V850E2/MN4

CPU-Integrated IEEE802.3 Ethernet Controller Sample Driver

R01AN0012EJ0100
Rev.1.00
Jan 31, 2012

Introduction

This application note explains how to set up the 10/100Mbps Ethernet controller included on the V850E2/MN4 and also gives an outline of the operation and describes the procedure for using sample program.

Target Device

V850E2/MN4 Microcontrollers

Contents

1. Ethernet Controller Sample Driver

1.1 Overview

This sample driver is used to operate the Ethernet controller included on the V850E2/MN4. The higher-level TCP/IP stack is a compact TCP/IP stack (C-NET) from Renesas Electronics.

The sample project including this sample driver consists of the following Ethernet driver and higher-level applications (TCP/IP stack and network applications) which use this sample driver:

(1) Ethernet driver

The Ethernet driver has a limited set of functions responsible for controlling Ethernet communication.
The Ethernet driver mainly includes the functional blocks below.

(a) Initialization processing block

Configures the initial parameter settings of the Ethernet controller required for using the Ethernet controller on the V850E2/MN4, the initial settings for data allocated in the H-bus shared memory space, and the CPU's built-in function settings for interrupt processing.

(b) Transmission processing block

Manages/controls the send descriptors and send data buffers. Passes packet data requested by an application to the Ethernet controller via the dedicated DMA controller.

(c) Reception processing block

Manages/controls the receive descriptors and receive data buffers. Passes packet data received by the Ethernet controller to a higher-level application via the dedicated DMA controller.

(d) Interrupt processing block

Hooks various interrupt signals from the Ethernet controller and analyzes the register which indicates the source of each interrupt signal.

(2) Compact TCP/IP stack (C-NET) and sample software

The compact TCP/IP stack (C-NET) from Renesas Electronics is installed to implement sample programs for web servers.

1.1.1 Internal Configuration

The figure below shows the internal configuration including this sample driver and sample applications.

