SH7216 Group

Configuration to Receive Ethernet Frames

R01AN0052EJ0200
Rev. 2.00
Sep. 17, 2010

Summary

This application note describes the configuration example of the SH7216 microcomputers (MCUs) to receive Ethernet frames.

Target Device

SH7216 MCU

Contents

1. Introduction

1.1 Specifications

This application receives 10 Ethernet frames continuously.

1.2 Modules Used

- Pin Function Controller (PFC)
- Ethernet Controller (EtherC)
- Ethernet Controller Direct Memory Access Controller (E-DMAC)

1.3 Applicable Conditions

MCU	SH7216
Operating Frequencies	Internal clock: 200 MHz Bus clock: 50 MHz Peripheral clock: 50 MHz
Integrated Development Environment	Renesas Electronics Corporation High-performance Embedded Workshop Ver.4.07.00
C Compiler	Renesas Electronics SuperH RISC engine Family C/C++ Compiler Package Ver.9.03 Release 00
Compiler Options	Default setting in the High-performance Embedded Workshop (-cpu=sh2afpu -fpu=single -debug -gbr=auto -global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1)

1.4 Related Application Notes

For more information, refer to the following application notes:

- SH7216 Group Example of Initialization
- SH7216 Group Configuring the Ethernet PHY-LSI Auto-Negotiation
- SH7216 Group Configuration to Transmit Ethernet Frames

2. Applications

This application uses the Ethernet Controller (EtherC), and the Ethernet Controller Direct Memory Access Controller (E-DMAC).

2.1 Overview

The SH7216 always uses the EtherC and the E-DMAC in the Ethernet communication. The EtherC controls reception, and the E-DMAC handles the DMA transfer between the transmit or receive FIFOs and the area to store data specified by user (buffer).

2.1.1 EtherC Overview

The SH7216 includes an Ethernet Controller (EtherC) which is compliant with the IEEE 802.3 MAC (Media Access Control) protocol. Connect the EtherC with the IEEE 802.3-compliant physical layer LSI (PHY-LSI) to transmit and receive Ethernet/IEEE 802.3 frames. The EtherC includes one MAC layer interface. As it is connected with the E-DMAC internally, the EtherC can access memory in high-speed.

Figure 1 shows the EtherC configuration.

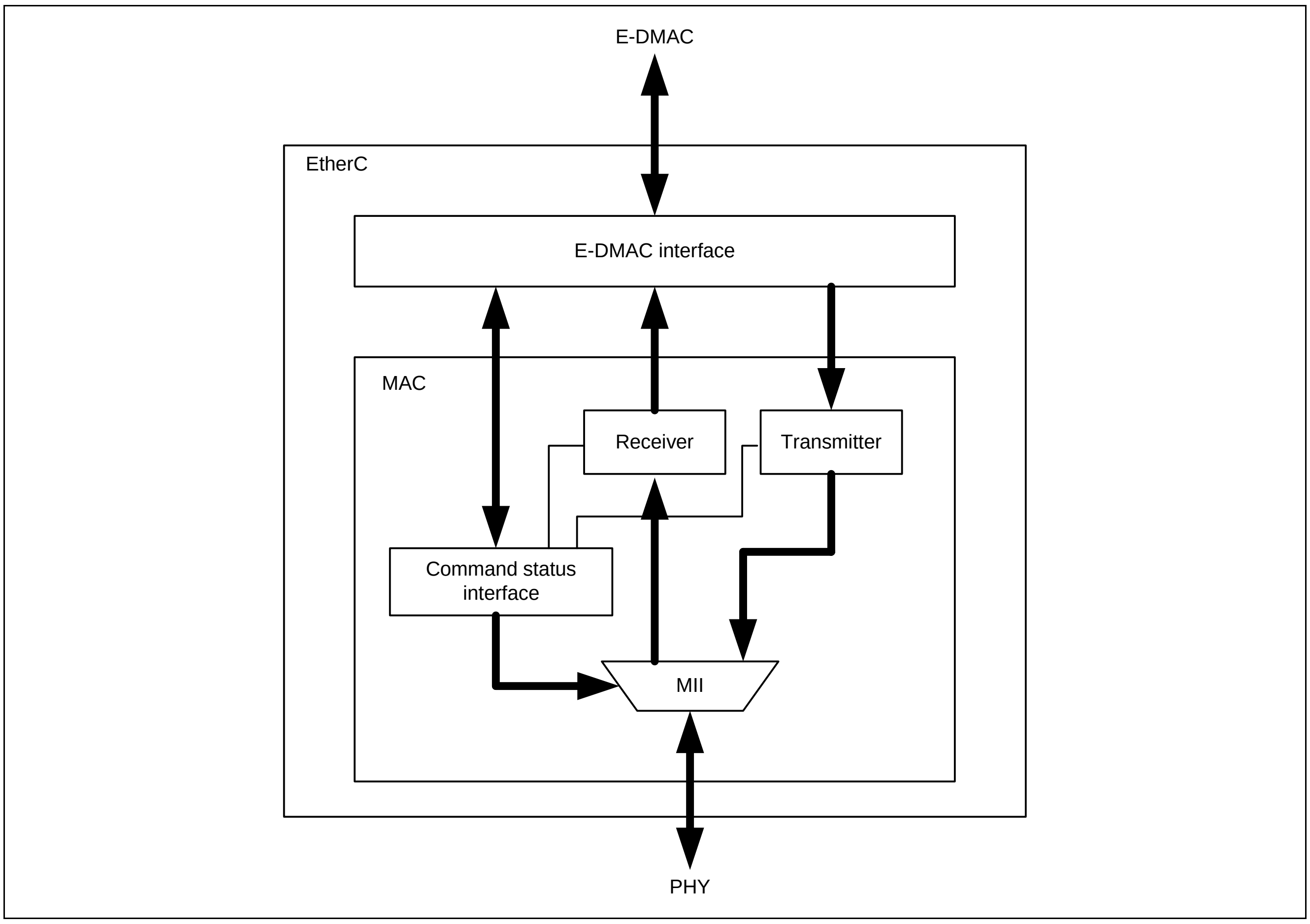


Figure 1 EtherC Configuration

2.1.2 EtherC Receiver Overview

The EtherC receiver divides a frame input from the MII (Media Independent Interface) into a preamble, SFD (Start Frame Delimiter), data, and CRC (Cyclic Redundancy Check) data. Then, the EtherC receiver outputs the portion of the frame other than the preamble, SFD, and CRC data to the E-DMAC receiver. Figure 2 shows the state transition diagram of the EtherC receiver. The reception sequence is as follows;

1. When the RE (Reception Enable) bit in the EtherC mode register (ECMR) is set, the EtherC transitions to the receive idle state.
2. When it detects the SFD which follows the preamble in the receive frame, the EtherC starts reception. When the pattern is invalid, the frame is discarded.
3. In normal mode, the EtherC starts to receive data when (i) the destination MAC address is the SH7216. (ii) the frame is the broadcast frame, or (iii) the frame is the multicast frame. In promiscuous mode, the EtherC automatically starts reception despite the type of frame.
4. The EtherC checks the CRC in the frame data after receiving the frame from the MII. It reflects the CRC result in the descriptor as the status after writing the frame data to the memory. When in error, the EtherC sets the error status in the EtherC/E-DMAC status register (EESR).
5. When the EtherC receives a frame, it transitions to idle state and is ready to receive the next frame.

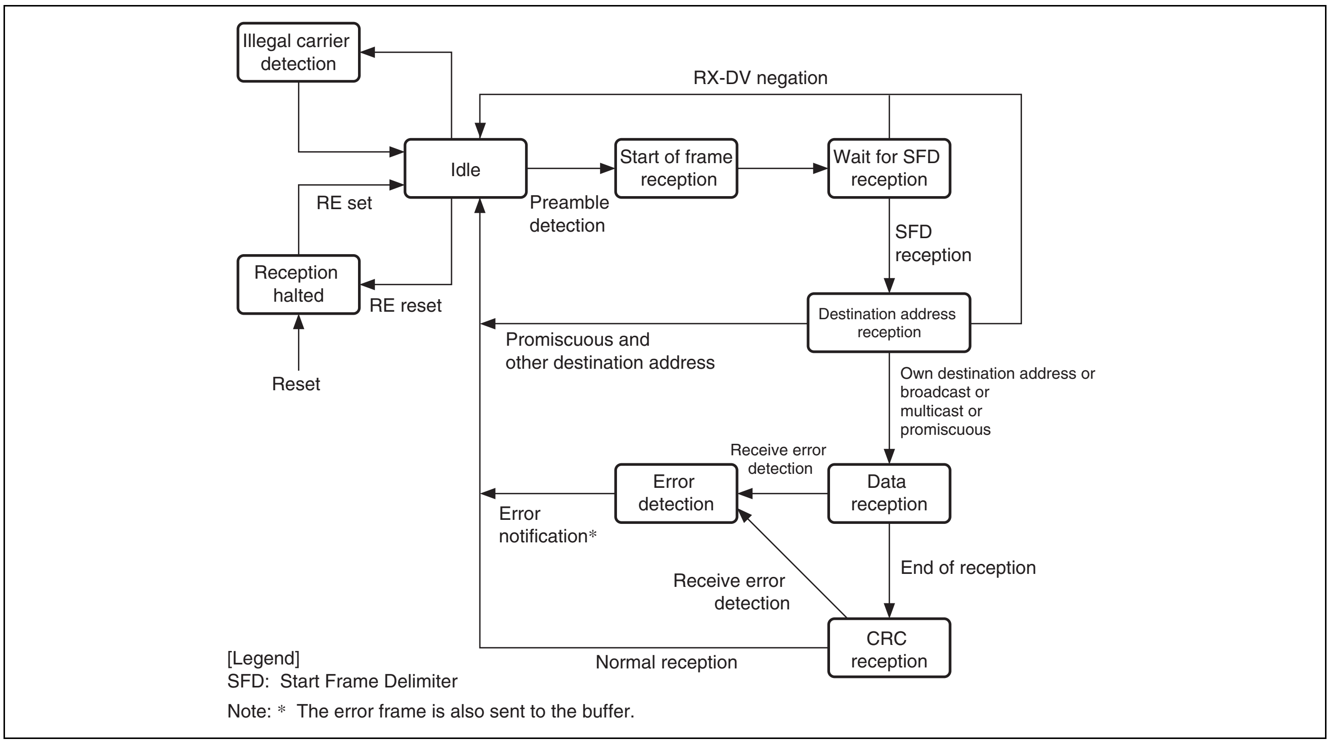


Figure 2 EtherC Receiver State Transition Diagram

2.1.3 E-DMAC Overview

The SH7216 includes the Direct Memory Access Controller (E-DMAC) which is directly connected with the EtherC. The E-DMAC uses its internal DMAC to handle the DMA transfer between the transmit or receive FIFOs in the E-DMAC and the area to store data specified by user (transmit or receive buffer). CPU cannot read or write the FIFO data directly. The information that the E-DMAC refers during the DMA transfer is the transmit or receive descriptor, and user must allocate these descriptors on memory. The E-DMAC retrieves the descriptor information before transmitting or receiving Ethernet frames. Then, it reads the transmit data from the transmit buffer or writes the receive data to the receive buffer, according to the descriptor information. Allocating multiple descriptors to make up the descriptor strings (list) allows for transmitting or receiving multiple Ethernet frames sequentially.

This E-DMAC feature reduces the load on the CPU to transmit or receive data efficiently. Figure 3 shows the configuration of the E-DMAC, descriptors, and buffer.

The features of the E-DMAC are as follows;

- Includes two channels (transmit and receive) of the DMAC
- Manages descriptors to reduce the load on the CPU
- Reflects the transmit/receive frame status to the descriptor
- Uses the system bus efficiently by the DMA block transfer (in units of 16-byte)
- Supports one frame per one descriptor, and one frame per multiple frames (multi buffer) - refer to section 2.1.5.

