NEC

USER'S MANUAL

ATM-LAN LOW-LEVEL DRIVER

Document No. S11654EJ1V0UM00 (1st edition)
Date Published December 1996 N
Printed in Japan

INTRODUCTION

Readers

This manual is intended for user engineers who wish to understand the functions of the ATM-LAN LSI and design device drivers for it.

Purpose

This manual explains the device driver of the ATM network interface card (NIC) and the functions of the ATM-LAN LSI.

Organization

This manual contains the following information:

- General
- Development Environment
- Hardware control module
- Evaluation software

How to Read This Manual

It is assumed that the readers of this manual have basic knowledge of ATM networks.

To understand the overall functions of the low-level driver
→ Read through this manual in the order of the Table of Contents.

Legend

The following symbols are used throughout this manual.

Note	:	Explanation of items marked with Note in the text
Caution	:	Important information
Remark	:	Supplement
␣	:	1 space
↵	:	Return
Numeric notation	:	Binary number ... xxxx or xxxxB
		Decimal number ... xxxx
		Hexadecimal number ... xxxxH

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

- μPD98401, 98402A

Document Name / Part Number	Brochure	Data Sheet	User's Manual	Application Note
μPD98401	S11294E	S11403E	S11380E	S11441E
μPD98402A		S10835E	S10673E	

- Documents related to tools
  - ATM-LAN Low-Level Driver User's Manual (this manual)
  - ATM Adapter Card for PCI Bus User's Manual (S11655E)

TABLE OF CONTENTS

CHAPTER 1 GENERAL 1

CHAPTER 2 DEVELOPMENT ENVIRONMENT 3
2.1 Hardware Development Environment 3
2.2 Software Development Environment 3
2.3 Operating Environment 3

CHAPTER 3 HARDWARE CONTROL MODULE 5
3.1 Configuration of HCM 5
3.2 HCM Functions 5
3.2.1 PCI bus Setting 5
3.2.2 Setting of interrupt vector 5
3.2.3 Initialization of μPD98401 6
3.2.4 Preparation for transmission/reception 7
3.2.5 Setting of shaper 7
3.2.6 Securing transmit/receive buffer 7
3.2.7 Setting of packet descriptor 8
3.2.8 Setting of transmit VC table 9
3.2.9 Setting receive batch and buffer structure 10
3.2.10 Setting of pool descriptor 11
3.2.11 Setting of receive VC table 12
3.2.12 Setting of receive look-up table 12
3.2.13 Obtaining receive look-up table address 13
3.2.14 Closing transmit/receive channel 13
3.2.15 Termination processing 13
3.2.16 Interrupt service 13
3.3 HCM Command 14
3.4 Relation among Function Calls 16
3.5 Function of Assembler Module 18

CHAPTER 4 EVALUATION SOFTWARE 19
4.1 Function of Evaluation Module 19
4.2 Execution Screen of Evaluation Software 21
4.3 Command Description 22
4.3.1 Reading/writing direct address register 22
4.3.2 Reading/writing indirect address register 23
4.3.3 Reading/writing control memory 23
4.3.4 Reading/writing PHY chip register 24
4.3.5 Dumping system memory 24
4.3.6 Setting shaper 25
4.3.7 Open channel 25
4.3.8 Transmit/receive close channel 26
4.3.9 Setting packet descriptor 27
4.3.10 Setting of transmit VC table 27

4.3.11 Starting transmission 28
4.3.12 Setting transmit data 28
4.3.13 Setting of pool descriptor 28
4.3.14 Setting of RAW cell pool descriptor 29
4.3.15 Setting of receive VC table 29
4.3.16 Enabling reception 29
4.3.17 Disabling reception 30
4.3.18 NOP command 30
4.3.19 AddBatches command 30
4.3.20 Initialize command 30
4.3.21 Transmit buffer, packet descriptor address read command 31
4.3.22 Receive buffer, batch address read command 31
4.3.23 Mailbox address read command 31
4.3.24 RAW cell buffer address read command 32
4.3.25 File open command 32
4.3.26 REM command 32
4.3.27 Quit command 32
4.4 Status of Control Memory 33
4.5 Example of Using Evaluation Software 34

