APPLICATION NOTE

H8S/2472, 2463 and 2462 Groups

Example of Settings for Transmission and Reception of Ethernet Frames

R01AN0645EJ0102
Rev.1.02
Jul 06, 2011

Introduction

This application note describes an example of settings for connecting the Ethernet controller of the H8S/2472, 2463 and 2462.

Target Device

H8S/2472

Contents

1. Preface

1.1 Specifications

- In this sample program, three types of processing (A, B and C below) are selected for the transmission and reception of Ethernet frames.
  - A. Two rounds of transmitting one Ethernet frame and receiving one Ethernet frame proceed.
  - B. Ten Ethernet frames are transmitted.
  - C. Ten Ethernet frames are received.
- After the transmission of each frame is completed, transmission of the next proceeds.
- The frame transmission complete interrupt is used to judge whether frame transmission has been completed or not.
- Every time the function of reception is called, the single frame of data is copied to the user buffer.
- In obtaining the result of automatic negotiation by the physical-layer LSI circuit (PHY-LSI), the connection mode (full-duplex mode or half-duplex mode) determined by the automatic negotiation function of the PHY-LSI is obtained.
- The LAN8700* manufactured by SMSC is used as the Ethernet PHY-LSI.

Note: * The LAN8700 is the Ethernet physical layer transceiver.

1.2 Modules Used

- Ethernet controller (EtherC)
- Direct memory access controller for Ethernet controller (E-DMAC)
- Interrupt controller

1.3 Applicable Conditions

• MCU:	H8S/2472, 2463, and 2462
• Operating frequency:	System clock: 32 MHz
• Integrated development environment:	High-performance Embedded Workshop Ver.4.07.00.007 from Renesas Electronics
• Toolchain:	H8S, H8/300 Standard Toolchain (V.6.2.2.0)
• Compiler options:	-cpu=2600A:24 -object="$(CONFIGDIR)\$(FILELEAF).obj" -debug -nolist -chgincpath -nologo

1.4 Example of Connecting an MCU to a Physical-Layer LSI Circuit

Figure 1 shows an example of the connections between an MCU and the LAN8700 from SMSC.

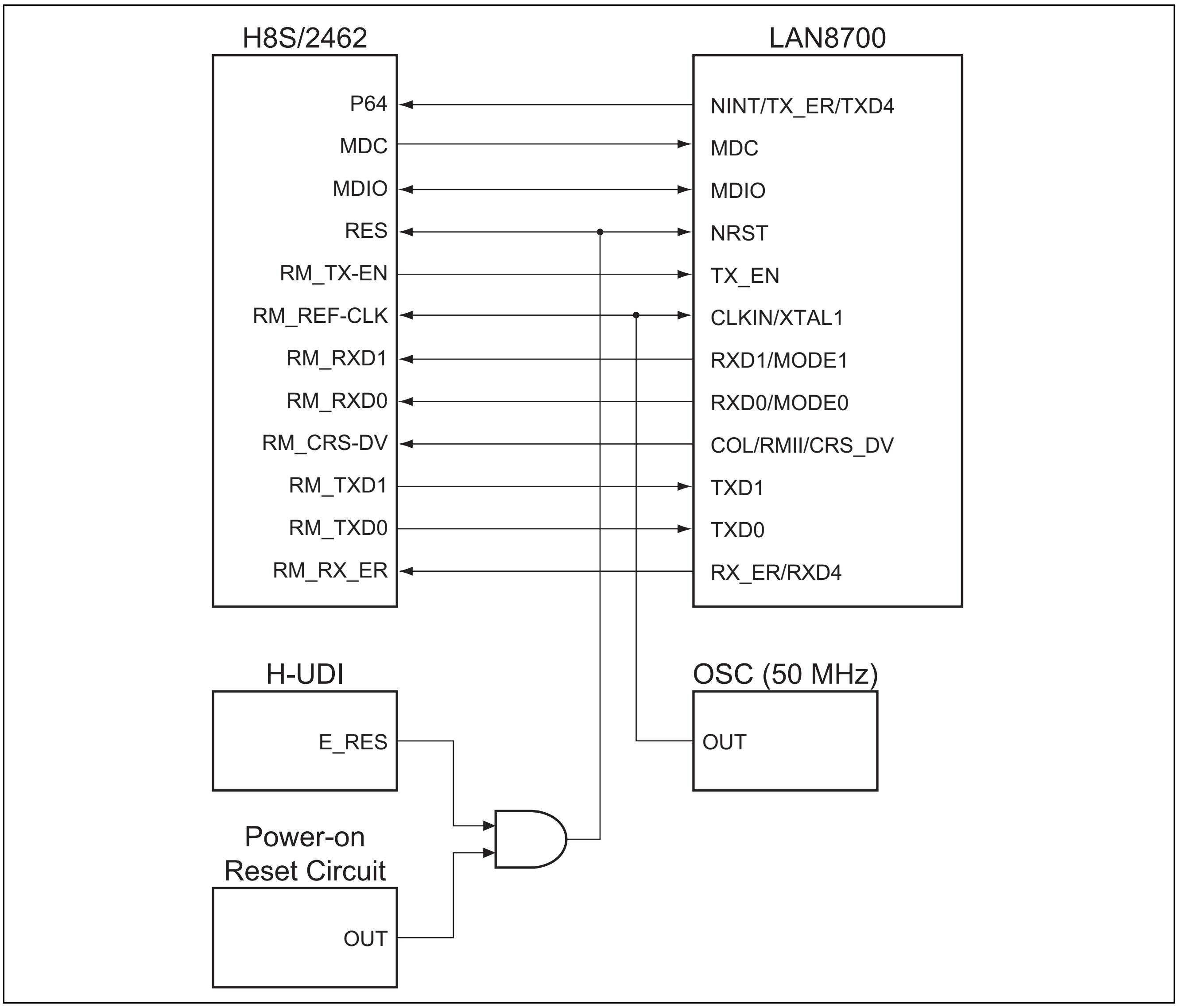


Figure 1 Example of Connecting an MCU to the LAN8700 (Reduced Media Independent Interface)

2. Description of the Sample Application

The sample program employs the Ethernet controller (EtherC) and direct memory access controller for the Ethernet controller (E-DMAC).

In this sample program, the Ethernet PHY-LSI is used for automatic negotiation. The result of the automatic negotiation is read from the PHY interface register (PIR) of the controller.

2.1 Operational Overview of Modules Used

Be sure to use the EtherC and E-DMAC modules to handle Ethernet communications for this LSI. The EtherC module controls the transmission and reception of Ethernet frames and their transfer between Media Access Control (MAC) layers. The E-DMAC specifically handles DMA transfer between its transmission/reception FIFO and data-storage areas (buffers) specified by the user.

The Media Independence Interface (MII) registers in the Ethernet PHY-LSI are accessed via the PIR of the EtherC module. Figure 2 shows the MII management frame format. Figures 3 to 5 show examples of the timing of access to MII registers. However, please note that the pulse width and duration of a clock cycle are limited.

Access Type	MII Management Frame							
Item	PRE	ST	OP	PHYAD	REGAD	TA	DATA	IDLE
Number of bits	32	2	2	5	5	2	16	—
Read	1..1	01	10	00001	RRRRR	Z0	D..D	—
Write	1..1	01	01	00001	RRRRR	10	D..D	X

[Legend]

PRE:	32 consecutive 1s
ST:	Write of B'01 indicating start of frame
OP:	Write of code indicating access type
PHYAD:	Write of B'00001 if the PHY address is 1 (sequential write starting with the MSB). This bit changes depending on the PHY address.
REGAD:	Write of B'00001 if the register address is 1 (sequential write starting with the MSB). This bit changes depending on the PHY register address.
TA:	Time for switching data transmission source on MII interface (a) Read: Bus is released (indicated as Z0). (b) Write: B'10 is written.
DATA:	16-bit data. Sequential write or read from MSB (a) Read: 16-bit data read (b) Write: 16-bit data write
IDLE:	Wait time until next MII management format input (a) Read: Since the bus has been released at TA already, control is not required. (b) Write: Independent bus release (indicated as X) is performed.

Figure 2 MII Management Frame Format

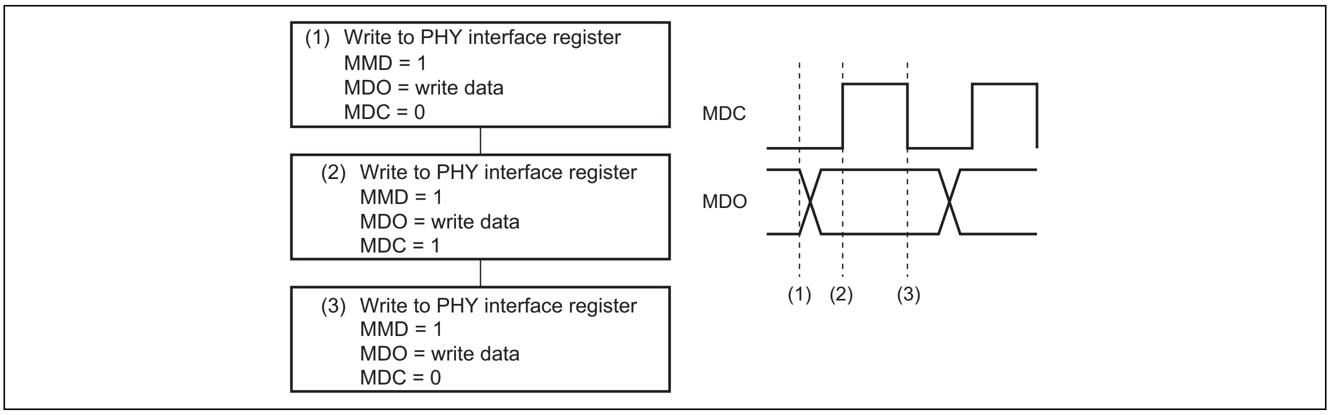


Figure 3 1-Bit Data Write Flow

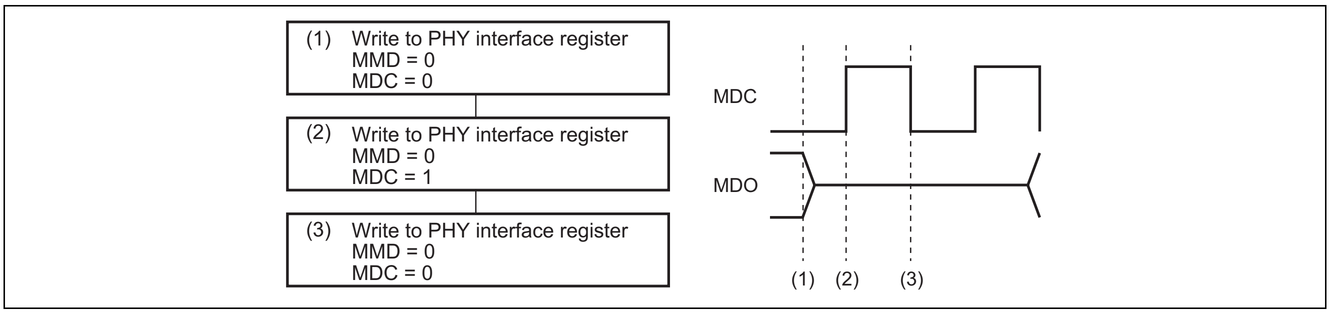

(1) Write to PHY interface register
MMD = 0
MDC = 0
(2) Write to PHY interface register
MMD = 0
MDC = 1
(3) Write to PHY interface register
MMD = 0
MDC = 0
MDC
MDO
(1) (2) (3)

Figure 4 Bus Release Flow

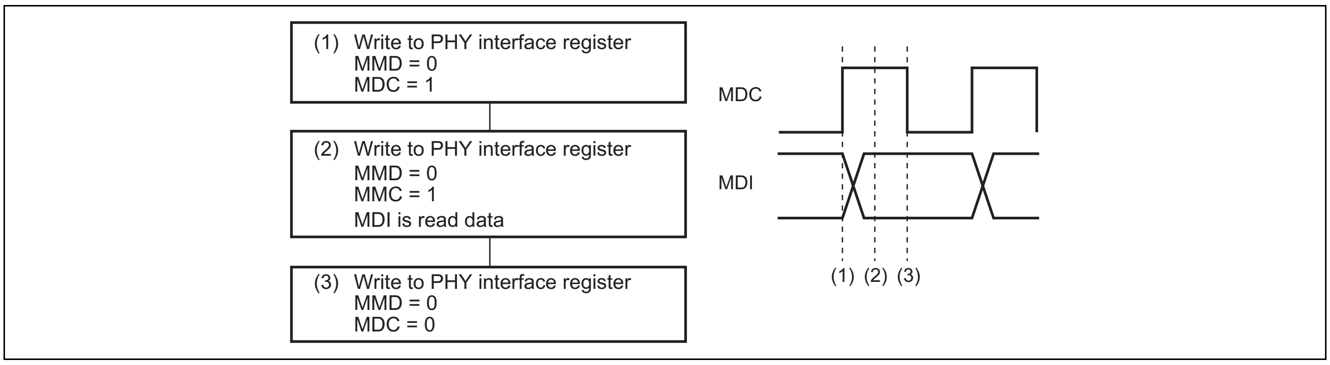

(1) Write to PHY interface register
MMD = 0
MDC = 1
(2) Write to PHY interface register
MMD = 0
MMC = 1
MDI is read data
(3) Write to PHY interface register
MMD = 0
MDC = 0
MDC
MDI
(1) (2) (3)

Figure 5 1-Bit Data Read Flow

2.1.1 Overview of the EtherC

This LSI has an on-chip Ethernet controller (EtherC) that conforms to the Ethernet or IEEE802.3 MAC layer standard. Connecting a PHY-LSI complying with this standard enables the EtherC to perform transmission and reception of Ethernet/IEEE802.3 frames. This LSI has one MAC layer interface.

The Ethernet controller is connected to the direct memory access controller for Ethernet controller (E-DMAC) inside this LSI, and carries out high-speed data transfer to and from the memory.

Figure 6 shows configuration of EtherC.

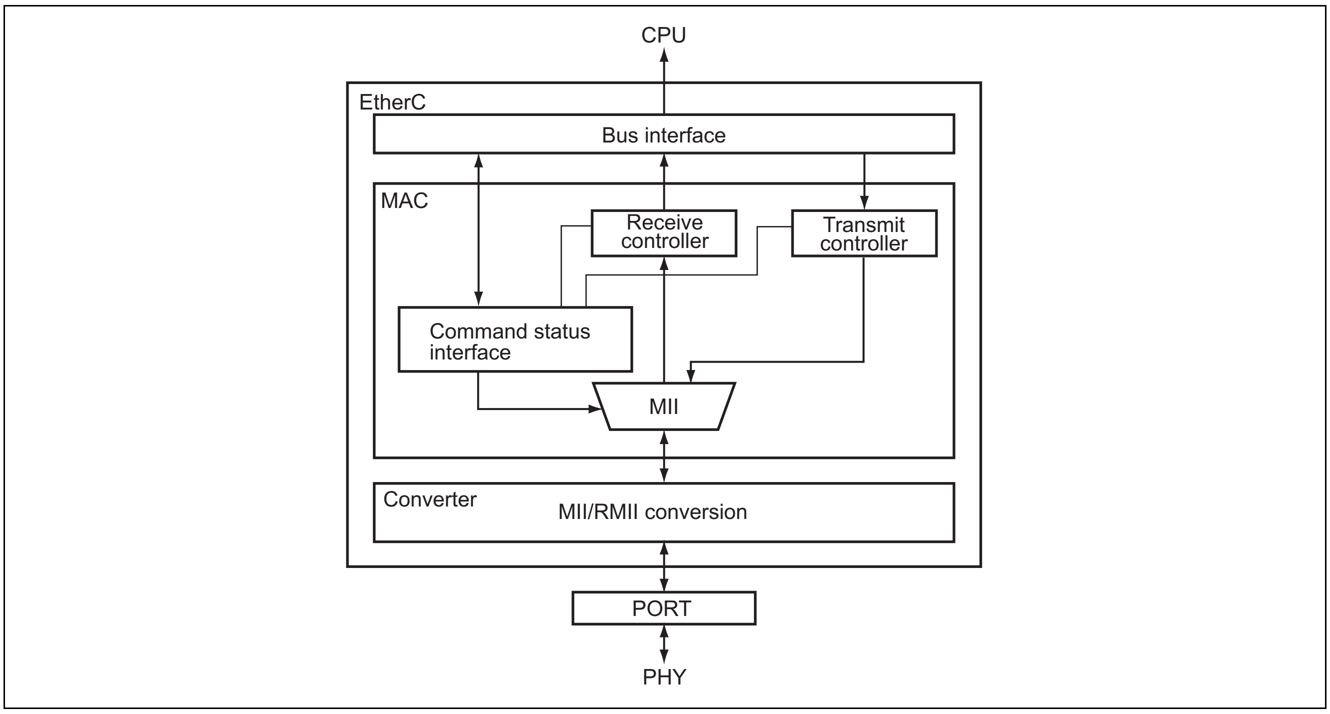
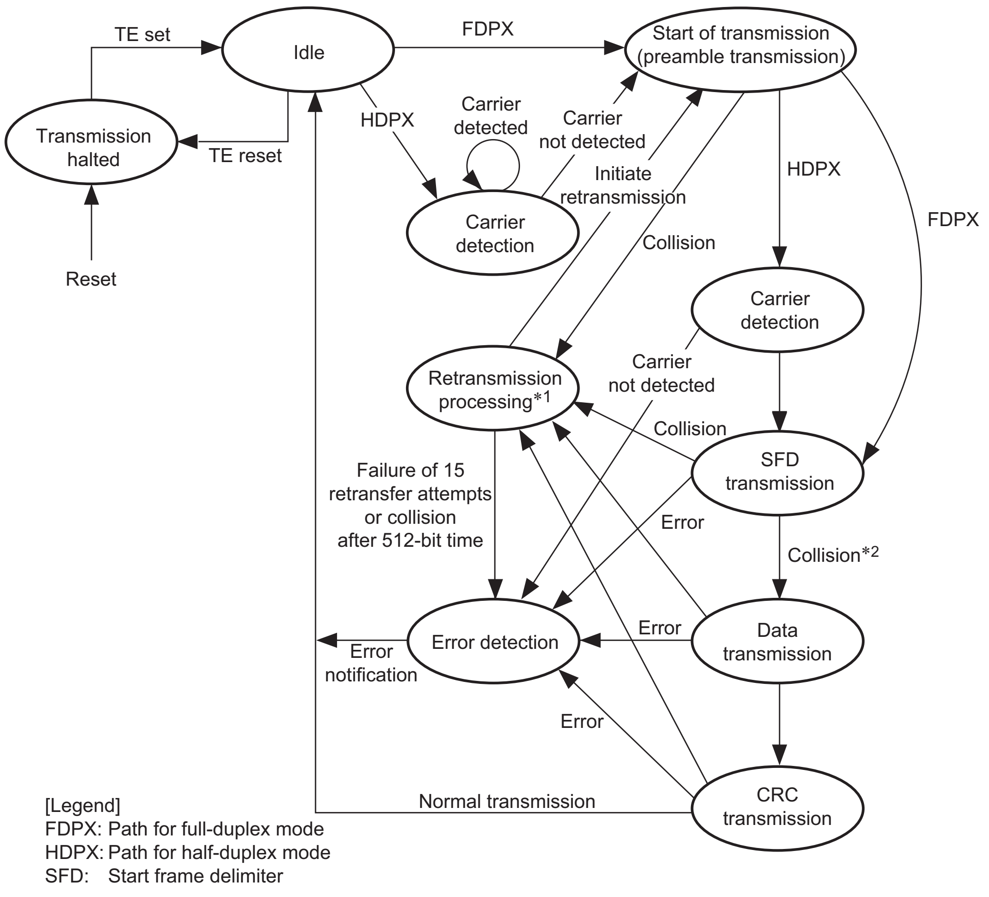


Figure 6 Configuration of the EtherC

2.1.2 Overview of the EtherC Transmitter

In response to a request for transmission from the E-DMAC, the EtherC transmitter arranges the data for transmission into a frame and sends them to the reduced media independent interface (RMII). Once the data have gone through the RMII, they are output onto the lines by the PHY-LSI. Figure 7 shows the state transitions of the EtherC transmitter. The following describes the flow of operations in transmission.

1. When the transmit enable (TE) bit of the EtherC mode register (ECMR) is set, the EtherC transmitter enters the idle state.
2. (A) When a request for transmission is issued by the transmitter E-DMAC while half-duplex transfer has been selected, the EtherC module attempts to detect a carrier. If it does not detect a carrier, the EtherC module sends the preamble to the RMII after a transmission delay equivalent to the time required by the frame interval. If a carrier is detected, the EtherC module waits until the carrier disappears and then sends the preamble to the RMII after a transmission delay equivalent to the time required by the frame interval.
   (B) Full-duplex transfer does not require carrier detection, so if this is selected, the preamble is sent as soon as the request for transmission is issued by the E-DMAC. In continuous transmission, however, the preamble is sent from the frame which has been transmitted at the last minute surely after a transmission delay equivalent to the time required by frame interval.
3. The EtherC transmitter sends the start frame delimiter (SFD), data, and cyclic redundancy check (CRC) code in sequence. At the end of transmission, the transmitter E-DMAC generates a frame transmission complete (TC) interrupt. If a collision occurs or the EtherC transmitter enters the carrier-not-detected state, an interrupt corresponding to the given state will be generated.
4. The EtherC transmitter enters the idle state and then, if there are more data for transmission, continues to transmit.



Notes: 1. Retransmission processing includes both jam transmission that depends on collision detection and the adjustment of transmission intervals based on the back-off algorithm.
2. Retransmission processing is only performed when data of 512 bits or less (including the preamble and SFD) is transmitted. When a collision is detected during transmission of data greater than 512 bits, only jam is transmitted and retransmission based on the back-off algorithm is not performed.

Figure 7 State Transitions of the EtherC Transmitter

2.1.3 Overview of the EtherC Receiver

The EtherC receiver separates the frame of data which have been input from the RMII into preamble, SFD, data, and CRC code, and outputs the portion from the destination address (DA) to the CRC data to the receiver E-DMAC. Figure 8 shows the state transitions of the EtherC receiver. The flow of operations in reception is described below.

1. When the receive enable (RE) bit of the EtherC mode register (ECMR) is set, the EtherC receiver enters the idle state.
2. When the start frame delimiter (SFD) is detected after the preamble of a frame to be received, the EtherC receiver starts processing for reception. A frame with an invalid pattern is discarded.
3. In normal mode, the EtherC receiver starts reception of data (i) if the destination MAC address matches the receiver's own address, (ii) in the case of a broadcast frame, and (iii) in the case of a multicast frame. If promiscuous mode has been specified, the EtherC receiver starts reception of data irrespective of the frame type.
4. After a frame has been received from the RMII, the EtherC receiver carries out a CRC of the frame data. The result is indicated as a status bit in the descriptor after the frame of data has been written to memory. If an error is found, the error state is reported to the EtherC/E-DMAC status register (EESR).
5. After one frame has been received, the EtherC receiver enters the idle state in readiness for receiving the next frame.

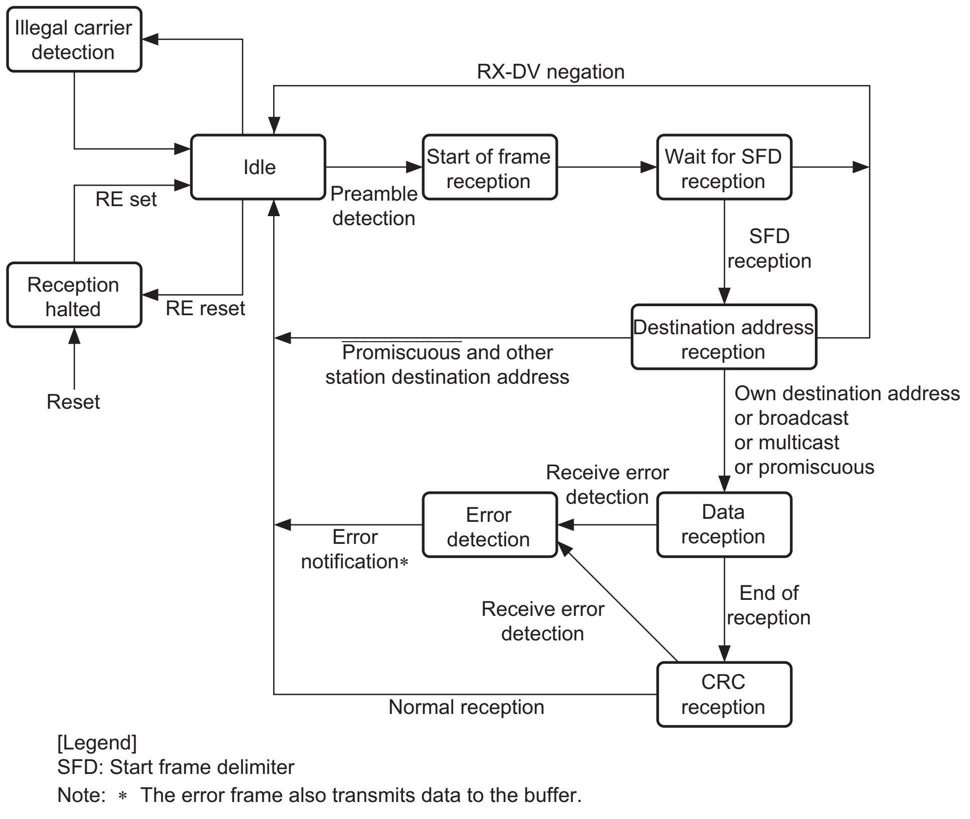


Figure 8 State Transitions of the EtherC Receiver

2.1.4 Overview of the E-DMAC

This LSI includes a direct memory access controller (E-DMAC) that is directly connected to the Ethernet controller (EtherC). The E-DMAC employs descriptors to control a large portion of buffer management. This lightens the load on the CPU and enables efficiency in data transfer control. Directly writing data to or reading data from the transmit/receive FIFO by the CPU is not possible.

During DMA transfer, the E-DMAC refers to information called transmit and receive descriptors; these are placed in memory by the user. The E-DMAC reads the descriptor information before transmitting or receiving an Ethernet frame, and follows the descriptor in reading data for transmission from the transmission buffer or writing received data to the receiving buffer. By setting up a number of consecutive descriptors (a descriptor list), it is possible to execute the consecutive transfer of multiple Ethernet frames.

Figure 9 shows the configuration of the E-DMAC, and of the related descriptors and buffers.