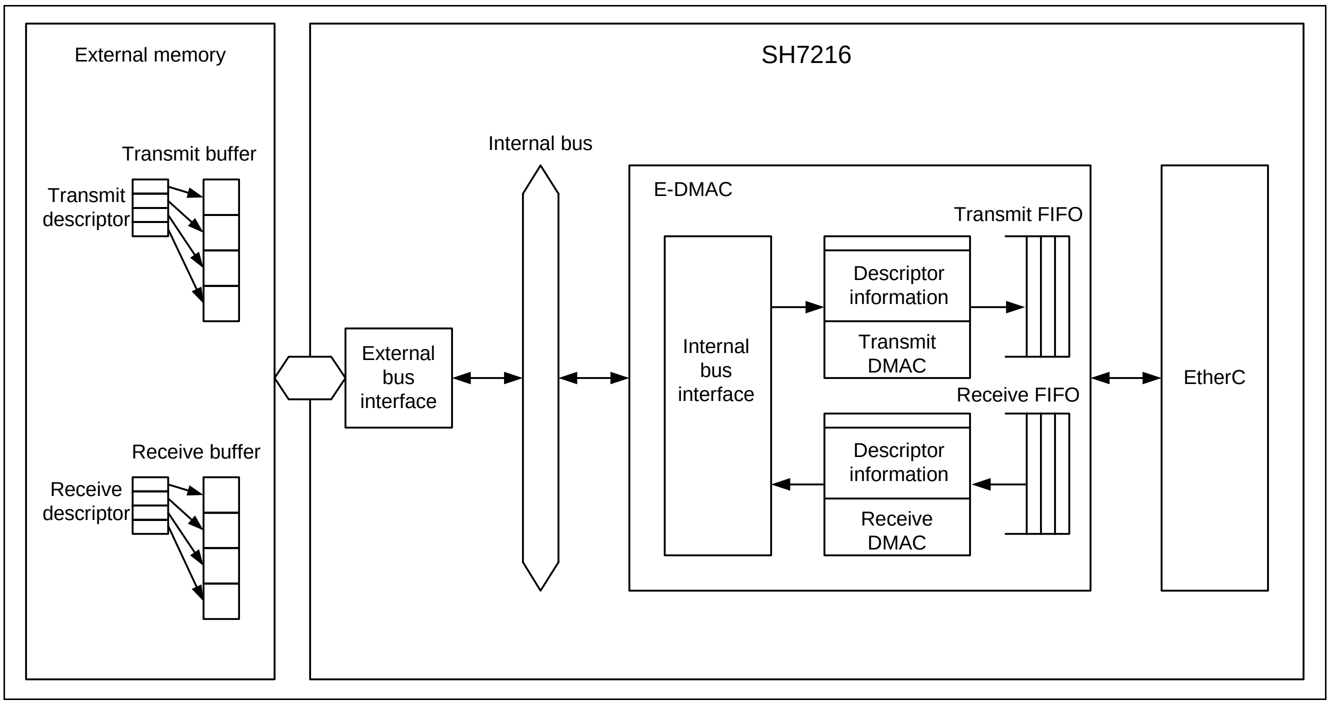


Figure 3 E-DMAC, Descriptors, and Buffer Configuration

2.1.4 Descriptor Overview

The E-DMAC requires the descriptor information (data), which includes the address storing transmit or receive data to use the DMA transfer. There are two types of descriptors; the transmit descriptor and receive descriptor. When the TR bit in the E-DMAC transmit request register (EDTRR) is set to 1, the E-DMAC automatically starts reading the transmit descriptor. When the RR bit in the E-DMAC receive request register (EDRRR) is set to 1, the E-DMAC automatically starts reading the receive descriptor. Before starting transmission or reception, user must include information regarding the DMA transfer of the transmit or receive data in the transmit or receive descriptor. After transmitting or receiving Ethernet frames are completed, the E-DMAC sets the enable/disable bit in the descriptor (TACT bit when transmitting, RACT bit when receiving), and reflects the transmission or reception result in the status bit (TFS25 to TFS0 when transmitting, RFS26 to RFS0 when receiving).

Align the descriptor on the read- and write-enabled memory, and set the starting descriptor (The first descriptor read by the E-DMAC) address in the Transmit descriptor list start address register (TDLAR) or the Receive descriptor list start address register (RDLAR). When using multiple descriptors as the descriptor string (descriptor list), align the descriptors on the contiguous addresses, according to the length of descriptor set in bits DL0 and DL1 in the E-DMAC mode register (EDMR).

2.1.5 Receive Descriptor Overview

Figure 4 shows the relationship between the receive descriptor and the receive buffer.

The receive descriptor consists of RD0, RD1, RD2, and padding in units of 32-bit from the top of the data. RD0 indicates if the receive descriptor is valid or invalid, the descriptor configuration information and status information. RD1 indicates the size of the receive buffer (RBL) and the data length of the received frame (RFL) referred by the receive descriptor. RD2 indicates the starting address in the receive buffer. The length of padding is determined by the descriptor length specified in bits DL0 and DL1 in the EDMR register.

According to the receive descriptor setting, both storing all receive data in one frame in the receive buffer by one descriptor (one frame per one descriptor), and storing all receive data in one frame in the receive buffer by multiple descriptors (one frame per multiple descriptors) are allowed. When using one frame per multiple descriptors, user must prepare multiple descriptors (descriptor list). When the E-DMAC receives the frame longer than the RBL in the descriptor, it uses following descriptors to transfer the frame to the receive buffer.

For example, when the E-DMAC receives 1514 bytes of the Ethernet frames upon setting the RBL as 500 bytes, it transfers the 500-byte Ethernet frames at a time from the first descriptor to buffers, and transfers the last 14-byte frames to the fourth buffer.

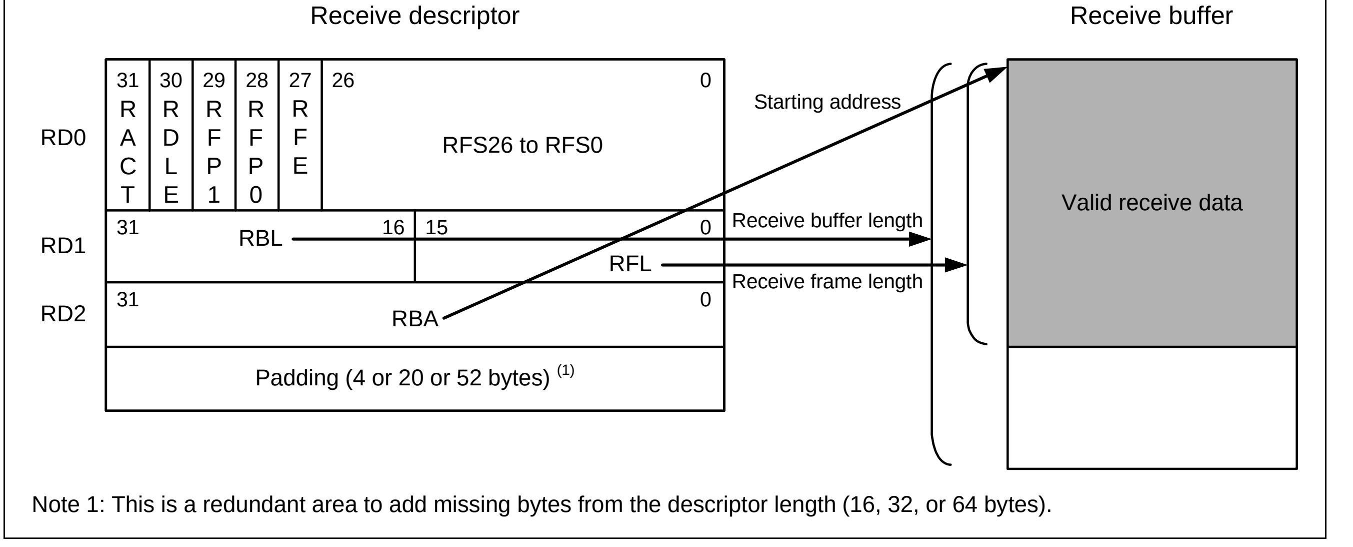


Figure 4 Relationship between the Receive Descriptor and the Receive Buffer

2.1.6 Setting the Receive Descriptor

Figure 5 shows an example of setting to use three receive descriptors and three receive buffers. This example uses 1536 bytes in each receive buffer to set one frame per one descriptor. The following figure simplifies the receive descriptors as only RD0. Numbers shown in the figure indicates sequence to execute.

Set the receive descriptor as following steps;

1. Set 0 in bits RFP1, RFP0, RFE, RFS26 to RFS0 in all descriptors.
2. Set 0 in the RDLE bit in descriptors 1 and 2. Then, set 1 in the RDLE bit in descriptor 3 and complete the descriptor processing to read descriptor 1. These settings create the descriptor ring structure.
3. Before starting to receive data, set 1536 bytes (receive buffer size) in the RBL in RD1 in all descriptors, and specify the RBA bit in RD2 as the starting address in the corresponding receive buffer. (This step is not described in Figure 5.)
4. Set 1 in the RACT bit in all descriptors to receive data continuously. Details on the setting procedure are described in the next chapter.