APPENDIX A COMMAND LIST 37

APPENDIX B REGISTER NUMBER LIST 1 (direct address register) 38

APPENDIX C REGISTER NUMBER LIST 2 (indirect address register) 39

CHAPTER 1 GENERAL

This User's Manual explains the functions and specifications of the device driver of the ATM network interface card for the PCI bus (hereafter referred to as "**NIC**") manufactured by ZeitNet/NEC. The NIC uses a μPD98401 ATM chip, and this device driver controls that chip. This device driver actually consists of a hardware control module (hereafter referred to as "**HCM**") that accesses the μPD98401 and evaluation software that evaluates the chip.

[MEMO]

CHAPTER 2 DEVELOPMENT ENVIRONMENT

2.1 Hardware Development Environment

The hardware development environment under which this device driver was developed as follows:

Compaq™ Deskpro® XL560

2.2 Software Development Environment

The software development environment under which this device driver was developed as follows:

Microsoft® Visual C++ ver 1.0

Microsoft MacroAssembler ver 5.0

The device driver source code consists of the following files:

s10hcm.c	: HCM
s10asm.asm	: Assembler of HCM
sarsmp.c	: Evaluation software
ibm_pc.c	: PC identification
s10head.h	: Header
s10func.h	: Header
sarsmp.mak	: Make file

Only s10asm.asm must be assembled with MacroAssembler to create an object file. At assembly time, use the /Mx option (that distinguishes between uppercase and lowercase characters).

The rest of the files and created object file must be compiled and linked with the C compiler, using the make file sarsmp.mak.

2.3 Operating Environment

The hardware environment under which this device driver runs is as follows:

IBM PC/AT™ or compatible machine, or NEC PC-9800 series

ZeitNet/NEC ATM-NIC card

This device driver runs as an application on MS-DOS™ by executing the created EXE file at the command prompt.

[MEMO]

CHAPTER 3 HARDWARE CONTROL MODULE

3.1 Configuration of HCM

HCM consists of two files: s10hcm.c and s10asm.asm. s10hcm.c includes a control block that controls transmission and reception, and commands that access the μPD98401. s10asm.asm executes BIOS call and port input/output.

3.2 HCM Functions

3.2.1 PCI bus Setting

int set_pci (void)

Accesses the PCI configuration space by using the BIOS call of the PCI.

Sets the PCI by obtaining the base address.

(1) Obtains a bus number and a device number from the device ID and vendor ID {f_dev()}.
(2) Writes to the command register that the I/O space can be accessed from the bus number and device number, and that the HCM can operate as a bus master (command 05H).
NIC is mapped to the I/O space {w_config()}.
(3) Obtains the base address from the bus number and device number {r_base()}.
(4) Obtains an interrupt line from the bus number and device number {r_intr()}.

Remark Each module, which is accessing PCI configuration space, is written in assembler.

3.2.2 Setting of interrupt vector

int set_intr_vect (void)

Obtains an interrupt number from an interrupt line, sets an interrupt vector, and clears the interrupt mask. Saves the initial values of both the interrupt vector and interrupt mask, and restores them on termination of the program.

(1) Obtains an interrupt number.
(2) Saves the interrupt vector, and writes a new vector.
(3) Saves the interrupt mask and clears the mask.

3.2.3 Initialization of μPD98401

int init_s10 (void)

Clears the control memory to 0 and executes software reset.

After that, sets the registers, allocates mailboxes, and clears the shaper to 0.

(1) Clearing control memory to 0 {init_CltMem()}

Clears address 0H through 8000H to 0.

(2) Software reset

Executes software reset by writing a value to the SWR register.

(3) Register setting

GMR = 80000020H

IMR = 00000000H

TOS = 00002050H

SMA = 00002040H

PMA = 00002000H

VRR = 000A1FFFH

T1R = 7200

(4) Securing mailbox {init_MailBox()}

Allocates four mailboxes in the system memory for transmission/reception indication.

The size of all the mailboxes (MB_SIZE) is 10K bytes.