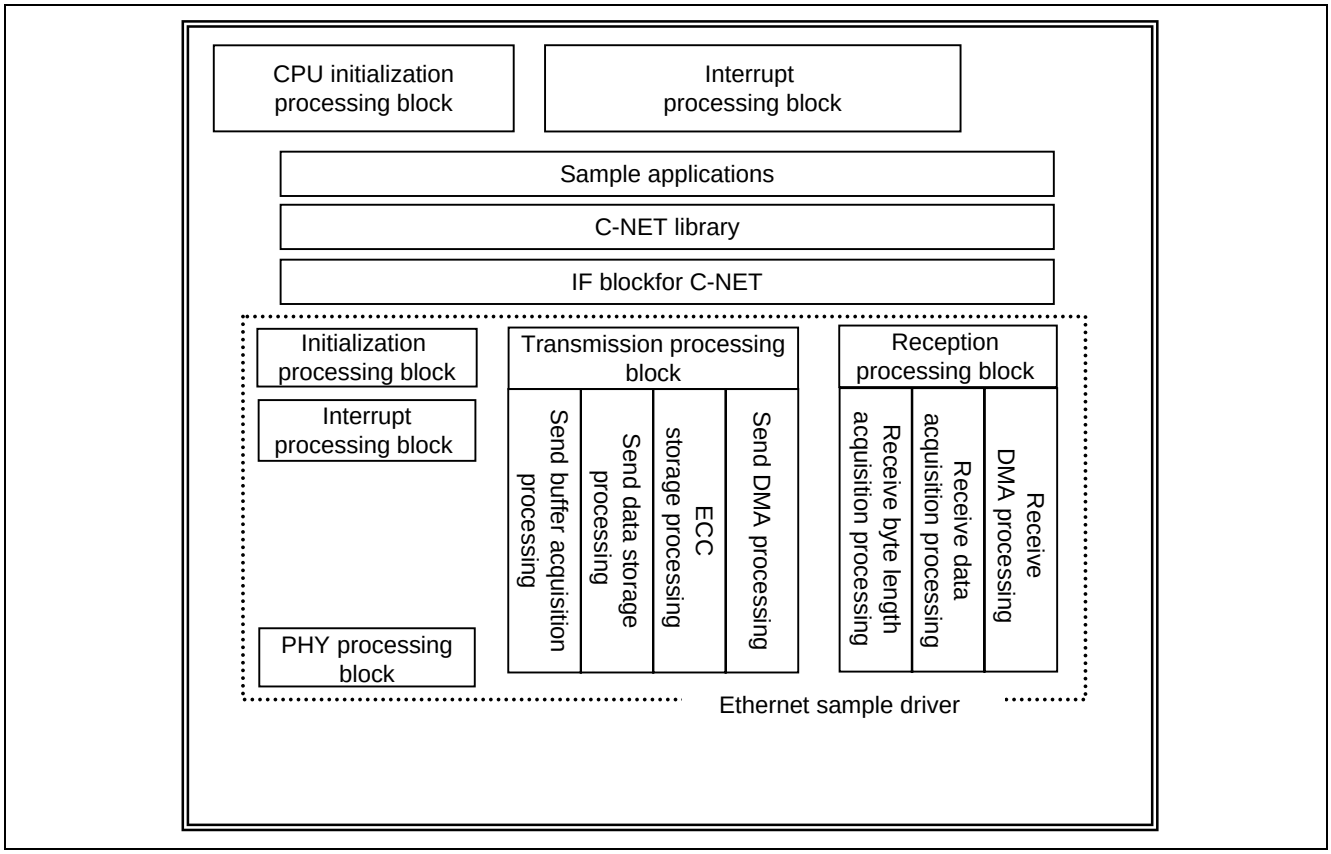


Figure 1.1 Internal Configuration

1.2 Configuration

1.2.1 Device Configuration

This sample driver uses the following evaluation board as a hardware operation environment:

- RTE-V850E2/MN4-EB-S board (by Midas lab Inc.) equipped with the V850E2/MN4

Remarks: For details about the RTE-V850E2/MN4-EB-S board, refer to the URL below.

http://www.midas.co.jp/ download/english/manual.htm

The figure below shows the device configuration for executing this sample driver.