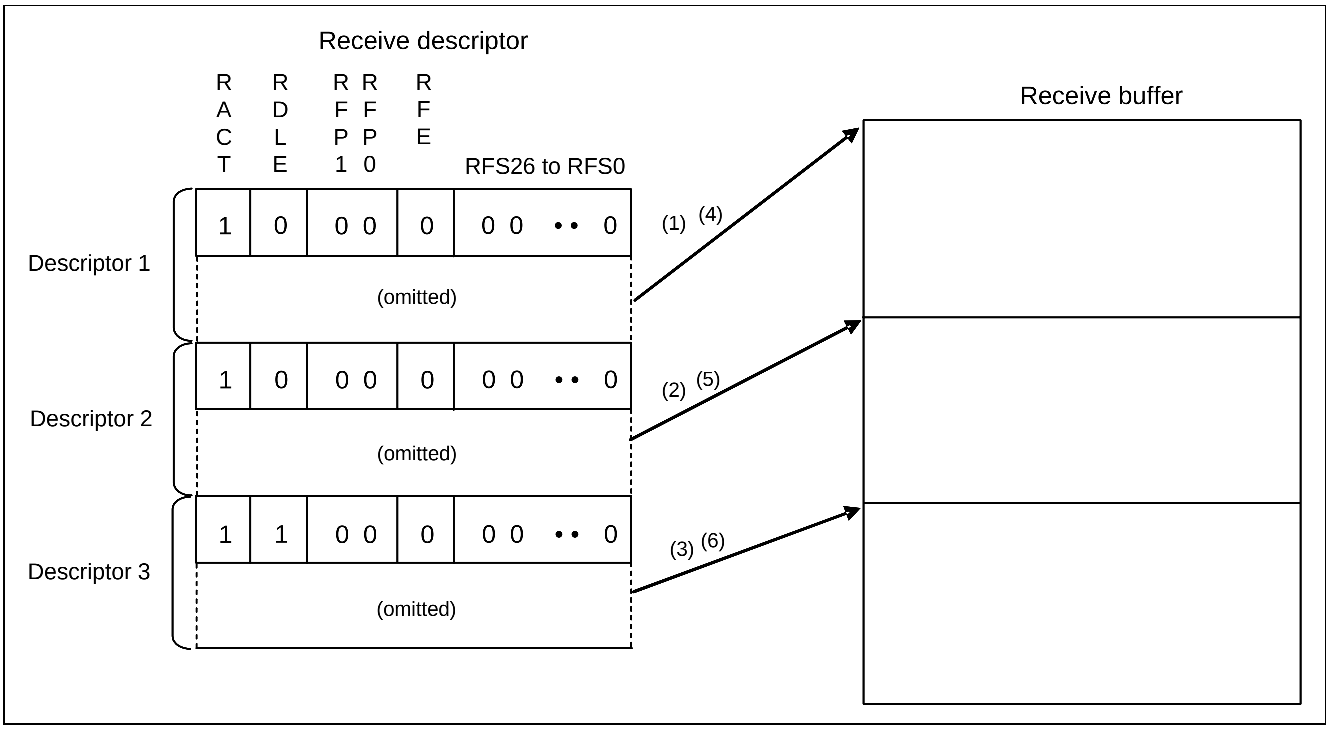


Figure 5 Relationship between 3 Receive Descriptors and 3 Receive Buffers

2.1.7 Operation Procedure (When Receiving)

The E-DMAC activates when writing 1 to the Receive request (RR) bit in the EDRRR register upon the RE bit in the ECMR is 1. After the EtherC and E-DMAC are reset by software, the E-DMAC reads the descriptor specified by the Receive descriptor list start address register (RDLAR) and enters the receive-ready state if the RACT bit is 1 (valid). When the EtherC receives frames to the local node (the address allowed by the EtherC to receive), it stores the receive data in the receive FIFO. When the RACT bit in the receive descriptor is 1, the EtherC transfers the data to the receive buffer specified in RD2 (When the RACT bit is 0, which indicates invalid, the EtherC clears the RR bit to stop the E-DMAC reception). When the length of the frame received is longer than the buffer length specified in RD1, the E-DMAC writes back the descriptor (RFP = B'10 or B'00) when the buffer is full. Then, it reads the following descriptor. When receiving frames is completed or receive operation is suspended due to error, the E-DMAC writes back such descriptor (RFP = B'11 or B'01). If the E-DMAC is set to receive data continuously, that is when the Receive request bit reset (RNR) bit in the Receiving method control register (RMCR) is 1, it reads the next descriptor and enters the receive-ready state when the RACT bit is 1. When the RACT bit is 0 (the receive buffer is empty) or the E-DMAC is not set to receive data continuously (the RNR bit in the RMCR register is 0), the E-DMAC sets the RR bit in the EDRRR register to 0 and completes receiving. When setting the RR bit to 1 again, the E-DMAC receives the descriptor next to the last descriptor received. If the Receive request bit non-reset mode bit (RNC) in the RMCR register is set to 1, the RR bit in the EDRRR register will not be set to 0 when the receive descriptor is empty. The E-DMAC continues receiving data.

Figure 6 shows the flow chart of receiving frames (one frame per one descriptor, specifying to receive data continuously).

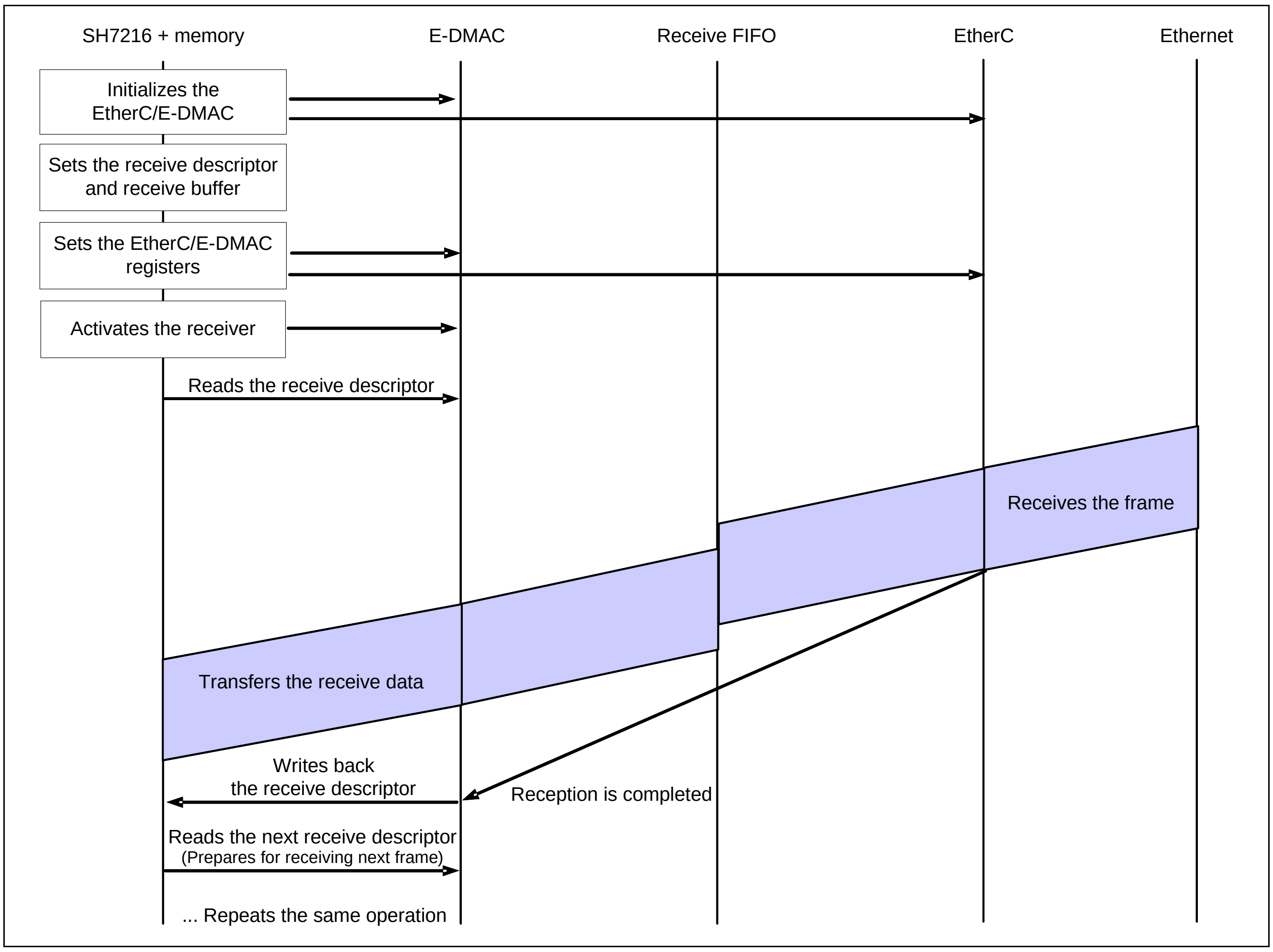

SH7216 + memory
E-DMAC
Receive FIFO
EtherC
Ethernet
Initializes the EtherC/E-DMAC
Sets the receive descriptor and receive buffer
Sets the EtherC/E-DMAC registers
Activates the receiver
Reads the receive descriptor
Receives the frame
Transfers the receive data
Writes back the receive descriptor
Reception is completed
Reads the next receive descriptor (Prepares for receiving next frame)
... Repeats the same operation

Figure 6 Flow Chart of Receiving (One Frame per One Descriptor)

2.1.8 Setting Procedure (When Receiving)

This section describes the basic settings for receiving Ethernet frames. Figure 7 and Figure 8 show flow charts of receiving Ethernet frames.

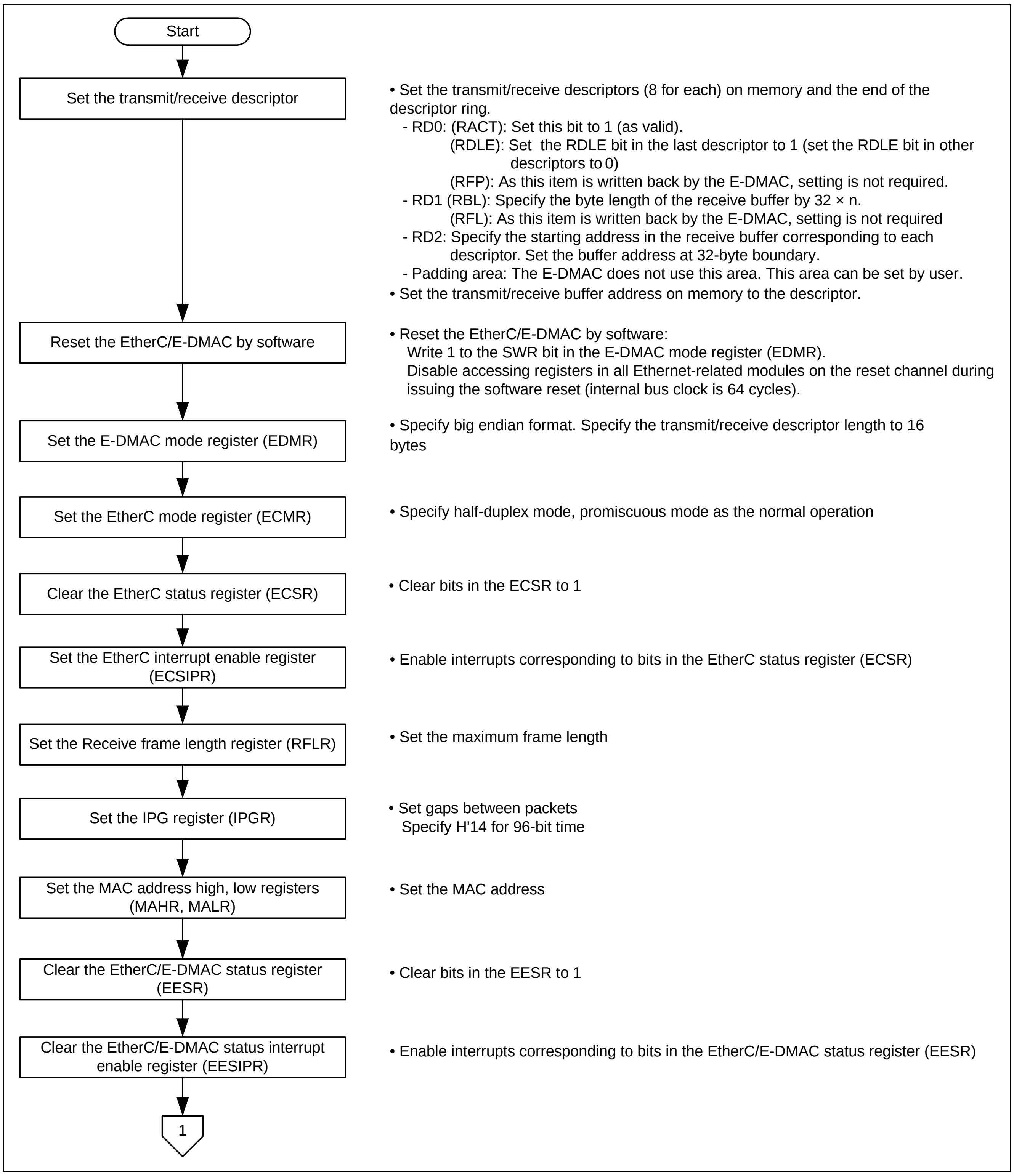

Start
Set the transmit/receive descriptor
• Set the transmit/receive descriptors (8 for each) on memory and the end of the descriptor ring.
- RD0: (RACT): Set this bit to 1 (as valid).
(RDLE): Set the RDLE bit in the last descriptor to 1 (set the RDLE bit in other descriptors to 0)
(RFP): As this item is written back by the E-DMAC, setting is not required.
- RD1 (RBL): Specify the byte length of the receive buffer by 32 × n.
(RFL): As this item is written back by the E-DMAC, setting is not required
- RD2: Specify the starting address in the receive buffer corresponding to each descriptor. Set the buffer address at 32-byte boundary.
- Padding area: The E-DMAC does not use this area. This area can be set by user.
• Set the transmit/receive buffer address on memory to the descriptor.
Reset the EtherC/E-DMAC by software
• Reset the EtherC/E-DMAC by software: Write 1 to the SWR bit in the E-DMAC mode register (EDMR). Disable accessing registers in all Ethernet-related modules on the reset channel during issuing the software reset (internal bus clock is 64 cycles).
Set the E-DMAC mode register (EDMR)
• Specify big endian format. Specify the transmit/receive descriptor length to 16 bytes
Set the EtherC mode register (ECMR)
• Specify half-duplex mode, promiscuous mode as the normal operation
Clear the EtherC status register (ECSR)
• Clear bits in the ECSR to 1
Set the EtherC interrupt enable register (ECSIPR)
• Enable interrupts corresponding to bits in the EtherC status register (ECSR)
Set the Receive frame length register (RFLR)
• Set the maximum frame length
Set the IPG register (IPGR)
• Set gaps between packets Specify H'14 for 96-bit time
Set the MAC address high, low registers (MAHR, MALR)
• Set the MAC address
Clear the EtherC/E-DMAC status register (EESR)
• Clear bits in the EESR to 1
Clear the EtherC/E-DMAC status interrupt enable register (EESIPR)
• Enable interrupts corresponding to bits in the EtherC/E-DMAC status register (EESR)
1

