LAN Driver
Design & Interface Reference Guide
For
AMD79C973
(PCI bus Interface)

Document revision history

14th Dec, 2002	Document Created

MiSPO Co., Ltd.
http://www.mispo.co.jp

Table of Contents

1 Introduction

This document is intended to give detailed design and interface information about AMD79C973 Ethernet driver for NORTi Network Protocol stack.

The LAN driver is designed for AMD79C973 Ethernet controller with PCI bus interface.

The detailed information about NORTi OS internal functions and features can be obtained from “NORTi User’s Guide”. PCI driver interface information is explained in this document. However for more detailed information about PCI driver, please refer to PCI driver document. Information regarding NORTi Network Protocol stack is available with “Network Protocol Stack User’s Guide”.
For AMD79C973 Ethernet controller register addresses and functional details, please refer to corresponding controller interface and application manual.

2 Controller Details and Specifications

The low-power high-performance AMD79C973 offers full Ethernet control with integrated 10/100Base PHY. The major features of AMD79C973 Ethernet controller are described below.
For more details about the AMD79C973 hardware interface and internal register addressing, please refer to AMD79C973 Product Manual book.

2.1 Controller Features supported by Driver

- Support for 10BaseT, 100Base-TX and 100Base-FX (IEEE 802.3 standard) interface.
- 32-bit PCI bus interface with host CPU.
- Supports both half and full duplex operation.
- Support for both Little Endian and Big Endian memory system.
- Supports variable Receiver and Transmitter buffer size.
- Automatic data padding option for transmitted frames.
- Support for software debug using serial RS232 interface.
- Support for Internal Loopback mode operation.

2.2 Unsupported Features

- Does not support Receiver and Transmitter descriptor buffer size <1514 bytes.
- Does not support external PHY interface.
- Does not support external EEPROM interface.

3 File Contents

Source and header files used for defining AMD79C973 Ethernet I/O driver are listed below.

File Name	*Description*
am79c973.c	LAN driver program source file. All source code and driver operational functions are described in this file.
nonecfg.c	Network Protocol Configuration file for sample program. NORTi Network Protocol configuration parameters such as TCP/IP Communication end points, maximum network protocol tasks, node IP address, Ethernet address etc. are defined in this code file.
am79c973.h	LAN driver and controller specific header file. AMD79C793 internal registers and control bit definitions are described in this header file.
nonethw.h	Board specific hardware definitions for network driver. This header file defines the hardware interface for AMD79C973 with host CPU. The file also describes LAN controller I/O access functions and interrupt handler definition.

The Network Protocol Stack library files are provided with the NORTi OS program. Proper library file as per memory system byte order type (Little Endian/Big Endian) should be selected and linked with main project.
NORTi Network Protocol Stack source files are described in..\NORTi\SRC folder while Network application programs are described in ..\NORTi\NETSMP directory.

4 PCI Driver Interface

AMD79C973 Ethernet controller operates using 32-bit PCI bus interface with host CPU. PCI bus interface functions used by Ethernet driver are described below.

4.1 PCI Bus Interface functions

PCI bus Interface functions utilized in Ethernet driver code is listed as below. These functions should be defined separately in PCI driver source code.

- *pci_ioread_byte*
- *pci_iowrite_byte*
- *pci_ioread_word*
- *pci_iowrite_word*
- *pci_ioread_long*
- *pci_iowrite_long*
- *addr_pci_to_cpu*
- *addr_cpu_to_pci*
- *htopci*

4.1.1 pci_ioread_byte

Declaration:	void pci_ioread_byte(UW pci_io_addr, UB * pci_io_data, UW io_lock);
Input:	pci_io_addr = location address in PCI area
	pci_io_data = Memory pointer where byte data will be stored.
	io_lock = memory lock option (Normally 0x0)
Output:	none
Description:	This function reads 8-bit data from the PCI memory space.

4.1.2 pci_iowrite_byte

Declaration:	void pci_iowrite_byte(UW pci_io_addr, UB pci_io_data, UW io_lock);
Input:	pci_io_addr = location address in PCI area
	pci_io_data = 8-bit data to be stored to PCI memory space.
	io_lock = memory lock option (Normally 0x0)
Output:	none
Description:	This function writes 8-bit data to the PCI memory space.

4.1.3 pci_ioread_word

Declaration:	void pci_ioread_word(UW pci_io_addr, UH * pci_io_data, UW io_lock);
Input:	pci_io_addr = location address in PCI area
	pci_io_data = Memory pointer where 16-bit data will be stored.
	io_lock = memory lock option (Normally 0x0)
Output:	none
Description:	This function reads 16-bit data from the PCI memory space.