(5) Clearing shaper to 0 {init_Shaper()}

Initializes (to 0) I, M, P, C, and Priority of each shaper indicated by a shaper number (i) using the following modules:

- I = 0, M = 0 {shaper_set_im (i, 0, 0)}
- X = 0 {shaper_set_x (i, 0)}
- Y = 0 {shaper_set_y (i, 0)}
- P = 0, C = 0 {shaper set_pc(i, 0, 0)}
- Priority = 0 {shaper set_pri(i, 0)}

3.2.4 Preparation for transmission/reception

int pre_tx_rx (void)

To prepare for transmission/reception, set the RE bit and SE bit of GMR, and set shaper number 0.

(1) Sets the RE and SE bits of GMR.
(2) Clears the interrupt mask.
    Store FFFFFFFFH to the IMR register.
(3) Sets the shaper {set_shaper_reg(&sp)}.
    no = 0, I = 1, M = 2, P = 1, C = 15, Priority =0

3.2.5 Setting of shaper

*int set_shaper_reg (sshp*x)*

Sets the shaper indicated by the shaper number.

The argument is indicated by structure x of sshp type, as follows:

```
typedef struct shaper_param{
    long no;                    : Shaper number
    long pri;                   : Priority
    long I;                     : I
    long M;                     : M
    long P;                     : P
    long C;                     : C
    long enableflag;            : Enable flag
}sshp;
```

3.2.6 Securing transmit/receive buffer

*long *bufalloc (size_t bufsize)*

Allocates a buffer of size (in bytes) indicated by bufsize. The buffer size that can be set during transmission or reception is as follows:

- Transmit buffer size: 0 to 32K bytes
- Receive buffer size : 64 to 32K bytes

As a return value, the address of the buffer allocated is returned. This address is located at a double word boundary.

This module is also used to allocate packet descriptors and batches.

3.2.7 Setting of packet descriptor

*int set_txpd (long *ptxpd, long bufaddr, spp*x)*

Sets the address of the buffer indicated by bufaddr and the parameter indicated by structure x of spp type to the packet descriptor indicated by ptxpd.

The configuration of ptxpd is as follows:

ptxpd = word0
ptxpd + 1 = word1
ptxpd + 2 = word2
ptxpd + 3 = word 3

The structure of ssp type is as follows:

```
typedef struct packetdesc_param {
    long v;          :'1': valid descriptor, '0': blank descriptor
    long dp;         : Descriptor/pointer
    long sm;         : Single/multi-buffer mode
    long clpm;       : CLP mode
    long pti;        : PTI pattern
    long gfc;        : GFC pattern
    long c10;        : CRC-10
    long aal;        : AAL type of cell
    long mb;         : Mailbox number
    long cpcsuu;     : Indication between user machines
    long cpi;        : Common part type indication
    long size;       : Buffer size (0 to 64K bytes)
}spp;
```

The transmit data buffer of the following structure is allocated in system memory by this function.

3.2.8 Setting of transmit VC table

*int set_txvctable (long txvcnum, stxvcp*x, *ptxpd)*

Sets the shaper number, VPI/VCI, and the address of the packet descriptor in the transmit VC table indicated by txvcnum.

The structure of stxvcp type is as follows:

```
typedef struct txvctable_param {
    long shaperno;          : Shaper number
    long vpivci;            : VPI/VCI
} stxvcp;
```

The contents of the transmit VC table are as follows:

word0	: Set to 0.
word1	: Sets L = 1, shaperno and vpivci.
word2	: Set to 0.
word3	: Set to 0.
word4	: Set to 0.
word5	: Set to 0.
word6	: Convert the address (logical address) of the packet descriptor indicated by ptxpd to a physical address for setting. {phy_addr((long)ptxpd)}
word7	: Not used, because a link pointer is set.