The E-DMAC has the following features:

— The descriptor management system reduced the load on the CPU.
— The descriptions indicate information on the states of frames to be transmitted and received frames.
— Block transfer (16-byte units) achieves efficient utilization of the system bus.
— Supports single-frame/multi-buffer operation

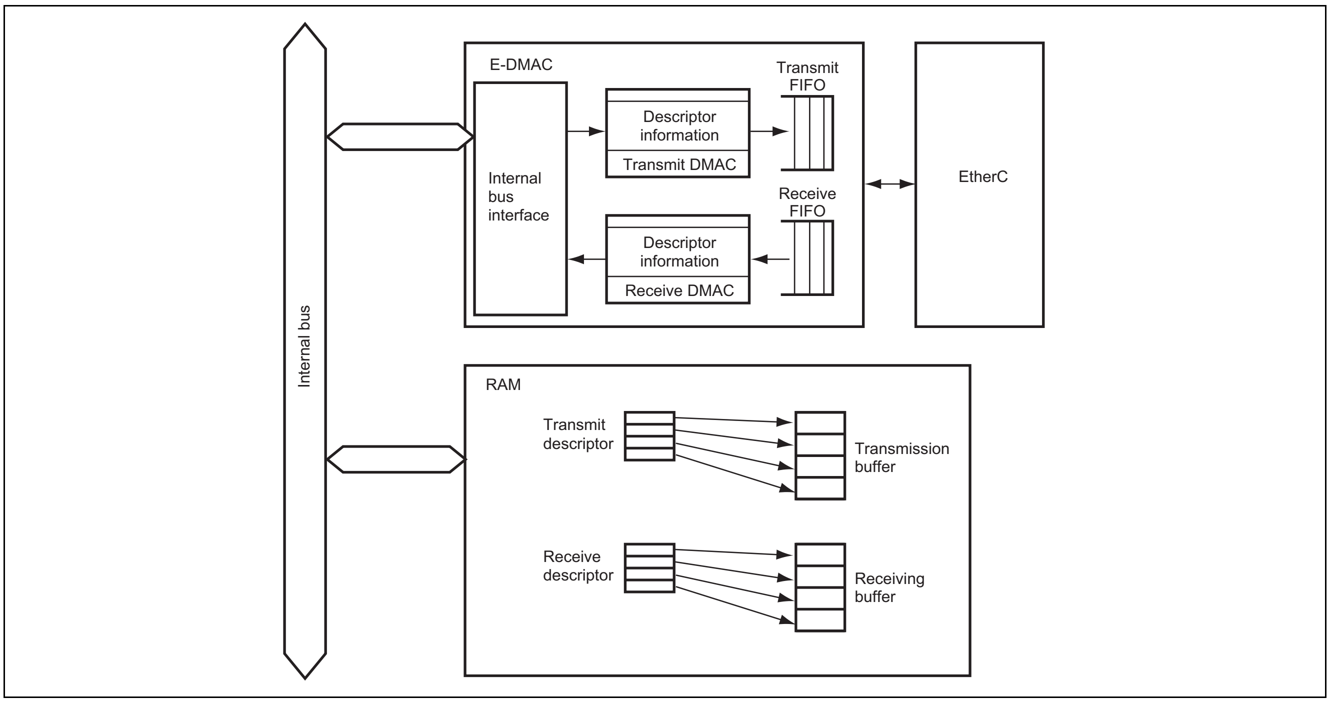


Figure 9 Configuration of the E-DAMC, and of Related Descriptors and Buffers

2.1.5 Overview of E-DMAC Descriptors

When the E-DMAC performs DMA transfer, it employs descriptor information that includes the storage address for the data for transfer, etc. There are two types of descriptors: transmit descriptors and receive descriptors. When the TR bit in the E-DMAC transmit request register (EDTRR) is set to 1, the E-DMAC automatically starts reading a transmit descriptor. When the RR bit in the E-DMAC receive request register (EDRRR) is set to 1, the E-DMAC automatically starts reading a receive descriptor. The user must enter information related to the DMA transfer of Ethernet data in the transmit/receive descriptors before the transfer can proceed. After transmission or reception of an Ethernet frame has been completed, the E-DMAC switches the descriptor active/inactive bit (TACT bit for transmission, RACT bit for reception) to the inactive setting and indicates the result of transmission or reception in the status bits (TFS26 to TFS0 for transmission, RFS26 to RFS0 for reception).

Descriptors are placed in readable and writable memory, and the address where the first descriptors start (the addresses of the first descriptors of each type to be read by the E-DMAC) are set in the transmit descriptor list address register (TDLAR) and receive descriptor list address register (RDLAR). When multiple descriptors are set up in a descriptor list, the descriptors are placed in contiguous address ranges in accord with the descriptor length as indicated by bits DL1 and DL0 in the E-DMAC mode register (EDMR).

2.1.6 Overview of Transmit Descriptors

Figure 10 shows the relationship between a transmit descriptor and a transmission buffer.

In order from its first address, a transmit descriptor consists of TD0, TD1, TD2 (each is a 32-bit unit), and padding. TD0 indicates whether the descriptor is active or inactive, describes the configuration of the descriptor, and contains state information. TD1 indicates the size of the transmission buffer indicated by the descriptor. The TD2 indicates the address where the transmission buffer starts. The length of padding is determined by the descriptor length as specified by bits DL0 and DL1 in the EDMR register.

According to the settings of transmit descriptors, either a single descriptor or multiple descriptors can specify a single frame of data for transmission (one frame/one descriptor and one frame/multi-descriptor, respectively). As an example where the one frame/multi-descriptor type of setting may be useful, multiple descriptors might be set up for data in Ethernet frames which are used in transmission every time. Specifically, data for the destination and source addresses within the Ethernet frame may be shared among multiple descriptors, with the remaining data stored in individual buffers.

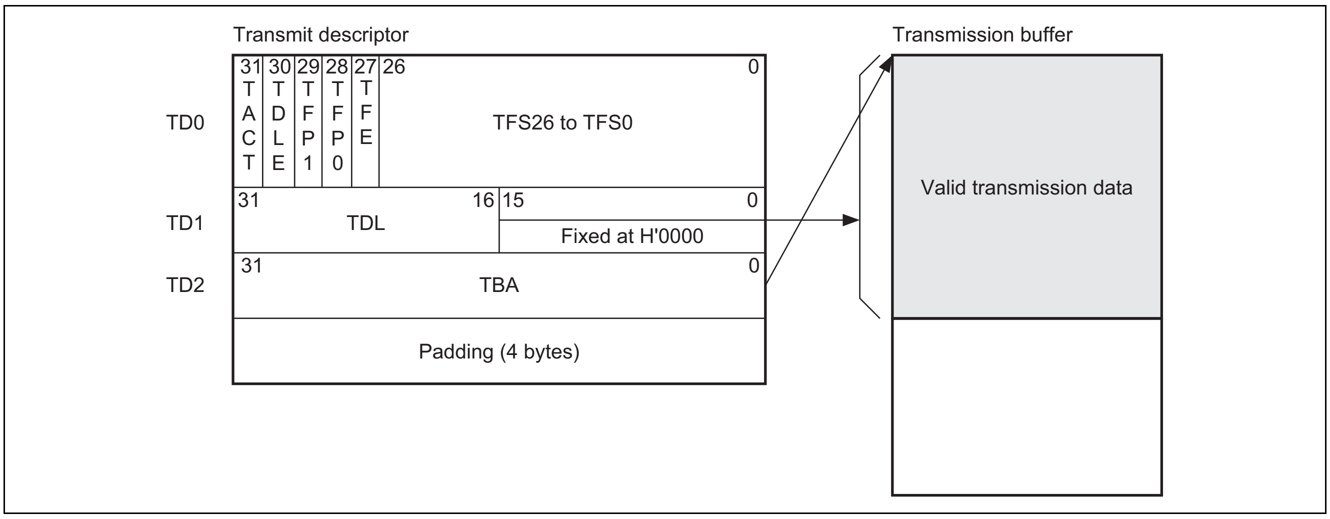


Figure 10 Relationship between a Transmit Descriptor and Transmission Buffer

2.1.7 Overview of Receive Descriptors

Figure 11 shows the relationship between a receive descriptor and a receiving buffer.

In order from its first address, a receive descriptor consists of RD0, RD1, RD2 (each is a 32-bit unit), and padding. RD0 indicates whether the descriptor is active or inactive, describes the configuration of the descriptor, and contains state information. RD1 indicates the size of the receiving buffer (RBL) to which the descriptor refers, and the length of the received frame (RDL). RD2 indicates the address where the receiving buffer starts. The length of padding is determined by the descriptor length as specified by bits DL0 and DL1 in the EDMR register.

According to the settings of receive descriptors, either a single descriptor or multiple descriptors can specify a single frame of received data (one frame/one descriptor and one frame/multi-descriptor, respectively). In one frame/multi-descriptor cases, multiple descriptors are prepared in advance to form a descriptor list. If a frame is longer than the setting of the descriptor's RBL field, the E-DMAC uses the next descriptor in the sequence to continue transferring the frame to the receiving buffer. For example, if the E-DMAC receives an Ethernet frame with 1,514 bytes while the RBL of each descriptor is 500 bytes, the received Ethernet frame is transferred to the receiving buffer in 500-byte portions until the final 14 bytes that remain are transferred to the fourth buffer.

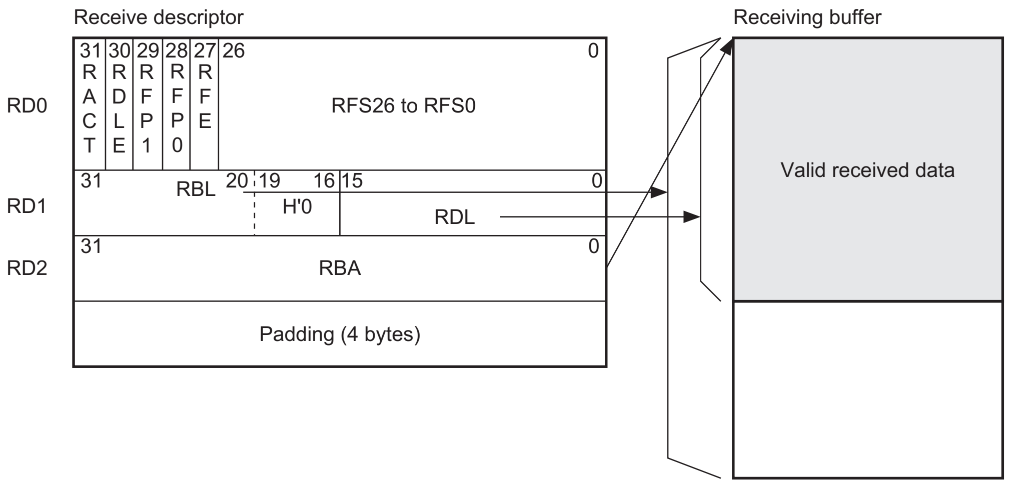


Figure 11 Relationship between a Receive Descriptor and Receiving Buffer

2.1.8 Example of Setting Transmit Descriptors

Figure 12 shows an example (one frame/one descriptor) where three transmit descriptors and three areas of the transmission buffer are in use. In this case, a single frame is transmitted in response to a single request for transmission. The transmit descriptors are simplified in the figure, with only TD0 being shown. Numbers (1), (2), etc. in the figure indicate the sequence of execution.

The Settings are as follows.

1. Due to one-frame/one-descriptor operation, the TFP1 and TFP0 bits of all descriptors are set to B'11.
2. Bits TACT, TFE, and TFS26 to TFS0 of individual descriptors are all set to 0 as the initial value.
3. In the first and second descriptors, the TDLE bit is set to 0. The TDLE bit of the third descriptor is set to 1, so the E-DMAC reads the first descriptor on completion of processing of the third descriptor. Settings like this can be used to arrange descriptors in a ring structure.
4. Although the following settings have been left out of figure 12, the data length of the transmission buffer referred to by the respective descriptors is set in TDL, and the addresses where individual areas of the transmission buffer start are set in TBA.
5. Since only one frame is transmitted in response to each request in this example, only the TACT bit of the first descriptor is set to 1 for the first transmission. For the next transmission, only the TACT bit of the second descriptor is set to 1.

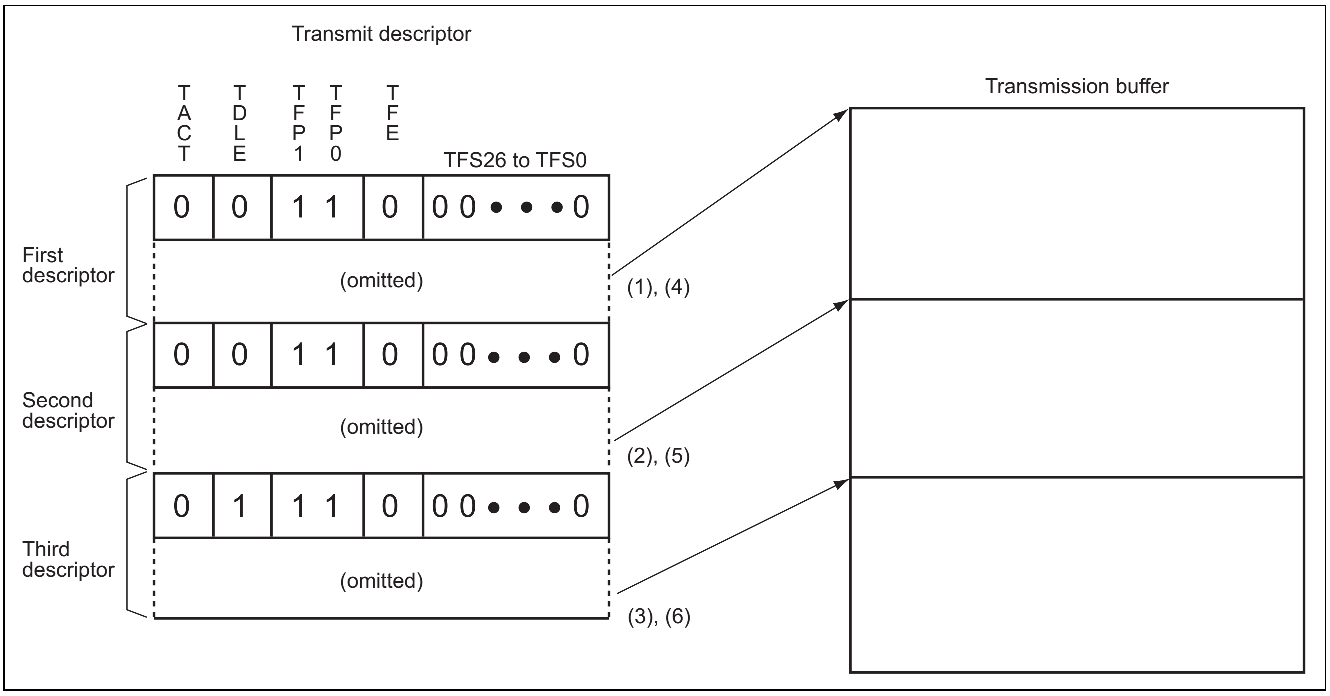


Figure 12 Relationship between Transmit Descriptors and Areas of Transmission Buffer

2.1.9 Example of Setting Receive Descriptors

Figure 13 shows an example where three receive descriptors and three areas of the receiving buffer are in use. Each area of the receiving buffer has a size of 1,520 bytes, and operation is of the one-frame/one-descriptor type. The receive descriptors are simplified in the figure, with only RD0 being shown. Numbers (1), (2), etc. in the figure indicate the sequence of execution.

The settings are as follows.

1. Bits RFP1, RFP0, RFE, and RFS26 to RFS0 of all descriptors are set to 0.
2. In the first and second descriptors, the RDLE bit is set to 0. The RDLE bit of the third descriptor is set to 1, so the E-DMAC reads the first descriptor on completion of processing of the third descriptor. Settings like this can be used to arrange descriptors in a ring structure.
3. Although the following settings for each of the descriptors have been left out of figure 13, prior to the start of reception, the RBL of RD1 is set for a size of each area of the receiving buffer, 1,520 bytes, and the RBA of RD2 is set to the address where the corresponding area of the receiving buffer starts.
4. To enable continuous reception, the RACT bit of each descriptor is set to 1.

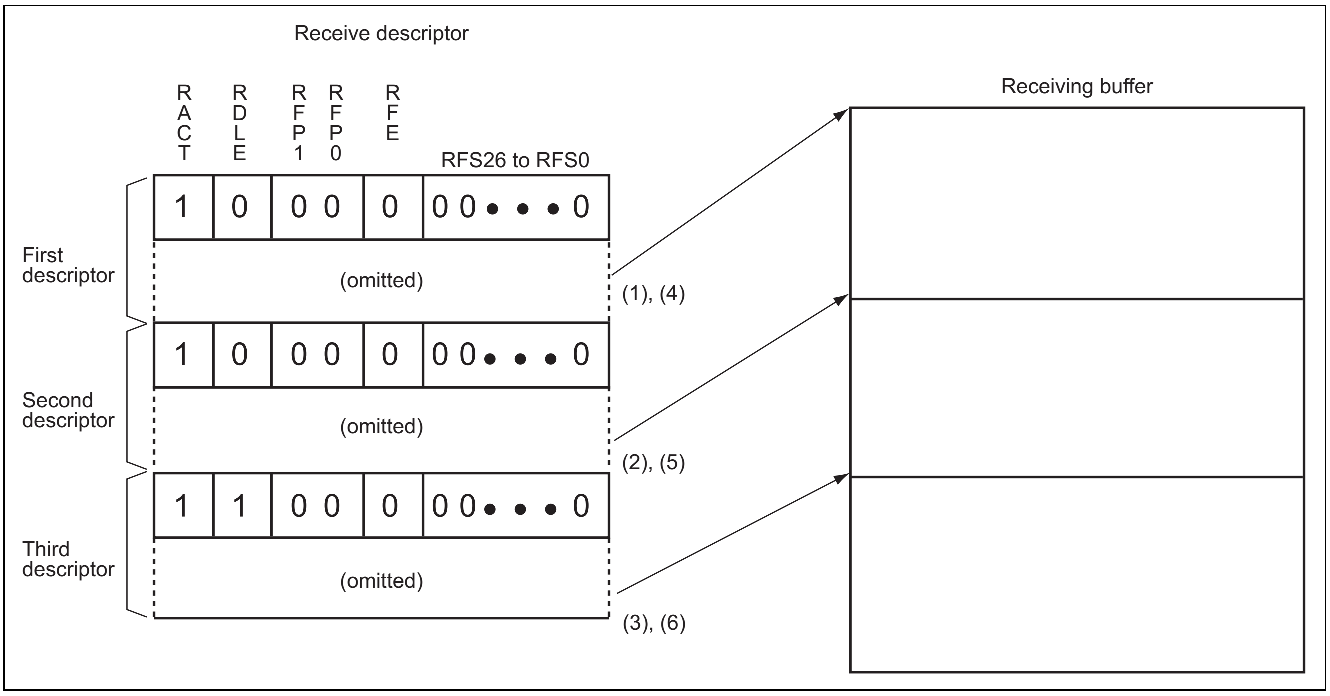


Figure 13 Relationship between Receive Descriptors and Areas of Receiving Buffers

2.1.10 Procedure for Setting Modules Used (Transmission)

When the setting of the TE bit of the EtherC mode register (ECMR) is 1 and 1 is written to the transmit request (TR) bit in the E-DMAC transmit request register (EDTRR), the transmission section of the E-DMAC is activated. After a software reset of the EtherC and E-DMAC modules, the E-DMAC reads the descriptor indicated by the transmit descriptor list address register (TDLAR). If the setting of the TACT bit of that descriptor is 1 (active), the E-DMAC reads the frame of data for transmission in sequence from the first address for the transmission buffer as specified by TD2 of the transmit descriptor, and transfers it to the EtherC module.

The EtherC module creates a frame for transmission and starts transmitting it to the RMII. After DMA transfer equivalent to the buffer length specified in the descriptor, the value of the TFP bits determines further processing in the way described below.

- TFP = B'00 or B'10 (frame continuation):
  Writing back to the descriptor (to write 0 to the TACT bit) proceeds after the DMA transfer. The TACT bit of the next descriptor is then read.
- TFP = B'01 or B'11 (frame end):
  Writing back to the descriptor (to write 0 to the TACT bit or to write state information) proceeds after transmission of the frame is complete (writing of 0 or status to the TACT bit). The TACT bit of the next descriptor is then read.

If the TACT bit read from the next descriptor is 1, transmission of frames continues and the descriptor itself is read. If the TACT bit read from the next descriptor is 0 (inactive), the E-DMAC sets the TR bit in EDTRR to 0, and transmission ends. When 1 is written to the TR bit after its setting was 0, the transmission section of the E-DMAC is reactivated. In this case, however, the descriptor that is read will be that which follows the last descriptor to have been used in transmission.

Figure 14 shows an example of the flow of transmission (in the one-frame/one-descriptor and multiple-descriptor cases).

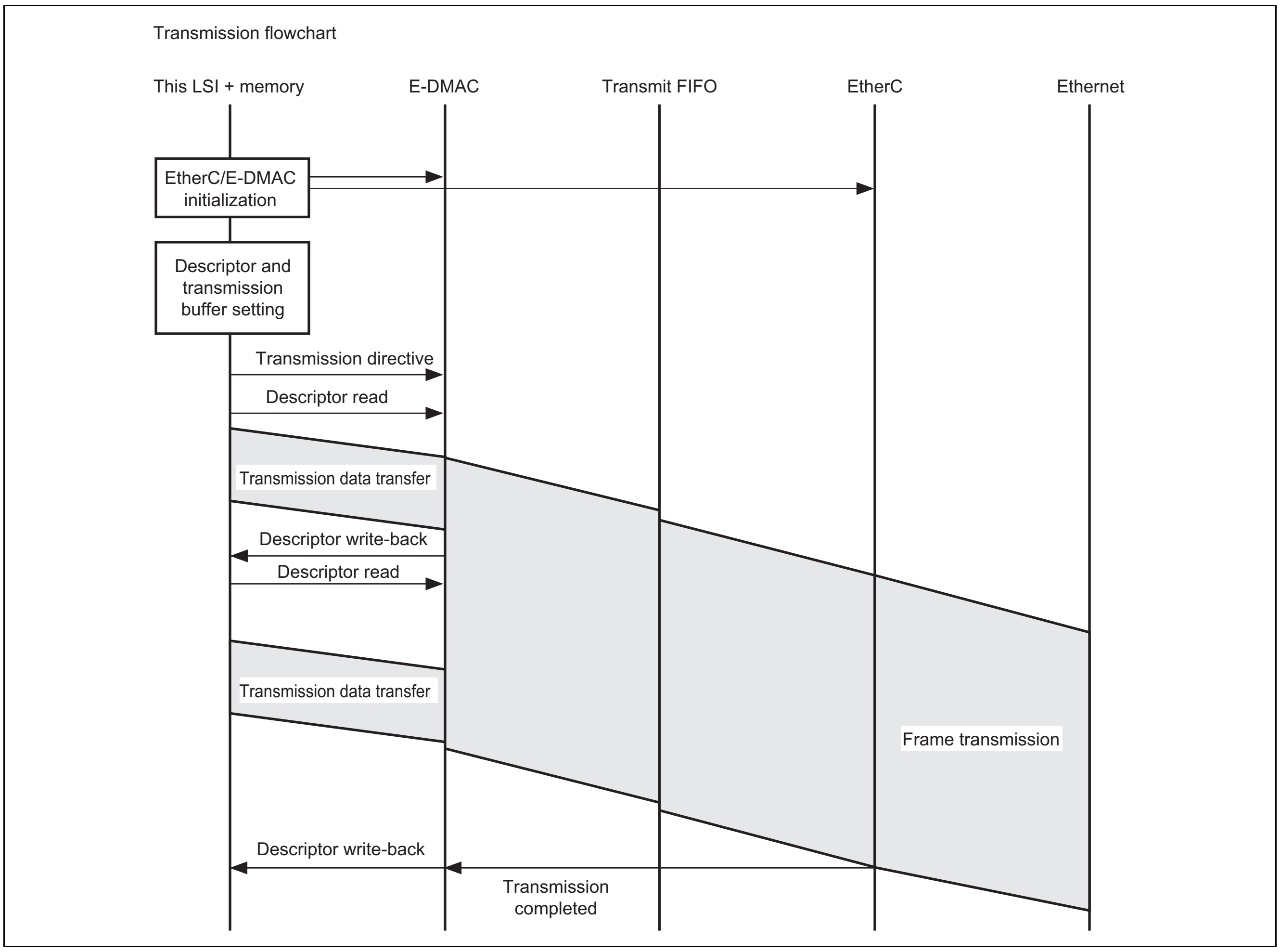


Figure 14 Example of the Flow of Transmission

2.1.11 Procedure for Setting Modules Used (Reception)

When the setting of the reception enable (RE) bit of the EtherC mode register (ECMR) is 1, and 1 is written to the receive request (RR) bit in the E-DMAC receive request register (EDRRR), the reception section of the E-DMAC is activated. After a software reset of the EtherC and E-DMAC modules, the E-DMAC reads the descriptor indicated by the receive descriptor list address register (RDLAR), and enters the reception-standby state if the setting of the RACT bit is 1 (active). If the EtherC module then receives a frame addressed to itself (the address of the frame allows for reception by the EtherC module), it stores the received data in the receive FIFO. If the setting of the RACT bit of the receive descriptor is 1, the received data are transferred to the receiving buffer specified by RD2 (if the setting of the RACT bit is 0 (inactive), the RR bit is cleared to 0 and E-DMAC operation for reception is halted). If the received frame contains more data than the buffer length given by RD1, the E-DMAC writes back to the descriptor when the buffer is full (to set RFP = B'10 or B'00), and then reads the next descriptor.

When reception of the frame is completed or is suspended because of any kind of error, the E-DMAC writes back to the current descriptor (to set RFP = B'11 or B'01). If continuous reception has been selected (i.e. cases where the setting of the receive enable control (RNC) bit in the receiving method control register (RMCR) is 1), the E-DMAC then reads the next descriptor and enters the reception-standby state if the setting of the RACT bit is 1. If continuous reception has not been selected (i.e. cases where the setting of the RNC bit in the RMCR is 0), the RR bit in EDRRR is cleared to 0 and E-DMAC operation for reception is halted. If the RR bit is again set to 1, the E-DMAC reads the descriptor which follows the last descriptor to have been used in reception, and then enters the reception-standby state.

Figure 15 shows an example of the flow of reception (in the one-frame/one-descriptor and continuous-reception cases).

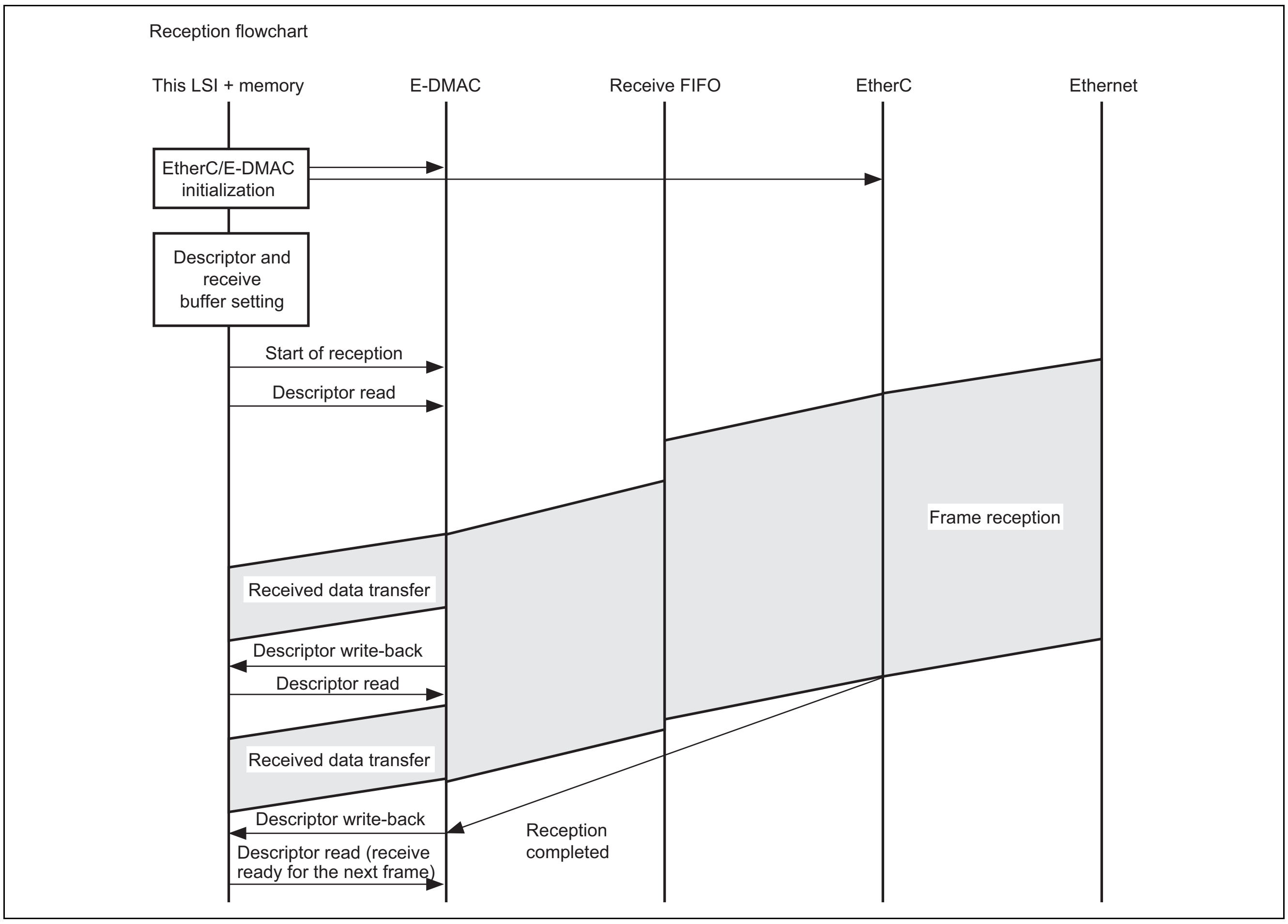


Figure 15 Example of the Flow of Reception

2.1.12 Procedure for Setting Modules Used (In Case of Transmission and Reception)

This section describes an example of fundamental settings for transmission and reception of the Ethernet frames. Figures 16 and 17 show an example of flowchart for setting the Ethernet.