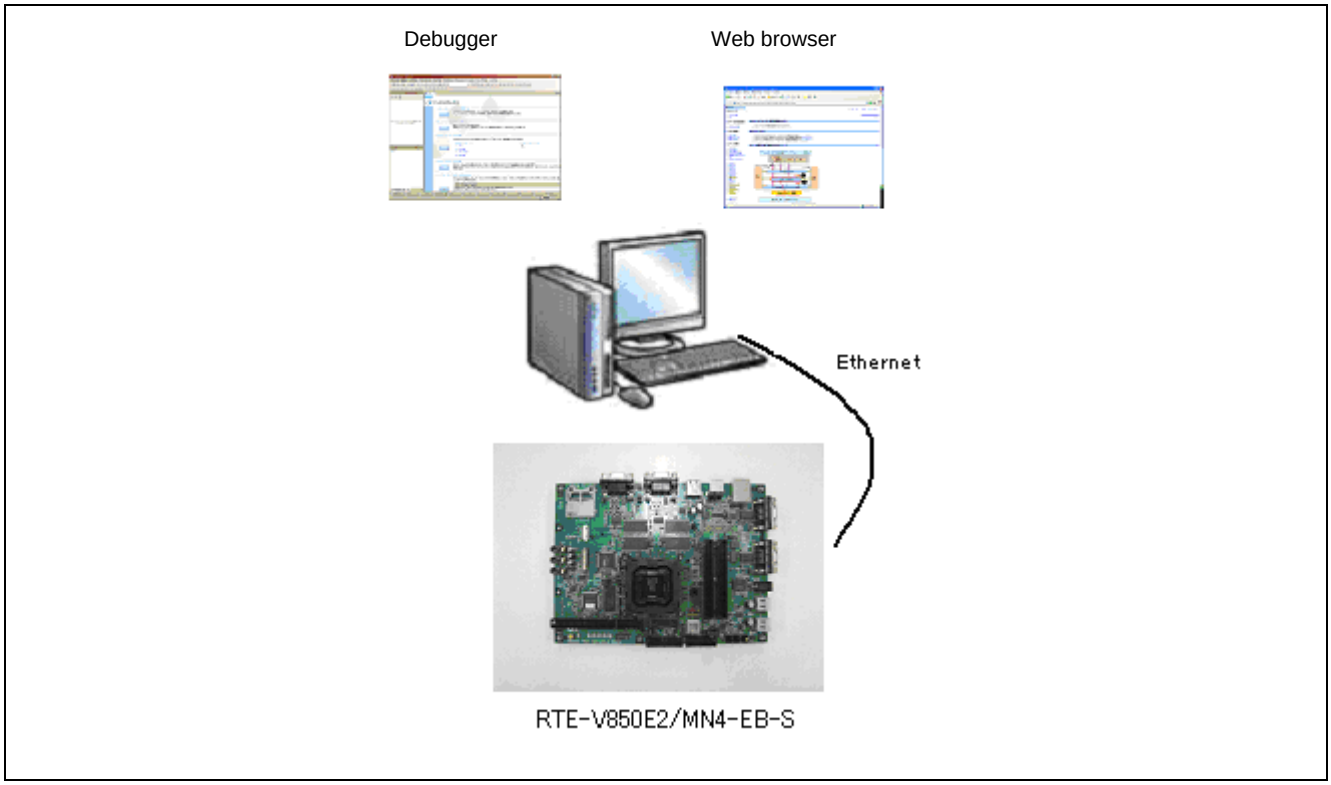


Figure 1.2 Device Configuration

1.2.2 Software Configuration

(1) **File Configuration**

The list below shows the file configuration of this sample driver.

List 1.1 File Configuration

```
CNET_V2 (Top folder)
├─ Lib (C-NET library storage folder)
└─ Program Files (Program file folder)
   └─ Renesas Electronics Tools
      ├─ bin (Web file registration tool (GMakefsdata.exe) storage folder)
      ├─ inc850 (C-NET header file storage folder)
      ├─ lib850e (C-NET library file storage folder)
      └─ smp850e (Sample program storage folder)
         └─ cnet
            └─ V850E2MN4_EvBoard (Target folder)
               ├─ fs (Web content file storage folder)
               ├─ include (Header file storage folder)
               │  └─ ether100 (Include file storage folder for sample driver)
               │     ├─ ether_driver.h (Include file for sample driver)
               │     ├─ ether_register.h (CPU-integrated Ethernet controller bit definition file)
               │     ├─ types.h (Type definition file)
               │     └─ debug.h (Include file for evaluation functions)
               ├─ Osless (Project file storage folder)
               │  └─ CNET (CubeSuite project folder)
               └─ src (Appliction source file storage folder)
                  ├─ fs (Web contents converted source storage folder)
                  ├─ httpd (Function definition file storage folder for HTTP server)
                  ├─ OSless (Sample application source file storage folder)
                  └─ driver (Driver-related storage folder)
                     └─ ether100 (Source file storage folder for sample driver)
                        ├─ ether_driver.c (Ethernet driver file)
                        ├─ eth_mem.s (Memory copy processing file (assembler code))
                        └─ ether_phy.c (PHY driver sample)
```

1.3 Specifications

1.3.1 Ethernet Controller Register Address Mapping

The Ethernet controller integrated on the V850E2/MN4 has registers which are divided into functional blocks, such as, MAC, FIFO controller, and dedicated DMA controllers.

The figure below shows the registers of the Ethernet controller integrated on the V850E2/MN4 mapped to addresses.