Figure 7 Receiving Ethernet Frames (1/2)

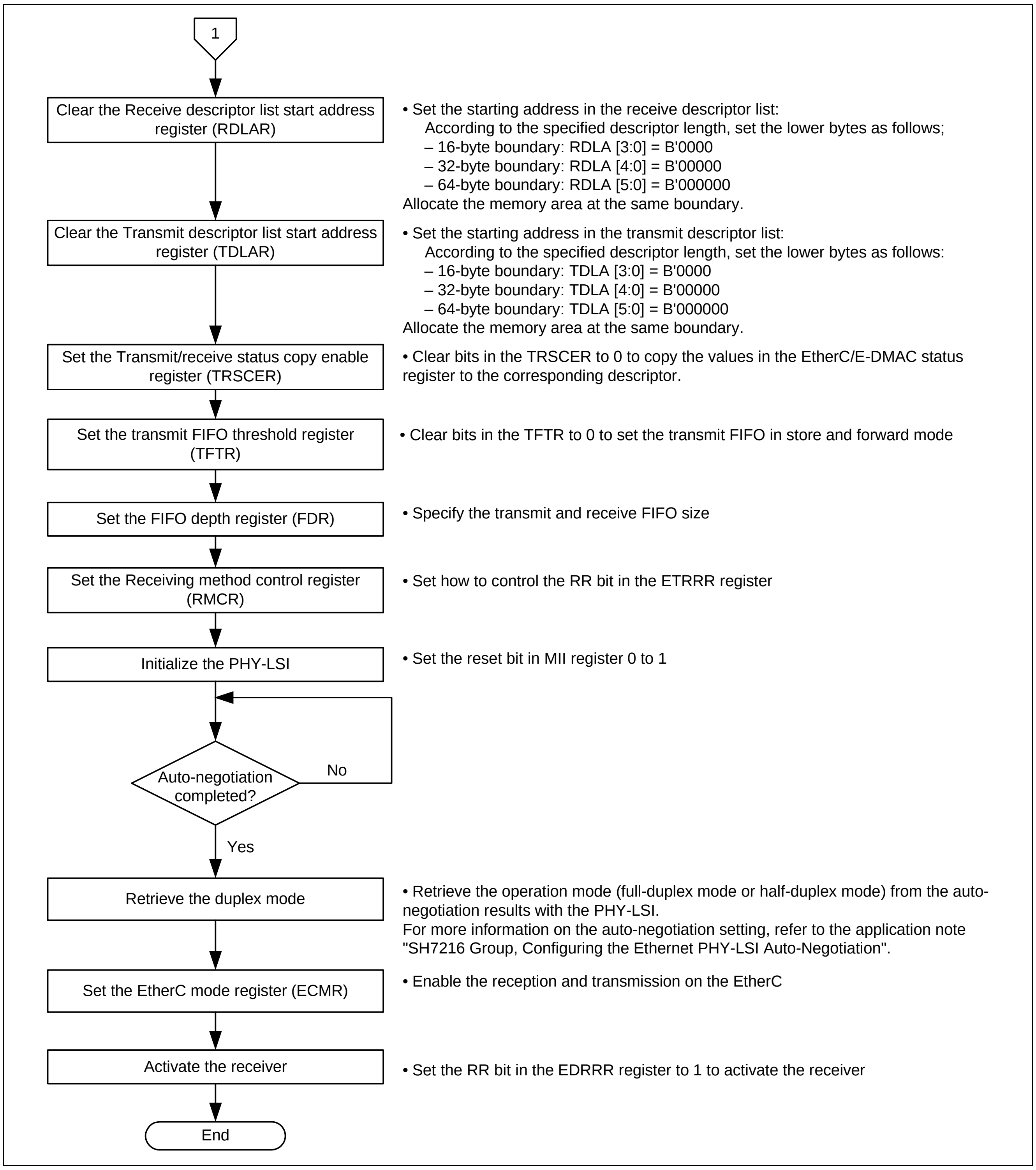

1
Clear the Receive descriptor list start address register (RDLAR)
• Set the starting address in the receive descriptor list:
According to the specified descriptor length, set the lower bytes as follows;
– 16-byte boundary: RDLA [3:0] = B'0000
– 32-byte boundary: RDLA [4:0] = B'00000
– 64-byte boundary: RDLA [5:0] = B'000000
Allocate the memory area at the same boundary.
Clear the Transmit descriptor list start address register (TDLAR)
• Set the starting address in the transmit descriptor list:
According to the specified descriptor length, set the lower bytes as follows:
– 16-byte boundary: TDLA [3:0] = B'0000
– 32-byte boundary: TDLA [4:0] = B'00000
– 64-byte boundary: TDLA [5:0] = B'000000
Allocate the memory area at the same boundary.
Set the Transmit/receive status copy enable register (TRSCER)
• Clear bits in the TRSCER to 0 to copy the values in the EtherC/E-DMAC status register to the corresponding descriptor.
Set the transmit FIFO threshold register (TFTR)
• Clear bits in the TFTR to 0 to set the transmit FIFO in store and forward mode
Set the FIFO depth register (FDR)
• Specify the transmit and receive FIFO size
Set the Receiving method control register (RMCR)
• Set how to control the RR bit in the ETRRR register
Initialize the PHY-LSI
• Set the reset bit in MII register 0 to 1
Auto-negotiation completed?
No
Yes
Retrieve the duplex mode
• Retrieve the operation mode (full-duplex mode or half-duplex mode) from the auto-negotiation results with the PHY-LSI.
For more information on the auto-negotiation setting, refer to the application note "SH7216 Group, Configuring the Ethernet PHY-LSI Auto-Negotiation".
Set the EtherC mode register (ECMR)
• Enable the reception and transmission on the EtherC
Activate the receiver
• Set the RR bit in the EDRRR register to 1 to activate the receiver
End

Figure 8 Receiving Ethernet Frames (2/2)

2.2 Sample Program Operation

The sample program uses the EtherC and the E-DMAC to receive 10 Ethernet frames from the host at the other end. It has receive descriptors and 256-byte receive buffers (total: 8). Also, it sets the RNR bit in the RMCR register to 1 to receive frames continuously.

The sample program transfers the portion of the Ethernet frame other than the preamble, SFD, and CRC data.

Figure 9 shows the operation environment of the sample program. Figure 10 shows the Ethernet frame format.

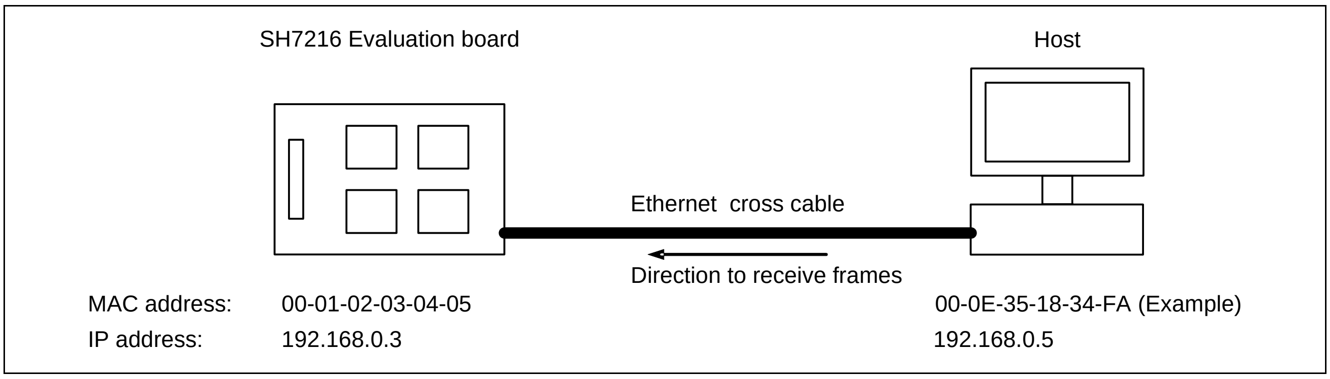


Figure 9 Sample Program Operation Environment

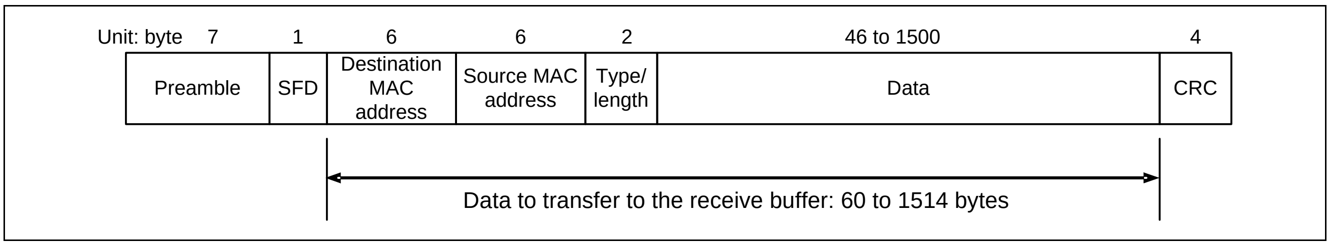


Figure 10 Ethernet Frame Format

2.3 Descriptor Definition in the Sample Program

The E-DMAC does not use the padding area in the descriptor, and that area can be used by user. The sample program sets the starting address in the next descriptor in the padding area to create the ring structure by software. Figure 11 shows the definition of the receive descriptor structure in the sample program and example to use the receive descriptor string.

Receive descriptor structure definition

```
typedef struct Descriptor
{
        uint32_t            status;
        uint16_t            bufsize;
        uint16_t            size;
        int8_t              *buf_p;
        struct Descriptor   *next;
} ethfifo;
```

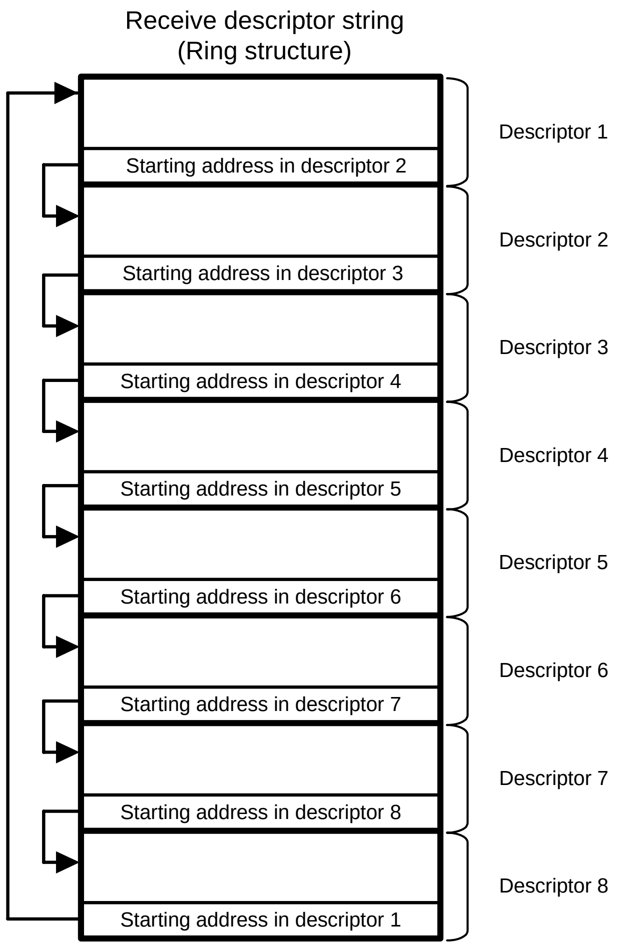

Receive descriptor string
(Ring structure)
Starting address in descriptor 2
Descriptor 1
Starting address in descriptor 3
Descriptor 2
Starting address in descriptor 4
Descriptor 3
Starting address in descriptor 5
Descriptor 4
Starting address in descriptor 6
Descriptor 5
Starting address in descriptor 7
Descriptor 6
Starting address in descriptor 8
Descriptor 7
Starting address in descriptor 1
Descriptor 8

Figure 11 Receive Descriptor Structure Definition and Example to Use the Receive Descriptor String

2.4 Sample Program Flow Chart

Figure 12 to Figure 15 shows flow charts of the sample program. For details on the function to retrieve the auto-negotiation result (phy_set_autonegotiate function), refer to the application note "SH7216 Group, Configuring the Ethernet PHY-LSI Auto-Negotiation".