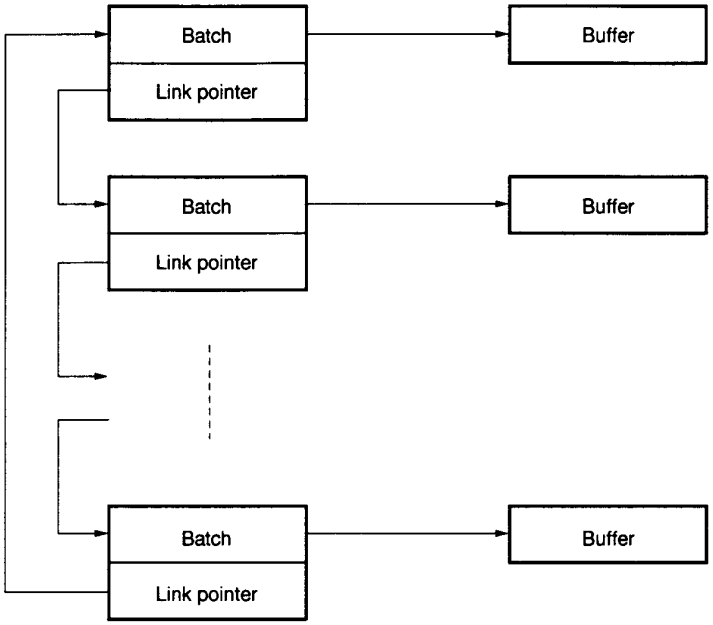
3.2.9 Setting receive batch and buffer structure

*long *set_rxbat (size_t batsize, size_t rxbufsize, int rxbatnum)*

batsize	: Receive batch size (8 bytes)
rxbufsize	: Receive buffer size
rxbatnum	: Number of receive batches

The receive batch and buffer are configured from the above arguments.

"set_rxbat" links the receive buffer (prxbuff[i]) with the receive batch (prxbatch[i]) by using the batch in the link format, and links the last batch with the first batch. As a return value, the address of the first batch is returned. The receive batch and buffer of the following structure is allocated in system memory by this function.



3.2.10 Setting of pool descriptor

*int set_pooldesc(srxvcp*x, long bataddr, srp*y)*

Sets the address and parameter (srp type structure) of the receive batch indicated by bataddr to the pool descriptor indicated by the pool number (contents of srxvcp type structure).

The structure of srxvcp type is as follows:

```
typedef struct rxvctable_param {
    long mb;                    : Mailbox specification
    long poolno;                : Pool number
    long unifo;                 : User information
    long od;                    : OAM cell drop
    long ar;                    : Cell type
    long maxseg;                : Maximum segment
}srxvcp;
```

The structure of srp type is as follows:

```
typedef struct rxpooldesc_param {
    long alert;                 : Alert level
    long bufsize;               : Buffer size
    long batsize;               : Batch size
    long poolbat;               : Number of batches remaining in pool
}srp;
```

3.2.11 Setting of receive VC table

*int set_rxvctable (long rxvcnum, srxvcp*y)*

Sets the parameter indicated by structure y of srxvcp type to the receive VC table indicated by rxvcnum.
The structure of srxvcp type is as follows:

```
typedef struct rxvctable_param {
    long mb;            : Mailbox specification
    long poolno;        : Pool number
    long uinfo;         : User information
    long od;            : OAM cell drop
    long ar;            : Cell type
    long maxseg;        : Maximum segment
} srxvcp;
```

The contents of the receive VC table are as follows:

word0	: Sets mb, poolno, and uinfo.
word1	: Sets od, ar, and maxseg.
word2	: Set to 0.
word3	: Set to 0.
word4	: Set to 0.
word5	: Set to 0.
word6	: Set to 0.
word7	: Not used, because a link pointer is set.

3.2.12 Setting of receive look-up table

*int set_lookuptable (long rxvcnum, srxvpvc*vpvc)*

(1) Obtains the value (VC number) to be set to the look-up table address. At the same time, the look-up table (LUTENABLE = 8000H) is enabled and stores the value to cnum.
(2) Combines (synthesize) the VPI/VCI value and VRR to obtain the synthesized look-up table address {vpvcmap(vpvc)}.
(3) Sets the value (cnum) to the look-up table address[Note].

The look-up table address (vrrvpvc) is the return value.

Note Sets the value to the high-order 16 bits if the LSB of the synthesized look-up table address is "0"; sets the value to the low-order 16 bits if the LSB is "1".

3.2.13 Obtaining receive look-up table address

*int vpvcmap (srxvpvc*vpvc)*

(1) Shifts the VPI by VRR_SHIFT and OR the VPI with VCI.

(2) Masks VRR_MASK.

The return value is the address obtained through synthesis.