4.1.4 pci_iowrite_word

Declaration:	void pci_iowrite_word(UW pci_io_addr, UH pci_io_data, UW io_lock);
Input:	pci_io_addr = location address in PCI area
	pci_io_data = 16-bit data to be stored to PCI memory space.
	io_lock = memory lock option (Normally 0x0)
Output:	none
Description:	This function writes 16-bit data to the PCI memory space.

4.1.5 pci_ioread_long

Declaration:	void pci_ioread_long(UW pci_io_addr, UW * pci_io_data, UW io_lock);
Input:	pci_io_addr = location address in PCI area

	pci_io_data = Memory pointer where 32-bit data will be stored.
	io_lock = memory lock option (Normally 0x0)
Output:	none
Description:	This function reads 32-bit data from the PCI memory space.

4.1.6 pci_iowrite_long

Declaration:	void pci_iowrite_byte(UW pci_io_addr, UW pci_io_data, UW io_lock);
Input:	pci_io_addr = location address in PCI area
	pci_io_data = 32-bit data to be stored to PCI memory space.
	io_lock = memory lock option (Normally 0x0)
Output:	none
Description:	This function writes 32-bit data to the PCI memory space.

4.1.7 addr_pci_to_cpu

Declaration:	UW addr_pci_to_cpu(UW pci_addr);
Input:	pci_addr = location address in PCI area
Output:	returns back corresponding 32-bit address in system memory area.
Description:	This function returns 32-bit address from system memory space, which is mapped, to PCI memory space.

4.1.8 addr_cpu_to_pci

Declaration:	UW addr_cpu_to_pci(UW cpu_addr);
Input:	cpu_addr = location address in System memory area
Output:	returns back corresponding 32-bit address in PCI memory area.
Description:	This function returns 32-bit address from PCI memory space, which is mapped, to System memory space.

4.1.9 htopci

Declaration:	UW htopci(UW src_data);
Input:	src_data = 32-bit data in Little Endian memory format.
Output:	returns back corresponding 32-bit data in current memory system format.
Description:	This function converts 32-bit input data to current memory system format.

5 Driver Control Structures and Parameters

Ethernet driver is designed to operate with a single structure, which includes driver run-time control parameters and status parameters. *LAN_DEVICE* data structure along with sub structures *RX_STS*, *TX_STS* and *LAN_CFG* are defined as below.

```
typedef struct LAN_DEVICE {
   Lan_Cfg   lan_cfg;          LAN driver configuration settings
   Rx_Sts    rx_sts;           Receiver controller status
   UW        rx_start_ptr;     Start address of received packet
   UW        rx_stop_ptr;      Stop address of received packet
   UW        rx_pkt_size;      Byte Count / Size of received packet
   UW        rx_read_ptr;      Read pointer to current byte in Rx Data buffer
   Tx_Sts    tx_sts;           Transmit controller status
   UW        tx_start_ptr;     Packet start address in Tx Buffer
   UW        tx_pkt_size;      Byte length / size of packet to be transmitted
   UW        tx_write_ptr;     Position of current byte in Tx buffer
   UW        lnk_sts_old;      LINK/Media status backup
}Lan_Device;
```

Lan_Device is the main data structure, which included LAN driver status and control parameters.

```
typedef struct RX_STS {
    BOOL    rx_on;               0 = Receiver controller disabled.
                                 1 = Receiver Controller enabled.
    BOOL    rx_int_flag;         0 = Waiting for Rx interrupt.
                                 1 = Rx interrupt acknowledged.
    BOOL    rx_flag;             0 = Waiting for packet reception.
                                 1 = Packet received.
    BOOL    rx_error_flag;       0 = Data reception OK.
                                 1 = Data reception error.
    BOOL    rx_overflow_flag;    0 = Rx controller no-overflow detected.
                                 1 = Receive ring overflow detected.
    BOOL    phy_addr_match;      0 = No physical address match found.
                                 1 = Packet with physical address match detected.
    BOOL    broadcast_match;     0 = No broadcast address match found.
                                 1 = Packet with broadcast address match detected.
}Rx_Sts;
```

Rx_Sts data structure describes the Receiver control status parameters.

```
typedef struct TX_STS {
    BOOL    tx_on;               0 = Transmit controller disabled.
                                 1 = Transmit Controller enabled.
    BOOL    tx_int_flag;         0 = Waiting for Tx interrupt.
                                 1 = Tx interrupt acknowledged.
    BOOL    tx_flag;             0 = Last Packet Transmission over.
                                 1 = Packet transmission busy.
    BOOL    tx_error_flag;       0 = Packet transmission OK.
                                 1 = Packet transmission error.
}Tx_Sts;
```