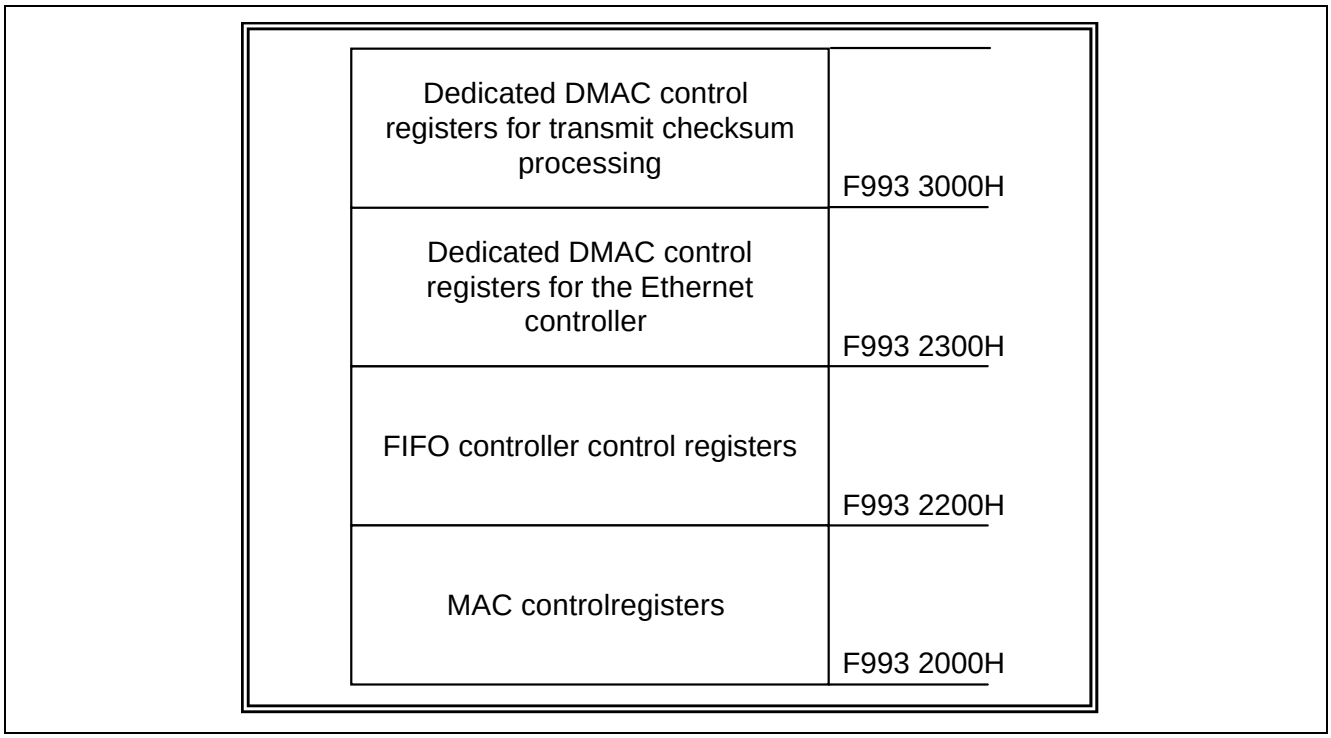


Figure 1.3 Registers Mapped to Addresses

(1) MAC control registers

The MAC control registers are mapped to addresses starting from F993 2000H.

Table 1.1 Addresses of the MAC Control Registers

Address	SFR Name	Register name	R/W
F993 2000H	MACC1	MAC setting register 1	R/W
F993 2004H	MACC2	MAC setting register 2	R/W
F993 2008H	IPGT	Back-to-back IPG register	R/W
.	.	.	.
.	.	.	.
.	.	.	.

(2) FIFO controller control registers

The FIFO controller control registers are mapped to addresses starting from F993 2200H.

