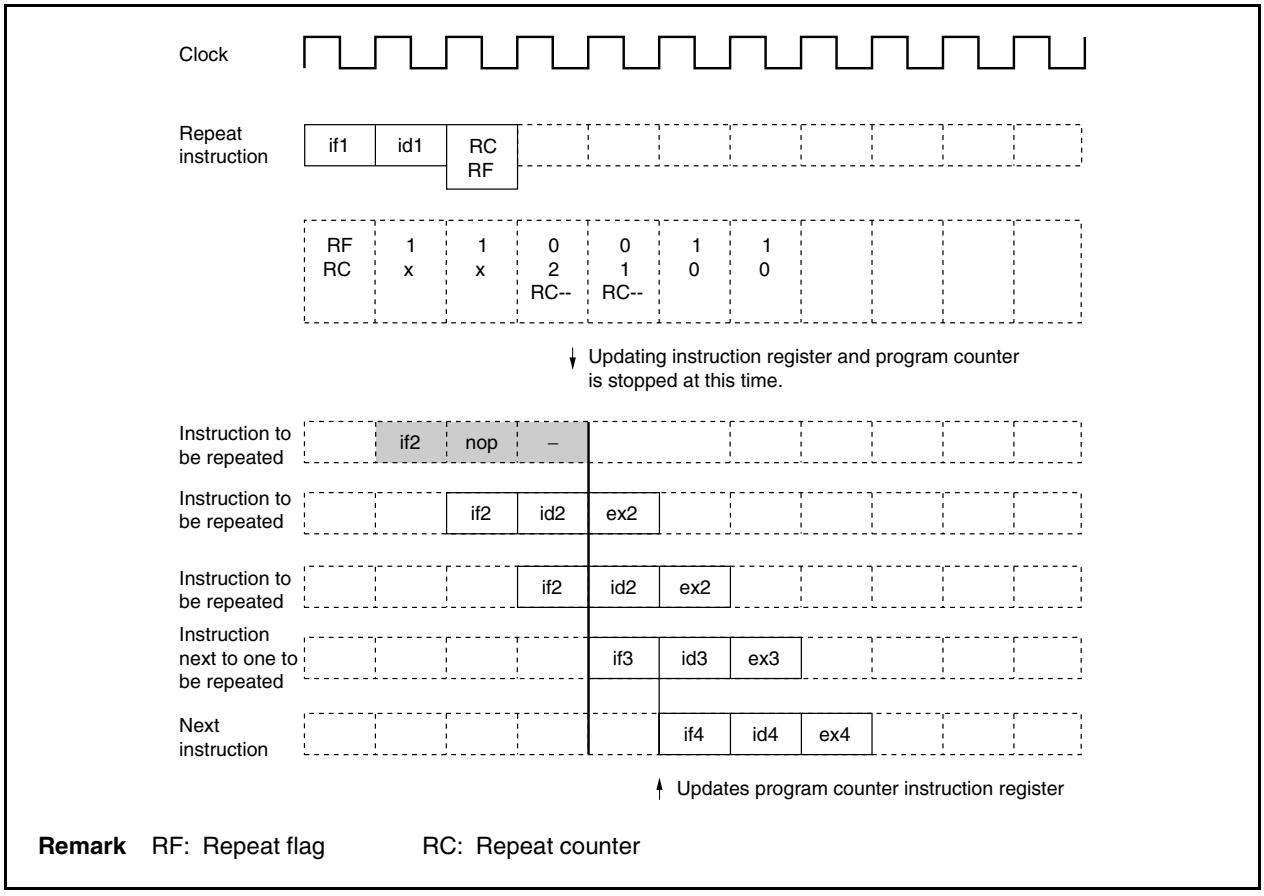
(d) Repeat execution timing

The following figures show an example in which the REP instruction is repeated two times.
Figure 4-13 shows the assembly program, and Figure 4-14 shows the execution timing.

Figure 4-13. Example of Repeat Instruction (Repetition of 2 Times)

```
REP     2;
R0    /= R1;
```

Figure 4-14. Repeat Execution Timing (Repetition of 2 Times)



Remark RF: Repeat flag RC: Repeat counter

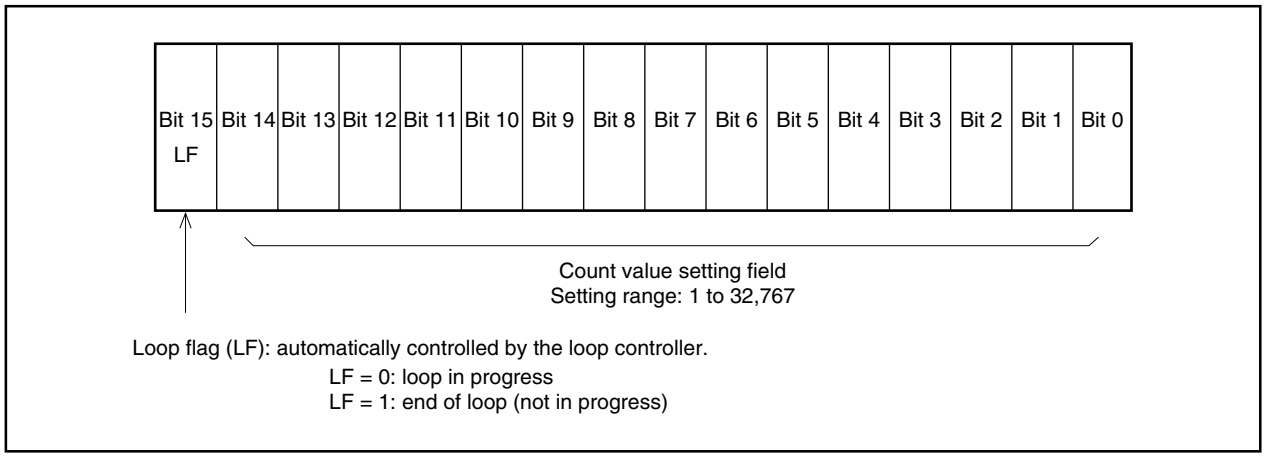
(2) Loop function

The loop function that is described by using the LOOP instruction realizes loop flow of an instruction group consisting of 2 to 255 instructions on a count basis. Nesting of loop is supported by a four level hardware loop stack. To escape from the loop at any point, the LPOP instruction is provided, so that flexible loop control is performed.

(a) Format of loop counter (LC)

Figure 4-15 shows the format of the loop counter (LC).

Figure 4-15. Format of LC



Remark The loop flag LC is also contained in the status register (SR) (refer to **4.4.4 Interrupts**).

(b) Summary of loop function

The loop function can be summarized as follows:

- Groups 2 to 255 instructions as a loop element.
- The number of loops can be given directly by a numeric value or by using a general-purpose register (R0L to R7L).
- The number of loops ranges from 1 to 32,767.
- Nesting of up to 4 levels can be realized by the loop stack.
- Execution can be escaped from the loop when:
  (1) The count value reaches 1
  (2) The LPOP instruction and then JMP instruction are executed

Remark For interrupt processing in conjunction with loop operations (refer to **4.4.4 Interrupts**).

(c) Loop function execution procedure

When the LOOP instruction is executed, the loop function is implemented in the following procedure:

<1> When loop is started

1. The value of LSP is incremented (pre-increment).
2. The current LSA, LEA, and LC are saved to LSTK indicated by LSP.
3. The loop start address is set to LSA.
4. The loop end address is calculated and set to LEA.
5. The number of loops is set to LC.

<2> During loop operation

1. The value of LC is decremented if the values of PC and LEA are equal.
2. The value of LSA is set to PC if LC is not 1. If LC is 1, the loop end processing is executed.

<3> Loop end processing

1. The value of PC is incremented.
2. The value of LSTK indicated by LSP is restored to LSA, LEA, and LC.
3. The value of LSP is decremented (post-decrement).

<4> Loop end processing by LPOP instruction

The LPOP instruction discards one level of loop by performing the following processing.

1. Restores the value of LSTK indicated by LSP to LSA, LEA, and LC.
2. Decrements the value of LSP (post-decrement).

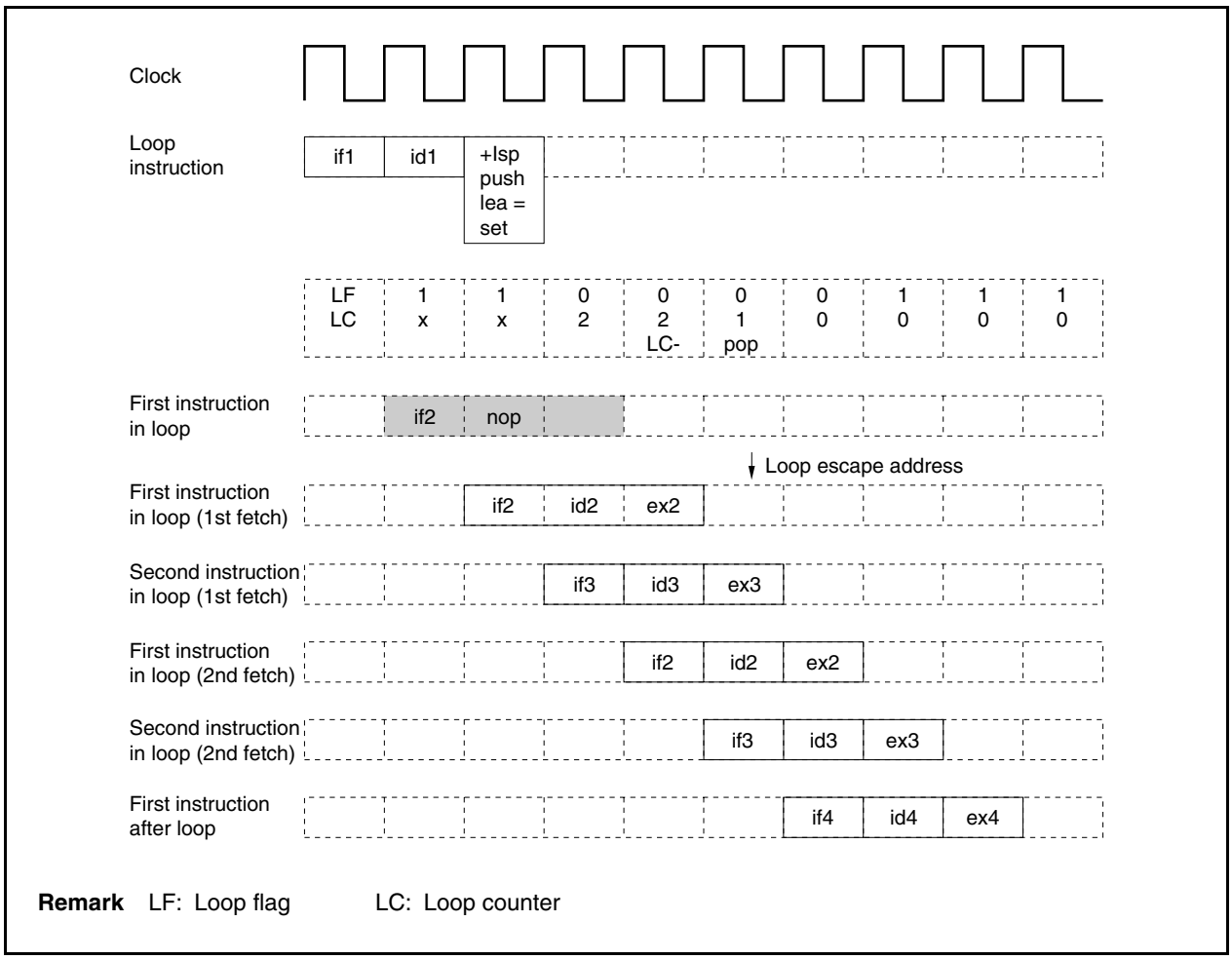
For the LOOP and LPOP instructions, refer to **μPD77016 Family Instructions User's Manual**.

Caution The LPOP instruction does not automatically control PC for escaping from the loop. Therefore, execute the LPOP instruction after escaping from the loop by using the JMP instruction, or execute the LPOP instruction and then escape from the loop by using the JMP instruction (refer to μPD77016 Family Instructions User's Manual).

(d) Timing of loop execution (example of two loops operation)

Figure 4-16 shows an example of the LOOP instruction execution timing. In this example, two loops operation in which a group of two instructions is executed only once is performed.

Figure 4-16. Loop Execution Timing (Example of 2 Loops Operation)



Remark LF: Loop flag LC: Loop counter

(e) Software loop stack

Performing a loop of five or more levels will cause the occurrence of a loop stack overflow. The consequent loss of the return address means that a normal loop operation can no longer be performed.

When it is apparent that loop processing with five or more levels is about to occur, saving the contents of the loop stack (LSTK) to the memory before it overflows will enable normal operation, even if a further loop is performed. This procedure is known as a software loop stack.

Note, however, that when the LSTK contents are saved to the memory, these contents must be written back to LSTK at the corresponding stack level. An example of a software loop stack program is shown below.

- **Example of software loop stack**

- Push (DP0: Saved address)

```
R0L = LSR3;
*DP0-- = R0L;
R0L = LSR2;
*DP0-- = R0L;
R0L = LSR1;
*DP0-- = R0L;
```

- Pop (DP0: Restored address)

```
R0L = *DP0++;
LSR1 = R0L;
R0L = *DP0++;
LSR2 = R0L;
R0L = *DP0++;
LSR3 = R0L
```

4.4.4 Interrupts

μPD77210 Family devices feature powerful interrupt functions. They fall into two types: interrupt functions for the DSP core kernel and interrupt controller functions for peripherals. The following describes the interrupt functions for the DSP core kernel. For description of the interrupt controller functions for peripherals, see **5.10 Interrupt Controller (INTC)**.

- Interrupt factors (sources)
- Interrupt control functions
- Interrupt acknowledgement conditions
- Interrupt vectors

(1) Interrupt factors (sources)

There are a total of 12 interrupt factors for the DSP core kernel. All of these interrupts are input to the DSP core kernel via the interrupt controller (a peripheral). The interrupt factors for the DSP core kernel are triggered by interrupt requests from the interrupt controller.

(2) Interrupt control function

All interrupt causes, regardless of whether they are handled as independent events and at independent levels. Here is the summary of the functions to control the interrupts:

- Each interrupt source can be enabled or disabled independently.
- All interrupts can be enabled or disabled.
- A stack for global interrupt enable function is provided, so that multiple interrupts can be handled.
- The interrupt vectors (entry points of each interrupt source routine at interrupt acknowledgement) are fixed.
- When an interrupt has been acknowledged, the current instruction is aborted, and program execution control is transferred to the specified entry point.
- After restoring from the interrupt program, control is returned to the instruction that was suspended by the interrupt.
- During the execution of the Jump instruction, interrupt acknowledgment is delayed.

(3) Interrupt acknowledgment condition

When an interrupt request is generated by an interrupt cause, the interrupt will be acknowledged if both following conditions are satisfied:

- Global interrupt enable (EI) flag value is 0 (enable).
- Interrupt cause enable flag value corresponding to the requested interrupt is 0 (enable).

Note, however, that acknowledging the interrupt is delayed in any of the following cases:

- While a jump instruction is fetched, decoded, or executed
- While a repeat instruction or a repeat target instruction is fetched, decoded, or executed
- While a loop instruction is fetched, decoded, or executed
- While a loop termination instruction (instruction at loop end address) is fetched

(4) Interrupt vectors

All interrupt factors have a fixed entry point (called a vector). The vector for each interrupt factor is set sequentially from the start address position (address 0x200) in the internal instruction area, which configures a 64-word table. Each factor is assigned four instruction addresses. If interrupt servicing is not completed within four instructions including the interrupt return instruction (RETI), interrupt servicing must branch beyond address 0x240 to complete servicing.

(a) Interrupt vector table

Table 4-13 shows an interrupt vector table.

In μPD77210 Family devices, an interrupt peripheral has been added to support expanded interrupt factors. For details, see **5.10 Interrupt Controller (INTC)**.

Table 4-13. Interrupt Vector Table

Vector Address	Interrupt No.
0x200	Reset
0x204	Reserved
0x208	
0x20C	
0x210	0
0x214	1
0x218	2
0x21C	3
0x220	4
0x224	5
0x228	6
0x22C	7
0x230	8
0x234	9
0x238	10
0x23C	11

Cautions 1. Although a reset is not an interrupt, it is still handled in the same way as a vector entry.

2. It is recommended that interrupt factors that are not used should be branched to error processing routines.

3. Do not code a RETI instruction at the start address of an interrupt vector, or else normal operation will not occur in the μPD77210 Family device if that interrupt actually occurs.

4. In products that include mask ROM, the vector area also exists in the internal RAM area. Consequently, this area must be booted. Since the entry point after a reset is 0x200, address 0x200 must be booted even when the internal instruction RAM and interrupts are not used.

(b) Example of interrupt vector processing

An example of interrupt vector processing is shown below.

```
 ; Definitions
#define     SI1   0x3800          ; Address of serial input register
#define     SO1   0x3800          ; Address of serial output register
#define     ICR4   0x3884         ; Interrupt control register for interrupt No. 4
#define     ICR5   0x3885         ; Interrupt control register for interrupt No. 5

;Interrupt vector table
int_vec     imseg at 0x200        ; Vector table
   :
   :
            org 0x220
                                  ; Serial input #1 (TSI) interrupt vector
(0x220)     JMP INPUT             ; Branches to application area for at least four instructions
(0x221)     NOP                   ;
(0x222)     NOP                   ;
(0x223)     NOP                   ;
                                  ; Serial output #1 (TSO) interrupt vector
                                  ; Example of interrupt servicing within four instructions
 (0x224)    R0H=*DP4++            ; Fetches data from Y memory
 (0x225)    *SO1:y=R0H            ; Transfers to serial output #1
 (0x226)    RETI                  ; Recovery from interrupt
 (0x227)    NOP                   ;
                                  ; Serial input #2 (ASI) interrupt vector
 (0x228)    NOP                   ; SI2 is not used (do not code RETI instruction to start of vector)
 (0x229)    RETI                  ;
 (0x22A)    NOP                   ;
 (0x22B)    NOP                   ;
   :
   :
 ; Main program segment
main        imseg                 ;
             :
             :
            R0L=SR                ;
            R0=R0 & 0x7FCF        ; Enables overall interrupts and interrupt No. 4 and No. 5
            SR=R0L                ;
            R0L=0x0010            ; Setting for interrupt controller (a peripheral)
            ICR4:x=R0L            ; Enables mask mode and TSI interrupts
            ICR5:x=R0L            ; Enables mask mode and TSO interrupts
             :
                                  ; Serial input #1 interrupt servicing routine
INPUT:      R0H=*SI1:y            ; Fetches data from serial input  #1
            R1=*DP0               ;
            R1=R1+R0H*R2H         ;
            *DP0=R1H              ;
            RETI                  ; Recovery from interrupt
```

(5) Interrupt control software

Interrupts are controlled by the following registers (refer to **Figure 4-5 Program Control Unit**):

- Status register (SR)
- Interrupt enable flag stack register (EIR)

(a) Status register (SR)

This is a 16-bit register that enables or disables all the interrupts (general interrupt enable/disable), and enables or disables each interrupt cause separately. When the value of a bit of this register is 0, the corresponding interrupt is enabled; when the bit is 1, the interrupt is disabled.

The values of SR can be read and written by executing the register-to-register transfer instruction. This register is set to 0xF000 at reset.

Interrupt enable flag			**Note**	Factor-specific interrupt enable flags (interrupt Nos.)											
EI	EP	EB	LF	11	10	9	8	7	6	5	4	3	2	1	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note Loop flag

<1> Interrupt enable flags (EI: enable interrupt, EP: enable interrupt previous, EB: enable interrupt before)

The EI, EP, and EB flags enable or disable all the interrupts. When the value of these flags is 0, the interrupts are enabled; when it is 1, the interrupts are disabled. These three flags, EI (enable interrupt), EP (enable interrupt previous), and EB (enable interrupt before), enable or disable the current interrupts, and interrupts one levels before and two levels before.

The EI, EP, and EB flags are the same as the EI, EP, and EB flags of bits 15 to 13 of the interrupt enable flag stack register (EIR). Bits 15 to 13 of the SR and EIR registers always have the same values.

The following nesting of interrupt and stack manipulation are handled by the EI, EP, and EB flags and E3 through E15 flags of the interrupt enable flag stack register (EIR).

When interrupt has been acknowledged;

value of EB → E3 of EIR register
value of EP → shifted to EB
value of EI → shifted to EP
EI → set to 1 (all interrupt disable)

Vice versa at RETI instruction;

value of EI → wasted
value of EP → shifted to EI
value of EB → shifted to EP
value of E3 of EIR register → EB

For multiple interrupts, refer to **(b) Interrupt enable flag stack register**.

The interrupt flag before updating is valid while a transfer instruction which specifies SR as the destination is fetched and executed, that is, between the transfer instruction and the next instruction, and between the next instruction and the instruction that follows.

The following shows on example of changing interrupt enable flag (enabled →disabled).

```
Initial status: EI = 0        ; (interrupt enabled)
R0L = EIR                     ;
R0 = R0 | 0x8000              ;
EIR = R0L                     ;
Next instruction              ;   May branch to interrupt servicing
Instruction that follows      ;
```

Caution To rewrite the EP and EB flag, be sure to disable all the interrupts (EI = 1).

<2> Loop flag (LF)

This flag indicates whether execution is in a loop or not. The value "0" shows that the execution is in a loop, and "1" for not in a loop.

Caution Do not change this flag when modifying any interrupt mask flags. Modify interrupt mask flags always by reading the current SR contents and mask only the dedicated flags (refer to following examples).

<3> Reserved flags

A write to these flags is ignored. Undefined values are returned when these flags are read.

<4> Factor-specific interrupt enable flags

These flags indicate the enabled/disabled status of interrupts for specific factors. When the flag's bit value is 0, the corresponding interrupt is enabled and when this bit value is 1 the corresponding interrupt is disabled. This value does not change when the interrupt is acknowledged.

Caution Set disabled status (EI = 1) for all interrupts before overwriting any factor-specific interrupt enable flags.

An example of how a factor-specific interrupt enable flag can be changed (from enabled to disabled status) is shown below.

```
R0L = EIR          ; Use EIR to set disabled status for all interrupts.
R0 = R0 | 0x8000   ; EI = 1
EIR = R0L          ; Returned (written) to EIR
NOP                ; Wait until value set to EIR becomes valid.
R0L = SR           ; Use SR to disable interrupts for interrupt No. 0.
R0 = R0 | 0x0001   ; (Interrupt No. 0 = 1)
SR = R0L           ; Returned (written) to SR
```

(b) Interrupt enable flag stack register (EIR)

This 16-bit register stacks the general interrupt enable flags. When a bit of this register is 0, the corresponding interrupt is enabled; when the bit is 1, the interrupt is disabled.

The values of EIR can be read and written by executing the register-to-register transfer instruction.

The value of this register is set to 0xFFFF at reset.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EI	EP	EB	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15

When an interrupt has been acknowledged, the contents of this register are shifted 1 bit to the right, and the bit EI is set to 1 to disable all interrupts generally. The register contents are shifted 1 bit to the left by execution of the interrupt return instruction RETI where E15 is set to 1 simultaneously. Cause of them, multiple interrupts of up to 16 levels are guaranteed.

Bits 15 to 13 (EI, EP, EB) are the same as the bits 15 to 13 of the SR register.

The interrupt enable/disable status can be changed by writing EIR with the register-to-register transfer instruction. However, note that this change will be valid three instructions after writing EIR.

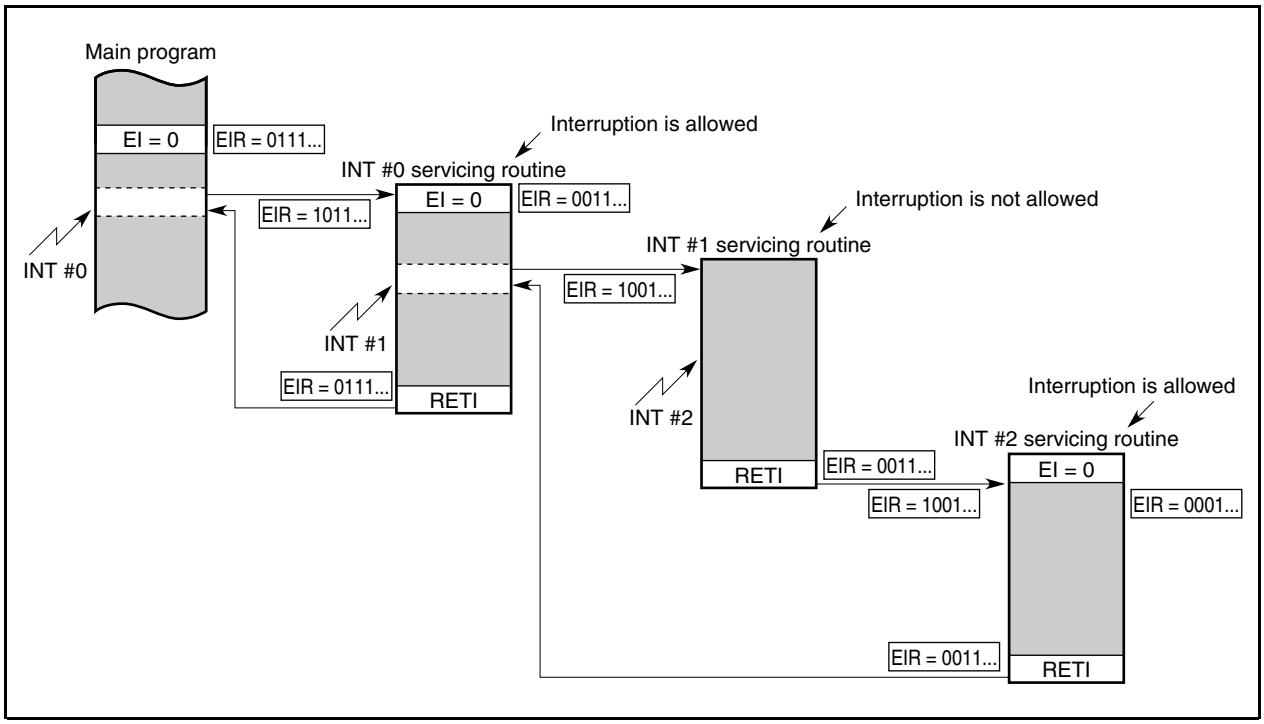
The following shows an example of enabling interrupt (disabled status → enabled)

```
Initial status: EI = 1 (interrupt disabled)
R0L = EIR          ;
R0 = R0&0x7FFF     ;
EIR = R0L          ;  }
Instruction 1      ;  }  Interrupt disabled during this period
Instruction 2      ;  }
Instruction 3      ; Interrupt enabled
```

(c) EIR and multiple interrupts

As described earlier, a multiple interrupt system can be configured by using the EIR register. This paragraph describes the concept of multiple interrupts, taking an example shown in Figure 4-17 and focusing on EIR.

Figure 4-17. Multiple Interrupt Processing



[Process]

<1> Clear the EI bit to 0 to enable all interrupts.

<2> INT #0 is acknowledged, and control is transferred to the INT #0 servicing routine. At this time, the contents of the EIR register are shifted 1 bit to the right, and one level of interrupt status is stacked. At the same time, bit 15 (EI) is set to 1, disabling the other interrupts.

<3> The interrupts are enabled (EI = 0) in the INT #0 servicing routine.

<4> INT #1 is acknowledged, and control is transferred to the INT #1 servicing routine. In the same manner as before, the contents of EIR are shifted 1 bit to the right, and EI is set to 1, disabling the interrupts.

<5> INT #2 request is generated while the INT #1 servicing routine is executed. However, this interrupt is not acknowledged because it is disabled, but recorded.

<6> When the INT #1 servicing routine is ended in the RETI instruction, the contents of EIR are shifted 1 bit to the left. Consequently, the status before acknowledging the INT #1 interrupt is restored. In this status, EI = 0, enabling the interrupts.

<7> The recorded INT #2 is now acknowledged, and control is transferred to the INT #2 servicing routine. The contents of EIR are shifted 1 bit to the right again, and EI is set to 1. If necessary, clear EI to 0.

<8> When the INT #2 servicing routine is ended in the RETI instruction, the contents of EIR are shifted 1 bit to the left, and the status before INT #2 was acknowledged is restored (INT #0 is being processed).

<9> Execution of the INT #0 servicing routine continues. When the RETI instruction is executed at the end of this routine, the status before INT #0 was acknowledged is restored.

(d) Differences between SR and EIR

The most significant three bits of the SR and EIR registers (EI, EP, and EB) are accessed as common bits. The EI bit directly enables or disables the current interrupt, and therefore care must be exercised in manipulating this bit. The differences between SR and EIR are as follows, when the EI bit is manipulated:

- To enable the interrupts, the EI bit of either the SR or EIR register can be used.
- To disable the interrupts, use of the EI bit of the EIR register is recommended.

There is no problem when the interrupts are enabled by the EI bit because the interrupts have been disabled up to that point. When the interrupts are disabled, however, the following situation may arise:

```
R0L = SR                    ; disable interrupt generally here:  via SR register

        ← Interrupt occurs and jump to interrupt servicing routine:

        ; Interrupt servicing routine
        ; This routine disables INT #0 interrupt individually
        R1L = SR            ;
        R1 = R1|0x0001      ; set INT #0 = 1 (disabled)
        SR = R1L            ; write back to SR
        RETI                ; return from interrupt

        ← SR has changed meanwhile

R0 = R0|0x8000              ; set EI = 1
SR = R0L                    ; write back to SR
      :
      :
```

In this case, writing data to the SR register is ignored while the interrupt is serviced. To avoid this situation, it is recommended to use the EI bit of the EIR register, rather than that of the SR register, to disable the interrupts.

(6) Interrupt sequence

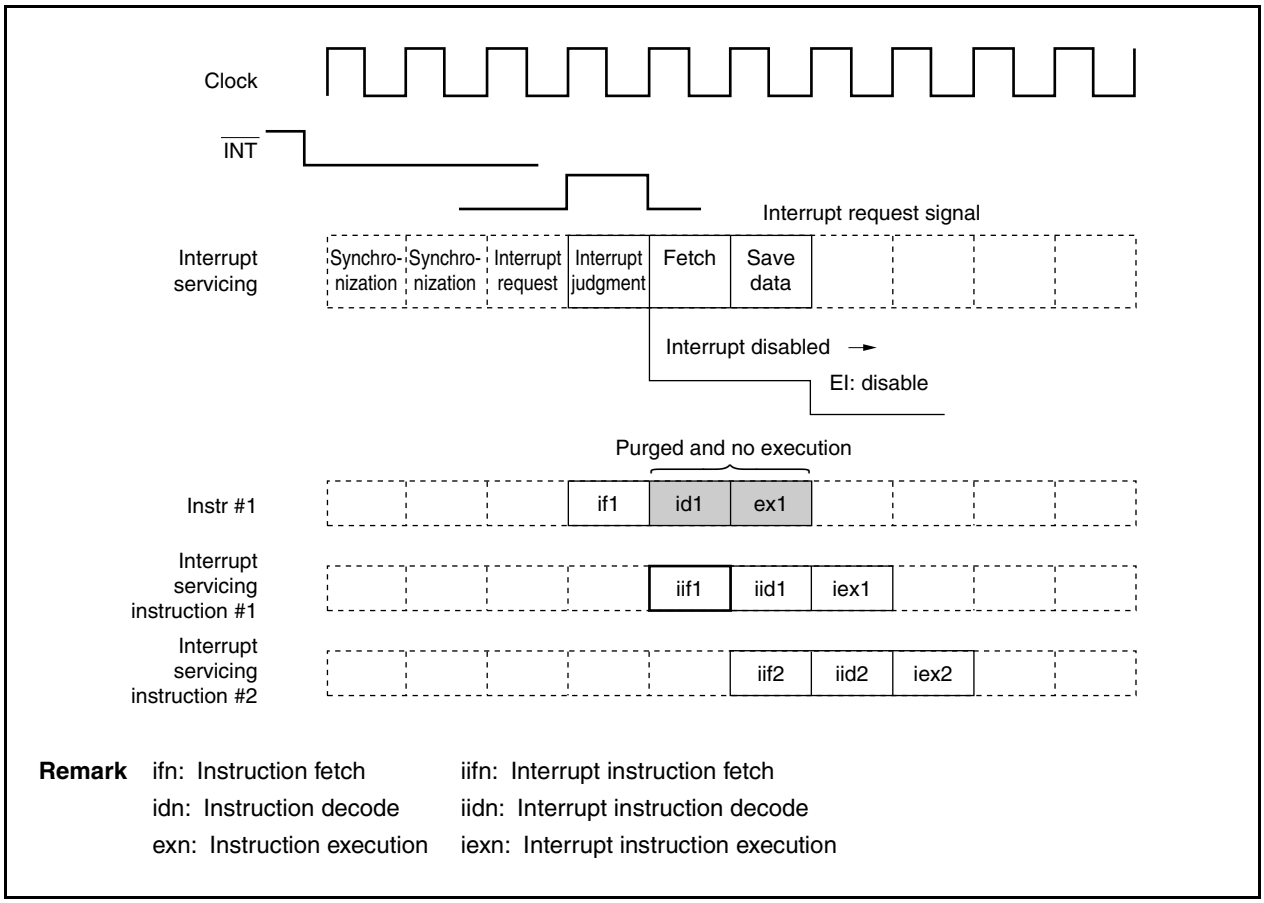
(a) Acknowledging an interrupt

When an interrupt has been acknowledged, the following operations are performed:

- An instruction that was fetched immediately before the interrupt has been acknowledged is kept pending.
- The EIR register is shifted 1 bit to the right to stack 1 level.
- The EI bit is set to 1 to disable the interrupts.
- SP is incremented.
- The address of the pending instruction is saved to STK specified by SP.
- A specified interrupt vector address is set to PC, and execution branches to interrupt servicing routine.

Figure 4-18 shows timing of acknowledging an interrupt.

Figure 4-18. Interrupt Acknowledging Timing



Remark ifn: Instruction fetch iifn: Interrupt instruction fetch
idn: Instruction decode iidn: Interrupt instruction decode
exn: Instruction execution iexn: Interrupt instruction execution

(b) Returning from interrupt

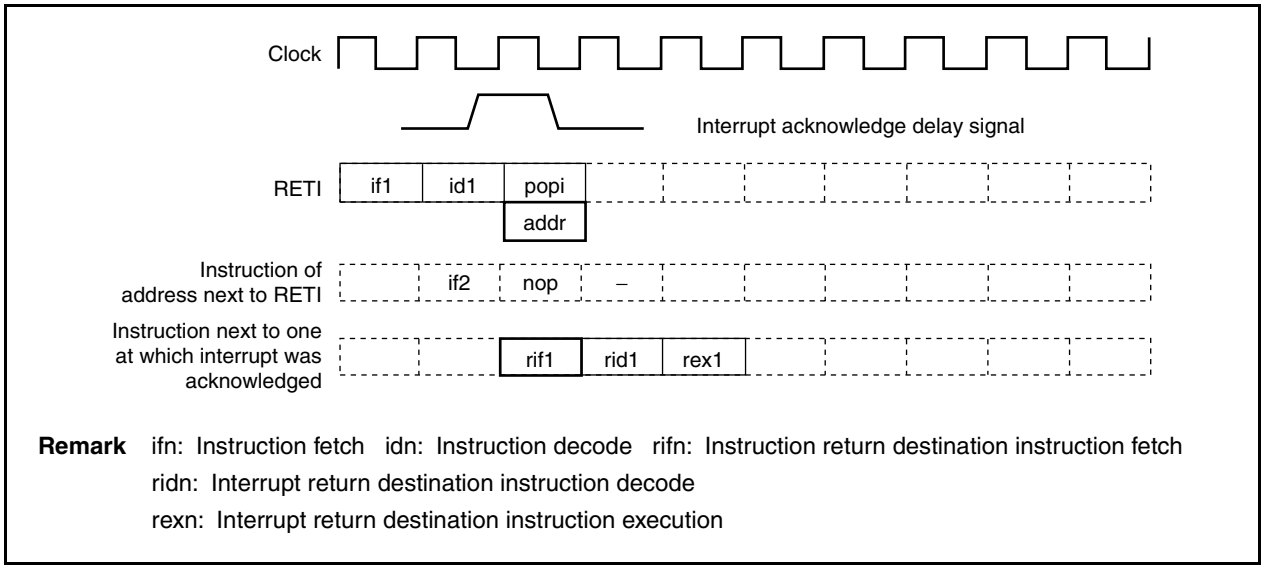
When the RETI instruction (interrupt return) is executed, the following are processed in two to three instruction cycles, and execution returns from the interrupt servicing routine.

- The value of STK indicated by SP is restored to PC.
- SP is decremented.
- The EIR register is shifted to the left, and interrupt enable flags are restored.
- Execution branches to the return address (the instruction that was kept pending when the interrupt was acknowledged).

Figure 4-19 (a) and (b) shows the return timings by using an unconditional RETI instruction and a conditional RETI instruction with the condition satisfied, respectively.

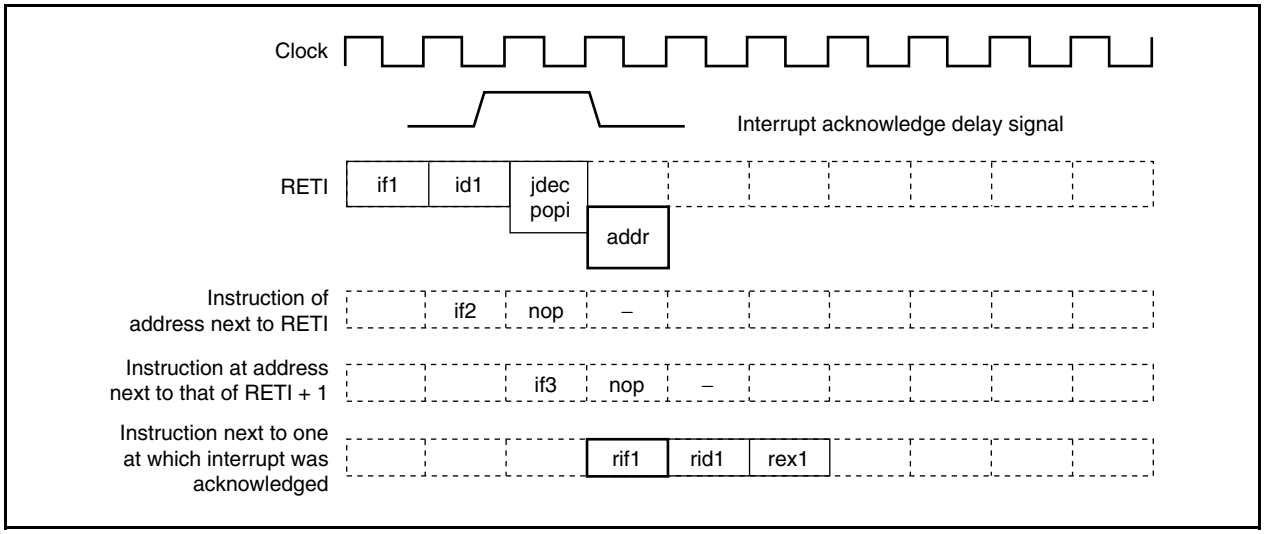
Figure 4-19. Timing by RETI Instruction

(a) Unconditional



Remark ifn: Instruction fetch idn: Instruction decode rifn: Instruction return destination instruction fetch
ridn: Interrupt return destination instruction decode
rexn: Interrupt return destination instruction execution

(b) Conditional instruction: Condition satisfied



(7) Delaying interrupt acknowledgment

In the course of acknowledging an interrupt, registers SP, STK, and PC are automatically managed. To prevent conflicts with instructions that address these registers, acknowledging an interrupt is delayed when any of the following instructions that may cause such a conflict is executed. Note that the interrupt acknowledgement itself (branching to the interrupt servicing routine) still introduces only a single delay cycle.

Caution Interrupt is not acknowledged under following conditions and interrupt request is held until interrupt enables;

- **During peripheral I/O wait function**
- **During external memory access wait cycles**
- **During repeat process**

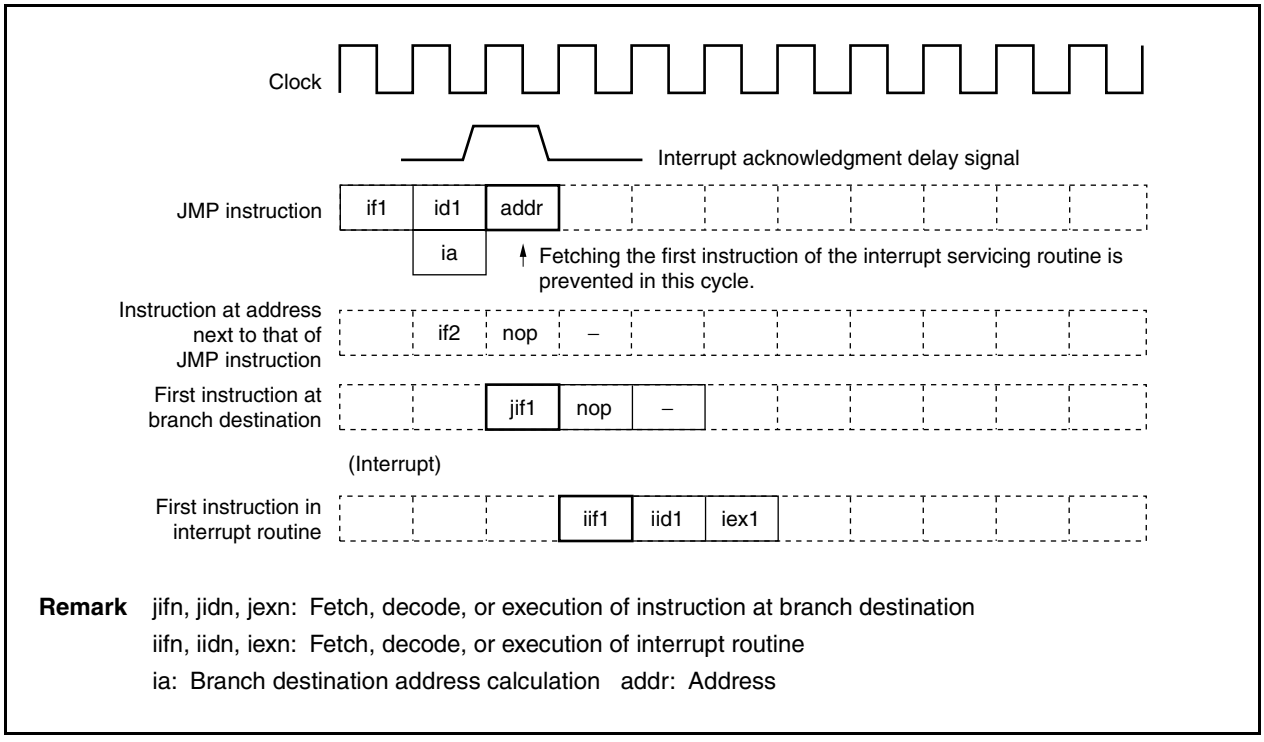
(a) Instructions generating delay of one instruction cycle

The following instructions cause a delay of interrupt acknowledgment of one instruction cycle:

- Decoding of unconditional JMP instruction (PC-relative jump by immediate data)
- Decoding of unconditional CALL instruction (PC-relative jump by immediate data)
- Decoding of unconditional RET instruction
- Decoding of unconditional RETI instruction
- Decoding of FINT instruction
- Fetching of loop end instruction

Figure 4-20 illustrates how an interrupt is delayed that occurs during the processing of any of these instructions.

Figure 4-20. Interrupt Delay Timing (One-Cycle Delay)



Remark jifn, jidn, jexn: Fetch, decode, or execution of instruction at branch destination
iifn, iidn, iexn: Fetch, decode, or execution of interrupt routine
ia: Branch destination address calculation addr: Address

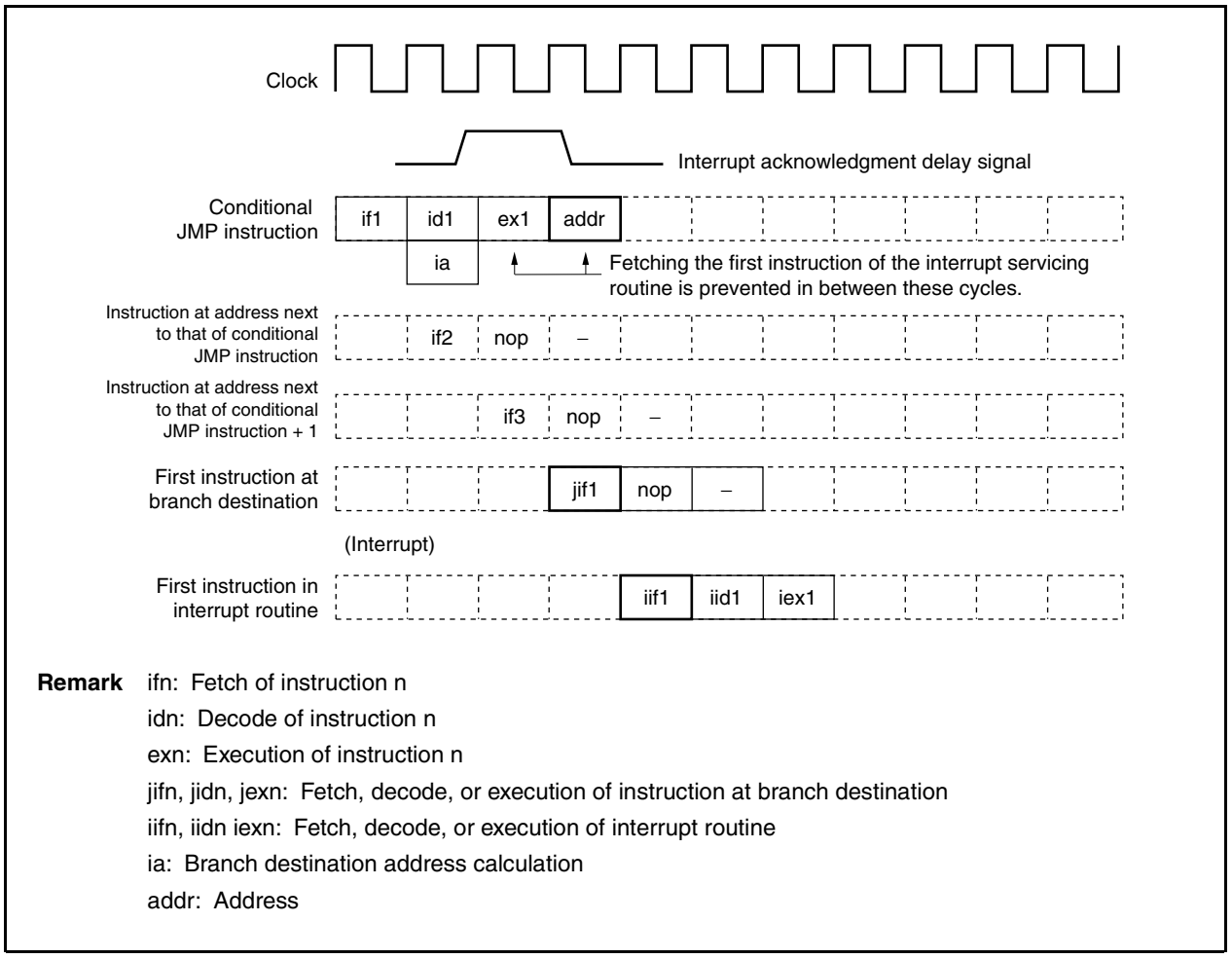
(b) Instructions generating delay of two instruction cycles

The following instructions cause a delay of interrupt acknowledgment of two instruction cycles:

- Decoding of conditional JMP instruction (PC-relative jump by immediate data)
- Decoding of conditional CALL instruction (PC-relative jump by immediate data)
- Decoding of conditional RET instruction
- Decoding of conditional RETI instruction
- Decoding of unconditional/conditional register-indirect JMP instruction
- Decoding of unconditional/conditional register-indirect CALL instruction
- Decoding of REP instruction
- Decoding of LOOP instruction

Figure 4-21 illustrates how an interrupt is delayed that occurs during the execution of any of these instructions.

Figure 4-21. Interrupt Delay Timing (Two-Cycle Delay)



Remark ifn: Fetch of instruction n
idn: Decode of instruction n
exn: Execution of instruction n
jifn, jidn, jexn: Fetch, decode, or execution of instruction at branch destination
iifn, iidn iexn: Fetch, decode, or execution of interrupt routine
ia: Branch destination address calculation
addr: Address

(8) Conflict and recording of interrupt

(a) Recording interrupt

When an interrupt has been acknowledged, an interrupt servicing program is executed. During the execution, the global interrupt enable flag "EI" is automatically set to 1 (disable). Therefore, if another interrupt occurs during this period, it is not acknowledged immediately, but is recorded classified by the cause. When the interrupt servicing program has been ended in the RETI (return from interrupt) instruction, the EI flag is cleared to 0, enabling other interrupts. Consequently, the recorded interrupt is acknowledged and processed. This interrupt recording function works not only when EI is 1, but also when the corresponding interrupt enable flags are set to the disable state.

Cautions 1. All interrupt request are recorded, disregarding the settings of all interrupt enable/disable flags.

2. Only one level of interrupt can be recorded per cause.

3. The internal flag that records the occurrence of an interrupt is not cleared unless the corresponding interrupt is acknowledged.

4. The FINT instruction discards all interrupt requests. For further details refer to μPD77016 Family Instructions User's Manual.

(b) Priority of interrupt

It is undefined which interrupt is served first if two or more interrupts occur at the same time.

4.4.5 Error status register (ESR)

This 16-bit register holds error flags which indicate some error status's of the μPD77210 Family. A write to bits 15 to 4 of this register is ignored. Undefined values are returned when these flags are read.

Bits 3 to 0 of ESR are set to 1 when an error occurs. The values of these bits are not clear to 0 unless a hardware reset is applied or they are rewritten by program (inter-register instruction).

The values of ESR can be read and written by executing the inter-register transfer instruction.

The value of this register is cleared to 0 at reset.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–	–	–	–	–	ovf	ste	lse	–

<1> ovf: Overflow error flag

This flag is set to 1 if an overflow occurs while the operation unit calculates data in the 40-bit two's complement format.

<2> ste: Stack error flag

This flag is set to 1 when the stack overflows or underflows.

<3> lse: Loop stack error flag

This flag is set to 1 when the loop stack overflows or underflows.

Caution Although bit 0 of the ESR is the bac (bus access error) flag in the μPD7701X Family, this flag does not exist in the μPD77210 Family. It is therefore impossible to detect the error that occurs when a prohibited memory area combination is accessed.

4.5 Data Addressing Unit

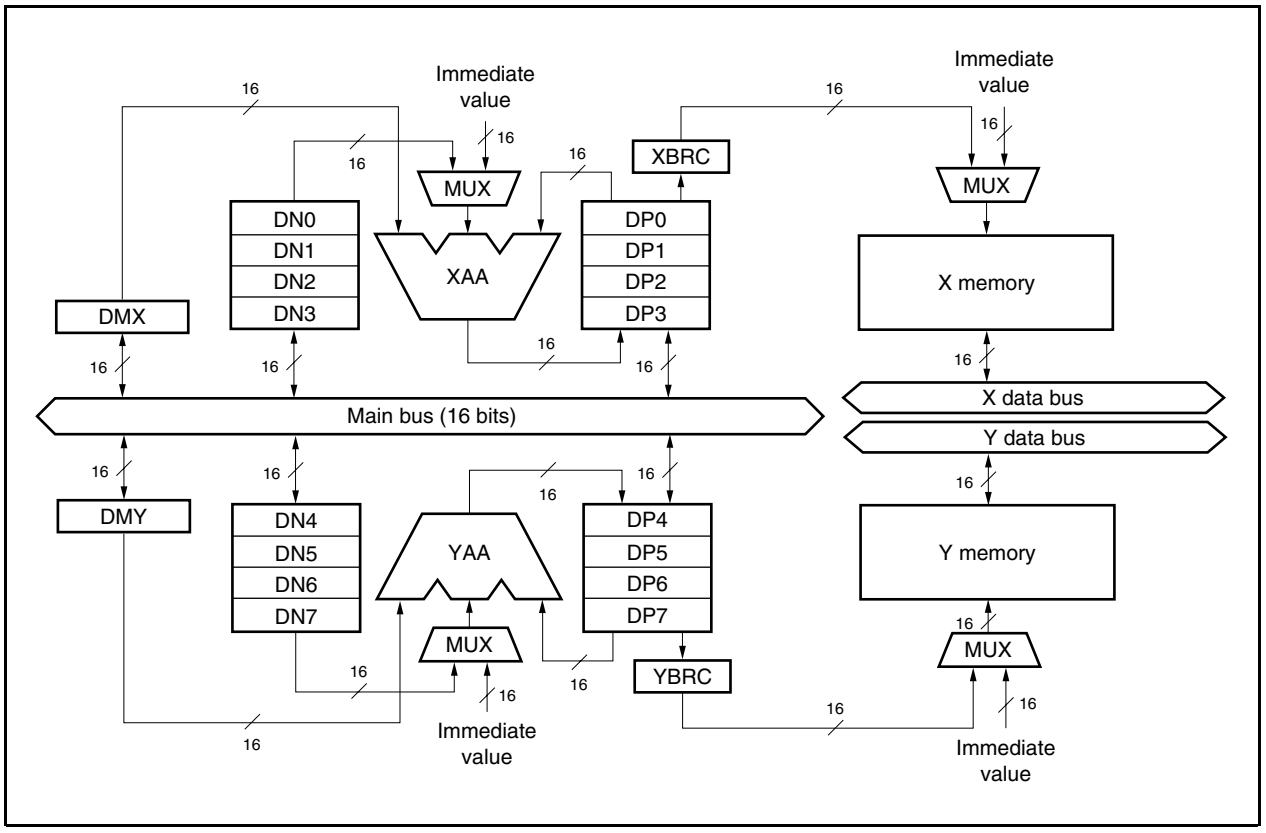
Generally, a DSP is required to access a large quantity of data flexibly and efficiently.

The μPD77210 Family is provided with dedicated data addressing units to efficiently access the data memory spaces.

4.5.1 Block configuration

Figure 4-22 is the block diagram of the data address unit.

Figure 4-22. Data Addressing Unit



4.5.2 Data memory space

The devices of the μPD77210 Family have two independent data memory spaces, X and Y, to which data can be accessed flexibly. Each of the X and Y data memory spaces is divided into internal memory and external memory areas. The internal memory area can always be accessed at high speeds as an internal resource of the device. The internal memory areas of both the X and Y memory spaces can be accessed simultaneously. The external memory area allows connection of memories of various speed range, using the incorporated software and hardware wait functions. In addition, the internal memory area is further divided into ROM and RAM areas.

This subsection describes the memory spaces.