3.2.14 Closing transmit/receive channel

[Transmit]

int close_txch (long vcnum)

(1) Issues the DeactivateChannel command. {s10_DeactivateChannel (vcnum, tx)}

(2) Issues the close channel command. {s10_CloseChannel (vcnum, tx)}

[Receive]

int close_rxch (long vcnum, long vrrvpvc)

(1) Disables the look-up table.

(2) Issues the NOP command two times. {s10_Nop()}

(3) Issues the DeactivateChannel command. [s10_DeactivateChannel (vcnum, rx)}

(4) Issues the close channel command. [s10_CloseChannel (vcnum, rx)}

3.2.15 Termination processing

int end_ope(void)

(1) Restores the interrupt vector.

(2) Restores the interrupt mask.

3.2.16 Interrupt service

void interrupt far gsr_check(void)

(1) Masks the interrupt of the μPD98401.

(2) Reads the GSR register.

(3) Issues the EOI command to the interrupt controller.

(4) Updates the mailbox pointer.

(5) Clears the interrupt mask of the μPD98401.

3.3 HCM Command

long s10_ReadDReg(long reg)

Reads a direct address register.

Reads the value of the register indicated by the number **reg**. The value of the register is the return value.

int s10_WriteDReg(long reg, long value)

Writes a direct address register.

Writes the value indicated by **value** to the register indicated by the number **reg**.

long s10_OpenChannel(void)

Open channel command.

Opens a channel. A VC number is the return value.

long s10_CloseChannel(long vcnum, long rt)

Close channel command.

Closes the channel indicated by **vcnum** (VC number) and **rt** (transmit: 0, receive: 1).

A VC number is the return value.

int s10_DeactivateChannel(long vcnum, long rt)

Deactivate channel command.

Deactivates the channel indicated by **vcnum** (VC number) and **rt** (transmit: 0, receive: 1).

int s10_TxReady(long vcnum)

TxReady command.

Issues the TxReady command to the channel indicated by **vcnum** (VC number).

int s10_Nop(void)

NOP command.

Issues the NOP command.

long s10_IndirectAccessR(long tgt, long byte, long address)

Indirect access read command.

Reads the contents indicated by **tgt**, **byte**, and **address**. The contents are returned in a 32 bit value.

int s10_IndirectAccessW(long tgt, long byte, long address, long data)

Indirect access write command.

Writes the value indicated by **data** to the location indicated by **tgt**, **byte**, and **address**.

long s10_ReadIReg(long reg)

Reads the indirect address register.

Reads the value of the register indicated by the number **reg**. The value of the register is the return value.

int s10_WriteIReg(long reg, long value)

Writes the indirect address register.

Writes the value indicated by **value** to the register indicated by the number **reg**.

long s10_ReadMem(long address)

Reads the control memory.

Reads the value of the address of the control memory indicated by **address**. The value of the control memory is the return value.

int s10_WriteMem(long address, long value)

Writes the control memory.

Writes the value indicated by **value** to the control memory indicated by **address**.

long s10_ReadMem_lh(long address)

Reads the low-order bits of the control memory.

Reads the low-order 16 bits of the value of the address of the control memory indicated by **address**. The value of the control memory is the return value.

int s10_WriteMem_lh(long address, long value)

Writes the low-order bits of the control memory.

Writes the value indicated by **value** to the low-order 16 bits of the control memory indicated by **address**.

long s10_ReadMem_uh(long address)

Reads the high-order bits of the control memory.

Reads the high-order 16 bits of the value of the address of the control memory indicated by **address**. The value of the control memory is the return value.

int s10_WriteMem_uh(long address, long value)

Writes the high-order bits of the control memory.

Writes the value indicated by **value** to the high-order 16 bits of the control memory indicated by **address**.

long p10_ReadReg(long address)

Reads a μPD98402A register.

Reads the register of the μPD98402A indicated by **address**. The value of the register is the return value.

int p10_WriteReg(long address, long value)

Writes a μPD98402A register.

Writes the value indicated by **value** to the register of the μPD98402A indicated by **address**.

long phy_addr(long addr)

Obtains a physical address.

Converts the 32-bit address indicated by **addr** to a physical address. The physical address is the return value.