Table 1.2 Addresses of the FIFO Controller Control Registers

Address	SFR name	Register name	R/W
F993 2200H	MFFCONT	FIFO controller control register	R/W
F993 2204H	RSTCNT	Software reset control register 2	R/W
F993 2218H	FLOWTHRESH	Flow control threshold register	R/W
.	.	.	.
.	.	.	.
.	.	.	.

(3) Dedicated DMAC control registers for the Ethernet controller

The dedicated DMAC control registers for the Ethernet controller are mapped to addresses starting from F993 2300H.

Table 1.3 Addresses of the Dedicated DMAC Control Registers for the Ethernet Controller

Address	SFR name	Register name	R/W
F993 2300H	ETHMODE	Core function setting register	R/W
F993 2304H	INTMS	Interrupt register	R/W
F993 2308H	TRANSCTL	Transmit control register	R/W
.	.	.	.
.	.	.	.
.	.	.	.

(4) Dedicated DMAC control registers for transmit checksum processing

The DMAC control registers dedicated to transmit checksum processing are mapped to addresses starting from F993 3000H.

Table 1.4 Addresses of the Dedicated DMAC Control Registers for Transmit Checksum Processing

Address	SFR name	Register name	R/W
F993 3000H	ETHA0CMODE	Transmit checksum unit function setting register	R/W
F993 3004H	ETHA0CINTMS	Transmit checksum interrupt register	R/W
F993 3008H	ETHA0CTRANSCT	Transmit checksum transmit control register	R/W
.	.	.	.
.	.	.	.
.	.	.	.

Note: This sample driver does not use the DMAC control registers dedicated to transmit checksum processing.

1.3.2 Descriptors

The V850E2/MN4 contains a descriptor mechanism in case send/receive data is stored in nonconsecutive data buffer memory spaces.

The Ethernet controller uses the following three types of descriptors:

- Buffer descriptors
- Link pointers
- End-of-chain descriptors

Each descriptor consists of two words (64 bits). The high-order word indicates the address of the data buffer pointed to by the descriptor. The low-order word contains descriptor control bits.

Remarks: For details about the descriptors, refer to the “V850E2/MN4 User’s Manual, Hardware (R01UH0011EJ).”

1.3.3 Data Buffers

A data buffers refers to a memory space which holds send/receive data and is pointed to by the buffer descriptor.

Data buffers can start at a byte-aligned address.

A DMA of the Ethernet controller does not require that the address boundary be considered when setting the start address/transfer byte count for the data buffer.

1.3.4 Restrictions on Descriptor/Data Buffer Allocation

The following restrictions apply to the allocation of the descriptors and data buffers for the V850E2/MN4:

- The buffer descriptors should be allocated in word-aligned consecutive memory regions.
  However, the descriptors can be stored in nonconsecutive memory spaces by linking them with a link pointer.

- The descriptors and send/receive data are referenced or transferred by the dedicated DMA controller for the Ethernet controller. However, this dedicated DMA controller cannot directly access the CPU internal memory space.
  Accordingly, the descriptors and data buffers should always be allocated in an H-bus shared memory space.

1.3.5 Examples of Packet Transmission

Ethernet transmission involves using the send descriptors (buffer descriptor, link pointer, and end-of-chain) and data buffers which hold send packet data.

The packet transmission steps are as follows.

(1) Preparing send data and send buffer descriptors (CPU)
The CPU creates send data and send descriptors in the H-bus shared memory. The CPU sets the T bit (T = 1) and E bit (Note 1) and clears the U bit (U = 0) for the buffer descriptor pointing to the send data buffer. It sets the send data size in the SIZE field and the data buffer address indicated by this buffer descriptor in the BAP field. To indicate the end of the send data, the CPU places end-of-chain adjacent to the valid buffer descriptor. The CPU sets the T bit (T = 1), sets the E bit (E = 1) and sets the buffer address to 0 for end-of-chain.

(2) Setting the send descriptor address and permitting transmission (CPU)
The CPU sets the first send buffer descriptor's address in the ETHA0TXDP register. Then, it sets the ETHA0MODE.TXS bit (ETHA0MODE.TXS = 1) to permit transmission. This starts send DMA transfer.