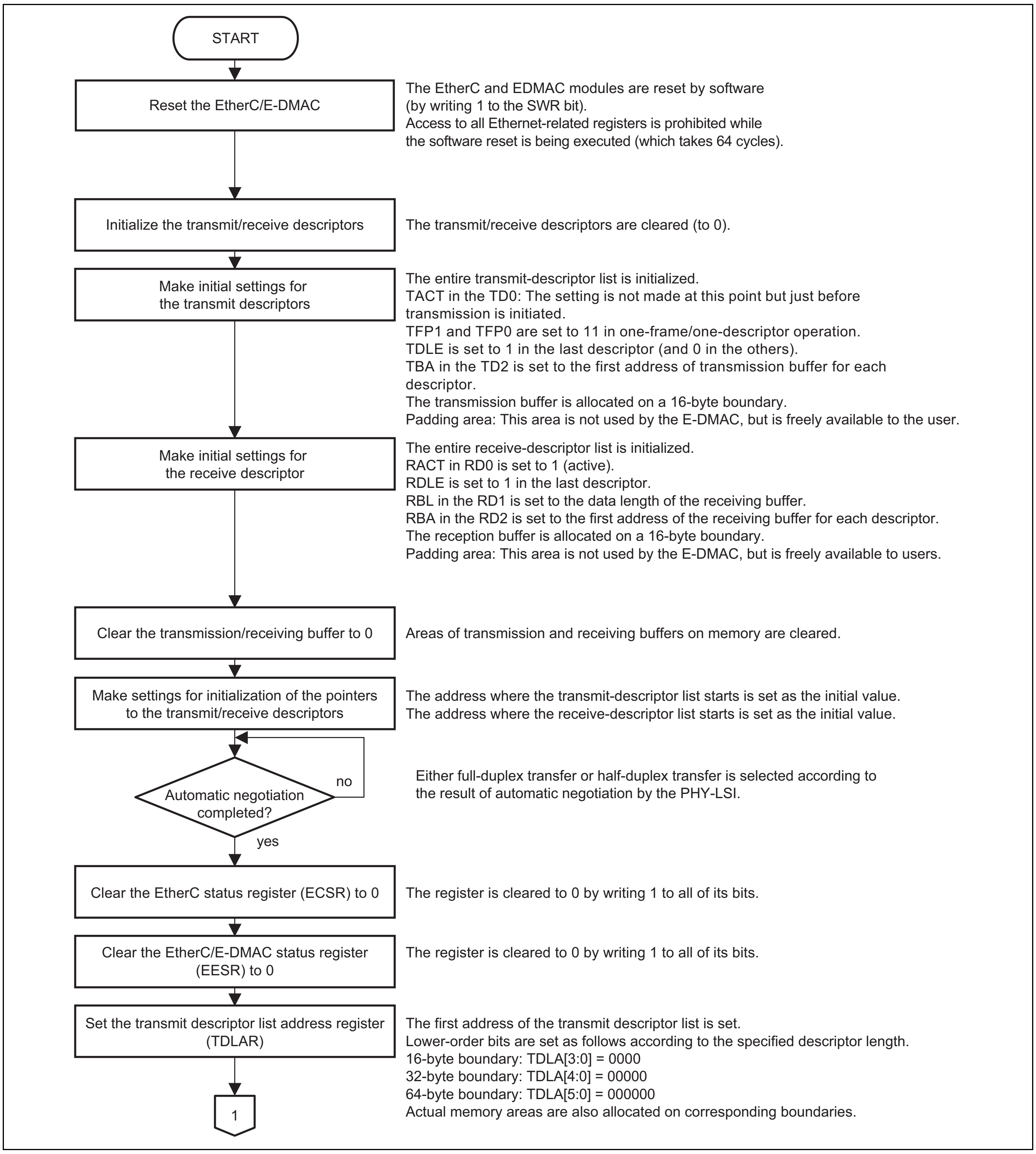
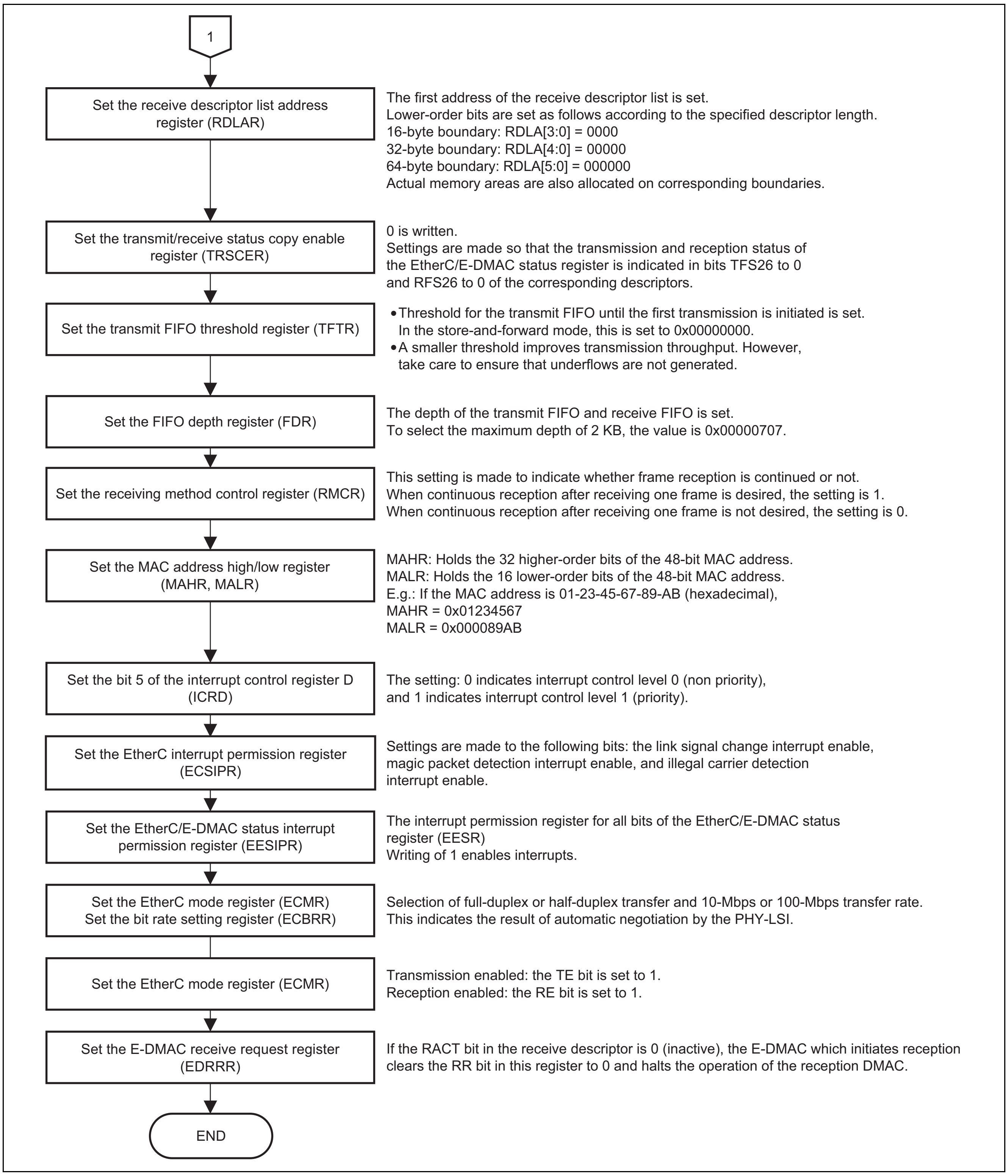

START
Reset the EtherC/E-DMAC
The EtherC and EDMAC modules are reset by software (by writing 1 to the SWR bit).
Access to all Ethernet-related registers is prohibited while the software reset is being executed (which takes 64 cycles).
Initialize the transmit/receive descriptors
The transmit/receive descriptors are cleared (to 0).
Make initial settings for the transmit descriptors
The entire transmit-descriptor list is initialized.
TACT in the TD0: The setting is not made at this point but just before transmission is initiated.
TFP1 and TFP0 are set to 11 in one-frame/one-descriptor operation.
TDLE is set to 1 in the last descriptor (and 0 in the others).
TBA in the TD2 is set to the first address of transmission buffer for each descriptor.
The transmission buffer is allocated on a 16-byte boundary.
Padding area: This area is not used by the E-DMAC, but is freely available to the user.
Make initial settings for the receive descriptor
The entire receive-descriptor list is initialized.
RACT in RD0 is set to 1 (active).
RDLE is set to 1 in the last descriptor.
RBL in the RD1 is set to the data length of the receiving buffer.
RBA in the RD2 is set to the first address of the receiving buffer for each descriptor.
The reception buffer is allocated on a 16-byte boundary.
Padding area: This area is not used by the E-DMAC, but is freely available to users.
Clear the transmission/receiving buffer to 0
Areas of transmission and receiving buffers on memory are cleared.
Make settings for initialization of the pointers to the transmit/receive descriptors
The address where the transmit-descriptor list starts is set as the initial value.
The address where the receive-descriptor list starts is set as the initial value.
Automatic negotiation completed?
no
yes
Either full-duplex transfer or half-duplex transfer is selected according to the result of automatic negotiation by the PHY-LSI.
Clear the EtherC status register (ECSR) to 0
The register is cleared to 0 by writing 1 to all of its bits.
Clear the EtherC/E-DMAC status register (EESR) to 0
The register is cleared to 0 by writing 1 to all of its bits.
Set the transmit descriptor list address register (TDLAR)
The first address of the transmit descriptor list is set.
Lower-order bits are set as follows according to the specified descriptor length.
16-byte boundary: TDLA[3:0] = 0000
32-byte boundary: TDLA[4:0] = 00000
64-byte boundary: TDLA[5:0] = 000000
Actual memory areas are also allocated on corresponding boundaries.
1

Figure 16 Example of Flowchart for Ethernet Setting (1)


1
Set the receive descriptor list address register (RDLAR)
Set the transmit/receive status copy enable register (TRSCER)
Set the transmit FIFO threshold register (TFTR)
Set the FIFO depth register (FDR)
Set the receiving method control register (RMCR)
Set the MAC address high/low register (MAHR, MALR)
Set the bit 5 of the interrupt control register D (ICRD)
Set the EtherC interrupt permission register (ECSIPR)
Set the EtherC/E-DMAC status interrupt permission register (EESIPR)
Set the EtherC mode register (ECMR)
Set the bit rate setting register (ECBRR)
Set the EtherC mode register (ECMR)
Set the E-DMAC receive request register (EDRRR)
END

The first address of the receive descriptor list is set.
Lower-order bits are set as follows according to the specified descriptor length.
16-byte boundary: RDLA[3:0] = 0000
32-byte boundary: RDLA[4:0] = 00000
64-byte boundary: RDLA[5:0] = 000000
Actual memory areas are also allocated on corresponding boundaries.

0 is written.
Settings are made so that the transmission and reception status of the EtherC/E-DMAC status register is indicated in bits TFS26 to 0 and RFS26 to 0 of the corresponding descriptors.

- Threshold for the transmit FIFO until the first transmission is initiated is set. In the store-and-forward mode, this is set to 0x00000000.
- A smaller threshold improves transmission throughput. However, take care to ensure that underflows are not generated.

The depth of the transmit FIFO and receive FIFO is set.
To select the maximum depth of 2 KB, the value is 0x00000707.

This setting is made to indicate whether frame reception is continued or not.
When continuous reception after receiving one frame is desired, the setting is 1.
When continuous reception after receiving one frame is not desired, the setting is 0.

MAHR: Holds the 32 higher-order bits of the 48-bit MAC address.
MALR: Holds the 16 lower-order bits of the 48-bit MAC address.
E.g.: If the MAC address is 01-23-45-67-89-AB (hexadecimal),
MAHR = 0x01234567
MALR = 0x000089AB

The setting: 0 indicates interrupt control level 0 (non priority), and 1 indicates interrupt control level 1 (priority).

Settings are made to the following bits: the link signal change interrupt enable, magic packet detection interrupt enable, and illegal carrier detection interrupt enable.

The interrupt permission register for all bits of the EtherC/E-DMAC status register (EESR)
Writing of 1 enables interrupts.

Selection of full-duplex or half-duplex transfer and 10-Mbps or 100-Mbps transfer rate.
This indicates the result of automatic negotiation by the PHY-LSI.

Transmission enabled: the TE bit is set to 1.
Reception enabled: the RE bit is set to 1.

If the RACT bit in the receive descriptor is 0 (inactive), the E-DMAC which initiates reception clears the RR bit in this register to 0 and halts the operation of the reception DMAC.

Figure 17 Example of Flowchart for Ethernet Setting (2)

2.2 Operation of the Sample Program (in Transmission)

This sample program employs the EtherC and E-DMAC modules to transmit 10 frames to the host personal computer at the other end. In this sample program, there are four transmit descriptors, and four areas of the transmission buffer each with 1,520 bytes (one-frame/one-descriptor operation). The transmit descriptors are used in a ring structure. Completion of the transmission of a frame is indicated by the transmission complete interrupt (TCIP), and transmission of the next frame then proceeds.

Data of the Ethernet frame other than the preamble, start frame delimiter (SFD) and CRC must be provided as data for transmission. The MAC addresses of the source and destination for the transmission in the headers must be changed to the MAC addresses of the products in use. The EtherC module does not check the MAC address of the source.

2.3 Operation of the Sample Program (in Reception)

This sample program employs the EtherC and the E-DMAC modules to receive 10 Ethernet frames from the host personal computer at the other end. In this sample program, there are four receive descriptors, and four areas of the receiving buffer each with 1,520 bytes. The receive enable control (RNC) bit in the receiving method control register (RMCR) is set to 1 to enable continuous reception operations. Every time the function of reception is called, the RFE bit (bit 27 in the RD0) of the receive descriptor is checked, and if no errors are found (i.e. RFE = 0) the single frame of data in the receive buffer is copied to the user buffer. The corresponding descriptor is then initialized in readiness for its next round of reception. If an error is found (i.e. RFE = 1), data in the receiving buffer are not copied to the user buffer but the corresponding descriptor is initialized.

Additionally, data other than the preamble, SFD, and CRC in the Ethernet frame are transferred to the receiving buffer.

2.4 Operation of the Sample Program (in Transmission and Reception)

This sample program employs the EtherC and E-DMAC modules to perform two rounds of single-Ethernet-frame transmission to the host personal computer at the other end and single-Ethernet-frame reception from the host. Four transmit descriptors and four areas of the transmission buffer each with 1,520 bytes, and four receive descriptors and four areas of the receiving buffer each with 1,520 bytes, are prepared. Transmission operations are the same as were described under 2.2, Operation of the Sample Program (for Transmission), and receiving operations are the same as were described under 2.3, Operation of the Sample Program (for Reception).

2.5 Operating Environment of the Sample Program

Figure 18 shows operating environment of the sample program.

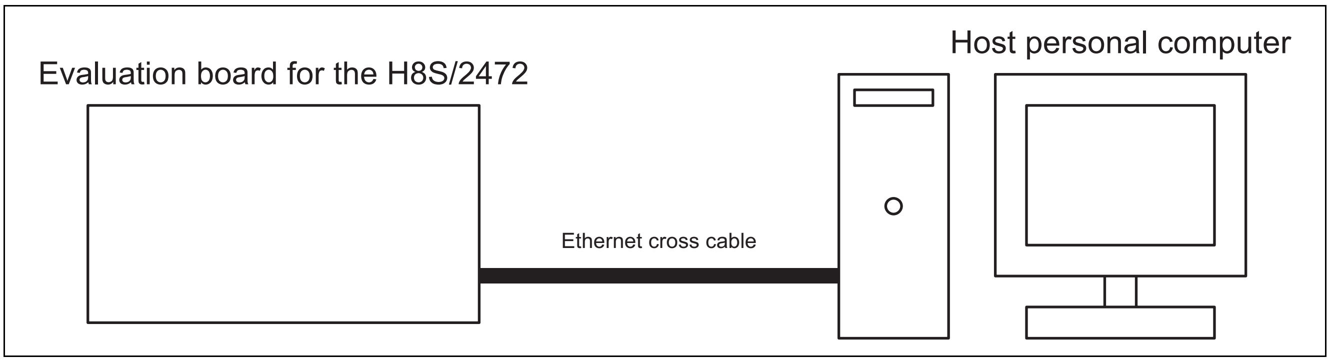


Figure 18 Operating Environment of the Sample Program

2.6 Ethernet Frame Format

Figure 19 shows a format of the Ethernet frame.

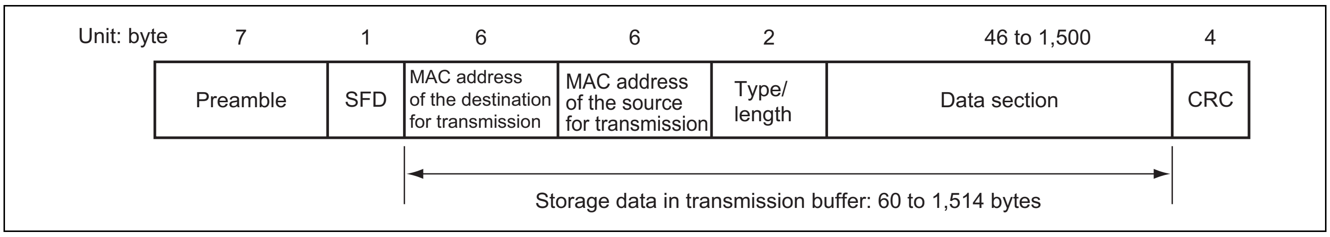
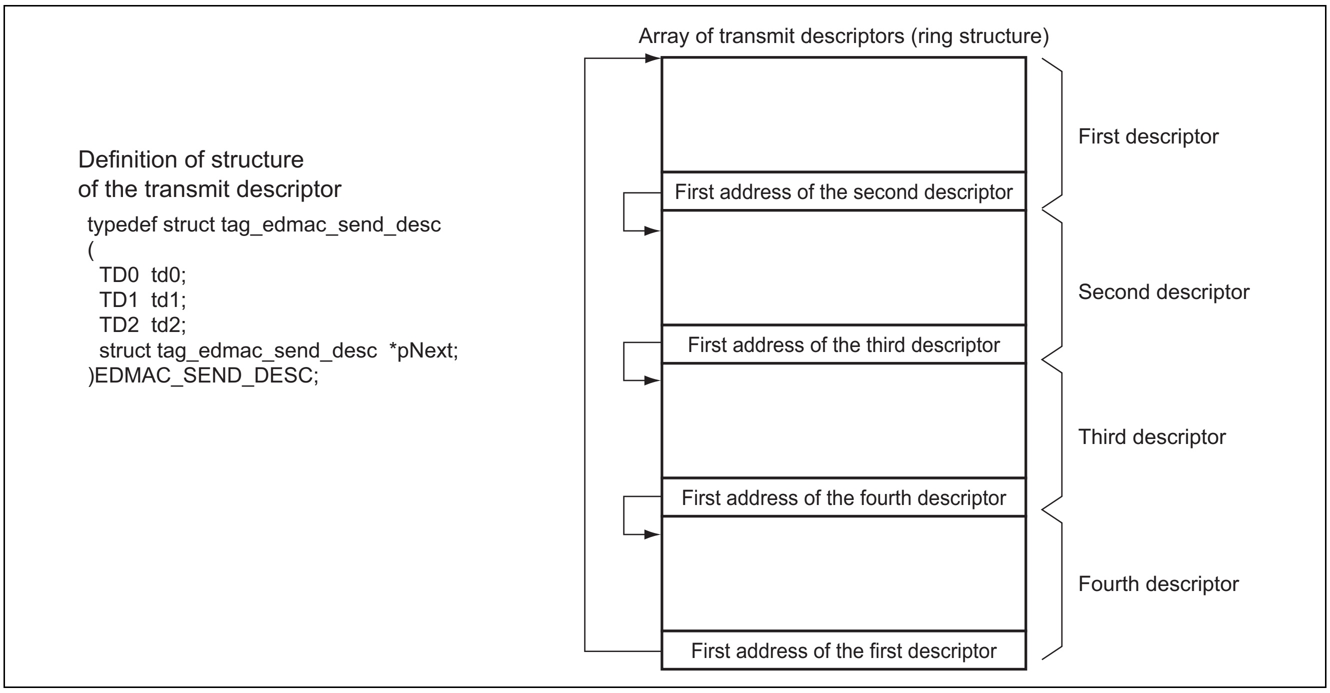

Unit: byte
7
1
6
6
2
46 to 1,500
4
Preamble
SFD
MAC address of the destination for transmission
MAC address of the source for transmission
Type/ length
Data section
CRC
Storage data in transmission buffer: 60 to 1,514 bytes

Figure 19 Ethernet Frame Format

2.7 Definition of Descriptors Used in the Sample Program

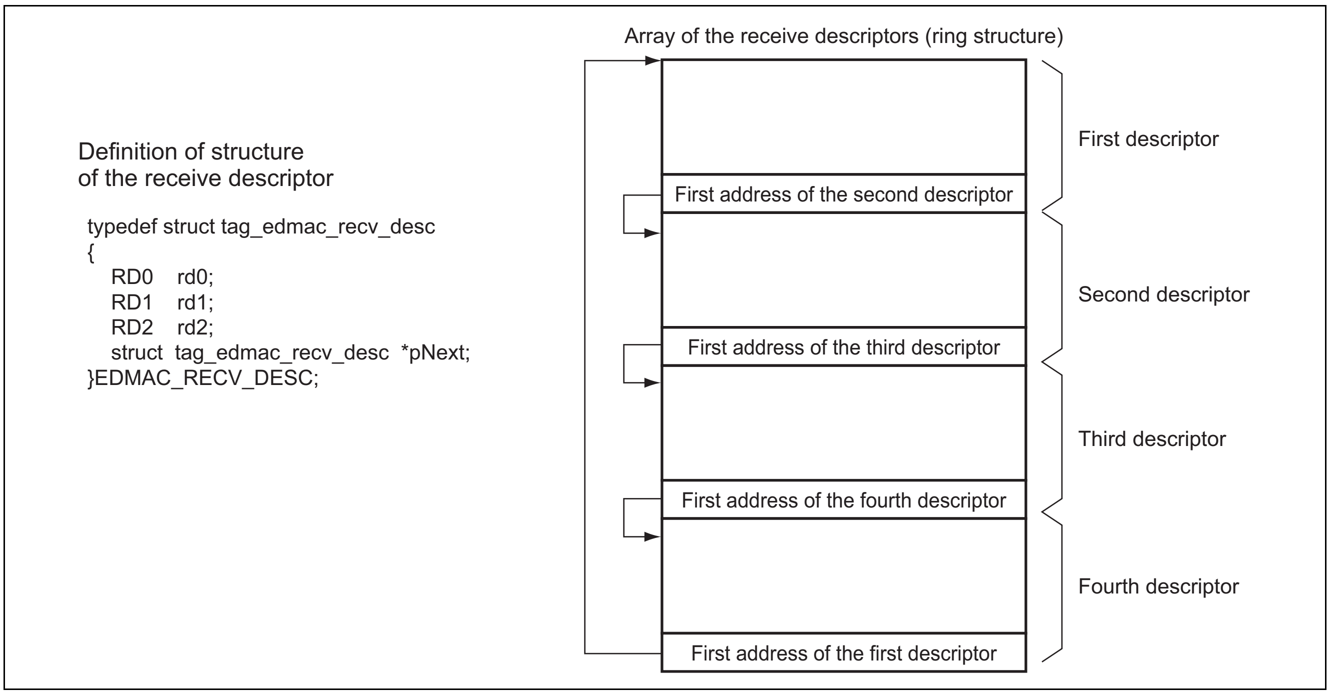
The E-DMAC does not use the padding area of a descriptor, this area is freely available to the user. In this sample program, this area is used to specify the address where the next descriptor starts, and this in conjunction with software is used to arrange the descriptors in a ring structure.

Figure 20 shows the definition of the transmit-descriptor structure in the sample program and an example of how the array of transmit descriptors is used. Figure 21 shows the definition of the receive-descriptor structure in the sample program and an example of how the array of receive descriptors is used.


Definition of structure of the transmit descriptor
Array of transmit descriptors (ring structure)
First address of the second descriptor
First address of the third descriptor
First address of the fourth descriptor
First address of the first descriptor
First descriptor
Second descriptor
Third descriptor
Fourth descriptor

```
typedef struct tag_edmac_send_desc
(
  TD0  td0;
  TD1  td1;
  TD2  td2;
  struct tag_edmac_send_desc  *pNext;
)EDMAC_SEND_DESC;
```

Figure 20 Definition of Transmit Descriptor and Usage Example of Transmit Descriptor Array


Definition of structure
of the receive descriptor
Array of the receive descriptors (ring structure)
First address of the second descriptor
First address of the third descriptor
First address of the fourth descriptor
First address of the first descriptor
First descriptor
Second descriptor
Third descriptor
Fourth descriptor

```
typedef struct tag_edmac_recv_desc
{
    RD0   rd0;
    RD1   rd1;
    RD2   rd2;
    struct  tag_edmac_recv_desc  *pNext;
}EDMAC_RECV_DESC;
```

Figure 21 Definition of Receive Descriptor and Usage Example of Receive Descriptor Array

2.8 Sequence of Processing by the Sample Program

Figures 22 to 31 show flows of handling the sample program.