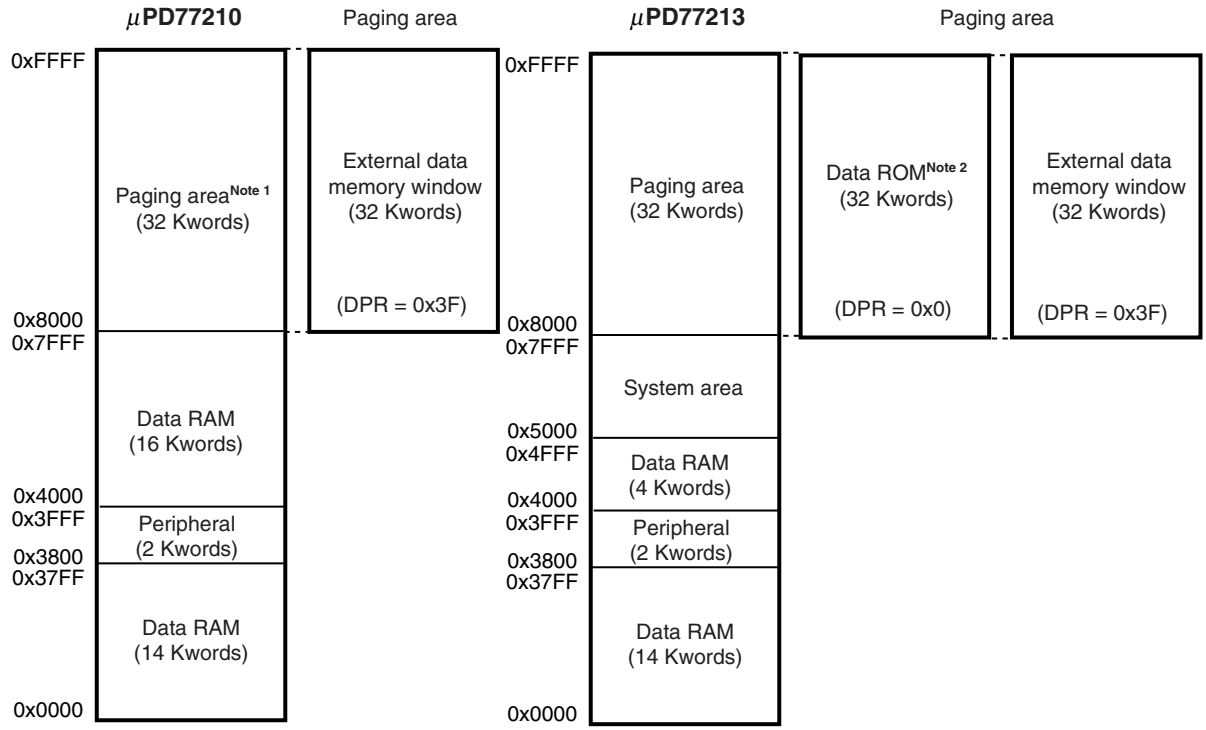
(1) X and Y memory spaces

The devices of the μPD77210 Family have two independent data memory spaces: X and Y. These spaces are respectively accessed via the X and Y data buses (refer to **4.2.2 Data bus**). The features of these memory spaces are as follows:

- One word consists of 16 bits.
- Both X and Y spaces have 64 Kwords.

Although the memory maps of the X and Y memory spaces are the same, there are some differences among the products in the μPD77210 Family. The following figure shows the X and Y memory maps of each product in the Family.

Figure 4-23. Data Memory Map



Notes 1. When the value of the paging register is anything other than 0x3F (external data memory window) or 0x80 (internal instruction RAM area), addresses in the paging area should not be accessed.

2. The MSB eight words (0xFFF8 to 0xFFFF) of data ROM comprise the system area.

Caution Neither programs nor data can be allocated and accessed to the "system area." If system area addresses are accessed, the device's normal operation cannot be guaranteed. Also, it is not possible to access the paging area for an DPR page that does not exist. If access of such pages is attempted, the device's normal operation cannot be guaranteed.

(2) Internal data memory

As is shown in Figure 4-23, the 32-Kword area that starts from address 0 (in the μPD77210) or the 18-Kword area that starts from address 0 (in the μPD77213) functions as an internal area that has been mapped within the device (the peripheral area is from address 0x3800 to address 0x3FFF). This internal area is divided into a ROM area, RAM area, peripheral area, and system area, and ROM and RAM among these function as the data memory. The internal data memory capacity differs from product to product, but it is possible to select the capacity that best suits the target application.

For details of the peripheral area, see **5.2 Peripheral Register**.

Caution Access to the system area is prohibited.

(a) Internal ROM and internal RAM

As mentioned above, the capacity of the ROM and RAM components of internal area differs from product to product. This variation in capacity is summarized below.

Table 4-14. ROM and RAM Capacity

Product	ROM		RAM	
	X	Y	X	Y
μPD77210	None	None	30 Kwords	30 Kwords
μPD77213	32 Kwords	32 Kwords	18 Kwords	18 Kwords

(3) External data memory interface

In μPD77210 Family devices, the external memory interface is used as a peripheral. For details, see **5.7 External Data Memory Interface (MIO)**.

(a) External data memory capacity

As is shown in Figure 4-23, external data memory can be expanded in μPD77210 Family devices. The amount of expandable memory capacity differs from product to product, as is shown in Table 4-15 below.

Table 4-15. External Data Memory Capacity

Product	Memory capacity
μPD77210	1 Mword
μPD77213	1 Mword (or 8 Kwords when using the SD card interface)

(4) Restrictions on simultaneous access

μPD77210 Family devices are able to access two objects in the X and Y memory spaces at the same time, such as through parallel loading, but such simultaneous access is subject to the following restrictions.

- Parallel access cannot be targeted at the peripheral area (0x3800 to 0x3FFF).
- Example memory access to the area from 0x8000 to 0xFFFF when DPR = 0x3F is routed via the MIO peripheral, and parallel access is not permitted when it involves a peripheral.
- Since DPR is shared by X and Y memories, simultaneous access to different pages is not possible.

Table 4-16 lists conditions for simultaneous access.

Table 4-16. Conditions for Simultaneous Access to X and Y Memories

Y Memory \ X Memory		0x0000 to 0x37FF	0x3800 to 0x3FFF	0x4000 to 0x7FFF	0x8000 to 0xFFFF			
					DPR = 0x0 to 0x3E	DPR = 0x3F	DPR = 0x80	DPR = 0xC0 to 0xFF
0x0000 to 0x37FF		OK	–	OK	OK	OK	OK	OK
0x3800 to 0x3FFF		–	–	–	–	–	–	–
0x4000 to 0x7FFF		OK	–	OK	OK	OK	OK	OK
0x8000 to 0xFFFF	DPR = 0x0 to 0x3E	OK	–	OK	OK	–	–	–
	DPR = 0x3F	OK	–	OK	–	–	–	–
	DPR = 0x80	OK	–	OK	–	–	OK	–
	DPR = 0xC0 to 0xFF	OK	–	OK	–	–	–	OK

In addition, the following restrictions exist when accessing instruction memory contents as data.

- When accessing instructions in the area from 0x0000 to 0x7FFF while DPR = 0x80, one wait is inserted to prevent access conflict.
- When accessing instructions in the area from 0x8000 to 0xFFFF while DPR = 0xC0 to 0xFF, one wait is inserted to prevent access conflict.

4.5.3 Instruction memory aliasing

Pages with aliases in instruction memory exist in the data page memory, and when those pages are specified in the data page register instruction memory contents can be accessed as data.

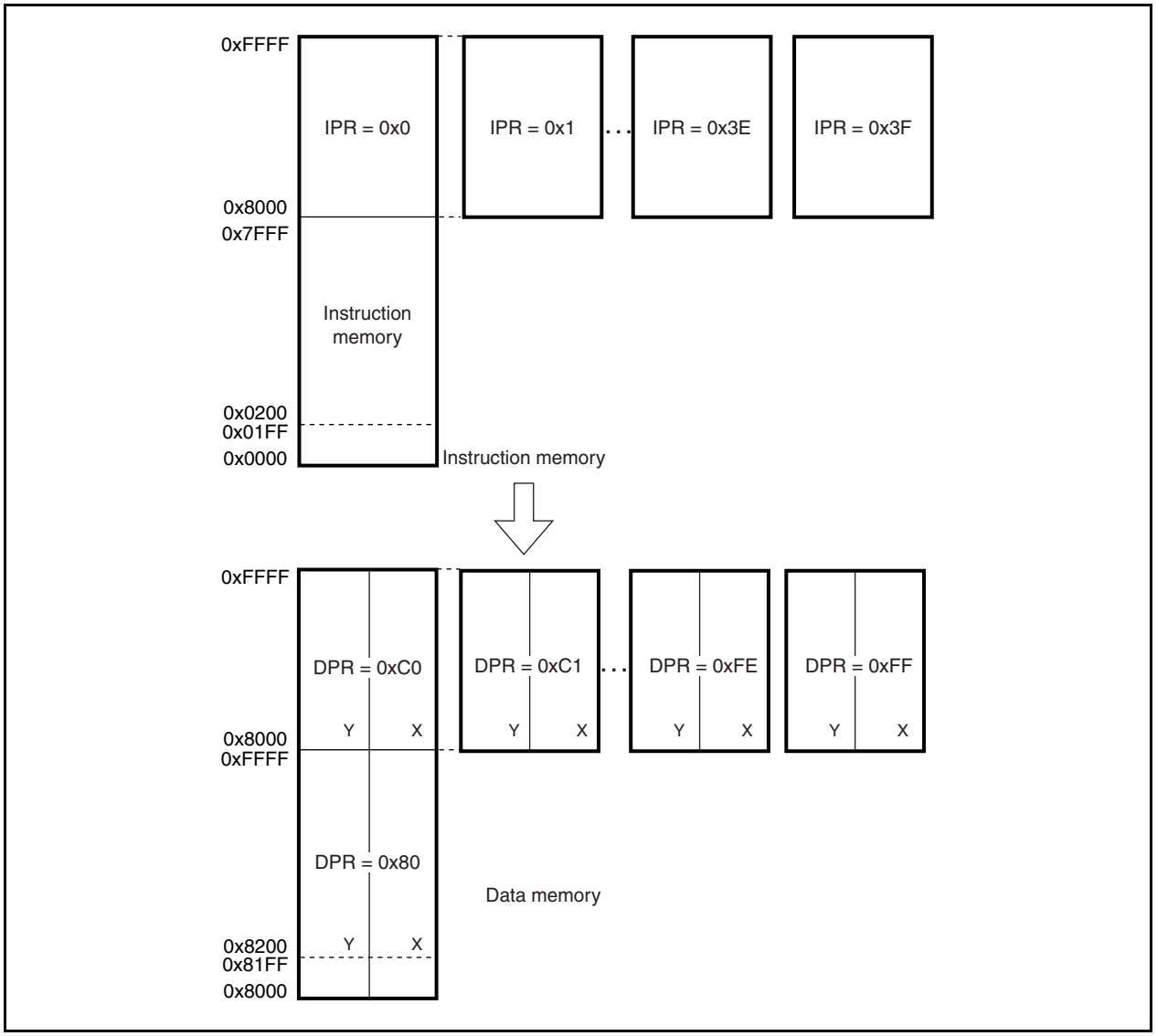
Figure 4-24 shows an image of such an alias. Since the instruction memory has 32-bit width while the data memory has 16-bit width, the lower 16 bits of the instruction memory contents are aliased in X memory and the higher 16 bits of the instruction memory contents are aliased in Y memory.

In order to access from page memory, an offset of 0x8000 is required to access the lower 32-Kword space (0x0000 to 0x7FFF) in instruction memory.

When accessing instruction memory contents as data, a conflict may occur if there is access to the same memory area (such as access to the memory area from 0x0000 to 0x7FFF when DPR = 0x80) as the access instruction's area. In such cases, one wait is inserted and data access is performed after the instruction has been fetched.

Caution Since μPD77210 Family devices do not have an instruction exception interrupt function, if values that were written to memory as data are fetched as an instruction, the instruction will not be recognized even if it is not a correct instruction. Note with caution that this can result in a runaway program.

Figure 4-24. Aliasing of Instruction Memory as Data Memory



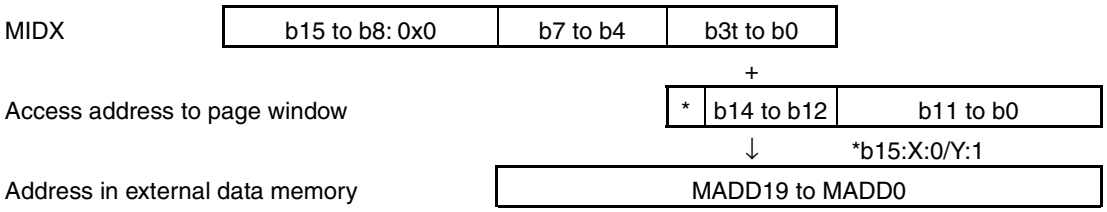
4.5.4 External data memory map

An external data memory space of 1 Mword × 16 bits is supported. The two access methods are as follows.

(1) Access using page memory (when DPR = 0x3F) as a window
(2) Access via the MIO data register (MDT)

For description of the second method, see **5.7 External Data Memory Interface (MIO)**. The first access method is described below.

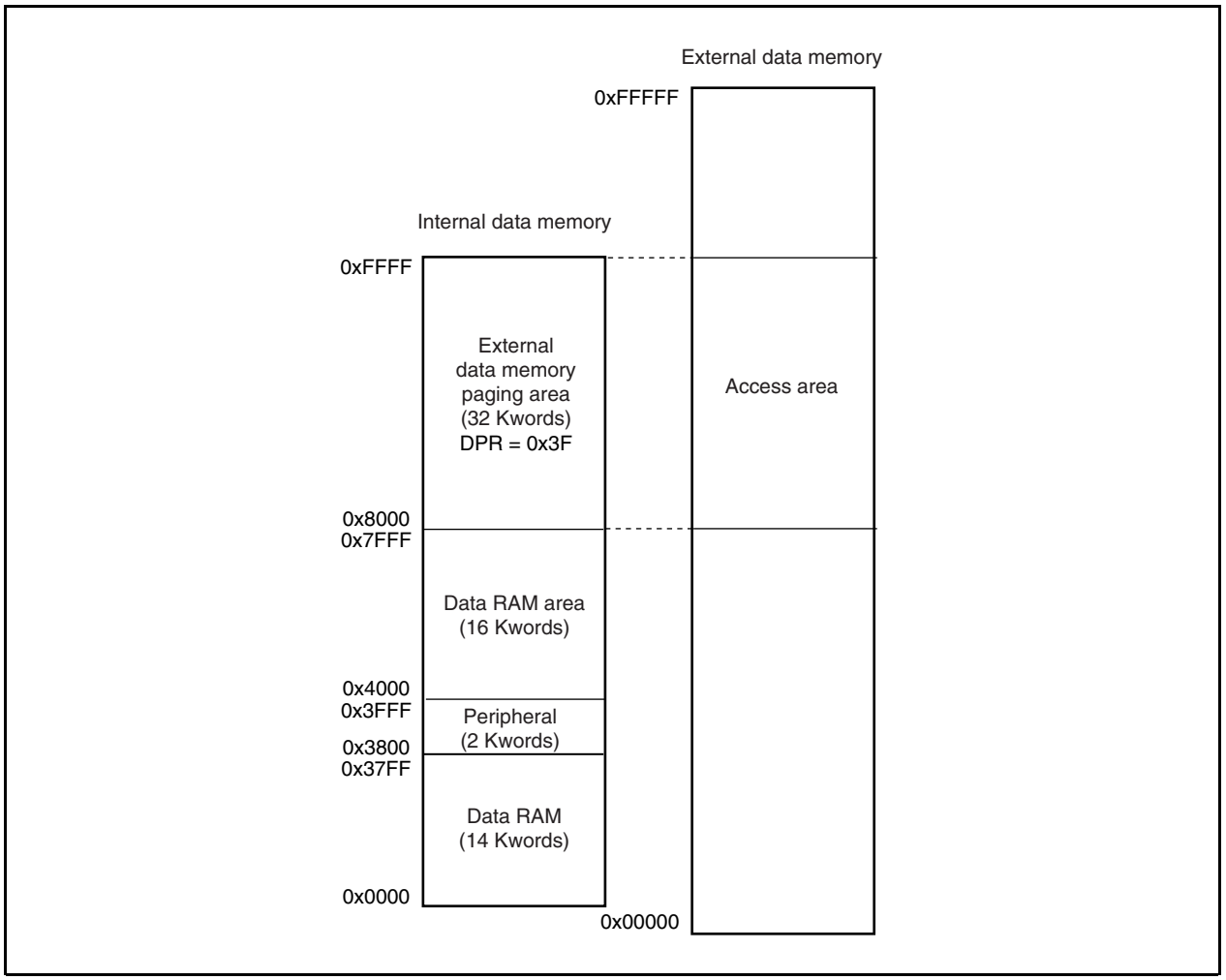
Figure 4-25 shows an image of how page memory can be used as a window for accessing external data memory. When accessing while using page memory as a window, the accessed address is a 20-bit address that is the sum of the target page memory address and the value of the MIO index register (MIDX). However, this addition is based on "0" being used as the address's MSB for page memory access when accessing X memory, while "1" is being used as the address's MSB for page memory access when accessing Y memory. An image of this is shown below.



For example, when MIDX = 0x11 while DPR = 0x3F, access is as follows.

Access to 0x8000 in X memory → Access to 0x11000 in external memory
Access to 0x8000 in Y memory → Access to 0x19000 in external memory

Figure 4-25. Image of Access to External Data Memory Map



4.5.5 Addressing mode

The μPD77210 Family is provided with a powerful architecture to realize high-speed, flexible data memory access. The X and Y memory areas are addressed by completely independent but functionally identically addressing units. This subsection describes the architecture and addressing modes implemented.

(1) Function of each part of addressing unit

The functions of the blocks (see **Figure 4-22 Data Addressing Unit**) and the registers in the addressing unit are as follows.

(a) Data pointers (DP0 to DP7)

These eight 16-bit registers are used for indirect addressing. DP0 to DP3 are used to specify an address of the X memory space, while DP4 to DP7 are used to specify an address of the Y memory space.

The values of DP0 to DP7 can be input/output via the main bus.

(b) Index registers (DN0 to DN7)

These eight 16-bit registers modify DP0 to DP7. After the memory has been accessed, DPn is modified by the value of DNn (n: 0 to 7, each corresponds respectively). The values of DN0 to DN7 can be input/output via the main bus.

The valid number range of this register is given by –32,768 (0x8000) to +32,767 (0x7FFF).

(c) Modulo registers (DMX, DMY)

These two 16-bit registers specify the ring count range when DP0 to DP7 are modified during the ring count operation performed.

The ring count range for DP0 to DP3 is specified by DMX. DMY is used to specify that for DP4 to DP7.

The values of DMX and DMY can be input/output via the main bus.

The valid number range of this register is given by +1 (0x0001) to +32,767 (0x7FFF).

(d) Address ALUs (XAA, YAA: X and Y Address ALUs)

These two 16-bit ALUs are used to modify DP0 to DP7.

XAA is used to modify DP0 to DP3, while YAA modifies DP4 to DP7.

(e) Bit reverse circuits (XBRC, YBRC: X and Y Bit Reverse Circuits)

When a bit reverse access is performed, these circuits output an address that reverses the order of the DP0 to DP7 values, so that the highest bit becomes the lowest, and vice versa.

(f) Multiplexer (MUX)

This circuit selects one of several signals for output.

(2) Types of addressing modes

The data memory addressing modes are hierarchically classified below.

There are one type of direct addressing mode and seven types of indirect addressing modes that are implemented by using data pointers (DPs) as the base address indicator.

- Direct addressing
- Indirect addressing
  * DPn (no change)
  * DPn++ (post increment)
  * DPn– – (post decrement)
  * DPn## (post index addition)
  * DPn%% (post modulo index addition)
  * !DPn## (pre-bit reverse and post index addition)
  * DPn##imm (immediate addition)

(a) Direct addressing

Direct addressing is to directly express an address value and address division (X or Y) in an instruction word. Data of 16 bits is exchanged between a specified address of a specified division (X or Y) and a general-purpose register via X or Y data bus.

For details of the instruction word, refer to **μPD77016 Family Instructions User's Manual**.

Example 1: Load

```
R0H = *0x1234:X;
```

16-bit data is loaded to the H part (middle 16 bits) of general-purpose register R0 from address 0x1234 of the X memory.

Example 2: Store

```
*0x1234:X = R0H;
```

16-bit data is stored to address 0x1234 of the X memory from the H part (middle 16 bits) of general-purpose register R0.

Caution The X and Y memory spaces cannot be accessed simultaneously by means of direct addressing.

(b) Indirect addressing

In all the indirect addressing modes, the DPn register (data pointer) is used. The basic features of indirect addressing are summarized below.

- As the address value, the current value of specified DPn is output in all the modes except the bit reverse index addition mode. In the bit reverse index addition mode, the current value of the specified DPn is reversed and output (refer to **Figure 4-26**).
- If it is specified to modify DPn, DPn is modified after the data memory has been accessed.
- The modified DPn value, i.e. the new address, is effective from the next instruction onwards.
- DPn alone cannot be modified.
- If an immediate value has been set to DPn, either by an inter-register transfer or an immediate value set instruction, the new address is effective from the next but one following instruction (refer to ***μ*PD77016 Family Instructions User's Manual**).
- DP0 to DP3 are used to access the X memory space, and DP4 to DP7 are used to access the Y memory space.

Each indirect addressing mode is described next.

<1> *DPn (no change)

The memory is accessed with the value of DPn. The value of DPn is preserved after the access has been completed.

Example:

```
R1L = *DP0;
```

16-bit data is loaded from the X memory address indicated by the value of DP0 to the L part (lower 16 bits) of R1.

<2> *DPn++ (post increment)

The memory is accessed with the value of DPn. The value of DPn is incremented (+1) after the access has been completed.

Example:

```
R2H = *DP4++;
```

16-bit data is loaded from the Y memory address indicated by the value of DP4 to the H part (middle 16 bits) of R2, and then the value of DP4 is incremented.

<3> *DPn– – (post decrement)

The memory is accessed with the value of DPn. The value of DPn is decremented (Ð1) after the access has been completed.

Example:

```
R3E = *DP1--;
```

8-bit data (the lower 8 bits of the 16 bits) is loaded from the X memory address indicated by the value of DP1 to the E part (higher 8 bits) of R3, and then the value of DP1 is decremented.

<4> *DPn## (post index addition)

The memory is accessed with the value of DPn. After the access has been completed, the value of DNn is added to DPn. Note that the n-th index register DNn corresponds only to the n-th data pointer DPn (e.g. DN1 to DP1).

The valid number range of DNn is given by –32,768 (0x8000) to +32,767 (0x7FFF).

Example:

```
R4L = *DP5##;
```

16-bit data is loaded from the Y memory address specified by the value of DP5 to the L part (lower 16 bits) of R4, and then the value of DN5 is added to DP5.

<5> *DPn%% (post modulo index addition)

The memory is accessed with the value of DPn. After the access has been completed, the value of DNn is added to DPn. In addition, modulo adjustment is made by DMX or DMY (DMX is used when n = 0 to 3, and DMY is used when n = 4 to 7). Note that the n-th index register DNn corresponds only to the n-th data pointer DPn (e.g. DN1 to DP1).

The valid number range of DNn is given by –32,768 (0x8000) to +32,767 (0x7FFF).

For the details of modulo index addition and modulo adjustment, refer to **<9> Modulo index addition and cyclic buffer**.

Example:

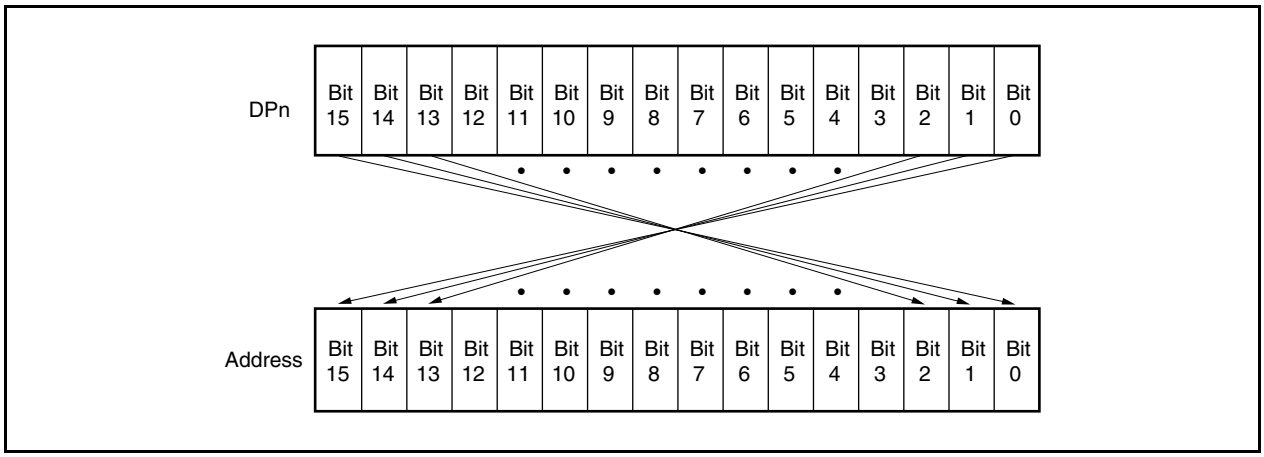
```
R5H = *DP3%%;
```

16-bit data is loaded from the X memory address specified by the value of DP3 to the H part (middle 16 bits) of R5. Then the value of DN3 is added to DP3, and modulo adjustment is made by DMX.

<6> *!DPn## (pre-bit reverse and post index addition)

The memory is accessed by using the value that reverses the order of the DPn values, as shown in Figure 4-26, and the value of DNn is added to DPn after the access has been completed. Note that the value of DNn having the same number as that of DPn must be added to DPn (for example, DN1 to DP1). This function is suitable for applications such as FFT.

Figure 4-26. Reversing Bits of DPn



Example:

```
R6H = *!DP6##;
```

16-bit data is loaded from the Y memory address specified by the reserved bits of DP6 to the H part (middle 16 bits) of R6, and then value of DN6 is added to DP6.

Remark DPn is not modified by bit-reversed access to the address, and the bit-reversed address is not fed back to DPn. After bit-reversed access, the value of DNn is added to the value of DP6 (original DP6 value) before bit reversion.

<7> *DPn##imm (post immediate addition)

The memory is accessed with the value of DPn, and immediate value imm is added to DPn after the access has been completed.
The valid number range of imm is given by –32,768 (0x8000) to +32,767 (0x7FFF).

Example:

```
R7L = *DP2##100;
```

16-bit data is loaded from the X memory address specified by the value of DP2 to the L part (lower 16 bits) of R7, and then immediate value “100” is added to DP2.

Caution The immediate addition addressing mode cannot be used to access the X and Y memories simultaneously.

<8> Modifying data pointers

Table 4-17 summarizes how the data pointers are modified as a result of accessing the memory in the above addressing modes.

Table 4-17. Modifying Data Pointers

(a) Operation

Example	Operation
DPn	No modification
DPn++	DPn ← DPn + 1
DPn– –	DPn ← DPn – 1
DPn##	DPn ← DPn + DNn (Values of corresponding DN0 to DN7 are added to DP0 to DP7). **Example**: DP0 ← DP0 + DN0
DPn%%	(n = 0 to 3) DPn = ((DPL + DNn) mod (DMX + 1)) + DPH
	(n = 4 to 7) DPn = ((DPL + DNn) mod (DMY + 1)) + DPH
!DPn##	Reverses bits of DPn and then accesses memory. After memory has been accessed, DPn ← DPn + DNn
DPn##imm	DPn ← DPn + imm

(b) Value range

Register Name	Hexadecimal	Decimal
DPn	0x0000 to 0xFFFF	0 to +65,535
DNn	0x8000 to 0x7FFF	–32,768 to +32,767
DMX/DMY	0x0001 to 0x7FFF	+1 to +32,767
Immediate value	0x8000 to 0x7FFF	–32,768 to +32,767

<9> Modulo index addition and cyclic buffer

The modulo index addition mode is provided for configuring a cyclic buffer (also called a ring buffer).

- **Rule of operation**

After the memory has been accessed by using the value of DPn, DPn is modified. At this time, the operation is performed according to the following rules:

(1) Executes operation of $DP_L = DP_L + DNn$.

(2) If $DP_L \leq DMa$ as a result,
$DPn = DP_L + DP_H$ is treated as the operation result.
If not (i.e., when $DP_L > DMa$),
$DPn = (DP_L + DNn) \bmod (DMa + 1) + DP_H$ is treated as the operation result.

Where,

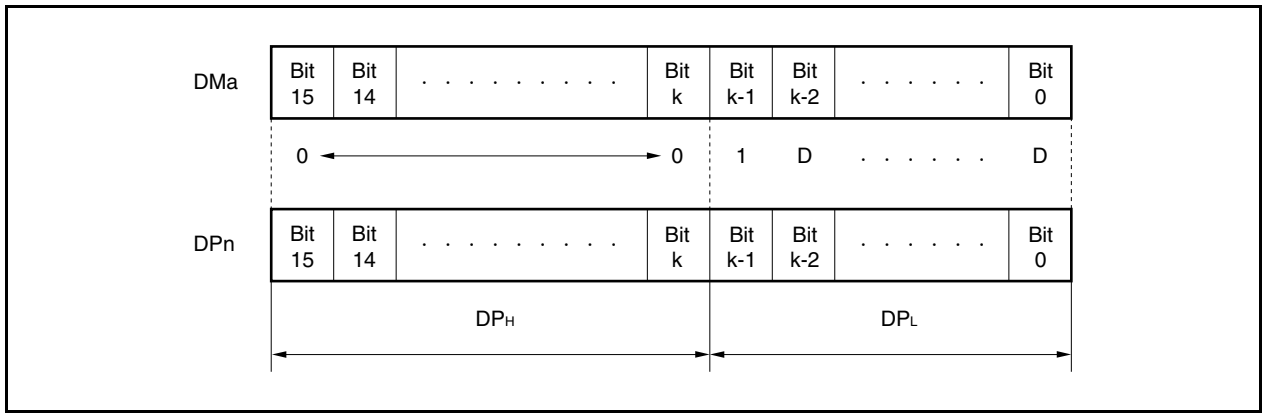
DP_H: lower k bits of the initial value of DPn, which is 0, if the value of DMa is in the range of [2k, 2 (k – 1)] (refer to **Figure 4-27**.)

DP_L: value of lower k bits of DPn in the above case (refer to **Figure 4-27**.)

DMa: specified DPn corresponding to DMX or DMY

Remark The process (2) above is called modulo adjustment.

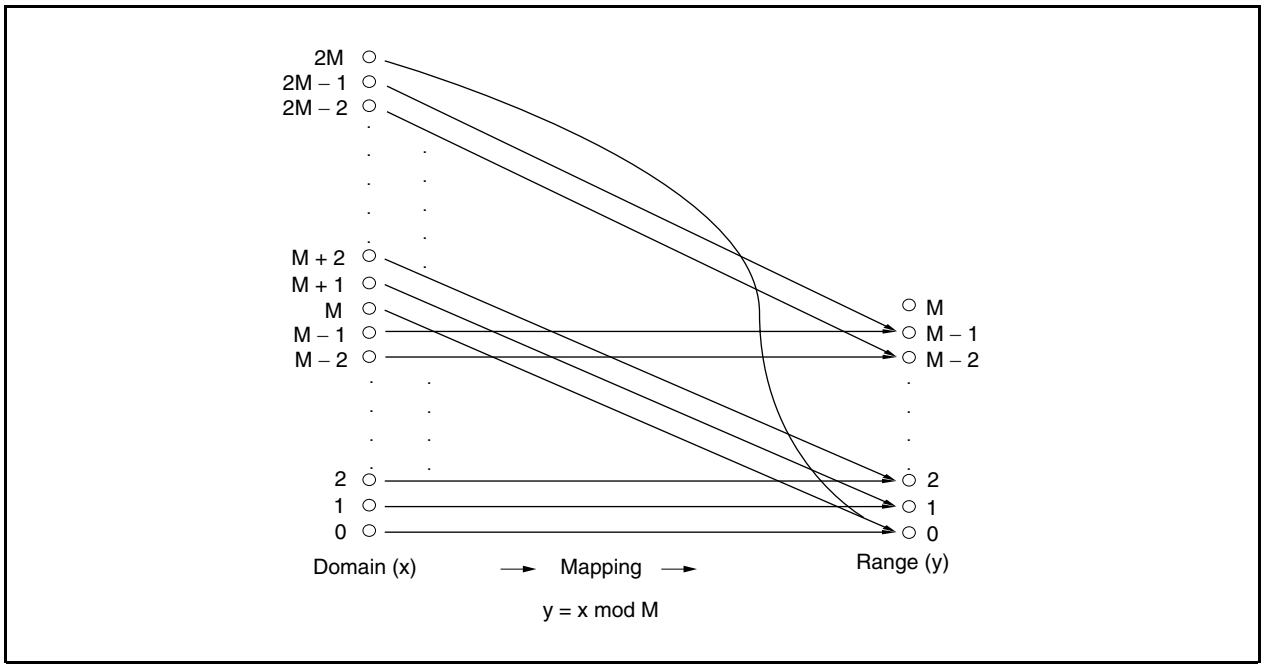
Figure 4-27. Division of DPn



- **Meaning**

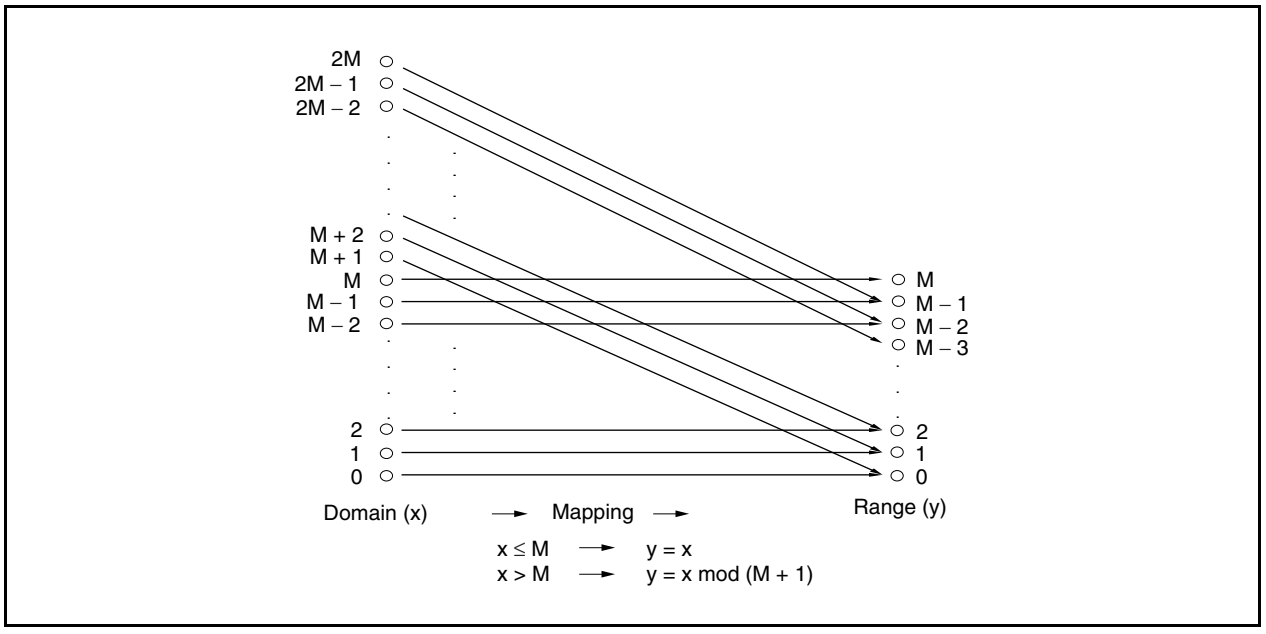
  The ordinary modulo operation can be considered as the mapping shown in Figure 4-28.

Figure 4-28. Mapping of Ordinary Modulo Operation



In contrast, modulo adjustment can be considered as the mapping shown in Figure 4-29.

Figure 4-29. Mapping of Modulo Adjustment



The difference between the two in terms of the range is that the usable buffer size in Figure 4-28 is M, while it is M + 1 in Figure 4-29, where the value set to DMa is M, because the range in this case corresponding to the size of the buffer. Consequently, the maximum buffer size of 0x8000 can be used, as described below, despite the maximum set value of DMa is 0x7FFF.

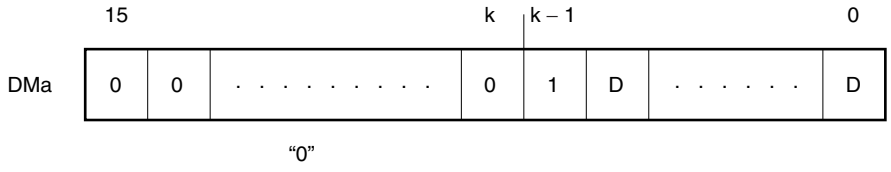
- **Operation range of ring count**

The first address of the range in which a ring count operation is executed in connection with the modulo index addition is determined by the values of the data pointer and modulo register.

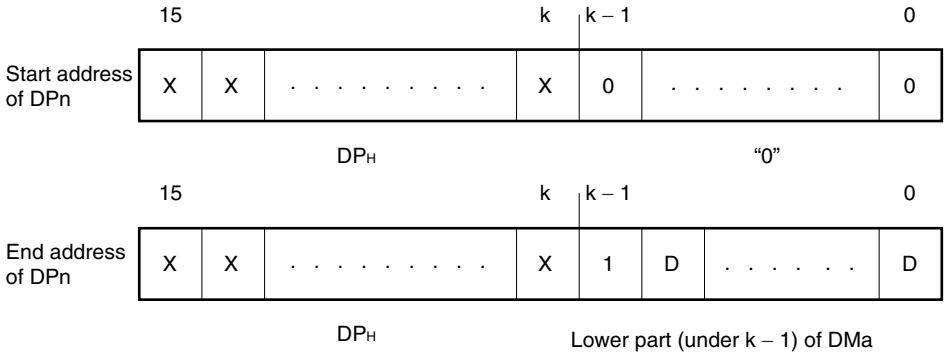
Where the value of the modulo register (DMX/Y) is $2k - 1 \leq DMX/Y < 2k$, the starting address of the ring count operation is the 16-bit value whose higher "16 – k" bits are the same as those of the data pointer and whose lower k bits are zeros. The end address is the value whose higher "16 – k" bits are the same as those of the data pointer and whose lower k bits are the same as those of the modulo register. The higher "16 – k" bits of the data pointer always remain unchanged.

Ring Count Operation Range

When DMa is set as follows,



Ring count start address and end address are like follows.



- **Restriction**

Observe the following restrictions in executing modulo addressing:

- Keep the range of DMa to [1 to 0x7FFF].
- Make sure that the absolute value of the value of DNn does not exceed DMa.

Caution Because 0 cannot be set to DMa, a cyclic buffer with buffer size = 1 cannot be configured.

• **Example of modulo index addition**

An example of operation process when a cyclic buffer is configured by using modulo index addition is shown below.

Example 1.

```
DMX=0x7;
DN0=1;
DP0=0x0;
```

At this time, the value of DP0 is updated as follows by means of modulo index addition:

```
DP0=0x0
  ↓      0x0+1
DP0=0x1
  ↓      0x1+1
DP0=0x2
  ↓      0x2+1
DP0=0x3
  ↓      0x3+1
DP0=0x4
  ↓      0x4+1
DP0=0x5
  ↓      0x5+1
DP0=0x6
  ↓      0x6+1
DP0=0x7
  ↓      0x7+1=0x8 → 0x8–(0x7+1)=0x0
DP0=0x0
  ↓      0x0+1
DP0=0x1
  ↓      0x1+1
DP0=0x2
  ↓      0x2+1
DP0=0x3
  ↓      0x3+1
DP0=0x4
  :
```

Example 2.

```
DMX=0xA;
DN0=3;
DP0=0x10;
```

At this time, the value of DP0 is updated as follows by means of modulo index addition:

```
DP0=0x10
  ↓     0x10+3
DP0=0x13
  ↓     0x13+3
DP0=0x16
  ↓     0x16+3
DP0=0x19
  ↓     0x19+3=0x1C → 0x1C–(0xA+1)=0x11
DP0=0x11
  ↓     0x11+3
DP0=0x14
  ↓     0x14+3
DP0=0x17
  ↓     0x17+3
DP0=0x1A
  ↓     0x1A+3=0x1D → 0x1D–(0xA+1)=0x12
DP0=0x12
  :
```

4.6 Operation Unit

The general-purpose registers in this unit are source of all operands and destination of all results of arithmetic/logic operations. The general-purpose registers are connected to the

- Main bus for inter-register transfers
- X and Y data bus for data exchange with the data memories and peripheral registers

All kinds of arithmetic/logic operations which are part of the following instruction types are carried out in the operation unit:

- Trinomial instructions
  All operations which involve 3 input operands, e.g.

  MADD: R0 = R0 + R1H * R2H

- Binomial instructions
  All operations which involve 2 input operands, e.g.

  ADD: R0 = R2 + R3

- Monomial instructions
  All operations which involve 1 input operand, e.g.

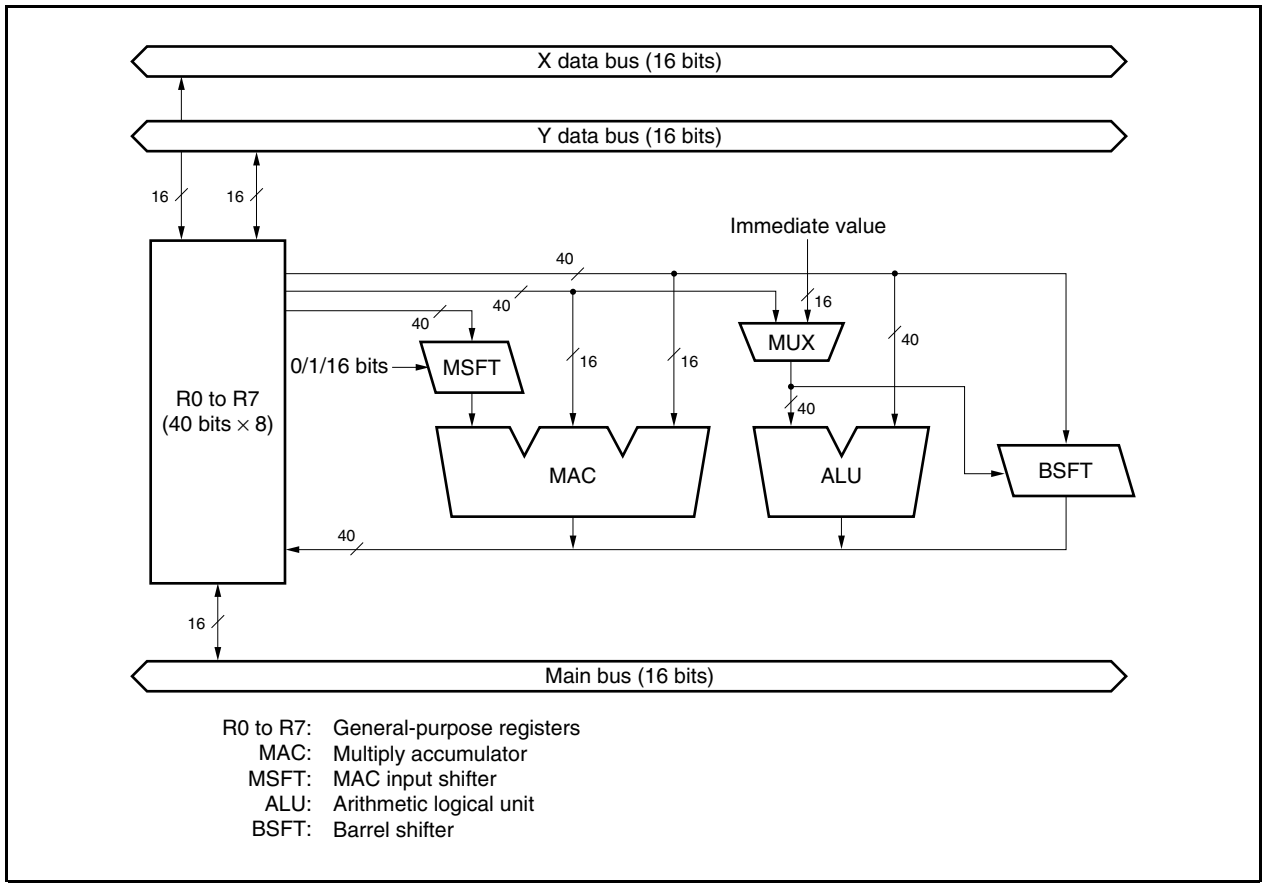
  NEG: R0 = −R1

The section describes in details the functions and data formats of the general-purpose registers R0 to R7, of the multiplier /accumulator MAC and the MAC input shifter MSFT, of the arithmetic/logic unit ALU, and of the barrel shifter BSFT. For the block diagram of this unit, refer to **4.6.1 Block configuration**.

4.6.1 Block configuration

Figure 4-30 is the block diagram of the operation unit.

Figure 4-30. Operation Unit



4.6.2 General-purpose registers and data formats

The features of the general-purpose registers are as follows:

- 40-bit registers
- Eight registers (R0 to R7) available
- Function as input/output parameters of operation instructions
  (only general-purpose registers, in addition to immediate data, can be described as parameters of operation instructions)
- Exchange data with X and Y data memories and peripheral registers (load/store function)
- Transfer data with other registers

(1) Partitioning of the general-purpose registers

Although a general-purpose register consists of 40 bits, the register is divided into three parts, as follows, so that only a specified part of the register can be used to transfer and load/store data or to execute arithmetic operations. In this case, the three parts are exclusive to each other.

- L part: Bits 15 to 0 (lower 16 bits)
- H part: Bits 31 to 16 (middle 16 bits)
- E part: Bits 39 to 32 (higher 8 bits)

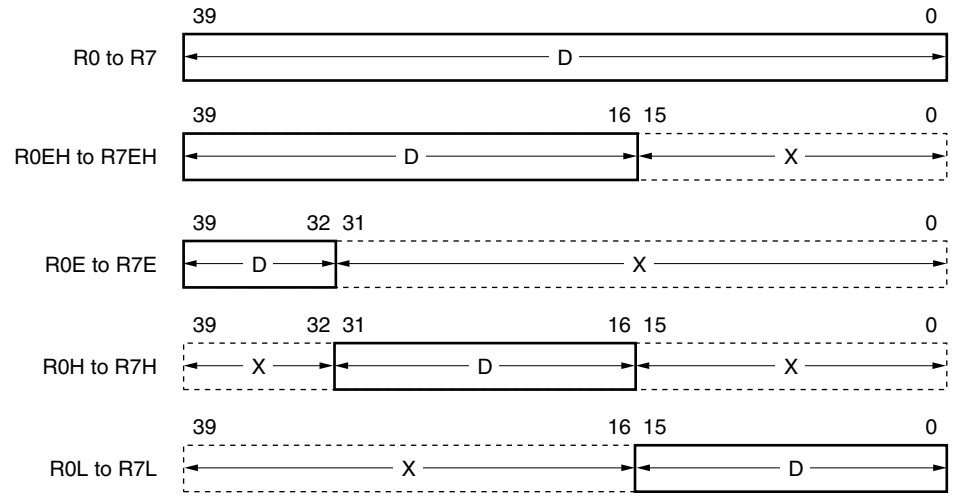
Depending on the type of arithmetic/logic operation respectively data transfer different parts of a general-purpose register are involved, as shown in Table 4-18.

The figure below shows these five formats (except R0HL to R7HL) with assembly names.

Table 4-18. Formats of General-Purpose Registers

	R0 to R7 40 Bits	R0L to R7L 16 Bits	R0H to R7H 16 Bits	R0E to R7E 8 Bits	R0HL to R7HL 32 Bits	R0EH to R7EH 24 Bits
MAC multiply/accumulate	√	√	√	–	–	–
MAC exclusive multiply	√	–	√	–	–	–
ALU	√	–	–	–	√	–
BSFT	√	√	–	–	–	–
X/Y bus transfer	√	√	√	√	–	√
Inter-register transfer	–	√	–	–	–	–

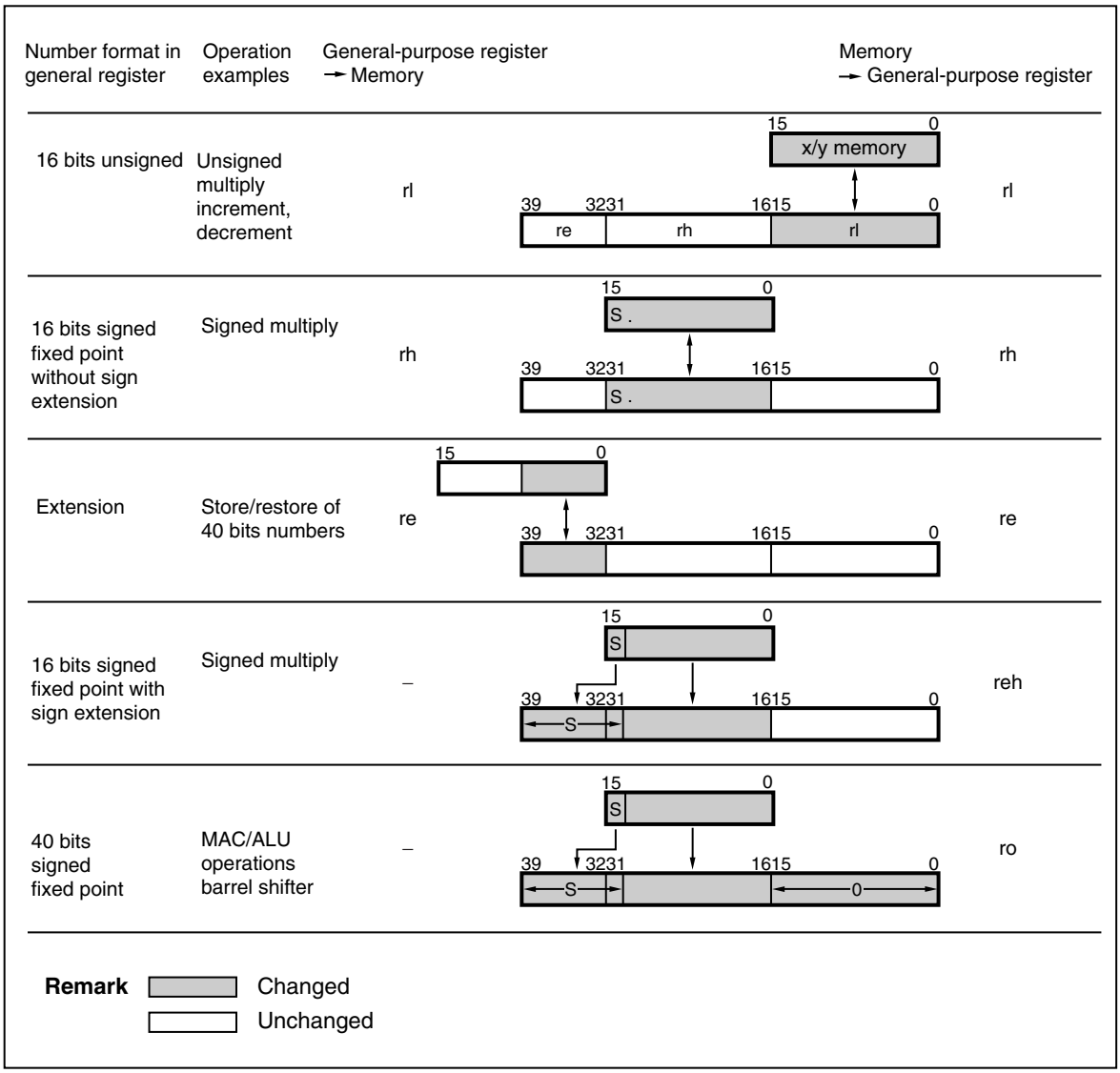
Figure 4-31. Formats of General-Purpose Registers



Remark D: Numeric value
X: Invalid

Figure 4-32 shows data exchange between general-purpose registers and data memory.

Figure 4-32. Data Exchange Between General-Purpose Registers and Data Memory



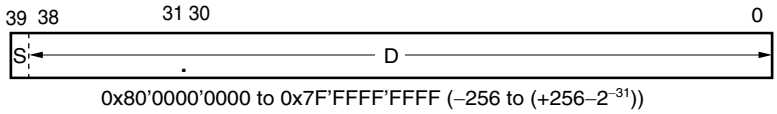
(2) Numeric format

The general-purpose registers of the μPD77210 Family can process fixed-point and integer data. The architecture places an emphasis on operations of fixed-point data, however.

(a) Fixed-point format

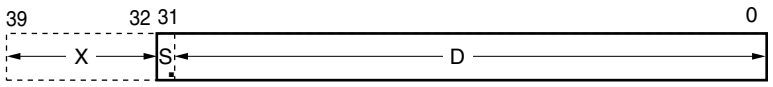
The fixed-point format uses the position between bits 31 and 30 as the decimal point. Fixed-point data can be expressed in three ways: in 40-bit, 32-bit, and 16-bit units.

- 40-bit data format (input for addition/subtraction/and/or/xor)



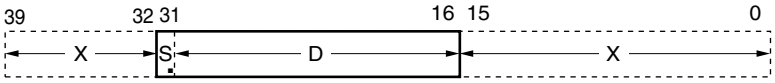
0x80'0000'0000 to 0x7F'FFFF'FFFF (-256 to $(+256-2^{-31})$)

- 32-bit data format (input for exponent instruction)



0x800'0000 to 0x7FFF'FFFF (-1 to $(+1-2^{-31})$)

- 16-bit data format (input for multiplication instruction)



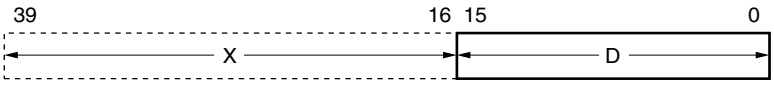
0x8000 to 0x7FFF (-1 to $(+1-2^{-15})$)

Remark The absolute value of data never exceeds 1 in the 32-bit fixed-point format or 16-bit fixed-point format. As long as an accumulative operation is executed on a general-purpose register with these formats as operands, therefore, the E part functions as an overflow absorbing area (called a head room). This function allows omission of judgment of overflow when 256 accumulative operations are performed even if it is assumed that an overflow of 1LSB (of the E part) occurs as a result of one accumulative operation.

(b) Integer format

The integer format is illustrated below.

- 16-bit data format (input for multiplication and shift instructions)



0x0000 to 0xFFFF (0 to 65,535)

4.6.3 Operation functions of multiply accumulator (MAC) and MAC input shifter (MSFT)

The multiply accumulator performs the following functions:

- Multiplication
  MPY: ro = rh * rh'

- Extends multiplication and its result to 40 bits and adds the result to specified general-purpose register
  MADD: ro = ro + rh * rh' (signed-signed multiply)
  MSUB: ro = ro – rh * rh' (signed-signed multiply)
  SUMA: ro = ro + rh * rl (signed-unsigned multiply)
  UUMA: ro = ro + rl * rl' (unsigned-unsigned multiply)

- Extends multiplication and its result to 40 bits and adds the result to the result of shifting specified general-purpose register 1 or 16 bits to the right
  MAS1: ro = (ro>>1) + rh * rh'
  MAS16: ro = (ro>>16) + rh * rh'

Caution MAC, ALU, and BSFT cannot operate simultaneously.

(1) Multiplication function

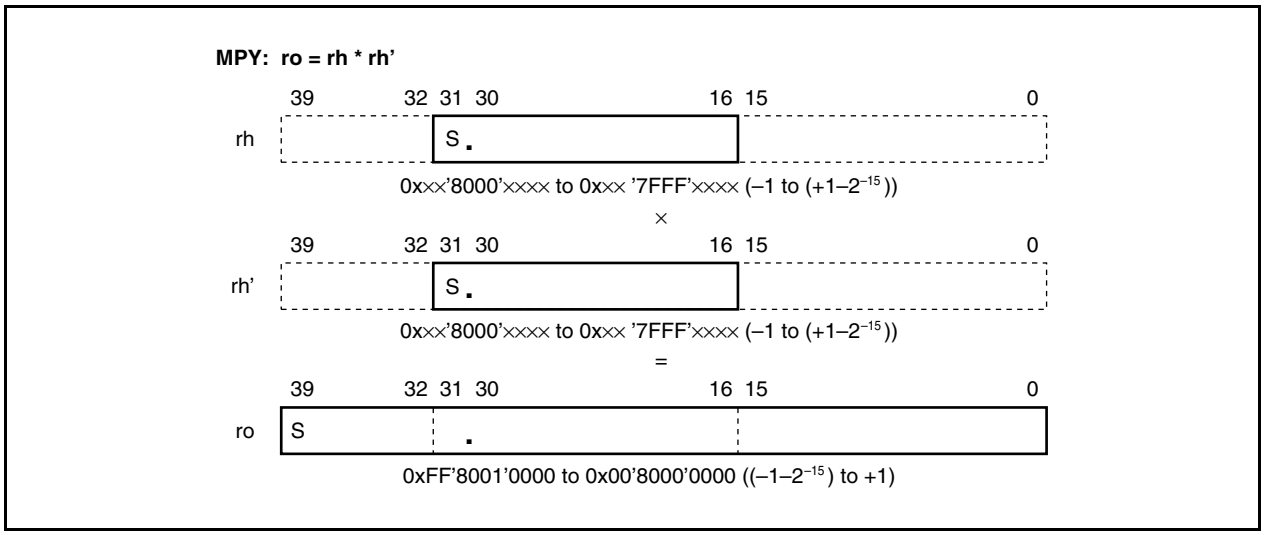
The multiplication function is implemented by the multiply accumulator (MAC). This function can be implemented in the following three ways, depending on the data type to be handled:

- Signed-signed multiply
- Signed-unsigned multiply
- Unsigned-unsigned multiply

(a) Signed-signed multiply

Both of the two operands are of signed 16-bit fixed-point type. Therefore, data is set to the H part of a general-purpose register whose bit 31 indicates the sign. A representation of this operation process is illustrated in Figure 4-33.