Tx_Sts data structure describes the Ethernet packet Transmitter status parameters.

```
typedef struct LAN_CFG {
    BOOL    sel_autoneg;         0 = Auto-Negotiation disabled
                                 1 = Auto-Negotiation enabled
    BOOL    sel_speed;           0 = 10 Mbps speed selection
                                 1 = 100 Mbps speed selection
    BOOL    sel_duplex;          0 = Half Duplex mode selection
                                 1 = Full Duplex mode selection
    BOOL    sel_loopback;        0 = Normal operation, No loop-back mode
                                 1 = Internal loop-back mode
    BOOL    rcv_all_pkt;         0 = Disable all packets reception.
                                 1 = Receive packets with all physical address
    BOOL    rcv_mcast_pkt;       0 = Disable multicast packets reception.
                                 1 = Receive packets with multicast address match
}Lan_Cfg;
```

Lan_Cfg data structure describes the Ethernet Driver PHY configuration parameters.

Driver allocates some memory from system memory area for receiver and transmitter buffer. This memory needs to be in non-Cache area. Driver supports multiple descriptors for receiver and transmitter. The size of each descriptor and number of descriptors is user programmable.

6 Ethernet Driver Configuration Settings

User can configure Ethernet driver to operate in desired mode by using various configuration macros. Supported configuration macros are described below.

6.1 Setting I/O base address.

Ethernet controller I/O base address is defined by pre-processor macro LANC_IO_BASE. It is declared as pre-compilation macro as shown in following example.

(Example) To set LAN controller base address.
(SH Hitachi Compiler)

```
  shc <compiler option> -def=LANC_IO_BASE=0x04000000 am79c973.c
```

I/O base address can also be configured in “nonethw.h” hardware interface definition file.

6.2 Setting Interrupt handler

Interrupt handler for Ethernet controller is defined using interrupt line number and interrupt priority value. User must configure driver for proper values of interrupt line number and priority. The pre-compilation macro for interrupt priority is IP.

(Example) To set Interrupt handler properties.
Configure driver for interrupt priority level 6.
(SH Hitachi C Compiler)

```
            shc <compiler option> -def=IP=6 am79c973.c
```

6.3 Setting Memory Byte Order type

Driver is configurable to either of little or big endian memory system. Default setting for byte order is Little Endian type. The pre-compilation macro for memory byte order selection is set as shown in this example.

(Example) To set memory byte order.
(SH Hitachi C Compiler)

```
            shc (compiler options) -def=LITTLE_ENDIAN am79c973.c
            shc (compiler options) -def=LITTLE_ENDIAN nonecfg.c
                      For Little endian memory system
                                          OR
            shc (compiler options) -def=BIG_ENDIAN am79c973.c
            shc (compiler options) -def=BIG_ENDIAN nonecfg.c
                      For Big Endian memory system
```

6.4 Setting Receiver and Transmitter buffer size

AM79C973 ethernet controller receiver and transmitter buffer is configured as ring of packet buffers. User can configure receiver and transmitter packet buffer size along with count of buffers in a ring. User selective macros are defined as shown below.

RX_SIZE	- Buffer size for single Rx descriptor(bytes).
RX_RING_COUNT	- Number of descriptors/buffers in a receiver ring.
TX_SIZE	- Buffer size for single Tx descriptor(bytes).
TX_RING_COUNT	- Number of descriptors/buffers in a transmitter ring.

Total memory occupied by receiver buffer = RX_RING_COUNT x RX_SIZE. Total memory occupied by transmitter buffer = TX_RING_COUNT x TX_SIZE. The configurable values for RX_RING_COUNT and TX_RING_COUNT are { 1/ 2/ 4/ 8/ 16/ 32/ 64/ 128/ 256/ 512).
Buffer count in ring is 1 when RX_RING_COUNT or TX_RING_COUNT = 0.
Please note that values other than above may give improper results.