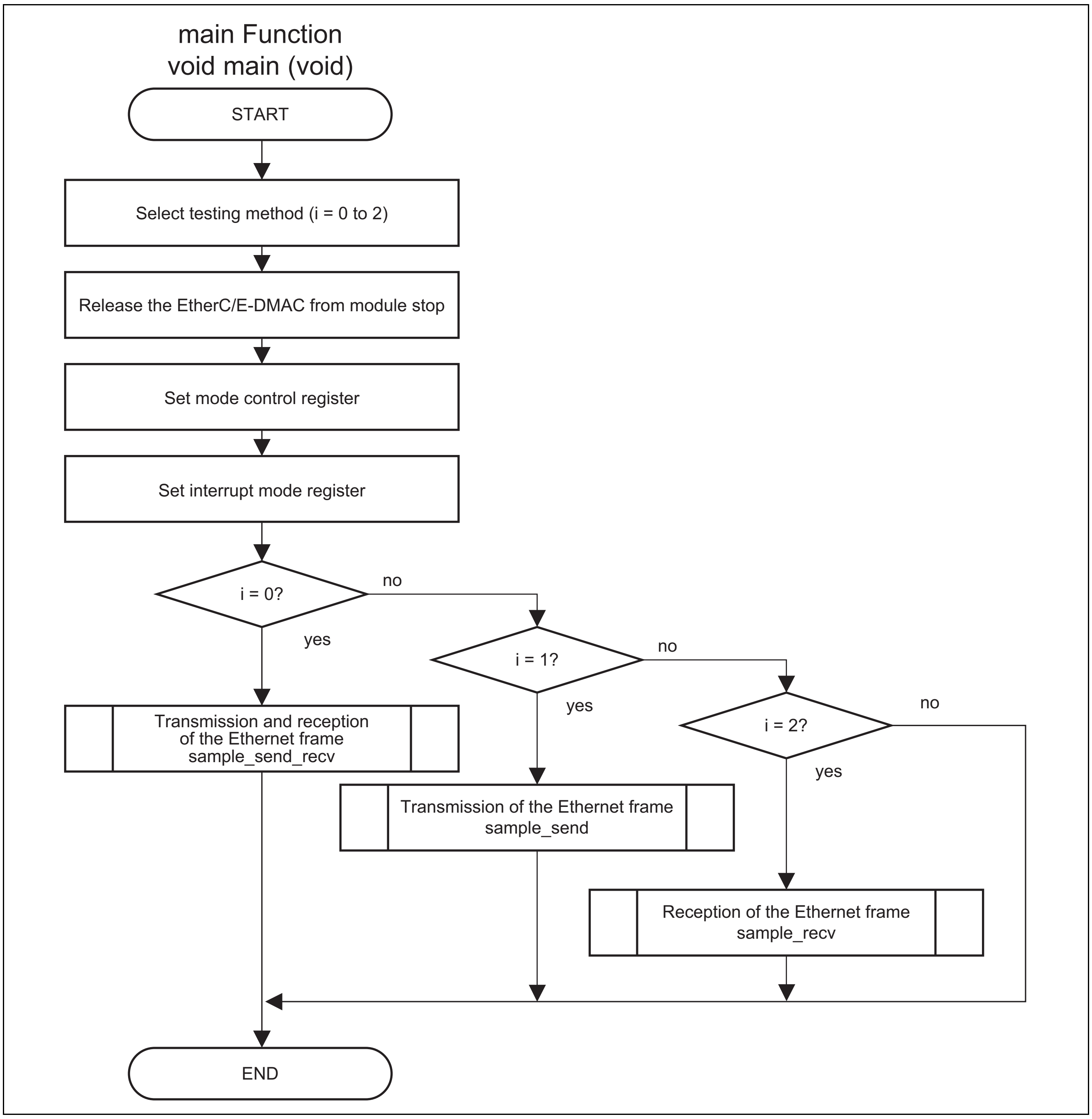

main Function
void main (void)
START
Select testing method (i = 0 to 2)
Release the EtherC/E-DMAC from module stop
Set mode control register
Set interrupt mode register
i = 0?
no
yes
i = 1?
no
yes
i = 2?
no
yes
Transmission and reception of the Ethernet frame sample_send_recv
Transmission of the Ethernet frame sample_send
Reception of the Ethernet frame sample_recv
END

Figure 22 Flow of Handling in the Sample Program (1)

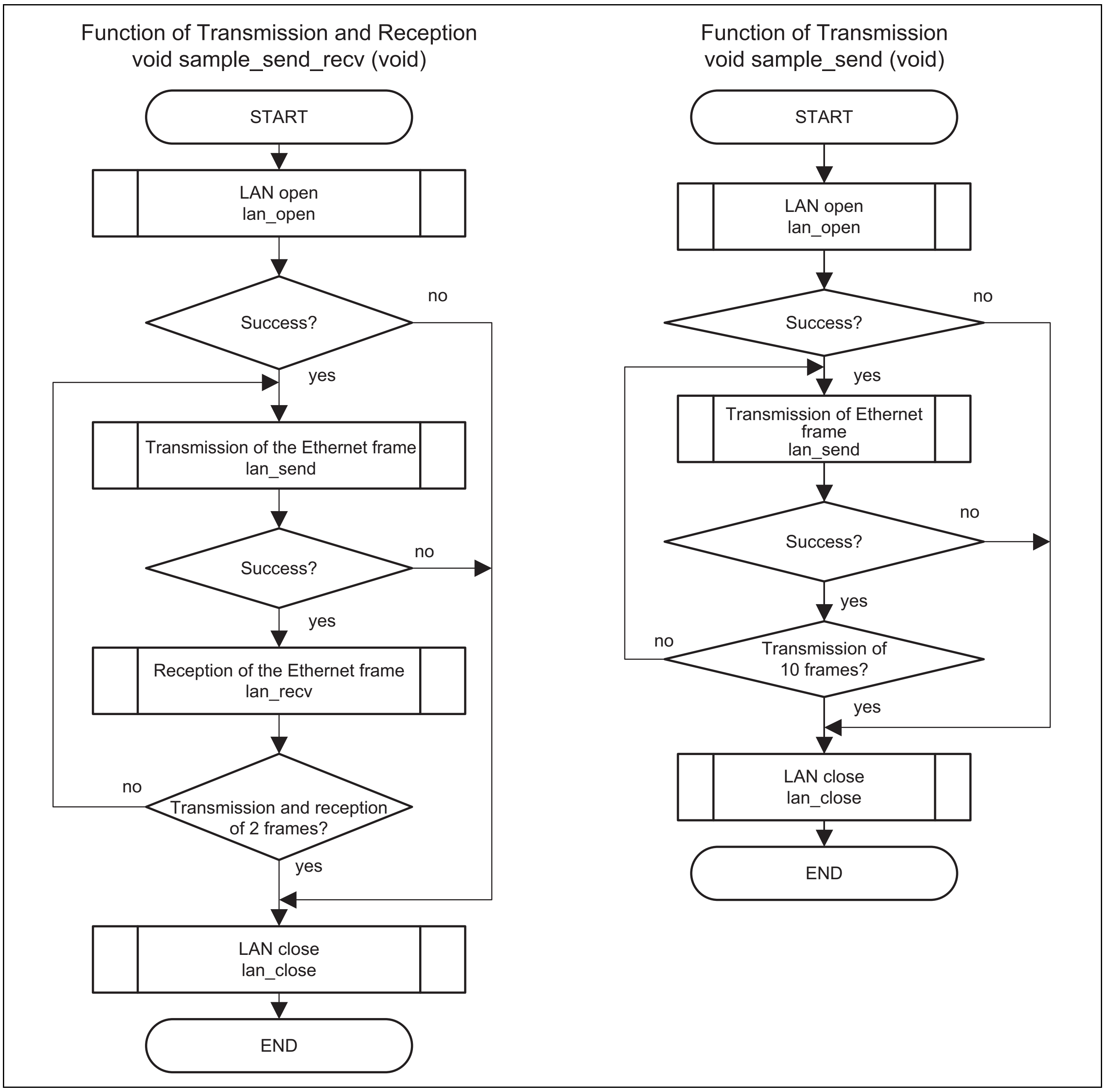

Function of Transmission and Reception
void sample_send_recv (void)
START
LAN open
lan_open
Success?
no
yes
Transmission of the Ethernet frame
lan_send
Success?
no
yes
Reception of the Ethernet frame
lan_recv
no
Transmission and reception of 2 frames?
yes
LAN close
lan_close
END
Function of Transmission
void sample_send (void)
START
LAN open
lan_open
Success?
no
yes
Transmission of Ethernet frame
lan_send
Success?
no
yes
no
Transmission of 10 frames?
yes
LAN close
lan_close
END

Figure 23 Flow of Handling in the Sample Program (2)

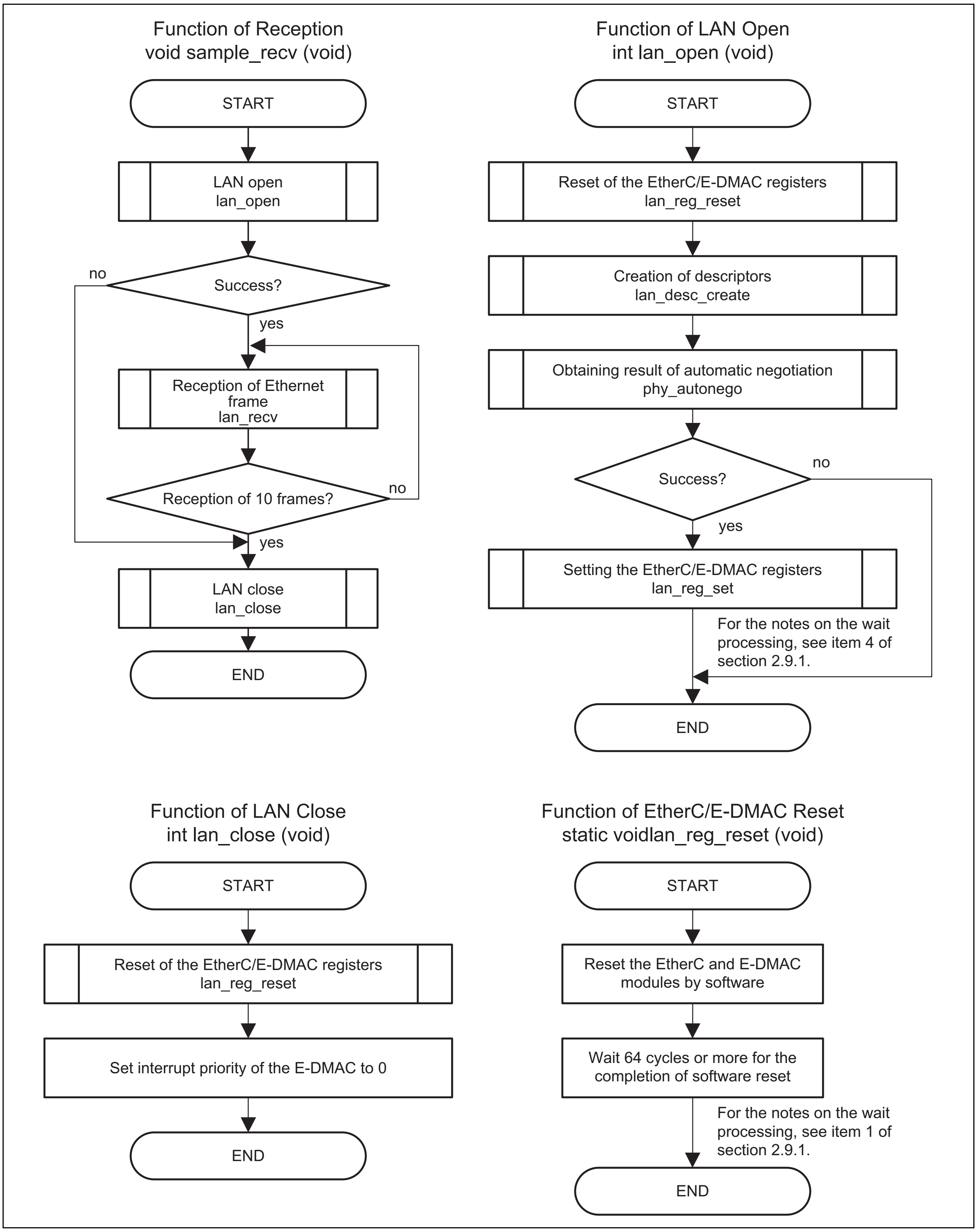

Function of Reception
void sample_recv (void)
START
LAN open
lan_open
Success?
no
yes
Reception of Ethernet frame
lan_recv
Reception of 10 frames?
no
yes
LAN close
lan_close
END
Function of LAN Open
int lan_open (void)
START
Reset of the EtherC/E-DMAC registers
lan_reg_reset
Creation of descriptors
lan_desc_create
Obtaining result of automatic negotiation
phy_autonego
Success?
no
yes
Setting the EtherC/E-DMAC registers
lan_reg_set
For the notes on the wait processing, see item 4 of section 2.9.1.
END
Function of LAN Close
int lan_close (void)
START
Reset of the EtherC/E-DMAC registers
lan_reg_reset
Set interrupt priority of the E-DMAC to 0
END
Function of EtherC/E-DMAC Reset
static voidlan_reg_reset (void)
START
Reset the EtherC and E-DMAC modules by software
Wait 64 cycles or more for the completion of software reset
For the notes on the wait processing, see item 1 of section 2.9.1.
END

Figure 24 Flow of Handling in the Sample Program (3)

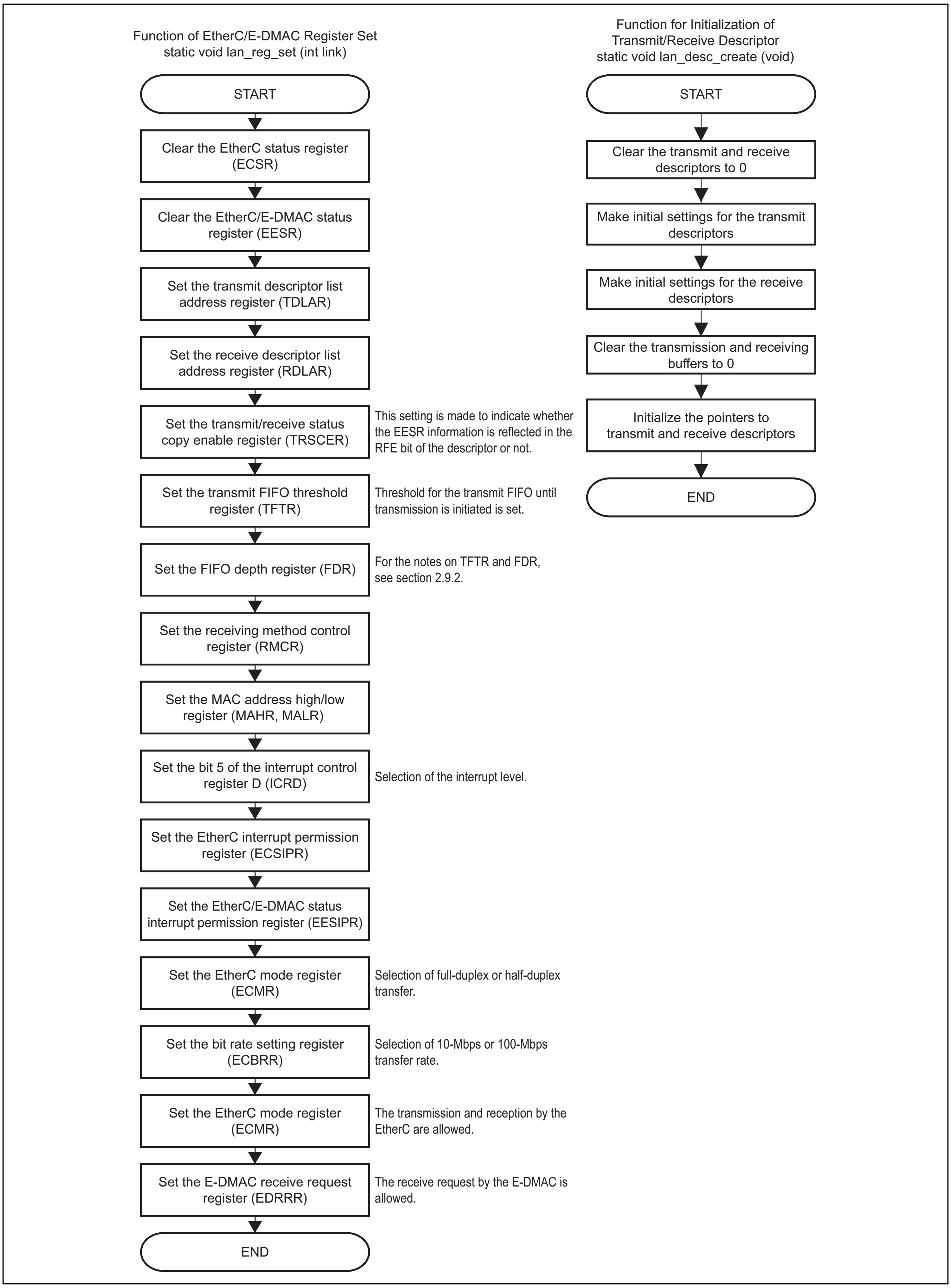

Function of EtherC/E-DMAC Register Set
static void lan_reg_set (int link)
START
Clear the EtherC status register (ECSR)
Clear the EtherC/E-DMAC status register (EESR)
Set the transmit descriptor list address register (TDLAR)
Set the receive descriptor list address register (RDLAR)
Set the transmit/receive status copy enable register (TRSCER)
This setting is made to indicate whether the EESR information is reflected in the RFE bit of the descriptor or not.
Set the transmit FIFO threshold register (TFTR)
Threshold for the transmit FIFO until transmission is initiated is set.
Set the FIFO depth register (FDR)
For the notes on TFTR and FDR, see section 2.9.2.
Set the receiving method control register (RMCR)
Set the MAC address high/low register (MAHR, MALR)
Set the bit 5 of the interrupt control register D (ICRD)
Selection of the interrupt level.
Set the EtherC interrupt permission register (ECSIPR)
Set the EtherC/E-DMAC status interrupt permission register (EESIPR)
Set the EtherC mode register (ECMR)
Selection of full-duplex or half-duplex transfer.
Set the bit rate setting register (ECBRR)
Selection of 10-Mbps or 100-Mbps transfer rate.
Set the EtherC mode register (ECMR)
The transmission and reception by the EtherC are allowed.
Set the E-DMAC receive request register (EDRRR)
The receive request by the E-DMAC is allowed.
END
Function for Initialization of Transmit/Receive Descriptor
static void lan_desc_create (void)
START
Clear the transmit and receive descriptors to 0
Make initial settings for the transmit descriptors
Make initial settings for the receive descriptors
Clear the transmission and receiving buffers to 0
Initialize the pointers to transmit and receive descriptors
END

Figure 25 Flow of Handling in the Sample Program (4)

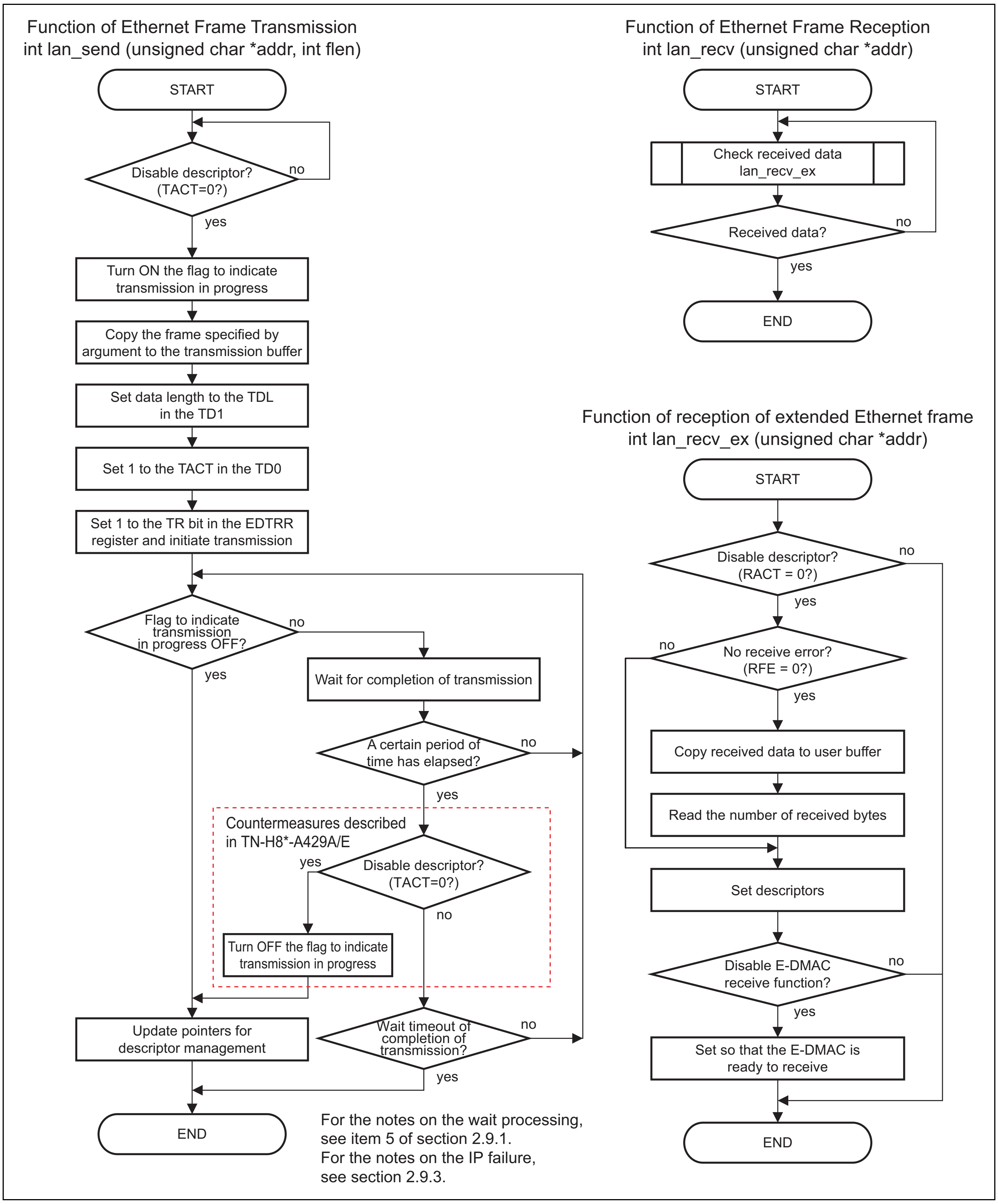

Function of Ethernet Frame Transmission
int lan_send (unsigned char *addr, int flen)
START
Disable descriptor? (TACT=0?)
no
yes
Turn ON the flag to indicate transmission in progress
Copy the frame specified by argument to the transmission buffer
Set data length to the TDL in the TD1
Set 1 to the TACT in the TD0
Set 1 to the TR bit in the EDTRR register and initiate transmission
Flag to indicate transmission in progress OFF?
no
yes
Wait for completion of transmission
A certain period of time has elapsed?
no
yes
Countermeasures described in TN-H8*-A429A/E
Disable descriptor? (TACT=0?)
yes
no
Turn OFF the flag to indicate transmission in progress
Update pointers for descriptor management
Wait timeout of completion of transmission?
no
yes
END
For the notes on the wait processing, see item 5 of section 2.9.1.
For the notes on the IP failure, see section 2.9.3.
Function of Ethernet Frame Reception
int lan_recv (unsigned char *addr)
START
Check received data lan_recv_ex
Received data?
no
yes
END
Function of reception of extended Ethernet frame
int lan_recv_ex (unsigned char *addr)
START
Disable descriptor? (RACT = 0?)
no
yes
No receive error? (RFE = 0?)
no
yes
Copy received data to user buffer
Read the number of received bytes
Set descriptors
Disable E-DMAC receive function?
no
yes
Set so that the E-DMAC is ready to receive
END

Figure 26 Flow of Handling in the Sample Program (5)

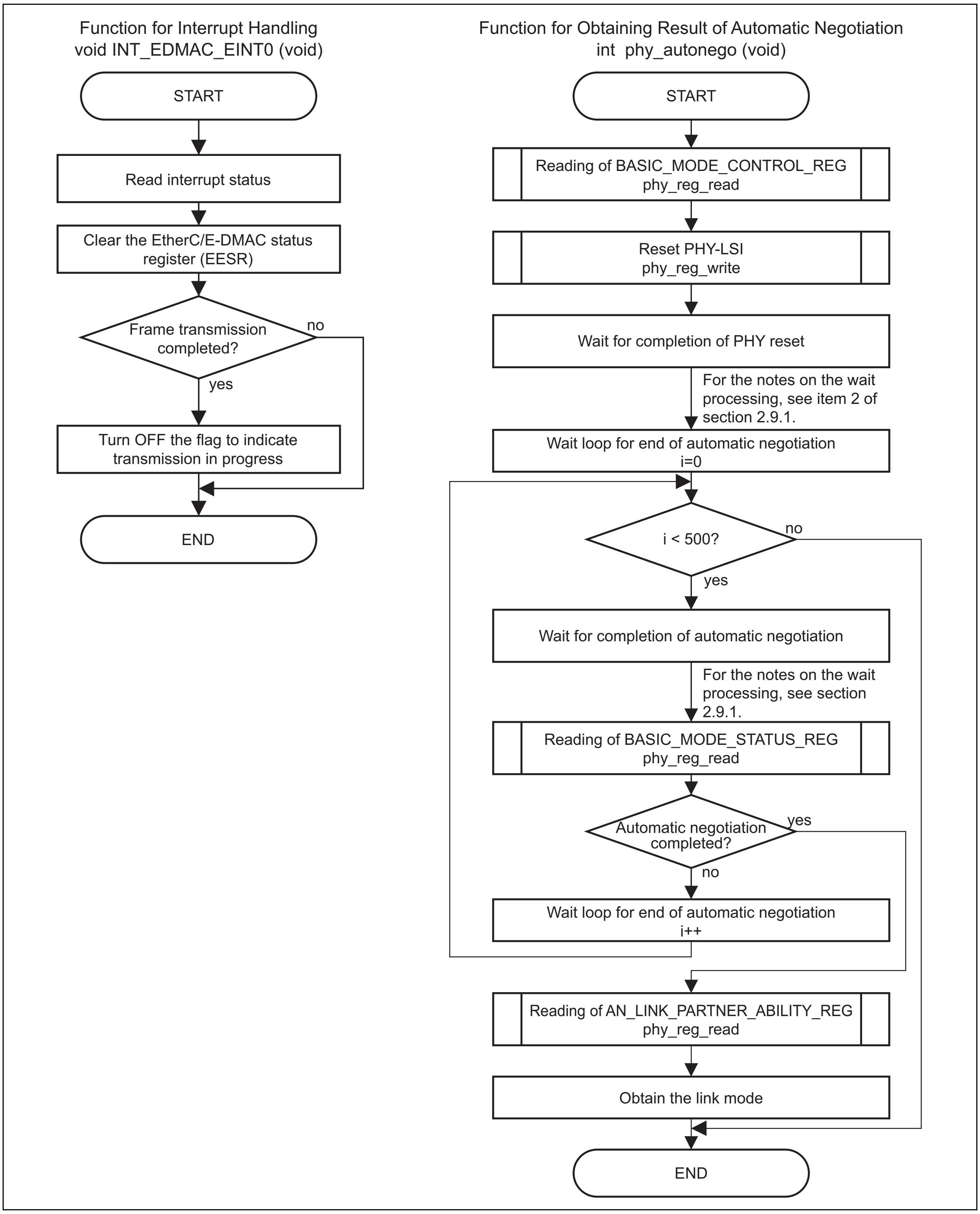

Function for Interrupt Handling
void INT_EDMAC_EINT0 (void)
START
Read interrupt status
Clear the EtherC/E-DMAC status register (EESR)
Frame transmission completed?
no
yes
Turn OFF the flag to indicate transmission in progress
END
Function for Obtaining Result of Automatic Negotiation
int phy_autonego (void)
START
Reading of BASIC_MODE_CONTROL_REG
phy_reg_read
Reset PHY-LSI
phy_reg_write
Wait for completion of PHY reset
For the notes on the wait processing, see item 2 of section 2.9.1.
Wait loop for end of automatic negotiation
i=0
i < 500?
no
yes
Wait for completion of automatic negotiation
For the notes on the wait processing, see section 2.9.1.
Reading of BASIC_MODE_STATUS_REG
phy_reg_read
Automatic negotiation completed?
yes
no
Wait loop for end of automatic negotiation
i++
Reading of AN_LINK_PARTNER_ABILITY_REG
phy_reg_read
Obtain the link mode
END

Figure 27 Flow of Handling in the Sample Program (6)

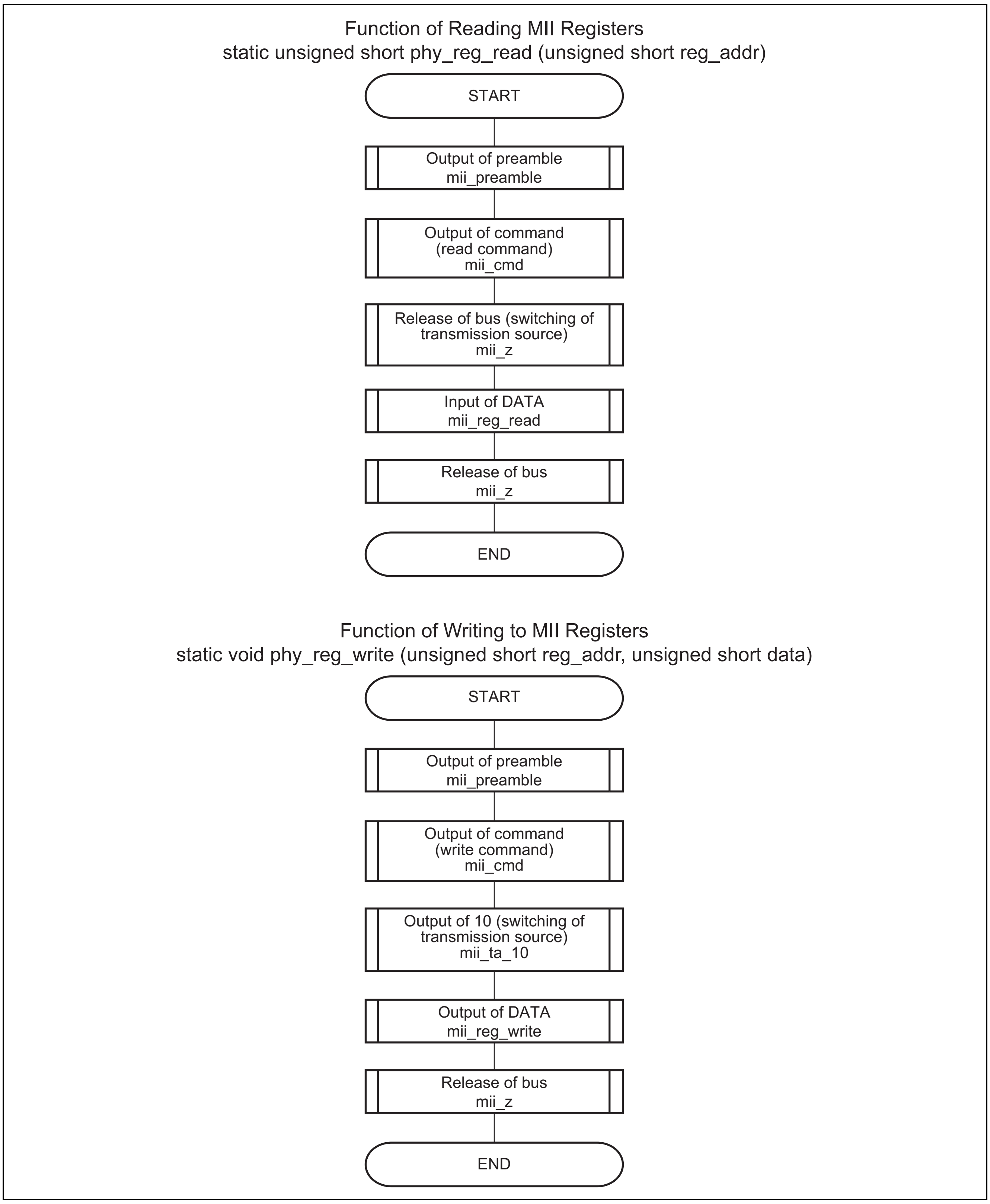

Function of Reading MII Registers
static unsigned short phy_reg_read (unsigned short reg_addr)
START
Output of preamble
mii_preamble
Output of command
(read command)
mii_cmd
Release of bus (switching of transmission source)
mii_z
Input of DATA
mii_reg_read
Release of bus
mii_z
END
Function of Writing to MII Registers
static void phy_reg_write (unsigned short reg_addr, unsigned short data)
START
Output of preamble
mii_preamble
Output of command
(write command)
mii_cmd
Output of 10 (switching of transmission source)
mii_ta_10
Output of DATA
mii_reg_write
Release of bus
mii_z
END