Figure 4-33. Signed-Signed Multiply



Remarks 1. If multiplication between 0x8000 (–1) is executed, the result is 0x00'8000'0000. However, because +1 cannot be expressed in the range of the 32-bit fixed-point format, an overflow occurs (extension bit re = 0x00 is different from the sign bit of the 32-bit format in this case). However, the value is accurate when viewed from the point of the 40-bit format (0x00'8000'0000 = +1).

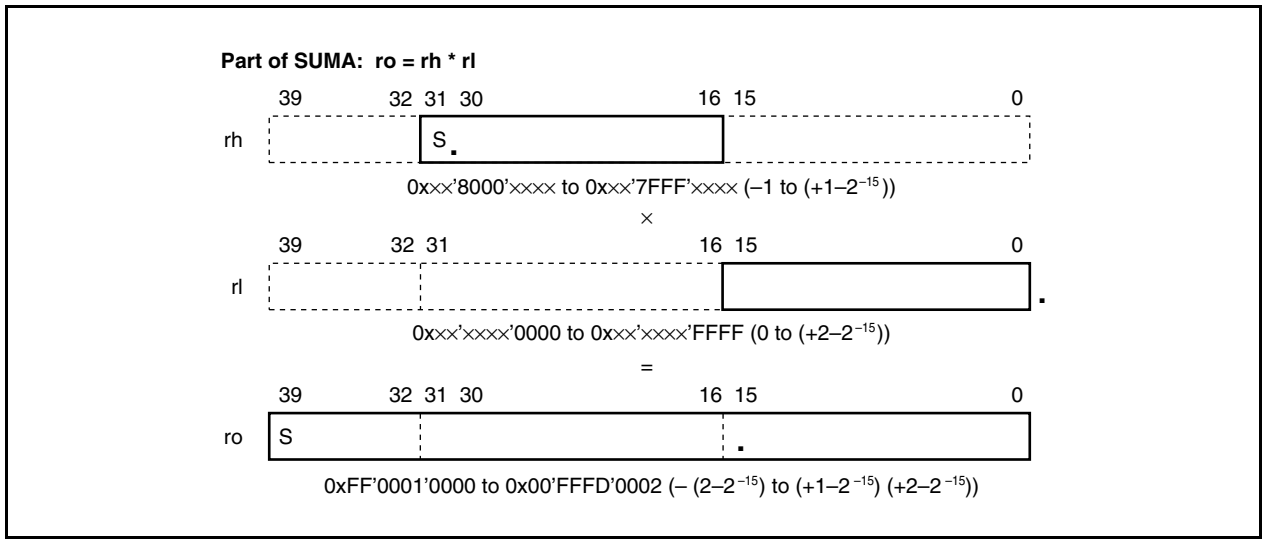
2. Since a multiplication of two 16-bit values produces maximum 31 valid bits, the LSB of the result registers is always 0.

(b) Signed-unsigned multiply

One of the two operands is set to the H part of a general-purpose register in the 16-bit fixed-point type, where bit 31 of the register indicates a sign. The other parameter is set to the L part of a general-purpose register in the integer format. Figure 4-34 shows the image of this operation process.

Caution There is no exclusive instruction that executes this operation. This operation is performed as part of the sign-unsign multiply add instruction.

Figure 4-34. Signed-Unsigned Multiply

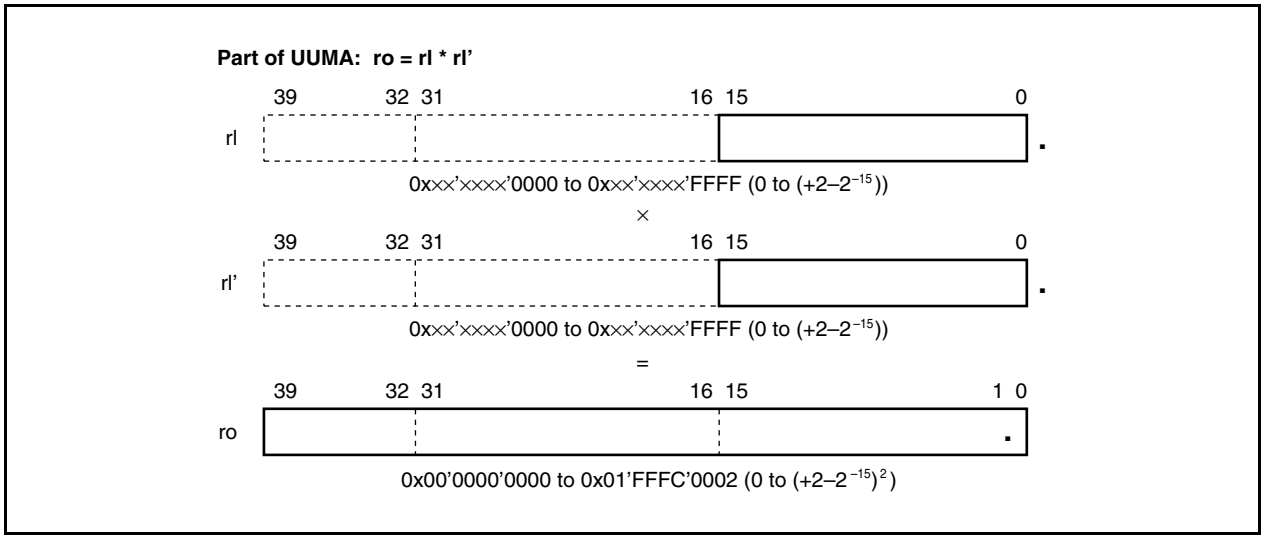


(c) Unsigned-unsigned multiply

Both of the two operands are set to the L parts of general-purpose registers in the integer format. Figure 4-35 shows the image of this operation process.

Caution There is no exclusive instruction that executes this operation. This operation is performed as part of the unsign-unsign multiply add instruction.

Figure 4-35. Unsigned-Unsigned Multiply



(2) Accumulative multiplication function (trinomial operation)

All the trinomial operations executed by the μPD77210 Family are accumulative multiplication. The accumulative multiplication can be implemented in the following three ways, depending on the shift command to the register that is used for the accumulative operation (two accumulative operations, accumulative addition and accumulative subtraction, can be executed, however). At this time, the shift processing is executed by the MAC input shifter (MSFT).

- Accumulative multiplication
- 1-bit shift accumulative multiplication
- 16-bit shift accumulative multiplication

The accumulative multiplication can also be classified into the following three types by the data type of the parameters used for the operation:

- Signed-signed multiply
- Signed-unsigned multiply
- Unsigned-unsigned multiply

In all, therefore, the following six types of trinomial operation instructions are available:

- Multiply add (signed-signed multiply and accumulative add)
- Multiply sub (signed-signed multiply and accumulative sub)
- Sign-unsign multiply add (signed-unsigned multiply and accumulative add)
- Unsign-unsign multiply add (unsigned-unsigned multiply and accumulative add)
- 1-bit shift multiply add (signed-signed multiply and accumulative add after 1-bit shift)
- 16-bit shift multiply add (signed-signed multiply and accumulative add after 16-bit shift)

The accumulative multiplication function implements trinomial operations where three parameters are used. Of these, two are the parameters for multiplication and the other is for accumulative operation. General-purpose registers are specified for these parameters. In this case, registers can be specified in duplicate.
Table 4-19 shows these combination.

Table 4-19. Accumulative Multiplication Function

		Signed-Signed	Signed-Unsigned	Unsigned-Unsigned
Multiply/ accumulate	MSFT 0 bit	ro = ro ± rh * rh' (MADD, MSUB)	ro = ro + rh * rl (SUMA)	ro = ro + rl * rl' (UUMA)
	MSFT 1 bit	ro = (ro>>1) + rh * rh' (MAS1)	–	–
	MSFT 16 bits	ro = (ro>>16) + rh * rh' (MAS16)	–	–
Exclusive multiply (Binomial operation)		ro = rh * rh' (MPY)	–	–

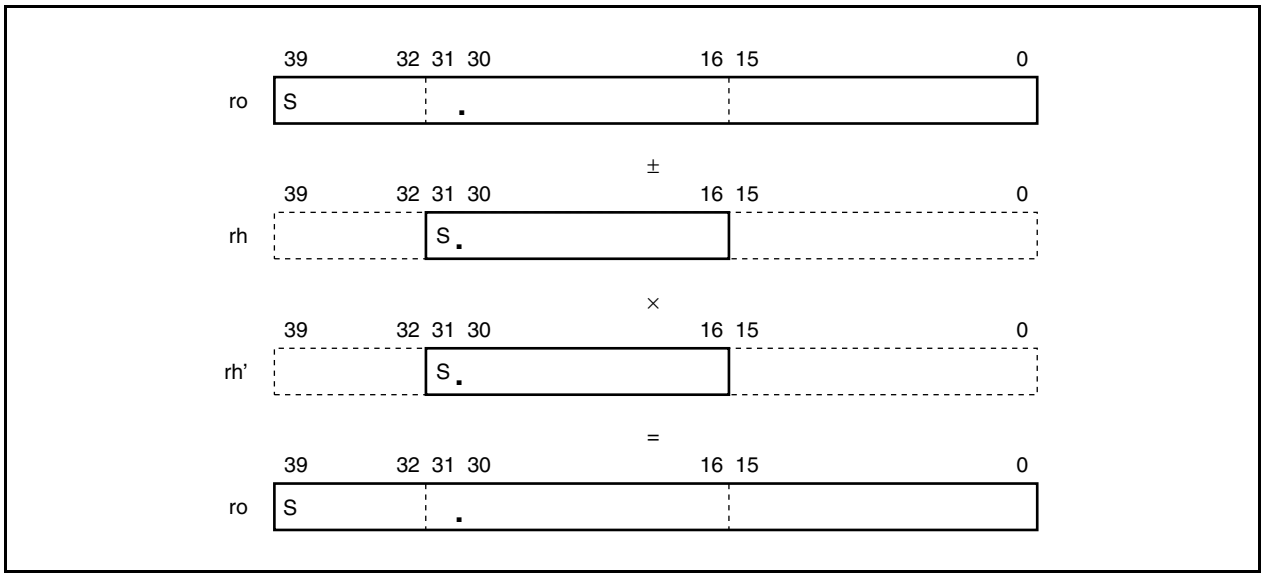
(a) Accumulative multiplication

The multiplier input operands are of (signed) 16-bit fixed point type. The multiplication result is added to respectively subtracted from a 40-bit fixed-point operand. The related instructions are:

MADD: ro = ro + rh * rh'
MSUB: ro = ro – rh * rh'

Figure 4-36 shows the image of this operation.

Figure 4-36. Accumulative Multiplication



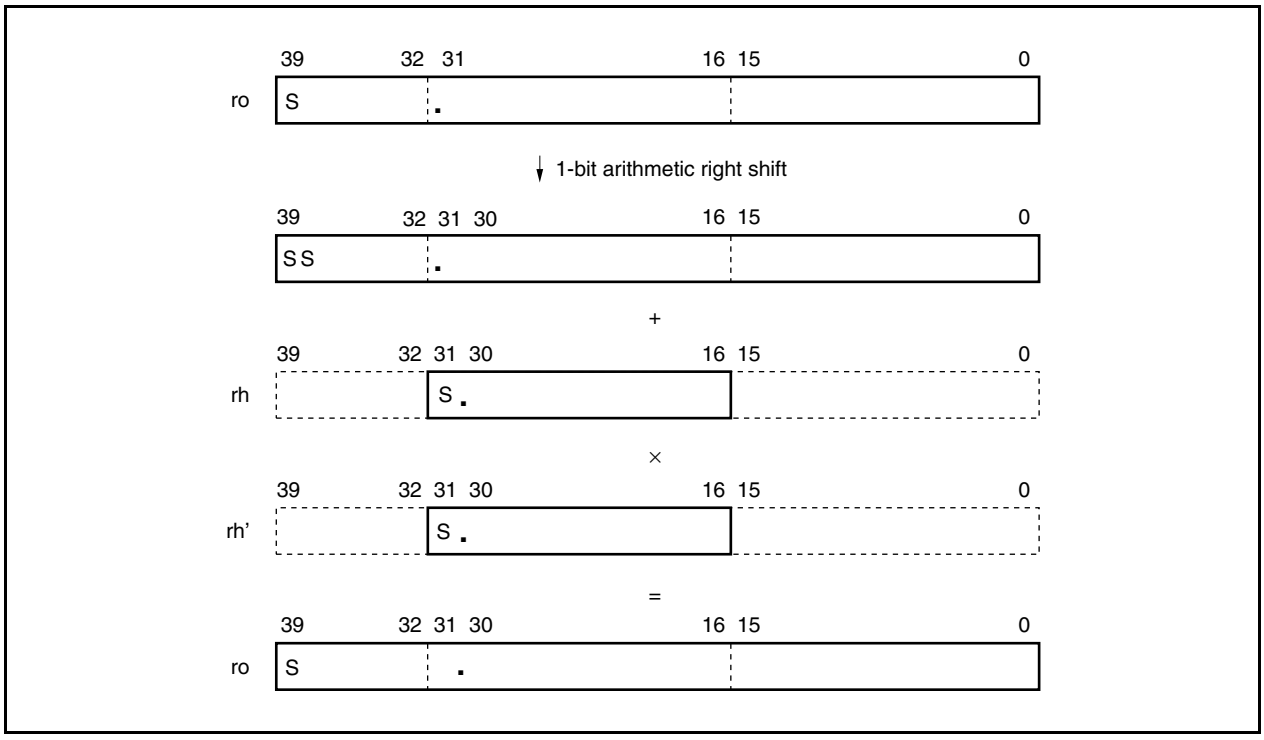
(b) 1-bit shift accumulative multiplication

The multiplier input operands are of (signed) 16-bit fixed point type. The multiplication result is added to a 1 bit right shifted 40-bit fixed-point operand. The related instruction is:

MAS1: ro = (ro>>1) + rh * rh'

Figure 4-37 shows the image of this operation:

Figure 4-37. 1-Bit Shift Accumulative Multiplication



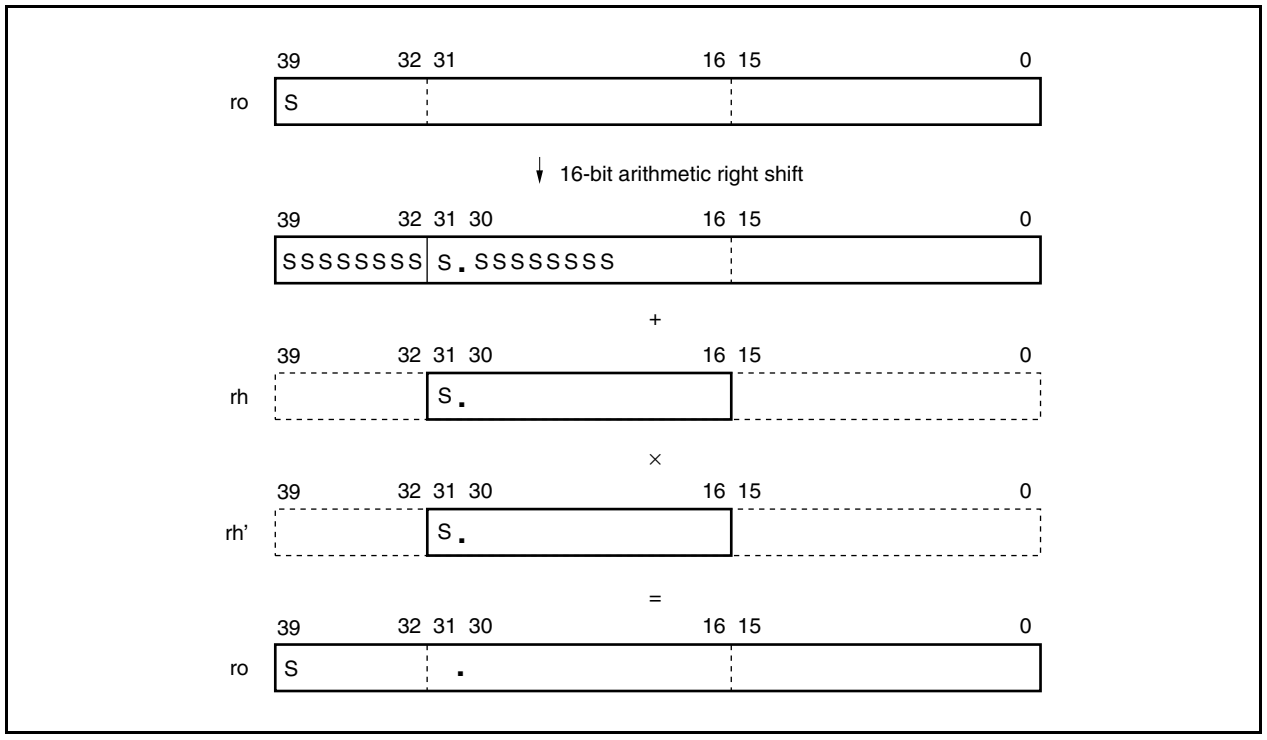
(c) 16-bit shift accumulative multiplication

The multiplier input operands are of (signed) 16-bit fixed point type. The multiplication result is added to a 16 bit right shifted 40-bit fixed-point operand. The related instruction is:

MAS16: ro = (ro>>16) + rh * rh'

Figure 4-38 shows the image of this operation:

Figure 4-38. 16-Bit Shift Accumulative Multiplication



4.6.4 Operation functions of arithmetic and logic unit (ALU)

The arithmetic and logic unit (ALU) executes an arithmetic or logical operation on two or one 40-bit input data, and outputs one 40-bit data.

As both two operands for a binomial operation, general-purpose registers can be specified, or a register can be specified as one of the operands with immediate data specified as the other (immediate data cannot be used with the LT instruction, however). If general-purpose registers are specified as both operands, they can be in duplicate. Any general-purpose register can be specified for a monomial operation. It is also possible to specify any general-purpose register to store the result of the operation.

Caution MAC, ALU, and BSFT cannot operate simultaneously.

(1) Arithmetic operation

(a) Binomial arithmetic operation

The following binomial arithmetic operation instructions are available. For each instruction, refer to ***μ*PD77016 Family Instructions User's Manual**.

- Multiply instruction (MPY: executed by MAC)
- Add instruction (ADD)
- Immediate add instruction (IADD)
- Subtract instruction (SUB)
- Immediate subtract instruction (ISUB)
- Less-than instruction (LT)

(b) Monomial arithmetic operation

The following monomial arithmetic operation instructions are available. For each instruction, refer to ***μ*PD77016 Family Instructions User's Manual**.

- Clear instruction (CLR)
- Increment instruction (INC)
- Decrement instruction (DEC)
- Absolute value instruction (ABS)
- Two's complement instruction (NEG)
- Clip instruction (CLIP)
- Round instruction (RND)
- Exponent instruction (EXP)
- Substitute instruction (PUT) (mainly used for data transfer between general-purpose registers)
- Accumulative addition instruction (ACA)
- Accumulative subtraction instruction (ACS)
- Division instruction (DIV)

(2) Logical operation

(a) Binomial logical operation

The following binomial logical operation instructions are available. For each instruction, refer to **μPD77016 Family Instructions User's Manual**.

- And instruction (AND)
- Immediate and instruction (IAND)
- Or instruction (OR)
- Immediate or instruction (IOR)
- Exclusive or instruction (XOR)
- Immediate exclusive or instruction (IXOR)

(b) Monomial logical operation

The following monomial logical operation instruction is available. For each instruction, refer to **μPD77016 Family Instructions User's Manual**.

- One's complement instruction (NOT)

Caution The number range of immediate data is 0 to 0xFFFF (0 to 65,536), and set to bit 15 to 0. Each operation is executed with 40-bit data that 39 to 16 are 0 extended to this immediate 16-bit data.

4.6.5 Operation functions of barrel shifter (BSFT)

The barrel shifter (BSFT) executes shift operations. All the shift operations are binomial operations. The BSFT outputs any shift pattern as 40-bit data in one instruction cycle in response to 40-bit input data.

As both two operands for a binomial operation, general-purpose registers can be specified, or a register can be specified as one of the operands with immediate data specified as the other. If general-purpose registers are specified as both operands, they can be in duplicate. Any general-purpose register can be specified for a monomial operation. It is also possible to specify any general-purpose register to store the result of the operation.

Caution MAC, ALU, and BSFT cannot operate simultaneously.

(1) Shift operation instruction

All the shift operations are binomial operations. The following shift operations instructions are available. For each instruction, refer to **μPD77016 Family Instructions User's Manual**.

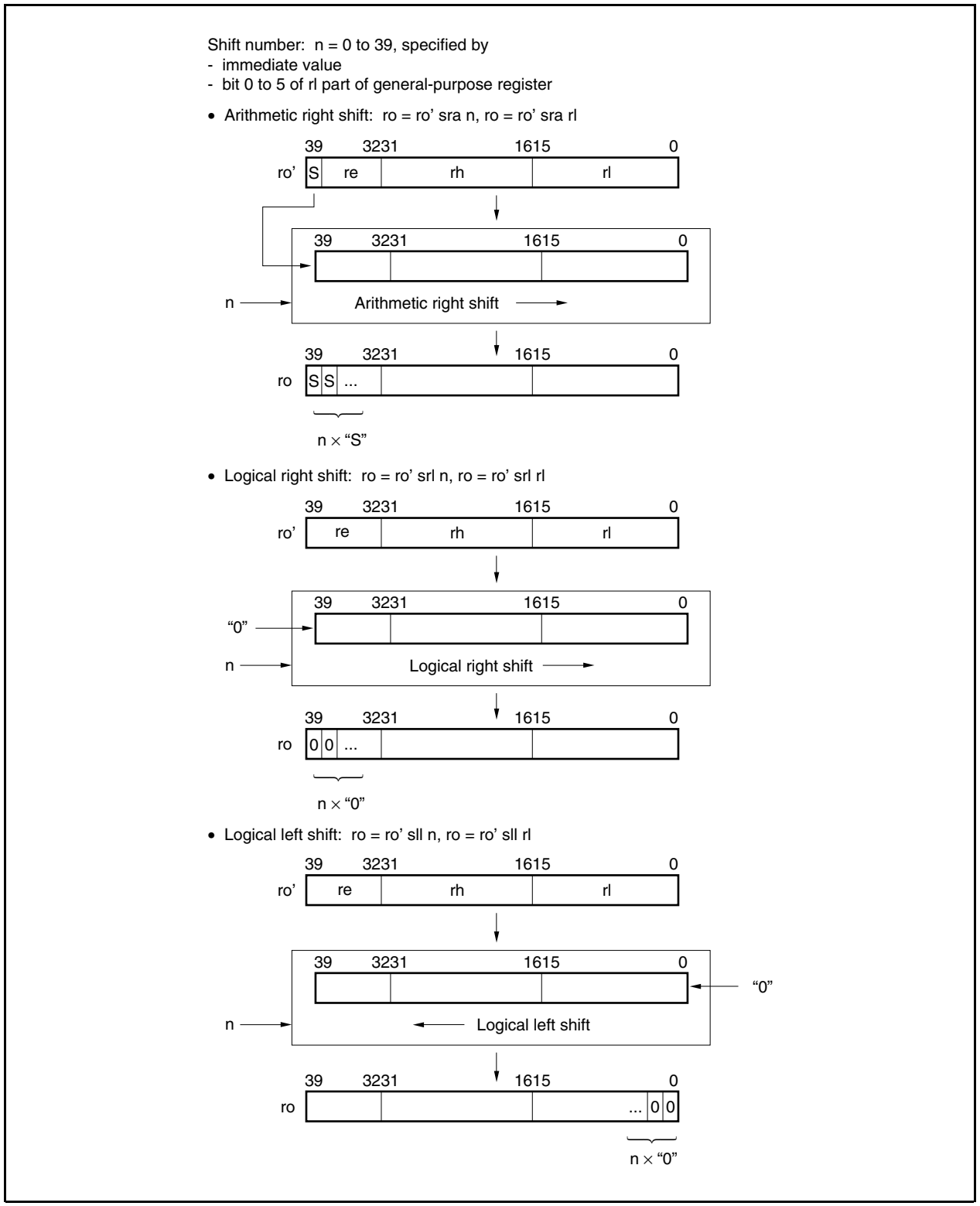
- Arithmetic right shift instruction (SRA)
- Immediate arithmetic right shift instruction (ISRA)
- Logical right shift instruction (SRL)
- Immediate logical right shift instruction (ISRL)
- Logical left instruction (SLL)
- Immediate logical left shift instruction (ISLL)

Caution The number range of general-purpose register or immediate data as shift value is 0 to 0x27 (0 to 39), and set to bit 5 to 0. The values of bit 15 to 6 are ignored.

(2) Shift operation function

Figure 4-39 shows each BSFT operations.

Figure 4-39. Barrel Shifter Operations



CHAPTER 5 PERIPHERALS

The following peripheral interface functions are included in all μPD77210 Family products. These peripheral units can be handled via registers that have been mapped to internal data areas in μPD77210 Family devices.

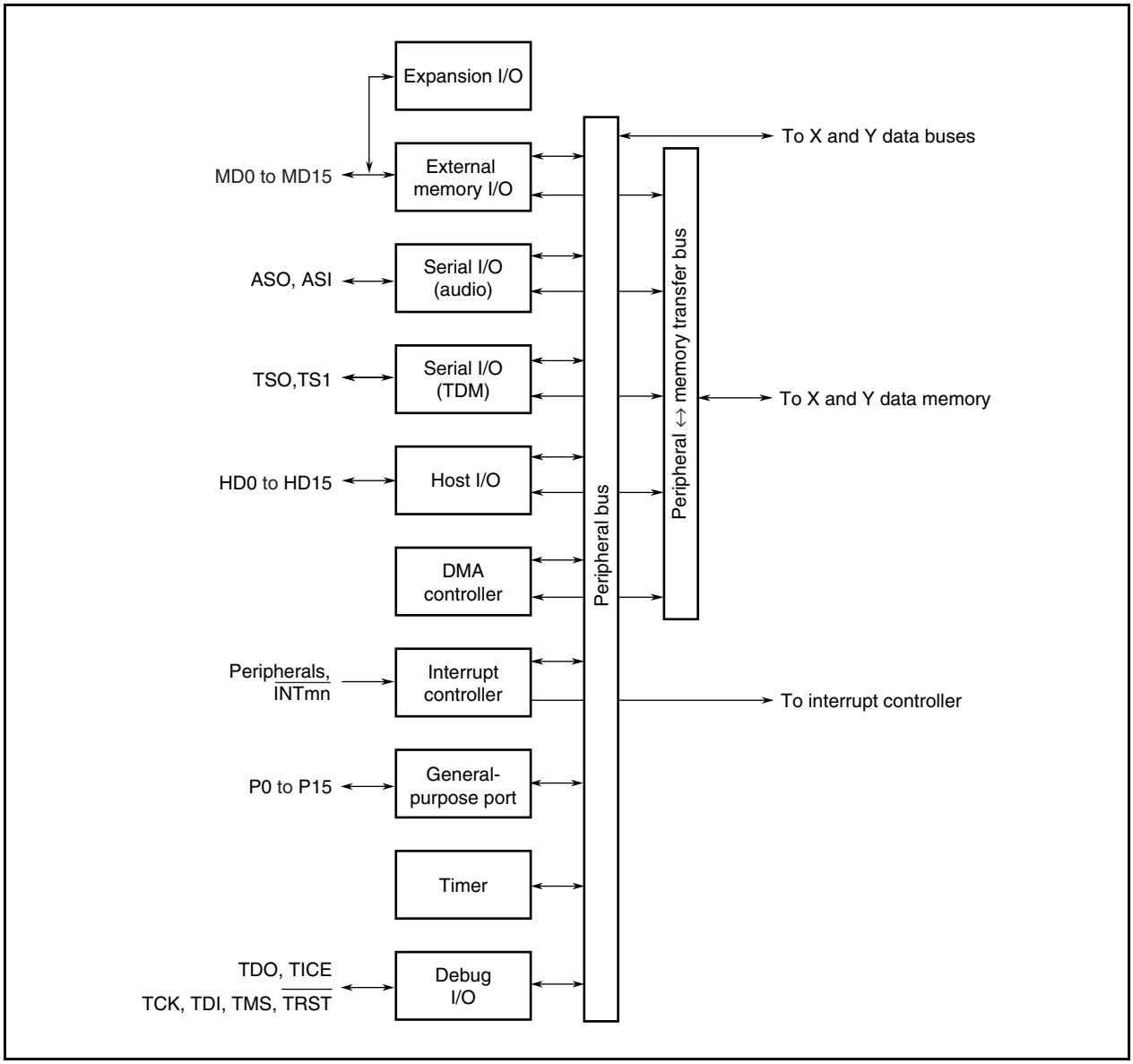
- Time division multiplexing (TDM) serial interface: TSIO
- Audio serial interface: ASIO
- Standard serial interface: SIO
- Host interface (parallel interface): HIO
- External data memory interface: MIO
- Peripheral memory transfer (DMA): PMT
- General-purpose I/O port: PIO
- Interrupt controller: INTC
- Timer: TIM
- Clock controller: CLKC
- Debug interface: IEIO[Note]
- Expansion interface: Additional I/O (μPD77213 only)
  SD card interface: SDCIF

Note The debug interface cannot be operated by a user program.

5.1 Block Configuration

Figure 5-1 shows the block configuration of the peripheral units.

Figure 5-1. Peripheral Units



5.2 Peripheral Registers

The peripheral registers and their mapping in the memory space are listed in Table 5-1.

Table 5-1. Memory Mapping of Peripheral Registers (1/3)

X/Y Memory Address	Register Name	Function	Peripheral Name
0x3800	TSDT/SDT1	TDM serial data register/serial data register 1	TSIO (SIO1)
0x3801	SST1	Serial status register 1	
0x3802	TSST	TDM serial status register	
0x3803	TFMT	TDM frame format register	
0x3804	TTXL	TDM transmit slot register (lower)	
0x3805	TTXH	TDM transmit slot register (higher)	
0x3806	TRXL	TDM receive slot register (lower)	
0x3807	TRXH	TDM receive slot register (higher)	
0x3808 to 0x380F	Reserved area	**Caution: Do not access this area.**	–
0x3810	ASDT/SDT2	Audio serial data register/Serial data register 2	ASIO (SIO2)
0x3811	SST2	Serial status register 2	
0x3812	ASST	Audio serial status register	
0x3813 to 0x381F	Reserved area	**Caution: Do not access this area.**	–
0x3820	HDT	Host interface data register	HIO
0x3821	HST	Host interface status register	
0x3822 to 0x383F	Reserved area	**Caution: Do not access this area.**	–
0x3840	MDT	Memory data register	MIO
0x3841	MSHW	Memory interface setup/hold width setting register	
0x3842	MCST	Memory interface control/status register	
0x3843	MWAIT	Memory interface wait register	
0x3844	MIDX	Direct access index register	
0x3845	MADRLI	Start address register (lower) for memory interface input	
0x3846	MADRHI	Start address register (higher) for memory interface input	
0x3847	MOFSI	Line offset register for memory interface input	
0x3848	MLENI	Line length register for memory interface input	
0x3849	MADRLO	Start address register (lower) for memory interface output	
0x384A	MADRHO	Start address register (higher) for memory interface output	
0x384B	MOFSO	Line offset register for memory interface output	
0x384C	MLENO	Line length register for memory interface output	
0x384D to 0x384F	Reserved area	**Caution: Do not access this area.**	–
0x3850	PMSA0	PMT status address register 0	PMT ch0
0x3851	PMS0	PMT size register 0	
0x3852	PMC0	PMT control register 0	
0x3853	PMP0	PMT address pointer 0	

Table 5-1. Memory Mapping of Peripheral Registers (2/3)

X/Y Memory Address	Register Name	Function	Peripheral Name
0x3854	PMSA1	PMT status address register 1	PMT ch1
0x3855	PMS1	PMT size register 1	
0x3856	PMC1	PMT control register 1	
0x3857	PMP1	PMT address pointer 1	
0x3858	PMSA2	PMT status address register 2	PMT ch2
0x3859	PMS2	PMT size register 2	
0x385A	PMC2	PMT control register 2	
0x385B	PMP2	PMT address pointer 2	
0x385C	PMSA3	PMT status address register 3	PMT ch3
0x385D	PMS3	PMT size register 3	
0x385E	PMC3	PMT control register 3	
0x385F	PMP3	PMT address pointer 3	
0x3860	PMSA4	PMT status address register 4	PMT ch4
0x3861	PMS4	PMT size register 4	
0x3862	PMC4	PMT control register 4	
0x3863	PMP4	PMT address pointer 4	
0x3864	PMSA5	PMT status address register 5	PMT ch5
0x3865	PMS5	PMT size register 5	
0x3866	PMC5	PMT control register 5	
0x3867	PMP5	PMT address pointer 5	
0x3868	PMSA6	PMT status address register 6	PMT ch6
0x3869	PMS6	PMT size register 6	
0x386A	PMC6	PMT control register 6	
0x386B	PMP6	PMT address pointer 6	
0x386C	PMSA7	PMT start address register 7	PMT ch7
0x386D	PMS7	PMT size register 7	
0x386E	PMC7	PMT control register 7	
0x386F	PMP7	PMT address pointer 7	
0x3870	PDT0	Port data register 0	PIO
0x3871	PCD0	Port command register 0	
0x3872	PDT1	Port data register 1	
0x3873	PCD1	Port command register 1	
0x3874	PDT2	Port data register 2	
0x3875	PCD2	Port command register 2	
0x3876	PDT3	Port data register 3	
0x3877	PCD3	Port command register 3	
0x3878 and 0x3879	Reserved area	**Caution: Do not access this area.**	–
0x387A to 0x387B	POWC	Power control register	Peripheral STOP mode
0x387C to 0x3879	Reserved area	**Caution: Do not access this area.**	–
0x3880	ICR0	Interrupt control register 0	INTC
0x3881	ICR1	Interrupt control register 1	
0x3882	ICR2	Interrupt control register 2	

Table 5-1. Memory Mapping of Peripheral Registers (3/3)

X/Y Memory Address	Register Name	Function	Peripheral Name
0x3883	ICR3	Interrupt control register 3	INTC
0x3884	ICR4	Interrupt control register 4	
0x3885	ICR5	Interrupt control register 5	
0x3886	ICR6	Interrupt control register 6	
0x3887	ICR7	Interrupt control register 7	
0x3888	ICR8	Interrupt control register 8	
0x3889	ICR9	Interrupt control register 9	
0x388A	ICR10	Interrupt control register 10	
0x388B	ICR11	Interrupt control register 11	
0x388C to 0x388F	Reserved area	**Caution: Do not access this area.**	–
0x3890	TIR0	Timer initialization register 0	TIM0
0x3891	TCR0	Timer count register 0	
0x3892	TCSR0	Timer control register 0	
0x3893	Reserved area	**Caution: Do not access this area.**	–
0x3894	TIR1	Timer initialization register 1	TIM1
0x3895	TCR1	Timer count register 1	
0x3896	TCSR1	Timer control register 1	
0x3897 to 0x389F	Reserved area	**Caution: Do not access this area.**	–
0x38A0	CEFR	Correction enable flag register	IMC
0x38A1	CPR0	Correction page register 0	
0x38A2	CAR0	Correction address register 0	
0x38A3	CLIR0	Correction instruction code register (higher) 0	
0x38A4	CUIR0	Correction instruction code register (lower) 0	
0x38A5	CPR1	Correction page register 1	
0x38A6	CAR1	Correction address register 1	
0x38A7	CLIR1	Correction instruction code register (higher) 1	
0x38A8	CUIR1	Correction instruction code register (lower) 1	
0x38A9 to 0x38AF	Reserved area	**Caution: Do not access this area.**	–
0x38B0	CLKC	Clock control register	CLKC
0x38B1 to 0x38BF	Reserved area	**Caution: Do not access this area.**	–
0x38C0	IPR	Instruction paging register	Page Register
0x38C1	DPR	Data paging register	
0x38C2 to 0x38CF	Reserved area	**Caution: Do not access this area.**	–
0x38D0	APCR[Note]	Expansion interface control register	Additional IO
0x38D1 to 0x3FFF	Reserved area	**Caution: Do not access this area.**	–

Note Available in μPD77213 only. This is a reserved area in the μPD77210.

Cautions 1. The register names in this table are not reserved words in the assembler or in C language. Therefore, when using these names with the assembler or C language, the user must define them.

2. As long as the address is the same, the same register can be accessed from both the X memory space and Y memory space.

3. Registers cannot be access from the X memory space and Y memory space at the same time, even if they are different registers.

5.3 Time Division Multiplexing (TDM) Serial Interface (TSIO)

The TDM serial interface (TSIO) is not only a standard mode serial interface that can be directly connected to a standard speech codec; it also includes a time division multiplexing mode serial transfer function that can be set for up to 128 slots. The TDM function makes it easy to communicate between multiprocessors.

For further description of the standard mode, see **5.5 Standard Serial Interface (SIO)**.

The TSIO's main features are as follows.

- Serial clock supply
  Enables external supply and common clock for input and output.
- Bit length
  Selectable as 8 or 16 bits and as MSB first or LSB first format.
- Internal handshaking
  Handshaking is enabled by means of polling, wait, or interrupt.
- Supports T1 and E1 formats
  Delay I/O settings for frame signals, and single-bit dummy bit I/O and frame signal polarity selection settings can be made.
- Supports up to 128 slots
  Up to 32 slots can be defined per frame, and frames can be positioned at 16-slot intervals, with up to 128 slots settable.

Figure 5-2 shows a block diagram of the TDM serial interface and Table 5-2 lists the registers.

Figure 5-2. Block Diagram of TDM Serial Interface

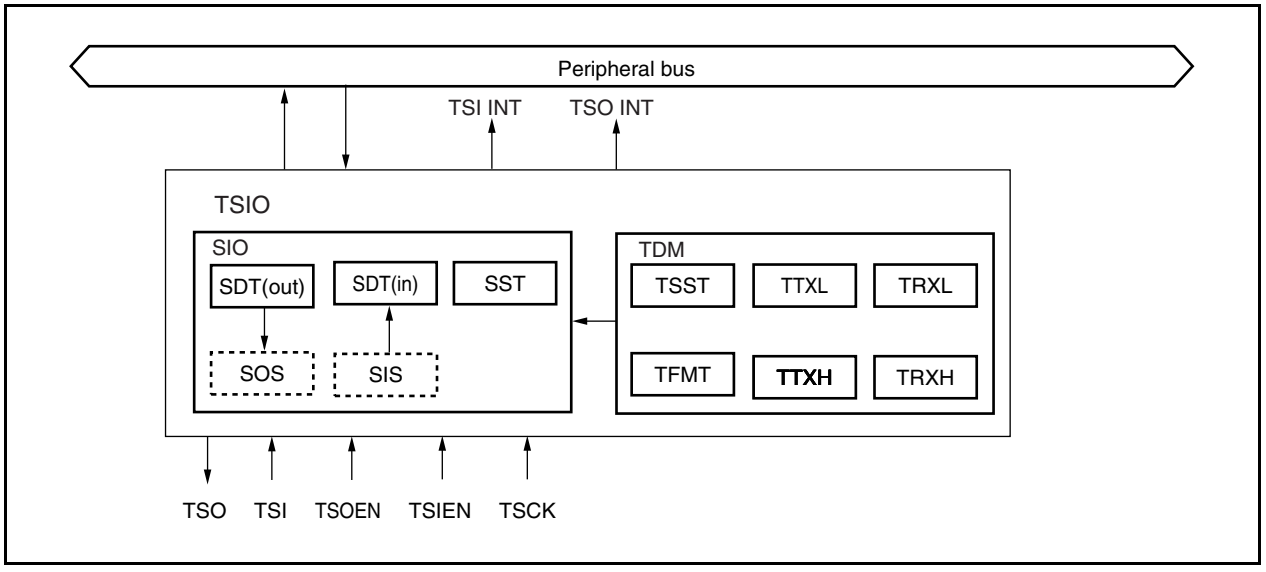


Table 5-2. TDM Serial Interface Registers

X/Y Memory Address	Register Name	Function	Load/store
0x3800	TSDT	TDM serial data register	L/S
0x3801	SST1	Serial status register 1	L/S
0x3802	TSST	TDM serial status register	L/S
0x3803	TFMT	TDM format register	L/S
0x3804	TTXL	TDM transmit slot register (lower)	L/S
0x3805	TTXH	TDM transmit slot register (higher)	L/S
0x3806	TRXL	TDM receive slot register (lower)	L/S
0x3807	TRXH	TDM receive slot register (higher)	L/S

5.3.1 TDM serial interface pins

(1) TSCK (serial clock - input)

This clock pin is used for TDM serial data input and output.

Input and output of serial data and output and sampling of various serial interface signals are performed in synchronization with the TSCK signal.

(2) TSOEN (TDM serial output enable – input)

This pin is used to input TDM serial data output enable signals.

This signal is sampled at the falling edge of TSCK.

This signal is asserted active (high level) when an external device is ready to input serial output data.

(3) TSIEN (TDM serial input enable – input)

This pin is used to input TDM serial data input enable signals.

This signal is sampled at the falling edge of TSCK.

This signal is asserted active (high level) when an external device is ready to output serial input data.

(4) TSI (TDM serial data input – input)

This pin is used to input TDM serial data.

This signal is sampled at the falling edge of TCLK.

(5) TSO (TDM serial data output – output)

This pin is used to output TDM serial data.

The output changes in synchronization with the rising edge of TCLK.

5.3.2 TDM serial interface registers

(1) TSDT (TDM serial data transfer register)

TSDT is a 16-bit register that is used for input and output of TDM serial data. It includes separate registers for input and output.

SDT(out) is a 16-bit register that sets data to be output. When a store instruction is executed to TSDT, data is input to this register from the peripheral bus.

When the serial output shift register (SOS) becomes empty, the value of the TSDT register is set to SOS and is output via the TSO pin. When the SSEF flag is 0 and a store instruction to SDT is executed, the SSER flag is set to 1 (store error).

SDT(in) is a 16-bit register that reads the input data. When a load instruction to TSDT is executed, data is output to the peripheral bus.

When the last bit is input to the serial input shift register (SIS) from the TSI pin, the value of SIS is set to TSDT. When the SLEF flag is 0 and a load instruction from SDT is executed, the SLER flag is set to 1 (load error).

(2) TSST (TDM serial status register)

TSST is a 16-bit register that indicates the mode setting and status of the TDM serial I/O. This register specifies the interface with the μPD77210 Family DSP core kernel, such as standard/TDM serial interface mode switching and frame signal handling, and indicates overrun and underrun.

The bit configuration of TSST is shown in Table 5-3. The value after reset is 0x5.

(3) SST1 (serial status register)

SST1 is a 16-bit register that indicates the mode setting and status of the serial I/O. This register indicates whether data is transferred MSB first or LSB first as well as specification of the μPD77210 Family DSP core kernel interface and overrun or underrun status. The bit configuration of SST1 is shown in Table 5-4. The value after reset is 0x00C2.

(4) TFMT (TDM frame format register)

TFMT is a 16-bit register that sets the format of TDM serial I/O. The bit configuration of TFMT is shown in Table 5-6. The value after reset is 0x0.

(5) TTXL and TTXH (TDM transmit slot registers)

These are 16-bit registers that are used to set the transmit access frames for TDM serial data output. TTXH specifies the higher 16 slots and TTXL specifies the lower 16 slots.

The slot that is assigned to be transmitted is the slot corresponding to a bit set to 0. No data is output from any slot set to 1. If multiple μPD77210 Family devices are connected, do not set the same slot to 0 in multiple μPD77210 Family devices. The value after reset is 0xFFFF.

(6) TRXL and TRXH (TDM receive slot registers)

These are 16-bit registers that are used to set the receive access frames for TDM serial data input. TRXH specifies the higher 16 slots and TRXL specifies the lower 16 slots.

The slot that is assigned for reception is the slot corresponding to a bit set to 0. Data in any slot set to 1 is ignored. The value of these registers is 0xFFFF when reset.

(7) SOS (serial output shift register)

SOS is a 16-bit shift register that shifts TDM serial data while outputting it from the TSO pin. When the specified number of bits have been output, new data is input from TSDT.

SOS is not connected to the peripheral bus.

(8) SIS (serial input shift register)

SIS is a 16-bit shift register that inputs the bit strings input from the TSI pin. When the specified number of bits have been input, new data is output to TSDT.

SIS is not connected to the peripheral bus.

Table 5-3. Bit Configuration of TSST

Bit	Name	Function	Load/Store
15 to 8	Reserved	• Values other than 0 cannot be written. • Undefined during a read operation	–
7	FSP	Frame synchronization signal's polarity setting bit • 0: Synchronized with the rising edge of the frame synchronization signal (default) • 1: Synchronized with the falling edge of the frame synchronization signal	L/S
6	FSD	Data I/O delay setting bit • 0: Starts data I/O to the frame synchronization signal at the next SCK clock (default) • 1: Starts data I/O to the frame synchronization signal after 2 SCK clocks	L/S
5	OBP	Dummy bit (1 bit) insert enable bit • 0: Dummy bit not inserted (default) • 1: Dummy bit inserted	L/S
4	TDME	Standard/TDM serial interface selection bit • 0: Standard serial mode (default) • 1: TDM serial mode	L/S
3	ERRTX	Transmit frame error flag • 0: No TFSx frame synchronization error (default) • 1: TFSx frame synchronization error exists This flag is 1 when the next frame signal is input in a shorter time than the number of clocks of the specified frame. When this bit is set (= 1), its status does not change until a μPD77210 Family device writes a 0 to it.	L/S
2	Reserved	• Values other than 0 cannot be written. • Undefined during a read operation	–
1	ERRRX	Receive frame error flag • 0: No RFSx frame synchronization error (default) • 1: RFSx frame synchronization error exists This flag is 1 when the next frame signal is input in a shorter time than the number of clocks of the specified frame. When this bit is set (= 1), its status does not change until a μPD77210 Family device writes a 0 to it.	L/S
0	Reserved	• Values other than 0 cannot be written. • Undefined during a read operation	–

Table 5-4. Bit Configuration of SST1 (1/2)

Bit	Name	Function	Load/Store
15	SOTF	Serial output transfer format setting bit • 0: Serial output with MSB first (default) • 1: Serial output with LSB first	L/S
14	SITF	Serial input transfer format setting bit • 0: Serial input with MSB first (default) • 1: Serial input with LSB first	L/S
13	SOBL	Serial output word length setting bit • 0: 16-bit serial output (default) • 1: 8-bit serial output	L/S
12	SIBL	Serial input word length setting bit • 0: 16-bit serial input (default) • 1: 8-bit serial input	L/S
11	SSWE	TSDT store wait enable bit • 0: Does not use store wait function (default) • 1: Uses store wait function. Inserts wait when the μPD77210 Family device stores data to TSDT when SSEF = 0.	L/S
10	SLWE	TSDT load wait enable bit • 0: Does not use load wait function (default) • 1: Uses load wait function. Inserts wait when the μPD77210 Family device loads data from TSDT when SLEF = 0.	L/S
9	SICM	Serial input continuous mode setting flag • 0: Enters single serial input mode after completion of current serial input. • 1: Enters single serial input mode to start serial input.	L/S
8	SIEF	Single serial input enable flag • 0: (default) • 1: Starts serial input processing in single serial input mode (only once). SIEF that has been set to 1 is automatically reset in the next instruction cycle.	L/S
7	SOE	Serial output enable bit • 0: Use of serial output disabled • 1: Use of serial output enabled (default)	L/S
6	SIE	Serial input enable bit • 0: Use of serial input disabled • 1: Use of serial input enabled (default)	L/S
5	SORST[Note]	Serial output reset enable bit This resets the serial output circuit. Afterward, SSER = 0 and SSEF = 1. This bit is automatically cleared to zero after serial output is reset. • 0: (default) • 1: SO reset request	L/S

Table 5-4. Bit Configuration of SST1 (2/2)

Bit	Name	Function	Load/Store
4	SIRST[Note]	Serial input reset enable bit This resets the serial input circuit. Afterward, SLER = 0 and SLEF = 0. This bit is automatically cleared to zero after serial input is reset. • 0: (default) • 1: SI reset request	L/S
3	SSER	TSDT store error flag • 0: No error (default) • 1: Error When SSEF = 0, this bit is set (= 1) when a μPD77210 Family device writes to SDT. When this bit is set (= 1), its status does not change until a μPD77210 Family device writes a 0 to it.	L/S
2	SLER	SDT load error flag • 0: No error (default) • 1: Error When SLEF = 0, this bit is set (= 1) when a μPD77210 Family device reads SDT. When this bit is set (= 1), its status does not change until a μPD77210 Family device writes a 0 to it.	L/S
1	SSEF	SDT store enable flag • This bit is set (= 1) when the value of SDT has been transferred to the serial output shift register. • It is cleared to zero when a μPD77210 Family device loads data from SDT.	L
0	SLEF	SDT load enable flag • This bit is set (= 1) when the value of SDT has been transferred to the serial input shift register. • It is cleared to zero when a μPD77210 Family device loads data from SDT.	L

Note Once the serial reset bit has been set (= 1), TSIO cannot be accessed for three cycles. Results are not reflected normally if TSIO is accessed during these three cycles.
The serial reset bit must not be set (= 1) at the same time that another SST bit is being set.

Table 5-5. Combination of SICM and SIEF Bits

SICM	SIEF	Function
0	0	Status transition mode • This mode is set when serial input is not performed.
1	0	Continuous serial input mode
0	1	Single serial input mode • When SIEF bit is set (= 1), it is automatically reset (to 0) during the next instruction cycle.
1	1	Setting prohibited

Table 5-6. Bit Configuration of TFMT

Bit	Name	Function	Load/Store
15 to 13	SRTX	Transmit slot position setting bit • 000: Slots 0 to 31 (default) • 001: Slots 16 to 47 • 010: Slots 32 to 63 • 011: Slots 48 to 79 • 100: Slots 64 to 95 • 101: Slots 80 to 111 • 110: Slots 96 to 127 • 111: Slots 112 to 143	L/S
12 to 8	FLTX	Transmit slot count setting bit • 00000: 32 slots per frame (default) • 00001 to 11111: 1 to 31 slots per frame	L/S
7 to 5	SRRX	Receive slot position setting bit • 000: Slots 0 to 31 (default) • 001: Slots 16 to 47 • 010: Slots 32 to 63 • 011: Slots 48 to 79 • 100: Slots 64 to 95 • 101: Slots 80 to 111 • 110: Slots 96 to 127 • 111: Slots 112 to 143	L/S
4 to 0	FLRX	Receive slot count setting bit • 00000: 32 slots per frame (default) • 00001 to 11111: 1 to 31 slots per frame	L/S

5.3.3 Timing of TDM serial interface

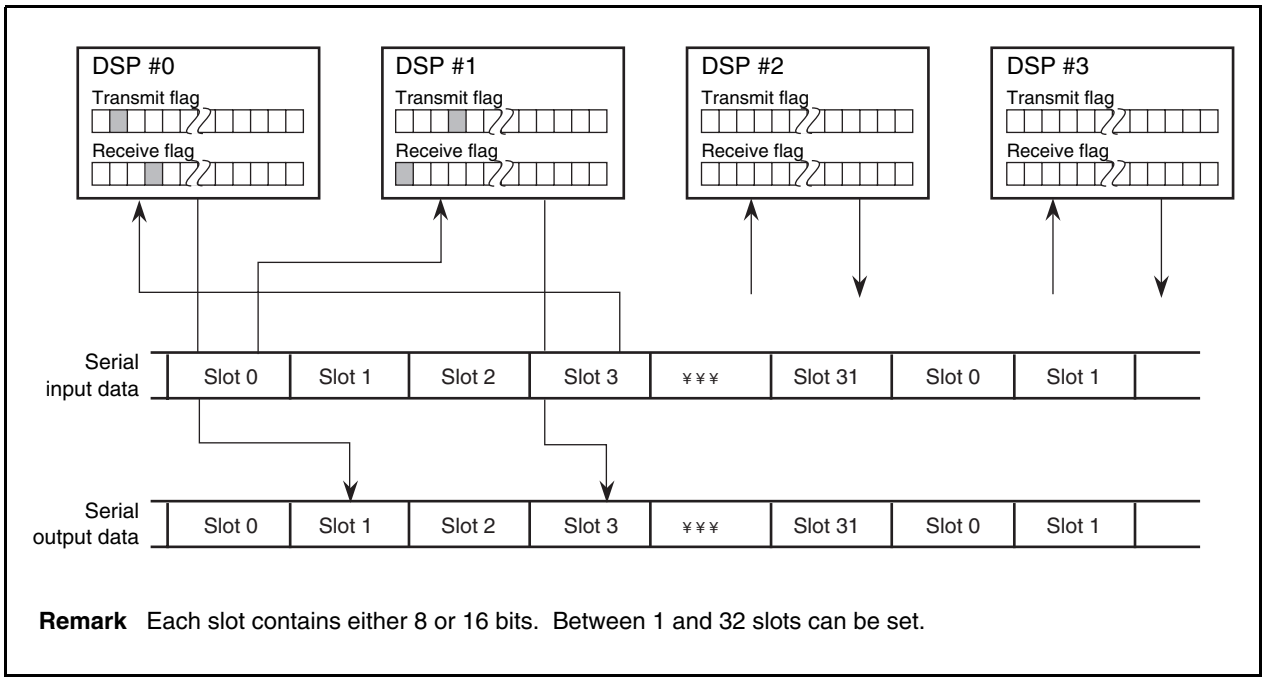
This method uses multiple DSP blocks to perform time division multiplexing for one input port and one output port.

The serial data is divided every 8 bits or every 16 bits to define slots. The number of slots can be set from 1 to 32. Under the expansion specification, up to 128 slots can be set. In such cases, setting of flags is limited to 32 slots, starting from a number of slots that is a multiple of 16.

Figure 5-3 shows an image of the slots and transfers and Figure 5-4 shows an image of the expanded number of slots.

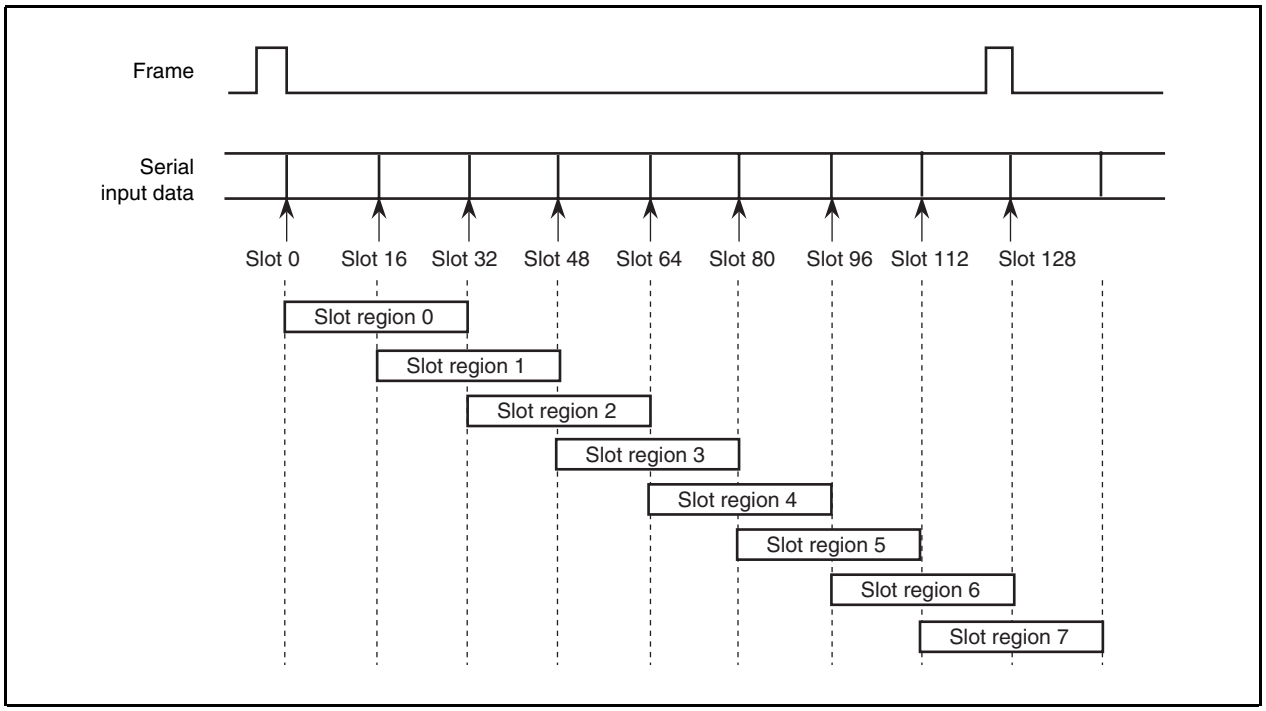
Caution Up to 128 slots (slots 0 to 127) can be managed within one frame. Therefore, when a slot definition extends into the next frame, as shown in slot region 7 in Figure 5-4, slots 128 and subsequent slots are use-prohibited.