In case not defined by user, the default configuration is as as shown below.

```
  RX_SIZE            = 0x0600 (1536 bytes).
  RX_RING_COUNT      = 16 (16 data buffers in receiver ring)
  TX_SIZE            = 0x0600 (1536 bytes).
  TX_RING_COUNT      = 4 (4 data buffers in transmitter ring)
```

Following command line compilation code illustrates receiver and transmitter buffer size configuration.
{Example with Hitachi C compiler}

```
  shc (compiler options) -def=RX_SIZE=2048, RX_RING_COUNT=32 AM79C973.c
                    ;Configure Rx Buffer size = (2048*32 = 65536 bytes).
                    ;Rx ring configured as 32 number of buffers of size 2048 bytes each.
          ----- next example ----
  shc (compiler options) -def=TX_SIZE=1550, TX_RING_COUNT=0 AM79C973.c
                    ;Configure Tx Buffer size = (1550*1 = 1550 bytes).
                    ;Tx ring configured as single buffer of size 1550 bytes.
          ----- next example ----
  shc (compiler options) -def=RX_SIZE=1800 AM79C973.c
                    ;Configure Rx Buffer size = (1800*16 = 28800 bytes).
                    ;Rx ring configured as 16 number (default) of buffers of size
                     1800 bytes each.
          ----- next example ----
  shc (compiler options) -def=RX_RING_COUNT=8 AM79C973.c
                    ;Configure Rx Buffer size = (1536*8 = 12288 bytes).
                    ;Rx ring configured as 8 number of buffers of individual size
                     1536 bytes(default) each.
          ----- next example ----
  shc (compiler options) -def=TX_SIZE=1500 AM79C973.c
                    ;Configure Tx Buffer size = (1500*4 = 6000 bytes).
                    ;Tx ring configured as 4 number (default) of buffers of size
                     1500 bytes each.
          ----- next example ----
  shc (compiler options) -def=TX_RING_COUNT=1 AM79C973.c
                    ;Configure Tx Buffer size = (1536*1 = 1536 bytes).
                    ;Tx ring configured as single buffer of size 1792 bytes(default).
          ----- next example ----
  shc (compiler options) AM79C973.c
                    ;Configure Rx Buffer size = (1536*16 = 24576 bytes). --(default)
                    ;Configure Tx Buffer size = (1536*4 = 6144 bytes). --(default)
```

User need to be careful about not to specify TX_SIZE or RX_SIZE values less than maximum packet size. Besides this, receiver and transmitter data buffer should not be in Cache area.

6.5 Media Speed selection

Driver provides macro option SPEED_100M to select media speed i.e. 10Mbps or 100Mbps.

(Example) To change Media speed.
(SH Hitachi C Compiler)

```
          shc <compiler option> -def= SPEED_100M am79c973.c
                                       For 100Mbps operation.
                              OR
          Shc <compiler option> am79c973.c
                                       For 10Mbps operation (default)
```

6.6 Duplex mode selection

Either of Half duplex or full duplex operation mode can be configured with the help of pre-compilation macro FULL_DUPLEX. Please see following command line compilation example for setting duplex mode.

(Example) To change Duplex mode.
(SH Hitachi C Compiler)

```
shc <compiler option> -def=FULL_DUPLEX am79c973.c
                                For Full duplex operation.
                        OR
Shc <compiler option> am79c973.c
                                For Half duplex operation (default)
```

6.7 Auto Negotiation mode selection

AM79C973 controller internal PHY Media support Auto-negotiation and smooth changeover from 10Mbps to 100Mbps media or vice versa is possible. Auto-negotiation will be enabled if specified by pre-compilation macro as shown in following compilation command code.

(Example) To change Duplex mode.
(SH Hitachi C Compiler)

```
shc <compiler option> -def= AUTO_NEG am79c973.c
                                For Auto-negotiation mode.
```

By default Auto-negotiation mode is not configured.
Please note that when in Auto-negotiation mode, driver will ignore media speed and duplex mode selection macro i.e. SPEED_100M and FULL_DUPLEX respectively. Driver will configure the media speed and duplex mode automatically as per the available media options.

6.8 Loopback mode selection

Internal loopback mode is set if user compile driver code with pre-compilation macro LOOPBACK. In other case normal mode is selected, which is default.
In case LOOPBACK mode is selected, Auto-negotiation setting is ignored.

(SH Hitachi C Compiler)

```
shc <compiler option> -def= LOOPBACK am79c973.c
                                For internal loopback mode.
                        OR
Shc <compiler option> am79c973.c
                                For normal operation.
```

6.9 Setting receiver packet filter

Driver provides user selective option for filtering packet reception. The parameter setting is done as shown below. If none is specified, default configuration is to accept all packets with matching physical address and broadcast packets.