3.4 Relation among Function Calls

```
set_pci              : Sets PCI
  f_dev              : Obtains config space address
  w_config           : Sets command register
  r_base             : Obtains base address
  r_intr             : Obtains interrupt line

set_intr_vect        : Sets interrupt vector

init_s10             : Initializes μPD98401
  init_CtlMem        : Clears control memory to 0
  init_MailBox       : Allocates mailbox
  init_Shaper\       : Clears shaper to 0
    shaper_set_im    : Sets IM
    shaper_set_x     : Sets x
    shaper_set_y     : Sets y
    shaper_set_pc    : Sets PC
    shaper_set_pri   : Sets Priority
```

pre_tx_rx	: Preparation for transmission/reception
set_shaper_reg	: Sets shaper
shaper_set_im	: Sets IM
shaper_set_x	: Sets X
shaper_set_y	: Sets y
shaper_set_pc	: Sets PC
shaper_set_pri	: Sets Priority
shaper_set_enable	: Sets enable
bufalloc	: Allocates buffer
send_data	: Transmission processing
set_txpd	: Sets packet descriptor
set_txvctable	: Sets transmit VC table
s10_TxReady	: TxReady command
set_rxbat	: Sets receive data structure
receive_data	: Reception processing
set_pooldesc	: Sets pool descriptor
set_rxvctable	: Sets receive VC table
set_lookuptable	: Sets lookup table
vpvcmap	: Shifts and masks VPI/VCI
close_txch	: Closes transmit channel
close_rxch	: Closes receive channel
end_ope	: Termination processing
gsr_check	: Interrupt processing

3.5 Function of Assembler Module

The assembler module (s10asm.asm) executes the BIOS call of the PCI bus and 32-bit input/output through ports.

- BIOS call of PCI bus
  When software interrupt 1AH is generated with values set to the CPU register, a BIOS call is executed. As a result, information such as a base address is stored to the register. If the BIOS call fails, a carry flag is set.

- 32-bit I/O
  32-bit input/output is realized by using the in and out instructions of the assembler. The μPD98401 is actually accessed in this way.

Only the above two processes are described with assembler.

CHAPTER 4 EVALUATION SOFTWARE

4.1 Function of Evaluation Module

int set_drv(void)

(1) Sets PCI {set_pci()}
(2) Sets interrupt vector {set_intr_vect()}
(3) Initializes μPD98401 {init_s10()}
(4) Enables SE and RE bits
(5) Enables interrupt
(6) Allocates system memory area
    1. Transmit buffer (ptxbuf) : 20K bytes
    2. Packet descriptor (prxbat) : 4100 bytes
    3. Receive buffer (prxbat) : 10 batches, each 11000-byte buffer
    4. RAW cell receive buffer (prawbuf) : 64 bytes
    5. RAW cell receive batch (prawbat) : 16

char *prompt(char*pletter)

Displays 'my:\\>' and prompt, and allocates the input value to pletter. The input value is returned as a return value.

int cmd_str(char*pletter)

Looks up and codes the input character string (cmd). The codes (retcod) are as follows:

Command (cmd)	Code (retcod)	Command (cmd)	Code (retcod)	Command (cmd)	Code (retcod)
rdreg	1	opcha	12	setpool	23
wdreg	2	clchat	13	setrxvc	24
rireg	3	clchar	14	rxenbl	25
wireg	4	nop	15	rxdis	26
rcmem	5	setpd	16	lkmbox	27
wcmem	6	settxvc	17	addbat	28
dm	7	rp10	18	rawpool	29
lkbuft	8	wp10	19	lkraw	30
lkbufr	9	txrdy	20	quit	100
@	10	init	21	rem	101
setshap	11	txdata	22		

long *value_str(char*pletter, long*pvalue)

Stores data.

The data (pvalue) is the return value.

int f_input(char*fname, charfilestr)**

Reads from a file.

int call_command(int cmd, long*pvalue, char*pletter)

Performs processing corresponding to the code (retcod).

4.2 Execution Screen of Evaluation Software

To execute the evaluation software, input sarsmp ↵ in response to the command prompt of MS-DOS. When the software is executed, the following message is displayed.

```
**************************************************
  sar chip samplesoftware v1.0 1995
  1st systems,NEC
**************************************************
my:\\>
```

Input a command and data after prompt "my:\\>".