(3) Reading the buffer descriptor and send data (Ethernet controller)
The Ethernet controller analyzes the descriptor, and via the DMA, reads packet data from the send data buffer. When the E bit of the send buffer descriptor is cleared (E = 0), the Ethernet controller automatically reads the next descriptor. It repeats operation until it processes the descriptor (E = 1) for the last packet data.

(4) Setting the U bit of the buffer descriptor (Ethernet controller)
The Ethernet controller sets the U bit (U = 1) of the descriptor when the send DMA transfer is completed with that descriptor.

(5) Reporting the completion of transmission (Ethernet controller)
The Ethernet controller generates a TXI interrupt request to inform that the processing of the descriptor (E = 1) for the last data has ended. Then, it starts processing the next descriptor.

(6) Detecting the end of transmission (Ethernet controller)
If the descriptor read by the Ethernet controller is end-of-chain or the buffer descriptor (U = 1) with which the DMA transfer is completed, the Ethernet controller halts the send DMA and generates a TECI interrupt request.

Note 1: The E bit is set (E = 1) when the last data in the packet is buffered. Otherwise, the E bit is cleared (E = 0).
Remarks: If the ETHA0MACC1.CRCEN bit is set, FCS is added to the end of data. If the ETHA0MACC1.PADEN bit is set, 0 PAD is added automatically when a short frame is sent.
If the send frame length exceeds 1,518 bytes, an interrupt request occurs.

The following describes an example of packet transmission which involves a configuration of multiple descriptors and send buffers.

(1) One packet of data handled using multiple buffer descriptors (example 1)

One packet of data is divided and stored separately in data buffers (1), (2), and (3), and the data is handled using multiple descriptors.

In this example, buffer descriptor (1) is the first descriptor (address set in ETHA0TXDP) and one-packet data is treated via subsequent buffer descriptors (2) and (3).

The BAP (Base Address Pointer) for each buffer descriptor (n) specifies the address of associated data buffer (n).