Figure 28 Flow of Handling in the Sample Program (7)

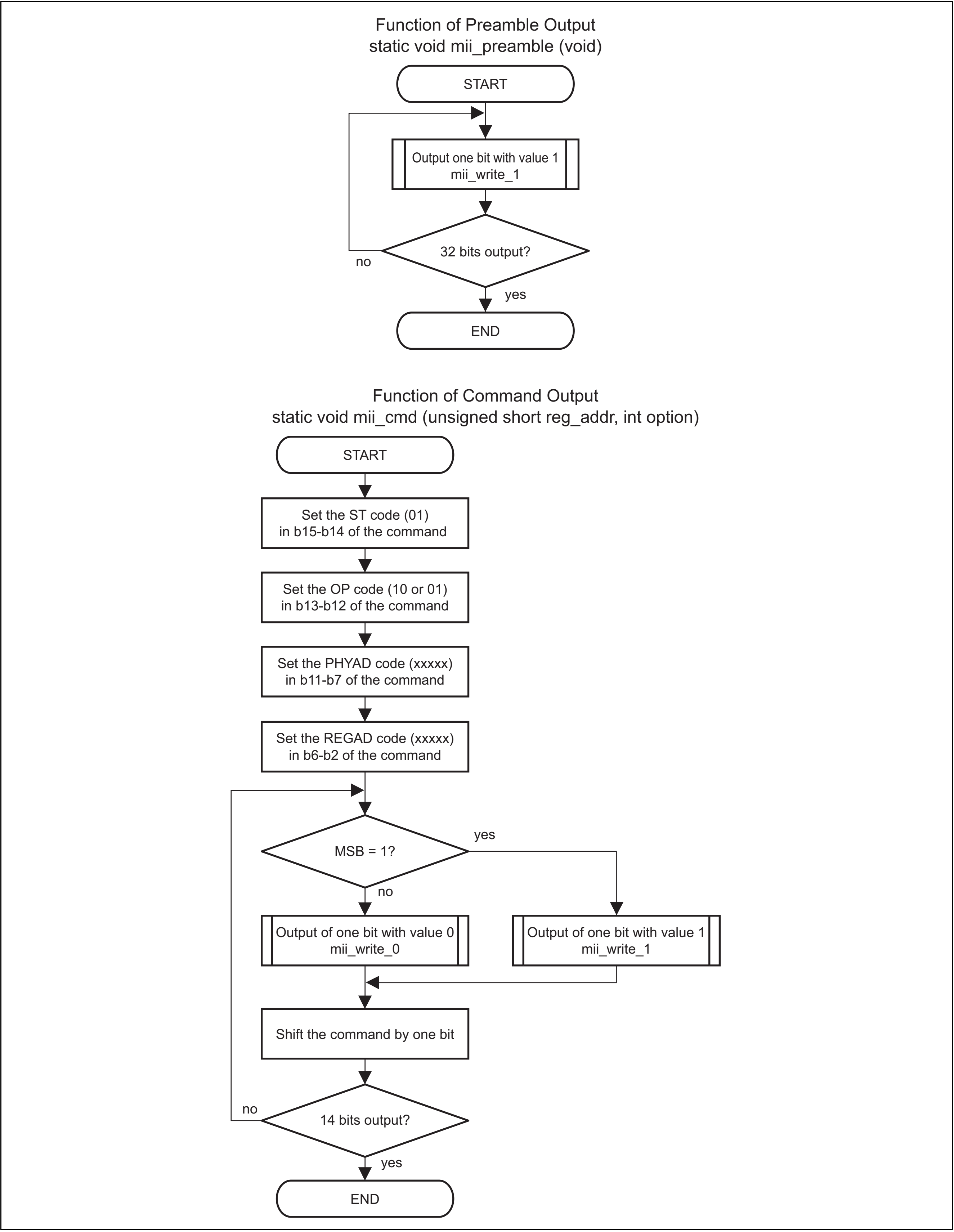

Function of Preamble Output
static void mii_preamble (void)
START
Output one bit with value 1
mii_write_1
32 bits output?
no
yes
END
Function of Command Output
static void mii_cmd (unsigned short reg_addr, int option)
START
Set the ST code (01)
in b15-b14 of the command
Set the OP code (10 or 01)
in b13-b12 of the command
Set the PHYAD code (xxxxx)
in b11-b7 of the command
Set the REGAD code (xxxxx)
in b6-b2 of the command
MSB = 1?
yes
no
Output of one bit with value 0
mii_write_0
Output of one bit with value 1
mii_write_1
Shift the command by one bit
14 bits output?
no
yes
END

Figure 29 Flow of Handling in the Sample Program (8)

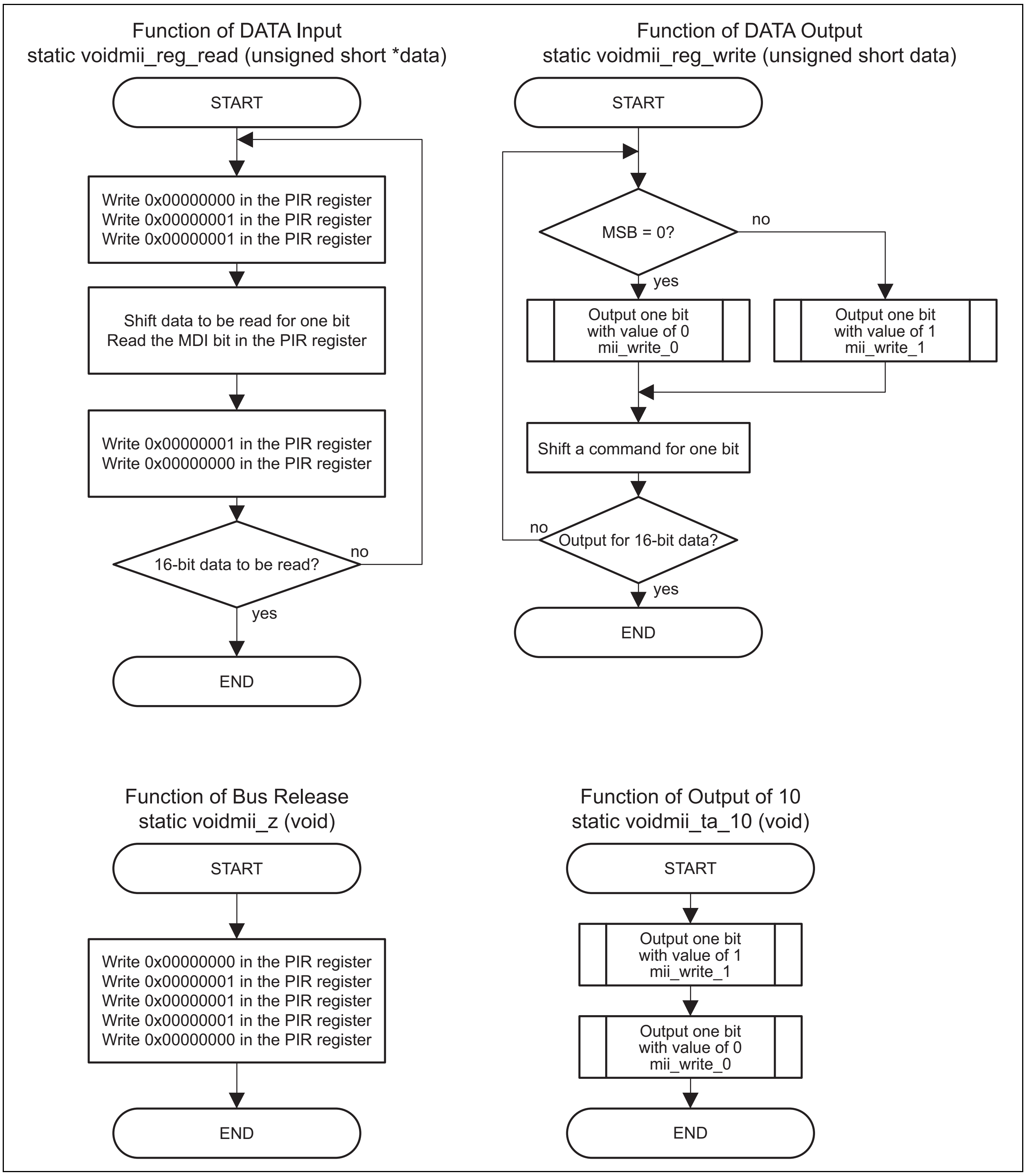

Function of DATA Input
static voidmii_reg_read (unsigned short *data)
START
Write 0x00000000 in the PIR register
Write 0x00000001 in the PIR register
Write 0x00000001 in the PIR register
Shift data to be read for one bit
Read the MDI bit in the PIR register
Write 0x00000001 in the PIR register
Write 0x00000000 in the PIR register
16-bit data to be read?
no
yes
END
Function of DATA Output
static voidmii_reg_write (unsigned short data)
START
MSB = 0?
no
yes
Output one bit with value of 0 mii_write_0
Output one bit with value of 1 mii_write_1
Shift a command for one bit
Output for 16-bit data?
no
yes
END
Function of Bus Release
static voidmii_z (void)
START
Write 0x00000000 in the PIR register
Write 0x00000001 in the PIR register
Write 0x00000001 in the PIR register
Write 0x00000001 in the PIR register
Write 0x00000000 in the PIR register
END
Function of Output of 10
static voidmii_ta_10 (void)
START
Output one bit with value of 1 mii_write_1
Output one bit with value of 0 mii_write_0
END

Figure 30 Flow of Handling in the Sample Program (9)

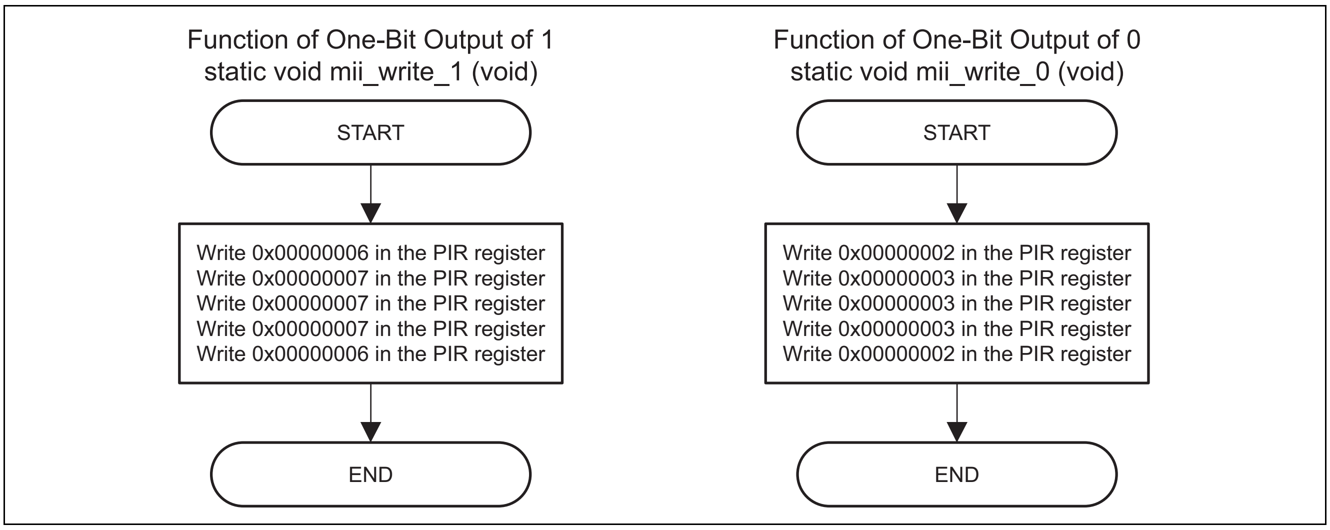

Function of One-Bit Output of 1
static void mii_write_1 (void)
START
Write 0x00000006 in the PIR register
Write 0x00000007 in the PIR register
Write 0x00000007 in the PIR register
Write 0x00000007 in the PIR register
Write 0x00000006 in the PIR register
END
Function of One-Bit Output of 0
static void mii_write_0 (void)
START
Write 0x00000002 in the PIR register
Write 0x00000003 in the PIR register
Write 0x00000003 in the PIR register
Write 0x00000003 in the PIR register
Write 0x00000002 in the PIR register
END

Figure 31 Flow of Handling in the Sample Program (10)

2.9 Notes on Sample Program

2.9.1 Notes on Wait Processing

The waiting time of this sample program is for reference. Furthermore, the code for the wait processing is written in the C language, so the waiting time will depend on the operating frequency and the compile option or the compiler version.

The waiting time must be evaluated to ensure that it suits the system in use.

The value set in this sample program is for the program on a system operating at 32 MHz.

1. Processing to Wait for Completion of Software Reset
   The EtherC and E-DMAC units are within the scope of a software reset so access to the registers of all Ethernet-related modules is inhibited while the reset is in progress. Since access to the registers of all Ethernet-related modules must not proceed until the software reset is completed (which takes 64 cycles), the sample program produces a corresponding period of waiting. This wait processing is handled by the following code of the lan_reg_reset function in the "ether.c" file (the 194th to 197th lines). To change the waiting time, change the value set in the local variable “t.”
   The value set in this sample program is for the program on a system operating at 32 MHz. This setting provides an ample margin over the required specification for the H8S/2472.

```
  /* ==== Wait for 64 cycles at φ (approx. 2 us@φ = 32 MHz) is required. ====
 */
  while(--t){
    /* wait */
  }
```

2. Processing to Wait for Completion of PHY-LSI Reset
   The phy_autonego function of this sample program resets the PHY-LSI. This wait processing is executed by the following code of the phy_autonego function in the "phy.c" file (the 130th to 136th lines). To change the length of the waiting time, change the values set in the local variables "t" and "i."
   This waiting time also covers "4. Processing to Wait for Stabilization of System Operation" described below. If this waiting time is shortened, communications may fail despite automatic negotiation having succeeded. Evaluate this waiting time on the system in use.

```
  /* ---- Wait ---- */
  for(i=0;i<1000;i++){
    t=0x27C0;
    while( --t){              /* approx. 2.9-ms wait counting@32 MHz */
        ;
    }
  }
```

3. Processing to Wait for Completion of Automatic Negotiation
   This sample program uses the automatic negotiaion function to select the communications method, performing wait processing until the automatic negotiation ends. This wait processing is handled by the following code of the phy_autonego function in the "phy.c" file (the 168th to 175th lines). To change this waiting time, change the local variables "t" and "i."

```
/* ==== Wait loop for end of automatic negotiation ==== */
for( i=0; i<500; i++){

  /* ---- approx. 10-ms wait ---- */
  t=36000;
  while( --t){             /* approx. 10-ms wait counting@32 MHz */
     ;
  }
```

4. Processing to Wait for Stabilization of System Operation
   This sample program uses the automatic negotiation function to select the communications method. If the completion of automatic negotiation between the H8S/2472 and whatever is connected to the H8S/2472 takes a long time, communications may fail despite that automatic negotiation having succeeded.
   If whatever is connected to the H8S/2472 is not ready to receive data despite the PHY in the H8S/2472 having succeeded in automatic negotiation, execute this wait processing on the H8S/2472 so that it waits until the connected device is ready to receive data.
   The time that whatever is connected to the H8S/2472 takes to become ready to receive data depends on the system. Evaluate this waiting time for the system in use.
   The waiting time to stabilize the operation of the system is covered in "2. Waiting Time of Completion of PHY-LSI Reset" described above.
   To change this waiting time, enable the following code within the lan_open function in the "ether.c file (the 139th to 143rd lines). Or, change the processing to wait until completion of PHY-LSI reset.

```
#if 0
  /* Delay to stabilize */
  /* Set the count according to the system */
  for( i = 0 ; i < 0x00100000 ; i++ );
#endif
```

5. Processing to Wait for Completion of Transmission

   The function for transmission (the lan_send function) of this sample program uses the following code (the 392nd to 415th lines in the "ether.c" file) to confirm the completion of transmission.

```
  /* ==== Confirmation of completion of data transmission  ==== */
  while(tx_flag0 == TX_FLAG_ON){  /* A flag indicating transmission in progress is
turned ON */

    /* ==== approx. 10us wait ==== */
    for(w=0;w<16;w++){
        ;
    }

    /* ==== Workaround of Technical Update "TN-H8*-A429A/J" or "TN-H8*-A429A/E" ==== */
    if((--t1ms) <= 0){
        t1ms = 100;   /*   */
        if(psenddesc0->td0.BIT.TACT == 0){
            tx_flag0 = TX_FLAG_OFF;
            break;
        }
        /* ==== Workaround of Technical Update "TN-H8*-A428A/J" or "TN-H8*-A428A/E" ====
*/
        else if((--t400ms) <= 0){
            return SEND_NG;
        }
        else{
            /* DO NOTHING */
        }
    }
  }
```

   To confirm the completion of transmission, this sample program checks the global variable "tx_flag0" once per interval of waiting time described under the "for" statement. The variable "tx_flag0" indicates the state in terms of whether or not transmission is completed, and is updated by an interrupt function.

   Confirming the completion of transmission includes a countermeasure against a malfunction of an IP module and timeout processing. For details on the malfunction, see technical updates TN-H8*-A429A/E and TN-H8*-A428A/E. To work around the malfunction of the IP module, the TACT bit is checked with the timing set by the local variable "t1ms." This timing is set as desired. Use the local variable t400ms to set the time for timeout of waiting for the completion of transmission. The time is based on the maximum times (maximum time to transmit a single frame of data and the maximum flow-control time) described in technical update TN-H8*-A428A/E.

   Change the waiting times by changing the three local variables "w," "t1ms," and "t400ms."

   These waiting times affect the performance in transmission. Evaluate the waiting times on the system in use.

2.9.2 Notes on Changing Values Set in FIFO Depth Register (FDR) and Transmit FIFO Threshold Register (TFTR)

As a workaround for a malfunction of an IP module in the H8S/2472, this sample program sets the depth of the FIFO depth register (FDR) to 2,048 bytes and puts the transmit FIFO threshold register (TFTR) in the store-and-forward mode.

If the FIFO depth register (FDR) and the transmit FIFO threshold register (TFTR) are set to values other than those in the sample program, the malfunction described in technical update TN-H8*-A428A/E may appear. Do not change the values from those set in this sample program.

Note that if the values in FDR and TFTR must be changed, working around the malfunction in accord with technical update TN-H8*-A428A/E.

2.9.3 Notes on Malfunctions in the IP Modules of the H8S/2472

The IP modules of the H8S/2472 have two malfunctions. For details, see technical updates TN-H8*-A429A/E and TNH8*-A428A/E.

This sample program deals with the malfunctions as described in note 5 "Processing to Wait for Completion of Transmission" of section 2.9.1, "Notes on Wait Processing," and section 2.9.2, "Notes on Changing Values Set in FIFO Depth Register (FDR) and Transmit FIFO Threshold Register (TFTR)."

3. Listing of Sample Program

3.1 Sample Program Listing: "LAN_2472.c"

```
/*******************************************************************************
* DISCLAIMER

* This software is supplied by Renesas Electronics Corporation and is only
* intended for use with Renesas products. No other uses are authorized.

* This software is owned by Renesas Electronics Corporation and is protected under
* all applicable laws, including copyright laws.

* THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES
* REGARDING THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY,
* INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NON-INFRINGEMENT. ALL SUCH WARRANTIES ARE EXPRESSLY
* DISCLAIMED.

* TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
* ELECTRONICS CORPORATION NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
* FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
* FOR ANY REASON RELATED TO THIS SOFTWARE, EVEN IF RENESAS OR ITS
* AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

* Renesas reserves the right, without notice, to make changes to this
* software and to discontinue the availability of this software.
* By using this software, you agree to the additional terms and
* conditions found by accessing the following link:
* http://www.renesas.com/disclaimer
*******************************************************************************
* Copyright (C) 2010 Renesas Electronics Corporation. All rights reserved.
*******************************************************************************
* File Name    : LAN_2472.c
* Version      : 1.01
* Device(s)    : H8S/2472
* Tool-Chain   : HEW, H8S,H8/300 Standard Toolchain (V.6.2.2.0)
* OS           : None
* H/W Platform : R0K402472D000BR,R0K402472D001BR,R0K402472D002BR
* Description  : Main Program
* Operation    :
* Limitations  : None
******************************************************************************
* History : DD.MM.YYYY Version Description
*         : 18.10.2007 1.00    First Release
*         : 26.04.2010 1.01    Modification of test program
*******************************************************************************/


/*******************************************************************************
Includes   <System Includes> , "Project Includes"
*******************************************************************************/
#include <machine.h>
#include "ether\iodefine2472.h"
#include "ether\ether.h"

/*******************************************************************************
Typedef definitions
*******************************************************************************/

/*******************************************************************************
Macro definitions
*******************************************************************************/
```

```

/******************************************************************************
Imported global variables and functions (from other files)
******************************************************************************/

/******************************************************************************
Exported global variables and functions (to be accessed by other files)
******************************************************************************/

/******************************************************************************
Private global variables and functions
******************************************************************************/
#ifdef __cplusplus
extern "C" {
void abort(void);
#endif
void main(void);
#ifdef __cplusplus
}
#endif

static unsigned char frame[] =
{
    /* MAC header */
    0x00,0x0e,0x35,0x18,0x34,0xfa,  /* Destination MAC Address(00-0E-35-18-34-FA) */
    0x00,0x11,0x25,0xbc,0xfd,0x0a,  /* Source MAC Address(00-11-25-BC-FD-0A) */
    0x08,0x00,                      /* Type(IP) */
    /* IP header */
    0x45,0x00,0x00,0x2e,            /* Version(IPv4), IHL(20byte), TOS, Total Leneght(46byte), */
    0x02,0xf6,0x00,0x00,            /* Identification, Flags, Fragment Offset */
    0x80,0x01,0x43,0x67,            /* TTL, Protocol(ICMP), Header Checksum */
    0xac,0x1e,0x4e,0x24,            /* Source Address(172.30.78.36) */
    0xac,0x1e,0x4e,0x22,            /* Destination Address(172.30.78.34) */
    /* ICMP header(Echo request) */
    0x08,0x00,0xb1,0xff,            /* Type, Code, Checksum */
    0x02,0x00,0x04,0x00,            /* Identifier, Sequence Number */
    /* Data */
    0x40,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00
};

/* ==== Declaration of global variables ==== */
unsigned char user_buffer[10][1520];
unsigned short    receive_size[10];


/*""FUNC COMMENT""*************************************************************
* Outline     : Sample program "main"
*-----------------------------------------------------------------------------
* Declaration : void sample_send_recv(void)
*-----------------------------------------------------------------------------
* Description : Two rounds of transmitting one frame and receiving one frame proceed.
*-----------------------------------------------------------------------------
* Argument        : None
*-----------------------------------------------------------------------------
* Return Value    : None
*-----------------------------------------------------------------------------
* Note            :
*""FUNC COMMENT END""*********************************************************/
void sample_send_recv(void)
{
    OPEN_STATUS      opensts;
    SEND_STATUS      sendsts;
    CLOSE_STATUS  closests;

    static int i = 10;

    /* ==== Initialization of the EtherC/E-DMAC, PHY, and buffer memory  ==== */
    opensts = lan_open();     /*  ch0 is selected */

    /* ==== Transmission if open  ==== */
    if(opensts == OPEN_OK){
      /* ==== Packet transmission  ==== */
      for(i=0;i<2;i++){
          sendsts = lan_send(frame,sizeof(frame));
          if(sendsts == SEND_NG){ /* Transmission error */
              break;
          }
          receive_size[i] = lan_recv(&user_buffer[i][0]);
      }
    }

    /* ==== LAN close ==== */
    closests = lan_close();
    if(closests == CLOSE_NG){
      ;   /* waiting */
    }
}

/*""FUNC COMMENT""*************************************************************
* Outline     : Sample program "main"
*-----------------------------------------------------------------------------
* Declaration : void sample_main(void)
*-----------------------------------------------------------------------------
* Description : 10 frames are transmitted from the Ethernet.
*-----------------------------------------------------------------------------
* Argument        : None
*-----------------------------------------------------------------------------
* Return Value    : None
*-----------------------------------------------------------------------------
* Note            :
*""FUNC COMMENT END""*********************************************************/
void sample_send(void)
{
    OPEN_STATUS      opensts;
    SEND_STATUS      sendsts;
    CLOSE_STATUS  closests;

    int i = 10;

    /* ==== Initialization of the EtherC/E-DMAC, PHY, and buffer memory  ==== */
    opensts = lan_open();     /*  ch0 is selected */

    /* ==== Transmission if open  ==== */
    if(opensts == OPEN_OK){

      /* ==== Packet transmission  ==== */
      for(i=0;i<10;i++){
          sendsts = lan_send(frame,sizeof(frame));
          if(sendsts == SEND_NG){ /* Transmission error */
              break;
          }
      }
    }

    /* ==== LAN close ==== */
    closests = lan_close();
    while(closests == CLOSE_NG){
      ;   /* waiting */
    }
}

/*""FUNC COMMENT""***************************************************************
* Outline     : Sample program "main"
*-------------------------------------------------------------------------------
* Declaration : void sample_recv(void)
*-------------------------------------------------------------------------------
* Description : 10 frames are received from the Ethernet.
*-------------------------------------------------------------------------------
* Argument        : None
*-------------------------------------------------------------------------------
* Return Value    : None
*-------------------------------------------------------------------------------
* Note            :
*""FUNC COMMENT END""**********************************************************/
void sample_recv(void)
{
    OPEN_STATUS     opensts;
    CLOSE_STATUS    closests;

    int i;

    /* ==== Initialization of the EtherC/E-DMAC, PHY, and buffer memory  ==== */
    opensts = lan_open();      /*  Ch0 is selected */

    /* ==== Reception if open  ==== */
    if(opensts == OPEN_OK){

        /* ==== Packet reception  ==== */
        for(i=0;i<10;i++){
            receive_size[i] = lan_recv(&user_buffer[i][0]);
        }
    }

    /* ==== LAN close ==== */
    closests = lan_close();
    while(closests == CLOSE_NG){
        ;   /* waiting */
    }
}


void main(void)
{
    int   i;
    unsigned char dummy;

    i = 0;

    dummy = SYSTEM.MDCR.BYTE;
    SYSTEM.SUBMSTPBH.BIT.EtherC = 0;
    SYSTEM.SUBMSTPBH.BIT.EDMAC = 0;
    SYSTEM.MDCR.BIT.EXPE = 0;     // 0: Single-chip mode   1: Extended mode
    SYSTEM.SYSCR.BIT.INTM = 1;       // Interrupt control mode1

    set_imask_exr(0);
    and_ccr(0x3F); // Interrupt level 0

    switch(i){
      case 0:
          sample_send_recv();
```

```
                while(1);
                break;
            case 1:
                sample_send();
                while(1);
                break;
            case 2:
                sample_recv();
                while(1);
                break;
            default:
                break;
        }
}


#ifdef __cplusplus
void abort(void)
{

}
#endif
```