Remarks 1. Only lowercase characters can be input as commands.

2. All input data and displayed data are in hexadecimal numbers.

3. The transmit/receive buffer is fixed.

- One transmit buffer, 10 packet descriptors
- Ten receive buffers, 10 batches (one buffer per batch)
- Nine RAW cell receive buffers, three batches (three buffers per batch)
- If VC tables, packet descriptors, and pool descriptors violate the above fixed settings, the personal computer may hang up.

4. Maximum transmit buffer size (TXBUFSIZE) : 20K bytes
Maximum packet descriptor size (PDSIZE) : 4K bytes
Maximum receive buffer size (RXBUFSIZE) : 11000 bytes
Receive batch (RXBATNUM) : 10
Receive batch size (RXBATSIZE) : 8 bytes
Maximum number of open channels (OPNUM) : 50
Maximum packet descriptors (PDNUM) : 10

4.3 Command Description

4.3.1 Reading/writing direct address register

[Read]

"rdreg" reads one direct address register indicated by a register number (registers are numbered 0, 4, 8, c, 10, 14, and so on).

<Execution>

my:\\>rdreg␣x↵

x: register number

<Display>

data = xxxxxxxx

[Write]

"wdreg" writes a value to the direct address register indicated by a register number.

<Execution>

my:\\>wdreg␣x1␣x2↵

x1: register number

x2: value

<Display>

ok!

4.3.2 Reading/writing indirect address register

[Read]

"rireg" reads one indirect address register indicated by a register number.

<Execution>

my:\\>rireg␣x↵

x: register number

<Display>

date=xxxxxxxx

[Write]

"wireg" writes a value to the indirect address register indicated by a register number.

<Execution>

my:\\>wireg␣x1␣x2↵

x1: register number

x2: value

<Display>

ok!

4.3.3 Reading/writing control memory

[Read]

"rcmem" reads the control memory indicated by an address.

<Execution>

my:\\>rcmem␣x1␣x2↵

x1: address

x2: number of words

<Display> (if x2 = 5)

xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx

xxxxxxxx

[Write]

"wcmem" writes a value to the control memory indicated by an address.

<Execution>

my:\\>wcmem␣x1␣x2↵

x1: address

x2: value

<Display>

ok!

4.3.4 Reading/writing PHY chip register

[Read]

"rp10" reads a register of the μPD98402A.

<Execution>

my:\\>rp10␣x↵

x: register number

<Display>

data = xxxxxxxx

[Write]

"wp10" writes a value to a register of the μPD98402A.

<Execution>

my:\\>wp10␣x1␣x2↵

x1: register number

x2: value

<Display>

ok!

4.3.5 Dumping system memory

"dm" dumps the system memory indicated by an address.

<Execution>

my:\\>dm␣x1␣x2↵

x1: address

x2: number of words

<Display> (if x2 = 5)

xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx

xxxxxxxx

4.3.6 Setting shaper

"setshap" sets the shaper indicated by a shaper number.

<Execution>

my:\\>setshap␣x1␣x2␣x3␣x4␣x5␣x6↵

x1: shaper number

x2: priority

x3: I

x4: M

x5: P

x6: C

<Display>

ok!

Remark The shaper is enabled when it is set.

4.3.7 Open channel

"opcha" opens one VC. The address of the VC is saved to a number specified by the number of the open channel.

<Execution>

my:\\>opcha␣x↵

x: number of open channel (The address of the VC is saved to this.)

<Display>

xxxx (The address of the VC)

4.3.8 Transmit/receive close channel

[Transmit]

"clchat" executes the close channel command to the transmit channel specified by the number of the close channel (the number has been specified when the channel was opened).

<Execution>

my:\\>clchat␣x ↵

x: number of close channel

<Display>

xxxx (VC number)

[Receive]

"clchar" executes the close channel command to the receive channel specified by the number of the close channel (the number has been specified when the channel was opened).

<Execution>

my:\\>clchar␣x ↵

x: number of close channel

<Display>

xxxx (VC number)

4.3.9 Setting packet descriptor