(SH Hitachi C Compiler)

```
shc <compiler option> -def= RCV_ALL am79c973.c
        Configure driver to receive all packets with valid physical address.
                        OR
shc <compiler option> -def= RCV_MCAST am79c973.c
        Configure driver to receive all packets matching multicast address.
```

6.10 Setting Debug mode

It is possible to configure Ethernet driver in software debug mode by specifying pre-compilation macro DEBUG. The Debug code is compiled only when DEBUG macro is set hence it reduces the code size when debug option is not selected.
In debug mode current media status, received packet status and transmission status is send to serial console defined by channel number DBG_CH. User need to specify debug serial console number along with DEBUG macro. Default value for DBG_CH is 0. Serial console initialization is not performed by LAN driver. Instead serial console initialization need to be done before starting LAN driver in debug mode. The command line compilation code for debug configuration is as shown below.

(SH Hitachi C Compiler)

```
shc <compiler option> -def= DEBUG, DBG_CH=1 am79c973.c
        Configure driver in debug mode with serial channel 1 as Debug console.
```

Please note that while in debug mode driver speed performance will be lowered. For better performance please do not use Debug option in normal running. Debug option is not selected by default.

6.11 Summery of Configuration macros

The list of configuration macros and their description is summarized below.

Macro Name	Description	Default Setting
LANC_IOBASE	Controller I/O register offset address	undefined
IP	Interrupt priority value	12
SPEED_100M	Select 100Mbps Media speed.	False (10 Mbps)
FULL_DUPLEX	To set Full Duplex LAN operation mode.	False (Half-Duplex)
LOOPBACK	Set Internal loopback mode.	False (Normal mode)
AUTO_NEG	Set Auto Negotiation mode	True
RX_RING_COUNT	Set Number of Descriptors in Receiver Ring	0x10 (16 no.)
TX_RING_COUNT	Set Number of Descriptors in Transmitter Ring	0x04 (4 no.)
RX_SIZE	Size of each Receiver Descriptor Buffer (Bytes)	1536 Bytes
TX_SIZE	Size of each Transmitter Descriptor Buffer (Bytes)	1536 Bytes
BIG_ENDIAN	Set Big Endian byte order for memory system	False
LITTLE_ENDIAN	Set Little Endian byte order for memory system	True
RCV_ALL	Receive all type packets with valid physical address.	False
RCV_MCAST	Receive all multicast packets.	False
DEBUG	Set Driver in debug mode operation.	False
DBG_CH	Serial I/O channel for Debug mode	0

7 Functional Description

Ethernet driver functional flow is managed by IP Send Task (ip_snd_tsk) and IP Receive Task (ip_rcv_tsk) from IP layer control. Following is the list of driver functions, which are directly called by IP layer control tasks.

- ER lan_ini (UB *macaddr)
- ER lan_wai_rcv (TMO tmout)
- ER lan_wai_snd (TMO tmout)
- ER lan_get_len (UH *len)
- ER lan_get_pkt (void *buf, int len)
- ER lan_get_end (void)
- ER lan_skp_pkt (int len)
- ER lan_set_len (int len)
- ER lan_put_pkt (const void *data, int len)
- ER lan_put_dmy (int len)
- ER lan_put_end (void)

These functions are briefly described in the following text.

7.1 lan_ini

Declaration

ER lan_ini (UB * macaddr);

Input

Address pointer to Ethernet MAC address.

Output

ER = E_OK	(Normal completion)
ER = E_PAR	(Parameter Error during Interrupt handler definition)

Updated Ethernet MAC address is also returned back through pointer.

Description

This function performs the software initialization of LAN controller and driver control parameters.

The operational steps followed during Driver Initialization are listed below.

- ➢ Non-Cacheable Memory resource is allocated to receiver and transmitter descriptors and data buffers.
- ➢ Set LAN controller AMD79C973 to 32-bit access mode.
- ➢ All Driver control parameters and structures are reset. Receiver and Transmitter buffer is cleared to zero data.
- ➢ Initialize LAN controller. Set Receiver and Transmitter control properties.
- ➢ Define Interrupt handler for Ethernet controller.
- ➢ Start Receiver and Transmitter operations.
- ➢ Enable Interrupt line for Ethernet controller operations.

7.2 lan_wai_rcv

Declaration

ER lan_wai_rcv (TMO tmout);

Input

Time to wait till new data is received.

Output

ER = E_OK	(Normal completion)
ER = E_LINK	(Link failure error)
ER = E_TMOUT	(Returned back due to Timeout error)
ER = E_RXERR	(Error during new packet reception)

Description

This function will wait till Ethernet controller receives new packet. The wait time can be controlled by specification of timeout time as input argument. Before waiting for reception, the link status is checked. If there is no link connection then E_LINK error code is returned back. However, when driver operate in loopback mode, link status is not checked.