Figure 5-3. Slots



Remark Each slot contains either 8 or 16 bits. Between 1 and 32 slots can be set.

Figure 5-4. Expanded Slots



When serial input and serial output are used separately, a TDM data input system and a TDM data output system can be defined for one channel of TSIO. However, since the clock frequency is the same for both, the data rate is the same for input and output.

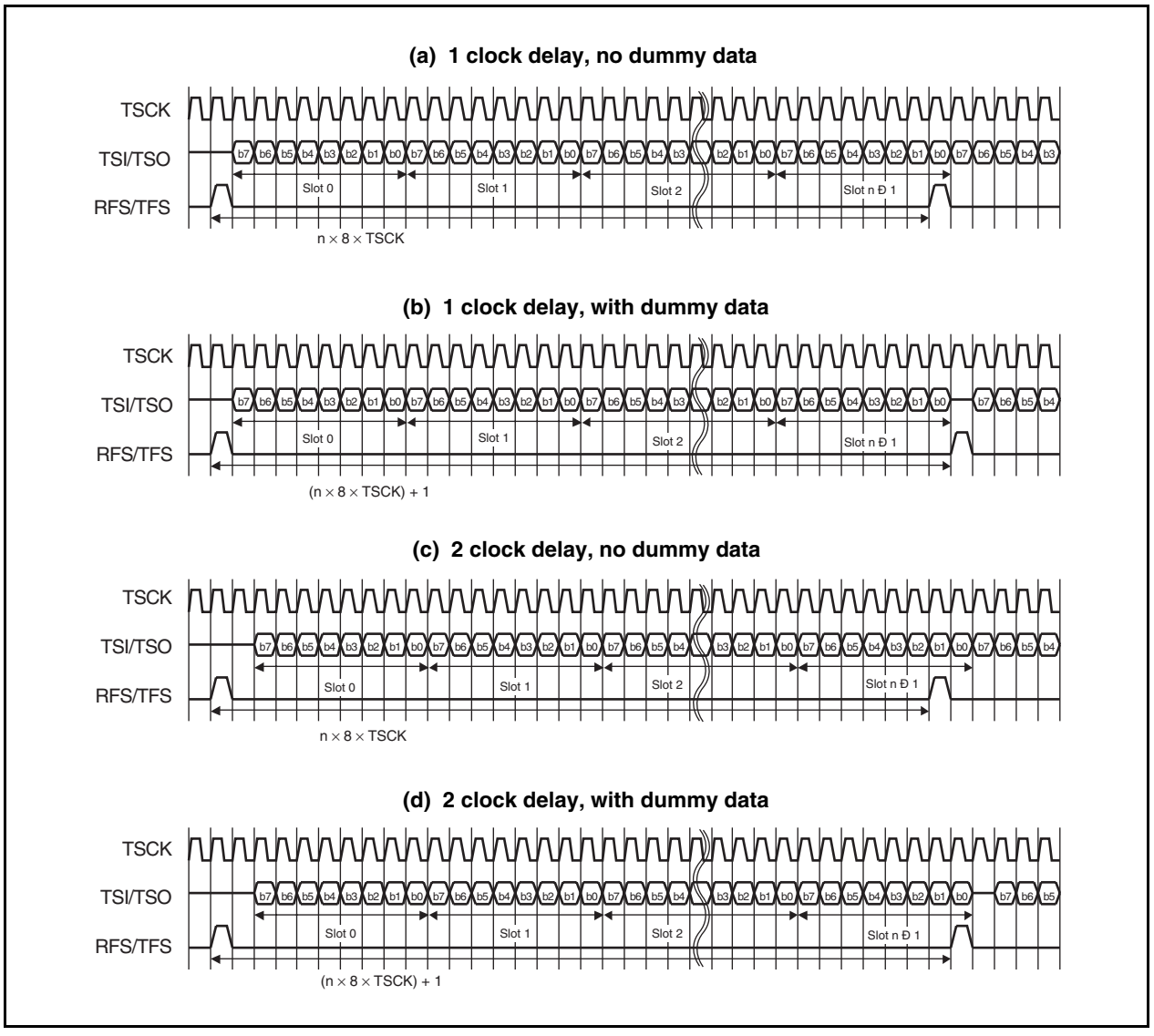
The slots from which data will be read and the slots to which data will be written can be defined by setting the respective 32-bit transmit and receive flags. In addition, multiple flags can be set. Data can be read from or transferred to the slots for which a flag has been set.

When TDM serial data is transferred, the required data can be read and input and the data can be output at the necessary timing in order to generate TDM data.

The TDM function makes it easy to work with standard T1 frame signals and E1 frame signals.

The transfer timing is shown in Figure 5-5.

Figure 5-5. Timing of TDM Serial Interface



The start of the serial signal data is switched between two types of timing: at the first data that occurs at the frame signal's next clock (for standard serial signals or E1 signals) or at the first data that occurs at the frame signal's second clock (for T1 signals).

The length of each frame is switched between (multiple of 8) bits and (multiple of 8) + 1 bits. The latter case is used to skip reading the T1's frame bit. A single-bit dummy bit is inserted for this purpose.

Thus, there are four types of timing for the combinations described above. In part (a) of Figure 5-5, data starts at the frame's next clock and there is no dummy bit. In part (b), data starts at the frame's next clock and a one-bit dummy bit is inserted. In part (c), data starts at the frame's second clock and there is no dummy bit. In part (d), data starts at the frame's second clock and a dummy bit is inserted. To handle E1 signals, the timing in part (a) should be set for 32 slots. To handle T1 signals, the timing in part (d) should be set for 24 slots.

Both RTZ pulses and their inverted RTO pulses (shown in Figure 5-5) are supported for these frame signals.

Remark E1 is (32 slots × 8 bits =) 256 bits per frame (European standard) and T1 is (24 slots × 8 bits + 1 bit =) 193 bits per frame (American and Japanese standard).

5.4 Audio Serial Interface (ASIO)

The audio serial interface (ASIO) is a peripheral circuit that includes not only a standard speech codec and a directly connectable serial interface (standard mode) but also an audio serial interface that supports input and output of 32/64-bit stereo audio signals.

The audio serial interface supports two modes: master mode, during which a serial clock is supplied to the audio codec from the μPD77210 Family device, and slave mode, during which a serial clock is supplied from the audio codec to the μPD77210 Family device.

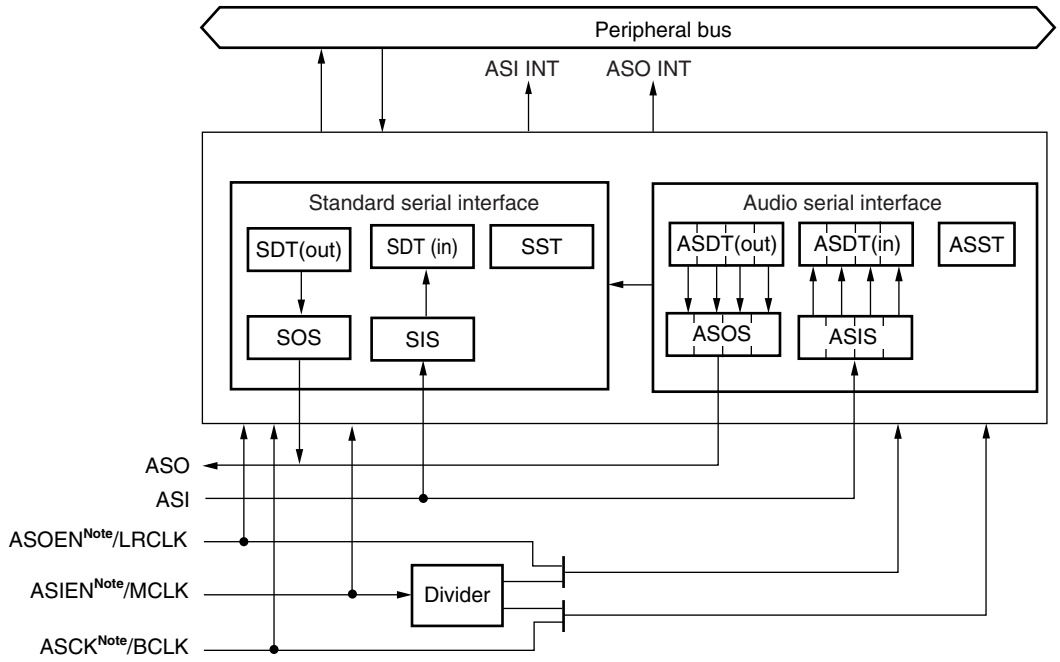
For details of the standard mode, see **5.5 Standard Serial Interface (SIO)**.

The ASIO's main features are as follows.

- Serial clock supply
  Master mode
    MCLK pin: Master clock (input)
    BCLK pin: Audio serial clock (output)
    LRCLK pin: Left/right clock (output)
  Slave mode
    MCLK pin: Not used (pulled down from external source)
    BCLK pin: Audio serial clock (input)
    LRCLK pin: Left/right clock (input)
- Bit length
  32-bit or 64-bit length can be selected independently for input and output.
- Internal handshaking
  Handshaking is enabled by means of polling, wait, or interrupt.

Figure 5-6 shows a block diagram of the audio signal interface and Table 5-7 lists the registers.

Figure 5-6. Block Diagram of Audio Serial Interface



Note This pin is used during standard serial interface mode. It is not used during audio serial interface mode.

Table 5-7. Audio Serial Interface Registers

X/Y Memory Address	Register Name	Function	Load/Store
0x3810	ASDT	Audio serial data register	L/S
0x3811	SST2	Serial status register 2	L/S
0x3812	ASST	Audio serial interface setting register	L/S

5.4.1 Audio serial interface pins

(1) BCLK (serial bit clock – I/O)

This clock pin is used for audio serial data input and output. Is it used for output during master mode and for input (default) during slave mode.

In master mode, the divided master clock input from the MCLK pin, BCLK, is output and supplied the audio codec's BCLK pin.

In slave mode, the input BCLK clock is used as the direct serial clock.

Input and output of serial data and output and sampling of various serial interface signals are performed in synchronization with the BCLK signal.

(2) LRCLK (left/right clock – I/O)

This signal pin is used to synchronize audio serial data frames. It is an output pin in master mode and an input pin in slave mode.

LRCLK is synchronized every 32 bits or 64 bits of audio serial data so that one cycle of LRCLK corresponds to 32 cycles or 64 cycles of BCLK.

In master mode, the divided master clock input from the MCLK pin, LRCLK, is output and supplied the audio codec's LRCLK pin.

In slave mode, the input LRCLK clock is used as the direct frame clock. Input and output of serial data and output and sampling of various serial interface signals are performed in synchronization with the BCLK signal.

(3) MCLK (master clock – input)

This is the input pin for the BCLK and LRCLK's master clock. It is used only during master mode and cannot be used during slave mode (pull-down processing is recommended).

The clock that is input from the MCLK pin is divided and BCLK and LRCLK are generated within the audio serial interface block.

(4) ASI (serial data input – input)

This pin is used to input audio serial data.

These signals are sampled at the rising edge of BCLK.

(5) ASO (serial data output – output)

This pin is used to output audio serial data.

Output changes at the falling edge of BCLK.

5.4.2 Audio serial interface registers

(1) ASDT (audio serial data transfer register)

ASDT is a 64-bit register that is used to input and output audio serial data. It includes separate registers for output and input.

SDT (out) is a 64-bit register that sets data to be output. When a store instruction is executed to ASDT, data is input to SDT (out) from the peripheral bus.

When the serial output shift register (SOS) becomes empty, the value of the ASDT register is set to SOS and is output via the ASO pin in MSB first order. When the ASSEF flag's value is 0 and a store instruction to ASDT is executed, the ASSER flag value is set to 1 (store error).

When a 64-bit data transfer is executed, the data is stored to ASDT via four 16-bit data write operations: first bits 63:48 are written, then bits 47:32, 31:16, and 15:0 in that order.

When a 32-bit data transfer is executed, the data is stored to ASDT via two 16-bit data write operations: first bits 31:16 are written, then bits 15:0.

The ASSEF flag is stored either four times (for 64-bit transfers) or two times (for 32-bit transfers), after which the flag's value becomes 0 (store disabled status).

SDT(in) is a 64-bit register that reads data that has been input. When a load instruction to ASDT is executed, data is output to the peripheral bus.

When the last bit is input in MSB first order from the ASI pin to the serial input shift register (SIS), the value of SIS is set to ASDT. When the ASLEF flag's value is 0 and a load instruction from ASDT is executed, the ASLER flag value is set to 1 (load error).

When a 64-bit data transfer is executed, the data is loaded from SDT via four 16-bit data read operations: first bits 63:48 are read, then bits 47:32, 31:16, and 15:0 in that order.

When a 32-bit data transfer is executed, the data is loaded from ASDT via two 16-bit data read operations: first bits 31:16 are read, then bits 15:0.

The ASLEF flag is loaded either four times (for 64-bit transfers) or two times (for 32-bit transfers), after which the flag's value becomes 0 (load disabled status).

(2) ASST (audio serial status register)

ASST is a 16-bit register that indicates the mode setting and status of the audio serial I/O. This register indicates specification of the μPD77210 Family DSP core kernel interface such as the standard/audio serial interface mode selection, the master clock division settings, and the overrun or underrun status.

The bit configuration of ASST is shown in Table 5-8. The value after reset is 0x8012.

(3) SOS (serial output shift register)

SOS is a 64-bit shift register that outputs and shifts audio serial data from the ASO pin. When the specified number of bits have been output, new data is input from ASDT.

SOS is not connected to the peripheral bus.

(4) SIS (serial input shift register)

SIS is a 64-bit shift register that receives bit strings input from the ASI pin. When the specified number of bits have been input, new data is output to ASDT.

SIS is not connected to the peripheral bus.

(5) SST1 (serial status register)

This register cannot be used when in audio serial interface mode.

Table 5-8. Bit Configuration of ASST (1/2)

Bit	Name	Function	Load/Store
15	SOAD	Standard/audio serial interface selection bit • 0: Standard serial • 1: Audio serial (default)	L/S
14	ASOEN	Audio serial output use enable/disable bit • 0: Output use disable (default) • 1: Output use enable	L/S
13	ASIEN	Audio serial input use enable/disable bit • 0: Input use disable (default) • 1: Input use enable	L/S
12 to 10	BDV[Note 1]	Time division setting bit to generate BCLK/LRCLK from master clock • 000: Divide by 4/divide by 256 (default) • 001: Divide by 8/divide by 256 • 010: Divide by 6/divide by 384 • 011: Divide by 12/divide by 384 • 100: Divide by 8/divide by 512 • 101: Divide by 16/divide by 512	L/S
9	ADRST[Note 2]	Audio serial reset enable bit • 0: (Default) • 1: Audio serial I/O reset request The I/O circuits are reset, so that ASSER = 0, ASLER = 0, ASSEF = 1, and ASLEF = 0. After ADRST is reset, it is automatically cleared to zero.	L/S
8	ASSWE	ASDT store wait enable bit • 0: Does not use store wait function (default) • 1: Uses store wait function. Inserts wait to μPD77210 Family device if a store is executed when data remains in SDT(out).	L/S
7	ASLWE	ASDT load wait enable bit • 0: Does not use load wait function (default) • 1: Uses load wait function. Inserts wait to μPD77210 Family device if a load is executed when there is no data in SDT(in).	L/S
6	ADOBL	Audio data output frame bit length setting bit • 0: 64 bits (default) • 1: 32 bits	L/S
5	ADIBL	Audio data input frame bit length setting bit • 0: 64 bits (default) • 1: 32 bits	L/S

Notes 1. Be sure to set a value when changing to master mode. New (changed) settings cannot be made while in master mode. If any such changes are attempted while in master mode, ASIO will not operate normally.

2. Once the serial reset bit has been set (= 1), ASIO cannot be accessed for three cycles. Results are not reflected normally if ASIO is accessed during these three cycles.
The serial rest bit must not be set (= 1) at the same time that another ASST bit is being set.

Table 5-8. Bit Configuration of ASST (2/2)

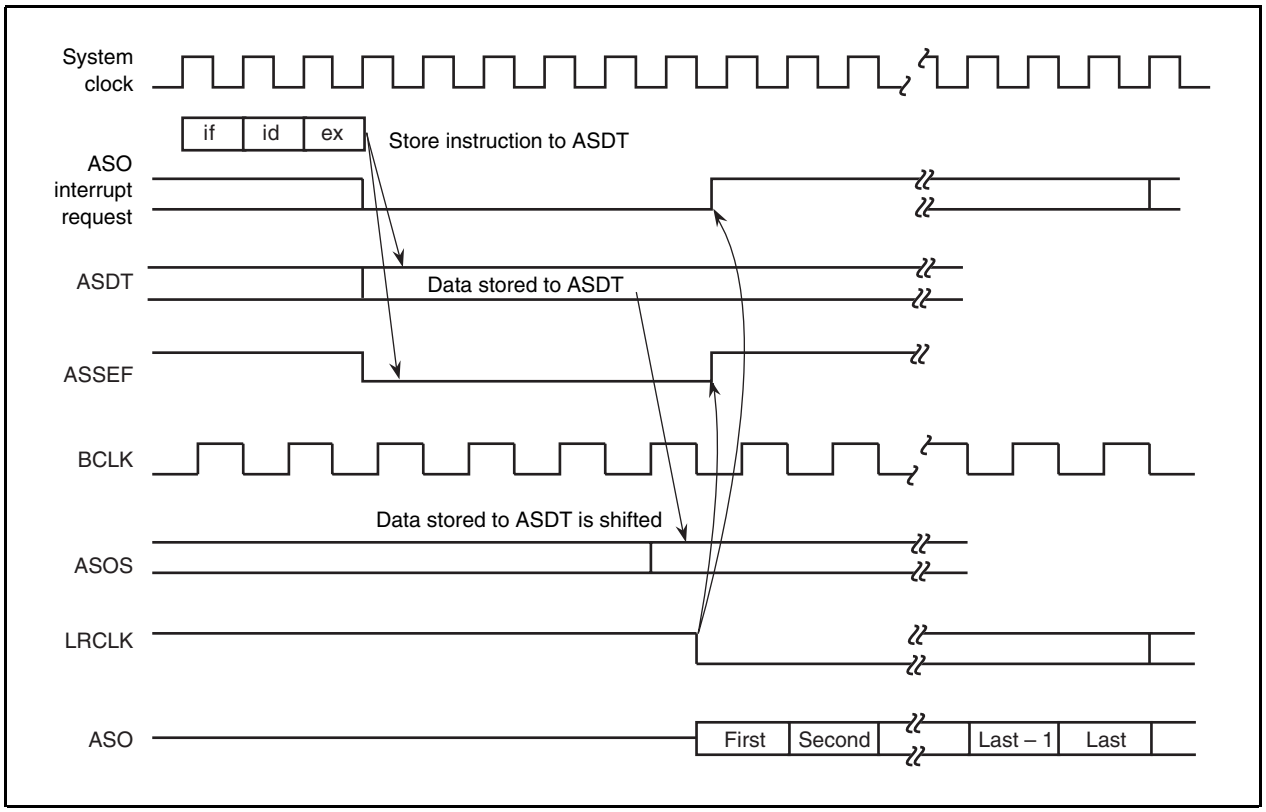
Bit	Name	Function	Load/Store
4	MSSEL[Note]	Audio serial clock mode setting bit • 0: Master mode • 1: Slave mode (default)	L/S
3	ASSER	ASDT store error flag • 0: No error (default) • 1: Error When ASSEF = 0, this bit is set (= 1) when a μPD77210 Family device writes to ASDT. When this bit is set (= 1), its status does not change until a μPD77210 Family device writes 0 to it.	L/S
2	ASLER	ASDT load error flag • 0: No error (default) • 1: Error When ASLEF = 0, this bit is set (= 1) when a μPD77210 Family device reads ASDT. When this bit is set (= 1), its status does not change until a μPD77210 Family device writes 0 to it.	L/S
1	ASSEF	ASDT store enable flag • Set (= 1) when the value of ASDT has been transferred to the SOS register (default) • Cleared to zero when a μPD77210 Family device writes to ASDT	L
0	ASLEF	ASDT load enable flag • Set (= 1) when the value of SIS has been transferred to the ASDT register • Cleared to zero when a μPD77210 Family device reads from ASDT (default)	L

Note After switching to master mode, do not switch back to slave mode, or else ASIO will not operate normally.

5.4.3 Timing of audio serial interface

Figure 5-7 shows the audio serial interface's output timing.

Figure 5-7. Output Timing of Audio Serial Interface



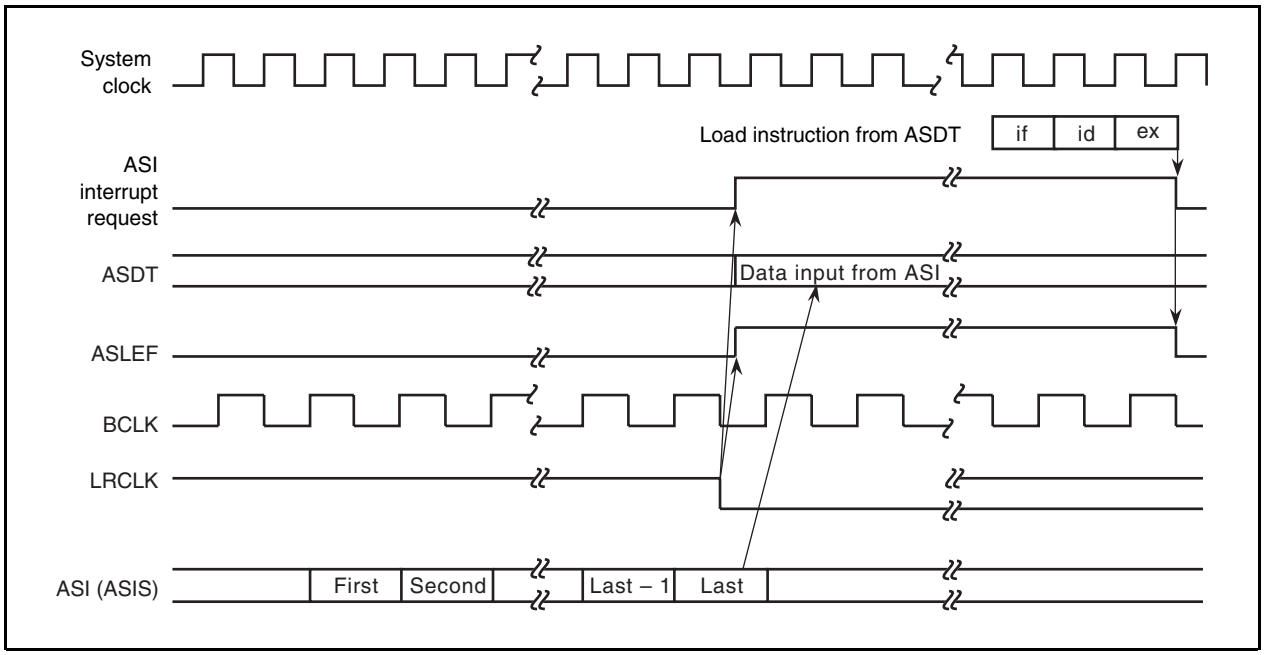
When data is stored in ASDT twice in the 32-bit mode or four times in the 64-bit mode, the interrupt and store enable flag becomes inactive.

The data stored in ASDT is transferred from ASDT to ASOS at the first rising edge of BCLK following the falling edge of LRCLK, and is then output via ASO in MSB-first order in synchronization with the falling edge of BCLK.

The store enable flag and interrupt signal become active at the rising edge of the first system clock (CLK) following the falling edge of LRCLK, then the next serial output data is requested.

Figure 5-8 shows the audio serial interface's input timing.

Figure 5-8. Input Timing of Audio Serial Interface



When data is stored in ASDT twice in the 32-bit mode or four times in the 64-bit mode, the interrupt and load enable flag becomes inactive.

The serial data input from ASI is sampled at the rising edge of BCLK and input to ASIS. After the last bit has been input, the data is transferred from ASIS to ASDT at the rising edge of the system clock. The transfer is completed at the next rising edge of the system clock (CLK), at which time the interrupt and load enable flag are set.

The next serial input is started at the first rising edge of the BCLK following the falling edge of LRCLK.

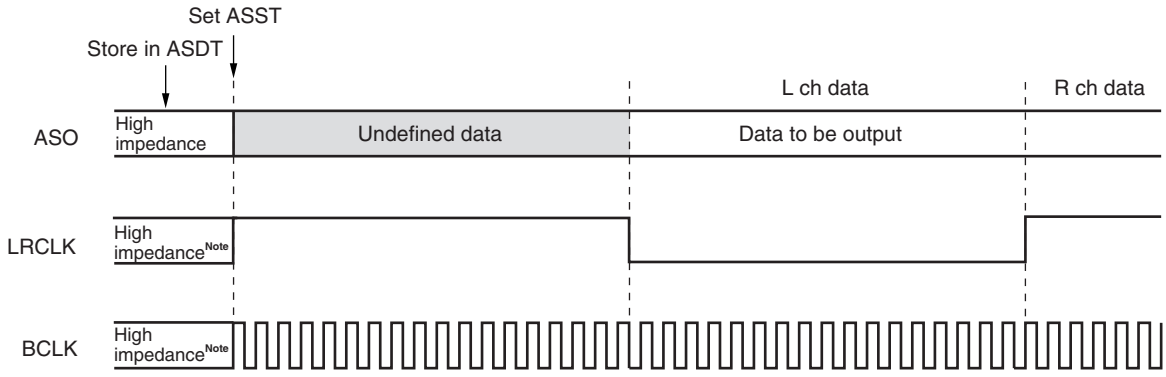
5.4.4 Precautions on ASIO during startup

When starting the ASIO operation after the μPD77210 Family has been started, the ASO pin outputs undefined data that is not anticipated. This is because the ASOS register is not initialized immediately after the power is applied and the contents of the register become undefined. Therefore, even if the data to be output is stored in the ADST register, undefined data will be output before LRCLK falls.

Undefined data is output both in the master and slave modes.

This phenomenon cannot be prevented. When using ASO, take appropriate measures, such as muting on the codec side.

Figure 5-9. ASO Operation When ASIO Operation Is Started (in Master Mode)



Note In slave mode, the LRCLK and BCLK signals are output immediately after startup.

5.5 Standard Serial Interface (SIO)

The standard serial interface (SIO) is a serial interface that is compatible with conventional μPD7701x Family and μPD77111 Family devices. This mode can be set by switching the audio serial interface setting or TDM serial interface setting.

The SIO's main features are as follows.

- Serial clock supply
  Enables external supply and common clock for input and output.
- Bit length
  Selectable as 8 or 16 bits and as MSB- or LSB-first format.
- Handshaking is enabled by means of polling, wait, or interrupt.
- External handshaking
  Handling is enabled by means of dedicated status signals (TSIAK, TSORQ).

Figure 5-10 shows a block diagram of the standard serial interface and Table 5-9 lists the registers.

Figure 5-10. Block Diagram of Standard Serial Interface

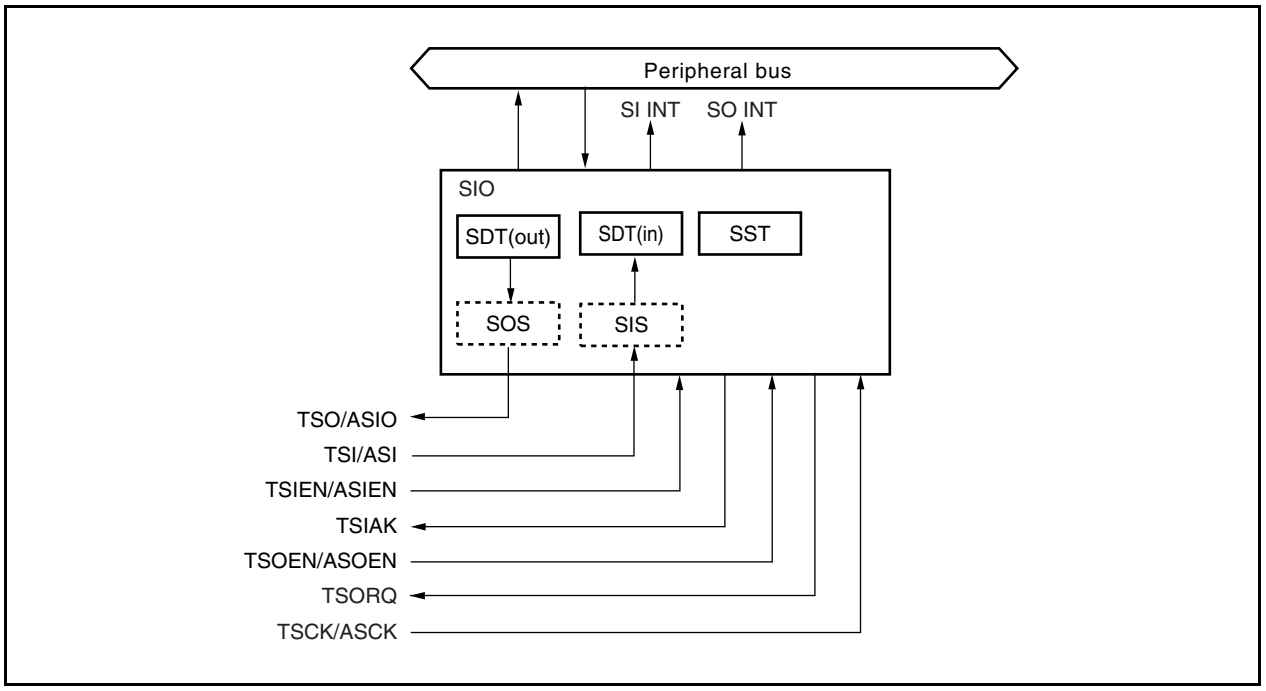


Table 5-9. Standard Serial Interface Registers

X/Y Memory Address	Register Name	Function	Load/Store
0x3800	SDT1	Serial data register 1	L/S
0x3801	SST1	Serial status register 1	L/S
0x3810	SDT2	Serial data register 2	L/S
0x3811	SST2	Serial status register 2	L/S

5.5.1 Standard serial interface pins

(1) TSCK/ASCK (serial clock – input)

This clock pin is used for standard serial data input and output.

Input and output of serial data and output and sampling of various serial interface signals are performed in synchronization with the TSCK signal.

(2) TSOEN/ASOEN (serial output enable – input)

This pin is used to input serial data output enable signals.

These signals are sampled at the falling edge of TSCK.

These signals are asserted active (high level) when an external device is ready to input serial output data.

(3) TSIEN/ASIEN (standard serial input enable – input)

This pin is used to input serial data input enable signals.

These signals are sampled at the falling edge of TSCK.

These signals are asserted active (high level) when an external device is ready to output serial input data.

(4) TSORQ (standard serial output request – output)

This pin is used to output serial data output request signals.

Output changes at the rising edge of TSCK.

This signal is asserted active (high level) when serial data is written to SOS. It is deasserted inactive (low level) when serial output begins.

This pin does not exist when the audio serial interface is used as the standard serial interface.

(5) TSIAK (standard serial input acknowledge – output)

This pin is used to output the serial data input reception signal.

Output changes at the rising edge of TSCK.

This signal is asserted active (high level) when serial data is input-enabled. It is deasserted inactive (low level) when serial input begins.

This pin does not exist when the audio serial interface is used as the standard serial interface.

(6) TSI/ASI (standard serial data input – input)

This pin is used to input serial data.

These signals are sampled at the falling edge of TSCK/ASCK.

(7) TSO/ASO (standard serial data output – output)

This pin is used to output serial data.

These signals are sampled at the rising edge of TCLK/ASCK.

5.5.2 Standard serial interface registers

(1) SDT (serial data transfer register)

SDT is a 16-bit register that is used for input and output of standard serial data. It includes separate registers for output and input.

SDT(out) is a 16-bit register that sets data to be output. When a store instruction is executed to SDT, data is input to SDT(out) from the peripheral bus.

When the serial output shift register (SOS) becomes empty, the value of the SDT register is set to SOS and is output via the TSO/ASO pin. When the SSEF flag is 0 and a store instruction to SDT is executed, the SSER flag is set to 1 (store error).

SDT(in) is a 16-bit register that reads data that has been input. When a load instruction to SDT is executed, data is output to the peripheral bus.

When the last bit is input to the serial input shift register (SIS) from the TSI/ASI pin, the value of SIS is set to SDT. When the SLEF flag is 0 and a load instruction from SDT is executed, the SLER flag is set to 1 (load error).

(2) SST1 (serial status register)

SST is a 16-bit register that indicates the mode setting and status of the serial I/O. This register indicates the MSB/LSB first setting as well as specification of the μPD77210 Family DSP core kernel interface and overrun or underrun status.

The bit configuration of SST is shown in Table 5-10. The value after reset is 0x0002.

(3) SOS (serial output shift register)

SOS is a 16-bit shift register that outputs and shifts serial data from the TSO/ASO pin. When the specified number of bits have been output, new data is input from SDT.

SOS is not connected to the peripheral bus.

(4) SIS (serial input shift register)

SIS is a 16-bit shift register that inputs bit strings input from the TSI/ASI pin. When the specified number of bits have been input, data is output to SDT.

SIS is not connected to the peripheral bus.

Table 5-10. Bit Configuration of SST (1/2)

Bit	Name	Function	Load/Store
15	SOTF	Serial output transfer format setting bit • 0: Serial output with MSB first (default) • 1: Serial output with LSB first	L/S
14	SITF	Serial input transfer format setting bit • 0: Serial input with MSB first (default) • 1: Serial input with LSB first	L/S
13	SOBL	Serial output word length setting bit • 0: 16-bit serial output (default) • 1: 8-bit serial output	L/S
12	SIBL	Serial input word length setting bit • 0: 16-bit serial input (default) • 1: 8-bit serial input	L/S
11	SSWE	SDT store wait enable bit • 0: Does not use store wait function (default) • 1: Uses store wait function. Inserts wait when SSEF = 0 and the μPD77210 Family device stores data to SDT.	L/S
10	SLWE	SDT load wait enable bit • 0: Does not use load wait function (default) • 1: Uses load wait function. Inserts wait when SLEF = 0 and the μPD77210 Family device loads data from SDT.	L/S
9	SICM	Serial input continuous mode setting flag • 0: Enters single serial input mode after completion of current serial input. • 1: Enters continuous serial input mode to start serial input.	L/S
8	SIEF	Single serial input enable flag • 0: (default) • 1: Starts serial input processing in single serial input mode (only once). SIEF that has been set to 1 is automatically reset in the next instruction cycle.	L/S
7[Note]	SOE	Serial output enable bit • 0: Use of serial output disable • 1: Use of serial output enable (default)	L/S
6[Note]	SIE	Serial input enable bit • 0: Use of serial input disable • 1: Use of serial input enable (default)	L/S

Note Bits 6 and 7 function only as SIO (SST1) bits on the TSIO side. They are reserved bits for SIO (SST2) on the ASIO side.

Table 5-10. Bit Configuration of SST (2/2)

Bit	Name	Function	Load/Store
5	SORST[Note]	Serial output reset enable bit This resets the serial output circuit. Afterward, SSER = 0 and SSEF = 1. This bit is automatically cleared to zero after serial output is reset. • 0: (default) • 1: SO reset request	L/S
4	SIRST[Note]	Serial input reset enable bit This resets the serial input circuit. Afterward, SLER = 0 and SLEF = 0. This bit is automatically cleared to zero after serial input is reset. • 0: (default) • 1: SI reset request	L/S
3	SSER	SDT store error flag • 0: No error (default) • 1: Error When SSEF = 0, this bit is set (= 1) when a μPD77210 Family device writes to SDT. When this bit is set (= 1), its status does not change until a μPD77210 Family device writes a 0 to it.	L/S
2	SLER	SDT load error flag • 0: No error (default) • 1: Error When SLEF = 0, this bit is set (= 1) when a μPD77210 Family device reads SDT. When this bit is set (= 1), its status does not change until a μPD77210 Family device writes a 0 to it.	L/S
1	SSEF	SDT store enable flag This bit is set (= 1) when the value of SDT has been transferred to the serial output shift register. It is cleared to zero when a μPD77210 Family device stores data to SDT.	L
0	SLEF	SDT load enable flag This bit is set (= 1) when the value of SDT has been transferred to the serial input shift register. It is cleared to zero when a μPD77210 Family device loads data from SDT.	L

Note Once the serial reset bit has been set (= 1), SIO cannot be accessed for three cycles. Results are not reflected normally if SIO is accessed during these three cycles.
The serial rest bit cannot be set (= 1) at the same time that another SST bit is being set.

Table 5-11. Combination of SICM Bits and SIEF Bits

SICM	SIEF	Function
0	0	Status transition mode. This mode is also set when serial input is not performed.
1	0	Continuous serial input mode
0	1	Single serial input mode. When SIEF bit is set (= 1), it is automatically reset (to 0) during the next instruction cycle.
1	1	Setting prohibited

5.5.3 Timing of serial interface

(1) Serial output timing

Generally, serial output is performed in the following steps. Operations in steps **<1>** through **<6>** without SDT store wait cycles are illustrated in Figure 3-49 (a) and (b) for continuous and non-continuous data, respectively.

<1> The application program executes a store to SDT (serial data register).

<2> Consequently, the SDT store enable flag (SSEF) of the serial status register (SST) is cleared to 0, notifying the application program that no more data must be written to SDT. If the SDT store wait enable bit (SSWE) is set, the SDT store wait function is validated, automatically blocking a write access to SDT.

<3> If the serial output shift register (SOS) is empty, the data set to SDT is transferred to SOS after 3 SCK cycles. The serial output request pin (TSORQ) becomes active (high), informing an external device of issuance of a serial output request.

<4> When the external device makes the serial output enable pin (TSOEN/ASOEN) active (high level) (a), this pin is sampled at the falling edge of the serial clock pin (TSCK/ASCK) immediately after[Note 1] (b), at the next rising edge of TSCK/ASCK, TSORQ becomes low (c) and data output to the serial data output pin (TSO/ASO) is started (d).

<5> After SDT has become empty, SSEF is set to 1, notifying the application program that the next data can be written (a), the SDT store wait function, which has been validated with SSWE = 1, is invalidated. At this time, an interrupt request is generated by TSO/ASO (b). However, the interrupt is serviced as a valid interrupt or is recorded, depending on the status of the corresponding interrupt enable flag and EI status (refer to **4.4.4 Interrupts**).

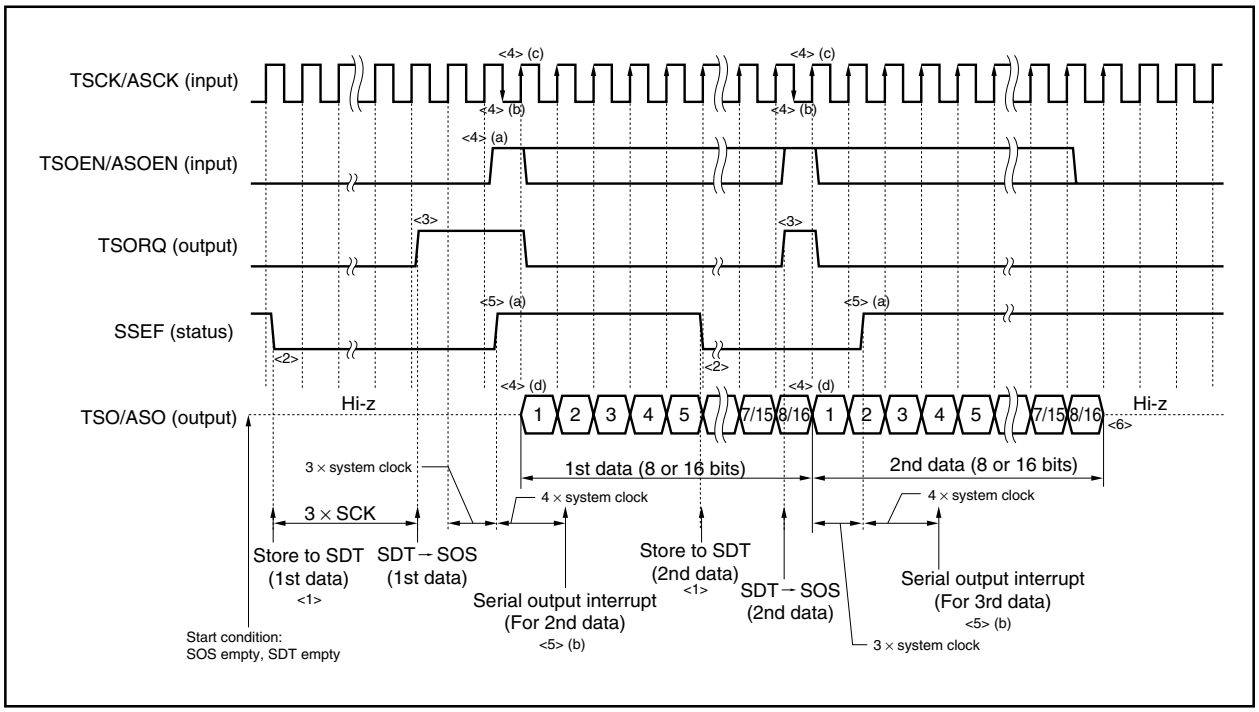
<6> If the next data is not supplied when the output of the last bit data has been completed, TSO/ASO goes into a high-impedance state at the next rising edge of the TSCK/ASCK[Note 2].

Notes 1. Before TSOEN/ASOEN becomes active, TSCK/ASCK must rise at least three times.
Bear this in mind especially in a system configuration where the clock is used in burst mode for only inputting/outputting data.

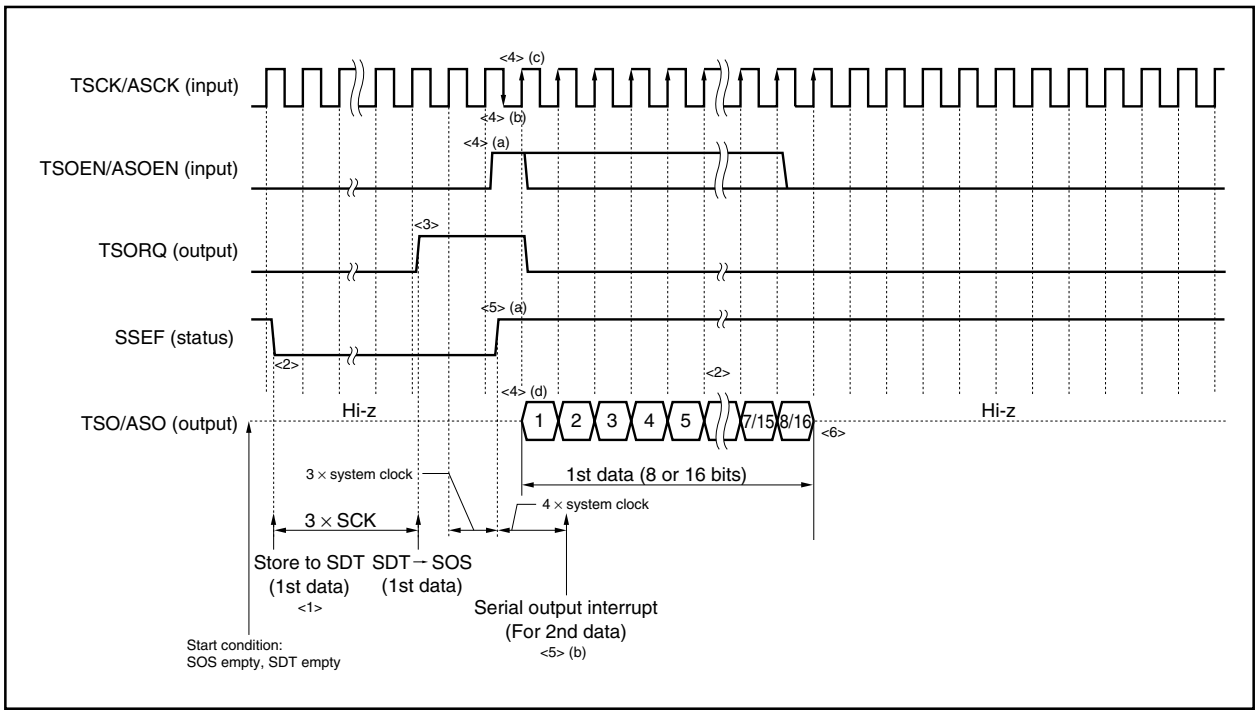
2. Under the following conditions, TSO/ASO does not go into a high-impedance state but successively outputs the next data:
if the next data has been already supplied before the last bit is output, and if TSOEN/ASOEN becomes active before falling of TSCK/ASCK in the last bit output cycle and is sampled as valid (refer to **Figure 5-11 Serial Interface Output Timing**).
After the last bit has been output, the rising edge of TSCK/ASCK must be supplied at least once.

Figure 5-11. Serial Interface Output Timing

(a) Continuous data



(b) Non-continuous data



(2) Timing of serial input

Generally, serial input is performed in the following steps. Operations in steps **<1>** through **<4>** without SDT load wait cycles are illustrated in Figure 5-12.

<1> Serial data input sequence is started when an external device makes the serial input enable pin (TSIEN/ASIEN) active (high level) with the serial input enable (TSIAK) pin being active (high level).

<2> Changes in TSIEN/ASIEN in <1> are sampled at the falling edge of TSCK/ASCK immediately after[Note 1] (a), TSIAK goes low at the next rising edge of TSCK/ASCK (b), and inputting data given to the serial data input pin (TSI/ASI) is started from the falling edge of the same TSCK/ASCK cycle (c). The data is loaded from the TSI/ASI pin to the serial input shift register (SIS) bit by bit in synchronization with the falling edge of TSCK/ASCK.

<3> TSIAK becomes active (a) in synchronization with the rising edge of the TSCK/ASCK cycle, in which the last bit of the specified number of bits is loaded, immediately before the loading, informing the external device that the next data can be input. When the last bit has been loaded[Note 2] (b) and if the SDT load enable flag (SLEF) is 0, the loaded bit is immediately transferred from SIS to SDT[Note 3].

After that, SLEF changes to 1, informing the application program that the serial input data word has been completed (c). If the data wait status is set with the SDT load wait enable bit (SLWE) set to 1, the wait function is released. Although an interrupt request is generated by TSI/ASI (d) at this time, the interrupt is serviced as a valid interrupt or is recorded, depending on the statuses of the corresponding interrupt enable flags and the EI bit (refer to **4.4.4 Interrupts**).

<4> When the application program executes a load from SDT (a), SLEF is cleared to 0, indicating that the input data is empty (b). If SLWE is 1 at this time, the SDT load wait function is validated, automatically blocking further read access to SDT.

Notes 1. Before TSIEN/ASIEN becomes active, TSCK/ASCK must rise at least three times. The hardware of the serial input/output block performs a pipeline operation with TSCK/ASCK used as a timing clock.

Bear this in mind especially in a system configuration where the clock is used in burst mode for only inputting/outputting data.

With a system configuration where the clock is successively supplied, exercise care in respect to the first data after reset.

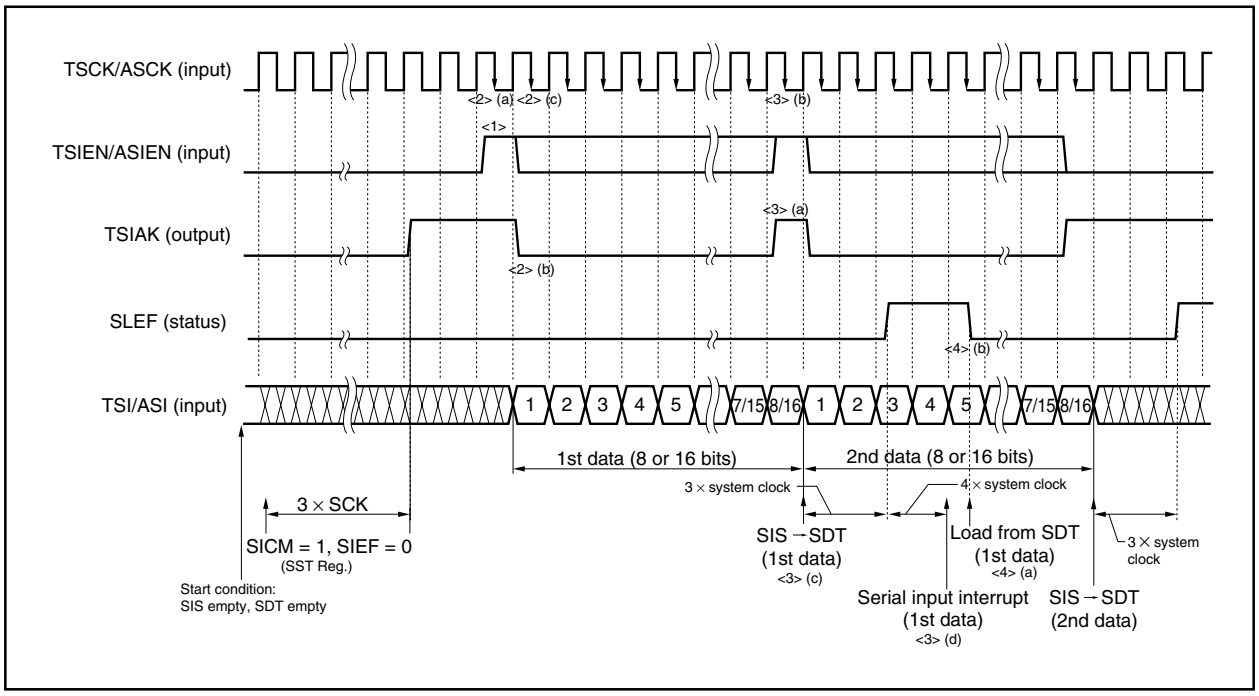
After the last bit has been output, the rising edge of TSCK/ASCK must be supplied at least two times.

2. If TSIEN/ASIEN becomes active and sampled as valid before TSCK/ASCK falls in the last bit input cycle, the next data is loaded from the successive next TSCK/ASCK cycle (refer to **Figure 5-12 Serial Interface Input Timing**).

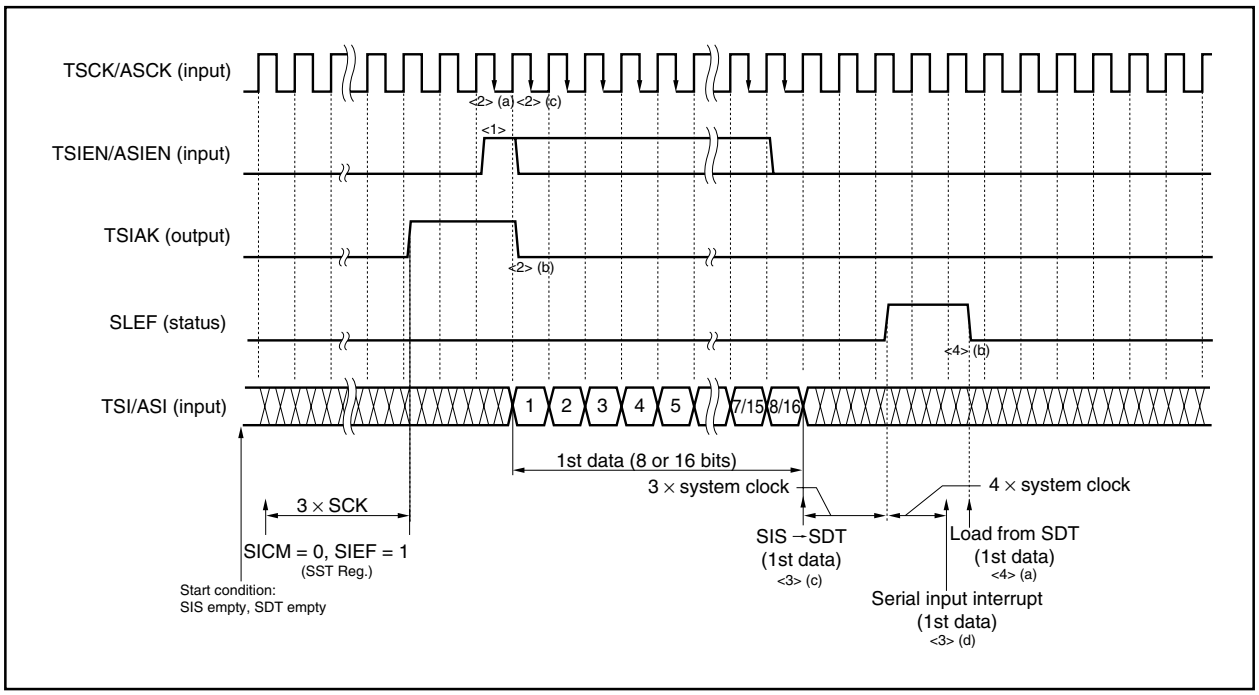
3. SDTs are used separately for serial input and serial output.

Figure 5-12. Serial Interface Input Timing

(a) SICM = 1, SIEF = 0; Continuous mode



(b) SICM = 1, SIEF = 0; Single mode

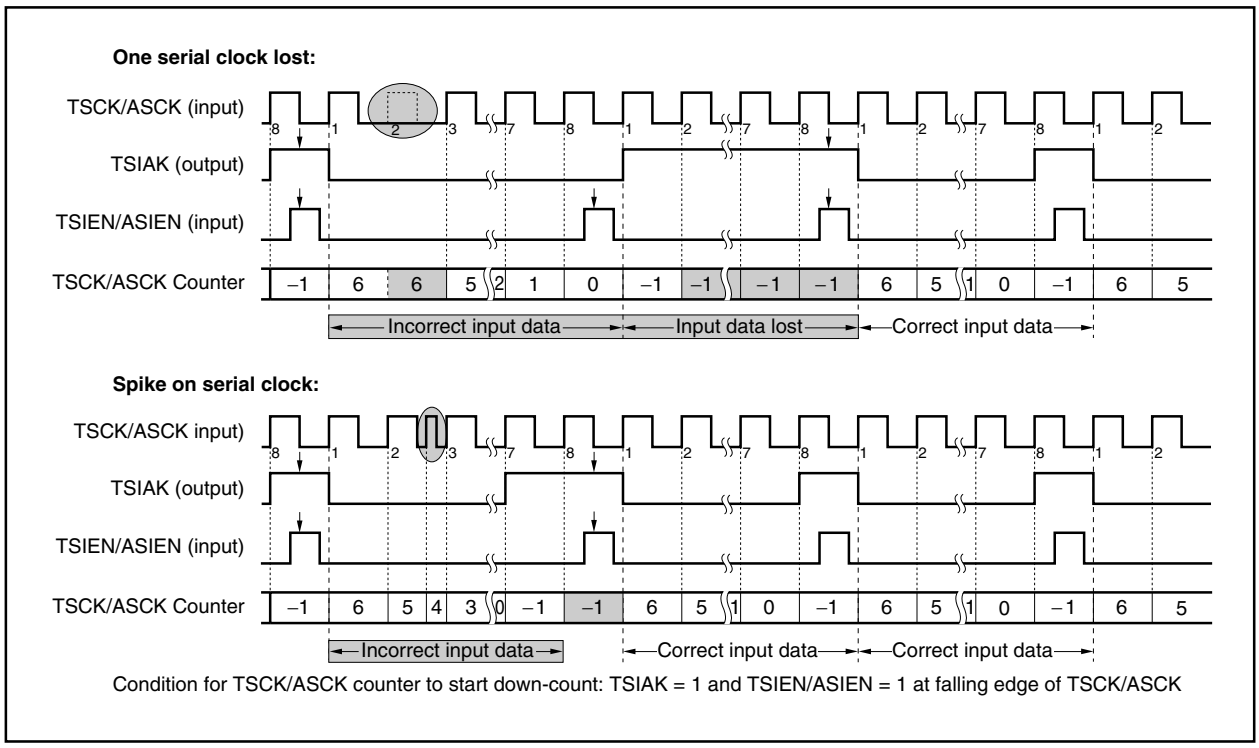


(3) I/O timing of non-standard serial clock

Figure 5-13 shows the operation of the serial clock counter which are caused by non-standard serial clock.