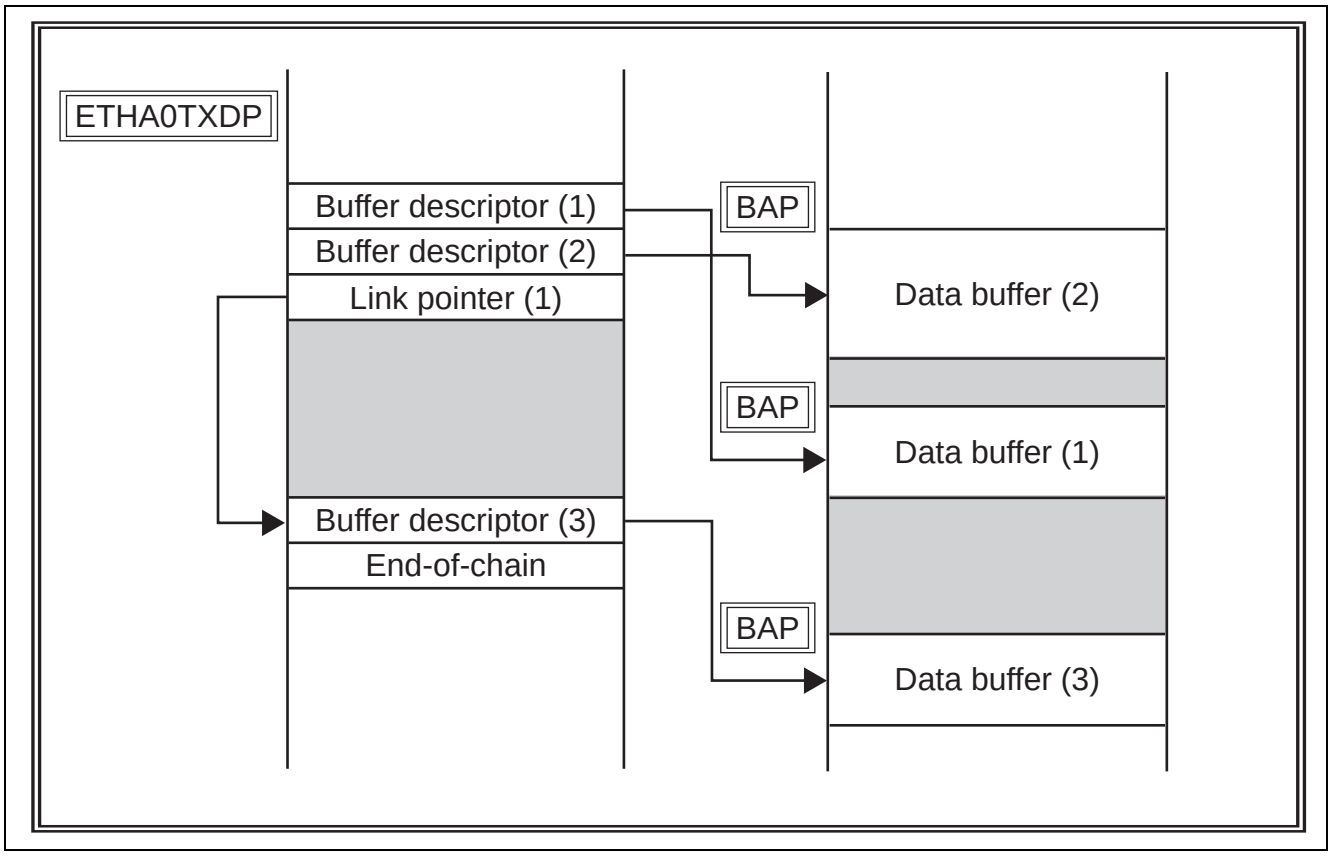


Figure 1.4 Example of Packet Transmission (1/4)

When multiple buffer descriptors are used for handling separated data buffers, the E bits in these buffer descriptors are used.

E = 1 is set when the data buffer specified by each buffer descriptor contains the last data in the packet. Otherwise, E = 0 is set.

The send packet stream ends with the end-of-chain descriptor (T = 1/E = 1).

In this example, the control bits of buffer descriptor (n) are as follows, and the address (in the H-bus shared memory space) of buffer descriptor (1) is set in ETHA0TXDP.

- Buffer Descriptor (1)

When transfer starts:								⇒	When transfer ends:							
T	E	U	D	S	O	Status	Size		T	E	U	D	S	O	Status	Size
0	0	0	-	-	-	-	Send byte length		0	-	1	0/1	-	0/1	-	-

- Buffer Descriptor (2)

When transfer starts:								⇒	When transfer ends:							
T	E	U	D	S	O	Status	Size		T	E	U	D	S	O	Status	Size
0	0	0	-	-	-	-	Send byte length		0	-	1	0/1	-	0/1	-	-

- Link Pointer (1)

T	E	U	D	S	O	Status	Size
1	0	-	-	-	-	-	-

- Buffer Descriptor (3)

When transfer starts:								⇒	When transfer ends:							
T	E	U	D	S	O	Status	Size		T	E	U	D	S	O	Status	Size
0	1	0	-	-	-	-	Send byte length		0	-	1	0/1	-	0/1	-	-

- End-of-Chain

T	E	U	D	S	O	Status	Size
1	1	-	-	-	-	-	-

Remarks: Even if the U bit of the buffer descriptor is read as 1, the Ethernet controller halts the DMA. This allows the U bit to indicate the end of the send packet stream.

(2) One packet of data handled using a single descriptor (example 2)

One data buffer which contains one-packet data is handled using a single descriptor.

Individual packet data stored in data buffer (1) and data buffer (2) is each sent using the descriptor which indicates its BAP.

The BAP (Base Address Pointer) for each buffer descriptor (n) specifies the address of associated data buffer (n).