Depending on the status of receiver interrupt mode flag (*lan_device.rx_sts.rx_int_flag*), controller will operate in either polling mode or interrupt mode. Please note that driver does not provide user selection flag for polling or interrupt mode of operation. Receive ready flag (*lan_device.rx_sts.rx_flag*) is updated after new data is detected in receiver buffer. This flag is turned off after acknowledgement of new data packet. If erroneous packet is received then E_RXERR error code is returned back.

The routine will terminate reception waiting after timeout period is exhausted.

7.3 lan_wai_snd

Declaration

ER lan_wai_snd (TMO tmout);

Input

Time to wait till new data is received.

Output

ER = E_OK	(Normal completion)
ER = E_TMOUT	(Returned back due to Timeout error)
ER = E_LINK	(Link failure error)

Description

This function will wait till Ethernet controller is ready for new packet transmission. This function will check for link status (except in loopback mode) and will return back E_LINK error code if link failure. Transmit ready status can not be checked till new packet data length is not specified. E_TMOUT error code will be returned back when transmit controller is not ready within the due time.

7.4 lan_get_len

Declaration

ER lan_get_len (UH *len);

Input

Data pointer address where received length information is to be stored.

Output

Length of received data packet in bytes.

ER = E_OK	(Normal completion)
ER = E_RXERR	(Error during reception)

Description

This function reads the total number of data bytes received in new packet. The packet length information is also available in Receiver Status data structure (*lan_device.rx_pkt_size*). If new received packet is not valid or erronous data packet received then error code is returned back.

Receive error status flag (*lan_device.rx_sts.rx_err_flag*) is raised if either packet status or count is not valid.

7.5 lan_get_pkt

Declaration

ER lan_get_pkt (void *buf, int len);

Input

void *buf	buffer pointer to store packet data

int len	number of bytes to be read

Output

ER = E_OK	(Normal completion)

Description

The specified length (*len*) of data bytes from current packet are read and transferred to main memory via buffer pointer (*buf*). Receiver buffer read pointer (*lan_device.rx_start_ptr*) is pointed to the current packet in receiver ring buffer whereas (*lan_device.rx_read_ptr*) points to the unread data in current packet.

7.6 lan_get_end

Declaration

ER lan_get_end (void);

Input

None.

Output

ER = E_OK	(Normal completion)

Description

The purpose of this function is to get end point of the current received packet. Nothing is done for this function.

7.7 lan_skp_pkt

Declaration

ER lan_skp_pkt (int len);

Input

Int len	size of the packet bytes to be skipped.

Output

ER = E_OK	(Normal completion)

Description

This function is similar to lan_get_pkt except there is no data read.

7.8 lan_set_len

Declaration

ER lan_set_len(int len);

Input

Int len	size of the packet bytes to be transmitted.

Output

ER = E_OK	(Normal completion)

Description

The data length of the packet to be transmitted along with transmit data buffer pointer address is specified to the controller.

7.9 lan_put_pkt

Declaration

ER lan_put_pkt (const void *data, int len);

Input

const void *data	buffer containing packet data
Int len	size of the packet bytes to be transmitted.

Output

ER = E_OK	(Normal completion)

Description

The data buffer of specified length (*len*) from main memory pointed by buffer pointer (const void *data) is uploaded to transmitter buffer.

7.10 lan_put_dmy

Declaration

ER lan_put_dmy (int len);

Input

Int len	size of the dummy data bytes to be transmitted.

Output

ER = E_OK	(Normal completion)

Description

This function is called whenever transmit packet length is smaller than 64 bytes. Since controller does data padding automatically, there is no dummy data upload to transmitter buffer.

7.11 lan_put_end

Declaration

ER lan_put_end (void);

Input

none.