Data can be input/output even when TSIEN/ASIEN and TSOEN/ASOEN are active. If a bit shift occurs, however, the I/O timing cannot be corrected because a non-standard serial clock is input. By deasserting TSIEN/ASIEN and TSOEN/ASOEN inactive, this bit shift can be corrected as shown above. In the above example, TSIEN/ASIEN is input by counting TSCK/ASCK. However, it is more accurate if TSIEN/ASIEN is input depending on the status of TSIAK.

Figure 5-13. Serial Interfaces - Operation of the Serial Clock Counter



(4) Handshake

There are three means to handshake with the serial interface of the μPD77210 Family, which can be implemented by application programs:

- Polling
- Wait
- Interrupt

(a) Polling

Synchronization of handshaking is established by always monitoring and evaluating the SDT store enable flag (SSEF) and SDT load enable flag (SLEF) of the serial status register (SST). Here is an example of serial output by means of polling:

```
/* Explicitly define SST1 and SO1 because they are not reserved words. */
#define SST1      0x3801
#define SO1       0x3800

/* Disable internal interrupts SO1 and SI1. */
R0L = SR                    ;
R0 = R1 | 0x0030            ;
SR = R0L                    ;

R0L = 0x0                   ; Set serial status as follows:
*SST1:X = R0L               ; • MSB first output
                            ; • MSB first input
                            ; • 16-bit word output
                            ; • 16-bit word input
                            ; • SDT store wait function is not used.
                            ; • SDT load wait function is not used.
                            ; • Serial input is not performed.
                            ; • Clear serial input/output error flag.

POLL: R0L = *SST1:X         ; Judge SSEF and loop to wait until store is enabled.
        R0 = R0 & 0x2       ;
        if (R0 == 0)jmp POLL ;

*SO1:X = R1H                ; Data of R1H is output because store is enabled.
```

(b) Wait

Under the following conditions, execution of data exchanges with the SDT (in) and/or SDT (out) registers cause instruction wait cycles:

- When the store wait function is enabled (SSWE = 1) and a store to SDT (out) for serial output is to be executed, while SSEF = 0 (valid data exists in SDT (out)).
- When the load wait function is enabled (SLWE = 1) and a load from SDT (in) for serial input is to be executed, while SLEF = 0 (valid data does not exist in SDT (in)).

The advantage of this format is that describing handshake procedures in the application program is not needed, because the handshake procedure is automatically executed by hardware. Here is an example of serial output by using the wait function:

```
/* Explicitly define SST1 and SO1 because they are not reserved words. */
#define SST1      0x3801
#define SO1       0x3800

/* Disable internal interrupts SO1 and SI1. */
R0L = SR                    ;
R0 = R0 | 0x0030            ;
SR = R0L                    ;

R0L = 0x800                 ; Set serial interface as follows:
*SST1:X = R0L               ; • MSB first output
                            ; • MSB first input
                            ; • 16-bit word output
                            ; • 16-bit word input
                            ; • SDT store wait function is used.
                            ; • SDT load wait function is not used.
                            ; • Serial input is not performed.
                            ; • Clear serial input/output error flag.

*SO1:X = R1H                ; Data of R1H is output as soon as SSEF = 1.
```

Caution When data is written from the application program to SDT, the wait is not released unless SDT is transferred to SOS (i.e., unless all the bits of the previous data of SOS are shifted out to the external device). If internal writing of DSP and external reading do not correspond on a one-to-one basis vis-a-vis SDT, a hang-up may occur.
During wait, interrupts are delayed (refer to 4.4.4 Interrupts).

(c) Interrupt

Handshaking is established by interrupts, if data can be stored to SDT(out) and data can be loaded from SDT(in). Therefore, the advantage of this format is that, even while other processing is under execution, serial I/O can be executed independently (asynchronously) of the processing. Here is an example of serial I/O using an interrupt:

```
/* definition of serial I/O register names          */
#define SST1      *0x3801:X
#define SI1       *0x3800:X
#define SO1       *0x3800:X

/* interrupt vector table entries                   */
SegSI1  IMSEG AT 0x220      ; SIO#1 input interrupt routine

        R0H = SI1           : load from SDT(in)
        R0 = R0H*R1H        ;
        *DP0++ = R0H        ; save to buffer
        RETI                ; return from interrupt

SegSO1 IMSEG AT 0x224       ; SIO#1 output interrupt routine

        R0H = *DP4++        ; read from buffer
        SO1 = R0H           ; save to SDT(out)
        RETI                ; return from interrupt
        NOP                 ;

/* disable interrupts to initialize serial input/output */
        R1L = EIR           ; disable interrupts generally
        R1 = R1 | 0x8000    ; EI = 1
        EIR = R1L           ;
        NOP                 ; wait 2 cycles until EI = 1 effective
        NOP                 ;

        R0L = SR            ; enable SI1 and SO1 interrupts
        R0 = R0 & 0xFFCF    ;
        SR = R0L            ;

        R1 = R1 & 0x7FFF    ; enable interrupts generally
        EIR = R1L           ;
        FINT                ; discard all pervious interrupts

                            ; initialize SST1
        R0L = 0x0200        ; input/output: MSB-first, 16-bit
                            ; no load/store wait function
        SST1 = R0L          ; serial input continuous mode
        SO1 = R0L           ; dummy store (see below)
```

Caution Note the following points when executing serial output interrupt because the interrupt occurs after data has been transferred from the SDT register to the serial output shift register:

(1) Transfer first dummy data and then forcibly generate an interrupt, or do not use interrupts during the transfer of the first data, depending on the interrupt mode.

(2) When transferring data in burst mode, first disable interrupts immediately before the instruction which transfers the last word to SDT, then execute the instruction introduced in (1) after the completion of the next burst data preparation, to transfer the next burst data. This is because the first word of data to be burst-transferred may not be completely prepared if an interrupt is generated during the transfer of the last burst data word.

Example:

```
                    ; /*When last word is stored to SDT. */
R0L = SR            ; /*(DI status during interrupt processing) */
R0 = R0 | 0x0020    ; /*SO1 interrupt is disabled. */
SR = R0L            ;
*SO1: X = R0H       ;
```

5.5.4 Precautions on SIO during startup

When starting the SIO operation after the μPD77210 Family has been started, the TSO pin outputs undefined data that is not anticipated. This is the same phenomenon that occurs when using the audio serial interface. For details, refer to **5.4.4 Precautions on ASIO during startup**.

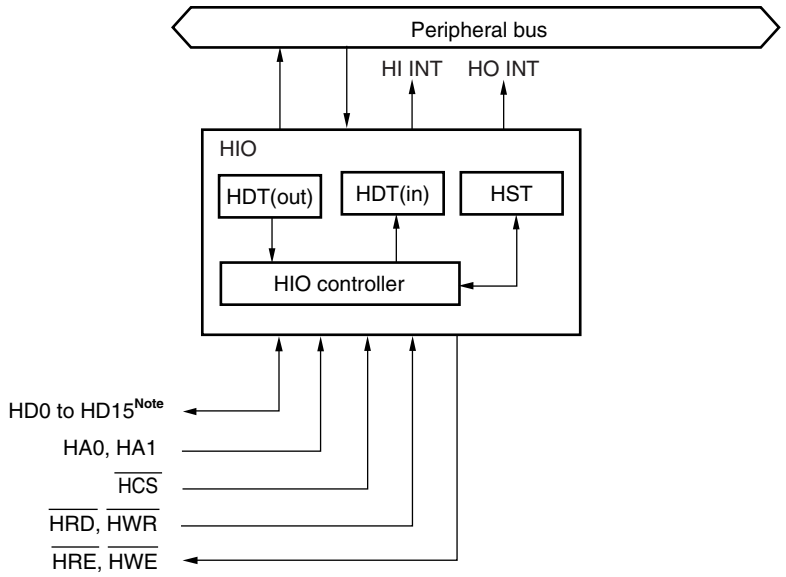
5.6 Host Interface (HIO)

The host interface (HIO) is an interface circuit that is used to transfer data to and from an external host CPU. The HIO's main features are as follows.

- Bus width
  Byte (8-bit) mode: Addresses for either the higher 8 bits or the lower 8 bits of the host interface register (HDT) are selected for external access via an 8-bit external bus.
  Word (16-bit) mode: The host interface register (HDT) is directly accessed (16-bit access) by the host CPU via a 16-bit external bus.
- Internal handshaking
  Handshaking is enabled by means of polling, wait, or interrupt.
- External handshaking
  Handling is enabled by means of dedicated status signals ($\overline{\text{HRE}}$, $\overline{\text{HWE}}$).

Figure 5-14 shows a block diagram of the host interface and Table 5-12 lists the registers.

Figure 5-14. Block Diagram of Host Interface



Note HD0 to HD15 when in 8-bit host mode.

Table 5-12. Host Interface Registers

X/Y Memory Address	Register Name	Function	Load/Store
0x3820	HDT	Host interface data register	L/S
0x3821	HST	Host interface status register	L/S

5.6.1 Host interface pins

(1) $\overline{\text{HCS}}$ (host interface select – I/O)

This pin is used for input and output of host interface select signals.

This pins is asserted active (low level) while the host is accessing the host interface registers.

(2) HA0 and HA1 (host address – input)

These are host interface address input pins.

They are used to specify the host interface registers to be accessed.

The status of these pins should not be changed while the host is accessing the host interface registers.

(3) $\overline{\text{HRD}}$ (host read strobe – input)

This is the host interface's read strobe signal input pin.

This signal is asserted active (low level) when the host reads a host interface register. It cannot be accessed at the same time as the $\overline{\text{HWR}}$ pin.

(4) $\overline{\text{HWR}}$ (host write strobe – input)

This is the host interface's write strobe signal input pin.

This signal is asserted active (low level) when the host writes to a host interface register. It cannot be accessed at the same time as the $\overline{\text{HRD}}$ pin.

(5) HD0 to HD15 (host data – I/O)

These are the host interface's data I/O pins. In 8-bit bus mode, they are used as the I/O pins for HD0 to HD8. Since HD8 to HD15 are used as both general-purpose I/O port pins and alternate-function pins in 16-bit bus mode, pins being used for alternate functions cannot be used as general-purpose I/O pins.

Data input or output occurs when the host interface's register is accessed.

A high-impedance status occurs when $\overline{\text{HCS}}$ is inactive (high level).

(6) $\overline{\text{HRE}}$ (host read enable – output)

This pin is used to output the signal indicating that HDT is read-enabled.

When HDT is read-enabled, this pin is asserted active (low level) and when HDT is read (the higher byte is read during 8-bit mode), it is deasserted inactive (high level) at the falling edge of the $\overline{\text{HRD}}$ pin.

A hardware reset also deasserts this pin as inactive.

(7) $\overline{\text{HWE}}$ (host write enable – output)

This pin is used to output the signal indicating that HDT is write-enabled.

When HDT is write-enabled, this pin is asserted active (low level) and when HDT written to (the higher byte is written to during 8-bit mode), it is deasserted inactive (high level) at the falling edge of the $\overline{\text{HWR}}$ pin.

A hardware reset also deasserts this pin as inactive.

5.6.2 Host interface registers

(1) HDT (host data transfer register)

This 16-bit register is used to input or output host data from the host interface. It includes separate registers for input and output.

HDT (out) is a 16-bit register that sets data to be output. When a store instruction is executed to HDT, data is input to this register from the peripheral bus.

SDT (in) is a 16-bit register that reads data that has been input. When a load instruction to HDT is executed, data is output to the peripheral bus.

(2) HST (host status register)

HST is a 16-bit register that indicates the mode setting and status of the host interface. This register indicates the 8-bit/16-bit bus mode selection as well as specification of the μPD77210 Family DSP core kernel interface and write or read errors.

The bit configuration of HST is shown in Table 5-13. The value of this register is 0x301 when reset.

Table 5-13. Bit Configuration of HST (1/2)

Bit	Name	Function	Load/Store (from DSP)	Read/Write (from host)
15 to 12	Reserved	Values other than 0 cannot be written. • Undefined during a read operation	–	–
11	HBM	Host interface bus width setting bit • 0: 8-bit mode (default) • 1: 16-bit mode	L/S	R
10	HAWE	HDT access wait enable bit • 0: Does not use wait function (default) • 1: Uses wait function. Inserts wait to the μPD77210 Family device when HREF = 1 and data is stored in HDT(out). Inserts wait to the μPD77210 Family device when HWEF = 1 and data is loaded from HDT(in).	L/S	R
9	HREM	HRE mask bit • 0: Does not mask • 1: Masks (default)	L/S	R
8	HWEM	HWE mask bit • 0: Does not mask • 1: Masks (default)	L/S	R
7	UF1	User flag	L/S	R
6	UF0			

Table 5-13. Bit Configuration of HST (2/2)

Bit	Name	Function	Load/Store (from DSP)	Read/Write (from host)
5	HRER	Host read error flag • 0: No error (default) • 1: Error This bit is set (= 1) when HREF = 0 and the CPU has read HDT. Once set, this bit does not change until a μPD77210 Family device writes a 0 to it.	L/S	R
4	HWER	Host write error flag • 0: No error (default) • 1: Error When HWEF = 0 , this bit is set (= 1) when the CPU writes to HDT. When this bit is set (= 1), its status does not change until a μPD77210 Family device writes a 0 to it.	L/S	R
3	HSER	HDT store error flag • 0: No error (default) • 1: Error When HREF = 1, this bit is set (= 1) when a μPD77210 Family device writes to HDT. When this bit is set (= 1), its status does not change until a μPD77210 Family device writes a 0 to it.	L/S	R
2	HLER	HDT load error flag • 0: No error (default) • 1: Error When HWEF = 0, this bit is set (= 1) when a μPD77210 Family device reads HDT. When this bit is set (= 1), its status does not change until a μPD77210 Family device writes a 0 to it.	L/S	R
1	HREF	Host read enable flag • 0: Read prohibited • 1: Read enabled This bit is set (= 1) when a μPD77210 Family device has written to HDT. It is cleared (= 0) when the host CPU has read HDT.	L	R
0	HWEF	Host write enable flag • 0: Write prohibited • 1: Write enabled This bit is set (= 1) when a μPD77210 Family device has read HDT. It is cleared (= 0) when the host CPU has written to HDT.	L	R

5.6.3 Host interface registers from perspective of host

The host CPU uses the HA0 and HA1 pins in the μPD77210 Family device to specify which registers will be accessed. Table 5-14 shows the host interface registers used during access from an external device.

Table 5-14. Selection of Registers for Host Interface

$\overline{HCS}$	$\overline{HRD}$	$\overline{HWR}$	HA1	HA0	Target Register for Transfer	Target Byte for Transfer During 8-Bit Mode	Target Byte for Transfer During 16-Bit Mode
0	0	0	×	×	Reserved (setting prohibited)	–	–
0	0	1	0	0	HDT(out)	Lower 8 bits	16 bits
0	0	1	0	1		Higher 8 bits	
0	0	1	1	0	HST	Lower 8 bits	16 bits
0	0	1	1	1		Higher 8 bits	
0	1	0	0	0	HDT(in)	Lower 8 bits	16 bits
0	1	0	0	1		Higher 8 bits	
0	1	0	1	×	Reserved (setting prohibited)	–	–
0	1	1	×	×	Not a target	–	–
1	×	×	×	×			

5.6.4 Timing of host interface

(1) Host read operation (μPD77210 Family → host)

Data is transferred from the μPD77210 Family to the host in the following steps. Figure 5-15 shows reading operations of 16-bit data to HDT without wait cycles.

<1> The application program of the μPD77210 Family stores data to the host data register (HDT) (a), (b).

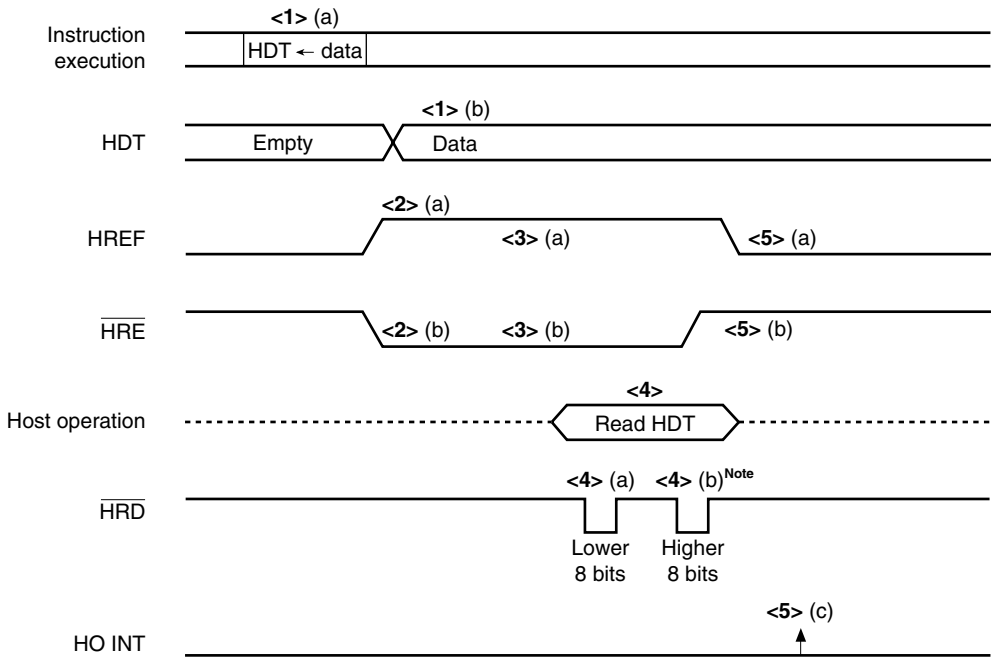
<2> Consequently, the host read enable flag (HREF) of the host interface status register (HST) is set to 1 (a). If the $\overline{\text{HRE}}$ mask bit (HREM) of HST is 0, the $\overline{\text{HRE}}$ pin becomes active (low), and is output to external devices as a hardware signal (b).

<3> The host can recognize that data is present in HDT by any of the following methods:
(1) Reads HST and detects HREF = 1 by software (a), or
(2) Detects the low level of the $\overline{\text{HRE}}$ pin (b).

<4> The host reads HDT. If 16-bit data is transferred at this time, the lower 8 bits (a) and then the higher 8 bits (b) must be read in this order. If 8-bit data is transferred, the higher 8 bits are always read (refer to the logic of HREF and $\overline{\text{HRE}}$).

<5> HREF of HST is cleared to 0 after step <4> (a), and the $\overline{\text{HRE}}$ pin becomes inactive (high) in step <4> (b). At this time, an interrupt request is generated by HO (c). This interrupt is processed as a valid interrupt or is recorded depending on the status of the corresponding interrupt enable flag and EI status (refer to **4.4.4 Interrupts**).

Figure 5-15. Host Read Sequence (μPD77210 Family → Host): HDT Write Without Wait



Note <4> (b) only in 16-bit bus mode.

(2) Host write operation (μPD77210 Family ← host)

Data is transferred from the host to the μPD77210 Family in the following steps. Figure 5-16 shows examples of writing HDT without wait cycles when 16-bit data is transferred.

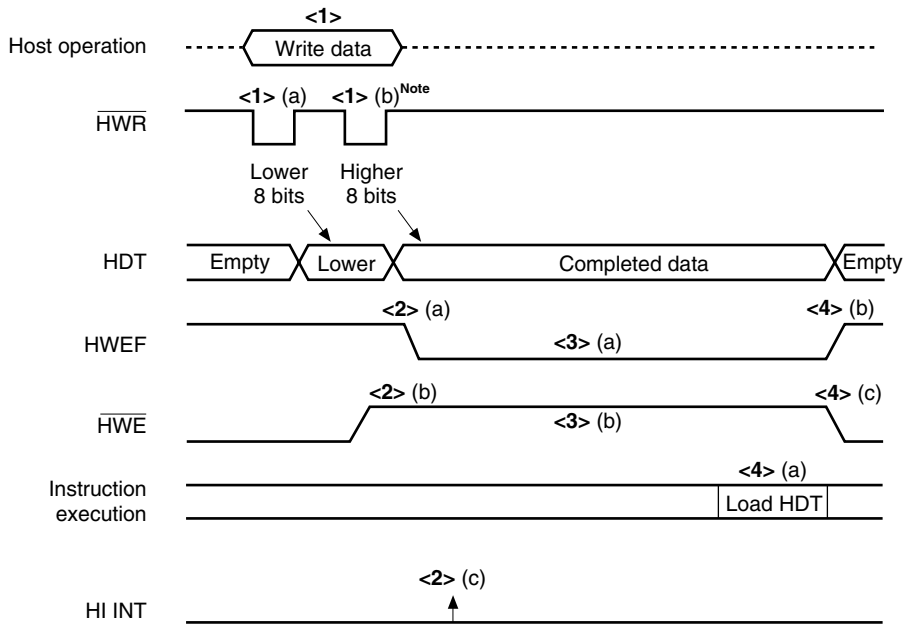
<1> The host writes data to the HDT of the μPD77210 Family. If 16-bit data is transferred at this time, the lower 8 bits (a) and the higher 8 bits (b) are written in this order; if 8-bit data is transferred, data is always written to the higher 8 bits of HDT (refer to the logic of HWEF and $\overline{\text{HWE}}$).

<2> Consequently, HWEF of HST is cleared to 0, informing the application program of the μPD77210 Family that data has been written to HDT (a). The $\overline{\text{HWE}}$ pin becomes inactive (high level in step (b)), informing an external device that HDT is busy (b). An interrupt request is also generated by HI (c). Whether this interrupt is processed as a valid interrupt or is recorded depends on the status of the corresponding interrupt request flag or EI status (refer to **4.4.4 Interrupts**).

<3> The application program of the μPD77210 Family can recognize that HDT is ready with the data from the host by either of the following methods:
(1) by detecting 0 of HWEF of HST (a), or
(2) by waiting for interrupt caused by HI (b).

<4> Consequently, the application program loads from HDT (a). As a result of the load, HWEF is set to 1 (b). At the same time, the $\overline{\text{HWE}}$ pin becomes active (low) (c), and the external circuit recognizes that HDT is enabled to write.

Figure 5-16. Host Write Sequence (μPD77210 Family ← Host): HDT Read Without Wait



Note <1> (b) only in 16-bit bus mode.

(3) Handshake

Handshaking between the μPD77210 Family and host can be established by:

- Polling
- Wait
- Interrupt

(a) Polling

Synchronization of handshaking is established by always monitoring and evaluating the host read enable flag (HREF) and host write enable flag (HWEF) of the host interface status register (HST). Here is an example of host read (μPD77210 Family → host) by means of polling:

```
/* Explicitly define HST and HDO because they are not reserved words. */
    #define HST         0x3807
    #define HDO         0x3806

/* Disable internal interrupts HO and HI. */
    R0L = SR            ;
    R0 = R0 | 0x0300    ;
    SR = R0L            ;

    R0L = 0x0           ; Set host status as follows:
    *HST:X = R0L        ; • Does not use HDT access wait function.
                        ; • Does not mask HRE function.
                        ; • Does not mask HWE function.
                        ; • Clears all user flags.
                        ; • Clears all error flags.

POLL: R0L = *HST:X      ; Judges HREF and loops to wait until host reads HDT.
      R0=R0 & 0x2       ;
      if (R0! = 0)jmp POLL ;

    *HDO:X = R1H        ; Data of R1H is output because HDT has become empty.
```

(b) Wait

Under the following conditions, execution of data exchanges with the HDT (in) and/or HDT (out) registers cause instruction wait cycles:

- When the load/store wait function is enabled (HAWE=1) and a store to HDT (out) is to be executed, while HREF=1 (valid data exists in HDT (out))
- When the load/store wait function is enabled (HAWE=1) and a load from HDT (in) is to be executed, while HWEF=1 (valid data does not exists in HDT (in))

Therefore, the advantage of this format is that writing handshake procedures is not required in application program, because the handshake procedure is automatically executed by hardware. Here is an example of host read by using the wait function:

```
/* Explicitly define HST and HDO because they are not reserved words. */
        #define HST         0x3807
        #define HDO         0x3806

/* Disable internal interrupts HO and HI. */
        R0L = SR            ;
        R0 = R0 | 0x0300    ;
        SR = R0L            ;

        R0L = 0x0400        ; Set host status as follows:
        *HST:X = R0L        ; • Uses HDT access wait function.
                            ; • Does not mask HRE function.
                            ; • Does not mask HWE function.
                            ; • Clears all user flags.
                            ; • Clears all error flags.

        *HDO:X = R1H        ; Outputs data of R1H.  If HDT is busy, wait cycle
                            ; is inserted.
```

Caution When data is written from the application program to HDT, the wait is not released unless HDT is read by the external device. If internal writing of DSP and external reading do not correspond on a one-to-one basis vis-a-vis HDT, a hang-up may occur.
During wait, interrupts are delayed (refer to 4.4.4 Interrupts).

(c) Interrupts

Handshaking can be established by generating an interrupt, if data can be stored to HDT(out) or loaded from HDT(in) by the μPD77210 Family. Therefore, the advantage of this format is that host input/output can be executed independently (asynchronously) of the other processing even while other processing is under execution. Here is an example of host input/output using an interrupt:

```
/* Define host I/O    */
      #define HST   *0x3807: X
      #define HDO   *0x3806: X
      #define HDI    *0x3806: X

/* Entry of interrupt vector table */
SegHi IMSEG AT 0x230      ; HIO input interrupt routine

      R0H = HDI           ; Read from HDT(in)
      *DP0++ = R0H        ; Saves to buffer
      RETI                ; Returns from interrupt
      NOP                 ;

SegHo IMSEG AT 0x234      ; HIO output interrupt routine

      R0H = *dp4++        ; Reads from buffer
      HDO = R0H           ; Writes to HDT(out)
      RETI                ; Returns from interrupt
      NOP                 ;

/* Disables interrupts to initialize host I/O */
      R1L = EIR           ; Disables all interrupts
      R1 = R1 | 0x8000    ; EI = 1
      EIR = R1L           ;
      NOP                 ; Two wait cycles are necessary until EI = 1 becomes
      NOP                 ; valid

      R0L = SR            ; Enables HI and HO interrupts
      R0 = R0&0xFCFF      ;
      SR = R0L            ;

      R1 = R1&0x7FFF      ; Enables all interrupts
      EIR = R1L           ;
      FINT                ; Discards previous interrupt

                          ; Initializes HDT
      R0L = 0x0           ; Without HDT access wait function
      HST = R0L           ; No HRE, HWE mask.  Clears user flag
      HD0 = R0L           ; Dummy store (Refer to Caution below)
```

Caution Because the host output interrupt occurs at the rising edge of the $\overline{\text{HRD}}$ pin when the higher byte of the HDT register is accessed, the following points must be noted.

(1) Transfer the first data by forcibly generating an interrupt by transferring dummy data or by transferring data without using an interrupt, depending on the interrupt mode.

(2) If data is transferred in the burst mode, the chances are that the first data for the next burst transfer is not generated if an interrupt occurs at the last word of the burst data. Therefore, disable the interrupt by the instruction immediately before the one that transfers the last word to HDT, execute the same instruction as (1) after generation of the next burst data has been completed, and transfer the next burst data.

5.7 External Data Memory Interface (MIO)

The external data memory interface (MIO) functions as an interface with external memory. A 1 Mword (20-bit address) external data memory space can be accessed from MIO.

MIO's main features are as follows.

- Direct access
  When 0x3F is set to the DPR (data page register), external data memory is accessed via the μPD77210 Family device's internal memory space from 0x8000 to 0xFFFF.
  During access, index address addition is executed to perform 20-bit addressing.

- MDT (memory data register) access and automatic address generation function
  The address where access will begin is set in advance and a load/store operation for the MDT register enables access to external data memory.
  For address generation, there are three types of access: zero-dimensional access in which the initial address is held (+0 increment), one-dimensional access in which an increment of one is added to the initial address, and two-dimensional access in which the line length and offset length are used as parameters.

- Timing adjustment function
  In the external data memory interface, the access cycle, data/address signal, and setup/hold cycle for memory write can be set programmable by setting registers.

Figure 5-17 shows a block diagram of the external data memory interface and Table 5-15 lists the registers.

Figure 5-17. Block Diagram of External Data Memory Interface

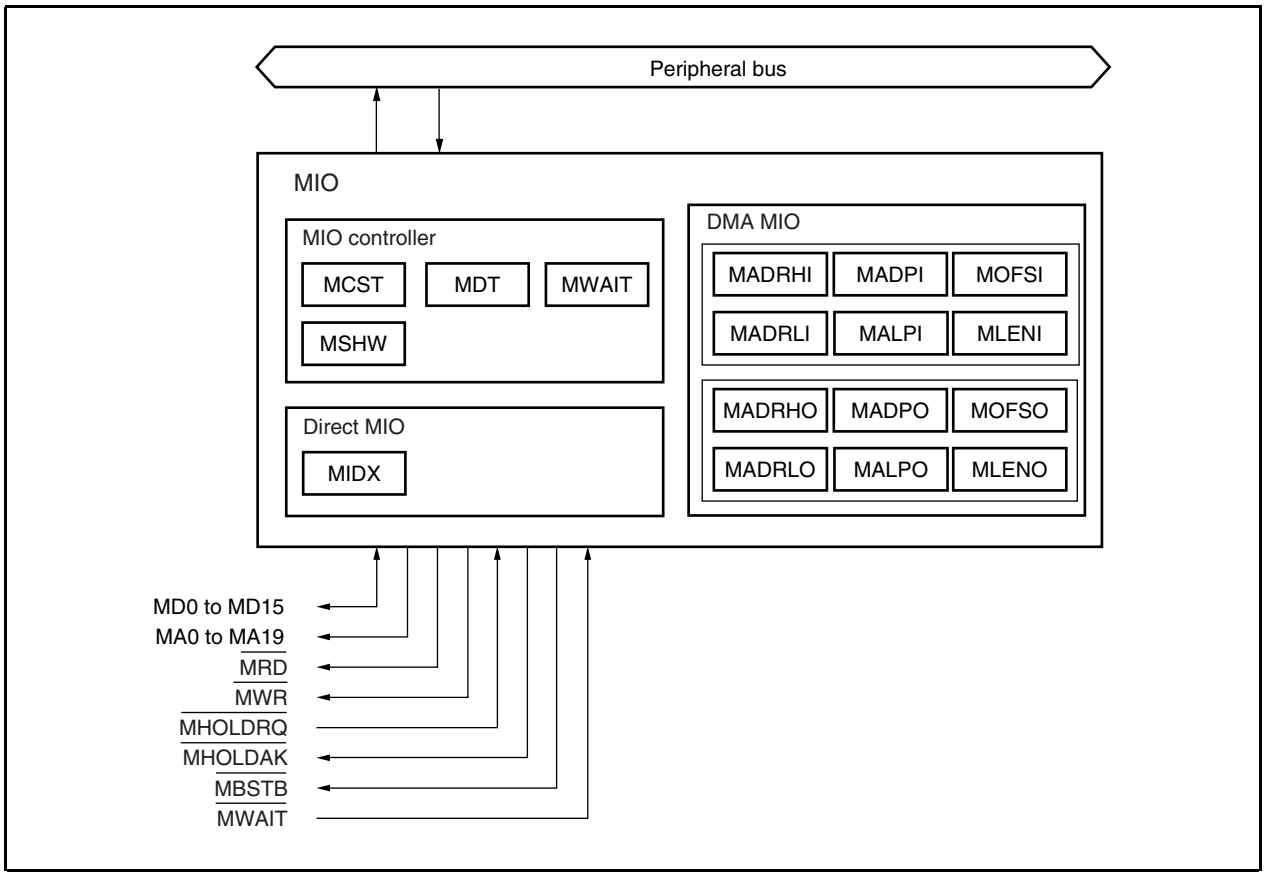


Table 5-15. External Data Memory Interface Registers

X/Y Memory Address	Register Name	Function	Load/Store
0x3840	MDT	Memory data register	L/S
0x3841	MSHW	Memory interface setup/hold width setting register	L/S
0x3842	MCST	Memory interface control/status register	L/S
0x3843	MWAIT	Memory interface wait register	L/S
0x3844	MIDX	Direct access index register	L/S
0x3845	MADRLI	Start address register (lower) for memory interface input	L/S[Note]
0x3846	MADRHI	Start address register (higher) for memory interface input	L/S[Note]
0x3847	MOFSI	Line offset register for memory interface input	L/S[Note]
0x3848	MLENI	Line length register for memory interface input	L/S[Note]
0x3849	MADRLO	Start address register (lower) for memory interface output	L/S[Note]
0x384A	MADRHO	Start address register (higher) for memory interface output	L/S[Note]
0x384B	MOFSO	Line offset register for memory interface output	L/S[Note]
0x384C	MLENO	Line length register for memory interface output	L/S[Note]

Note Load is disabled in the μPD77210.

5.7.1 Memory interface pins

(1) MA0 to MA19 (memory address – output)

These are 20-bit memory address pins. They become high impedance when the bus is released.

(2) MD0 to MD15 (memory data – I/O)

These are 16-bit data I/O pins. They become high impedance when the bus is released and when external data memory is not being accessed.

(3) $\overline{\text{MRD}}$ (memory read strobe – output)

This is a read strobe output pin for external data memory. It becomes high impedance when the bus is released.

(4) $\overline{\text{MWR}}$ (memory write strobe – output)

This is a write strobe output pin for external data memory. It becomes high impedance when the bus is released.

(5) $\overline{\text{MHOLDRQ}}$ (bus hold request – input)

This pin is used to input the external data memory's bus occupancy request signal. In a system where several devices share the same bus, this signal is used for bus arbitration.

When $\overline{\text{MHOLDRQ}}$ is low level, the bus is released after completion of the current bus cycle.

(6) $\overline{\text{MHOLDAK}}$ (bus hold acknowledge – output)

This pin is used to output bus access enabled signals to devices that share the external data memory bus. Output is low level when the bus is released to external device.

(7) $\overline{\text{MBSTB}}$ (bus strobe – output)

This pin is used to output external data memory bus use request signals.

(8) $\overline{\text{MWAIT}}$ (memory wait – input)

This pin is used to input inserted access wait cycles. A wait cycle is inserted when this pin becomes low level during external data memory access.

Insertion of wait cycles to the $\overline{\text{MWAIT}}$ pin is prohibited before the falling edge of $\overline{\text{MRD}}$ and $\overline{\text{MWR}}$ or when the bus is released.

5.7.2 External data memory interface registers

(1) MDT (MIO data transfer register)

MDT is a 16-bit register that is used for input and output of data via the external data memory interface.

(2) MSHW (MIO setup hold width register)

MSHW is a 16-bit register that is used to set the memory read/write strobe's setup/hold cycles. The setup/hold timing of the memory read/write strobe can be adjusted in cycle units.

Table 5-16 lists the functions of each bit in MSHW. The default value is 0xFFFF. The following cautions must be observed.

- The number of hold cycles cannot be controlled (it is fixed at zero cycles) for memory read operations by the μPD77213.
- Neither the setup cycles nor the hold cycles can be controlled (they are fixed at zero cycles) for memory read operations by the μPD77210.
- Access is prohibited when the number of setup or hold cycles is zero.

(3) MWAIT (MIO wait register)

MWAIT is a 16-bit register that is used to set the number of wait cycles to be inserted during external data memory access.

Table 5-17 lists the functions of each bit in MWAIT. The default value is 0xFFFF.

Access is prohibited when the number of wait cycles is zero.

(4) MCST (MIO control/status register)

MCST is an 8-bit register that is used to set the memory interface's status and DMA transfer mode. Table 5-18 lists the functions of each bit in MCST.

(5) MIDX (MIO direct access index register)

MIDX is an 8-bit register that is used for address modification during direct access.

The MIDX register provides the higher 8 bits of the 20-bit address used when accessing the external data memory via the external data memory window, which is assigned to 0x8000 to 0xFFFF in the data memory and which is valid when DPR = 0x3F.

(6) MADR (MIO start address register)

This register is used to retain the start address in external memory during DMA access.

To perform 20-bit addressing, one register is used for the higher four bits and another register is used for the lower 16 bits. In addition, there are separate registers for input and output. Data is transferred to MAPI or MAPO under the control of the MCST register.

MADRLI and MADRHI are registers that indicate the start address for input. MADRLI indicates the lower 16 bits and MADRHI indicates the higher 4 bits of the 20-bit start address.

MADRLO and MADRHO are registers that indicate the start address for output. MADRLO indicates the lower 16 bits and MADRHO indicates the higher 4 bits of the 20-bit start address.

Caution Load is disabled in the μPD77210.

(7) MOFS (MIO line offset register)

The MOFS register is used to set the line interval offset for MDT access. It sets 0x0 for zero-dimensional access, 0x1 for one-dimensional access, and the offset length for two-dimensional access.

There are separate input and output registers: MOFSI is for input and MOFSO is for output.

Caution Load is disabled in the μPD77210.

(8) MLEN (MIO line length register)

The MLEN register is used to set the line length for MDT access. It sets 0x1 for zero-dimensional and one-dimensional access and the line length for two-dimensional access.

There are separate input and output registers: MLENI is for input and MLENO is for output.

Caution Load is disabled in the μPD77210.

(9) MADP (DMA access data address register)

MADP is a 20-bit register that is used to indicate addresses during DMA access. It includes separate input and output registers.

MADPI is the address register for input and MADPO is the address register for output.

The value of MDAR is transferred to MADP when a store instruction to MCST is executed. The value of MADP is updated automatically when MDT is accessed.

MADP is not connected to the peripheral bus.

(10) MALP (DMA access remaining data length register)

MALP is a 16-bit register that is used indicate the number of transfer lines remaining during DMA access. It includes separate input and output registers.

MALPI is the remaining transfer lines register for input and MALPO is the remaining transfer lines register for output.

The value of MLEN is transferred to MALP when a store instruction to MCST is executed. The value of MALP is updated automatically when MDT is accessed.

MALP is not connected to the peripheral bus.

Table 5-16. Bit Configuration of MSHW

Bit	Name	Function	Load/Store
15 and 14	Dside	D side (0xC0000 to 0xFFFFF) access setup cycles • 0x0: 0 cycles • 0x1: 1 cycle • 0x2: 2 cycles • 0x3: 3 cycles (default)	L/S
13 and 12		D side (0xC0000 to 0xFFFFF) access hold cycles • 0x0: 0 cycles • 0x1: 1 cycle • 0x2: 2 cycles • 0x3: 3 cycles (default)	L/S
11 and 10	Cside	C side (0xC80000 to 0xBFFFF) access setup cycles • 0x0: 0 cycles • 0x1: 1 cycle • 0x2: 2 cycles • 0x3: 3 cycles (default)	L/S
9 and 8		C side (0x80000 to 0xBFFFF) access hold cycles • 0x0: 0 cycles • 0x1: 1 cycle • 0x2: 2 cycles • 0x3: 3 cycles (default)	L/S
7 and 6	Bside	B side (0x40000 to 0x7FFFF) access setup cycles • 0x0: 0 cycles • 0x1: 1 cycle • 0x2: 2 cycles • 0x3: 3 cycles (default)	L/S
5 and 4		B side (0x40000 to 0x7FFFF) access hold cycles • 0x0: 0 cycles • 0x1: 1 cycle • 0x2: 2 cycles • 0x3: 3 cycles (default)	L/S
3 and 2	Aside	A side (0x00000 to 0x3FFFF) access setup cycles • 0x0: 0 cycles • 0x1: 1 cycle • 0x2: 2 cycles • 0x3: 3 cycles (default)	L/S
1 and 0		A side (0x00000 to 0x3FFFF) access hold cycles • 0x0: 0 cycles • 0x1: 1 cycle • 0x2: 2 cycles • 0x3: 3 cycles (default)	L/S

Caution When the setup/hold value is zero, access to the corresponding memory area is prohibited.

Table 5-17. Bit Configuration of MWAIT

Bit	Name	Function	Load/Store
15 to 12	Dside	D side (0xC0000 to 0xFFFFF) access wait cycles • 0x0: 0 cycles • 0x1: 1 cycle • 0x2: 2 cycles • 0x3: 3 cycles • 0x4: 4 cycles • 0x5: 5 cycles • 0x6: 6 cycles • 0x7: 7 cycles • 0x8: 8 cycles • 0x9: 9 cycles • 0xA: 10 cycles • 0xB: 11 cycles • 0xC: 12 cycles • 0xD: 13 cycles • 0xE: 14 cycles • 0xF: 15 cycles (default)	L/S
11 to 8	Cside	C side (0x80000 to 0xBFFFF) access wait cycles • 0x0: 0 cycles • 0x1: 1 cycle • 0x2: 2 cycles • 0x3: 3 cycles • 0x4: 4 cycles • 0x5: 5 cycles • 0x6: 6 cycles • 0x7: 7 cycles • 0x8: 8 cycles • 0x9: 9 cycles • 0xA: 10 cycles • 0xB: 11 cycles • 0xC: 12 cycles • 0xD: 13 cycles • 0xE: 14 cycles • 0xF: 15 cycles (default)	L/S
7 to 4	Bside	B side (0x40000 to 0x7FFFF) access wait cycles • 0x0: 0 cycles • 0x1: 1 cycle • 0x2: 2 cycles • 0x3: 3 cycles • 0x4: 4 cycles • 0x5: 5 cycles • 0x6: 6 cycles • 0x7: 7 cycles • 0x8: 8 cycles • 0x9: 9 cycles • 0xA: 10 cycles • 0xB: 11 cycles • 0xC: 12 cycles • 0xD: 13 cycles • 0xE: 14 cycles • 0xF: 15 cycles (default)	L/S
3 to 0	Aside	A side (0x00000 to 0x3FFFF) access wait cycles • 0x0: 0 cycles • 0x1: 1 cycle • 0x2: 2 cycles • 0x3: 3 cycles • 0x4: 4 cycles • 0x5: 5 cycles • 0x6: 6 cycles • 0x7: 7 cycles • 0x8: 8 cycles • 0x9: 9 cycles • 0xA: 10 cycles • 0xB: 11 cycles • 0xC: 12 cycles • 0xD: 13 cycles • 0xE: 14 cycles • 0xF: 15 cycles (default)	L/S

Caution When the wait value is zero, access to the corresponding memory area is prohibited.

Table 5-18. Bit Configuration of MCST

Bit	Name	Function	Load/Store
7	IDS	Input MADP set request flag • 0: No request (default) • 1: Request sent When the value is 1, the contents of MADRLI and MADRHI are transferred to MADPI. After transfer is executed, the value becomes 0.	L/S
6	ODS	Output MADP set request flag • 0: No request (default) • 1: Request sent When the value is 1, the contents of MADRLO and MADRHO are transferred to MADPI. After transfer is executed, the value becomes 0.	L/S
5	ILS	Input MALP set request flag • 0: No request (default) • 1: Request sent When the value is 1, the contents of MLENI is transferred to MALPI. After transfer is executed, the value becomes 0.	L/S
4	OLS	Output MALP set request flag • 0: No request (default) • 1: Request sent When the value is 1, the contents of MLENO is transferred to MALPO. After transfer is executed, the value becomes 0.	L/S
3	WM	Memory wait request monitor Monitors the $\overline{\text{MWAIT}}$ pin. • 0: Request sent • 1: Request not sent	L
2	HR	Memory hold request monitor Monitors the $\overline{\text{MHOLDRQ}}$ pin. • 0: Request not sent • 1: Request sent	L
1	SB	Memory access monitor Monitors the $\overline{\text{MBSTB}}$ pin. • 0: No current access • 1: Currently being accessed	L
0	HA	Memory hold acknowledge monitor Monitors the $\overline{\text{MHOLDAK}}$ pin. • 0: No current access • 1: Currently being accessed	L

5.7.3 Direct access

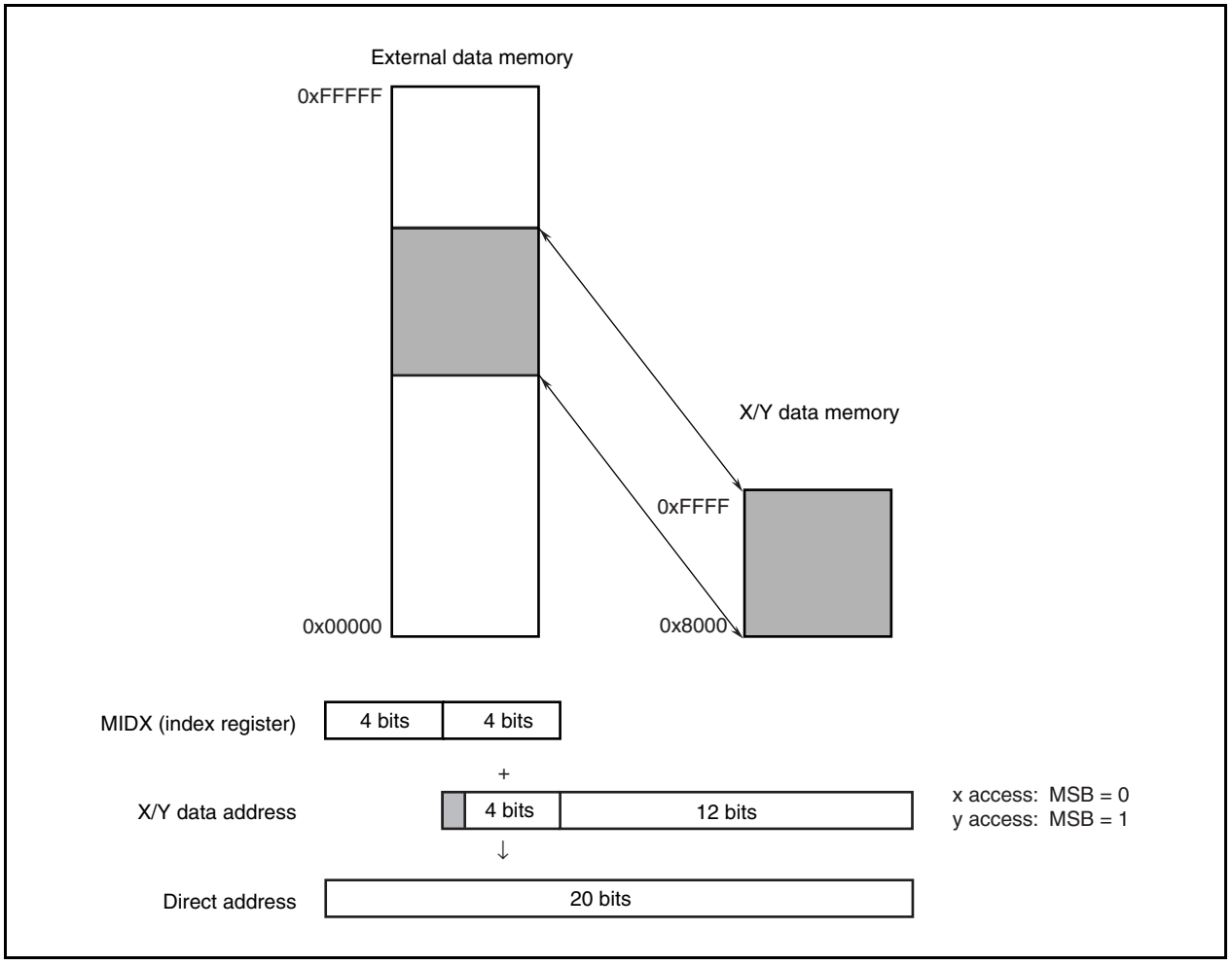
This is a method for directly accessing the external data memory via an external data memory access window. The external data memory access window is assigned to the range of 0x8000 to 0xFFFF in either X or Y data memory when the DPR is 0x3F.

The external data memory space is a 20-bit address space (1 Mword × 16 bits). Addresses are generated as described below.

- The MIDX (direct access index register) provides the higher 8 bits of the 20-bit address.
- The address used to access the range of 0x8000 to 0xFFFF (DPR = 0x3F) in X or Y data memory is added to MIDX to generate a 20-bit address.
- If accessing from the X data side, the MSB of the 16-bit address is 0 and an address in the range of 0x0000 to 0x7FFF is added to MIDX. If accessing from the Y data side, the MSB of the 16-bit address is 1 and an address in the range of 0x8000 to 0xFFFF is added to MIDX.

Figure 5-18 shows an image of direct access.

Figure 5-18. Direct Access



5.7.4 DMA access

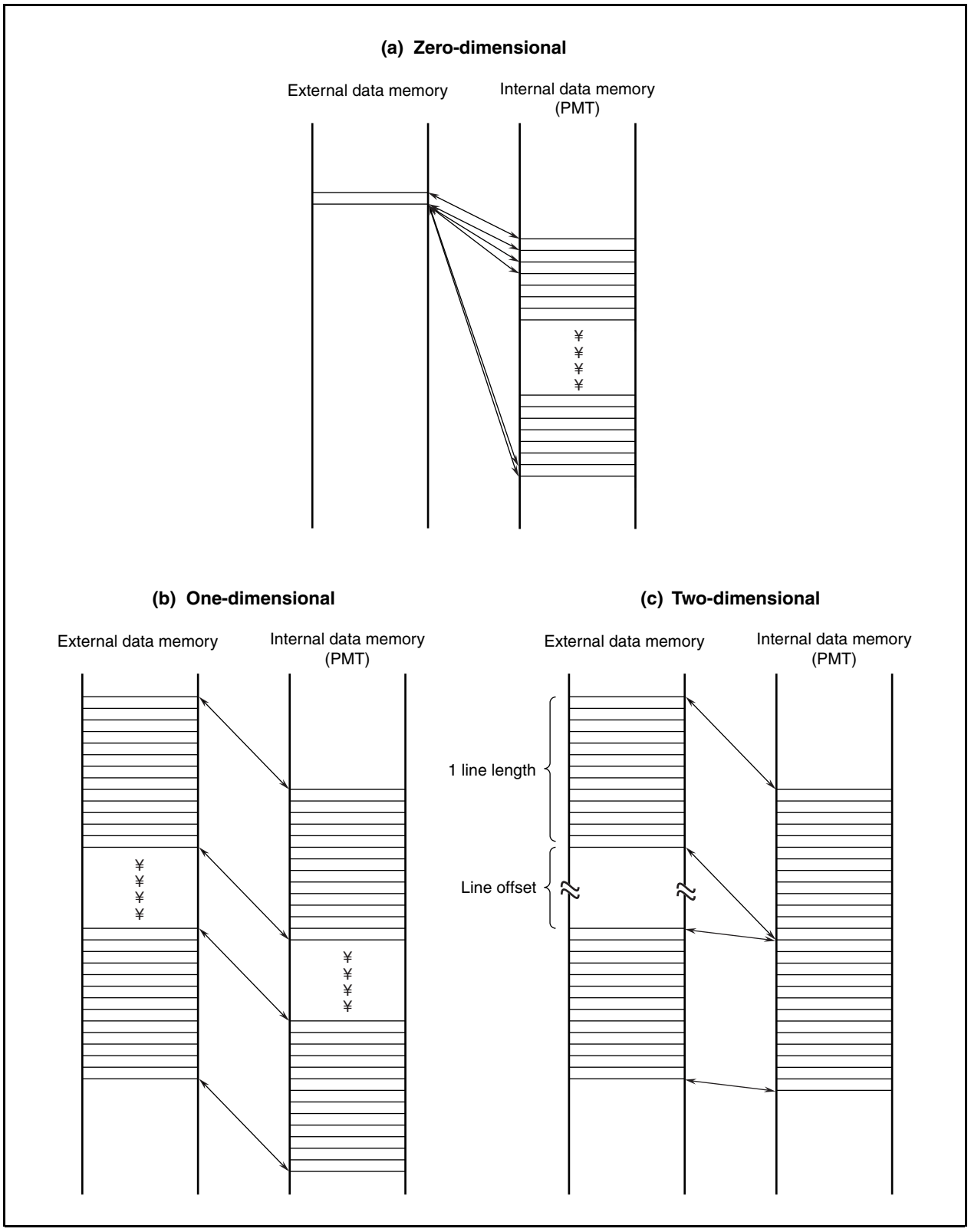
This is a method for accessing external data memory via a memory mapped MDT (memory data register).

The MADP value set by MADR is used as the address for accessing external data memory. MADR is a register that indicates the start address for DMA transfer, and when the corresponding bits in MCST are set, the transfer from MADR to MADP is executed. When MDT is accessed, the value of MADP is automatically updated. It can be updated in any of the following three ways.

- Zero-dimensional (0D) addressing that holds the current value
  It is assumed that this method will be used to access a particular external memory mapped peripheral or other device.
- One-dimensional (1D) addressing that increments (by one)
  It is assumed that this method will be used for block transfers between external memory and internal memory.
- Two-dimensional (2D) addressing that adds an offset to each line length
  It is assumed that this method will be used to access a rectangular area in memory, such as frame memory used for image processing.

The line length and offset length used for two-dimensional addressing are set in the MOFS and MLEN registers. 0x0 and 0x1 are set to these registers respectively during zero-dimensional addressing, and 0x1 and 0x1 are set respectively during one-dimensional addressing. Figure 5-19 shows an image of DMA access.

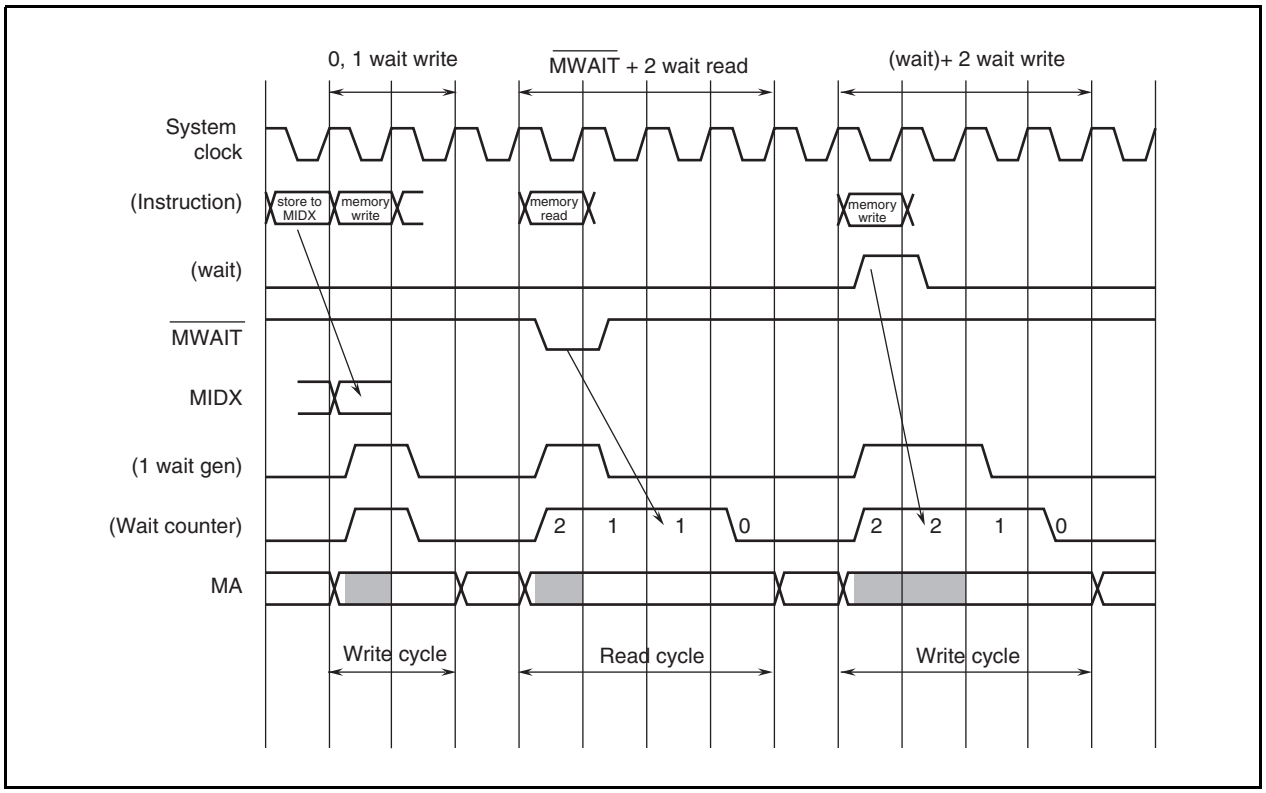
Figure 5-19. DMA Access



5.7.5 Direct access timing

Figure 5-20 shows the direct access timing.