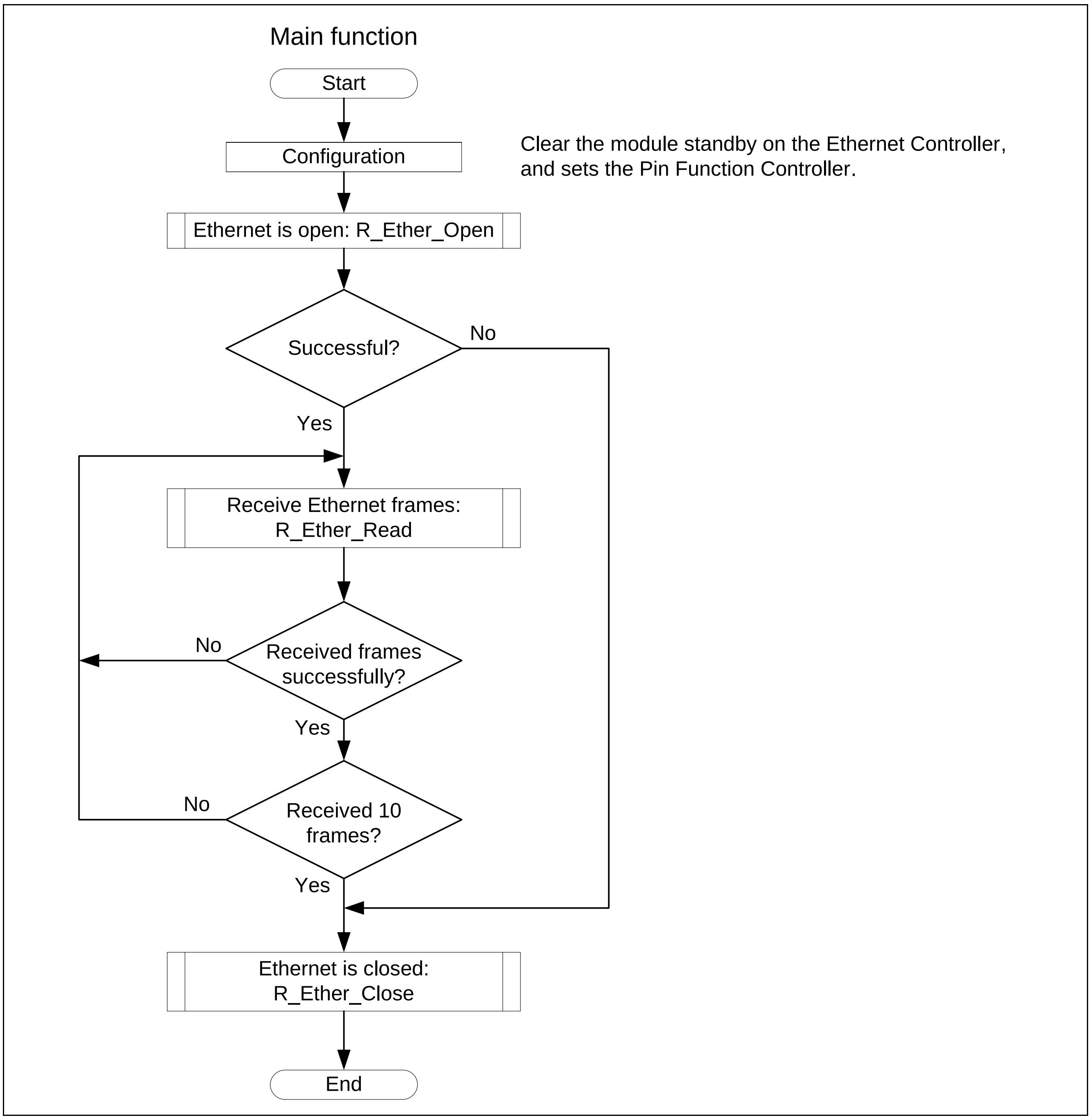
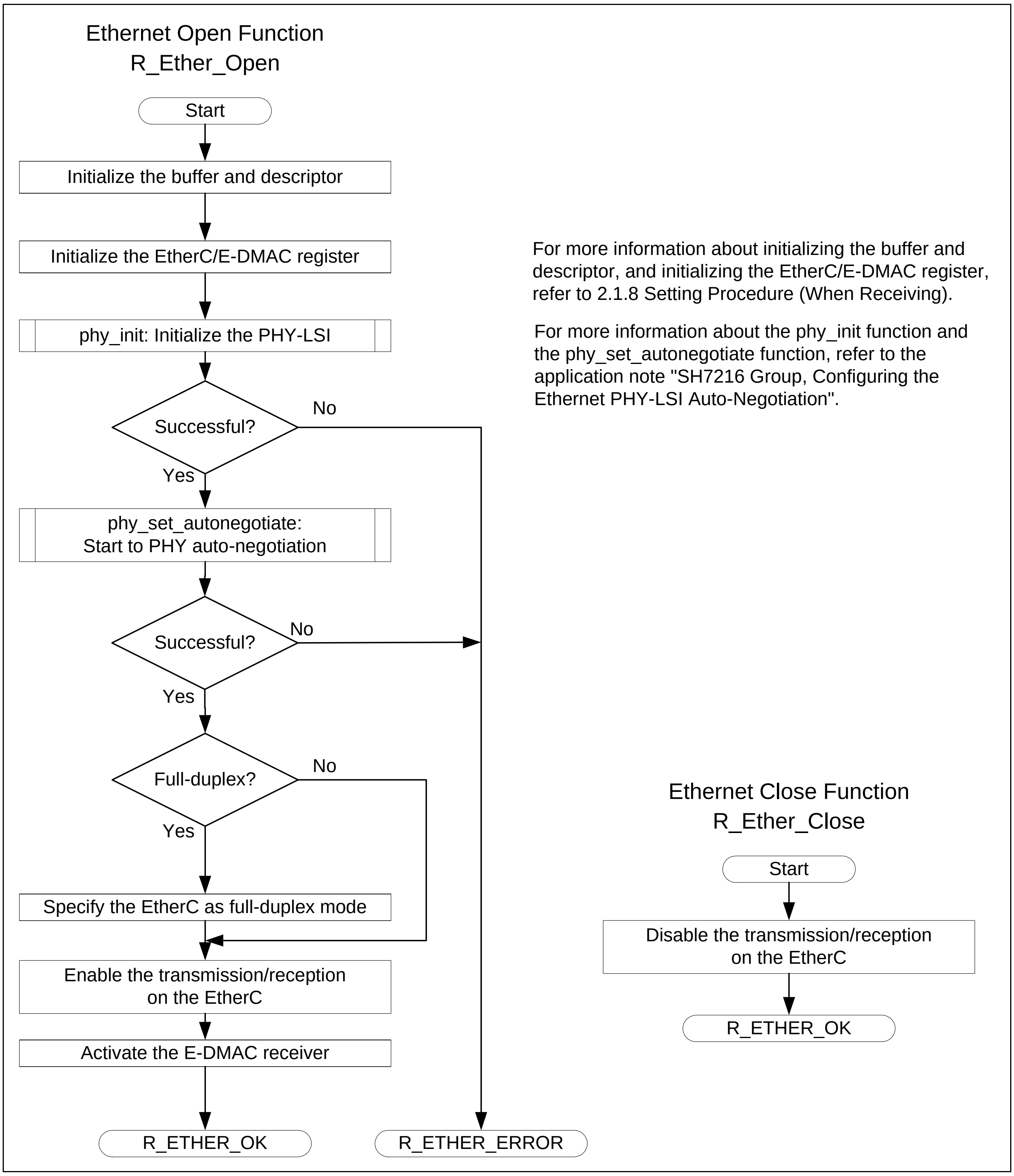

Main function
Start
Configuration
Clear the module standby on the Ethernet Controller, and sets the Pin Function Controller.
Ethernet is open: R_Ether_Open
Successful?
No
Yes
Receive Ethernet frames: R_Ether_Read
Received frames successfully?
No
Yes
Received 10 frames?
No
Yes
Ethernet is closed: R_Ether_Close
End

Figure 12 Sample Program Flow Chart (1/4)


Ethernet Open Function
R_Ether_Open
Start
Initialize the buffer and descriptor
Initialize the EtherC/E-DMAC register
phy_init: Initialize the PHY-LSI
Successful?
No
Yes
phy_set_autonegotiate: Start to PHY auto-negotiation
Successful?
No
Yes
Full-duplex?
No
Yes
Specify the EtherC as full-duplex mode
Enable the transmission/reception on the EtherC
Activate the E-DMAC receiver
R_ETHER_OK
R_ETHER_ERROR
Ethernet Close Function
R_Ether_Close
Start
Disable the transmission/reception on the EtherC
R_ETHER_OK

For more information about initializing the buffer and descriptor, and initializing the EtherC/E-DMAC register, refer to 2.1.8 Setting Procedure (When Receiving).

For more information about the phy_init function and the phy_set_autonegotiate function, refer to the application note "SH7216 Group, Configuring the Ethernet PHY-LSI Auto-Negotiation".

Figure 13 Sample Program Flow Chart (2/4)