3.2 Sample Program Listing: "ether.c"

```
/*******************************************************************************
* DISCLAIMER

* This software is supplied by Renesas Electronics Corporation and is only
* intended for use with Renesas products. No other uses are authorized.

* This software is owned by Renesas Electronics Corporation and is protected under
* all applicable laws, including copyright laws.

* THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES
* REGARDING THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY,
* INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NON-INFRINGEMENT. ALL SUCH WARRANTIES ARE EXPRESSLY
* DISCLAIMED.

* TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
* ELECTRONICS CORPORATION NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
* FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
* FOR ANY REASON RELATED TO THIS SOFTWARE, EVEN IF RENESAS OR ITS
* AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

* Renesas reserves the right, without notice, to make changes to this
* software and to discontinue the availability of this software.
* By using this software, you agree to the additional terms and
* conditions found by accessing the following link:
* http://www.renesas.com/disclaimer
********************************************************************************
* Copyright (C) 2010 Renesas Electronics Corporation. All rights reserved.
********************************************************************************
* File Name    : ether.c
* Version      : 1.02
* Device(s)    : H8S/2472
* Tool-Chain   : HEW, H8S,H8/300 Standard Toolchain (V.6.2.2.0)
* OS           : None
* H/W Platform : R0K402472D000BR,R0K402472D001BR,R0K402472D002BR
* Description  : Ethernet transmission processing
* Limitations  : None
*******************************************************************************
* History : DD.MM.YYYY Version Description
*         : 15.05.2007 1.00    First Release
*         : 19.04.2010 1.01    Bug fix of reception
*         : 29.07.2010 1.02    Modification of wait counter
*         :                    Modification of send complete check
*******************************************************************************/


/*******************************************************************************
Includes   <System Includes> , "Project Includes"
*******************************************************************************/
#include <machine.h>
#include <string.h>
#include "iodefine2472.h"
#include "ether.h"
#include "phy.h"

/*******************************************************************************
Typedef definitions
*******************************************************************************/

/*******************************************************************************
Macro definitions
*******************************************************************************/
```

```
//#define ETHER_LOOP_BACK

/******************************************************************************
Imported global variables and functions (from other files)
******************************************************************************/

/******************************************************************************
Exported global variables and functions (to be accessed by other files)
******************************************************************************/

/******************************************************************************
Private global variables and functions
******************************************************************************/
/* ==== Declaration of prototype ==== */
static void lan_reg_reset(void);
static void lan_reg_set(int link);
static void lan_desc_create(void);

/* ==== Declaration of variables ==== */
volatile static int tx_flag0;
static volatile  EDMAC_SEND_DESC  *psenddesc0;
static volatile  EDMAC_RECV_DESC   *precvdesc0;

/* Since descriptors need to be placed on 16-byte boundaries,
 a section on a 16-byte boundary is reserved for the descriptors. */
#pragma section TXRXBUFFDESC
static    TXRX_BUFFER_SET  buffer0;
static  TXRX_DESCRIPTOR_SET   descriptor0;
#pragma section

/*""FUNC COMMENT""*************************************************************
* Outline     : LAN Open Functions
*-----------------------------------------------------------------------------
* Declaration : int lan_open(void)
*-----------------------------------------------------------------------------
* Description : The E-DMAC, EtherC, transmit/receive descriptors, and memory
*               for the transmission/receiving buffers are initialized.
*             : The result of automatic negotiation by PHY-LSI is obtained,
*               and an error is returned if automatic negotiation is failed.
*-----------------------------------------------------------------------------
* Argument        : None
*-----------------------------------------------------------------------------
* Return Values : 0(OPEN_OK)  : Success in opening
*               : -1(OPEN_NG)  : Failure in opening
*-----------------------------------------------------------------------------
* Note            :
*""FUNC COMMENT END""*********************************************************/
int lan_open(void)
{
    unsigned int physts;
    // volatile unsigned long i;

    /* ==== Reset of the EtherC/E-DMAC ==== */
    lan_reg_reset();

    /* ==== Initialization of transmit/receive descriptors ==== */
    lan_desc_create();

#ifdef ETHER_LOOP_BACK
    MAC0.ECMR.BIT.ILB =1;
    physts = FULL_TX;
#else
    /* ==== Obtaining result of automatic negotiation by PHY-LSI ==== */
    physts = phy_autonego();
```

```
#endif

    /* ==== In case of success in automatic negotiation ==== */
    if(physts != NEGO_FAIL){

      /* ==== Setting of EtherC/E-DMAC registers ==== */
      lan_reg_set(physts);
    }
    else{
      return OPEN_NG;
    }

#if 0
    /* Delay to stabilize */
    /* Set the count according to the system */
    for( i = 0 ; i < 0x00100000 ; i++ );
#endif

    return OPEN_OK;
}


/*""FUNC COMMENT""***************************************************************
* Module Outline  : LAN Close Function
*-------------------------------------------------------------------------------
* Declaration     : int lan_close(void)
*-------------------------------------------------------------------------------
* Description     : The Ether function is halted, and transmission and reception are prohibited.
*-------------------------------------------------------------------------------
* Argument            : void
*-------------------------------------------------------------------------------
* Return Value        : 0(CLOSE_OK) : Success in closing
*                   : -1(CLOSE_NG) : Failure in closing
*-------------------------------------------------------------------------------
* Note                :
*""FUNC COMMENT END""**********************************************************/
int lan_close(void)
{
    /* ==== Reset of registers related to the EtherC and E-DMAC ==== */
    lan_reg_reset();

    /* ==== Setting of the interrupt control register (ICRD5) ==== */
    INT.ICRD.BIT.ICRD5 = 0;

    return CLOSE_OK;
}

/*""FUNC COMMENT""***************************************************************
* Outline     : EtherC/E-DMAC Reset Function
*-------------------------------------------------------------------------------
* Declaration : static void lan_reg_reset(void)
*-------------------------------------------------------------------------------
* Description : Software of the E-DMAC and EtherC is reset.
*-------------------------------------------------------------------------------
* Argument        : None
*-------------------------------------------------------------------------------
* Return Value    : None
*-------------------------------------------------------------------------------
* Note            : -
*""FUNC COMMENT END""**********************************************************/
static void lan_reg_reset(void)
{
    volatile int t = 10;  /* For approx. 3-us wait @32 MHz */

    /* ==== Setting of the E-DMAC mode register (EDMR) ==== */
    EDMAC0.EDMR.BIT.SWR =1;

    /* ==== Wait for 64 cycles at φ (approx. 2 us@φ = 32 MHz) is required. ==== */
    while(--t){
      /* wait */
    }
}

/*""FUNC COMMENT""***************************************************************
* Outline     : Initialization of the EtherC/E-DMAC Registers
*-------------------------------------------------------------------------------
* Declaration : static void lan_reg_set(int link)
*-------------------------------------------------------------------------------
* Description : Registers of the E-DMAC and EtherC are set so that
*               transmission and receiving operations are allowed.
*             : Settings for reception have been made but receiving
*               operations are not initiated.
*             : The initial value of TSU_FWSLC is changed so that signals
*               from pins CAMSEN0 and CAMSEN1 are not referred to during
*               receiving operations.
*-------------------------------------------------------------------------------
* Argument        : int link: I : Result of automatic negotiation by PHY
*             : HALF_10M(1), FULL_10M(2), HALF_TX(3), FULL_TX(4)
*-------------------------------------------------------------------------------
* Return Value    : None
*-------------------------------------------------------------------------------
* Note            : This function is based on the assumption that an external
*               CAM is not used.
*""FUNC COMMENT END""**********************************************************/
static void lan_reg_set(int link)
{
    /* ==== Clear of the EtherC status register (ECSR) ==== */
    MAC0.ECSR.LONG = 0x00000007; /* Clear of 1 write */

    /* ==== Clear of the EtherC/E-DMAC status register (EESR) ==== */
    EDMAC0.EESR.LONG = 0x47FF0F9F;    /* Clear of 1 write */

    /* ==== Setting of the transmit descriptor list address register (TDLAR) ==== */
    EDMAC0.TDLAR = descriptor0.send_desc;

    /* ==== Setting of the receive descriptor list address register (RDLAR) ==== */
    EDMAC0.RDLAR = descriptor0.recv_desc;

    /* ==== Setting of the transmit/receive status copy enable register (TRSCER) ==== */
    EDMAC0.TRSCER.LONG = 0x00000000;

    /* ==== Setting of transmit FIFO threshold register (TFTR) ==== */
    EDMAC0.TFTR.LONG = 0x00000000;   /* Store and forward mode */

    /* ==== Setting of the FIFO depth register (FDR) ==== */
    EDMAC0.FDR.LONG = 0x00000707;/* Capacity of transmit/receive FIFO is set to 2 KB. */

    /* ==== Setting of the receiving method control register (RMCR) ==== */
    EDMAC0.RMCR.BIT.RNC = 0x1;   /* Continuous reception */

    /* ==== Setting of the MAC address high register/MAC address low register (MAHR, MALR) ==== */
    MAC0.MAHR.LONG = MAC_ADDRESS_HIGH0;
    MAC0.MALR.LONG = MAC_ADDRESS_LOW0;

    /* ==== Setting of the interrupt control register (ICRD5) ==== */
    INT.ICRD.BIT.ICRD5 = 1;

    /* ==== Setting of the EtherC interrupt permission register (ECSIPR) ==== */
    MAC0.ECSIPR.LONG = 0x0000;    /* All disabled (change of LINK signals,
                                   Magic Packet detection,
                                   illegal carrier detection) */

    /* ==== Setting of the EtherC/E-DMAC status interrupt permission register (EESIPR) ==== */
    EDMAC0.EESIPR.LONG = 0x073c039f; /* Transmission/reception related sources are only enabled.
*/
//  EDMAC0.EESIPR.LONG = 0x04380300; /* Transmission related sources are only enabled. */
//  EDMAC0.EESIPR.LONG = 0x0304009f; /* Reception related sources are only enabled. */
                                     /* However, receive descriptor empty interrupt and receive
FIFO overflow bits are excluded */

    /* ==== Setting of the EtherC mode register (ECMR) ==== */
    if(link == FULL_TX || link == FULL_10M){
      MAC0.ECMR.BIT.DM =1;    /* Full-duplex transfer */
    }
    else {
      MAC0.ECMR.BIT.DM =0;    /* Half-duplex transfer */
    }

    if(link == FULL_TX || link == HALF_TX){
      EDMAC0.ECBRR.BIT.RTM =1;    /* 100Mbps */
    }
    else {
      EDMAC0.ECBRR.BIT.RTM =0;    /* 10Mbps */
    }

    MAC0.ECMR.BIT.RE = 1; /* Reception enabled */
    MAC0.ECMR.BIT.TE = 1; /* Transmission enabled */

    /* ==== Setting of the E-DMAC receive request register (EDRRR) ==== */
    EDMAC0.EDRRR.LONG = 0x00000001;  /* The E-DMAC is ready to receive. */
}

/*""FUNC COMMENT""***************************************************************
* Outline     : Initialization of Transmit/Receive Descriptors
*-------------------------------------------------------------------------------
* Declaration : static void lan_desc_create(void)
*-------------------------------------------------------------------------------
* Description : Transmit/receive descriptors and transmission/receiving
*               buffers are initialized.
*             : Descriptors are arranged in a ring structure.
*-------------------------------------------------------------------------------
* Argument        : None
*-------------------------------------------------------------------------------
* Return Value    : None
*-------------------------------------------------------------------------------
* Note            : The TACT and TDL bits are set to 0 for initialization.
*               These bits are set by the lan_send function.
*""FUNC COMMENT END""**********************************************************/
static void lan_desc_create(void)
{
    int i;
    EDMAC_SEND_DESC *psnd;
    EDMAC_RECV_DESC *prcv;

    /* ==== Clear of transmit/receive descriptors to 0 ==== */
    memset(&descriptor0,0x0,sizeof(descriptor0));

    /* ==== Initialization of transmit descriptors ==== */
    psnd = descriptor0.send_desc;
    for(i = 0; i<NUM_OF_TX_DESCRIPTOR ;i++){
      psnd->td2.TBA = &buffer0.send_buf[i][0];
      psnd->td0.BIT.TFP =0x3; /* 1 frame/1 descriptor */
```

```
          psnd->pNext = psnd +1;
          psnd++;
        }
        psnd--;
        psnd->td0.BIT.TDLE = 1;
        psnd->pNext = descriptor0.send_desc;

        /* ==== Initialization of receive descriptors ==== */
        prcv = descriptor0.recv_desc;
        for(i = 0; i<NUM_OF_RX_DESCRIPTOR ;i++){
          prcv->rd0.BIT.RACT =0x1; /* Restore the descriptor to the state where reception is possible
    */
          prcv->rd1.RBL = 0x05f0;
          prcv->rd2.RBA = &buffer0.recv_buf[i][0];
          prcv->pNext = prcv +1;
          prcv++;
        }
        prcv--;
        prcv->rd0.BIT.RDLE = 1;
        prcv->pNext = descriptor0.recv_desc;

        /* ==== Clear of transmission and receiving buffers to 0 ==== */
        memset(&buffer0,0x0,sizeof(buffer0));

        /* ==== Initialization of pointers to transmit & receive descriptors ==== */
        psenddesc0 = descriptor0.send_desc;
        precvdesc0 = descriptor0.recv_desc;

    }

    /*""FUNC COMMENT""***************************************************************
    * Outline      : Ethernet Frame Transmission Function
    *-------------------------------------------------------------------------------
    * Declaration      : int lan_send(unsigned char *addr, int flen)
    *-------------------------------------------------------------------------------
    * Description      : Data specified by the arguments are copied to the transmission buffer.
    *              : Transmit descriptors are set and transmission operation is initiated.
    *              : A flag indicating "transmission in progress" is checked.
    *                If the flag is OFF, transmission is judged to have been completed.
    *-------------------------------------------------------------------------------
    * Argument         : unsigned char *addr : I : Start address of the Ethernet frame for
    *                transmission
    *              : int flen : I : Frame size (number of bytes)
    *-------------------------------------------------------------------------------
    * Return value     : 0(SEND_OK)  : Success in transmission
    *                : -1(SEND_NG)  : Failure in transmission
    *-------------------------------------------------------------------------------
    * Note             :
    *""FUNC COMMENT END""**********************************************************/
    int lan_send(unsigned char *addr, int flen)
    {
        volatile int    w;
        volatile int  t1ms = 100; /* approx. 1-ms counter */
        volatile int  t400ms = 400;   /* approx. 400-ms counter */
        int              value;

        /* ==== Wait until the TACT bit of the transmit descriptor becomes 0 ==== */
        while(psenddesc0->td0.BIT.TACT == 1){
          /* wait */
        }

        /* ==== Turn ON the flag to indicate transmission in progress. ==== */
        tx_flag0 = TX_FLAG_ON;

        /* ==== Copy data for transmission indicated by arguments to the transmission buffer ==== */
        memcpy(psenddesc0->td2.TBA,addr,flen);

        /* ==== Setting of transmit descriptors==== */
        psenddesc0->td1.TDL = flen;
        psenddesc0->td0.BIT.TACT = 1;

        /* ==== Initiating transmission ==== */
        if(EDMAC0.EDTRR.BIT.TR == 0){
          EDMAC0.EDTRR.BIT.TR = 1;
        }

        /* ==== Confirmation of completion of data transmission  ==== */
        while(tx_flag0 == TX_FLAG_ON){   /* A flag indicating transmission in progress is turned ON
    */

          /* ==== approx. 10us wait ==== */
          for(w=0;w<16;w++){
              ;
          }

          /* ==== Workaround of Technical Update "TN-H8*-A429A/J" or "TN-H8*-A429A/E" ==== */
          if((--t1ms) <= 0){
              t1ms = 100; /*  */
              if(psenddesc0->td0.BIT.TACT == 0){
                  tx_flag0 = TX_FLAG_OFF;
                  break;
              }
              /* ==== Workaround of Technical Update "TN-H8*-A428A/J" or "TN-H8*-A428A/E" ==== */
              else if((--t400ms) <= 0){
                  return SEND_NG;
              }
              else{
                  /* DO NOTHING */
              }
          }
        }

        /* ==== Setting of transmit descriptors==== */
        psenddesc0 = psenddesc0->pNext;  /* Update to a pointer for descriptor management */

        return SEND_OK;
    }

    /*""FUNC COMMENT""***************************************************************
    * Outline         : Ethernet Frame Reception Function
    *-------------------------------------------------------------------------------
    * Declaration     : int lan_recv(unsigned char *addr)
    *-------------------------------------------------------------------------------
    * Description     : Ethernet frame of one frame is only received.
    *             : If there are no errors in the received frame, data are copied
    *               to the user buffer specified by an argument.
    *-------------------------------------------------------------------------------
    * Argument        : unsigned char *addr : 0 : Start address to which the received
    *               Ethernet frame is copied
    *-------------------------------------------------------------------------------
    * Return Value    : Number of bytes of the received frame : In case that receiving
    *               operation is succeeded
    *-------------------------------------------------------------------------------
    * Note            :
    *""FUNC COMMENT END""***********************************************************/
    int lan_recv(unsigned char *addr)
    {
        int ret;

        do {
          ret = lan_recv_ex(addr);
        } while (ret < 0);

        return ret;
    }


    /*""FUNC COMMENT""***************************************************************
    * Outline         : Ethernet Frame Reception Function
    *-----------------------------------------------------------------------------
    * Declaration     : int lan_recv_ex(unsigned char *addr)
    *-----------------------------------------------------------------------------
    * Description     : Ethernet frame of one frame is only received.
    *               : If there are no errors in the received frame, data are copied
    *                 to the user buffer specified by an argument.
    *-----------------------------------------------------------------------------
    * Argument        : unsigned char *addr : 0 : Start address to which the received
    *                 Ethernet frame is copied
    *-----------------------------------------------------------------------------
    * Return Value    : Number of bytes of the received frame : In case that receiving
    *                 operation is succeeded
    *               : -1 : No receive data
    *-----------------------------------------------------------------------------
    * Note            :
    *""FUNC COMMENT END""**********************************************************/
    int lan_recv_ex(unsigned char *addr)
    {
        int i;
        int dsize = 0;    /* Number of received data bytes */

        /* ==== Check whether receive data remains ==== */
        if(precvdesc0->rd0.BIT.RACT == 0x1)
        { /* No receive data */
           return -1;
        }
        else if(precvdesc0->rd0.BIT.RACT == 0x0)
        { /* Receive data remains */

          /* ==== Confirmation of received frame error ==== */
          if(precvdesc0->rd0.BIT.RFE == 0)
          { /* Case where no received frame errors occur */
              memcpy(addr,precvdesc0->rd2.RBA,precvdesc0->rd1.RDL);
              dsize = precvdesc0->rd1.RDL;
          }

          /* ==== Initialization of receive descriptors ==== */
          precvdesc0->rd0.LONG &= 0x40000000; /* Bits other than RDLE are cleared to 0 */
          precvdesc0->rd0.BIT.RACT =0x1; /* Restore the descriptor to the state where reception is
    possible */
          precvdesc0->rd1.RDL = 0x0000;
          precvdesc0 = precvdesc0->pNext;

          /* ==== Initiating data reception ==== */
          if(EDMAC0.EDRRR.BIT.RR == 0)
          {
              EDMAC0.EDRRR.BIT.RR = 1;
          }

          return dsize;
        }
    }

```

```
/*""FUNC COMMENT""*************************************************************
* Outline         : Interrupt Handling for Completion of E-DMAC Transmission (ch0)
*-----------------------------------------------------------------------------
* Declaration     : void INT_EDMAC_EINT0(void)
*-----------------------------------------------------------------------------
* Description     : Interrupt handling for completion of transmission and
*             : reception of frames When transmission of frames is completed,
*             : the flag to indicate transmission in progress is turned OFF.
*-----------------------------------------------------------------------------
* Argument        : None
*-----------------------------------------------------------------------------
* Return Value    : None
*-----------------------------------------------------------------------------
* Note            :
*""FUNC COMMENT END""*********************************************************/
#pragma section IntPRG
__interrupt(vect=119) void INT_EDMAC_EINT0(void)
{
    unsigned long status;

    /* ==== Reading of interrupt status ==== */
    status = EDMAC0.EESR.LONG & EDMAC0.EESIPR.LONG;

    /* ==== Clear of interrupt sources ==== */
    EDMAC0.EESR.LONG = status;    /* Clear of 1 write */

    /* ==== At the time frame transmission has been completed ==== */
    if(status & FRAME_TRANSMIT_COMPLETE){
      tx_flag0 = TX_FLAG_OFF;
    }
}
#pragma section
/* End of File */
```

3.3 Sample Program Listing: "ether.h"

```
/******************************************************************************
* DISCLAIMER
* Please refer to http://www.renesas.com/disclaimer
******************************************************************************
  Copyright (C) 2010 Renesas Electronics Corporation. All rights reserved.
******************************************************************************
* File Name    : ether.h
* Version      : 1.01
* Device(s)    : H8S/2472
* Tool-Chain   : HEW, H8S,H8/300 Standard Toolchain (V.6.2.2.0)
* OS           : None
* H/W Platform : R0K402472D000BR,R0K402472D001BR,R0K402472D002BR
* Description  : This is a sample program for setting the transmit/receive
*                descriptors and transmission/receiving buffers.
* Limitations  : None
******************************************************************************
* History : DD.MM.YYYY Version Description
*         : 19.02.2007 1.00    First Release
*         : 19.04.2010 1.01    Unnecessary code deleted
******************************************************************************/

#ifndef ETHER_H
#define ETHER_H

/******************************************************************************
Includes   <System Includes> , "Project Includes"
******************************************************************************/

/******************************************************************************
Macro definitions
******************************************************************************/
#define NUM_OF_TX_DESCRIPTOR     4     /* Number of descriptors for data transmission */
#define NUM_OF_RX_DESCRIPTOR     4     /* Number of descriptors for data reception */
#define  SIZE_OF_TX_BUFFER         1520    /* Transmission buffer size must be an integer
multiple of 16 bytes. */
#define  SIZE_OF_RX_BUFFER         1520     /* Receiving buffer size must be integer multiple
of 16 bytes. */
#define  MAC_ADDRESS_HIGH0  0x001125bc     /* In case that the MAC address is 00-11-25-bc-fd-
0a (hexadecimal) */
#define  MAC_ADDRESS_LOW0   0x0000fd0a
#define  TX_FLAG_ON  1
#define  TX_FLAG_OFF 0
#define FRAME_TRANSMIT_COMPLETE 0x00200000

/******************************************************************************
Typedef definitions
******************************************************************************/
/* ==== Definition of enumeration constant of return value for lan_open() ==== */
typedef enum{OPEN_OK= 0,OPEN_NG= -1}OPEN_STATUS;

/* ==== Definition of enumeration constant of return value for lan_send() ==== */
typedef enum{SEND_OK = 0,SEND_NG =-1}SEND_STATUS;

/* ==== Definition of enumeration constant of return value for lan_close() ==== */
typedef enum{CLOSE_OK= 0,CLOSE_NG= -1}CLOSE_STATUS;

/* ==== Definition of structure of transmit descriptors ==== */
typedef union
{
    unsigned long LONG;
    struct{
      unsigned int   TACT:1;        /* Transmit descriptor active bit */
      unsigned int   TDLE:1;        /* Transmit descriptor list end */
      unsigned int   TFP:2;         /* Transmit frame position */
      unsigned int   TFE:1;         /* Occurrence of the transmit frame error (refer to TFSx
for error source.) */
      unsigned int   reserved1:11;  /* Not in use */
      unsigned int   reserved2:7;   /* Not in use */
      unsigned int   TFS8:1;        /* Transmit abort */
      unsigned int   reserved3:4;   /* Not in use */
      unsigned int   TFS3:1;        /* Carrier not detected at the time of initiating data
transmission */
      unsigned int   TFS2:1;        /* Detect loss of carrier during transmission */
      unsigned int   TFS1:1;        /* Delayed collision detect */
      unsigned int   TFS0:1;        /* Transmit retry over */
    }BIT;
}TD0;
typedef struct
{
    unsigned short   TDL;           /* Size of transmission buffer (number of bytes) */
    unsigned short   reserved;
}TD1;
typedef struct
{
    unsigned char    *TBA;          /* Address of transmission buffer */
}TD2;

typedef struct tag_edmac_send_desc
{
    TD0    td0;
    TD1    td1;
    TD2    td2;
    struct   tag_edmac_send_desc    *pNext;
}EDMAC_SEND_DESC;

/* ==== Definition of structure of receive descriptors ==== */
typedef union
{
    unsigned long LONG;
    struct{
      unsigned int   RACT:1;        /* Receive descriptor active */
      unsigned int   RDLE:1;        /* Receive descriptor list end */
      unsigned int   RFP:2;         /* Received frame position */
      unsigned int   RFE:1;         /* Occurrence of received frame error (refer to TFSx for
error source.) */
      unsigned int   reserved1:3;   /* Not in use */
      unsigned int   reserved2:8;   /* Not in use */
      unsigned int   reserved3:6;   /* Not in use */
      unsigned int   RFS9:1;        /* Receive FIFO overflow */
      unsigned int   RFS8:1;        /* Abort detection during data reception */
      unsigned int   RFS7:1;        /* Multicast address frame received */
      unsigned int   reserved4:2;   /* Not in use */
      unsigned int   RFS4:1;        /* Fraction of bits for frame received error */
      unsigned int   RFS3:1;        /* Receive too-long frame error */
      unsigned int   RFS2:1;        /* Receive too-short frame error */
      unsigned int   RFS1:1;        /* PHY-LSI receive error */
      unsigned int   RFS0:1;        /* CRC error in received frame */
    }BIT;
}RD0;
typedef struct
{
    unsigned short   RBL;           /* Data length of receiving buffer (unit: bytes, specified
for 16-byte boundaries) */
    unsigned short   RDL;           /* Length of received data (this is set when the last
frame is received.) */
}RD1;
```

```
typedef struct
{
    unsigned char    *RBA;           /* Start address of receiving buffer, 16-byte boundary in
case of SDRAM */
}RD2;

typedef struct tag_edmac_recv_desc
{
    RD0    rd0;
    RD1    rd1;
    RD2    rd2;
    struct   tag_edmac_recv_desc  *pNext;
}EDMAC_RECV_DESC;

/* ==== Definition of structure of transmission and receiving buffers ==== */
typedef struct
{
    /* Area of transmission buffers (this must be aligned with a 16-byte boundary.) */
    unsigned char    send_buf[NUM_OF_TX_DESCRIPTOR][SIZE_OF_TX_BUFFER];

    /* Area of receiving buffers (this must be aligned with a 16-byte boundary.) */
    unsigned char    recv_buf[NUM_OF_RX_DESCRIPTOR][SIZE_OF_RX_BUFFER];
}TXRX_BUFFER_SET;

/* ==== Definition of structure of transmit and receive descriptors ==== */
typedef struct
{
    /* Transmit descriptor (it must be aligned on 16-byte boundary.) */
    EDMAC_SEND_DESC    send_desc[NUM_OF_TX_DESCRIPTOR];

    /* Receive descriptor (it must be aligned on 16-byte boundary.) */
    EDMAC_RECV_DESC    recv_desc[NUM_OF_RX_DESCRIPTOR];
}TXRX_DESCRIPTOR_SET;

/******************************************************************************
Variable Externs
******************************************************************************/

/******************************************************************************
Functions Prototypes
******************************************************************************/
int lan_open(void);
int lan_recv(unsigned char *addr);
int lan_recv_ex(unsigned char *addr);
int lan_send(unsigned char *addr, int flen);
int lan_close(void);





#endif /* ETHER_H */
```