Figure 5-20. Direct Access Timing



A store instruction to MIDX can be executed immediately prior to direct access.

In addition to the programmable wait (wait counter) provided by the MWAIT register there is a one wait circuit (1 wait gen), and at the first load/store cycle the index is incremented and the address pointer is refreshed. Accordingly, one wait will occur no matter what the programmable wait setting is.

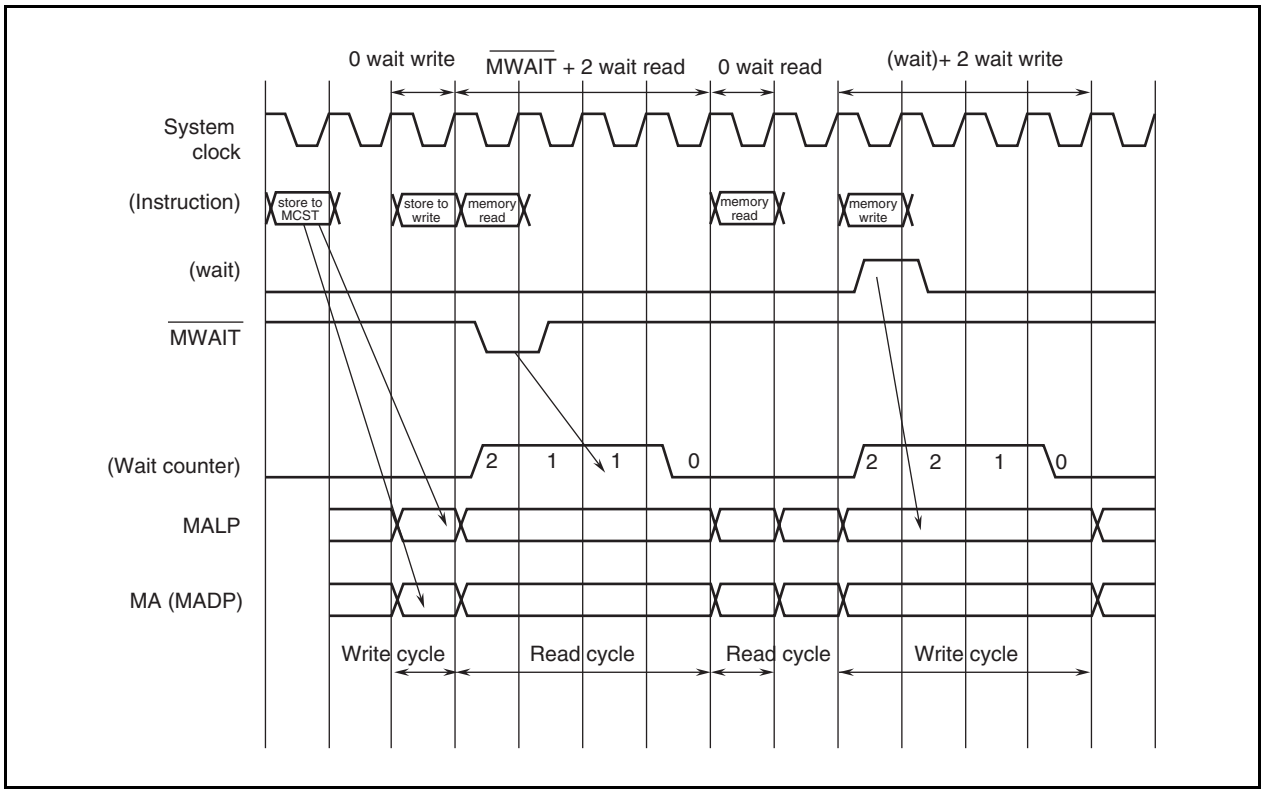
While the external $\overline{\text{MWAIT}}$ pin is asserted, the access cycle is extended. (The $\overline{\text{MWAIT}}$ pin is sampled during the access cycle and updating the programmable wait count is held pending. The $\overline{\text{MWAIT}}$ pin is used to implement access cycles over and above the programmable wait and it is assumed that an adequate number of wait cycles have been set.)

During the time that a wait by DSP core kernel access is asserted, the assert cycle is extended by the combination of the programmable wait and the one-shot wait from the address incrementation stage.

5.7.6 DMA access timing

Figure 5-21 shows the DMA access timing.

Figure 5-21. DMA Access Timing



When a store instruction is executed for the MCST register, the initial address is set to the MADP register and the number of remaining transfer lines is set to the MALP register (store instructions to MADR, MOFS, and MLEN are executed in advance). However, because these settings are not made until two cycles after the MCST store instruction is executed, access is not possible immediately after MSCT store instruction execution. When the access cycle is completed, MADP and MALP are both updated.

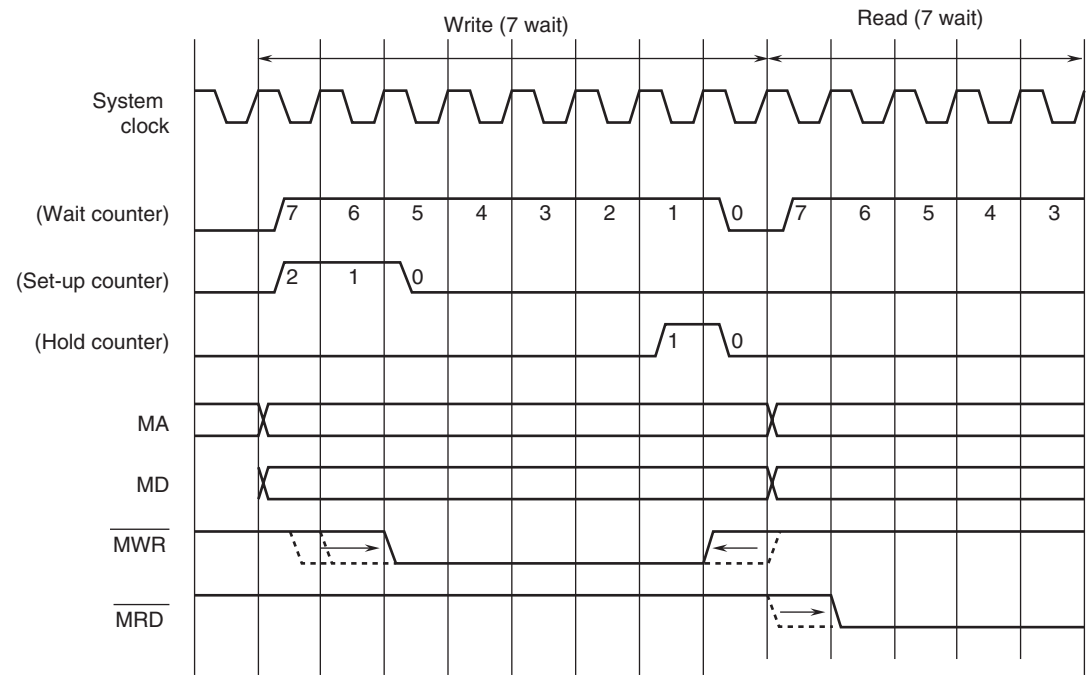
While the external $\overline{\text{MWAIT}}$ pin is asserted, the access cycle is extended. (The $\overline{\text{MWAIT}}$ pin is sampled during the access cycle and updating the programmable wait count is held pending. The $\overline{\text{MWAIT}}$ pin is used to implement access cycles over and above the programmable wait and it is assumed that an adequate number of wait cycles have been set.)

During the time that a wait by DSP core kernel access is asserted, updating the programmable wait count is held pending and the access cycle is extended.

5.7.7 Timing of memory access

Figure 5-22 shows the memory access timing.

Figure 5-22. Memory Access Timing



Remark $\overline{\text{MRD}}$ can be used for setup control of the μPD77213 only.

The memory write strobe ($\overline{\text{MWR}}$) is needed to ensure the setup/hold period due to the risk of write errors to the address bus (MADD) and the memory data bus (MD).

In this MIO, the setup/hold time can be set to the MSHW register using the machine cycle. The strobe just before and after the write cycle is masked. In the example shown in the figure, the setup value is 0x2 and the hold value is 0x1. The mask signal is extended in cases where the programmable wait count update is retained. The setup side is set to zero setup cycles in order to prevent timing problems caused by decoding, so that a 0.5-cycle setup cycle occurs.

The following control is used for the memory read strobe ($\overline{\text{MRD}}$).

- μPD77210: No control of setup or hold.
- μPD77213: Only setup is controlled (no hold control).

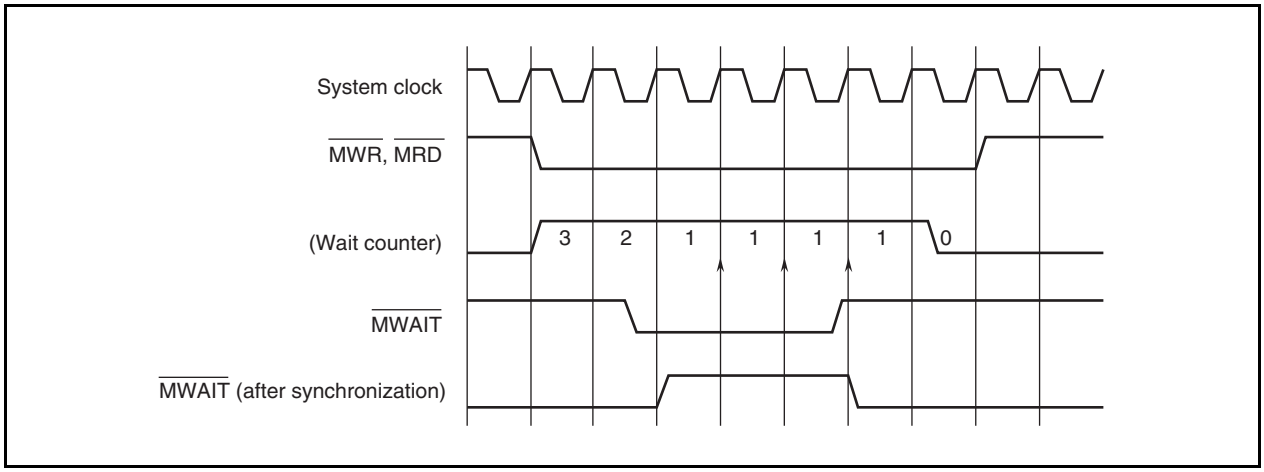
When in continuous read mode, the signal is active low during the read access cycle. In the μPD77213, if the setup value has been set as a value other than zero, a high level signal is inserted. Also, during DMA transfer, a high level signal is inserted once per transfer but if a read has occurred as part of the program during DMA transfer, the result is the same as when in continuous read mode.

When zero has been set for wait, setup, and hold, access to the corresponding areas is prohibited. The shortest memory access is a three-cycle access, in which the setup and hold are set to 1 and the wait to is set to 2.

The following describes wait insertion and operation using the external $\overline{\text{MWAIT}}$ pin. After the falling edge of the $\overline{\text{MRD}}$ and $\overline{\text{MWR}}$ pins, the $\overline{\text{MWAIT}}$ pin is asserted active (low level) and the signal used for this synchronization stops the updating of the programmable wait counter. When the $\overline{\text{MWAIT}}$ pin is deasserted inactive (high level), updating of the counter is resumed.

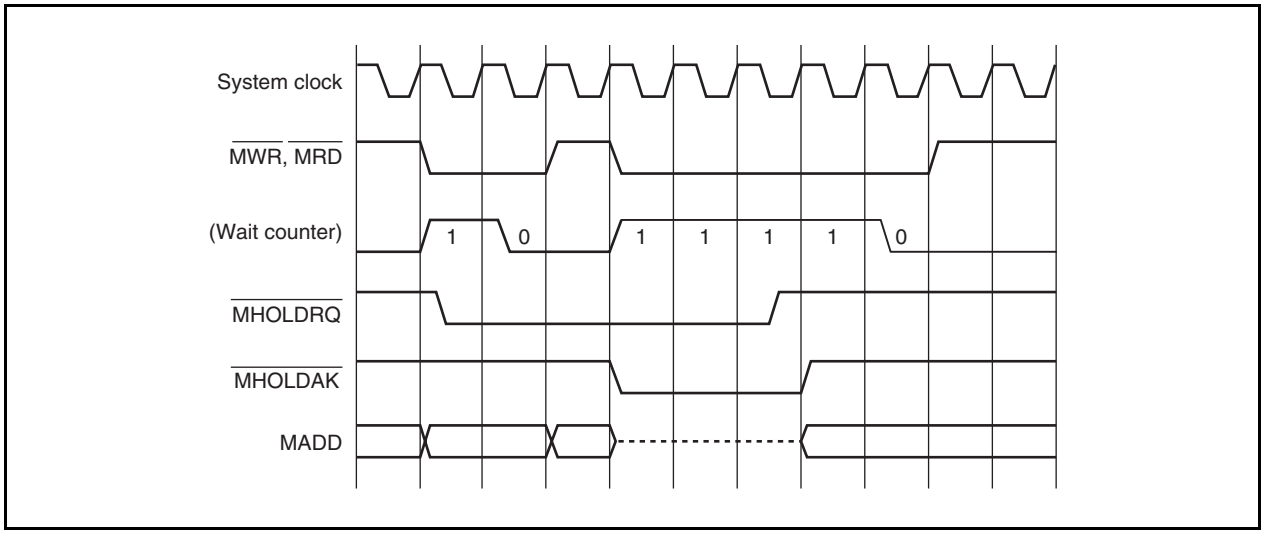
Assertion of the $\overline{\text{MWAIT}}$ pin as active is prohibited prior to the falling edge of the $\overline{\text{MRD}}$ and $\overline{\text{MWR}}$ pins or while the bus is released.

Figure 5-23. Insertion of Access Wait Cycles via $\overline{\text{MWAIT}}$ Pin



The following describes the bus arbitration. After the $\overline{\text{MHOLDRQ}}$ pin has been asserted active (low level), synchronization is executed, and the $\overline{\text{MHOLDAK}}$ pin goes low if the μPD77210 Family device is not accessing external data memory. This releases the bus (sets high impedance) and enables an external device to use the bus. Once the bus has been released, if the μPD77210 Family device accesses external data memory, the wait counter update operation is stopped.

Figure 5-24. Bus Arbitration



5.8 Peripheral ↔ Memory Transfer (PMT)

The peripheral ↔ memory transfer (PMT) controller is a DMA controller that transfers data from four peripheral circuits (TSIO, ASIO, HIO, and MIO) directly to the μPD77210 Family device's internal memory. There are eight channels in all: four each for input and output.

The μPD77210 Family device's internal memory area that can be accessed by the PMT controller is the 14-Kword data space from 0x0000 to 0x37FF.

Input and output to and from these peripheral circuits can be performed automatically simply by loading and storing data to this memory space.

During data input, once one word of data has been input to the peripheral circuits, it is transferred to memory.

During data output, once the data has been output from the peripheral circuits, the next word of data is loaded from memory. The peripheral circuits generate an interrupt signal after each word of data is input or output and a transfer request is sent to the PMT. Once the specified number of words have been input or output, an interrupt occurs in the DSP core kernel. Transfers are not halted by interrupts that occur during the transfers.

Figure 5-25 shows a block diagram of the PMT controller, and Table 5-19 lists its registers.

Figure 5-25. Block Diagram of PMT Controller

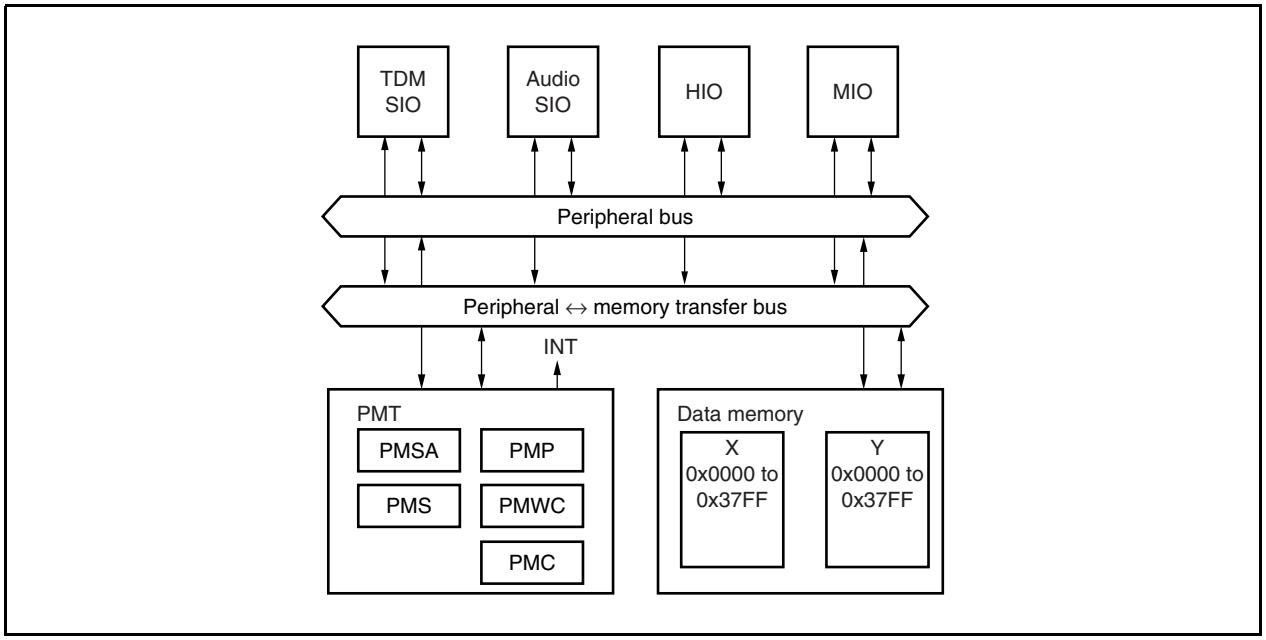


Table 5-19. PMT Registers

X/Y Memory Address	Register Name	Function	Load/Store
0x3850	PMSA0	PMT start address register 0	L/S
0x3851	PMS0	PMT size register 0	L/S
0x3852	PMC0	PMT control register 0	L/S
0x3853	PMP0	PMT address pointer 0	L
0x3854	PMSA1	PMT start address register 1	L/S
0x3855	PMS1	PMT size register 1	L/S
0x3856	PMC1	PMT control register 1	L/S
0x3857	PMP1	PMT address pointer 1	L
0x3858	PMSA2	PMT start address register 2	L/S
0x3859	PMS2	PMT size register 2	L/S
0x385A	PMC2	PMT control register 2	L/S
0x385B	PMP2	PMT address pointer 2	L
0x385C	PMSA3	PMT start address register 3	L/S
0x385D	PMS3	PMT size register 3	L/S
0x385E	PMC3	PMT control register 3	L/S
0x385F	PMP3	PMT address pointer 3	L
0x3860	PMSA4	PMT start address register 4	L/S
0x3861	PMS4	PMT size register 4	L/S
0x3862	PMC4	PMT control register 4	L/S
0x3863	PMP4	PMT address pointer 4	L
0x3864	PMSA5	PMT start address register 5	L/S
0x3865	PMS5	PMT size register 5	L/S
0x3866	PMC5	PMT control register 5	L/S
0x3867	PMP5	PMT address pointer 5	L
0x3868	PMSA6	PMT start address register 6	L/S
0x3869	PMS6	PMT size register 6	L/S
0x386A	PMC6	PMT control register 6	L/S
0x386B	PMP6	PMT address pointer 6	L
0x386C	PMSA7	PMT start address register 7	L/S
0x386D	PMS7	PMT size register 7	L/S
0x386E	PMC7	PMT control register 7	L/S
0x386F	PMP7	PMT address pointer 7	L

The PMT controller includes 8 transfer channels. The transfer direction is fixed for each transfer channel: the direction is “peripheral → memory” for even-numbered channels and “memory → peripheral” for odd-numbered channels.

Table 5-20 shows the relationship between PMT transfer channels and peripherals.

Table 5-20. PMT Transfer Channels and Target Peripherals

PMT Channel	Target Peripheral	Transfer Direction
0	TDM serial interface (input)	Peripheral → memory
1	TDM serial interface (output)	Memory → peripheral
2	Audio serial interface (input)	Peripheral → memory
3	Audio serial interface (output)	Memory → peripheral
4	Host interface (input)	Peripheral → memory
5	Host interface (output)	Memory → peripheral
6	Memory interface (input)	Peripheral → memory
7	Memory interface (output)	Memory → peripheral

5.8.1 PMT registers

(1) PMSA0 to PMSA7 (PMT start address registers)

These are 16-bit registers that are used to set and hold the start address of the buffer area that is established in the internal data RAM area. Its contents are transferred to PMP when PMT transfer starts. These registers exist for each channel and the default value is 0x0.

(2) PMS0 to PMS7 (PMT size registers)

These are 16-bit registers that are used to set the size of the buffer that is established in the internal data RAM area. Their contents are transferred to PMWC when PMT transfer starts. These registers exist for each channel and the default value is 0x0.

(3) PMC0 to PMC7 (PMT control registers)

These registers are used to set PMT operations and to indicate their status. See **Table 5-21 Bit Configuration of PMC** for description of the functions of the PMC bits. These registers exist for each channel and the default value is 0x0.

(4) PMP0 to PMP7 (PMT pointer)

This register is used to retain the target address in internal data RAM for transfers from PMT. It enables load only. These registers exist for each channel and the default value is 0x0.

(5) PMWC (PMT word counter)

This is a 16-bit register that is used to indicate the remaining number of transfers from PMT. This register exists for each channel.

PMWC is not connected to the peripheral bus. Neither load nor store is enabled.

Table 5-21. Bit Configuration of PMC

Bit	Name	Function	Load/Store
15 to 7	Reserved	Values other than 0 cannot be written. • Undefined during a read operation	–
6	BM	Transfer monitor flag • 0: Transfer in progress (or initial status) • 1: Transfer not in progress Value is 0 when EN is set (= 1). (BM, EN) = 10 is determined when PMT is judged as completed during a program.	L
5	ST	Transfer mode setting bit • 0: Continues transfer (default) • 1: Stops transfer This bit specifies whether to stop or continue transfer each time the buffer is finished.	L/S
4	EN	Buffering enable bit • 0: Specifies stopping PMT transfer (default) • 1: Specifies starting PMT transfer This bit sets the start of the buffering operation. When ST's value is 1, it becomes zero after a transfer is completed.	L/S
3 and 2	IPT	Interrupt cycle setting bits • 00: Full. When transfer of full (1/1) buffer size is completed (default) • 01: Half. When transfer of one half (1/2) of buffer size is completed • 10: Quarter. When transfer of one quarter (1/4) of buffer size is completed • 11: Reserved (setting prohibited) These bits set the interrupt interval. An interrupt occurs when the entire buffer is filled or when one half or one quarter is filled.	L/S
1 and 0	MSXY	Buffer memory setting bits • 00: X data memory (default) • 01: Y data memory • 10, 11: Reserved (setting prohibited) These bits set the memory to be used as a buffer.	L/S

5.8.2 PMT operation modes

PMT specifies the start address and buffer size (see **Figure 5-26**).

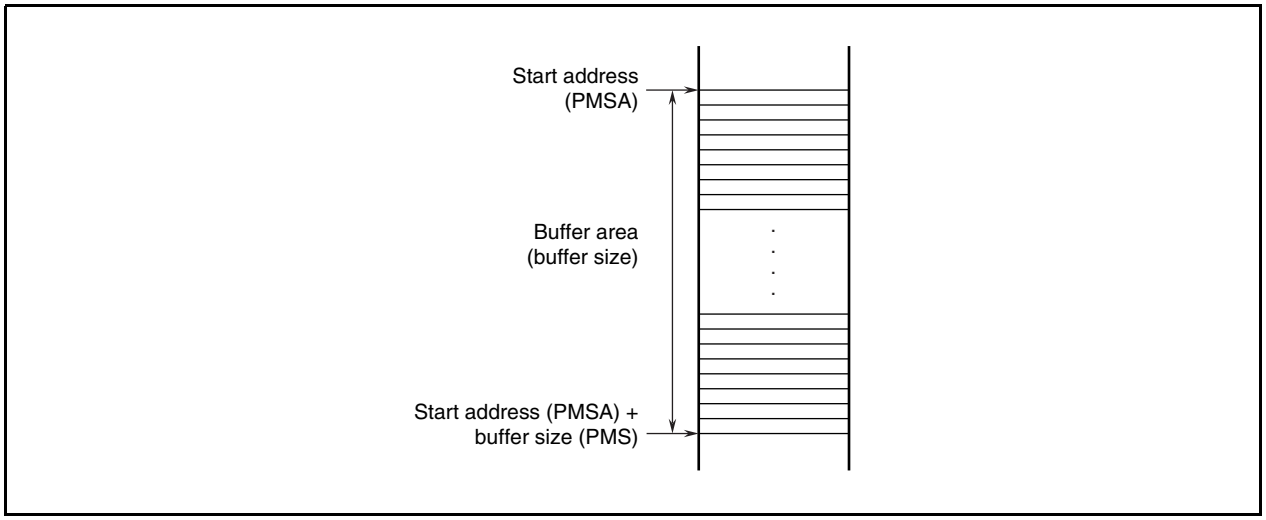
When the operation is started, if it is an input operation, data sent from the peripheral is stored beginning at the start address, and if it is an output operation, data is read beginning from the start address and is sent to the peripheral. When access reaches the end of the buffer, it returns to the start address.

The interrupt timing can be set as one half or one quarter of the buffer size. The interrupt occurs when one half or one quarter of the buffer data has been transferred.

When the mode is set for an interrupt to occur at one half of the buffer size, be sure to set 0 to the lower bit for buffer size specification. When the mode is set for an interrupt to occur at one quarter of the buffer size, be sure to set 0 to the lower two bits for buffer size specification.

There are two modes: a mode in which transfer stops when an interrupt occurs and a mode in which transfer continues when an interrupt occurs. Even if a synchronization signal, such as with a serial interface, is continuously transferred, the buffer fill timing is predetermined, so the interrupt interval can be determined. When data is transferred with asynchronous timing, such as with a host interface, the interrupt interval is not known until the operation starts, so the transfer is stopped at each interrupt and is resumed after the program has captured the data.

Figure 5-26. PMT Settings



5.8.3 PMT transfer steps

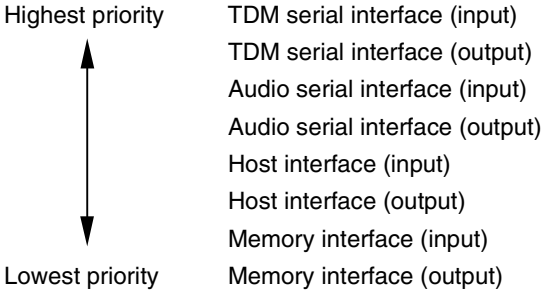
(1) The start address of the RAM area to be accessed is set to the PMSA register. Although this value can be changed during the PMT operation, the newly set value does not take effect until the next transfer operation starts.

(2) The buffer size is set to the PMS register. Although this value can be changed during the PMT operation, the newly set value does not take effect until the next transfer operation starts.

(3) When 1 is set to EN in the PMC register, PMSA and PMS are set to PMP and PMWC respectively to set the ready status.

(4) One word of data is transferred in response to a transfer request from a peripheral circuit. If transfer requests are sent from multiple channels, arbitration is performed.

The prioritization of transfer requests is shown below. Note, however, any channel request that is received before this is given the lowest priority.



Direct access is prohibited for any peripheral that is a PMT transfer target. For example, if the host register is accessed while the PMT is performing a host interface transfer, it is impossible to guarantee the prioritization between PMT access and direct access.

When the EN value is 0, an instruction to stop a PMT transfer stops the transfer regardless of how many words remain to be transferred. Basically, it is recommended to set ST to 1 to stop the transfer when the buffering is completed.

Four memory banks in internal RAM are used for PMT transfers: 0x0000 to 0x0FFF (4 Kwords), 0x1000 to 0x1FFF (4 Kwords), 0x2000 to 0x2FFF (4 Kwords), and 0x3000 to 0x37FF (2 Kwords). In each memory bank, if an access conflict occurs due to the DSP core kernel, a wait is inserted in the DSP core kernel operation and arbitration is performed.

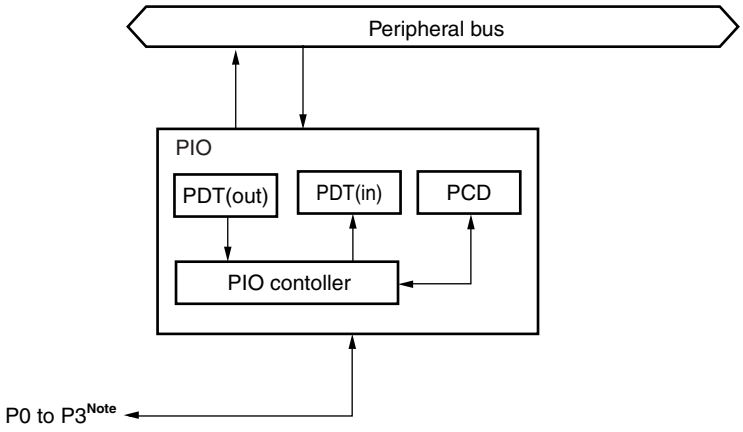
Accordingly, programming preventing bank access during DMA transfer enables data transfer without waits. Since X and Y memory are distinguished, if 0x0000 to 0xFFFF in Y memory is accessed by the DSP core kernel when 0x0000 to 0x0FFFF in X memory is used for a DMA transfer, conflict will not occur.

5.9 General-Purpose I/O Port (PIO)

The general-purpose I/O port is a 4-bit port used for input and output. In μPD77210 Family devices, four sets (P0 to P3, P4 to P7, P8 to P11, and P12 to P15) are included so that the entire group can be used as a 16-bit port. The following description uses P0 to P3 as an example, but the same configuration applies to the other general-purpose I/O ports as well.

Figure 5-27 shows a block diagram of a general-purpose I/O port and Table 5-22 lists its registers.

Figure 5-27. Block Diagram of General-Purpose I/O Port



Note P4 to P7, P8 to P11, and P12 to P15 have the same configuration.

Table 5-22. Registers of General-Purpose I/O Port

X/Y Memory Address	Register Name	Function	Load/Store
0x3870	PDT0	Port data register 0	L/S
0x3871	PCD0	Port command register 0	L/S
0x3872	PDT1	Port data register 1	L/S
0x3873	PCD1	Port command register 1	L/S
0x3874	PDT2	Port data register 2	L/S
0x3875	PCD2	Port command register 2	L/S
0x3876	PDT3	Port data register 3	L/S
0x3877	PCD3	Port command register 3	L/S

5.9.1 General-purpose I/O port pins

(1) P0 to P15 (general-purpose I/O port – I/O)

These are general-purpose I/O pins.

Since P8 to P15 are shared as host interface pins HD8 to HD15, they cannot be used as general-purpose I/O port pins when the host interface is in 16-bit bus mode.

5.9.2 General-purpose I/O port registers

(1) PDT0 to PDT3 (port data transfer registers)

These are 16-bit registers used for input and output of data via the general-purpose I/O port (only the lower four bits are valid). One register is allocated to each set of four general-purpose I/O port pins, and thus there are four registers (PDT0 to PDT3). PDT0 corresponds to pins P0 to P3, PDT1 to pins P4 to P7, PDT2 to pins P8 to P11, and PDT3 to pins P12 to P15. There are separate registers for output and input.

PDT(out) is a 16-bit register that is used to set the data to be output. When a store instruction to PDT is executed, data is input from the peripheral bus to PDT (out). If the general-purpose I/O port pins are set for output, the values of the lower four bits are the output values of the output pins, and each bit value of 1 indicates high-level output (0 indicates low-level output). The values of bits corresponding to pins set for input are invalid. The value after reset is 0x0.

PDT(in) is a 16-bit register that is used to read the input data. When a load instruction to PDT is executed, data is output to the peripheral bus. If the general-purpose I/O port pins are set for input, the values of the lower four bits are the input values of the input pins, and each bit value of 1 indicates high-level input (0 indicates low-level input). The values of bits corresponding to pins set for output are undefined.

(2) PCD0 to PCD3 (port command registers)

These are 16-bit registers used to specify the I/O direction as well as bit manipulation of the output pins. Table 5-23 lists the functions of the PCD bits. The value after reset is 0x0.

Table 5-23. Bit Configuration of PCD (1/2)

Bit	Name	Type	Bit's Function	Load/Store (L/S)
15	BE	Bit manipulation	Bit manipulation enable bit • 0: Do not manipulate bits • 1: Manipulate bits The type of manipulation is specified by B1, B0, and PSR. • Undefined during a read operation	S
14	PSR	Bit manipulation	Port set/reset specification bit • 0: Reset (low level) • 1: Set (high level) • The target port is specified by B1 and B0. • Valid when BE = 1 • Undefined during a read operation	S
13	ME	Mode setting	Mode setting enable bit • 0: Does not set mode • 1: Sets mode The mode setting is specified by IO and M3 to M0 • Undefined during a read operation	S

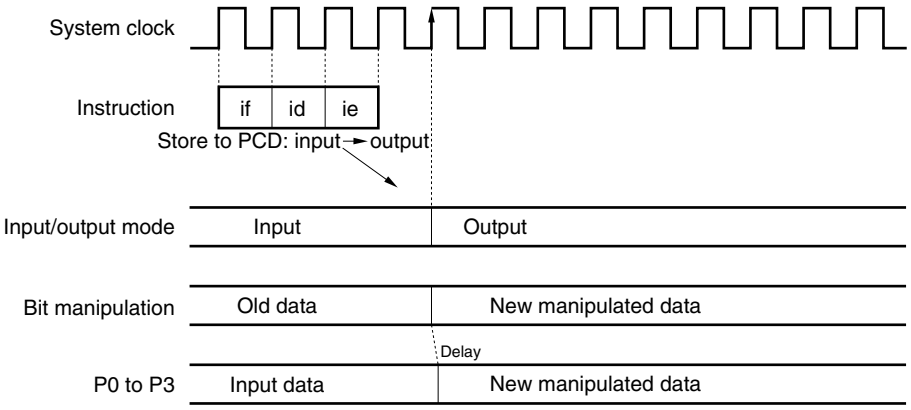
Table 5-23. Bit Configuration of PCD (2/2)

Bit	Name	Type	Bit's Function	Load/Store (L/S)
12	IO	Mode setting	I/O specification bit • 0: Specifies input port • 1: Specifies output port • The port setting is specified by M3 to M0. • Valid when ME = 1 • Undefined during a read operation	S
11 and 10	Reserved	–	Reserved bits • Values cannot be set to these bits. • Undefined during a read operation	–
9 and 8	B1 B0	Bit manipulation	These bits specify the bit manipulation port. • When B1, B0 = 00: P0 • When B1, B0 = 01: P1 • When B1, B0 = 10: P2 • When B1, B0 = 11: P3 • Set/reset is specified by PSR. • Valid when BE = 1 • Undefined during a read operation	S
7 to 4	Reserved	–	Reserved bits • Values cannot be set to these bits. • Undefined during a read operation	–
3 2 1 0	M3 M2 M1 M0	Mode setting	These bits specify the mode setting port. • When M3 = 0: P3 is not selected, = 1: P3 is selected • When M2 = 0: P2 is not selected, = 1: P2 is selected • When M1 = 0: P1 is not selected, = 1: P1 is selected • When M0 = 0: P0 is not selected, = 1: P0 is selected Each bit can be selected independently.	S
		Mode status	I/O mode status bits • When M3 = 0: P3 is input, = 1: P3 is output • When M2 = 0: P2 is input, = 1: P2 is output • When M1 = 0: P1 is input, = 1: P1 is output • When M0 = 0: P0 is input, = 1: P0 is output	L

5.9.3 Timing of general-purpose I/O port

The general-purpose I/O port is not assumed to be used synchronously, but is synchronized with the rising edge of CLKOUT during data input/output.

(1) Mode change from input to output



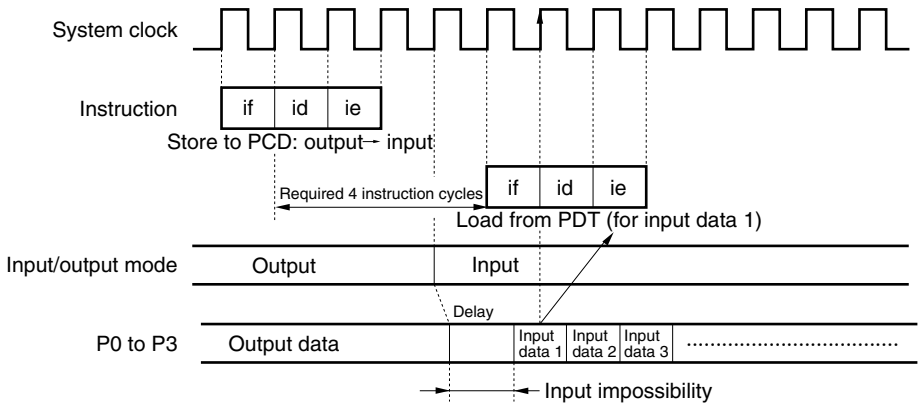
The mode of each pin is changed from input to output two system clocks after the execution cycle of the instruction that stores data to the PCD register.

Example program:

```
#define PCD = 0x3805
#define PDT = 0x3804
R1L = 0x0000        ;
*PDT:x = R1L        ; initialize PDT
R0L = 0x3001        ;
*PCD:x = R0L        ; P0 → output port
```

Caution Because the PDT register is undefined after hardware reset, write data to the PDT register before storing data to the PCD register.

(2) Mode change from output to input



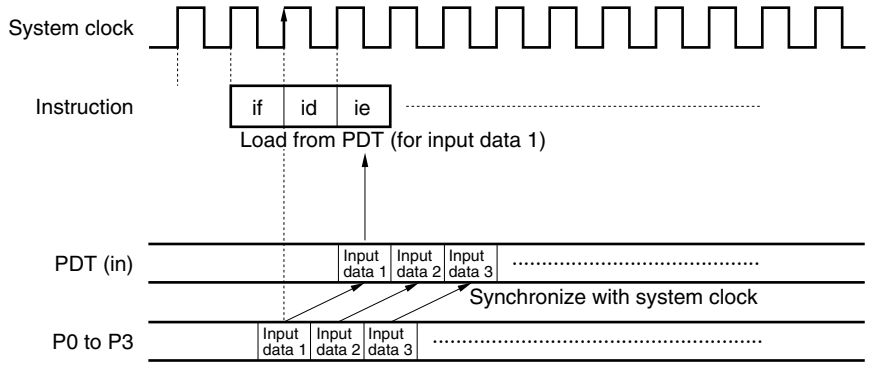
The mode of each pin changes from output to input after two system clocks since execution cycle of store to PCD register, but the μPD77210 Family inhibit the pin's data from being input during two system clocks after then. Therefore it is required that minimum 4 system clocks between store to PCD register and load from PDT register.

Example program:

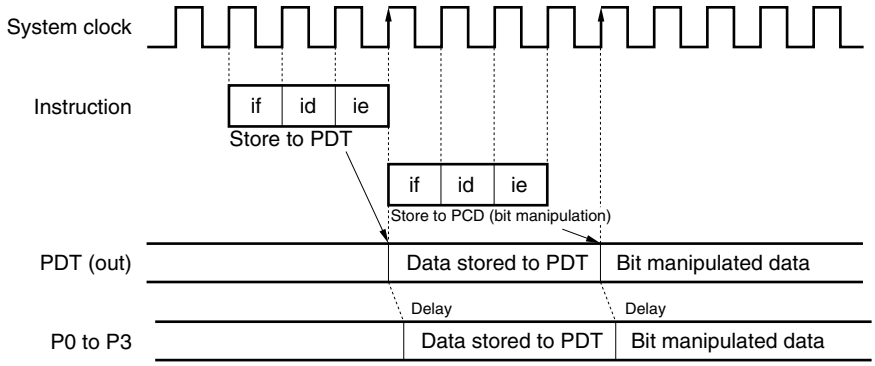
```
#define PCD = 0x3805
#define PDT = 0x3804
R0L = 0x200f          ;
*PCD:x = R0L          ; P0 to P3 output → input
<required minimum 4 system clocks between instructions>
R1L = *PDT:x          ; load from PDT
```

(3) Timing of input ports

The pin's input data is loaded after synchronized with the rising edge of two system clocks.



(4) Timing of output ports



(a) In case of store to PDT register

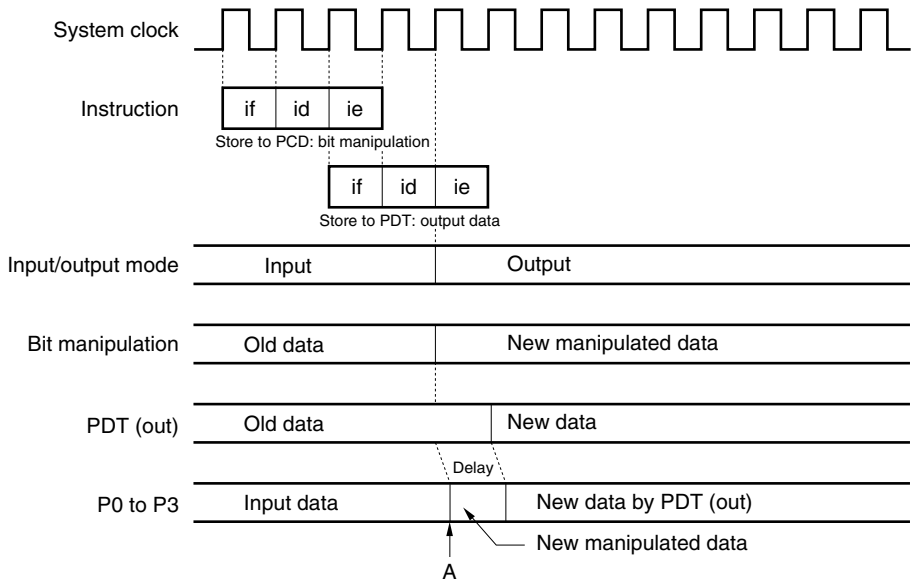
The output data is output after one system clock since execution cycle of store to PDT register.

(b) In case of store to PCD register

The 1-bit manipulated data is output after two system clocks since execution cycle of store to PCD register.

Caution If bit manipulation by the PCD register and data output by storing to the PDT register are executed at the same time, the data of the PDT register takes precedence over bit manipulation by the PCD register.

(5) Output port setting (by use of PCD and PDT registers)



The manipulated data is output after two system clocks since execution cycle of store to PCD register. Next, when the output data is output after one system clock since execution cycle of store to PDT register, spike noise may occur at point A. Therefore it is required that minimum 1 system clock between store to PCD register and store to PDT register.

Example program:

```
#define PCD = 0x3805
#define PDT = 0x3804
R0L = 0xf00f        ;
*PCD:x = R0L        ; P0 to P3 input → output, P0 → high
R1L = 0x0000        ; <required minimum 1 system clock between instructions>
*PDT:x = R1L        ; P0 to P3 → low
```

Cautions 1. If at least one system clock is not inserted between the instruction that stores data to the PCD register and the instruction that stores data to the PDT register, a spike may be generated at point A.

2. Because the value of the PDT register is undefined after hardware reset, set the PDT register before storing data to the PCD register.

5.9.4 Example of port programming

Here is an example of a program using the general-purpose input/output port. In this example, the following is executed:

- P0 and P1 are set in the output mode.
- P2 and P3 are set in the input mode.
- P0 outputs a low level, and P1 outputs a high level.

Example of programming general-purpose input/output port

```
#define PDT 0x3804
#define PCD 0x3805
#define All_In_mode 0x200F
#define P0_Out_mode 0x3001
#define P1_Out_mode 0x3002
#define Out_P0_Low 0x8000
#define Out_P1_High 0xC100

R0L = All_In_mode                ; P3 to P0 input pins
*PCD:x = R0L                     ;
R0L = P0_Out_mode+Out_P0_Low     ; P0 output pin (low level)
*PCD:x = R0L                     ;
R0L = P1_Out_mode+Out_P1_High    ; P1 output pin (high level)
*PCD:x = R0L;
```

5.10 Interrupt Controller (INTC)

Twelve interrupt ports are provided in the DSP core kernel.

The interrupt controller (INTC) expands these 12 interrupt ports by allocating four interrupt factors (sources) to each port to enable interrupt handling for up to 48 factors.

Table 5-24 lists the interrupt controller's registers.

Table 5-24. Interrupt Controller's Registers

X/Y Memory Address	Register Name	Function	Load/Store
0x3880	ICR0	Interrupt control register 0 (for vector address 0x210)	L/S
0x3881	ICR1	Interrupt control register 1 (for vector address 0x214)	L/S
0x3882	ICR2	Interrupt control register 2 (for vector address 0x218)	L/S
0x3883	ICR3	Interrupt control register 3 (for vector address 0x21C)	L/S
0x3884	ICR4	Interrupt control register 4 (for vector address 0x220)	L/S
0x3885	ICR5	Interrupt control register 5 (for vector address 0x224)	L/S
0x3886	ICR6	Interrupt control register 6 (for vector address 0x228)	L/S
0x3887	ICR7	Interrupt control register 7 (for vector address 0x22C)	L/S
0x3888	ICR8	Interrupt control register 8 (for vector address 0x230)	L/S
0x3889	ICR9	Interrupt control register 9 (for vector address 0x234)	L/S
0x388A	ICR10	Interrupt control register 10 (for vector address 0x238)	L/S
0x388B	ICR11	Interrupt control register 11 (for vector address 0x23C)	L/S

5.10.1 Interrupt controller's registers

(1) ICR0 to ICR11 (interrupt control registers)

These are 16-bit registers that are used to set the interrupt mode and interrupt masking. One of these registers exists for each of the DSP core kernel's interrupt ports (each interrupt vector address).

Table 5-25 lists the functions of each bit in ICR. The default value is 0x0.

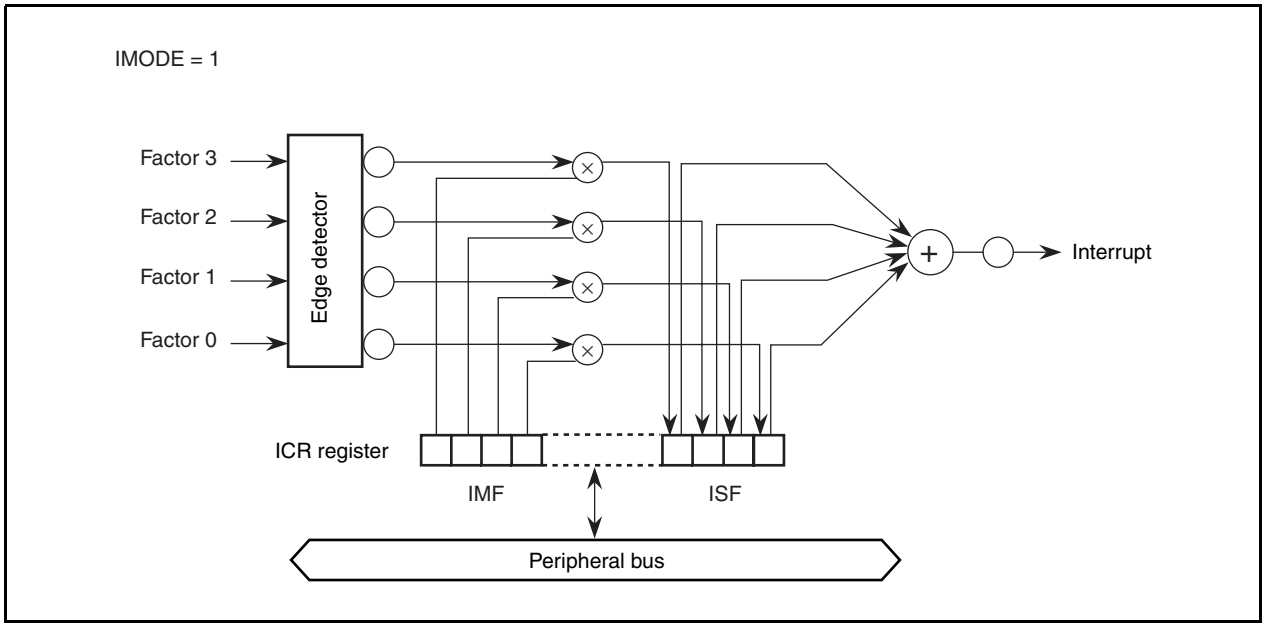
Table 5-25. Bit Configuration of ICR

Bit	Name	Bit's Function	Load/Store
15 to 9	Reserved	• Do not write any value other than zero. • Undefined during a read operation	–
8	IMODE	Interrupt mode setting bit • 0: Mask mode (default) • 1: Stamp mode	L/S
7 to 4	IMF	Interrupt mask flag • 0: Masks (default) • 1: Does not mask These set whether or not an interrupt that has been input to an interrupt port will be transferred to the DSP core kernel. Bits 7, 6, 5, and 4 correspond to Factors 3, 2, 1, and 0, respectively.	L/S
3 to 0	ISF	Interrupt stamp flag • 0: No interrupt (default) • 1: Interrupt When an interrupt is input, the corresponding bit's value is 1. When the ICR is read, the bit's value is 0. Bits 3, 2, and 1 correspond to Factors 3, 2, and 1, respectively.	L

5.10.2 Operation modes of interrupt controller

When in stamp mode (IMODE = 1), the DSP core kernel's interrupt ports are in factor expansion operation mode (see **Figure 5-28**).

Figure 5-28. Interrupt Controller in Stamp Mode



One ICR register set, which includes a 4-bit interrupt mask flag (IMF) and a 4-bit interrupt stamp flag (ISF), is provided for each of the DSP core kernel's interrupt ports, making 12 sets of registers in total. This expands into four factors for each of DSP core kernel's interrupt ports.

When the falling edges of the four interrupt signals corresponding to the four factors are detected, if they are not masked, a stamp signal is set to record the interrupt. The resulting interrupt signal for the DSP core kernel is output as the ORed result of the four stamp flags, and are branched to an interrupt vector via an interrupt input. The corresponding interrupt factor can be identified by referencing the ICR register with the interrupt vector.

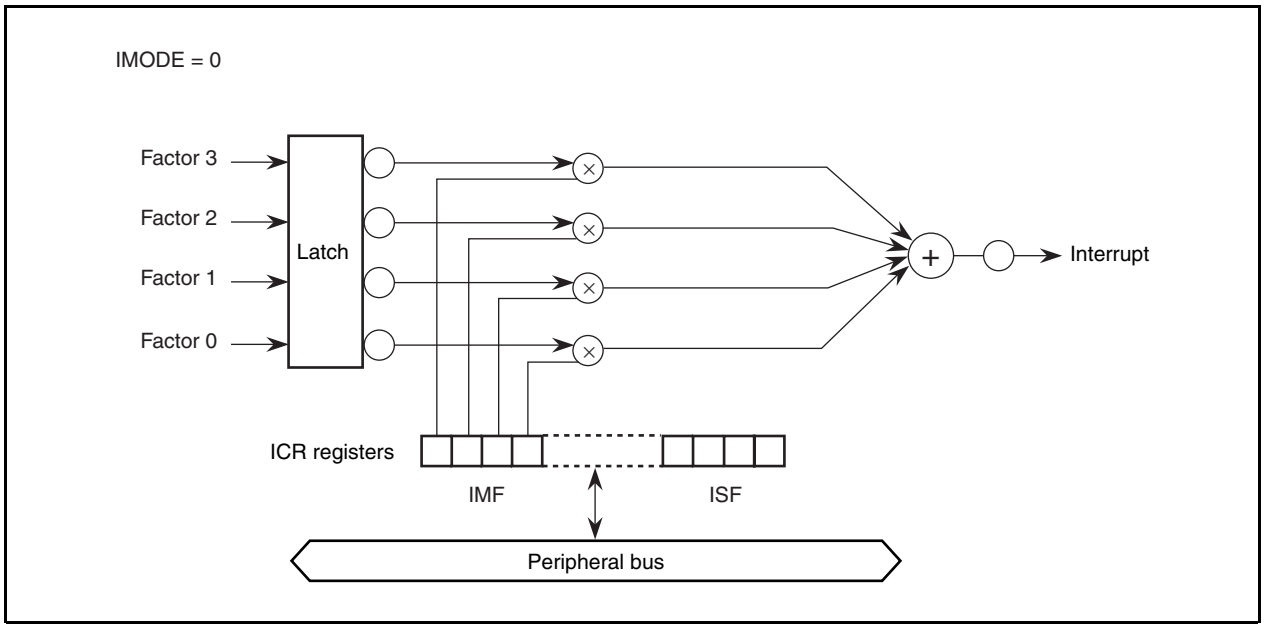
The stamp flag is reset when the ICR is loaded, and all interrupt signals to the DSP core kernel are forcibly masked for four cycles following each ICR load operation. In cases where and the stamp flag is set either simultaneously with or after the ICR load operation, the flag setting takes priority, and the interrupt occurs again after the load to the DSP core kernel has been executed.

As long as the ICR load processing does not occur within the interrupt vector, the interrupt signal to the DSP core kernel is held at low level, so subsequent interrupt signals are masked. Consequently, the ICR must be loaded.

Also, if more than one interrupt occurs for the same factor prior to the ICR load operation, it will be regarded as a single interrupt.

When in mask mode (IMODE = 0), the resulting operation is simply masking of four factors (see **Figure 5-29**).

Figure 5-29. Interrupt Controller in Mask Mode



For interrupt factors that have not been set as masked in the ICR, the ORed result is taken to issue an interrupt signal to the DSP core kernel. Selection of just one factor enables interrupt handling to maintain compatibility with μPD77111 Family devices. The ISF values are undefined in this mode.

5.10.3 Interrupt table

The interrupt table contains the interrupt vector start addresses for reset and the 12 DSP core kernel interrupts.

When a reset or interrupt is entered, processing jumps to the address of the vector corresponding to the relevant factor and a interrupt routine is executed.

A reset uses a 16-word vector area and an interrupt factor uses a four-word vector area. When processing greater lengths, an instruction to jump to the start of the programmed interrupt subroutine must be coded here.

Since external interrupts are assigned to DSP core kernel interrupt ports 0 to 3, the factors are expanded to 16 bits. The external interrupt pins are assigned to 4-bit PIO input signals so that the four sets of (i.e., 16) interrupts can be serviced.

The peripherals, TSIO, ASIO, and HIO are assigned to interrupt ports 4 to 9 according to their I/O direction. Factor 1 is assigned to a pin and unused peripherals can be used as external interrupt pins.