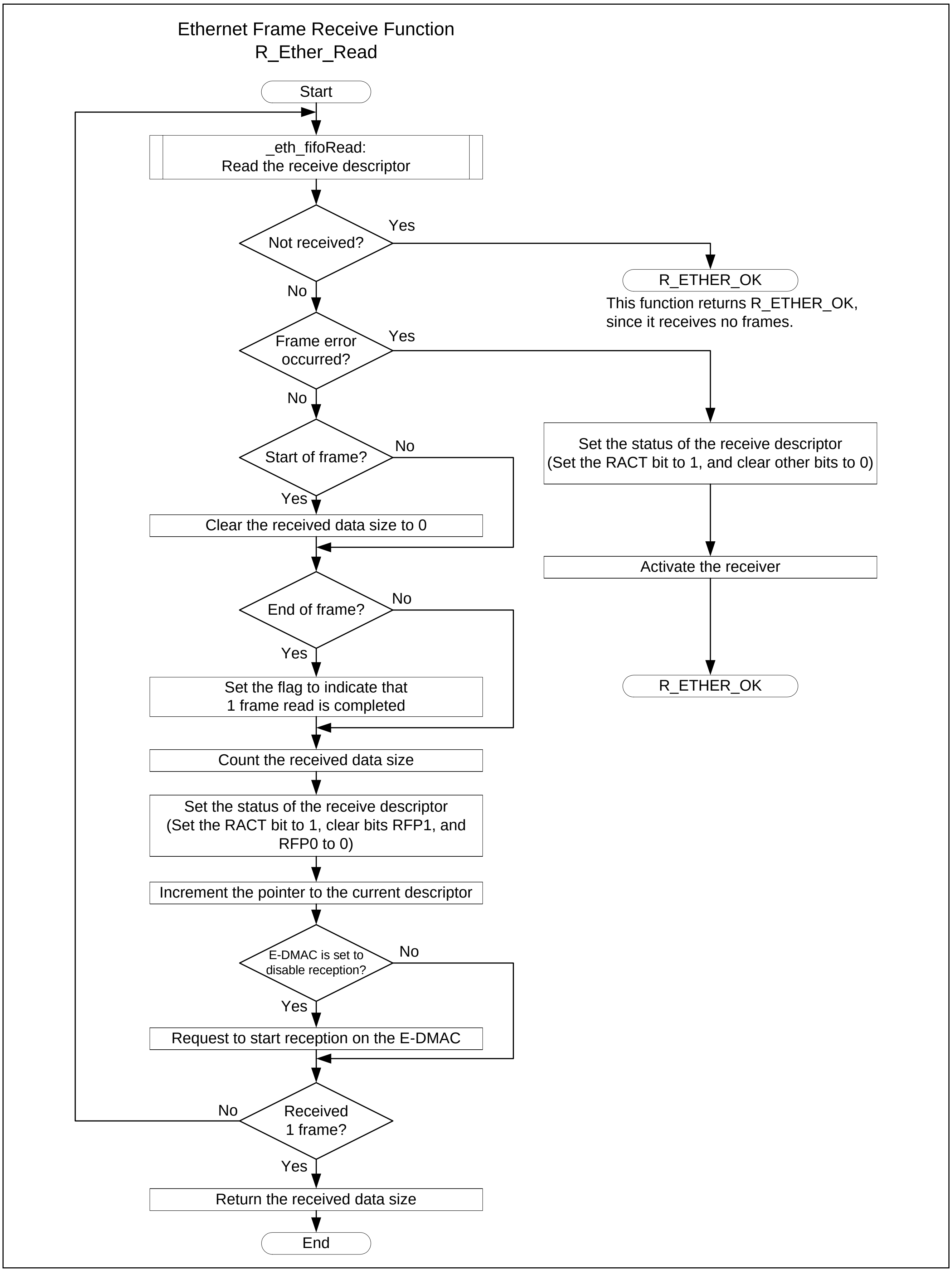

Ethernet Frame Receive Function
R_Ether_Read
Start
_eth_fifoRead:
Read the receive descriptor
Not received?
Yes
No
R_ETHER_OK
This function returns R_ETHER_OK,
since it receives no frames.
Frame error occurred?
Yes
No
Set the status of the receive descriptor
(Set the RACT bit to 1, and clear other bits to 0)
Activate the receiver
R_ETHER_OK
Start of frame?
No
Yes
Clear the received data size to 0
End of frame?
No
Yes
Set the flag to indicate that 1 frame read is completed
Count the received data size
Set the status of the receive descriptor
(Set the RACT bit to 1, clear bits RFP1, and RFP0 to 0)
Increment the pointer to the current descriptor
E-DMAC is set to disable reception?
No
Yes
Request to start reception on the E-DMAC
Received 1 frame?
No
Yes
Return the received data size
End

Figure 14 Sample Program Flow Chart (3/4)

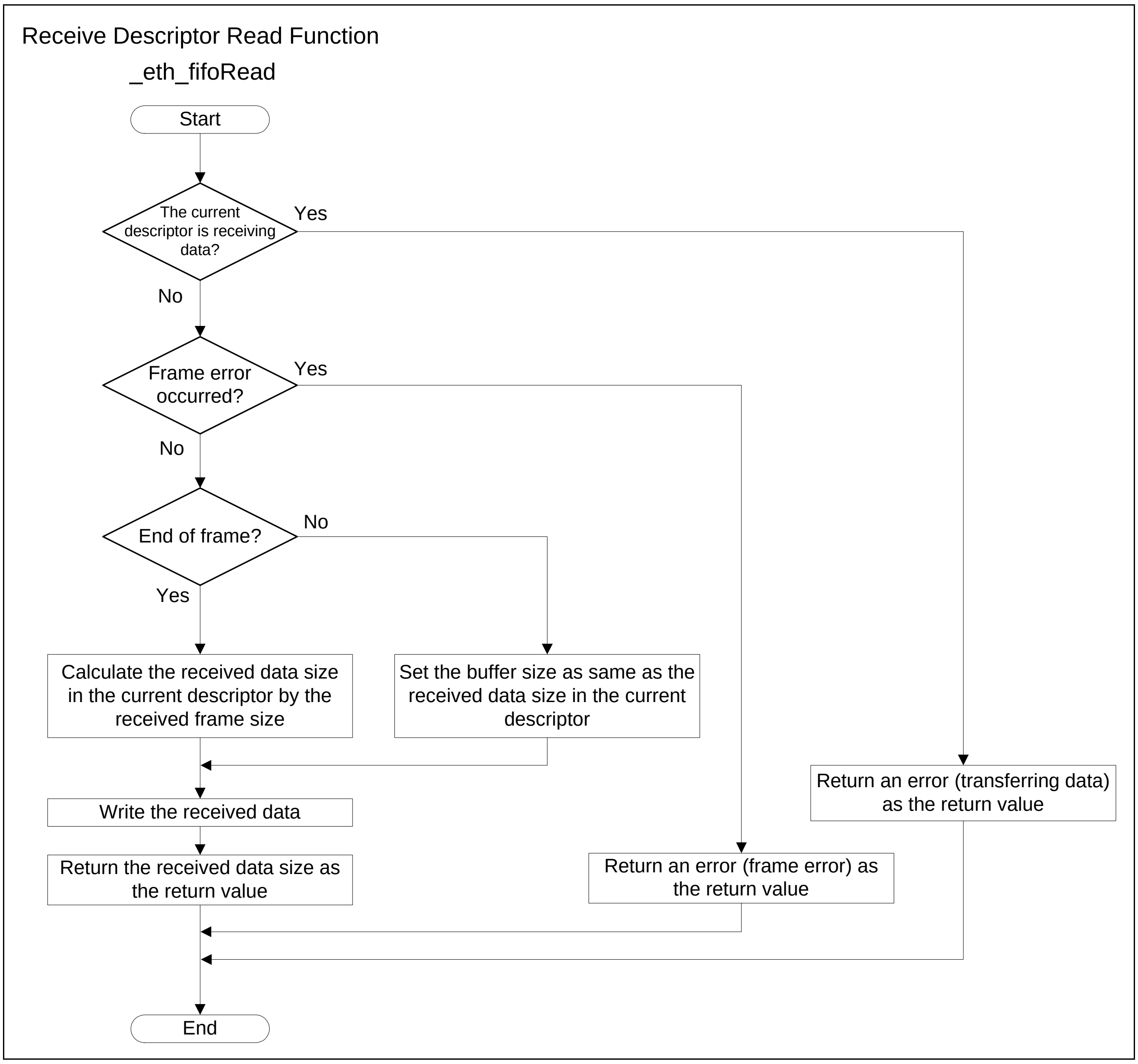

Receive Descriptor Read Function
_eth_fifoRead
Start
The current descriptor is receiving data?
Yes
No
Frame error occurred?
Yes
No
End of frame?
No
Yes
Calculate the received data size in the current descriptor by the received frame size
Set the buffer size as same as the received data size in the current descriptor
Return an error (transferring data) as the return value
Write the received data
Return the received data size as the return value
Return an error (frame error) as the return value
End

Figure 15 Sample Program Flow Chart (4/4)

3. Sample Program Listing

3.1 Sample Program Listing "main.c" (1/3)

```
/*******************************************************************************
*   DISCLAIMER
*
*   This software is supplied by Renesas Electronics Corp. and is only
*   intended for use with Renesas products.  No other uses are authorized.
*
*   This software is owned by Renesas Electronics Corp. 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 CORP. 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) 2009(2010). Renesas Electronics Corporation. All Rights Reserved.
*""FILE COMMENT""*********** Technical reference data **************************
*   System Name : SH7216 Sample Program
*   File Name   : main.c
*   Abstract    : Configuration to Receive Ethernet Frames
*   Version     : 2.00.00
*   Device      : SH7216
*   Tool-Chain  : High-performance Embedded Workshop (Ver.4.07.00).
*               : C/C++ compiler package for the SuperH RISC engine family
*               :                             (Ver.9.03 Release00).
*   OS          : None
*   H/W Platform: R0K572167 (CPU board)
*   Description : Configures the MCU for the Ethernet reception and receives
*               : Ethernet frames.
********************************************************************************
*   History     : Nov.18,2009 Ver.1.00.00
*               : Jul.23,2010 Ver.2.00.00 Comply with the Renesas API
*""FILE COMMENT END""**********************************************************/
```

3.2 Sample Program Listing "main.c" (2/3)

```
#include "iodefine.h"
#include "stdint.h"
#include "r_ether.h"
#include "phy.h"

/* ==== Prototype Declaration ==== */
void main(void);

/* ==== Variable Declaration ==== */
static uint8_t macaddr[] = {
  0x00,0x01,0x02,0x03,0x04,0x05,      /* Source MAC address (00:01:02:03:04:05) */
};

static uint8_t r_frame[10][1536];     /* Buffer to store the receive frame */

/*""FUNC COMMENT""*************************************************************
 * ID             :
 * Outline        : Sample program main
 *-----------------------------------------------------------------------------
 * Include        : "iodefine.h", "stdint.h", "r_ether.h", and "phy.h"
 *-----------------------------------------------------------------------------
 * Declaration    : void main(void);
 *-----------------------------------------------------------------------------
 * Description    : Uses the internal Ethernet Controller (EtherC) and the Ethernet
 *                : Controller Dynamic Memory Access Controller (E-DMAC) to receive
 *                : Ethernet frames. Ethernet PHY-LSI RTL8201CP (Realtek) is used
 *                : in this application.
 *                : Uses multiple receive descriptors to receive frames continuously.
 *-----------------------------------------------------------------------------
 * Argument       : void
 *-----------------------------------------------------------------------------
 * Return Value : void
 *-----------------------------------------------------------------------------
 * Note           : None
 *""FUNC COMMENT END""*********************************************************/
void main(void)
{
  int32_t            i;
  int32_t            ret;
  uint32_t           ch = 0;

```

3.3 Sample Program Listing "main.c" (3/3)

```
    /* ==== Clears the module standby on the EtherC/E-DMAC ==== */
    STB.CR4.BIT._ETHER = 0;
    /* ==== Sets the PFC (For the EtherC) ==== */
    PFC.PACRL4.BIT.PA12MD = 7; /* TX_CLK (input) */
    PFC.PACRL3.WORD = 0x7777;  /* TX_EN,MII_TXD0,MII_TXD1,MII_TXD2 (output) */
    PFC.PACRL2.BIT.PA7MD = 7;  /* MII_TXD3 (output) */
    PFC.PACRL2.BIT.PA6MD = 7;  /* TX_ER (output) */
    PFC.PDCRH4.WORD = 0x7777;  /* RX_DV,RX_ER,MII_RXD3,MII_RXD2 (input) */
    PFC.PDCRH3.WORD = 0x7777;  /* MII_RXD1,MII_RXD0,RX_CLK,CRS (input) */
    PFC.PDCRH2.WORD = 0x7777;  /* COL (input),WOL,EXOUT,MDC (input) */
    PFC.PDCRH1.BIT.PD19MD = 7; /* LINKSTA (input) */
    PFC.PDCRH1.BIT.PD18MD = 7; /* MDIO (input/output) */

    /* ==== Ethernet configuration ==== */
    ret = R_Ether_Open(ch, &macaddr[0]);

    if(R_ETHER_OK == ret){
      /* ==== Starts receiving 10 frames ==== */
      for(i = 0; i < 10; i++){
          /* ---- Receives frames ---- */
          ret = R_Ether_Read(ch, &r_frame[i][0]);
          if(ret == R_ETHER_OK){
              i--;
          }
      }
    }

    /* ==== Stops transmitting/receiving Ethernet frames ==== */
    R_Ether_Close(ch);

    while(1){
      /* sleep */
    }
  }

  /* End of file */
```