3.4 Sample Program Listing: "phy.c"

```
/*******************************************************************************
* DISCLAIMER

* This software is supplied by Renesas Electronics Corporation and is only
* intended for use with Renesas products. No other uses are authorized.

* This software is owned by Renesas Electronics Corporation and is protected under
* all applicable laws, including copyright laws.

* THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES
* REGARDING THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY,
* INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NON-INFRINGEMENT. ALL SUCH WARRANTIES ARE EXPRESSLY
* DISCLAIMED.

* TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
* ELECTRONICS CORPORATION NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
* FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
* FOR ANY REASON RELATED TO THIS SOFTWARE, EVEN IF RENESAS OR ITS
* AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

* Renesas reserves the right, without notice, to make changes to this
* software and to discontinue the availability of this software.
* By using this software, you agree to the additional terms and
* conditions found by accessing the following link:
* http://www.renesas.com/disclaimer
********************************************************************************
* Copyright (C) 2010 Renesas Electronics Corporation. All rights reserved.
********************************************************************************
* File Name    : phy.c
* Version      : 1.02
* Device(s)    : H8S/2472
* Tool-Chain   : HEW, H8S,H8/300 Standard Toolchain (V.6.2.2.0)
* OS           : None
* H/W Platform : R0K402472D000BR,R0K402472D001BR,R0K402472D002BR
* Description  : Initialization Example of the PHY-LSI Automatic Negotiation Function
* Limitations  : None
*******************************************************************************
* History : DD.MM.YYYY Version Description
*         : 01.01.2009 1.00    First Release
*         : 19.04.2010 1.01    Unnecessary code deleted
*         : 29.07.2010 1.02    Modification of wait counter
*******************************************************************************/


/*******************************************************************************
Includes   <System Includes> , "Project Includes"
*******************************************************************************/
#include "iodefine2472.h"
#include "phy.h"

/*******************************************************************************
Typedef definitions
*******************************************************************************/

/*******************************************************************************
Macro definitions
*******************************************************************************/
#define   NO_WAIT
//#undef  NO_WAIT

//#define  PHY_LOOP_BACK
```

```

/* The MII register of the PHY-LSI */
#define    PHY_ADD                         0x03c0
#define    BASIC_MODE_CONTROL_REG          PHY_ADD+(0x0000)
#define    BASIC_MODE_STATUS_REG       PHY_ADD+(0x0001)
#define    PHY_IDENTIFIER1_REG             PHY_ADD+(0x0002)
#define    PHY_IDENTIFIER2_REG             PHY_ADD+(0x0003)
#define    AN_ADVERTISEMENT_REG        PHY_ADD+(0x0004)
#define    AN_LINK_PARTNER_ABILITY_REG     PHY_ADD+(0x0005)
#define    AN_EXPANSION_REG            PHY_ADD+(0x0006)
/* For accessing the MII */
#define PHY_ST                      1
#define PHY_WRITE                   1
#define PHY_READ                    2
#define PHY_ADDR0                   1

/******************************************************************************
Imported global variables and functions (from other files)
******************************************************************************/

/******************************************************************************
Exported global variables and functions (to be accessed by other files)
******************************************************************************/

/******************************************************************************
Private global variables and functions
******************************************************************************/
static unsigned short  phy_reg_read(unsigned short reg_addr);
static void    phy_reg_write(unsigned short reg_addr, unsigned short data);
static void    mii_preamble(void);
static void    mii_cmd(unsigned short reg_addr, int option);
static void    mii_reg_read(unsigned short *data);
static void    mii_reg_write(unsigned short data);
static void    mii_z(void);
static void    mii_ta_10(void);
static void    mii_write_1(void);
static void    mii_write_0(void);

/*""FUNC COMMENT""*************************************************************
* Outline      : Detection of Negotiation Result of the PHY Link
*-----------------------------------------------------------------------------
* Declaration  : int phy_autonego(void);
*-----------------------------------------------------------------------------
* Description  : Result of automatic negotiation is read and returned as a return value.
*              : This function waits up to 5 seconds for automatic negotiation to end.
*-----------------------------------------------------------------------------
* Argument         : None
*-----------------------------------------------------------------------------
* Return Value  : 4(FULL_TX)   :100 Mbps, full-duplex transfer
*               : 3(HALF_TX)   :100 Mbps, half-duplex transfer
*               : 2(FULL_10M)  :10 Mbps, full-duplex transfer
*               : 1(HALF_10M)  :10 Mbps, half-duplex transfer
*               : 0(NEGO_FAIL) :Negotiation failed
*-----------------------------------------------------------------------------
* Notice       :
*""FUNC COMMENT END""*********************************************************/
int phy_autonego(void)
{
    unsigned short data0,data1;
    int link = NEGO_FAIL;
    volatile int t;
    int i;

    data0 = phy_reg_read(BASIC_MODE_CONTROL_REG);
    data0 = data0 | 0x8000;
    phy_reg_write(BASIC_MODE_CONTROL_REG, data0);

    /* ---- Wait ---- */
    for(i=0;i<1000;i++){
      t=0x27C0;
      while( --t){                  /* approx. 2.9-ms wait counting@32 MHz */
          ;
      }
    }

#ifdef PHY_LOOP_BACK
    data0 = phy_reg_read(BASIC_MODE_CONTROL_REG);
    data0 = data0 & ~0x1000;
    phy_reg_write(BASIC_MODE_CONTROL_REG, data0);
    /* ---- approx. 10-ms wait ---- */
    t=36000;
    while( --t){                    /* approx. 10-ms wait counting@32 MHz */
      ;
    }
    data0 = phy_reg_read(BASIC_MODE_CONTROL_REG);
    data0 = data0 | 0x4000;
    phy_reg_write(BASIC_MODE_CONTROL_REG, data0);
    /* ---- approx. 10-ms wait ---- */
    t=36000;
    while( --t){                    /* approx. 10-ms wait counting@32 MHz */
      ;
    }
    data0 = phy_reg_read(BASIC_MODE_CONTROL_REG);
    data0 = data0 | 0x0100;
    phy_reg_write(BASIC_MODE_CONTROL_REG, data0);
    /* ---- approx. 10-ms wait ---- */
    t=36000;
    while( --t){                    /* approx. 10-ms wait counting@32 MHz */
      ;
    }
    data0 = phy_reg_read(BASIC_MODE_CONTROL_REG);
    return FULL_TX;
//  return FULL_10M;
#endif

    /* ==== Wait loop for end of automatic negotiation ==== */
    for( i=0; i<500; i++){

      /* ---- approx. 10-ms wait ---- */
      t=36000;
      while( --t){                  /* approx. 10-ms wait counting@32 MHz */
          ;
      }
      data0 = phy_reg_read(BASIC_MODE_STATUS_REG);
      if(data0 & 0x0020){   /* Automatic negotiation is completed */

          /* ---- Result of automatic negotiation is obtained. ---- */
          data1 = phy_reg_read(AN_LINK_PARTNER_ABILITY_REG);
                                /* Result of automatic negotiation AN_LINK_PARTNER_ABILITY_REG *
                                 * If the device at the other end does not support automatic negotiation,
                                 parallel detection is performed, but the result is indicated in this
                                 register.                  */
          /* ---- Judgment of result -> break at the end of negotiation. ---- */
          if( data1&0x0100 ){
              link = FULL_TX;
          }
          else if( data1&0x0080 ){
              link = HALF_TX;
          }
          else if( data1&0x0040 ){
              link = FULL_10M;
          }
          else if( data1&0x0020 ){
              link = HALF_10M;
          }
          else{
              link = NEGO_FAIL;
          }
          break;
      }
    }
    return link;
}


/*""FUNC COMMENT""************************************************************
* Outline      : Reading of All MII Registers in the PHY-LSI
*-----------------------------------------------------------------------------
* Declaration  : static unsigned short phy_reg_read(unsigned short reg_addr)
*-----------------------------------------------------------------------------
* Description  : Values of all MII registers in the PHY-LSI are obtained.
*-----------------------------------------------------------------------------
* Argument         : unsigned short reg_addr : I : Address of the PHY-LSI register
*                from which a value is read
*-----------------------------------------------------------------------------
* Return Value     : Obtained register values
*-----------------------------------------------------------------------------
* Notice       :
*""FUNC COMMENT END""********************************************************/
static unsigned short  phy_reg_read(unsigned short reg_addr)
{
    unsigned short data;

    mii_preamble();
    mii_cmd(reg_addr, PHY_READ);
    mii_z();
    mii_reg_read(&data);
    mii_z();

    return data;
}

/*""FUNC COMMENT""************************************************************
* Outline      : Writing of All MII Registers in the PHY-LSI
*-----------------------------------------------------------------------------
* Declaration  : static void phy_reg_write(unsigned short reg_addr, unsigned short data )
*-----------------------------------------------------------------------------
* Description  : Values are set in all MII registers in the PHY-LSI.
*-----------------------------------------------------------------------------
* Argument         : unsigned short reg_addr : I : The PHY-LSI register address
*                to which values are written
*              : unsigned short data : I : Values set in registers of the PHY-LSI
*-----------------------------------------------------------------------------
* Return Value     : None
*-----------------------------------------------------------------------------
* Notice       :
*""FUNC COMMENT END""********************************************************/
static void    phy_reg_write(unsigned short reg_addr, unsigned short data)
{
    mii_preamble();
    mii_cmd(reg_addr, PHY_WRITE);
    mii_ta_10();
    mii_reg_write(data);
    mii_z();

}

/*""FUNC COMMENT""**************************************************************
* Outline      : Preparation for Accessing All MII Registers in the PHY-LSI
*-------------------------------------------------------------------------------
* Declaration  : static void mii_preamble(void)
*-------------------------------------------------------------------------------
* Description      : As advance preparation for access to PHY-LSI registers,
*              : one of 32 bits is output to the MII block.
*-------------------------------------------------------------------------------
* Argument         : None
*-------------------------------------------------------------------------------
* Return Value     : None
*-------------------------------------------------------------------------------
* Notice       :
*""FUNC COMMENT END""**********************************************************/
static void    mii_preamble(void)
{
    short i;

    i = 32;
    while( i > 0 ) {
      /* 1 is output to the MII (Media Independent Interface) block. */
      mii_write_1();
      i--;
    }
}

/*""FUNC COMMENT""**************************************************************
* Outline      : Setting Modes of All MII Registers in the PHY-LSI
*-------------------------------------------------------------------------------
* Declaration  : static void mii_cmd(unsigned short reg_addr, int option)
*-------------------------------------------------------------------------------
* Description  : R/W mode of all MII registers in the PHY-LSI is set.
*-------------------------------------------------------------------------------
* Argument         : unsigned short reg_addr : I : Register address of the PHY-LSI
*              : int option : I : Specification of R/W mode
*-------------------------------------------------------------------------------
* Return Value     : None
*-------------------------------------------------------------------------------
* Notice       :
*""FUNC COMMENT END""**********************************************************/
static void    mii_cmd(unsigned short reg_addr, int option)
{
    int   i;
    unsigned short data;

    data = 0;
    data = (PHY_ST << 14);             /* ST code       */
    if( option == PHY_READ ) {
      data |= (PHY_READ << 12);     /* OP code(RD)   */
    }
    else {
      data |= (PHY_WRITE << 12);        /* OP code(WT)   */
    }

    data |= (PHY_ADDR0 << 7);       /* PHY Address   */
    data |= (reg_addr << 2);            /* Reg Address   */

    for(i=14; i>0; i--){
      if( (data & 0x8000) == 0 ) {
            mii_write_0();
        }
        else {
            mii_write_1();
        }
        data <<= 1;
    }
}

/*""FUNC COMMENT""*************************************************************
* Outline       : Obtaining Value in All MII Registers in the PHY-LSI
*-----------------------------------------------------------------------------
* Declaration  : static void mii_reg_read (unsigned short *data)
*-----------------------------------------------------------------------------
* Description  : Acquires the values of all MII registers in the PHY-LSI,
*                one bit at a time.
*-----------------------------------------------------------------------------
* Argument          : unsigned short *data : O : Destination address for storage
*                of the acquired value
*-----------------------------------------------------------------------------
* Return Value      : None
*-----------------------------------------------------------------------------
* Notice        :
*""FUNC COMMENT END""*********************************************************/
static void    mii_reg_read(unsigned short *data)
{
    int   i;
    unsigned short reg_data;

#ifdef NO_WAIT
    /* One-bit-unit data is read. */
    reg_data = 0;
    for(i=16; i>0; i--){
      MAC0.PIR.LONG = 0x00000000;
      MAC0.PIR.LONG = 0x00000001;
      MAC0.PIR.LONG = 0x00000001;
      reg_data <<= 1;
      reg_data |= (MAC0.PIR.LONG & 0x00000008) >> 3;/* MDI read  */
      MAC0.PIR.LONG = 0x00000001;
      MAC0.PIR.LONG = 0x00000000;
    }
    *data = reg_data;
#else
    /* Data are read one bit at a time. */
    int   j;
    reg_data = 0;
    for(i=16; i>0; i--){
      for(j=0; j<3; j++)   MAC0.PIR.LONG = 0x00000000;
      for(j=0; j<3; j++)   MAC0.PIR.LONG = 0x00000001;
      for(j=0; j<3; j++)   MAC0.PIR.LONG = 0x00000001;
      reg_data <<= 1;
      reg_data |= (MAC0.PIR.LONG & 0x00000008) >> 3;/* MDI read  */
      for(j=0; j<3; j++)   MAC0.PIR.LONG = 0x00000001;
      for(j=0; j<3; j++)   MAC0.PIR.LONG = 0x00000000;
    }
    *data = reg_data;
#endif
}

/*""FUNC COMMENT""*************************************************************
* Outline       : Setting Value in All MII Registers in the PHY -LSI
*-----------------------------------------------------------------------------
* Declaration  : static void mii_reg_write (unsigned short data)
*-----------------------------------------------------------------------------
```

```
* Description  : One-bit-unit data is set for value in all MII registers in the PHY-LSI.
*--------------------------------------------------------------------------------
* Argument         : unsigned short data : I : Value set in a register
*--------------------------------------------------------------------------------
* Return Value     : None
*--------------------------------------------------------------------------------
* Notice       :
*""FUNC COMMENT END""*********************************************************/
static void    mii_reg_write(unsigned short data)
{
    int   i;

    /* One-bit-unit data is written. */
    for(i=16; i>0; i--){
      if( (data & 0x8000) == 0 ) {
          mii_write_0();
      }
      else {
          mii_write_1();
      }
      data <<= 1;
    }
}

/*""FUNC COMMENT""**************************************************************
* Outline      : Release of the MII Bus
*--------------------------------------------------------------------------------
* Declaration  : static void mii_z(void)
*--------------------------------------------------------------------------------
* Description  : Settings to release the bus from access to the MII.
*--------------------------------------------------------------------------------
* Argument         : None
*--------------------------------------------------------------------------------
* Return Value     : None
*--------------------------------------------------------------------------------
* Notice       :
*""FUNC COMMENT END""*********************************************************/
static void    mii_z(void)
{
#ifdef NO_WAIT
    MAC0.PIR.LONG = 0x00000000;
    MAC0.PIR.LONG = 0x00000001;
    MAC0.PIR.LONG = 0x00000001;
    MAC0.PIR.LONG = 0x00000001;
    MAC0.PIR.LONG = 0x00000000;
#else
    int   j;
    for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000000;
    for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000001;
    for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000001;
    for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000001;
    for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000000;
#endif
}

/*""FUNC COMMENT""**************************************************************
* Outline      : Output of the MII TA (1 or 0) Bit
*--------------------------------------------------------------------------------
* Declaration  : static void mii_ta_10(void)
*--------------------------------------------------------------------------------
* Description      : 1 or 0 is output to the MII.
*--------------------------------------------------------------------------------
* Argument         : None
*--------------------------------------------------------------------------------
```

```
* Return Value     : None
*-----------------------------------------------------------------------------
* Notice       :
*""FUNC COMMENT END""*********************************************************/
static void    mii_ta_10(void)
{
    mii_write_1();
    mii_write_0();
}

/*""FUNC COMMENT""*************************************************************
* Outline      : Output of One Bit (1) to the MII
*-----------------------------------------------------------------------------
* Declaration  : static void mii_write_1(void)
*-----------------------------------------------------------------------------
* Description  : 1 is output to the MII.
*-----------------------------------------------------------------------------
* Argument         : None
*-----------------------------------------------------------------------------
* Return Value     : None
*-----------------------------------------------------------------------------
* Notice       :
*""FUNC COMMENT END""*********************************************************/
static void    mii_write_1(void)
{
#ifdef NO_WAIT
    MAC0.PIR.LONG = 0x00000006;
    MAC0.PIR.LONG = 0x00000007;
    MAC0.PIR.LONG = 0x00000007;
    MAC0.PIR.LONG = 0x00000007;
    MAC0.PIR.LONG = 0x00000006;
#else
    int   j;
    for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000006;
    for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000007;
    for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000007;
    for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000007;
    for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000006;
#endif
}

/*""FUNC COMMENT""*************************************************************
* Outline      : Output of One Bit (0) to the MII
*-----------------------------------------------------------------------------
* Declaration  : static void mii_write_0(void)
*-----------------------------------------------------------------------------
* Description  : 0 is output to the MII.
*-----------------------------------------------------------------------------
* Argument         : None
*-----------------------------------------------------------------------------
* Return Value     : None
*-----------------------------------------------------------------------------
* Notice       :
*""FUNC COMMENT END""*********************************************************/
static void    mii_write_0(void)
{
#ifdef NO_WAIT
    MAC0.PIR.LONG = 0x00000002;
    MAC0.PIR.LONG = 0x00000003;
    MAC0.PIR.LONG = 0x00000003;
    MAC0.PIR.LONG = 0x00000003;
    MAC0.PIR.LONG = 0x00000002;
#else
    int   j;
```

```
        for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000002;
        for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000003;
        for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000003;
        for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000003;
        for(j=0; j<3; j++) MAC0.PIR.LONG = 0x00000002;
#endif
}

/* End of File */
```

3.5 Sample Program Listing: "phy.h"

```
/*******************************************************************************
* DISCLAIMER
* Please refer to http://www.renesas.com/disclaimer
******************************************************************************
  Copyright (C) 2010 Renesas Electronics Corporation. All rights reserved.
*******************************************************************************
* File Name    : phy.h
* Version      : 1.00
* Device(s)    : H8S/2472
* Tool-Chain   : HEW, H8S,H8/300 Standard Toolchain (V.6.2.2.0)
* OS           : None
* H/W Platform : R0K402472D000BR,R0K402472D001BR,R0K402472D002BR
* Description  : Header program of the PHY
* Limitations  : None
******************************************************************************
* History : DD.MM.YYYY Version Description
*         : 19.02.2007 1.00    First Release
*******************************************************************************/

#ifndef PHY_H
#define PHY_H

/*******************************************************************************
Includes   <System Includes> , "Project Includes"
*******************************************************************************/

/*******************************************************************************
Typedef definitions
*******************************************************************************/

/*******************************************************************************
Macro definitions
*******************************************************************************/
/* Link result */
#define NEGO_FAIL           0
#define HALF_10M        1
#define FULL_10M        2
#define HALF_TX             3
#define FULL_TX             4

/*******************************************************************************
Variable Externs
*******************************************************************************/

/*******************************************************************************
Functions Prototypes
*******************************************************************************/
int phy_autonego(void);




#endif /* PHY_H */
```