Table 5-26. Interrupt Table

Interrupt Number	Vector Address	Interrupt Factor (Source)			
		0	1	2	3
RESET	0x200	Reset	Reserved	Reserved	Reserved
0	0x210	$\overline{\text{INT00}}$	$\overline{\text{INT01}}$	$\overline{\text{INT02}}$	$\overline{\text{INT03}}$
1	0x214	$\overline{\text{INT10}}$	$\overline{\text{INT11}}$	$\overline{\text{INT12}}$	$\overline{\text{INT13}}$
2	0x218	$\overline{\text{INT20}}$	$\overline{\text{INT21}}$	$\overline{\text{INT22}}$	$\overline{\text{INT23}}$
3	0x21C	$\overline{\text{INT30}}$	$\overline{\text{INT31}}$	$\overline{\text{INT32}}$	$\overline{\text{INT33}}$
4	0x220	TSI input	TSIEN	PMT ch0	SDCR input[Note]
5	0x224	TSO output	TSOEN	PMT ch1	SDCR output[Note] (clear busy)
6	0x228	ASI input	ASIEN	PMT ch2	SDDAT input[Note]
7	0x22C	ASO output	ASOEN	PMT ch3	SDDAT output[Note]
8	0x230	HI input	$\overline{\text{HWR}}$	PMT ch4	Reserved
9	0x234	HO output	$\overline{\text{HRD}}$	PMT ch5	Reserved
10	0x238	TIMER ch0	TIMER ch1	PMT ch6	Reserved
11	0x23C	TIMER ch1	TIMER ch0	PMT ch7	Reserved

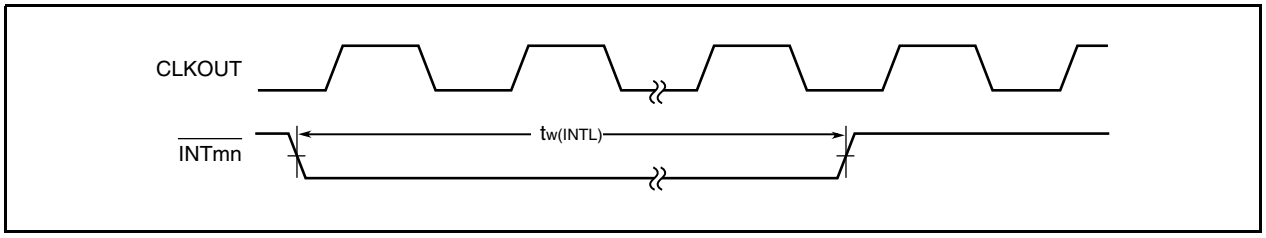
Note This applies to the μPD77213 only. For the μPD77210, all are reserved.

5.10.4 Hardware conditions during interrupt from external interrupt pins

External interrupts ($\overline{\text{INTmn}}$) are detected and acknowledged at the falling edge of various signal pins. Consequently, to execute a series of interrupts, the pin state should be returned to high level each time and then to low level afterward so that a falling edge is provided. However, time is required to detect both the high level and low level.

The respective external interrupt timing is shown in Figure 5-30.

Figure 5-30. External Interrupt Timing



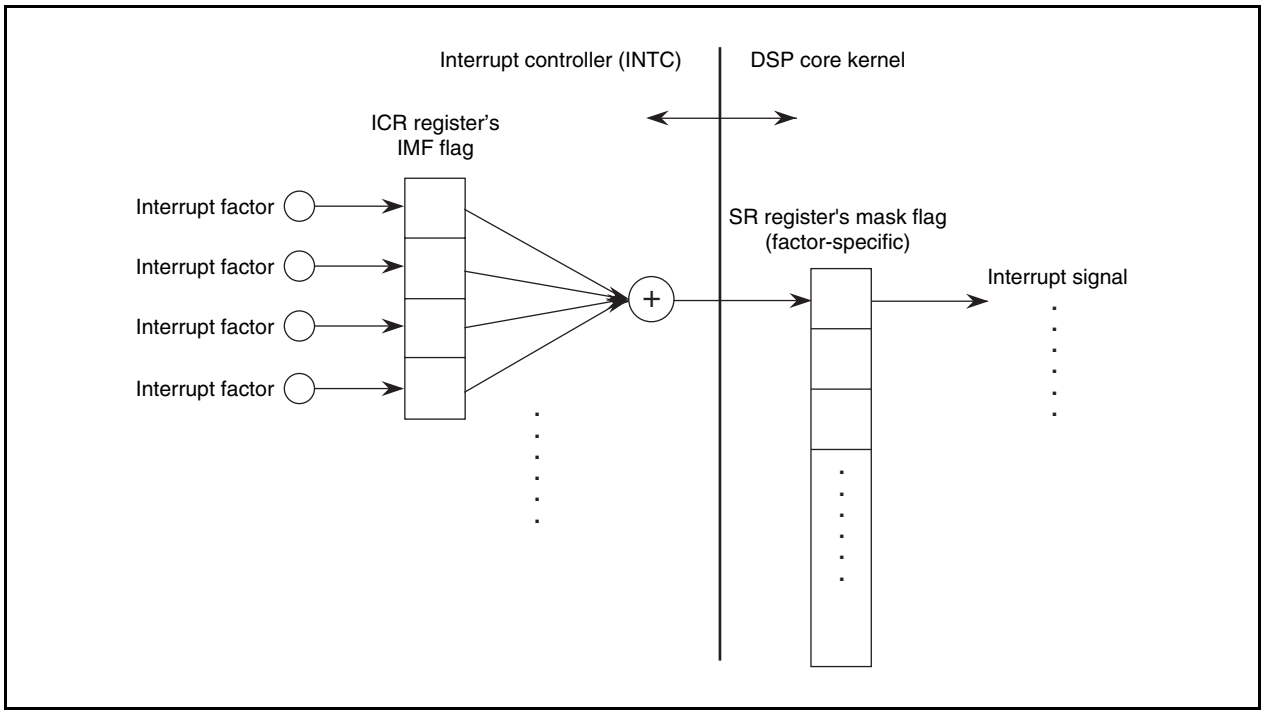
Caution The active period of an external pin's interrupt ($\overline{\text{INTmn}}$) is defined by the system clock. If an operation is performed according to a divided clock, such as in halt mode, the divided clock becomes the system clock.

5.10.5 Interrupt-related precautions

(1) SR register and ICR register

There are two types of interrupt mask flags. The relationship between the two types is shown in Figure 5-31.

Figure 5-31. Relationship Between Two Types of Mask Flags



The SR register, which is within the DSP core kernel, is used for mask control of factor-specific interrupts. Any interrupt input from outside the DSP core kernel (via the interrupt controller, which is a peripheral) is recorded as a single interrupt request, regardless of whether or not it is masked. In such cases, the recorded interrupt is executed when it is no longer masked.

The ICR register is within the interrupt controller, which is a peripheral, and the IMF flags in the ICR register are used for mask control of interrupt factors that are expanded by the interrupt controller. Masked interrupt factors are not input to the DSP core kernel. As a result, any interrupt that has been masked by an IMF flag is not recorded.

(2) fint instruction

If an interrupt is input while global or specific interrupts are disabled, there is no actual branching to interrupt servicing since interrupts are disabled, but each interrupt factor is nevertheless recorded as an interrupt by the DSP core kernel. These recorded and retained interrupts become valid at the time when interrupts are again enabled, at which time interrupt servicing is performed.

The fint instruction clears interrupt request signals that have been retained in the DSP core kernel. It is therefore used to discard retained interrupt requests that are not needed.

However, be sure to note the following concerning use of the fint instruction.

- The interrupt signals for peripherals are generated based on access enable signals for the peripherals.
- These access enable signals cannot be cleared by the fint instruction.

Consequently, when the fint instruction is executed for retained interrupts, no processing is acknowledged by the peripheral that issued the interrupt request, which means that the following kinds of standby states occur on both sides.

- DSP core kernel side:
  The DSP core kernel waits for an interrupt request (since the interrupt request has been cleared, there is no interrupt to be serviced, and the DSP core kernel is left waiting for the next interrupt signal).

- Peripheral side:
  The peripheral waits for servicing of an interrupt (since the interrupt request that was issued has not yet been serviced, it does not issue the next interrupt).

In such cases, this stalemate can be overcome by clearing the peripheral's access enable flag. To clear this flag, execute dummy access to the peripheral. Figure 5-32 shows an example that uses an input interrupt for the audio serial interface. The operation is similar for other peripherals as well. Use a dummy output or dummy input to clear the enable flag for the function that matches the factor being used for the interrupt. If the peripheral does not have an enable flag in its peripheral status register, this is not a problem.

Figure 5-32. fint Instruction Use Example: When Using an Audio Serial Input Interrupt While in 32-Bit Mode

```
;Interrupt disabled
 :
fint                        ;
clr (r0)                    ;
r0l = *ASST:x               ;
r0 = r0 & 0x0001            ; check "ASDT Load Enable Flag"
if (r0 == 0) jmp $+3        ;
r0l = *ASDT:x               ; dummy access to clear "ASDT Load Enable Flag"
r0l = *ASDT:x               ; dummy access to clear "ASDT Load Enable Flag"
any instruction             ;
 :
;Interrupt enabled
```

Note that since the objective of the fint instruction is to discard unneeded interrupts, this instruction should be executed during interrupt disabled mode.

(3) Sequence of setting interrupt enabled and interrupt disabled modes

Enabled/disabled status can be set for each interrupt factor via the SR register and ICR register (see **5.10.5 (1) SR register and ICR register**). Ultimately, both must be set to interrupt enable in order to enable interrupts, and the following describes the sequence of these settings.

As mentioned earlier, even if interrupt factors are masked (disabled) in the SR register, when an interrupt is input, it is recorded. Thus, if an unneeded interrupt has been recorded, the interrupt will be serviced when interrupts have been enabled. By contrast, an interrupt factor mask in the ICR register masks the interrupt's original signal so the interrupt is never input to the DSP core kernel.

Therefore, be sure to follow the sequence shown below to discard unneeded interrupts.

- Enable SR followed by ICR
- Disable ICR followed by SR

Of course, there is no need to set interrupt disable mode via both SR and ICR in cases where the objective is to temporarily disable interrupts. In such cases, setting the SR register is sufficient.

(4) Interrupt after a reboot

Note that an interrupt may occur after rebooting, depending on the reboot method. Note that interrupts must be disabled before rebooting, and operation cannot be guaranteed if a reboot is executed in interrupt enabled mode.

- Host reboot method
- Serial reboot method

After a reboot has been executed via either of these methods, either a host input interrupt or a serial input interrupt will occur once the interrupt mode is changed to interrupt enabled mode, depending on the reboot method.

This is because an interrupt signal is generated for host or serial input access regardless of whether or not an interrupt has been used, since one interrupt occurrence has been held pending.

Therefore, when using a host or serial input interrupt, after completing a reboot be sure to either execute a fint instruction or a dummy input (refer to **5.10.5 (2) fint instruction**).

During a boot operation, the ICR register is masked so there is no problem. Even during a reboot operation, if the corresponding interrupt has been masked in the ICR register, there will not be any problem.

(5) Difference between operation in mask mode and stamp mode during a reset

Note that the operation of any interrupt that occurs between when a reset is cleared and the interrupt controller's interrupt mask flag is cleared differs depending on whether the mode is mask mode or stamp mode.

The DSP core kernel recognizes interrupts by detecting the falling edge of the interrupt factor. Accordingly, such a falling edge is required for each interrupt. When the ICR register's mask flag is masked, the interrupt controller outputs a high level. Therefore, any interrupt that occurs between when a reset is cleared and the mask flag is cleared operates differently as described below, depending on whether the mode is mask mode or stamp mode.

- Mask mode

  During mask mode, an interrupt signal that is input to the interrupt controller while the mask flag is cleared is input as is to the DSP core kernel. If the interrupt mask flag is cleared while in mask mode, the interrupt controller outputs a low level at the time when the mask flag is cleared in order to retain low level for interrupts[Note] from an internal peripheral. Therefore, the internal peripheral's interrupt[Note] is issued to the DSP core kernel. As a result, when an interrupt is used for the corresponding peripheral, there is no need for a dummy access. Note that this mode differs from that in μPD77111 Family devices.

- Stamp mode

  In stamp mode, when the falling edge of the interrupt signal input to the interrupt controller is detected while the mask flag is cleared, the interrupt stamp flag is set.

  If the interrupt mask flag is cleared in stamp mode, the falling edge of an interrupt from an internal peripheral[Note] cannot be detected, and so the interrupt stamp flag cannot be set. As a result, the interrupt from an internal peripheral[Note] is not issued to the DSP core kernel. Therefore, when using an interrupt for such a peripheral, a dummy access is required prior to the first interrupt. This mode is the same in μPD77111 Family devices.

Note The "interrupt from an internal peripheral" described above is output via either a serial interface or a host interface. The same is true for interrupt pins (such as port pins used to set the boot mode) that have been fixed at low level.

5.11 Timer (TIM)

Two 16-bit resolution timer channels are included. These timers can be used as interval timers, event counters, free-running timers, or watchdog timers.

Select among five types of source clocks, with eight prescaler functions.

The two timer channels can also be combined to implement a 32-bit timer.

The main timer features are as follows.

- Timer sources

  Select from five types: system clock, external interrupts (two sources), serial clock, or time-up signal (other timer output).

- Source prescaler

  Prescaler input can be divided by 1, 2, 4, 8, 16, 32, 64, or 128.

- Repeat count

  After a one-time operation or count incrementation, the initial value can be loaded and the count started again as a repeat count operation.

- Count modification function

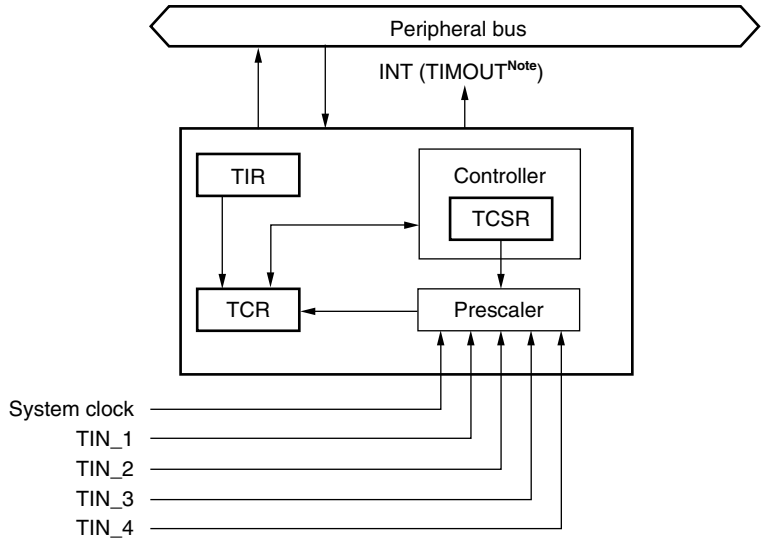
  Count data can be overwritten during a count operation (this function can be used to implement a watchdog timer by setting an initial value prior to a time incrementation).

- Interrupt generation function

  Time-up signals can be generated as interrupt signals.

Figure 5-33 shows a block diagram of the timer and Table 5-27 lists its registers.

Figure 5-33. Block Diagram of Timer (1 Channel)



Note TIMOUT pin exists only for ch0.

Table 5-27. Timer Registers

X/Y Memory Address	Register Name	Function	Load/Store
0x3890	TIR0	Timer initial register 0	L/S
0x3891	TCR0	Timer count register 0	L/S
0x3892	TCSR0	Timer control register 0	L/S
0x3894	TIR1	Timer initial register 1	L/S
0x3895	TCR1	Timer count register 1	L/S
0x3896	TCSR1	Timer control register 1	L/S

5.11.1 Timer pins

(1) TIMOUT (time-up output - output)

This pin is used to indicate when the timer's time-up status occurs. It exists only for timer ch0.

When the TCR register reaches zero (the time-up point), this pin outputs a high level for four cycles.

5.11.2 Timer registers

(1) TIR0 and TIR1 (timer initial registers)

These are 16-bit registers that are used to set the timer's initial value. When the timer is started, the initial value is transferred to the TCR register. The TIR value does not change during the count operation. The default value is 0xFFFF.

(2) TCR0 and TCR1 (timer count registers)

These are 16-bit registers that are used to retain the current timer count value. This value is decremented during a timer count operation. The default value is 0xFFFF. When using the timer as a watchdog timer, be sure to store a value matching the TIR value before reaching time-up.

(3) TCSR0 and TCSR1 (timer control/status registers)

These are 16-bit registers that are used to set timer operations and display the timer status. The default value is 0x0000. Table 5-28 describes the functions of the TCSR bits.

Table 5-28. Bit Configuration of TCSR

Bit	Name	Function	Load/Store
15 to 8	Reserved	Values other than 0 must not be written. • Undefined during a read operation	–
7	TEN	Timer enable flag • 0: Disable (default) • 1: Enable 1 starts count, 0 stops count.	L/S
6	TFNC	Timer function setting bit • 0: Repeat (default) • 1: One time Sets repeat operation of timer. Specifies one-time or repeat mode.	L/S
5 to 3	TCLKSEL	Timer clock source select bit • 000: System clock (default) • 001: TIN_1 • 011: TIN_2 • 011: TIN_3 • 100: TIN_4 • 101 to 111: Reserved	L/S
2 to 0	TCLKPS	Timer clock prescaler select bit • 000: 1/1 (default) • 001: 1/2 • 010: 1/4 • 011: 1/8 • 100: 1/16 • 101: 1/32 • 110: 1/64 • 111: 1/128 This sets the timer clock source's division rate.	L/S

Cautions 1. Do not switch the clock source or the prescaler setting while updating TEN or TFNC. Make sure that the prescaler setting is TEN = 0 before switching the clock source.

2. A count error may occur due to a hazard signal when switching the clock source, so be sure to set the clock source before starting the timer.

Table 5-29. Timer Clock Sources (TCSR Bits 5 to 3)

Bits 5 to 3	Clock Source	ch0	ch1
0 0 0	System clock	System clock	System clock
0 0 1	TIN_1	Serial clock for TSIO	Serial clock for ASIO
0 1 0	TIN_2	P0 pin	P2 pin
0 1 1	TIN_3	P1 pin	P3 pin
1 0 0	TIN_4	TIMOUT for ch1	TIMOUT for ch0

5.11.3 Operation of timer

To start the timer, store the count's initial value (cycle) in the TIR register and the count source and prescaler parameters to the TCSR register. Next, store the value of bits 7 and 6 again in TCSR to enable the timer operation. Once the timer operation has been enabled, TIR data is transferred to TCR. The TCR value is successively decremented in accordance with the count source.

The timer's operation ends based on a judgment made by monitoring the enable signal or the occurrence of an interrupt. When the TCR value reaches zero, TIMOUT outputs a high level for four cycles or, when in one-time mode, the enable signal becomes inactive while the TCR value is decremented again.

5.11.4 Precautions on use of timer

When the system clock is being switched, a regulating clock is inserted. For description of this regulating clock, see **5.12 Clock Controller (CLKC)**.

While the regulating clock is inserted into the timer operation, it may not be possible to count (sample) the timer's clock source. This can happen when switching the PLL or divider or during a mode change such as to halt mode. Although the system clock updates the timer's count, this happens because the timer operation cannot be sampled while the regulating clock is being inserted. The number of timer inputs that are not counted varies depending on the clock cycles before and after the timer is switched, the multiplication rate or division rate, and the timer's clock source cycle.

When the prescaler value is 1/1, the count occurs at the falling edge of the timer's clock source signal. When the prescaler's value is other than 1/1, the count occurs at the rising edge of the timer's clock source signal.

5.12 Clock Controller (CLKC)

The clock controller controls clock signals input from an external clock input (CLKIN). It includes a PLL and an output divider. The PLL is used for frequency multiplication and the output divider for frequency division in order to generate the system clock.

The multiplication rate is set by external pins PLL0 to PLL3.

The CLKC register is used to perform power on/off control for the PLL, clock source selection (external clock, PLL clock, divided and non-divided output), reset control and division rate setting of the divider used for output, and CLKOUT enable control. The user can utilize this register to control clock operations such as the external clock's operation, the PLL multiplier's operation, the divider's operation, and STOP.

When in standby mode (HALT/STOP mode), the clock source is automatically selected for divider circuit output. The divider setting must be made before entering standby mode. The clock cannot be changed while in HALT mode if the divider operation is already being executed in the program.

When in STOP mode, the system clock is masked. The PLL is never set to OFF automatically. This is because there is no need for any lockup time for the PLL, and startup can be achieved quickly from STOP mode. In such cases, the current consumption is the current used for PLL operation. Also, during this time the clock supplied to the CLKIN pin cannot be stopped. To reduce the current consumption, the PLL must be stopped by the user program. In this case, the clock supplied to the CLKIN pin can be stopped.

A reset sets the PLL to standby mode and an external clock is then supplied as the direct system clock. Therefore, if the PLL multiplier or divider circuit output is being used as the clock source, a reset can cause a hazard. If a reset occurs while the PLL multiplier or divider output is being used as the clock source, there is no guarantee that the contents of internal memory or the internal registers will be retained.

Figure 5-34 shows a block diagram of the clock controller and Table 5-30 lists its registers.

Figure 5-34. Block Diagram of Clock Controller

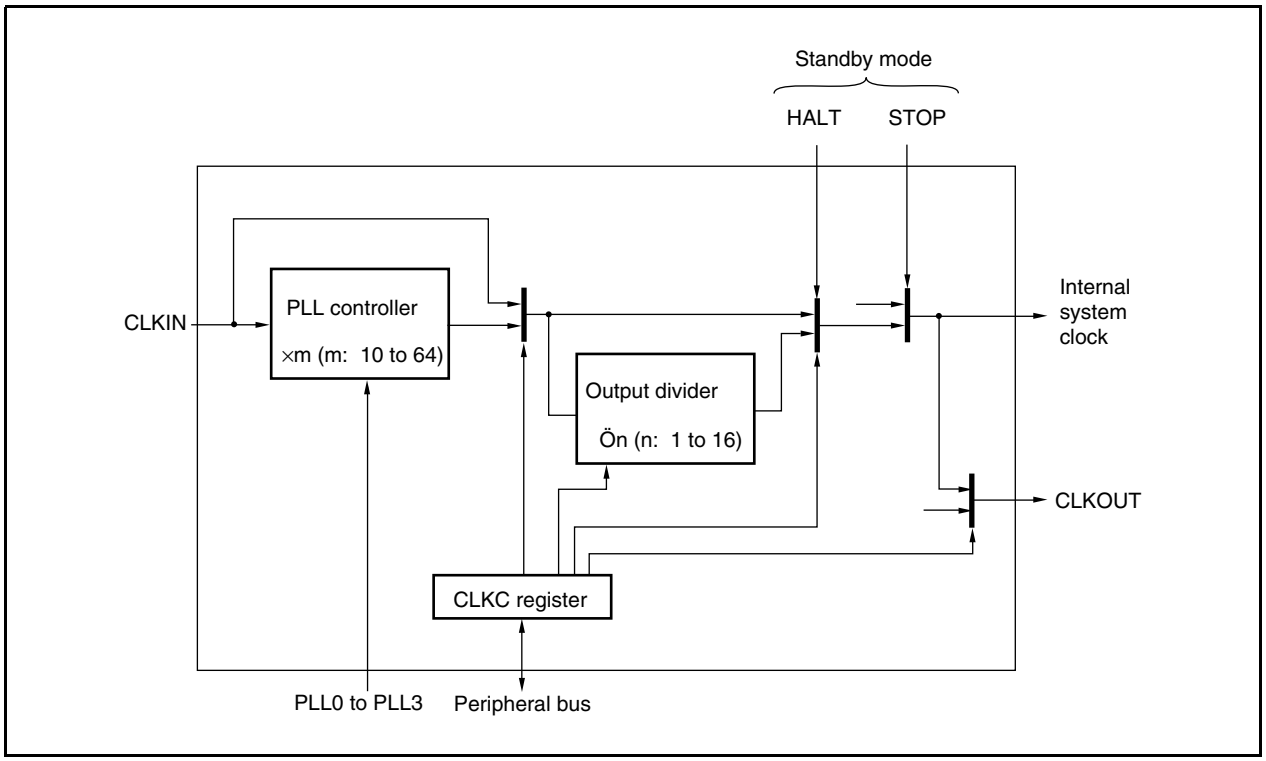


Table 5-30. Clock Controller's Register

X/Y Memory Address	Register Name	Function	Load/Store
0x38B0	CLKC	Clock control register	L/S

5.12.1 Clock controller's register

(1) CLKC (clock control register)

This register is used to set parameters for the PLL and divider. It sets the PLL's on/off mode, and selects the clock source, the divider's division rate, and the CLKOUT pin setting.

Table 5-31 describes the functions of the bits in CLKC. A reset sets 0x0, but this is changed by a boot operation. PLLENA and PLLSEL are changed when ODIVENA = 1, ODIV = 0001, or according to the boot mode.

Table 5-31. Bit Configuration of CLKC (1/2)

Bit	Name	Function	Load/Store
15 to 9	Reserved	Values other than 0 cannot be written. Undefined during a read operation	–
8	PLLENA	PLL enable bit • 0: Standby (default) • 1: Operates multiplier This bit is used to switch the PLL's operation. It can stop and start the PLL.	L/S
7	PLLSEL	PLL clock select bit • 0: Direct from external clock pin (default) • 1: PLL multiplier clock output This register bit is used to switch the clock supplied within the system.	L/S
6	CKOEN	CLKOUT enable bit • 0: Fixes CLKOUT pin at low level output (default) • 1: Outputs internal system clock from CLKOUT pin This bit is used to select whether to output or stop output of a system clock from the CLKOUT pin.	L/S
5	ODIVENA	Divider enable bit • 0: Standby (default) • 1: Operates divider This bit is used to switch the divider's operation mode. It is used to stop and start the divider.	L/S
4	ODIVSEL	Divider clock select bit • 0: Non-divided clock (default) • 1: Divided clock This register bit is used to select the clock that is supplied within the system. 0 selects an external clock or PLL clock and 1 selects a divided clock.	L/S

Table 5-31. Bit Configuration of CLKC (2/2)

Bit	Name	Function	Load/Store
3 to 0	ODIV	Division rate setting bit • 0000: 1/16 (default) • 0001: 1/1 • 0010: 1/2 • 0011: 1/3 • 0100: 1/4 • 0101: 1/5 • 0110: 1/6 • 0111: 1/7 • 1000: 1/8 • 1001: 1/9 • 1010: 1/10 • 1011: 1/11 • 1100: 1/12 • 1101: 1/13 • 1110: 1/14 • 1111: 1/15 This register bit is used to specify the divider's division rate.	L/S

Although the clock source can be changed by the program, the following caution points should be noted in such cases.

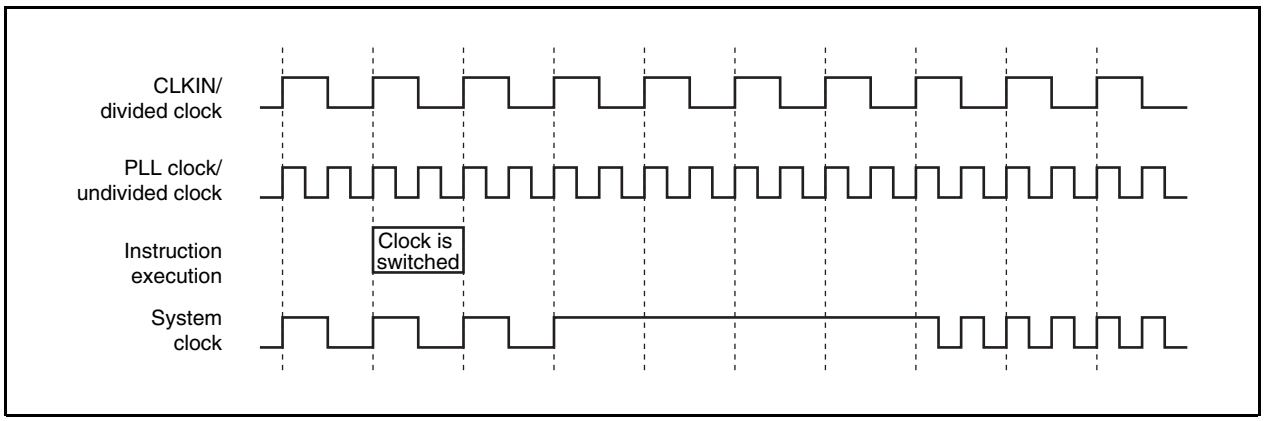
- Do not select a stopped divided clock or unlocked PLL clock as the source.
- For STOP/HALT, the divided clock is unconditionally selected as the source, so be sure to set the divider before executing STOP or HALT.
- After changing the clock source, do not stop the previous clock source for at least 16 instruction cycles. When repeating clock selection processing, make sure there are at least 16 instruction cycles between repetitions.
- Always set the divider to stop (standby) mode before changing the division rate.

5.12.2 Timing of clock switching

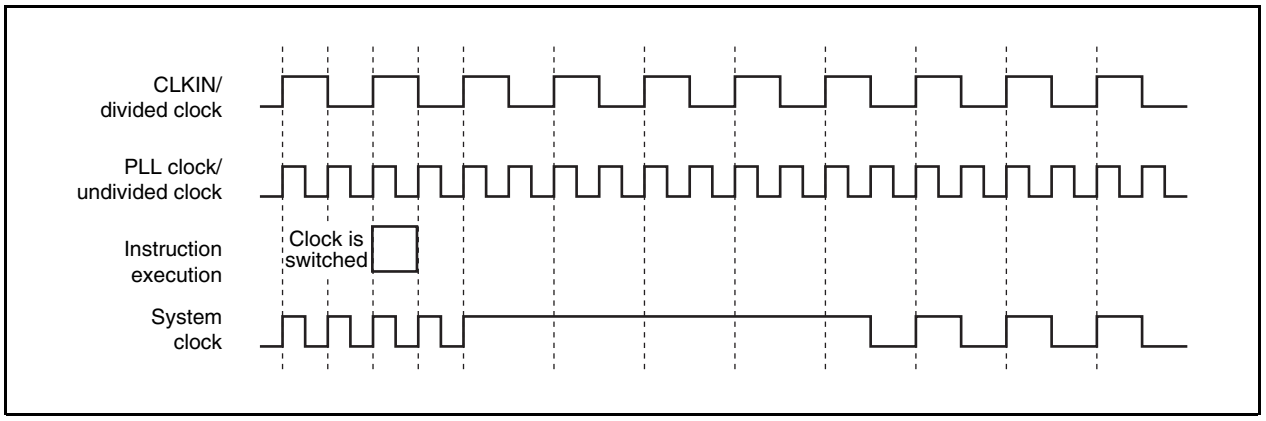
When the system clock is switched, a regulating clock that has a longer high level width than before or after switching is inserted. Figure 5-35 illustrates this.

Figure 5-35. Timing of Clock Switching

(a) External clock → PLL multiplier clock (divided clock → undivided clock)



(b) PLL multiplier clock → external clock (undivided clock → divided clock)



The regulating clock's high level width is as follows.

- When clock is switched from external clock to PLL multiplier clock (or from divided clock to undivided clock):
  (2 × division rate or multiplication rate + 4.5) × undivided clock cycle or PLL multiplied clock cycle
- When clock is switched from PLL multiplier clock to external clock (or from undivided clock to divided clock):
  (8 × division rate or multiplication rate + 1) × undivided clock cycle or PLL multiplied clock cycle

During this period, sampling errors may occur due to the lengthened system clock, so the timer and serial interface should be stopped before switching clocks.

Caution If the specifications, such as for the serial clock, stipulate required values for the system clock, the regulating clock must meet this requirement during the time it is being used as the system clock. For example, if the required value for the serial clock (standard serial interface) is "2 × system clock (Min.)", when using the serial interface, there must be two cycles of system clock (including the regulatory clock being used as the system clock) before the clock is changed.

5.12.3 Precaution points on clock control

The following describes clock control-related caution points and likely errors.

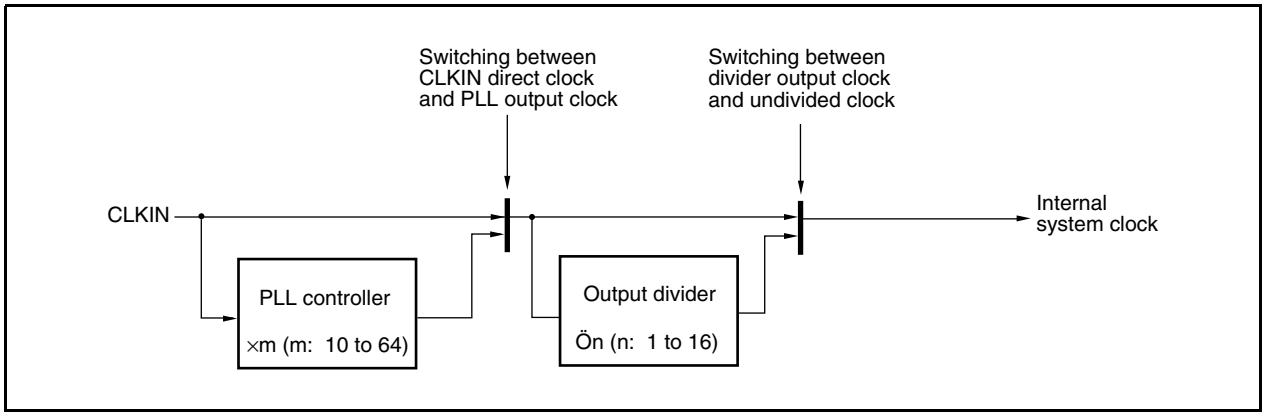
(1) Clock switching

The clock signal input from the CLKIN pin can be supplied as the system clock, either directly or via the PLL circuit or the divider. Although the CLKIN direct clock is used as the system clock when reset is active at startup, the boot process in this case is as follows.

- PLL output clock (except for external memory boot or non-boot)
- CLKIN direct clock (external memory boot or non-boot)
- Divider: Divider operation (1/1), divider output clock is not selected

Figure 5-36 shows the relationship when the clock input from the CLKIN pin is supplied as the system clock. For description of the PLL multiplier settings, CLKOUT pin settings, and the clock during standby mode, see **4.3.1 Clock generator**.

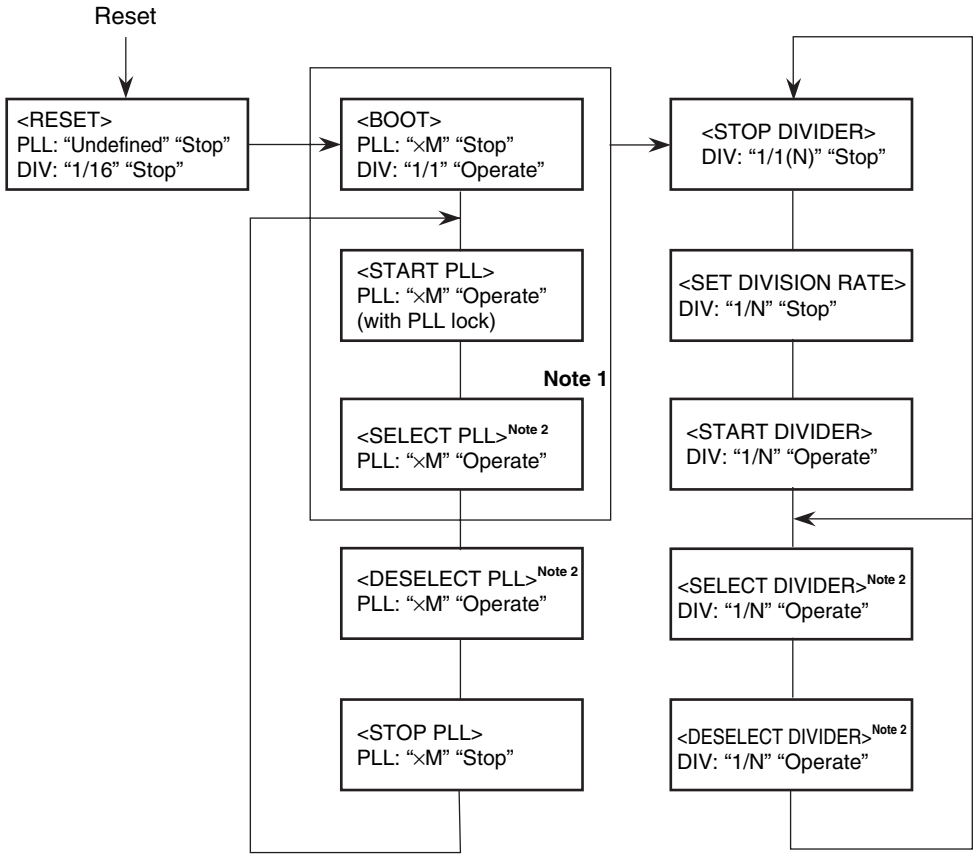
Figure 5-36. Clock Switching



The switching of the clock source (CLKIN direct, PLL output, or divider output) is performed via the CLKC register. Note with caution that at least 16 instruction cycles are required after switching the settings for the PLL and divider circuits. Also, since the clock (regulating clock) that is inserted during switching has a longer width than usual, sampling errors may occur in peripherals that use the system clock for their sampling operations.

Figure 5-37 shows the state transition and Figures 5-38 and 5-39 show an example of settings for the PLL and divider.

Figure 5-37. State Transitions During PLL Operations and Divider Operations



Notes 1. Boot is progressed as far as selecting the PLL during a boot operation other than external memory boot or non-boot.

2. Be sure to insert at least 16 operation cycles between select and deselect.

Remark M: Multiplication rate set by PLL0 to PLL3, N: Division rate

Figure 5-38. Example of PLL Settings

```
Start PLL circuit
clr (r0)                    ;
r0l = *CLKC:x               ;
r0 = r0 | 0x0100            ;
*CLKC:x = r0l               ;

Select PLL output clock (PLL circuit must have been started and PLL must be in lock mode)
clr (r0)                    ;
r0l = *CLKC:x               ;
r0 = r0 | 0x0180            ;
*CLKC:x = r0l               ;

Deselect PLL output clock
clr (r0)                    ;
r0l = *CLKC:x               ;
r0 = r0 & 0x017f            ;
*CLKC:x = r0l               ;

Stop PLL circuit (PLL output lock must be deselected)
clr (r0)                    ;
r0l = *CLKC:x               ;
r0 = r0 & 0x00ff            ;
*CLKC:x = r0l               ;
```

Figure 5-39. Example of Divider Settings

```
Division rate settings (divider must be stopped)
clr (r0)                    ;
r0l = *CLKC:x               ;
r0 = r0 & 0x01f0            ;
*CLKC:x = r0l               ;
r0 = r0 | 0x000z            ; z = divide ratio
*CLKC:x = r0l               ;

Start divider
clr (r0)                    ;
r0l = *CLKC:x               ;
r0 = r0 | 0x0020            ;
*CLKC:x = r0l               ;

Select divider's output clock (divider must be started)
clr (r0)                    ;
r0l = *CLKC:x               ;
r0 = r0 | 0x0010            ;
*CLKC:x = r0l               ;

Deselect divider's output clock
clr (r0)                    ;
r0l = *CLKC:x               ;
r0 = r0 & 0x01ef            ;
*CLKC:x = r0l               ;

Stop divider (divider's output clock must be deselected)
clr (r0)                    ;
r0l = *CLKC:x               ;
r0 = r0 & 0x01df            ;
*CLKC:x = r0l               ;
```

(2) Clock during standby mode

When in standby mode (HALT/STOP), the divider's output clock is selected automatically. Therefore, be sure to start the divider before using standby mode.

When using HALT mode while the divider output clock is operating, the divider output clock will remain as it is, the clock is not switched.

When entering STOP Mode, the clock to the system is also stopped. The PLL remains in operation, however. To reduce power consumption, be sure to stop the PLL before entering STOP mode.

(3) Clock monitor output

Monitor output for the internal system clock can be executed via the CLKOUT pin.

Figure 5-40 shows the state transition and Figure 5-41 shows an example of a CLKOUT output setting. When switching, a hazard may occur in the output clock from the CLKOUT pin.

Figure 5-40. State Transition of Clock Monitor Output

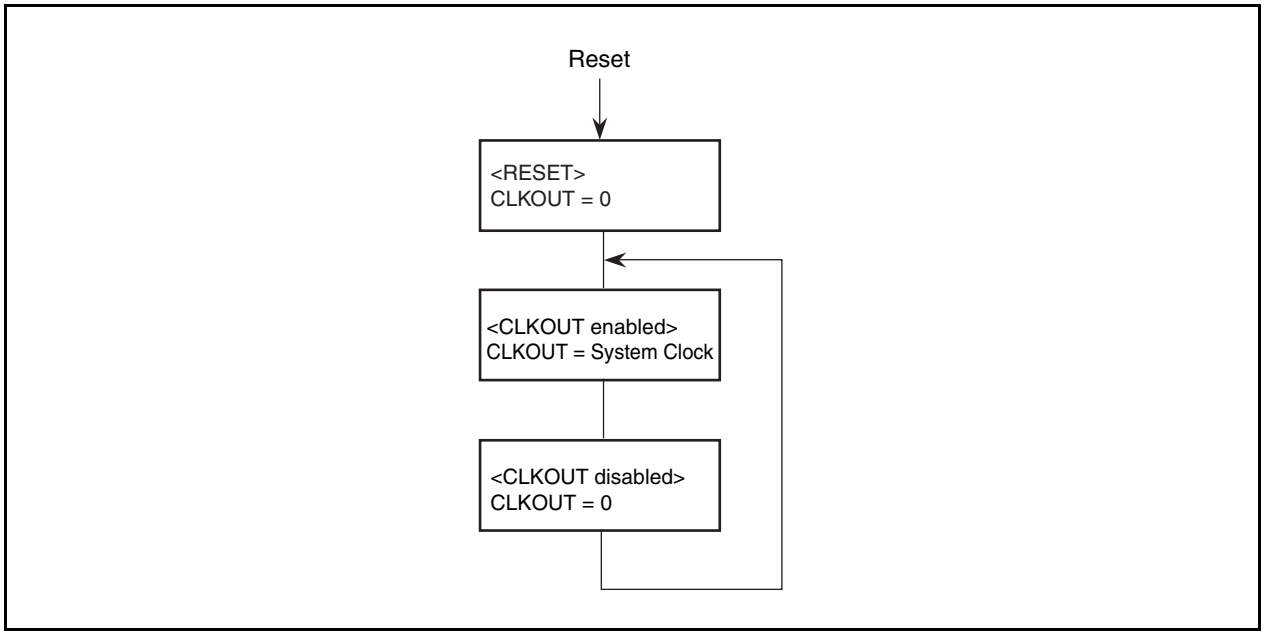


Figure 5-41. Clock Monitor Output Setting Example

```
CLKOUT output enabled
clr (r0)                ;
r0l = *CLKC:x           ;
r0 = r0 | 0x0040        ;
*CLKC:x = r0l           ;

CLKOUT output disabled
clr (r0)                ;
r0l = *CLKC:x           ;
r0 = r0 & 0xFFDF        ;
*CLKC:x = r0l           ;
```

5.13 Instruction Memory Correction Function (IMC)

The instruction memory correction (IMC) function

is used to fetch instruction memory by replacing it with data that has already been set. Since it is an instruction memory correction function, it is used when patching the contents of ROM in the μPD77213, which is a mask ROM device.

The main features of the IMC are as follows.

- Number of words that can be corrected
  Up to two instructions can be corrected.
- No overhead
  As with normal operation, this is a no-wait operation that does not generate any overhead.

Table 5-32 lists the registers for the instruction memory correction function.

Table 5-32. Registers for Instruction Memory Correction Function

X/Y Memory Address	Register Name	Function	Load/Store
0x38A0	CEFR	Correction enable flag register	L/S
0x38A1	CPR0	Correction page register 0	L/S
0x38A2	CAR0	Correction address register 0	L/S
0x38A3	CLIR0	Correction instruction code register (higher) 0	L/S
0x38A4	CUIR0	Correction instruction code register (lower) 0	L/S
0x38A5	CPR1	Correction page register 1	L/S
0x38A6	CAR1	Correction address register 1	L/S
0x38A7	CLIR1	Correction instruction code register (higher) 1	L/S
0x38A8	CUIR1	Correction instruction code register (lower) 1	L/S

5.13.1 Registers for instruction memory correction function

(1) CEFR (correction enable flag register)

This register is used to set the contents to be corrected as valid or invalid.

Table 5-33 describes the functions of the bits in CFER. The default value is 0x0.

(2) CPR0 and CPR1 (correction page registers)

These are 16-bit registers that are used to specify the instruction memory pages to be corrected. The lower six bits specify the pages. The range of specifiable values is 0x0 to 0x3F, which corresponds to the pages on which ROM actually exists.

(3) CAR0 and CAR1 (correction address registers)

These are 16-bit registers that are used to specify the instruction memory addresses to be corrected. The range of specifiable values is 0x8000 to 0xFFFF, which corresponds to the addresses at which ROM actually exists.

(4) CUIR0, CUIR1, CLIR0, and CLIR1 (correction instruction registers)

These are 16-bit registers that are used to specify the instruction code to be corrected (by performing patch processing). Since the instruction memory has 32-bit width, these registers are provided in pairs: one (CUIR) for the higher 16 bits and one (CLIR) for the lower 16 bits.

Table 5-33. Bit Configuration of CEFR

Bit	Name	Function	Load/Store
15 to 2	Reserved	Values other than 0 cannot be written. Undefined during a read operation	–
1	Correct1 enable	Enable bit for correction word 1 • 0: Disable (default) • 1: Enable The value set as correction word 1 (CPR1, CAR1, CUIR1, CLIR1) sets the correction function as valid or invalid.	L/S
0	Correct0 enable	Enable bit for correction word 0 • 0: Disable (default) • 1: Enable The value set as correction word 0 (CPR0, CAR0, CUIR0, CLIR0) sets the correction function as valid or invalid.	L/S

5.13.2 Operation of instruction correction function

The value of the instruction memory page IPR to be corrected is set to CPR, the address is set to CAR, and the instruction code to be corrected is set to CUIR and CLIR. Next, the CEFR register's enable flag is set (= 1) to start the instruction correction function. During an instruction fetch, the instruction pointer (IP) and CAR values are compared, and if they match, the instruction code set to the register is fetched. Up to two words can be corrected.

Cautions 1. Control of enabling the instruction memory correction function (via access to the CEFR register) should be executed in the area from 0x0000 to 0x7FFF in instruction RAM.

2. The two instructions immediately following an enable control value cannot be targeted for correction.

5.14 Paging Function

The DSP core kernel in μPD77210 Family devices supports a 16-bit address space (of 64 Kwords). The function that expands these 64 Kwords is called "paging".

The address range from 0x8000 to 0xFFFF in the instruction and data spaces are the paging area, and memory area expansion is performed by assigning several pages in these spaces.

Data pages can be allocated not only to the area for data page expansion but also to space accessed in external data memory or in instruction memory.

Table 5-34 lists the registers used for the paging function.

Table 5-34. Registers for Paging Function

X/Y Memory Address	Register Name	Function	Load/Store
0x38C0	IPR	Instruction page register	L/S
0c38C1	DPR	Data page register	L/S

5.14.1 Registers for paging function

(1) IPR (instruction page register)

This is a 16-bit register that is used to specify instruction pages in the instruction paging area. The paging is specified in the lower 6 bits. The specifiable range is from 0x0 to 0x3F. The default value is 0x0.

(2) DPR (data page register)

This is a 16-bit register that is used to specify data pages in the data paging area. The paging is specified in the lower 8 bits. Table 5-35 lists the specifiable values and the target area for specifying pages. The default value is 0x0.

Table 5-35. DPR Settings and Target Area

DPR	Target Area for Page Specification
0x00 to 0x3E	Data page
0x3F	External data memory window
0x80	Instruction memory (0x0000 to 0x7FFF)
0xC0 to 0xFF	Instruction memory (0x8000 to 0xFFFF) page

5.14.2 Operation of paging function

Corresponding pages are set in the paging area by setting the pages to be accessed to the IPR and DPR registers.

Cautions 1. Page switching (IPR and DPR register access) should be executed in the area from 0x0000 to 0x7FFF in instruction RAM.

2. Page switching (IPR and DPR register access) cannot be executed in parallel with data access of the paging area (0x8000 to 0xFFFF). Also, the instruction immediately following page switching cannot execute data access of the paging area (0x8000 to 0xFFFF).

3. Data page memory access using DPR = 0x80 in the instruction address area from 0x0000 to 0x7FFF causes instruction fetch and data access operations to occur in the same memory space, so the fetch operation must use one wait and two cycles. The same is true for data page memory access using DPR = 0xC0 to 0xFF when IPR = 0x0 to 0x3F in the instruction address area from 0x8000 to 0xFFFF.

5.15 Peripheral STOP Mode

Peripheral STOP mode enables the clocks supplied to TSIO, ASIO, MIO, and PMT to be stopped individually. This mode can therefore be used to reduce the power consumption of unused peripheral circuits.

Table 5-36 lists the register used by peripheral STOP mode.

Table 5-36. Peripheral STOP Mode Register

X/Y Memory Address	Register Name	Function	Load/Store
0x387A to 0x387B	POWC	Power control register	S

5.15.1 Peripheral STOP mode register

(1) POWC (power control register)

This register is used to start and stop peripheral circuits. The default value is 0x0. The functions of bits in the POWC register are described in Table 5-37.

Table 5-37. Bit Configuration of POWC

Bit	Name	Function	Load/Store
15 to 4	Reserved	Values other than 0 cannot be written. Undefined during a read operation	–
3	PMTENA	PMT enable bit • 0: Active (default) • 1: Stop	S
2	MIOENA	MIO enable bit • 0: Active (default) • 1: Stop	S
1	ASIOENA	ASIO enable bit • 0: Active (default) • 1: Stop	S
0	TSIOENA	TSIO enable bit • 0: Active (default) • 1: Stop	S

5.15.2 Operation of peripheral STOP mode

Among the settings in the POWC register, 0x387A is handled as a port control setting for PDT and 0x387B is handled likewise for PCD. Bit manipulation is performed after setting output mode. Figure 5-42 shows an example of settings.

Only output is possible; the input value is undefined.

Figure 5-42. Example of POWC Register Settings

```
Output setting for POWC register
clr (r0)                    ;
r0l = 0x300F                ;
*PCD(POWC):x = r0l          ;

TSIO stop
clr (r0)                    ;
r0l = *PDT(POWC):x          ;
r0 = r0 | 0x0001            ;
*PDT(POWC):x = r0l          ;

TSIO active
clr (r0)                    ;
r0l = *PDT(POWC):x          ;
r0 = r0 & 0x000e            ;
*PDT(POWC):x = r0l          ;
```

5.16 Debug Interface (IEIO)

The μPD77210 Family is provided with the following functions conforming to JTAG interface:

- JTAG port
- Boundary scan test function
- Debug function (In-circuit emulation function)

5.16.1 JTAG port

Joint Test Action Group (JTAG) is an organization founded to promote standardization of boundary scan, a technique to facilitate testing of printed wiring boards that are mounted in electronic systems, and a standardization plan by this organization is recommended as “IEEE1149.1.”

A device conforming to JTAG has an access port for testing, and the device can be tested independently of the internal logic.

The μPD77210 Family is provided with a register and a control circuit for In-Circuit Emulation, in addition to the instruction register, bypass register, and boundary scan register, which are specified to be essential by the above recommendation. For the details of JTAG, refer to “IEEE1149.1.”

5.16.2 Debug interface pins

Four pins and In-Circuit Emulation pin (TICE) and test pin ($\overline{\text{TRST}}$) conforming to the recommendation are provided.

- TCK (input): Test clock input pin.
  Input 0 when not used (conforms to recommendation).

  Caution Do not stop TCK while it is high.

- TMS (input): Test mode select input.
  Sampled at the rising edge of TCK. Internally pulled up.

- TDI (input): Test data input.
  Sampled at the rising edge of TCK. Internally pulled up.

- TDO (output): Test data output.
  Changes output in synchronization with the falling edge of TCK.

- TICE (output): Output to organize the break mode of In-Circuit Emulation.

- $\overline{\text{TRST}}$ (input): Test reset input
  0 during normal operation (active low) and 1 during a boundary scan when the debugger is used.
  Pulled down internally.

5.16.3 Boundary scan test function

The boundary scan test method allows testing of the board level and chip level of the target system in a consistent test phase. This is why this method has been widely employed for automatic systems at many production sites.

The μPD77210 Family has boundary scan functions as described below.

(1) Test instruction register

This 8-bit register is used to select test parameters and a test data register. Table 5-38 lists the supported instructions.

Table 5-38. Test Instructions

Bit	Instruction
7 6 5 4 3 2 1 0	
0 0 0 0 0 0 0 0	EXTEST instruction
0 0 0 0 0 0 0 1	SAMPLE/PRELOAD instruction
1 1 1 1 1 1 1 1	BYPASS instruction

(2) Test bypass register

This register outputs the data input from TDI to TDO.

5.16.4 Debug function (in-circuit emulator function)

The μPD77210 Family is provided with debug monitoring functions using JTAG with a run-time program. These functions have the following features:

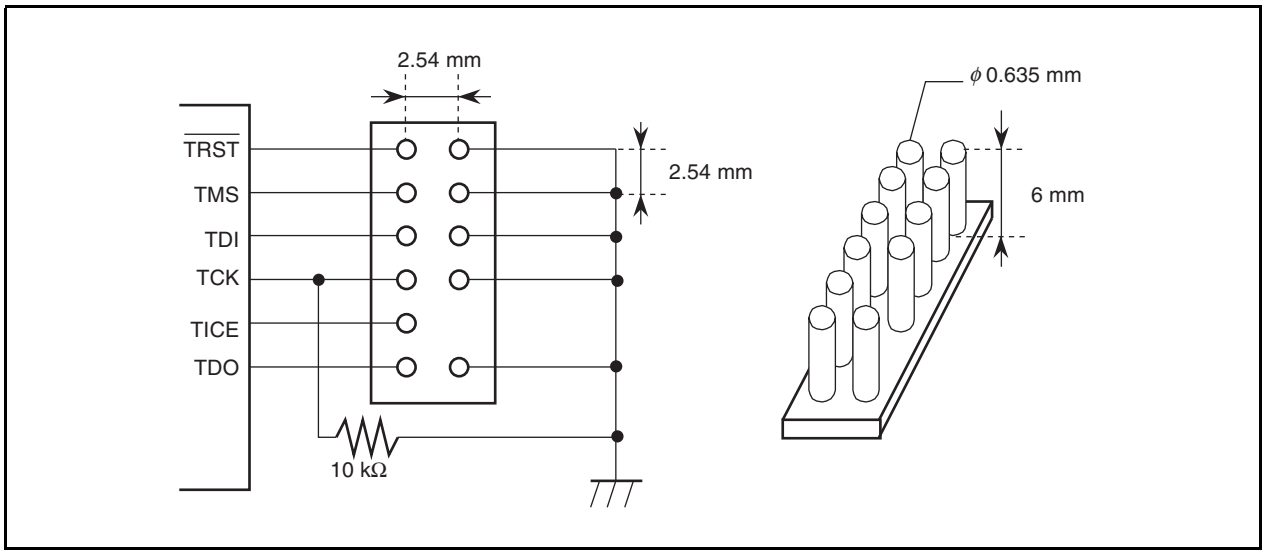
- **Break function**
  - Break by fetch of specified instruction address
  - Break by reading/writing specified data memory address

- **Non-break monitor function**
  - References or changes the contents of a register or memory during program execution

Cautions 1. The debugging functions are not available to the user.

2. The recommended hardware debugger is the NDSP_ICE002 (manufactured by Amdahl System Support). Figure 5-43 shows the JTAG pin handling when using NDSPICE002. If the user system does not use NDSP_ICE002, handle unused pins as described in 2.4 Handling of Unused Pins.

Figure 5-43. JTAG Pin Handling



5.17 Expansion Interfaces (Additional I/O)

The following describes additional inputs and outputs for the expansion interfaces. These additional inputs and outputs are interfaced by the SD card interface. The various expansion interfaces are described below starting on the next page. The following describes expansion interfaces in general. Only the μPD77213 includes an expansion interface (the μPD77210 does not include it).

Part of the external data memory space (from 0xC0000 to 0xFFFFF) is used as the expansion interface's register mapping area. This space can be used as external data memory space if the expansion interface will not be used.

The structure enables the registers in the expansion interface to be accessed via the MIO, and from the user's perspective the expansion interface exists as part of the external data memory.

Figure 5-44 shows a block diagram of the expansion interface and Table 5-39 lists its register.