3.4 Sample Program Listing "r_ether.c" (1/10)

```
/*******************************************************************************
*   DISCLAIMER
*
*   This software is supplied by Renesas Electronics Corp. and is only
*   intended for use with Renesas products.  No other uses are authorized.
*
*   This software is owned by Renesas Electronics Corp. 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 CORP. 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) 2009(2010). Renesas Electronics Corporation. All Rights Reserved.
*""FILE COMMENT""*********** Technical reference data **************************
*   System Name : SH7216 Sample Program
*   File Name   : r_ether.c
*   Version     : 2.00.00
*   Device      : SH7216
*   Tool-Chain  : High-performance Embedded Workshop (Ver.4.07.00).
*               : C/C++ compiler package for the SuperH RISC engine family
*               :                             (Ver.9.03 Release00).
*   OS          : None
*   H/W Platform: R0K572167 (CPU board)
*   Description : Ethernet module device driver
********************************************************************************
*   History     : Jun.10.2009 Ver.1.00.00
*               : Jul.23,2010 Ver.2.00.00 Comply with the Renesas API
*""FILE COMMENT END""**********************************************************/
#include <machine.h>
#include <string.h>
#include "iodefine.h"
#include "stdint.h"
#include "r_ether.h"
#include "phy.h"

/* ==== Prototype Declaration ==== */
void    _eth_fifoInit(ethfifo p[], uint32_t status);
int32_t _eth_fifoWrite(ethfifo *p, int8_t buf[], int32_t size);
int32_t _eth_fifoRead(ethfifo *p, int8_t buf[]);

```

3.5 Sample Program Listing "r_ether.c" (2/10)

```
#pragma section  _RX_DESC
volatile ethfifo rxDesc[ENTRY];          /* Receive descriptor */
#pragma section  _TX_DESC
volatile ethfifo txDesc[ENTRY];          /* Transmit descriptor */
#pragma section

#pragma section  _RX_BUFF
int8_t rxbuf[ENTRY][BUFSIZE];        /* Receive data buffer */
#pragma section  _TX_BUFF
int8_t txbuf[ENTRY][BUFSIZE];        /* Transmit data buffer */
#pragma section

/* ==== Initializes the Ethernet device driver control structure ==== */
struct ei_device  le0 =
{
  "eth0",      /* device name */
  0,           /* open */
  0,           /* Tx_act */
  0,           /* Rx_act */
  0,           /* txing */
  0,           /* irq lock */
  0,           /* dmaing */
  0,           /* current receive descriptor */
  0,           /* current transmit descriptor */
  0,           /* save irq */
  {
    0,      /* rx packets */
    0,      /* tx packets */
    0,      /* rx errors */
    0,      /* tx errors */
    0,      /* rx dropped */
    0,      /* tx dropped */
    0,      /* multicast */
    0,      /* collisions */

    0,      /* rx length errors */
    0,      /* rx over errors */
    0,      /* rx CRC errors */
    0,      /* rx frame errors */
    0,      /* rx fifo errors */
    0,      /* rx missed errors */

    0,      /* tx aborted errors */
    0,      /* tx carrier errors */
    0,      /* tx fifo errors */
    0,      /* tx heartbeat errors */
    0       /* tx window errors */
  },
  0,           /* MAC 0 */
  0,           /* MAC 1 */
  0,           /* MAC 2 */
  0,           /* MAC 3 */
  0,           /* MAC 4 */
  0        /* MAC 5 */
};
```

3.6 Sample Program Listing "r_ether.c" (3/10)

```

/*""FUNC COMMENT""*****************************************************************
 * ID            :
 * Outline       : Ethernet open
 *-------------------------------------------------------------------------------
 * Include       : "iodefine.h" , "phy.h", "r_ether.h" and "stdint.h"
 *-------------------------------------------------------------------------------
 * Declaration   : int32_t R_Ether_Open(uint32_t ch, uint8_t mac_addr[]);
 *-------------------------------------------------------------------------------
 * Description   : Initializes the EtherC, E-DMAC, PHY, and buffer memory.
 *               : Initializes the MCU for the Ethernet and enables the MCU to
 *               : transmit and receive Ethernet frames.
 *               : When failed to initialize, it returns an error.
 *-------------------------------------------------------------------------------
 * Argument      : uint32_t ch;        I : Ethernet channel number
 *               : uint8_t mac_addr[]; I : MAC address of such Ethernet channel
 *-------------------------------------------------------------------------------
 * Return Value  : R_ETHER_OK;    Succeeded to initialize
 *               : R_ETHER_ERROR; Failed to initialize
 *-------------------------------------------------------------------------------
 * Note          : None
 *""FUNC COMMENT END""**************************************************************/
int32_t R_Ether_Open(uint32_t ch, uint8_t mac_addr[])
{
  int32_t  i;
  uint32_t mac;
  uint16_t phydata;

  ch = ch;                            /* Avoids the warning */

  /* ==== Configures the Ethernet device driver ==== */
  le0.open = 1;
  /* ==== Sets the descriptor ==== */
  _eth_fifoInit(rxDesc, (uint32_t)ACT);
  _eth_fifoInit(txDesc, (uint32_t)0);
  le0.rxcurrent = &rxDesc[0];
  le0.txcurrent = &txDesc[0];
  /* ==== Sets the MAC address ==== */
  le0.mac_addr[0] = mac_addr[0];
  le0.mac_addr[1] = mac_addr[1];
  le0.mac_addr[2] = mac_addr[2];
  le0.mac_addr[3] = mac_addr[3];
  le0.mac_addr[4] = mac_addr[4];
  le0.mac_addr[5] = mac_addr[5];

  /* ==== Initializes the E-DMAC/EtherC ==== */
  EDMAC.EDMR.BIT.SWR = 1;                 /* Enables the software reset */
  for( i = 0 ; i < 0x00000100 ; i++ );    /* Waits until the E-DMAC/EtherC are initialized */
                                      /* (B clock: 64 cycles) */
  EDMAC.EDMR.LONG    = 0x00000000;        /* Sets the E-DMAC mode register */
                                      /* (Big endian mode) */
                                      /* (Transmit/receive descriptor length: 16 bytes) */
```

3.7 Sample Program Listing "r_ether.c" (4/10)

```
    /* ==== Initializes the EtherC ==== */
    EtherC.ECMR.LONG   = 0x00000000;       /* Sets the EtherC mode register */
                                       /* (Sets the duplex mode as half-duplex) */
                                       /* (Sets promiscuous mode as normal operation) */

    EtherC.ECSR.LONG   = 0x00000037;       /* Clears all of the EtherC status */
                                       /* (BFR, PSRTO, LCHNG, MPD, ICD) */
    EtherC.ECSIPR.LONG = 0x00000020;       /* Disables the EtherC interrupt */
         /* bit31~6 : Reserve : 0 ----- Reserved bits */
         /* bit5    : BFSIPR  : 1 ----- Disables the continuous broadcast frame */
         /*                             reception interrupt */
         /* bit4    : PSRTOIP : 0 ----- Disables the PAUSE frame retransmit retry over */
         /* bit3    : Reserve : 0 ----- Reserved bit */
         /* bit2    : LCHNGIP : 0 ----- Disables the link signal change interrupt */
         /* bit1    : MPDIP   : 0 ----- Disables the Magic Packet detection interrupt */
         /* bit0    : ICDIP   : 0 ----- Disables the illegal carrier detection interrupt */

    EtherC.RFLR.LONG   = 1518;             /* Sets the maximum receive frame length */
    EtherC.IPGR.LONG   = 0x00000014;       /* Sets the gap between packets (96-bit time) */

    /* ==== Sets the MAC address ==== */
    mac = ((uint32_t)mac_addr[0] << 24) |
    ((uint32_t)mac_addr[1] << 16) |
    ((uint32_t)mac_addr[2] << 8 ) |
    (uint32_t)mac_addr[3];
    EtherC.MAHR = mac;

    mac = ((uint32_t)mac_addr[4] << 8 ) |
    (uint32_t)mac_addr[5];
    EtherC.MALR.LONG = mac;

    /* ==== Initializes the E-DMAC ==== */
    EDMAC.EESR.LONG   = 0x47FF0F9F;        /* Initializes the EtherC/E-DMAC status register */
    EDMAC.EESIPR.LONG = 0x00000000;        /* Initializes the EtherC/E-DMAC status */
                                       /* interrupt enable register */
    EDMAC.RDLAR       = le0.rxcurrent;     /* Sets the receive descriptor start address */
    EDMAC.TDLAR       = le0.txcurrent;     /* Sets the transmit descriptor start address */
    EDMAC.TRSCER.LONG = 0x00000000;        /* Brings the EtherC/E-DMAC status register */
                                       /* value to the descriptor */
    EDMAC.TFTR.LONG   = 0x00000000;        /* Sets store and forward mode */
    EDMAC.FDR.LONG    = 0x00000000;        /* Sets the transmit/receive FIFO capacity */
                                       /* (256 bytes) */
    EDMAC.RMCR.LONG   = 0x00000001;        /* Sets to receive data continuously other */
                                       /* than the receive descriptor is empty */

    /* ==== Initializes the PHY ==== */
    phydata = phy_init();

    if(phydata == R_PHY_ERROR){
      return R_ETHER_ERROR;
    }

```

3.8 Sample Program Listing "r_ether.c" (5/10)

```
    /* ==== Starts the PHY auto-negotiation ==== */
    phydata = phy_set_autonegotiate();
    /* ---- Determines whether to auto-negotiate or not ---- */
    if(phydata == R_PHY_ERROR){            /* Failed to auto-negotiate */
      return R_ETHER_ERROR;
    }

    /* ---- Detects the performance of the link partner ---- */
    if(phydata & 0x0100){                  /* Detects PHY-LSI register 0 */
                                       /* bit8 : DuplexMode : 1 ---- Supports */
                                       /*                      full-duplex mode */
      EtherC.ECMR.BIT.DM = 1;              /* Full-duplex communication */
    }

    /* ==== Enables the EtherC transmission/reception ==== */
    EtherC.ECMR.BIT.RE = 1;
    EtherC.ECMR.BIT.TE = 1;

    /* ==== Enables the E-DMAC reception ==== */
    EDMAC.EDRRR.LONG = 0x00000001;

    return R_ETHER_OK;
  }

  /*""FUNC COMMENT""*************************************************************
   * ID           :
   * Outline      : Ethernet close
   *---------------------------------------------------------------------------
   * Include      : "iodefine.h" , "r_ether.h" and "stdint.h"
   *---------------------------------------------------------------------------
   * Declaration  : int32_t R_Ether_Close(uint32_t ch);
   *---------------------------------------------------------------------------
   * Description  : Stops the EtherC/E-DMAC.
   *---------------------------------------------------------------------------
   * Argument     : uint32_t ch; I : Ethernet channel number
   *---------------------------------------------------------------------------
   * Return Value : R_ETHER_OK; Disables the EtherC transmission/reception
   *---------------------------------------------------------------------------
   * Note         : None
   *""FUNC COMMENT END""*********************************************************/
  int32_t R_Ether_Close(uint32_t ch)
  {
    ch = ch;                           /* Avoids the warning */

    le0.open = 0;
    EtherC.ECMR.LONG = 0x00000000;         /* Disables the EtherC transmission/reception */
    le0.irqlock = 1;

    return R_ETHER_OK;
  }
  (omitted)