Output

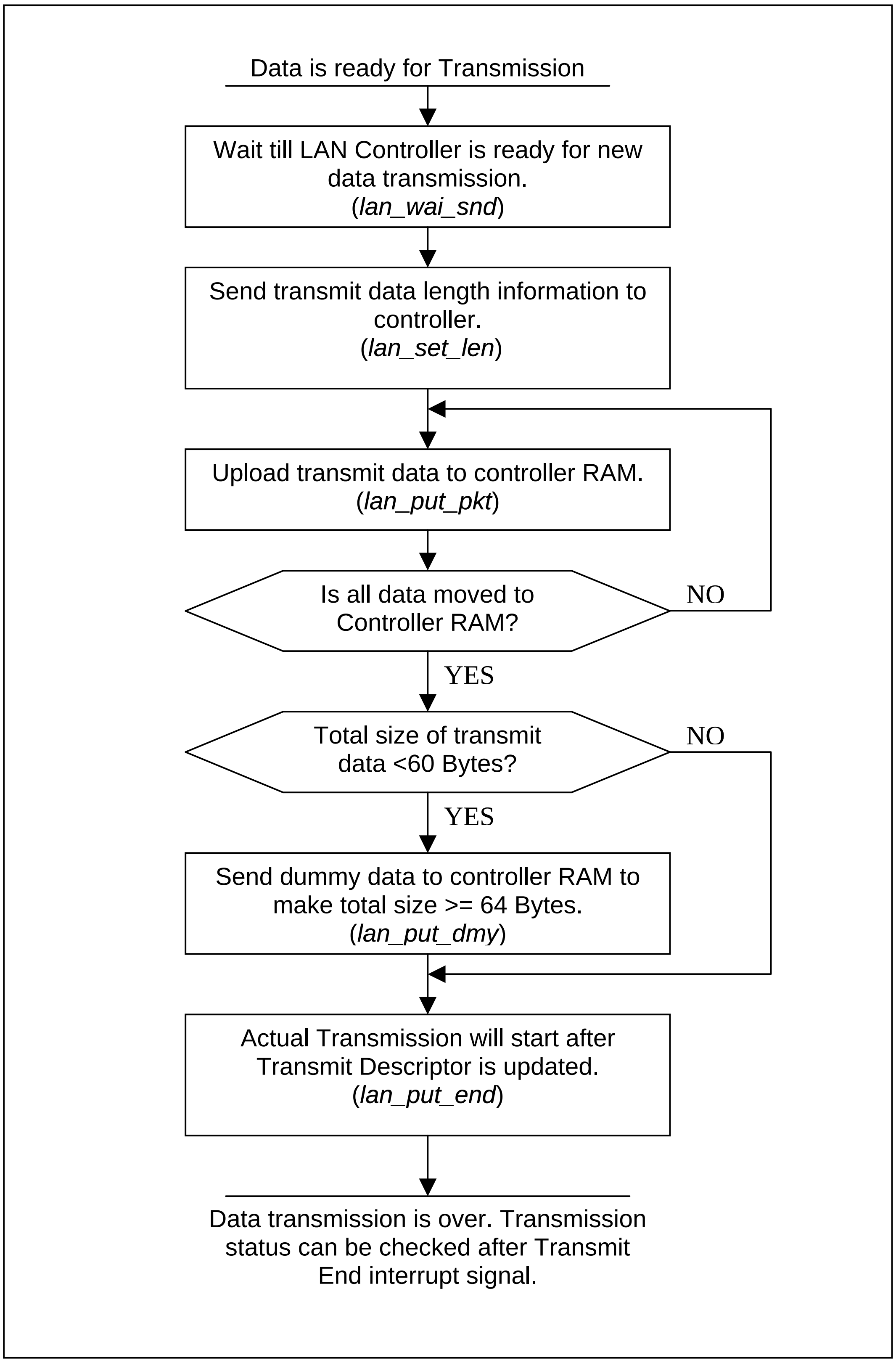
ER = E_OK	(Normal completion)

Description

The function starts actual transmission. All data to be transmitted is already moved to controller internal RAM area.