Figure 5-44. Block Diagram of Expansion Interface

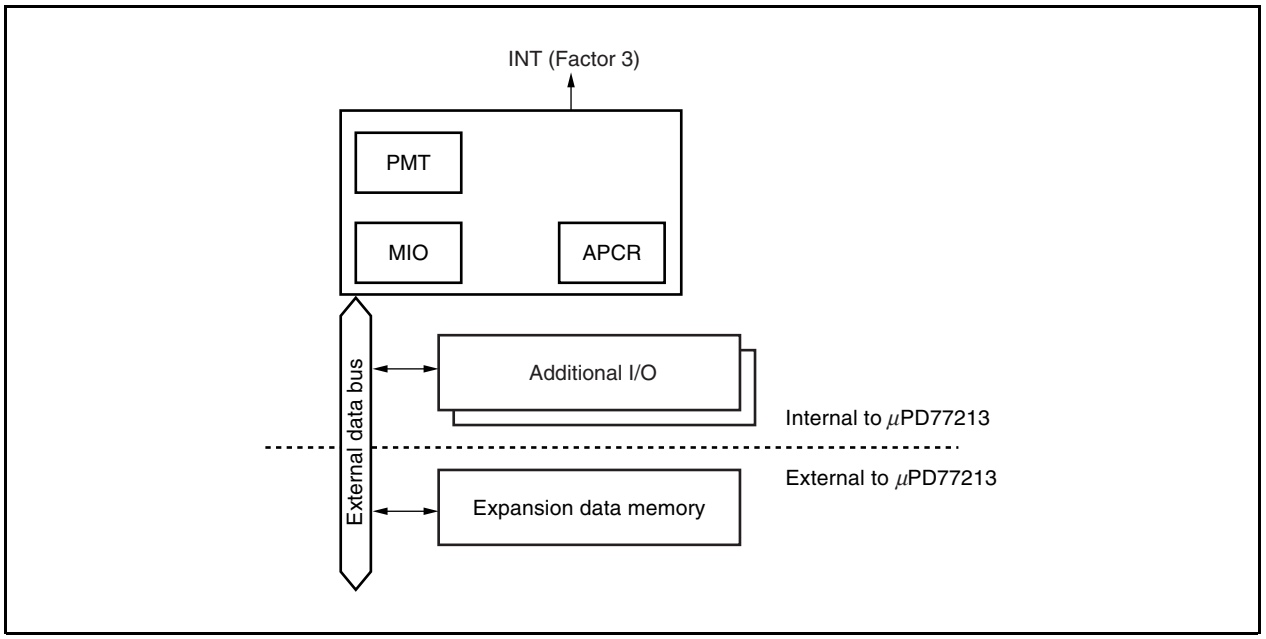


Table 5-39. Expansion Interface Register

X/Y Memory Address	Register Name	Function	Load/Store
0x38D0	APCR	Expansion interface control register	L/S

5.17.1 Expansion interface register

(1) APCR (additional peripheral control register)

This register is used to control the enabled/disabled status of the expansion interfaces (additional peripheral blocks) and to set control of PMT requests. It can control up to four peripheral blocks. One pair of load/store interrupts can be controlled per block. Multiple blocks are therefore required to control multiple pairs of load/store interrupts.

Table 5-40 describes the functions of the bits in the APCR register. The default value is 0x0.

Table 5-40. Bit Configuration of APCR

Bit	Name		Function	Load/Store
15	Peripheral A	SRE	Store request enable flag Enables store request signals to be sent from peripheral to PMT channel (ch7). 0: Disables (masks store request) 1: Enables (passes store request)	L/S
14		LRE	Load request enable flag Enables load request signals to be sent from peripheral to PMT channel (ch7). 0: Disables (masks load request) 1: Enables (passes load request)	L/S
13		PFE	Peripheral function enable flag Sets start/stop status of peripheral function. When disabled, no clock is supplied to the peripheral. 0: Enable 1: Disable	L/S
12		TOE	Pin output enable flag Sets the status of the peripheral's output pins when reset is active. 0: Disable (sets high impedance) 1: Enable (depends on peripheral) When PFE = 1, addresses are output when this status is "enable".	L/S
11 to 8	Peripheral B	Same as Peripheral A.		
7 to 4	Peripheral C			
3 to 0	Peripheral D			

Caution When performing PMT transfers, LRE and SRE must be set exclusively among the four blocks. In other words, if 1 is set for Peripheral A, 1 must not be set for Peripherals B, C, or D.

5.18 SD Card Interface (SDCIF)

The μPD77213 includes an SD (secure digital) card interface (SDCIF). This interface is configured within the expansion interfaces.

The SD card interface specification is based on SD Memory Card Specifications, Part 1 Physical Layer Specification, Version 1.0, March 2000.

Its main features are as follows.

- 3 pins
  These are a clock pin (SDCLK), command/response pin (SDCR), and a data pin (SDDAT0).

- Internal handshaking
  Handshaking is enabled by means of polling or interrupts.

Figure 5-45 shows a block diagram of the SD card interface and Table 5-41 lists its registers.

Figure 5-45. Block Diagram of SD Card Interface

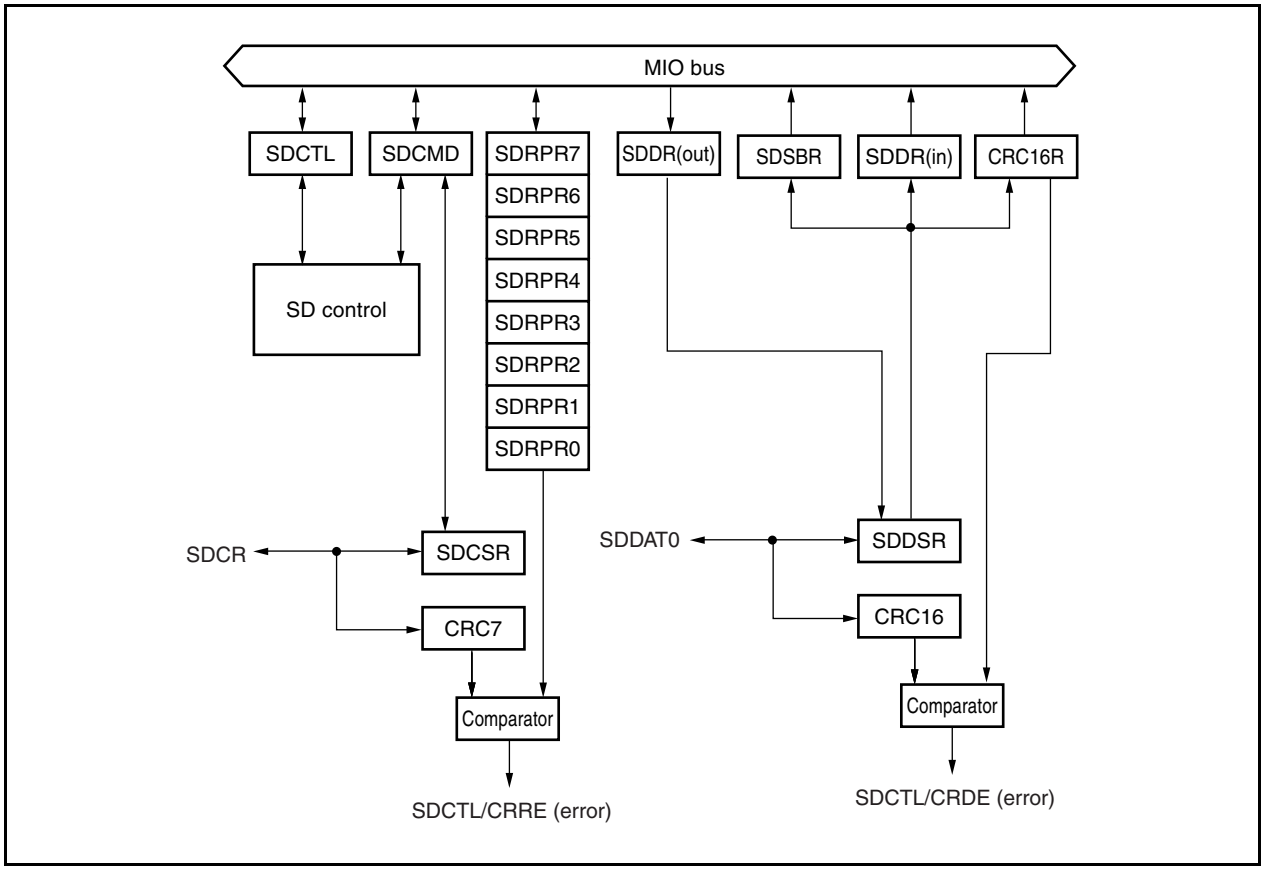


Table 5-41. SD Card Interface Registers

External Memory Address	Register Name	Function	Load/Store
0xFC000	SDDR	SD card data register	L/S
0xFC001	SDCMD_IDX	SD card command index register	L/S
0xFC002	SDCMD_AGH	SD card command argument register (H)	L/S
0xFC003	SDCMD_AGL	SD card command argument register (L)	L/S
0xFC004	SDCTL	SD card interface control register	L/S
0xFC005	SDRPR	SD card interface response register	L
0xFC006	SDSBR	SD card interface CRC status busy register	L
0xFC007	CRC16R	SD card interface command CRC register	L

5.18.1 SD card interface pins

(1) SDCLK (SD card clock – output)

The output pin connected to the SD card's CLK pin is used for the SD card's clock pin (SDCLK). The maximum operating frequency of the SDCLK pin is 25 MHz.

Shift operations in the SD card interface's shift register are clock-synchronous with SDCLK.

(2) SDCR (SD card command/response – I/O)

SDCR, a pin used for command output and response input, is connected to the SD card's CMD pin. It comprises part of a bi-directional bus that uses time sharing to transmit commands from the μPD77213 to the SD card and to receive responses from the SD card. When it is not being accessed, it should be to EV_{DD} level via a pull-up on the device's external board.

(3) SDDAT0 (SD card data – I/O)

SDDAT0, a pin used for data input and output, is connected to the SD card's DAT0 pin. Data is read after a read command is transmitted from the μPD77213 to the SD card, and it is written after a write command has been transmitted. As such, it comprises part of a bi-directional bus that the uses time sharing for its switching configuration. When it is not being accessed, it should be EV_{DD} level via a pull-up on the device's external board.

(4) SDMON (SD card access monitor – output)

SDMON is the SD card interface's access monitoring pin. Output from this pin is high level when the SD card is being accessed (i.e., when the area from 0xFC000 to 0xFFFFF is being accessed via MIO). At that time, output from MA0 to MA12 is low level.

5.18.2 SD card interface registers

(1) SDDR (SD card data register)

This is a 16-bit register that is used for input and output of the SD card's serial data. It includes separate registers for output and input. The value of SDDR can be input and output via MIO.

SDDR(out) is a 16-bit register that sets data to be output to the SD card. When a store instruction is executed to SDDR, data is input to this register from the MIO bus. When the SD card data shift register (SDDSR) becomes empty while in write mode in relation to the SD card, the value of the SDDR(out) is set to SDDDSR. When the SDCTL register's SDS bit value is 0 and a store instruction to SDDR is executed, the SDCTL's SDSE bit value is set to 1 (store error).

SDDR(in) is a 16-bit register that reads data that has been input from the SD card. When a load instruction to SDDR(in) is executed, data is output from SDDR(in) to the MIO bus. When the last bit is input to the SD card data shift register (SDDSR) while in read mode in relation to the SD card, the value of SDDSR is set to SDDR(in). When the SDCTL register's SDL bit value is 0 and a load instruction to SDDR is executed, the SDCTL register's SDLE bit value is set to 1 (load error).

(2) SDCMD (SD card command register)

SDCMD comprises three 16-bit registers that are used to set commands for controlling the SD card. SDCMD includes as separate registers SDCMD_IDX for command interpretation and command index output, SDCMD_AGH for higher command arguments, and SDCMD_AGL for lower command arguments.

The value of SDCMD can be input and output via MIO.

(a) SD card command index register (SDCMD_IDX)

This is a 16-bit register that is used as an index of commands output to the SD card and to set command interpretation information for controlling response data. When a store instruction is executed to SDCMD_IDX, data is input to this register from the MIO bus. When the SDCTL register's CEN bit value is set to 1, the values of bits 5 to 0 of in the SDCMD_IDX register are set to SDCSR and are transmitted to the SD card. The value after reset is 0x4800. Table 5-42 describes the functions of the bits in SDCMD_IDX.

(b) SD card command argument higher register (SDCMD_AGH)

This is a 16-bit register that is used to set the higher command arguments that are output to the SD card. When a store instruction to SDCMD_AGH is executed, data is input to SDCMD_AGH from the MIO bus. When SDCTL's CEN bit is set to 1, the value of SDCMD_AGH is set to SDCSR, allocated to command bits 39 to 24, and then transmitted to the SDR card.

(c) SD card command argument lower register (SDCMD_AGL)

This is a 16-bit register that is used to set the lower command arguments that are output to the SD card. When a store instruction to SDCMD_AGL is executed, data is input to SDCMD_AGL from the MIO bus. When SDCTL's CEN bit is set to 1, the value of SDCMD_AGL is set to SDCSR, allocated to command bits 23 to 8, and then transmitted to the SDR card.

(3) SDCTL (SD card interface control register)

This is a 16-bit register that is used to set values for controlling the SD card interface. The value of SDCTL can be input and output via the MIO bus.

The value after reset is 0x0202. Table 5-43 describes the functions of the bits in SDCTL.

(4) SDRPR (SD card interface response register)

This register is used to retain response data that has been received from the SD card. It has an (8 × 16 bit) FIFO configuration.

SDRPR includes eight separate registers (SDRPR7 to SDRPR0), and when SDMCD_IDX's BSL bit is 1 (response bits: 136 bits), response data is retained sequentially starting from SDRPR7. When SDMCD_IDX's BSL bit is cleared to 0 (response bits: 48 bits), response data is retained sequentially starting from SDRPR2 and insignificant data is input to SDRPR3 to SDRPR7.

SDRPR is mapped to a single peripheral address. The values of SDRPR0 to SDRPR7 can be output via the MIO bus by loading eight times from the program.

(5) SDDSR (SD card interface data shift register)

This is a 16-bit register that is used to directly transmit and receive data via the SDDAT0 pin in order to function as an interface between the SD card and the serial data. SDDSR is not connected to the MIO bus.

(6) SDCSR (SD card interface command shift register)

This is a 16-bit register that is used directly transmit and receive commands and responses via the SDCR pin in order to function as an interface with the SD card. SDCSR is not connected to the MIO bus.

(7) SDCRCR (SD card interface control register)

This includes two registers: SDCRCR7, which is used to retain the CRC7 bit of responses received from the SD card and SDCRCR16, which is used to retain the CRC16 bit from read data received from the SD card.

SDCRCR7, which retains the CRC7 bit of responses received from the SD card, shares bits 1 to 7 of SDRPR0. The 7-bit CRC data that is retained by SDCRCR7 is compared with the CRC checksum result of the response received in the μPD77213, and if they do not match an error flag is set. Since it is part of SDRPR0, it can be used for loading but not for storing.

SDCRCR16 is a 16-bit register that retains the CRC16 bit from read data received from the SD card. The 16-bit CRC data that is retained by SDCRCR16 is compared with the CRC checksum result of the read data received in the μPD77213, and if they do not match an error flag is set. SDCRCR16 can be used for both loading and storing.

(8) SDSBR (SD card interface control register)

After data is transmitted to the SD card, this register retains the CRC status and busy information that is sent to the SD card interface from the SD card.

SDSBR is a 4-bit register, in which bit 3 functions a busy flag and bits 0 to 2 function as CRC status bits. This register can be used for loading only.

The busy flag indicates busy status when set to 1 and not busy mode when cleared to 0.

The CRC status bit retains the CRC status that was transmitted from the SD card.

After data is transmitted to the SD card, the SDSBR register's busy flag and CRC status bits must be checked by software and, if the busy flag value is 1, the following action (command/data transmission) must not be performed for the SD card.

Table 5-42. Bit Configuration of SDCMD_IDX

Bit	Name	Function	Load/Store
15	Reserved	• Values other than 0 must not be written. • Undefined during a read operation	–
14 to 11	DBL	Data block range setting bits • 0000: 1 byte • 0001: 2 bytes • 0010: 4 bytes • 0011: 8 bytes • 0100: 16 bytes • 0101: 32 bytes • 0110: 64 bytes • 0111: 128 bytes • 1000: 256 bytes • 1001: 512 bytes (default) • 1010: 1024 bytes • 1011: 2048 bytes All other values are reserved (setting prohibited).	L/S
10	RES	Response setting bit This bit is used to set whether there is a response from the SD card. • 0: No response (default) This is handled as a “no response” operation. • 1: Response exists	L/S
9, 8	RW	Read/write setting bits These bits are used to set data read, write, or data transfer stop operations. • 00: No read/write (default) This is handled as a “no data” operation. • 01: Read operation • 10: Write operation • 11: No read/write	L/S
7	Reserved	• Values other than 0 must not be written. • Undefined during a read operation	–
6	BSL	Response bit count setting bit • 0: 48 bits (default) • 1: 136 bits	L/S
5 to 0	IDX	These bits are used to set the command index to be transmitted to the SD card. The values of these bits are allocated to command bits 45 to 40 when transmitting commands.	L/S

Table 5-43. Bit Configuration of SDCTL (1/2)

Bit	Name	Function	Load/Store
15	ALLE	This flag indicates the error status within all of the SDCIF (SD card interface). This flag is set to 1 when an error flag has been set in CRRE, CRDE, SDSE, or SDLE. • 0: No error (default) • 1: Error exists Set a zero to this bit to clear it.	L/S
14	CEN	Command execution enable This bit is used to transmit commands set to the SDCMD register to the SD card. When this bit is set to 1, the command that has been set to the SDCMD register is transmitted to the SD card and the SD card performs control operations after receiving the command. After the command has been transmitted, this bit (CEN) is automatically cleared to zero. • 0: No command transmission (default) • 1: Command transmission in progress	L/S
13 to 10	DIV	SD card clock division These bits are used to set the division rate for generating a clock for the SD card from the system clock. • 0000: Setting disabled • 0001: Setting disabled • 0010: Setting disabled • 0011: 1/4 (PMT enabled) • 0100: 1/5 (PMT enabled) • 0101: 1/7 (PMT enabled) • 0110: 1/8 (PMT enabled) • 0111: 1/9 • 1000: 1/10 • 1001: 1/12 • 1010: 1/16 • 1011: 1/32 • 1100: 1/64 • 1101: 1/128 • 1110: 1/256 • 1111: 1/512	L/S
9	SDSTP	SD card interface stop bit This bit is set to 1 when the SDCIF is not used or to stop accessing the SD card. When this bit's value becomes 1, SDCLK becomes fixed at low level and SD card access is stopped. At that time, the status of SDCIF is retained. • 0: Operates SD card interface • 1: Stops SD card interface (default) When oscillation of SDCLK is started, access must not begin until a stabilization period of at least two SDCLK cycles has been secured.	L/S
8	SDRST	SD card interface reset enable bit When this bit is set to 1, the contents (register values, etc.) of SDCIF are reset. After a reset, the contents are automatically cleared to zero. • 0: (default) • 1: Reset	L/S
7	Reserved	• Values other than 0 cannot be written. • Undefined during a read operation	L/S
6	CRRE	CRC response error flag The CRC result for the response received from the SD card is compared with the CRC result calculated within the μPD77213, and if the two results do not match this bit (CRRE) is set (= 1). • 0: No error (default) • 1: Error exists Set a zero to this bit to clear it.	L/S

Table 5-43. Bit Configuration of SDCTL (2/2)

Bit	Name	Function	Load/Store
5	CRDE	CRC data error flag The CRC result for the data received from the SD card is compared with the CRC result calculated within the μPD77213, and if the two results do not match this bit (CRDE) is set (= 1). • 0: No error (default) • 1: Error exists Set a zero to this bit to clear it.	L/S
4	SDSE	SDDR store error flag When the SDS flag's value is 0 and data has been stored to SDDOR, this bit is set to 1. • 0: No error (default) • 1: Error exists Set a zero to this bit to clear it.	L/S
3	SDLE	SDDR load error flag When the SDL flag's value is 0 and data has been loaded to SDDIR, this bit is set to 1. • 0: No error (default) • 1: Error exists Set a zero to this bit to clear it.	L/S
2	SDRL	SDRPR load enable flag SDRPR comprises eight 16-bit registers. • 0: Indicates that no data exists in SDRPR7 to SDRPR0 (default) This bit's value becomes 0 when SDRPR0 is loaded from a program. • 1: Indicates that data exists in SDRPR7 to SDRPR0 This bit's value becomes 1 when all responses from SDCSR to SDRPR have been transferred.	L/S
1	SDS	SDDR store enable flag • 0: Indicates that data exists in SDDR (out) This bit's value becomes 0 when data is stored to SDDDR from a program. • 1: Indicates that no data exists in SDDR (out) (default) This bit's value becomes 1 when data has been transferred from SDDR to SDDSR.	L/S
0	SDL	SDDR load enable flag • 0: Indicates that no data exists in SDDR (in) (default) This bit's value becomes 0 when data is loaded to SDDDR from a program. • 1: Indicates that data exists in SDDR (in) This bit's value becomes 1 when data has been transferred from SDDSR to SDDR.	L/S

Table 5-44. Configuration of SDRPR Register

Register Name	Function
SDRPR7	This register is used to retain responses in bits 127 to 112, out of bits 135 to 0 that have received responses from the SD card. The response in SDCSR are first retained, and after all valid responses have been retained in SDRPR, the first load operation can be performed for output via the MIO bus. This register is valid when the value of the SDCMD_IDX register's BSL bit is 1.
SDRPR6	This register is used to retain responses in bits 111 to 96, out of bits 135 to 0 that have received responses from the SD card. The responses in SDCSR are retained following SDRPR7. When the load operation following loading of SDRPR7 is performed, the responses from this register (SDRPR6) can be output via the MIO bus. This register is valid when the value of the SDCMD_IDX register's BSL bit is 1.
SDRPR5	This register is used to retain responses in bits 95 to 80, out of bits 135 to 0 that have received responses from the SD card. The responses in SDCSR are retained following SDRPR6. When the load operation following loading of SDRPR6 is performed, the responses from this register (SDRPR5) can be output via the MIO bus. This register is valid when the value of the SDCMD_IDX register's BSL bit is 1.
SDRPR4	This register is used to retain responses in bits 79 to 64, out of bits 135 to 0 that have received responses from the SD card. The responses in SDCSR are retained following SDRPR5. When the load operation following loading of SDRPR5 is performed, the responses from this register (SDRPR4) can be output via the MIO bus. This register is valid when the value of the SDCMD_IDX register's BSL bit is 1.
SDRPR3	This register is used to retain responses in bits 63 to 48, out of bits 135 to 0 that have received responses from the SD card. The responses in SDCSR are retained following SDRPR4. When the load operation following loading of SDRPR4 is performed, the responses from this register (SDRPR3) can be output via the MIO bus. This register is valid when the value of the SDCMD_IDX register's BSL bit is 1.
SDRPR2	This register is used to retain responses in bits 47 to 32, out of bits 135 to 0 or 47 to 0 that have received responses from the SD card. When the BSL bit of the SDCMD_IDX register is 1, the responses in SDCSR are retained following SDRPR3. When the load operation following loading of SDRPR3 is performed, the responses from this register (SDRPR2) can be output via the MIO bus. When the BSL bit of the SDCMD_IDX register is 0, the responses in the SDSCR register are first retained, after valid responses have been retained to SDRPR2 to SDRPR0, the first load operation can be performed for output via the MIO bus.
SDRPR1	This register is used to retain responses in bits 31 to 16 out of bits 135 to 0 or bits 47 to 0 that have received responses from the SD card. The responses in SDCSR are retained after retaining SDRPR2. When the load operation following loading of SDRPR2 is performed, the responses from this register (SDRPR1) can be output via the MIO bus.
SDRPR0	This register is used to retain responses in bits 15 to 0 out of bits 135 to 0 or bits 47 to 0 that have received responses from the SD card. The responses in SDCSR are retained following SDRPR1. When the load operation following loading of SDRPR1 is performed, the responses from this register (SDRPR1) can be output via the MIO bus. When responses are retained in SDRPR0, the SDCTL register's SDRL bit is set to 1 to indicate that responses have been entered in SDRPR7 to SDRPR0. When SDRPR0 has been loaded, the SDCTL register's SDRL bit is cleared to 0 to indicate that no responses are in SDRPR7 to SDRPR0.

5.18.3 CRC (Cyclic Redundancy Codes) circuit

The CRC circuit detects data errors between the μPD77213 and the SD card.

CRC data calculated within the μPD77213 concerning commands or write data transmitted from the SD card interface is added and output.

CRC data that has been added to responses or read data received from the SD card is compared with the CRC data calculated within the μPD77213 concerning responses or read data, and if they do not match an error flag is set.

5.18.4 Operation of SD card interface

Access to the SD card is implemented using the registers that have been mapped to the external memory space via the MIO bus. Status-based polling and interrupts can be used for this access. Wait-based access cannot be used.

Interrupts occur due to the following four factors.

- Response load request: Interrupt No. 4 (entry: 0x220), three factors
- Command store request (release busy)[Note]: Interrupt No. 5 (entry: 0x224), three factors
- Data load request: Interrupt No. 6 (entry: 0x228), three factors
- Data store request: Interrupt No. 7 (entry: 0x22C), three factors

Note Since the command store request is used to indicate “release busy” status, this interrupt request does not occur when busy status is not in effect.

The APCR register is used for SDCIF control of Peripheral C and Peripheral D. Peripheral A and Peripheral B are used for PMT request control during external data memory access. The default value is 0x0000.

Table 5-45. Bit Configuration of μPD77213's APCR Register (1/2)

Bit	Name		Function	Load/Store
15	Peripheral A (MIO)	SRE	Store request enable flag Enables data store request from external data memory interface. • 0: Disable (mask store request) • 1: Enable (allow store request)	L/S
14		LRE	Load request enable flag Enables data load request from external data memory interface. • 0: Disable (mask load request) • 1: Enable (allow load request)	L/S
13		PFE	Not used Set to 1.	L/S
12		TOE	Not used Set to 0.	L/S
11	Peripheral B (MIO)	SRE	Store request enable flag Enables data store request from external data memory interface. • 0: Disable (mask store request) • 1: Enable (allow store request)	L/S
10		LRE	Load request enable flag Enables data load request from external data memory interface. • 0: Disable (mask load request) • 1: Enable (allow load request)	L/S

Table 5-45. Bit Configuration of μPD77213's APCR Register (2/2)

Bit	Name		Function	Load/Store
9	Peripheral B (MIO)	PFE	Not used Set to 1.	L/S
8		TOE	Not used Set to 0.	L/S
7	Peripheral C (SDCIF)	SRE	Data store request interrupt enable flag Enables data store request interrupt signal from SDCIF. • 0: Disable (mask interrupt request) • 1: Enable (allow interrupt request)	L/S
6		LRE	Data load request interrupt enable flag Enables data load request interrupt signal from SDCIF. • 0: Disable (mask interrupt request) • 1: Enable (allow interrupt request)	L/S
5		PFE	Release busy interrupt enable flag Enables release busy interrupt signal from SDCIF. • 0: Disable (mask interrupt request) • 1: Enable (allow interrupt request)	L/S
4		TOE	Not used Set to 1.	L/S
3	Peripheral D (SDCIF)	SRE	Not used Set to 0.	L/S
2		LRE	Response load request interrupt enable flag Enables response load request interrupt signal from SDCIF. • 0: Disable (mask interrupt request) • 1: Enable (allow interrupt request)	L/S
1		PFE	SD card interface function enable flag Sets operation/stop status of SDCIF function. When disabled, no clock is supplied to SDCIF. • 0: Enable • 1: Disable (shared pins are used for external data memory address)	L/S
0		TOE	SD card interface pin output enable flag This bit sets the status of the SDCIF's output pins during reset active mode. • 0: Disable (sets high impedance) • 1: Enable (enables SDCIF signal)	L/S

Caution The LRE and SRE bits are set for PMT transfer requests, and do not have mask control over the interrupt controller. When PMT is transferred, the LRE and SRE bits are set (= 1). However, LRE and SRE must be set exclusively among the four blocks. In other words, if 1 is set for Peripheral A, 1 must not be set for Peripherals B, C, or D.

CHAPTER 6 BOOT FUNCTIONS

This chapter describes the boot functions in μPD77210 Family devices.

μPD77210 Family devices include interrupt vectors and instruction memory RAM that begins at address 0x0200. Application programs must be loaded to this instruction RAM space.

The boot routine that is used to boot instructions in the instruction RAM (part of the instruction memory space) is stored in ROM. The μPD77210 Family device executes boot processing after a system reset has been cleared and, once the boot processing is completed, instructions are executed starting at address 0x0200.

The following instruction code (op code) sources can be selected for boot processing.

- X data memory boot
- Y data memory boot
- XY data memory boot
- External data memory boot
- Host boot
- Serial boot
- Non-boot (jump to 0x0200 without boot processing)

The boot routine can also be called from a user's application program. After an initial reset boot (first boot after a reset) is completed, a reboot function that rewrites the instructions is supported.

6.1 Initial Reset Boot

6.1.1 Boot mode specification

After reset is released, the boot mode is determined based on the settings for the general-purpose port (pins P0 to P02). Table 6-1 describes the relationship between general-purpose port settings and boot modes. After reset is released, the PLL's lock range is specified based on the setting for the P3 pin. This setting is required in combination with the boot mode specification. Table 6-2 lists the PLL's lock range settings.

Since the transfer destination page (DPR = 0x80) is fixed by the values set to the general-purpose port (pins P0 to P2), the boot processing in the instruction memory area from 0x8000 to 0xFFFF should be performed by a boot routine.

The maximum number of boot words differs depending on the boot mode, and ranges from 0x4000 16 Kwords – 1 to 0x7D00 (31.5 Kwords). The contents of DPR are written to 0x00 by initial reset boot processing.

Table 6-1. Initial Reset Boot Mode

P2	P1	P0	Boot Mode
0	0	0	Non-boot
0	0	1	X memory boot
0	1	0	Y memory boot
0	1	1	XY memory boot
1	0	0	External data memory boot
1	0	1	Host boot
1	1	0	Serial boot
1	1	1	Setting prohibited

Table 6-2. PLL's Lock Range

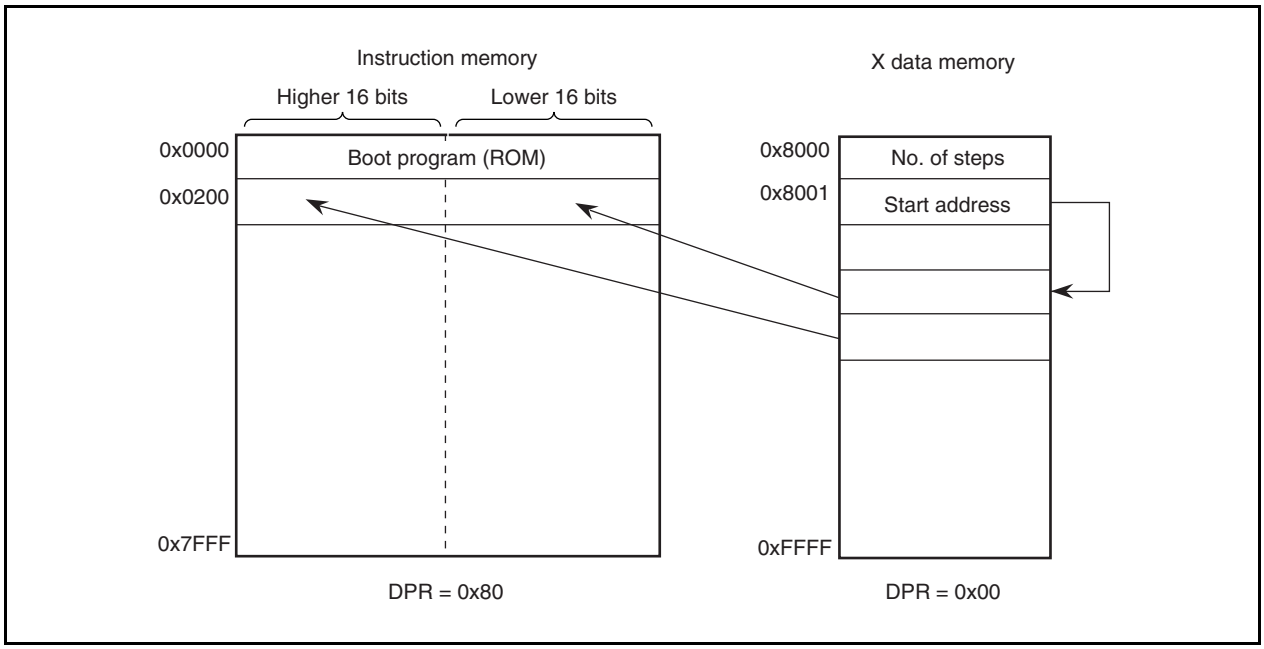
P3	Lock Range
0	120 to 160 MHz
1	80 to 120 MHz

Caution During boot processing, the general-purpose port settings should remain fixed from three cycles prior to system reset release to the instruction execution cycle at address 0x200 following operation of the boot program.

6.1.2 X memory boot

In this boot mode, instruction code is booted starting from address 0x8000 (DPR = 0x0) in internal X data memory. The number of instruction steps to be booted and the source start address are required as parameters. The maximum number of boot words is 0x4000 (16 Kwords) – 1.

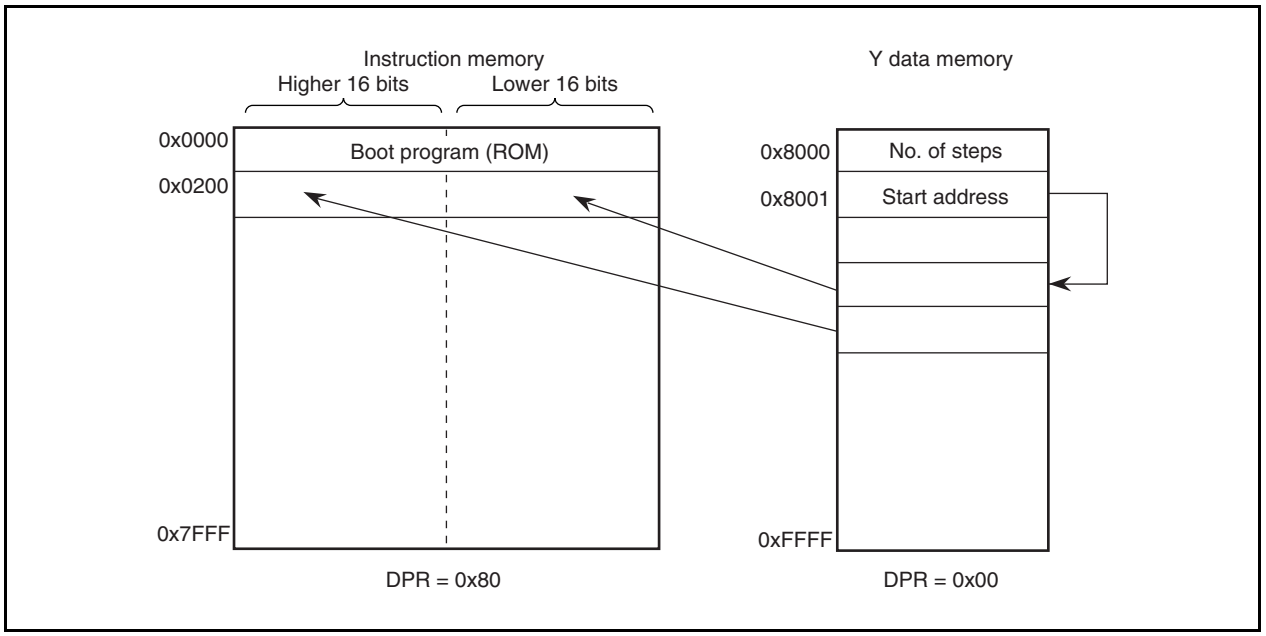
Figure 6-1. Boot from X Memory



6.1.3 Y memory boot

In this boot mode, instruction code is booted starting from address 0x8000 (DPR = 0x0) in internal Y data memory. The number of instruction steps to be booted and the source start address are required as parameters. The maximum number of boot words is 0x4000 (16 Kwords) – 1.

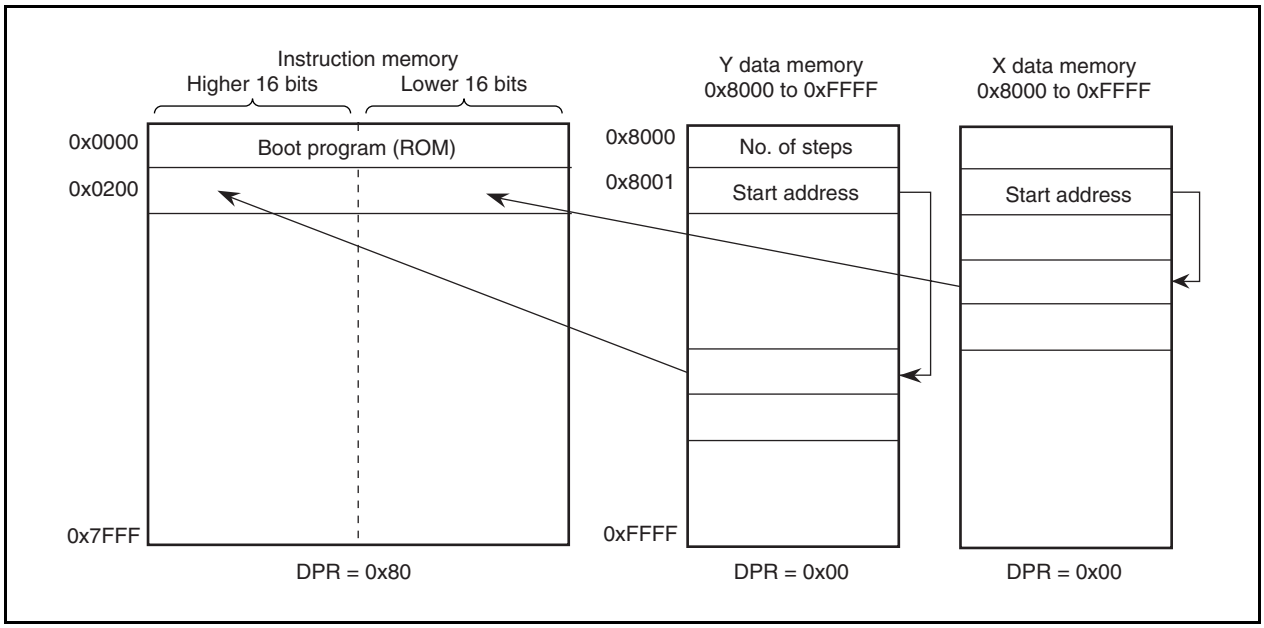
Figure 6-2. Boot from Y Memory



6.1.4 XY memory boot

In this boot mode, instruction code is booted starting from address 0x8000 (DPR = 0x0) in both internal X data memory and internal Y data memory. The higher 16 bits of the instruction code are booted from Y memory and the lower 16 bits are booted from X memory. The maximum number of boot words is 0x7D00 (31.5 Kwords).

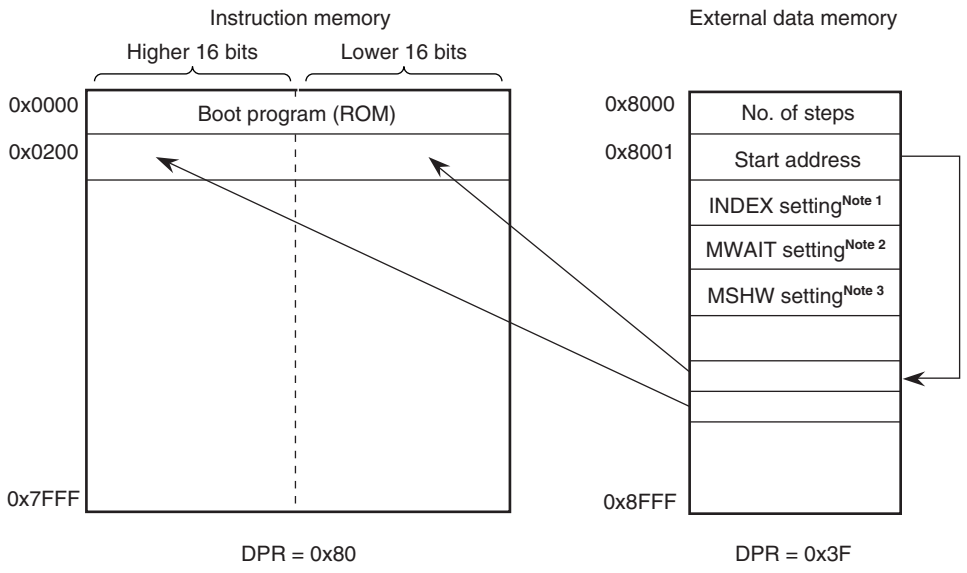
Figure 6-3. Boot from X Memory and Y Memory



6.1.5 External data memory boot

In this boot mode, instruction code is booted from external data memory. The number of instruction steps to be booted and the source start address are required as parameters, along with the wait count. The maximum number of boot words is 0x4000 (16 Kwords).

Figure 6-4. Boot from External Data Memory



Notes 1. Values set to index register
2. Values set to MWAIT register
3. Values set to MSHW register

The parameter settings for external memory are performed during boot processing from the μPD77210 Family device to the direct access on the X side. Examples are shown below.

Example 1. When instructions have been assigned to start at external memory address 0x00005 (20-bit space).
source start address: 0x8005
index data: 0x0000

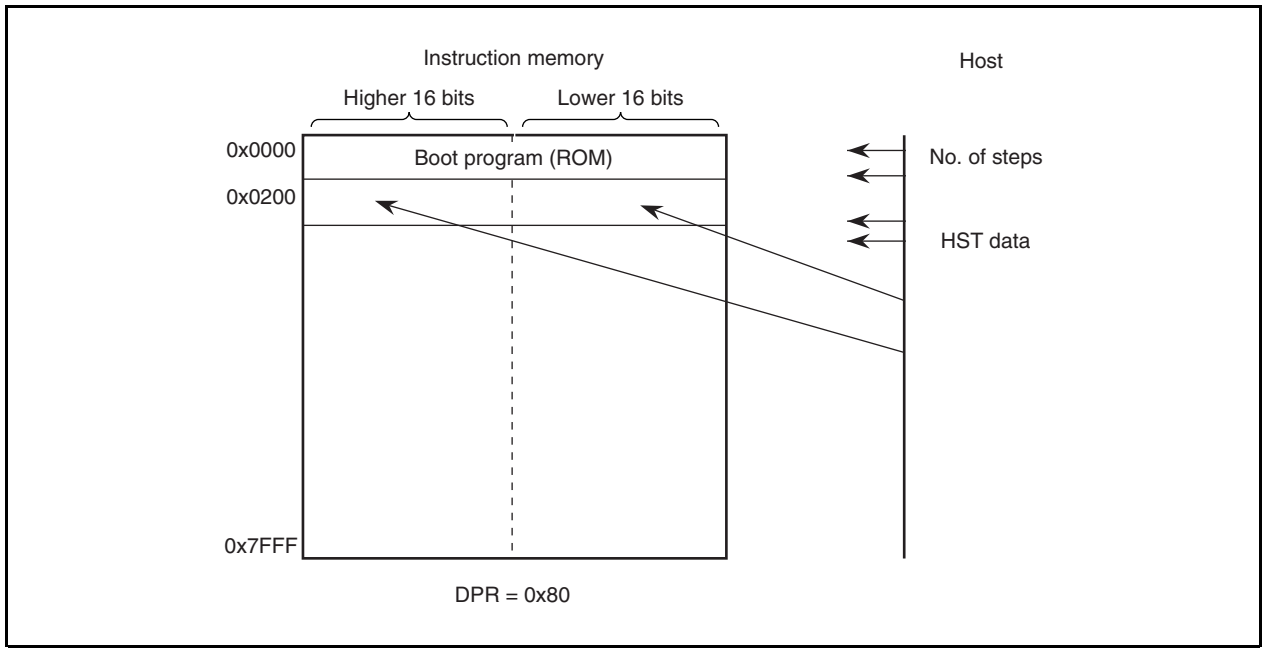
Example 2. When instructions have been assigned to start at external memory address 0x10005 (20-bit space).
source start address: 0x8005
index data: 0x0010

Caution In the μPD77213, the memory address pins are shared as SD card interface pins, so they occupy an 8-Kword space immediately after a reset. Consequently, the boot sources are limited to this space.

6.1.6 Host boot

In host boot mode, the host device boots the instruction code from the host device via the host interface. The number of instruction steps to be booted and the HST (host status register) are required as parameters. The wait operation performs handshaking with the host interface. The maximum number of boot words is 0x7D00 (31.5 Kwords).

Figure 6-5. Host Boot Operation



Caution The host interface is set to 8-bit bus mode after a reset is released. The parameters (number of steps, HST data) for the first two words should be transferred first to the parameters' lower 8-bit bus in the host bus while in 8-bit bus mode, then to the higher 8 bits. When a host boot operation is started, the boot program sets 0x400 to HST (this sets wait mode and enables output via the $\overline{\text{HWE}}$ pin). After the settings in HST are transferred, the transferred values are overwritten, during which time the wait mode specification should not be changed.

(1) Host interface settings

Before a host boot operation is started, the following settings are made to the host interface. With the exception of HAWE, these settings are overwritten by the transferred HST setting parameters.

HST = 0x0401

- HAWE = 1: Uses waits.
- HREM = 0: Does not mask HRE
- HWEM = 0: Does not mask HWE

Caution The values in the HST register are overwritten when the HST setting parameters are set during boot processing.

(2) Host boot parameters

The host boot parameters for a reset are described below.

- No. of boot instructions: This indicates the number of instruction steps (number of boot-related instructions) in the boot program. The amount of data that is actually transferred as data is double the number of instruction steps.
- HST settings: This is the data that is set to HST (host status register). Settings are made to all bits except HAWE (bit 10). The value of the HAWE bit remains "1".
- Instruction code (op code): First, the lower 16 bits (bits 15 to 0) of the 32-bit instruction are transferred, then the higher 16 bits (bits 31 to 16) are transferred. Consequently, when in 8-bit bus mode, the byte ordering of these transfers is: bits 7 to 0, bits 15 to 8, bits 23 to 16, and then bits 31 to 24. When in 16-bit bus mode, the word ordering of the transfers is bits 15 to 0 first, then bits 31 to 16.

The parameters are listed below, arranged in order of transfer from the perspective of the host.

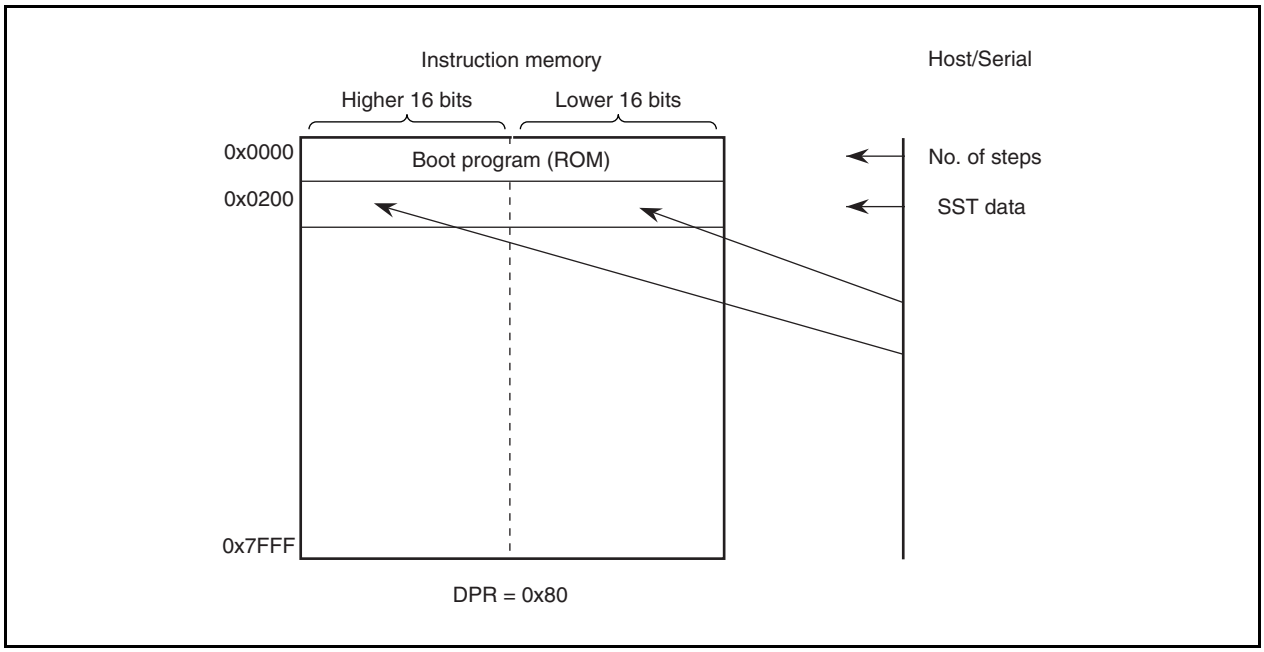
Order	8-Bit Bus Mode	16-Bit Bus Mode	
	HD7 to HD0	HD15 to HD8	HD7 to HD0
1	Lower 8 bits of number of steps	Dummy	Lower 8 bits of number of steps
2	Higher 8 bits of number of steps	Dummy	Higher 8 bits of number of steps
	Number of waits while μPD77210 Family device is loading data from HDT(in)		
3	Lower 8 bits of HST	Dummy	Lower 8 bits of HST[Note]
4	Higher 8 bits of HST	Dummy	Higher 8 bits of HST[Note]
	Number of waits while μPD77210 Family device is loading data from HDT(in)		
5	First instruction code (bits 7 to 0)	First instruction code (bits 15 to 8)	First instruction code (bits 7 to 0)
6	First instruction code (bits 15 to 8)	First instruction code (bits 31 to 24)	First instruction code (bits 23 to 16)
	Number of waits while μPD77210 Family device is loading data from HDT(in)		
7	First instruction code (bits 31 to 24)	Second instruction code (bits 15 to 8)	Second instruction code (bits 7 to 0)
8	First instruction code (bits 23 to 16)	Second instruction code (bits 31 to 24)	Second instruction code (bits 23 to 16)
	Number of waits while μPD77210 Family device is loading data from HDT(in)		
9	Second instruction code (bits 7 to 0)	Third instruction code (bits 15 to 8)	Third instruction code (bits 7 to 0)
10	Second instruction code (bits 15 to 8)	Third instruction code (bits 31 to 24)	Third instruction code (bits 23 to 16)
	Number of waits while μPD77210 Family device is loading data from HDT(in) • • •		

Note Specifies 16-bit bus mode.

6.1.7 Serial boot

In this mode, instruction code is booted by a serial device working via the TDM serial interface. The number of instruction steps to be booted and the SST (serial status register) are required as parameters. The wait operation performs handshaking with the serial interface. The maximum number of boot words is 0x7D00 (31.5 Kwords).

Figure 6-6. Serial Boot Operation



After reset is released, when the SIAK signal is asserted active, the parameters and program data are successively input via the TSI pin.

The initial settings of the serial status register (SST) is 0x0EC0, which sets the status as MSB first, 16 bits, access wait enabled, and continuous serial input mode.

Basically, the SST parameters are fixed as 0x0EC0. These settings are the same for a reboot.

6.1.8 Non-boot

In this mode, the instructions starting at 0x200 are executed after reset is released without any boot operation. Since the area starting at address 0x200 is a RAM area in μPD77210 Family devices, this area is undefined after power-on. Consequently, this mode cannot be specified as the boot mode after turning on the power.

When a reset is input after a boot operation has been completed, this mode can be set to avoid another boot operation.

However, non-boot mode cannot be used when restoring from a standby mode (HALT or STOP). The reason for this is that, during a reset, when the clock circuit is switched from the divider output to the direct clock input via the CLKIN pin, hazards may occur in the system clock, which means that the contents of the on-chip RAM and internal registers cannot be guaranteed.

6.2 Initial Reset Boot and PLL

The operation of the PLL differs depending on the initial reset boot mode.

A reset forcibly sets operation based on an externally input clock. In some boot modes, the PLL is started by the boot routine. Table 6-3 shows the relationship between boot modes and PLL activation.

In boot modes that do not start the PLL, an externally input clock is used for booting. In boot modes that use the PLL, after booting, the PLL must be started and the clock source must be switched by the user program.

In boot modes that start the PLL and switch the clock source to PLL output, the PLL's output clock is used for booting.

In any of these modes, the divider is started during processing of the boot routine. (However, the clock source is not switched to the divider. Also, the division rate is set as 1/1.)

Table 6-3. Boot Modes and PLL

Boot Mode	PLL
Non-boot	PLL is not started
X memory boot	PLL is started and is selected as the clock source.
Y memory boot	PLL is started and is selected as the clock source.
XY memory boot	PLL is started and is selected as the clock source.
External data memory boot	PLL is not started
Host boot	PLL is started and is selected as the clock source.
Serial boot	PLL is started and is selected as the clock source.

6.3 Reboot

Reboot means that an application calls a reboot routine from ROM in order to enable setting of instruction data to instruction RAM.

Table 6-4 lists reboot entry addresses and parameters for settings.

For reboot, the transfer source and transfer destination pages can be specified. Also, during a reboot there is no distinction between instructions and data as objects to be transferred. When 0x00 to 0x3F is used for the data page register (DPR), boot operations can also target data memory. However, when booting data memory, it uses the same instructions as are used for booting instruction memory, so the steps are managed in 32-bit units and the booting is done in the same way for X and Y memories. The maximum number of boot words, which varies depending on the boot mode, ranges from 0x4000 (16 Kwords) –1 to 0x7D00 (31.5 Kwords) and is performed in page units.

Table 6-4. Reboot Entry Addresses and Parameters

Reboot Mode	Entry Address	Parameter					
		No. of Instruction Steps	Start Address of Transfer Source	Transfer Source	Start Address of Transfer Destination	Transfer Destination Page (DPR)	Other Register Used
X reboot	0x1	R7L	DP3	R6L[Note 1]	DP2	R5L	DP6
Y reboot	0x2	R7L	DP7	R6L[Note 1]	DP6	R5L	DP2
XY reboot	0x3	R7L	DP3, DP7	R6L[Note 1]	DP2	R5L	DP6
External reboot	0x4	R7L	DP3	R6L[Note 2]	DP2	R5L	DP6
Host reboot	0x5	R7L	–	R6L[Note 3]	DP2	R5L	DP6
Serial reboot	0x6	R7L	–	R6L[Note 4]	DP2	R5L	DP6

Notes 1. page: Specifies the transfer source's data page

2. index: Specifies the transfer source's index register

3. hst: Specifies the settings in HST. Always use "1" for HAWE.

4. sst: Specifies the settings in SST. Always use "0xEC0" for HAWE.

Some reboot-related caution points are listed below.

Cautions 1. During reboot processing, the values in the registers being used are destroyed, one level of the program stack is used (for entries), and one level of the loop stack is also used. Be sure to save and restore any required registers. Make sure that all interrupts are disabled before rebooting.

2. Since the transfer destination page (DPR = 0x80) is fixed for an initial reset boot, use a reboot routine to boot the instruction memory from 0x8000 to 0xFFFF.

3. Set the SST parameter as 0x0EC0 for serial reboot.

4. When booting in the instruction memory area from 0x0200 to 0x7FFFF, when DPR is 0x80 the destination must be offset by 0x8000, which changes the area to 0x8200 to 0xFFFF.

5. Reboot processing overwrites the DPR setting to 0x00.

APPENDIX A INDEX

A.1 Terminology Index

[R]

[S]

[T]

[U]

[W]

[X]

[Y]

A.2 Register Index

A.2.1 Register name order

[P]

[R]

[S]

[T]

A.2.2 Register symbol order

[P]

[R]

[S]

[T]

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: +1-800-729-9288
+1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Market Communication Dept.
Fax: +49-211-6503-274

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

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

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

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

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

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

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

I would like to report the following error/make the following suggestion:

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❑	❑	❑	❑
Technical Accuracy	❑	❑	❑	❑
Organization	❑	❑	❑	❑