```

3.9 Sample Program Listing "r_ether.c" (6/10)

```
/*""FUNC COMMENT""**************************************************************
 * ID             :
 * Outline        : Receive frames
 *-----------------------------------------------------------------------------
 * Include        : "iodefine.h" , "r_ether.h" and "stdint.h"
 *-----------------------------------------------------------------------------
 * Declaration    : int32_t R_Ether_Read(uint32_t ch, void *buf);
 *-----------------------------------------------------------------------------
 * Description    : Copies the specified frame from the buffer registered in the
 *                : receive descriptor and receives the frame.
 *                : Detects the data not received, and error occurred.
 *-----------------------------------------------------------------------------
 * Argument       : uint32_t ch; I : Ethernet channel number
 *                : void *buf;   I : Pointer to the area to store the receive data
 *-----------------------------------------------------------------------------
 * Return Value   : R_ETHER_OK; No data received
 *                : 1 or bigger; Receive data size
 *-----------------------------------------------------------------------------
 * Note           : None
 *""FUNC COMMENT END""**********************************************************/
int32_t R_Ether_Read(uint32_t ch, void *buf)
{
  int32_t  receivesize = 0;           /* Receive data size */
  int32_t  recvd;                     /* Receive data size or receive descriptor status */
  int32_t  flag = 1;                  /* 1 frame reception incomplete flag */
                                  /* (1: 1 frame reception incomplete) */
  uint8_t  readcount = 0;             /* Number of times to read the receive */
                                  /* descriptor until completing to receive 1 frame */
  int8_t   *data = (int8_t *)buf;     /* Pointer to the area to store the receive data */

  ch = ch;                            /* Avoids the warning */

  /* ==== Receives 1 frame ==== */
  while (flag){
    recvd = _eth_fifoRead(le0.rxcurrent, data);
    readcount++;
    /* ---- No data received ---- */
    if (readcount >= 2 && receivesize == 0){
        return R_ETHER_OK;              /* No data received */
    }

    /* ---- No received data ---- */
    if (recvd == -1){
    }
    /* ---- Receive frame error ---- */
    else if (recvd == -2){
        le0.stat.rx_errors++;

```

3.10 Sample Program Listing "r_ether.c" (7/10)

```
        /* ---- Sets the receive descriptor to transmit again and exit ---- */
        receivesize = 0;                 /* to set the return value as "R_ETHER_OK" */
        le0.rxcurrent->status &= ~(FP1 | FP0 | FE);
        le0.rxcurrent->status &= ~(RMAF | RRF | RTLF | RTSF | PRE | CERF);
        le0.rxcurrent->status |= ACT;
        le0.rxcurrent = le0.rxcurrent->next;

        /* ---- Starts receiving frame ---- */
        if (EDMAC.EDRRR.LONG == 0x00000000L){
            EDMAC.EDRRR.LONG = 0x00000001L;
        }
    }
    /* ---- With the received data ---- */
    else{
        /* ---- Receives the start of the frame ---- */
        if ((le0.rxcurrent->status & FP1) == FP1){
            receivesize = 0;
        }
        /* ---- Receives the end of the frame (receiving the frame is completed) ---- */
        if ((le0.rxcurrent->status & FP0) == FP0){
            le0.stat.rx_packets++;   /* Counts the total number of receive frames */
            flag = 0;                /* Sets 1 frame receive incomplete flag */
                                    /* (0: receiving 1 frame is completed) */
        }

        /* ---- Counts the received data as the frame size ---- */
        receivesize += recvd;
        /* ---- Sets the receive descriptor to continue receiving data again ---- */
        le0.rxcurrent->status &= ~(FP1 | FP0);
        le0.rxcurrent->status |= ACT;
        le0.rxcurrent = le0.rxcurrent->next;
        /* ---- Updates the receive data storing area ---- */
        data += recvd;

        /* ==== Determines the E-DMAC receive request ==== */
        if (EDMAC.EDRRR.LONG == 0x00000000L){
            EDMAC.EDRRR.LONG = 0x00000001L;
        }
    }
  }

  return (int32_t)receivesize;
}

```

3.11 Sample Program Listing "r_ether.c" (8/10)

```
/*""FUNC COMMENT""************************************************************
 * ID             :
 * Outline        : Initialize the FIFO
 *----------------------------------------------------------------------------
 * Include        :
 *----------------------------------------------------------------------------
 * Declaration  : void _eth_fifoInit( ethfifo p[], uint32_t status );
 *----------------------------------------------------------------------------
 * Description  : Initializes the E-DMAC descriptor
 *----------------------------------------------------------------------------
 * Argument     : ethfifo p[];     O : Pointer to the descriptor
 *              : uint32_t status; I : Descriptor default status
 *----------------------------------------------------------------------------
 * Return Value : void
 *----------------------------------------------------------------------------
 * Note         : None
 *""FUNC COMMENT END""*********************************************************/
void _eth_fifoInit( ethfifo p[], uint32_t status )
{
  ethfifo  *current = 0;
  int32_t i, j;

  for( i = 0 ; i < ENTRY ; i++ ){
    current = &p[i];
    /* ==== Detects the descriptor status ==== */
    if( status == 0 ){
        current->buf_p = &txbuf[i][0];/* Determines to transmit when the ACT bit is 0 */
    }
    else{
        current->buf_p = &rxbuf[i][0];/* Determines to receive when the ACT bit is 1 */
    }

    /* ==== Clears the buffer ==== */
    for( j = 0 ; j < BUFSIZE ; j++ ){
        current->buf_p[j] = 0;
    }

    current->bufsize = BUFSIZE;
    current->size = 0;
    current->status = status;
    current->next = &p[i+1];
  }
  /* ==== Waits until the last FIFO entry is completed ==== */
  current->status |= DL;  /* Sets the current descriptor as the end of the descriptor ring
*/
  current->next = &p[0];
}
(omitted)
```

3.12 Sample Program Listing "r_ether.c" (9/10)

```
/*""FUNC COMMENT""***************************************************************
 * ID             :
 * Outline        : Read the receive descriptor
 *-------------------------------------------------------------------------------
 * Include        :
 *-------------------------------------------------------------------------------
 * Declaration    : int32_t _eth_fifoRead( ethfifo *p, int8_t buf[]);
 *-------------------------------------------------------------------------------
 * Description    : Reads the data from the receive descriptor to the area specified
 *                : by the argument.
 *-------------------------------------------------------------------------------
 * Argument       : ethfifo *p;    O : Pointer to the current descriptor
 *                : int8_t buf[]; O : Pointer to the area to store the receive data
 *-------------------------------------------------------------------------------
 * Return Value   : 0 or bigger; Retrieved data size
 *                : -1;        The current descriptor is receiving data or preparing
 *                :            for reception
 *                : -2;        Receive frame error occurred
 *-------------------------------------------------------------------------------
 * Note           : None
 *""FUNC COMMENT END""**********************************************************/
int32_t _eth_fifoRead( ethfifo *p, int8_t buf[])
{
  int32_t i, temp_size;                  /* Buffer size counter (bytes) */
  ethfifo  *current = p;

  /* ==== The current descriptor is receiving data or preparing for reception ==== */
  if( (current->status & ACT) != 0){
    return (-1);                        /* This is not an error */
  }

  /* ==== Receive frame error ==== */
  else if( (current->status & FE) != 0){
    return (-2);                        /* Descriptor must be updated */
  }

```

3.13 Sample Program Listing "r_ether.c" (10/10)

```
        /* ==== Normal reception ==== */
        else{
          /* ---- At the end of the frame (receiving 1 frame is completed) ---- */
          if ((current->status & FP0) == FP0){
              /* ---- Calculates the receive data size of the current descriptor ---- */
              temp_size = current->size;      /* Total number of bytes of the frame */

              while (temp_size > BUFSIZE){
                  temp_size -= BUFSIZE;       /* Calculates the data size received */
                                            /* at the end of the frame */
              }
          }
          /* ---- Not at the end of the frame ---- */
          else{
              temp_size = BUFSIZE;
          }

          /* ---- Copies data from the receive descriptor to receive buffer ---- */
          for (i = 0; i < temp_size; i++){
              buf[i] = current->buf_p[i];
          }

          return (temp_size);                   /* Receive data size */
        }
      }

      /* End of File */
```

3.14 Sample Program Listing "r_ether.h" (1/3)

```
/*******************************************************************************
*   DISCLAIMER
*
*   This software is supplied by Renesas Electronics Corp. and is only
*   intended for use with Renesas products.  No other uses are authorized.
*
*   This software is owned by Renesas Electronics Corp. 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 CORP. 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) 2009(2010). Renesas Electronics Corporation. All Rights Reserved.
*""FILE COMMENT""*********** Technical reference data **************************
*   System Name : SH7216 Sample Program
*   File Name   : r_ether.h
*   Version     : 2.00.00
*   Device      : SH7216
*   Tool-Chain  : High-performance Embedded Workshop (Ver.4.07.00).
*               : C/C++ compiler package for the SuperH RISC engine family
*               :                             (Ver.9.03 Release00).
*   OS          : None
*   H/W Platform: R0K572167 (CPU board)
*   Description : Ethernet module device driver
********************************************************************************
*   History     : Jun.10.2009 Ver.1.00.00
*               : Jul.23,2010 Ver.2.00.00 Comply with the Renesas API
*""FILE COMMENT END""**********************************************************/
#ifndef ETH_H
#define ETH_H

```

3.15 Sample Program Listing "r_ether.h" (2/3)

```
/* ==== Type definition ==== */
typedef struct Descriptor
{
  uint32_t     status;
  uint16_t     bufsize;
  uint16_t     size;
  int8_t           *buf_p;
  struct Descriptor *next;
} ethfifo;

/* ==== Macro definition ==== */
#define BUFSIZE   256
#define ENTRY     8

#define  ACT      0x80000000
#define  DL       0x40000000
#define  FP1      0x20000000
#define  FP0      0x10000000
#define  FE       0x08000000

#define  RFOVER   0x00000200
#define  RMAF     0x00000080
#define  RRF      0x00000010
#define  RTLF     0x00000008
#define  RTSF     0x00000004
#define  PRE      0x00000002
#define  CERF     0x00000001

#define  ITF      0x00000010
#define  CND      0x00000008
#define  DLC      0x00000004
#define  CD       0x00000002
#define  TRO      0x00000001

/* ==== Renesas Ethernet API return defines ==== */
#define R_ETHER_OK          0
#define R_ETHER_ERROR       -1
#define R_ETHER_HARD_ERROR  -3
#define R_ETHER_RECOVERABLE -4
#define R_ETHER_NO_DATA     -5

/* ==== Prototype Declaration ==== */
/* ==== Renesas Ethernet API prototypes ==== */
int32_t R_Ether_Open(uint32_t ch, uint8_t mac_addr[]);
int32_t R_Ether_Close(uint32_t ch);
int32_t R_Ether_Write(uint32_t ch, void *buf, uint32_t len);
int32_t R_Ether_Read(uint32_t ch, void *buf);

```

3.16 Sample Program Listing "r_ether.h" (3/3)

```
/* ==== Ethernet collected data ==== */
struct enet_stats
{
  uint32_t rx_packets;
  uint32_t tx_packets;
  uint32_t rx_errors;
  uint32_t tx_errors;
  uint32_t rx_dropped;
  uint32_t tx_dropped;
  uint32_t multicast;
  uint32_t collisions;

  /* ---- Receive error ---- */
  uint32_t rx_length_errors;
  uint32_t rx_over_errors;
  uint32_t rx_crc_errors;
  uint32_t rx_frame_errors;
  uint32_t rx_fifo_errors;
  uint32_t rx_missed_errors;

  /* ---- Transmit error ---- */
  uint32_t tx_aborted_errors;
  uint32_t tx_carrier_errors;
  uint32_t tx_fifo_errors;
  uint32_t tx_heartbeat_errors;
  uint32_t tx_window_errors;
};

struct ei_device
{
  const int8_t *name;     /* Device name */
  uint8_t      open;
  uint8_t      Tx_act;
  uint8_t      Rx_act;
  uint8_t      txing;
  uint8_t      irqlock;
  uint8_t      dmaing;
  ethfifo      *rxcurrent;    /* Receive current descriptor */
  ethfifo      *txcurrent;    /* Transmit current descriptor */
  uint8_t      save_irq;
  struct enet_stats stat;     /* Ethernet collected data */
  uint8_t      mac_addr[6];   /* MAC address storage area */
};

#endif /* ETH_H  */
```

4. References

- Software Manual
  SH-2A, SH-2A FPU Software Manual Rev. 3.00
  The latest version of the software manual can be downloaded from the Renesas Electronics website.

- Hardware Manual
  SH7214 Group, SH7216 Group Hardware User’s Manual Rev. 2.00
  The latest version of the hardware user’s manual can be downloaded from the Renesas Electronics website.

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	Feb.12.10	—	First edition issued
2.00	Sep.17.10	All pages	Updated to comply with the Renesas API

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.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan, R.O.C.
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.0