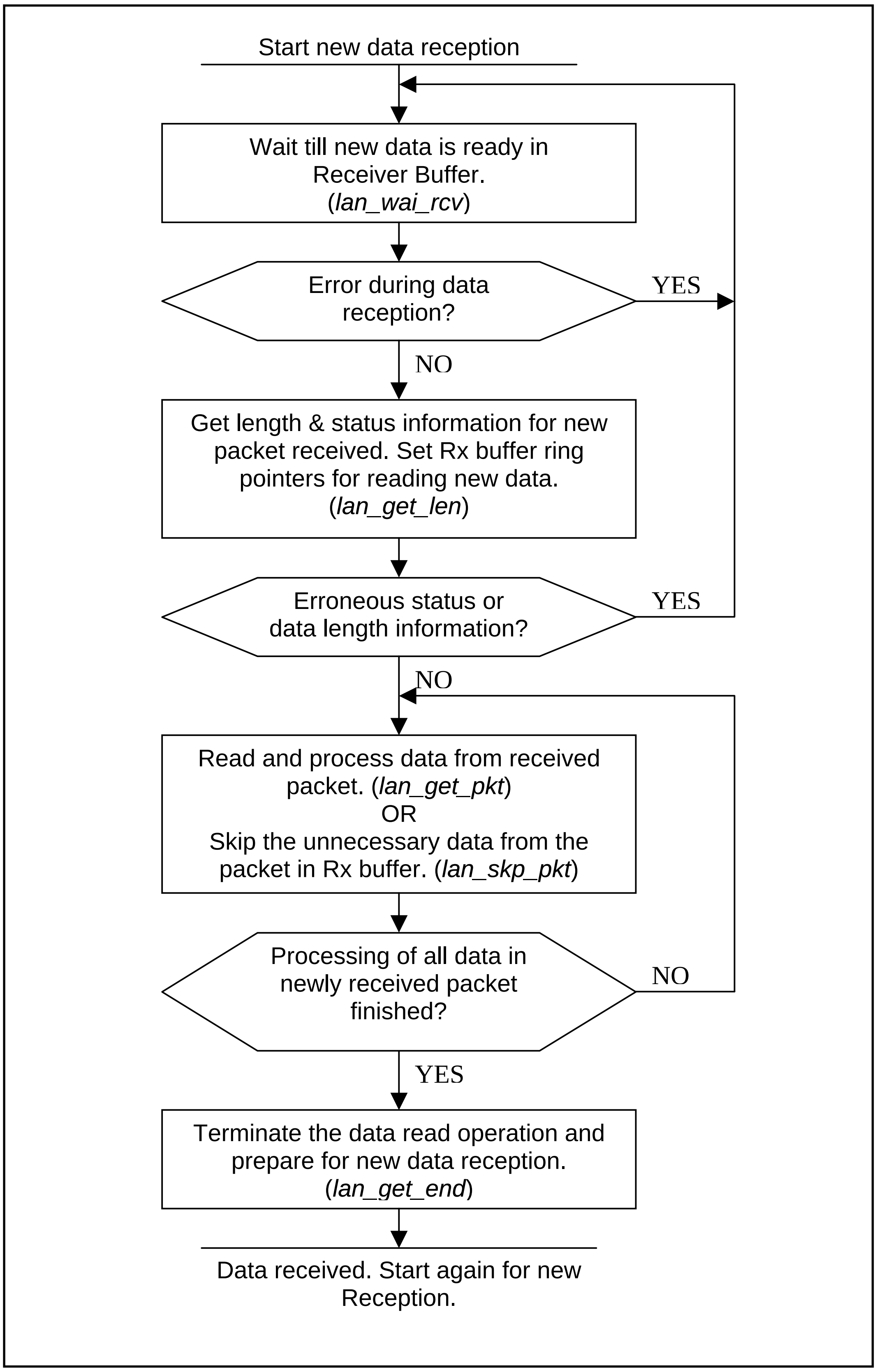
7.12 Basic Transmit operation

The functional flow for the basic transmit operation is described in the following graph.


Data is ready for Transmission
Wait till LAN Controller is ready for new data transmission. (*lan_wai_snd*)
Send transmit data length information to controller. (*lan_set_len*)
Upload transmit data to controller RAM. (*lan_put_pkt*)
Is all data moved to Controller RAM?
NO
YES
Total size of transmit data <60 Bytes?
NO
YES
Send dummy data to controller RAM to make total size >= 64 Bytes. (*lan_put_dmy*)
Actual Transmission will start after Transmit Descriptor is updated. (*lan_put_end*)
Data transmission is over. Transmission status can be checked after Transmit End interrupt signal.

7.13 Basic Receive operation

The functional flow for the basic receive operation is described in the following graph.


Start new data reception
Wait till new data is ready in Receiver Buffer. (*lan_wai_rcv*)
Error during data reception?
YES
NO
Get length & status information for new packet received. Set Rx buffer ring pointers for reading new data. (*lan_get_len*)
Erroneous status or data length information?
YES
NO
Read and process data from received packet. (*lan_get_pkt*) OR Skip the unnecessary data from the packet in Rx buffer. (*lan_skp_pkt*)
Processing of all data in newly received packet finished?
NO
YES
Terminate the data read operation and prepare for new data reception. (*lan_get_end*)
Data received. Start again for new Reception.

MiSPO
http://www.mispo.co.jp
MiSPO Co. Ltd.
KSP W300-G, 3-2-1, Sakado, Takatsu-ku, Kawasaki 213-0012, JAPAN
Tel. +81-44-829-3381
fax. +81-44-829-3382
E-mail: sales@mispo.co.jp