3.6 Sample Program Listing: "iodefine2472.h"

```
/*******************************************************************************
* DISCLAIMER
* Please refer to http://www.renesas.com/disclaimer
******************************************************************************
  Copyright (C) 2010 Renesas Electronics Corporation. All rights reserved.
*******************************************************************************
* File Name    : iodefine2472.h
* Version      : 1.01
* Device(s)    : H8S/2472
* Tool-Chain   : HEW, H8S,H8/300 Standard Toolchain (V.6.2.2.0)
* OS           : None
* H/W Platform : R0K402472D000BR,R0K402472D001BR,R0K402472D002BR
* Description  : register definition
* Limitations  : None
******************************************************************************
* History : DD.MM.YYYY Version Description
*         : 18.10.2007 1.00    First Release
*         : 19.04.2010 1.01    Unnecessary code deleted
******************************************************************************/

#ifndef IODEFINE2472_H
#define IODEFINE2472_H

/*******************************************************************************
Includes   <System Includes> , "Project Includes"
******************************************************************************/

/*******************************************************************************
Typedef definitions
******************************************************************************/
struct st_ether {                                       /* struct ETHER */
       union {                                          /* ECMR         */
             unsigned long LONG;                        /*  Long Access */
             struct {                                   /*  Word Access */
                    unsigned short H;                   /*   High       */
                    unsigned short L;                   /*   Low        */
                    } WORD;                             /*              */
             struct {                                   /*  Bit Access  */
                    unsigned long :12;                  /*              */
                    unsigned long ZPF:1;                /*   ZPF        */
                    unsigned long PFR:1;                /*   PFR        */
                    unsigned long RXF:1;                /*   RXF        */
                    unsigned long TXF:1;                /*   TXF        */
                    unsigned long :3;                   /*              */
                    unsigned long PRCEF:1;              /*   PRCEF      */
                    unsigned long :2;                   /*              */
                    unsigned long MPDE:1;               /*   MPDE       */
                    unsigned long :2;                   /*              */
                    unsigned long RE:1;                 /*   RE         */
                    unsigned long TE:1;                 /*   TE         */
                    unsigned long :1;                   /*              */
                    unsigned long ILB:1;                /*   ILB        */
                    unsigned long ELB:1;                /*   ELB        */
                    unsigned long DM:1;                 /*   DM         */
                    unsigned long PRM:1;                /*   PRM        */
                    } BIT;                              /*              */
             } ECMR;                                    /*              */
       union {                                          /* ECSR         */
             unsigned long LONG;                        /*  Long Access */
             struct {                                   /*  Word Access */
                    unsigned short H;                   /*   High       */
                    unsigned short L;                   /*   Low        */
```

```
                  } WORD;                            /*               */
           struct {                                  /*   Bit Access  */
                  unsigned long :27;                 /*               */
                  unsigned long PSRTO:1;             /*    PSRTO      */
                  unsigned long :1;                  /*               */
                  unsigned long LCHNG:1;             /*    LCHNG      */
                  unsigned long MPD:1;               /*    MPD        */
                  unsigned long ICD:1;               /*    ICD        */
                  } BIT;                             /*               */
           } ECSR;                                   /*               */
     union {                                         /* ECSIPR        */
           unsigned long LONG;                       /*   Long Access */
           struct {                                  /*   Word Access */
                  unsigned short H;                  /*    High       */
                  unsigned short L;                  /*    Low        */
                  } WORD;                            /*               */
           struct {                                  /*   Bit Access  */
                  unsigned long :27;                 /*               */
                  unsigned long PSRTOIP:1;           /*    PSRTOIP    */
                  unsigned long :1;                  /*               */
                  unsigned long LCHNGIP:1;           /*    LCHNGIP    */
                  unsigned long MPDIP:1;             /*    MPDIP      */
                  unsigned long ICDIP:1;             /*    ICDIP      */
                  } BIT;                             /*               */
           } ECSIPR;                                 /*               */
     union {                                         /* PIR           */
           unsigned long LONG;                       /*   Long Access */
           struct {                                  /*   Word Access */
                  unsigned short H;                  /*    High       */
                  unsigned short L;                  /*    Low        */
                  } WORD;                            /*               */
           struct {                                  /*   Bit Access  */
                  unsigned long :28;                 /*               */
                  unsigned long MDI:1;               /*    MDI        */
                  unsigned long MDO:1;               /*    MDO        */
                  unsigned long MMD:1;               /*    MMD        */
                  unsigned long MDC:1;               /*    MDC        */
                  } BIT;                             /*               */
           } PIR;                                    /*               */
     union {                                         /* MAHR          */
           unsigned long LONG;                       /*   Long Access */
           struct {                                  /*   Word Access */
                  unsigned short H;                  /*    High       */
                  unsigned short L;                  /*    Low        */
                  } WORD;                            /*               */
           struct {                                  /*   Bit Access  */
                  unsigned long MA47:1;              /*    MA47       */
                  unsigned long MA46:1;              /*    MA46       */
                  unsigned long MA45:1;              /*    MA45       */
                  unsigned long MA44:1;              /*    MA44       */
                  unsigned long MA43:1;              /*    MA43       */
                  unsigned long MA42:1;              /*    MA42       */
                  unsigned long MA41:1;              /*    MA41       */
                  unsigned long MA40:1;              /*    MA40       */
                  unsigned long MA39:1;              /*    MA39       */
                  unsigned long MA38:1;              /*    MA38       */
                  unsigned long MA37:1;              /*    MA37       */
                  unsigned long MA36:1;              /*    MA36       */
                  unsigned long MA35:1;              /*    MA35       */
                  unsigned long MA34:1;              /*    MA34       */
                  unsigned long MA33:1;              /*    MA33       */
                  unsigned long MA32:1;              /*    MA32       */
                  unsigned long MA31:1;              /*    MA31       */
                  unsigned long MA30:1;              /*    MA30       */
```

```
                        unsigned long MA29:1;          /*   MA29       */
                        unsigned long MA28:1;          /*   MA28       */
                        unsigned long MA27:1;          /*   MA27       */
                        unsigned long MA26:1;          /*   MA26       */
                        unsigned long MA25:1;          /*   MA25       */
                        unsigned long MA24:1;          /*   MA24       */
                        unsigned long MA23:1;          /*   MA23       */
                        unsigned long MA22:1;          /*   MA22       */
                        unsigned long MA21:1;          /*   MA21       */
                        unsigned long MA20:1;          /*   MA20       */
                        unsigned long MA19:1;          /*   MA19       */
                        unsigned long MA18:1;          /*   MA18       */
                        unsigned long MA17:1;          /*   MA17       */
                        unsigned long MA16:1;          /*   MA16       */
                        } BIT;                         /*              */
                } MAHR;                                /*              */
        union {                                        /* MALR         */
                unsigned long LONG;                    /*  Long Access */
                struct {                               /*  Bit Access  */
                        unsigned long :16;             /*              */
                        unsigned long MA:16;           /*   MA         */
                        } BIT;                         /*              */
                } MALR;                                /*              */
        union {                                        /* RFLR         */
                unsigned long LONG;                    /*  Long Access */
                struct {                               /*  Word Access */
                        unsigned short H;              /*   High       */
                        unsigned short L;              /*   Low        */
                        } WORD;                        /*              */
                struct {                               /*  Bit Access  */
                        unsigned long :20;             /*              */
                        unsigned long RFL:12;          /*   RFL        */
                        } BIT;                         /*              */
                } RFLR;                                /*              */
        union {                                        /* PSR          */
                unsigned long LONG;                    /*  Long Access */
                struct {                               /*  Word Access */
                        unsigned short H;              /*   High       */
                        unsigned short L;              /*   Low        */
                        } WORD;                        /*              */
                struct {                               /*  Bit Access  */
                        unsigned long :31;             /*              */
                        unsigned long LMON:1;          /*   LMON       */
                        } BIT;                         /*              */
                } PSR;                                 /*              */
        unsigned long TROCR;                           /* TROCR        */
        unsigned long CDCR;                            /* CDCR         */
        unsigned long LCCR;                            /* LCCR         */
        unsigned long CNDCR;                           /* CNDCR        */
        unsigned char wk0[4];                          /*              */
        unsigned long CEFCR;                           /* CEFCR        */
        unsigned long FRECR;                           /* FRECR        */
        unsigned long TSFRCR;                          /* TSFRCR       */
        unsigned long TLFRCR;                          /* TLFRCR       */
        unsigned long RFCR;                            /* RFCR         */
        unsigned long MAFCR;                           /* MAFCR        */
        unsigned char wk1[8];                          /*              */
        union {                                        /* IPGR         */
                unsigned long LONG;                    /*  Long Access */
                struct {                               /*  Word Access */
                        unsigned short H;              /*   High       */
                        unsigned short L;              /*   Low        */
                        } WORD;                        /*              */
                struct {                               /*  Bit Access  */
```

```
                        unsigned long :27;                /*                */
                        unsigned long IPG:5;              /*   IPG          */
                        } BIT;                            /*                */
                } IPGR;                                   /*                */
        union {                                           /* APR            */
                unsigned long LONG;                       /*  Long Access */
                struct {                                  /*  Bit Access  */
                        unsigned long :16;                /*                */
                        unsigned long AP:16;              /*   AP           */
                        } BIT;                            /*                */
                } APR;                                    /*                */
        union {                                           /* MPR            */
                unsigned long LONG;                       /*  Long Access */
                struct {                                  /*  Bit Access  */
                        unsigned long :16;                /*                */
                        unsigned long MP:16;              /*   MP           */
                        } BIT;                            /*                */
                } MPR;                                    /*                */
        unsigned char wk2[4];                             /*                */
        union {                                           /* TPAUSER        */
                unsigned long LONG;                       /*  Long Access */
                struct {                                  /*  Bit Access  */
                        unsigned long :16;                /*                */
                        unsigned long TPAUSE:16;          /*   TPAUSE       */
                        } BIT;                            /*                */
                } TPAUSER;                                /*                */
};                                                        /*                */
struct st_edmac {                                         /* struct EDMAC */
        union {                                           /* EDMR           */
                unsigned long LONG;                       /*  Long Access */
                struct {                                  /*  Word Access */
                        unsigned short H;                 /*   High         */
                        unsigned short L;                 /*   Low          */
                        } WORD;                           /*                */
                struct {                                  /*  Bit Access  */
                        unsigned long :25;                /*                */
                        unsigned long DE:1;               /*   DE           */
                        unsigned long DL:2;               /*   DL           */
                        unsigned long :3;                 /*                */
                        unsigned long SWR:1;              /*   SWR          */
                        } BIT;                            /*                */
                } EDMR;                                   /*                */
        union {                                           /* EDTRR          */
                unsigned long LONG;                       /*  Long Access */
                struct {                                  /*  Bit Access  */
                        unsigned long :31;                /*                */
                        unsigned long TR:1;               /*   TR           */
                        } BIT;                            /*                */
                } EDTRR;                                  /*                */
        union {                                           /* EDRRR          */
                unsigned long LONG;                       /*  Long Access */
                struct {                                  /*  Bit Access  */
                        unsigned long :31;                /*                */
                        unsigned long RR:1;               /*   RR           */
                        } BIT;                            /*                */
                } EDRRR;                                  /*                */
        void            *TDLAR;                           /* TDLAR          */
        void            *RDLAR;                           /* RDLAR          */
        union {                                           /* EESR           */
                unsigned long LONG;                       /*  Long Access */
                struct {                                  /*  Word Access */
                        unsigned short H;                 /*   High         */
                        unsigned short L;                 /*   Low          */
                        } WORD;                           /*                */
```

```
      struct {                                  /*  Bit Access  */
             unsigned long :1;                  /*              */
             unsigned long TWB:1;               /*   TWB        */
             unsigned long :3;                  /*              */
             unsigned long TABT:1;              /*   TABT       */
             unsigned long RABT:1;              /*   RABT       */
             unsigned long RFCOF:1;             /*   RFCOF      */
             unsigned long ADE:1;               /*   ADE        */
             unsigned long ECI:1;               /*   ECI        */
             unsigned long TC:1;                /*   TC         */
             unsigned long TDE:1;               /*   TDE        */
             unsigned long TFUF:1;              /*   TFUF       */
             unsigned long FR:1;                /*   FR         */
             unsigned long RDE:1;               /*   RDE        */
             unsigned long RFOF:1;              /*   RFOF       */
             unsigned long :4;                  /*              */
             unsigned long CND:1;               /*   CND        */
             unsigned long DLC:1;               /*   DLC        */
             unsigned long CD:1;                /*   CD         */
             unsigned long TRO:1;               /*   TRO        */
             unsigned long RMAF:1;              /*   RMAF       */
             unsigned long :2;                  /*              */
             unsigned long RRF:1;               /*   RRF        */
             unsigned long RTLF:1;              /*   RTLF       */
             unsigned long RTSF:1;              /*   RTSF       */
             unsigned long PRE:1;               /*   PRE        */
             unsigned long CERF:1;              /*   CERF       */
             } BIT;                             /*              */
      } EESR;                                   /*              */
union {                                         /* EESIPR       */
      unsigned long LONG;                       /*  Long Access */
      struct {                                  /*  Word Access */
             unsigned short H;                  /*   High       */
             unsigned short L;                  /*   Low        */
             } WORD;                            /*              */
      struct {                                  /*  Bit Access  */
             unsigned long :1;                  /*              */
             unsigned long TWBIP:1;             /*   TWBIP      */
             unsigned long :3;                  /*              */
             unsigned long TABTIP:1;            /*   TABTIP     */
             unsigned long RABTIP:1;            /*   RABTIP     */
             unsigned long RFCOFIP:1;           /*   RFCOFIP    */
             unsigned long ADEIP:1;             /*   ADEIP      */
             unsigned long ECIIP:1;             /*   ECIIP      */
             unsigned long TCIP:1;              /*   TCIP       */
             unsigned long TDEIP:1;             /*   TDEIP      */
             unsigned long TFUFIP:1;            /*   TFUFIP     */
             unsigned long FRIP:1;              /*   FRIP       */
             unsigned long RDEIP:1;             /*   RDEIP      */
             unsigned long RFOFIP:1;            /*   RFOFIP     */
             unsigned long :4;                  /*              */
             unsigned long CNDIP:1;             /*   CNDIP      */
             unsigned long DLCIP:1;             /*   DLCIP      */
             unsigned long CDIP:1;              /*   CDIP       */
             unsigned long TROIP:1;             /*   TROIP      */
             unsigned long RMAFIP:1;            /*   RMAFIP     */
             unsigned long :2;                  /*              */
             unsigned long RRFIP:1;             /*   RRFIP      */
             unsigned long RTLFIP:1;            /*   RTLFIP     */
             unsigned long RTSFIP:1;            /*   RTSFIP     */
             unsigned long PREIP:1;             /*   PREIP      */
             unsigned long CERFIP:1;            /*   CERFIP     */
             } BIT;                             /*              */
      } EESIPR;                                 /*              */
```

```
union {                                         /* TRSCER       */
      unsigned long LONG;                       /*  Long Access */
      struct {                                  /*  Word Access */
             unsigned short H;                  /*   High       */
             unsigned short L;                  /*   Low        */
             } WORD;                            /*              */
      struct {                                  /*  Bit Access  */
             unsigned long :20;                 /*              */
             unsigned long CNDCE:1;             /*   CNDCE      */
             unsigned long DLCCE:1;             /*   DLCCE      */
             unsigned long CDCE:1;              /*   CDCE       */
             unsigned long TROCE:1;             /*   TROCE      */
             unsigned long RMAFCE:1;            /*   RMAFCE     */
             unsigned long :2;                  /*              */
             unsigned long RRFCE:1;             /*   RRFCE      */
             unsigned long RTLFCE:1;            /*   RTLFCE     */
             unsigned long RTSFCE:1;            /*   RTSFCE     */
             unsigned long PRECE:1;             /*   PRECE      */
             unsigned long CERFCE:1;            /*   CERFCE     */
             } BIT;                             /*              */
      } TRSCER;                                 /*              */
union {                                         /* RMFCR        */
      unsigned long LONG;                       /*  Long Access */
      struct {                                  /*  Bit Access  */
             unsigned long :16;                 /*              */
             unsigned long MFC:16;              /*   MFC        */
             } BIT;                             /*              */
      } RMFCR;                                  /*              */
union {                                         /* TFTR         */
      unsigned long LONG;                       /*  Long Access */
      struct {                                  /*  Bit Access  */
             unsigned long :21;                 /*              */
             unsigned long TFT:11;              /*   TFT        */
             } BIT;                             /*              */
      } TFTR;                                   /*              */
union {                                         /* FDR          */
      unsigned long LONG;                       /*  Long Access */
      struct {                                  /*  Word Access */
             unsigned short H;                  /*   High       */
             unsigned short L;                  /*   Low        */
             } WORD;                            /*              */
      struct {                                  /*  Bit Access  */
             unsigned long :21;                 /*              */
             unsigned long TFD:3;               /*   TFD        */
             unsigned long :5;                  /*              */
             unsigned long RFD:3;               /*   RFD        */
             } BIT;                             /*              */
      } FDR;                                    /*              */
union {                                         /* RMCR         */
      unsigned long LONG;                       /*  Long Access */
      struct {                                  /*  Bit Access  */
             unsigned long :31;                 /*              */
             unsigned long RNC:1;               /*   RNC        */
             } BIT;                             /*              */
      } RMCR;                                   /*              */
unsigned char wk0[4];                           /*              */
union {                                         /* FCFTR        */
      unsigned long LONG;                       /*  Long Access */
      struct {                                  /*  Word Access */
             unsigned short H;                  /*   High       */
             unsigned short L;                  /*   Low        */
             } WORD;                            /*              */
      struct {                                  /*  Bit Access  */
             unsigned long :13;                 /*              */
```

```
                        unsigned long RFF:3;            /*   RFF        */
                        unsigned long :13;              /*              */
                        unsigned long RFD:3;            /*   RFD        */
                        } BIT;                          /*              */
                } FCFTR;                                /*              */
        unsigned char wk1[8];                           /*              */
        unsigned long RBWAR;                            /* RBWAR        */
        unsigned long RDFAR;                            /* RDFAR        */
        unsigned char wk2[4];                           /*              */
        unsigned long TBRAR;                            /* TBRAR        */
        unsigned long TDFAR;                            /* TDFAR        */
        union {                                         /* ECBRR        */
                unsigned char BYTE;                     /*  Byte Access */
                struct {                                /*  Bit Access  */
                        unsigned char :7;               /*              */
                        unsigned char RTM:1;            /*   RTM        */
                        } BIT;                          /*              */
                } ECBRR;                                /*              */
};                                                      /*              */
struct st_system {                                      /* struct SYSTEM */
        union {                                         /* SUBMSTPAH    */
                unsigned char BYTE;                     /*  Byte Access */
                struct {                                /*  Bit Access  */
                        unsigned char SMSTPA15:1;       /*   SMSTPA15   */
                        unsigned char EtherC:1;         /*   EtherC     */
                        unsigned char EDMAC:1;          /*   EDMAC      */
                        unsigned char USB:1;            /*   USB        */
                        unsigned char SMSTPA11:1;       /*   SMSTPA11   */
                        unsigned char SMSTPA10:1;       /*   SMSTPA10   */
                        unsigned char SMSTPA9:1;        /*   SMSTPA9    */
                        unsigned char SMSTPA8:1;        /*   SMSTPA8    */
                        } BIT;                          /*              */
                } SUBMSTPAH;                            /*              */
        union {                                         /* SUBMSTPAL    */
                unsigned char BYTE;                     /*  Byte Access */
                struct {                                /*  Bit Access  */
                        unsigned char SMSTPA7:1;        /*   SMSTPA7    */
                        unsigned char SMSTPA6:1;        /*   SMSTPA6    */
                        unsigned char SMSTPA5:1;        /*   SMSTPA5    */
                        unsigned char PECI:1;           /*   PECI       */
                        unsigned char SCIF:1;           /*   SCIF       */
                        unsigned char SSU:1;            /*   SSU        */
                        unsigned char LPC:1;            /*   LPC        */
                        unsigned char SMSTPA0:1;        /*   SMSTPA0    */
                        } BIT;                          /*              */
                } SUBMSTPAL;                            /*              */
        union {                                         /* SUBMSTPBH    */
                unsigned char BYTE;                     /*  Byte Access */
                struct {                                /*  Bit Access  */
                        unsigned char SMSTPB15:1;       /*   SMSTPB15   */
                        unsigned char EtherC:1;         /*   EtherC     */
                        unsigned char EDMAC:1;          /*   EDMAC      */
                        unsigned char USB:1;            /*   USB        */
                        unsigned char SMSTPB11:1;       /*   SMSTPB11   */
                        unsigned char SMSTPB10:1;       /*   SMSTPB10   */
                        unsigned char SMSTPB9:1;        /*   SMSTPB9    */
                        unsigned char SMSTPB8:1;        /*   SMSTPB8    */
                        } BIT;                          /*              */
                } SUBMSTPBH;                            /*              */
        union {                                         /* SUBMSTPBL    */
                unsigned char BYTE;                     /*  Byte Access */
                struct {                                /*  Bit Access  */
                        unsigned char SMSTPB7:1;        /*   SMSTPB7    */
                        unsigned char SMSTPB6:1;        /*   SMSTPB6    */
```

```
                     unsigned char SMSTPB5:1;           /*   SMSTPB5    */
                     unsigned char PECI:1;              /*   PECI       */
                     unsigned char SCIF:1;              /*   SCIF       */
                     unsigned char SSU:1;               /*   SSU        */
                     unsigned char LPC:1;               /*   LPC        */
                     unsigned char SMSTPB0:1;           /*   SMSTPB0    */
                     } BIT;                             /*              */
              } SUBMSTPBL;                              /*              */
        unsigned char wk0[3];                           /*              */
        unsigned char MSTPCRA;                          /* MSTPCRA      */
        unsigned char wk1[318];                         /*              */
        union {                                         /* PCSR         */
              unsigned char BYTE;                       /*  Byte Access */
              struct {                                  /*  Bit Access  */
                     unsigned char PWCKX1B:1;           /*   PWCKX1B    */
                     unsigned char PWCKX1A:1;           /*   PWCKX1A    */
                     unsigned char PWCKX0B:1;           /*   PWCKX0B    */
                     unsigned char PWCKX0A:1;           /*   PWCKX0A    */
                     unsigned char PWCKX1C:1;           /*   PWCKX1C    */
                     unsigned char PWCKB:1;             /*   PWCKB      */
                     unsigned char PWCKA:1;             /*   PWCKA      */
                     unsigned char PWCKX0C:1;           /*   PWCKX0C    */
                     } BIT;                             /*              */
              } PCSR;                                   /*              */
        union {                                         /* SYSCR2       */
              unsigned char BYTE;                       /*  Byte Access */
              struct {                                  /*  Bit Access  */
                     unsigned char :4;                  /*              */
                     unsigned char ADMXE:1;             /*   ADMXE      */
                     unsigned char :3;                  /*              */
                     } BIT;                             /*              */
              } SYSCR2;                                 /*              */
        union {                                         /* SBYCR        */
              unsigned char BYTE;                       /*  Byte Access */
              struct {                                  /*  Bit Access  */
                     unsigned char SSBY:1;              /*   SSBY       */
                     unsigned char STS2:1;              /*   STS2       */
                     unsigned char STS0:2;              /*   STS0       */
                     unsigned char DTSPEED:1;           /*   DTSPEED    */
                     unsigned char SCK:3;               /*   SCK        */
                     } BIT;                             /*              */
              } SBYCR;                                  /*              */
        union {                                         /* LPWRCR       */
              unsigned char BYTE;                       /*  Byte Access */
              struct {                                  /*  Bit Access  */
                     unsigned char DTON:1;              /*   DTON       */
                     unsigned char LSON:1;              /*   LSON       */
                     unsigned char NESEL:1;             /*   NESEL      */
                     unsigned char EXCLE:1;             /*   EXCLE      */
                     unsigned char :1;                  /*              */
                     unsigned char PNCCS:1;             /*   PNCCS      */
                     unsigned char PNCAH:1;             /*   PNCAH      */
                     unsigned char :1;                  /*              */
                     } BIT;                             /*              */
              } LPWRCR;                                 /*              */
        union {                                         /* MSTPCRH      */
              unsigned char BYTE;                       /*  Byte Access */
              struct {                                  /*  Bit Access  */
                     unsigned char MSTP15:1;            /*   MSTP15     */
                     unsigned char MSTP14:1;            /*   MSTP14     */
                     unsigned char MSTP13:1;            /*   MSTP13     */
                     unsigned char MSTP12:1;            /*   MSTP12     */
                     unsigned char MSTP11:1;            /*   MSTP11     */
                     unsigned char MSTP10:1;            /*   MSTP10     */
```

```
                    unsigned char MSTP9:1;             /*   MSTP9      */
                    unsigned char MSTP8:1;             /*   MSTP8      */
                    } BIT;                             /*              */
             } MSTPCRH;                                /*              */
       unsigned char MSTPCRL;                          /* MSTPCRL      */
       unsigned char wk2[59];                          /*              */
       union {                                         /* STCR         */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char IICX:3;              /*   IICX       */
                    unsigned char IICE:1;              /*   IICE       */
                    unsigned char FLSHE:1;             /*   FLSHE      */
                    unsigned char :1;                  /*              */
                    unsigned char ICKS:2;              /*   ICKS       */
                    } BIT;                             /*              */
             } STCR;                                   /*              */
       union {                                         /* SYSCR        */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char CS256E:1;            /*   CS256E     */
                    unsigned char IOSE:1;              /*   IOSE       */
                    unsigned char INTM:2;              /*   INTM       */
                    unsigned char XRST:1;              /*   XRST       */
                    unsigned char NMIEG:1;             /*   NMIEG      */
                    unsigned char :1;                  /*              */
                    unsigned char RAME:1;              /*   RAME       */
                    } BIT;                             /*              */
             } SYSCR;                                  /*              */
       union {                                         /* MDCR         */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char EXPE:1;              /*   EXPE       */
                    unsigned char :4;                  /*              */
                    unsigned char MDS:3;               /*   MDS        */
                    } BIT;                             /*              */
             } MDCR;                                   /*              */
};                                                     /*              */
struct st_int {                                        /* struct INT   */
       union {                                         /* ICRD         */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char ICRD7:1;             /*   ICRD7      */
                    unsigned char ICRD6:1;             /*   ICRD6      */
                    unsigned char ICRD5:1;             /*   ICRD6      */
                    unsigned char :5;                  /*              */
                    } BIT;                             /*              */
             } ICRD;                                   /*              */
       unsigned char ICRA;                             /* ICRA         */
       union {                                         /* ICRB         */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char ICRB7:1;             /*   ICRB7      */
                    unsigned char ICRB6:1;             /*   ICRB6      */
                    unsigned char :1;                  /*              */
                    unsigned char ICRB:5;              /*   ICRB       */
                    } BIT;                             /*              */
             } ICRB;                                   /*              */
       union {                                         /* ICRC         */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char ICRC7:1;             /*   ICRC7      */
                    unsigned char ICRC6:1;             /*   ICRC6      */
                    unsigned char ICRC5:1;             /*   ICRC5      */
                    unsigned char ICRC4:1;             /*   ICRC4      */
```

```
                        unsigned char ICRC3:1;          /*   ICRC3      */
                        unsigned char ICRC2:1;          /*   ICRC2      */
                        unsigned char ICRC1:1;          /*   ICRC1      */
                        unsigned char :1;               /*              */
                        } BIT;                          /*              */
                } ICRC;                                 /*              */
        union {                                         /* ISR          */
                unsigned char BYTE;                     /*  Byte Access */
                struct {                                /*  Bit Access  */
                        unsigned char IRQ7F:1;          /*   IRQ7F      */
                        unsigned char IRQ6F:1;          /*   IRQ6F      */
                        unsigned char IRQ5F:1;          /*   IRQ5F      */
                        unsigned char IRQ4F:1;          /*   IRQ4F      */
                        unsigned char IRQ3F:1;          /*   IRQ3F      */
                        unsigned char IRQ2F:1;          /*   IRQ2F      */
                        unsigned char IRQ1F:1;          /*   IRQ1F      */
                        unsigned char IRQ0F:1;          /*   IRQ0F      */
                        } BIT;                          /*              */
                } ISR;                                  /*              */
        union {                                         /* ISCRH        */
                unsigned char BYTE;                     /*  Byte Access */
                struct {                                /*  Bit Access  */
                        unsigned char IRQ7SCB:1;        /*   IRQ7SCB    */
                        unsigned char IRQ7SCA:1;        /*   IRQ7SCA    */
                        unsigned char IRQ6SCB:1;        /*   IRQ6SCB    */
                        unsigned char IRQ6SCA:1;        /*   IRQ6SCA    */
                        unsigned char IRQ5SCB:1;        /*   IRQ5SCB    */
                        unsigned char IRQ5SCA:1;        /*   IRQ5SCA    */
                        unsigned char IRQ4SCB:1;        /*   IRQ4SCB    */
                        unsigned char IRQ4SCA:1;        /*   IRQ4SCA    */
                        } BIT;                          /*              */
                } ISCRH;                                /*              */
        union {                                         /* ISCRL        */
                unsigned char BYTE;                     /*  Byte Access */
                struct {                                /*  Bit Access  */
                        unsigned char IRQ3SCB:1;        /*   IRQ3SCB    */
                        unsigned char IRQ3SCA:1;        /*   IRQ3SCA    */
                        unsigned char IRQ2SCB:1;        /*   IRQ2SCB    */
                        unsigned char IRQ2SCA:1;        /*   IRQ2SCA    */
                        unsigned char IRQ1SCB:1;        /*   IRQ1SCB    */
                        unsigned char IRQ1SCA:1;        /*   IRQ1SCA    */
                        unsigned char IRQ0SCB:1;        /*   IRQ0SCB    */
                        unsigned char IRQ0SCA:1;        /*   IRQ0SCA    */
                        } BIT;                          /*              */
                } ISCRL;                                /*              */
        unsigned char wk0[6];                           /*              */
        union {                                         /* ABRKCR       */
                unsigned char BYTE;                     /*  Byte Access */
                struct {                                /*  Bit Access  */
                        unsigned char CMF:1;            /*   CMF        */
                        unsigned char :4;               /*              */
                        unsigned char TESTSEL:2;        /*   TESTSEL    */
                        unsigned char BIE:1;            /*   BIE        */
                        } BIT;                          /*              */
                } ABRKCR;                               /*              */
        union {                                         /* BARA         */
                unsigned char BYTE;                     /*  Byte Access */
                struct {                                /*  Bit Access  */
                        unsigned char A23:1;            /*   A23        */
                        unsigned char A22:1;            /*   A22        */
                        unsigned char A21:1;            /*   A21        */
                        unsigned char A20:1;            /*   A20        */
                        unsigned char A19:1;            /*   A19        */
                        unsigned char A18:1;            /*   A18        */
```

```
                    unsigned char A17:1;            /*   A17        */
                    unsigned char A16:1;            /*   A16        */
                    } BIT;                          /*              */
             } BARA;                                /*              */
       union {                                      /* BARB         */
             unsigned char BYTE;                    /*  Byte Access */
             struct {                               /*  Bit Access  */
                    unsigned char A15:1;            /*   A15        */
                    unsigned char A14:1;            /*   A14        */
                    unsigned char A13:1;            /*   A13        */
                    unsigned char A12:1;            /*   A12        */
                    unsigned char A11:1;            /*   A11        */
                    unsigned char A10:1;            /*   A10        */
                    unsigned char A9:1;             /*   A9         */
                    unsigned char A8:1;             /*   A8         */
                    } BIT;                          /*              */
             } BARB;                                /*              */
       union {                                      /* BARC         */
             unsigned char BYTE;                    /*  Byte Access */
             struct {                               /*  Bit Access  */
                    unsigned char A7:1;             /*   A7         */
                    unsigned char A6:1;             /*   A6         */
                    unsigned char A5:1;             /*   A5         */
                    unsigned char A4:1;             /*   A4         */
                    unsigned char A3:1;             /*   A3         */
                    unsigned char A2:1;             /*   A2         */
                    unsigned char A1:1;             /*   A1         */
                    unsigned char :1;               /*              */
                    } BIT;                          /*              */
             } BARC;                                /*              */
       union {                                      /* IER16        */
             unsigned char BYTE;                    /*  Byte Access */
             struct {                               /*  Bit Access  */
                    unsigned char IRQ15E:1;         /*   IRQ15E     */
                    unsigned char IRQ14E:1;         /*   IRQ14E     */
                    unsigned char IRQ13E:1;         /*   IRQ13E     */
                    unsigned char IRQ12E:1;         /*   IRQ12E     */
                    unsigned char IRQ11E:1;         /*   IRQ11E     */
                    unsigned char IRQ10E:1;         /*   IRQ10E     */
                    unsigned char IRQ9E:1;          /*   IRQ9E      */
                    unsigned char IRQ8E:1;          /*   IRQ8E      */
                    } BIT;                          /*              */
             } IER16;                               /*              */
       union {                                      /* ISR16        */
             unsigned char BYTE;                    /*  Byte Access */
             struct {                               /*  Bit Access  */
                    unsigned char IRQ15F:1;         /*   IRQ15F     */
                    unsigned char IRQ14F:1;         /*   IRQ14F     */
                    unsigned char IRQ13F:1;         /*   IRQ13F     */
                    unsigned char IRQ12F:1;         /*   IRQ12F     */
                    unsigned char IRQ11F:1;         /*   IRQ11F     */
                    unsigned char IRQ10F:1;         /*   IRQ10F     */
                    unsigned char IRQ9F:1;          /*   IRQ9F      */
                    unsigned char IRQ8F:1;          /*   IRQ8F      */
                    } BIT;                          /*              */
             } ISR16;                               /*              */
       union {                                      /* ISCR16H      */
             unsigned char BYTE;                    /*  Byte Access */
             struct {                               /*  Bit Access  */
                    unsigned char IRQ15SCB:1;       /*   IRQ15SCB   */
                    unsigned char IRQ15SCA:1;       /*   IRQ15SCA   */
                    unsigned char IRQ14SCB:1;       /*   IRQ14SCB   */
                    unsigned char IRQ14SCA:1;       /*   IRQ14SCA   */
                    unsigned char IRQ13SCB:1;       /*   IRQ13SCB   */
```

```
                      unsigned char IRQ13SCA:1;          /*   IRQ13SCA   */
                      unsigned char IRQ12SCB:1;          /*   IRQ12SCB   */
                      unsigned char IRQ12SCA:1;          /*   IRQ12SCA   */
                      } BIT;                             /*              */
               } ISCR16H;                                /*              */
         union {                                         /* ISCR16L      */
               unsigned char BYTE;                       /*  Byte Access */
               struct {                                  /*  Bit Access  */
                      unsigned char IRQ11SCB:1;          /*   IRQ11SCB   */
                      unsigned char IRQ11SCA:1;          /*   IRQ11SCA   */
                      unsigned char IRQ10SCB:1;          /*   IRQ10SCB   */
                      unsigned char IRQ10SCA:1;          /*   IRQ10SCA   */
                      unsigned char IRQ9SCB:1;           /*   IRQ9SCB    */
                      unsigned char IRQ9SCA:1;           /*   IRQ9SCA    */
                      unsigned char IRQ8SCB:1;           /*   IRQ8SCB    */
                      unsigned char IRQ8SCA:1;           /*   IRQ8SCA    */
                      } BIT;                             /*              */
               } ISCR16L;                                /*              */
         unsigned char wk1[198];                         /*              */
         union {                                         /* IER          */
               unsigned char BYTE;                       /*  Byte Access */
               struct {                                  /*  Bit Access  */
                      unsigned char IRQ7E:1;             /*   IRQ7E      */
                      unsigned char IRQ6E:1;             /*   IRQ6E      */
                      unsigned char IRQ5E:1;             /*   IRQ5E      */
                      unsigned char IRQ4E:1;             /*   IRQ4E      */
                      unsigned char IRQ3E:1;             /*   IRQ3E      */
                      unsigned char IRQ2E:1;             /*   IRQ2E      */
                      unsigned char IRQ1E:1;             /*   IRQ1E      */
                      unsigned char IRQ0E:1;             /*   IRQ0E      */
                      } BIT;                             /*              */
               } IER;                                    /*              */
};                                                       /*              */

/******************************************************************************
Macro definitions
******************************************************************************/
#define MAC0 (*(volatile struct st_ether __evenaccess *)0xFFF900)    /* ETHER Address */
#define EDMAC0 (*(volatile struct st_edmac __evenaccess *)0xFFF980)    /* EDMAC Address */
#define SYSTEM (*(volatile struct st_system *)0xFFFE3C)  /* SYSTEM Address */
#define INT (*(volatile struct st_int *)0xFFFEE7)        /* INT Address  */

/******************************************************************************
Variable Externs
******************************************************************************/

/******************************************************************************
Functions Prototypes
******************************************************************************/




#endif /* IODEFINE2472_H */
```

Website and Support

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

Inquiries
http://www.renesas.com/inquiry

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jul 29, 2008	—	First edition issued
1.01	Feb 10, 2009	3	Oscillator added in figure 1
1.02	Jul 06, 2011		"Changed due to driver revision"
		2	Specifications and conditions changed
		18	Operation of sample program changed
		22	Figure 23 changed
		23	Figure 24 changed
		24	Figure 25 changed
		25	Figure 26 changed
		26	Figure 27 changed
		31 to 61	Listing of code for the sample program added and changed
		32	Section 2.9 added
			"Error in Application Note corrected"
		1	Description of device on which operation was confirmed changed
		3	Figure 1 changed
		17	Figure 17 changed
		20	Figure 21 changed
		28	Figure 29 changed
		29	Figure 30 changed

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.
   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products
   Before changing from one product to another, i.e. to one with a different type number, confirm that the change will not lead to problems.
   — The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.