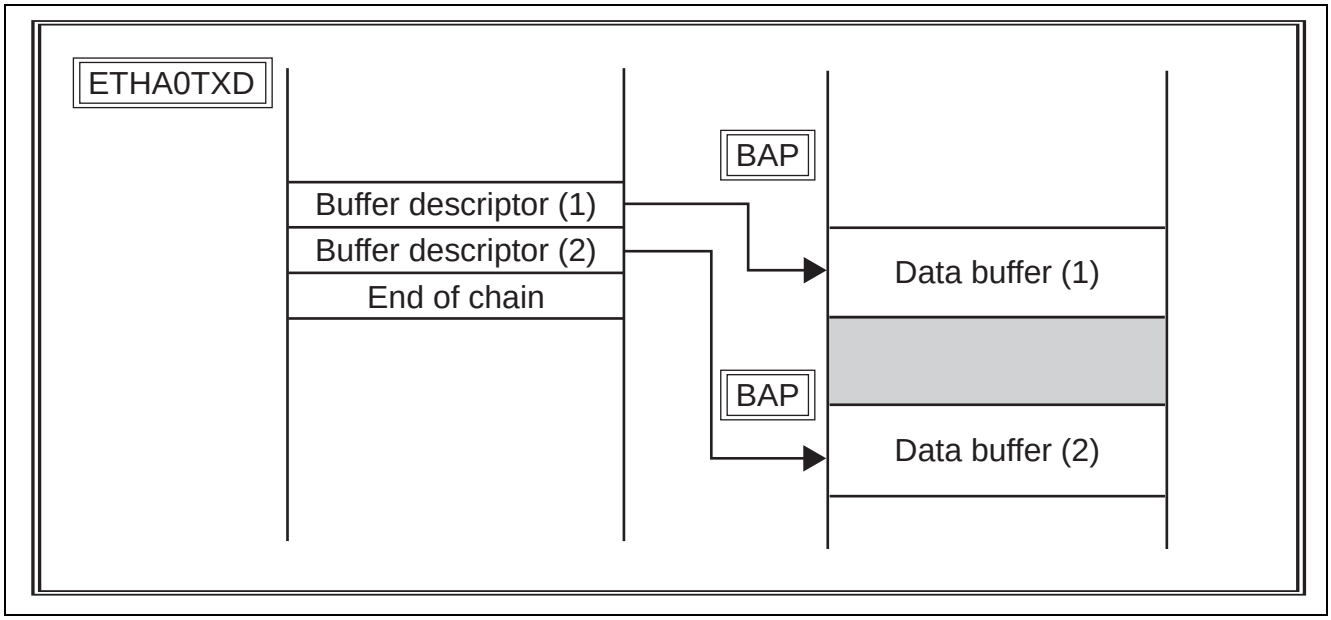


Figure 1.5 Example of Packet Transmission (2/4)

In this example, the control bits of buffer descriptor (n) are as follows, and the address (in the H-bus shared memory space) of buffer descriptor (1) is set in ETHA0TXDP.

- Buffer Descriptor (1)

When transfer starts:								⇒	When transfer ends:							
T	E	U	D	S	O	Status	Size		T	E	U	D	S	O	Status	Size
0	1	0	-	-	-	-	Send byte length		0	-	1	0/1	-	0/1	-	-

- Buffer Descriptor (2)

When transfer starts:								⇒	When transfer ends:							
T	E	U	D	S	O	Status	Size		T	E	U	D	S	O	Status	Size
0	1	0	-	-	-	-	Send byte length		0	-	1	0/1	-	0/1	-	-

- End-of-Chain

T	E	U	D	S	O	Status	Size
1	1	-	-	-	-	-	-

In the above example, both buffer descriptors (1) and (2) should already be set when transfer starts. However, applications can process buffer descriptor (2) while the Ethernet controller is processing buffer descriptor (1). This capability is described below using an example.

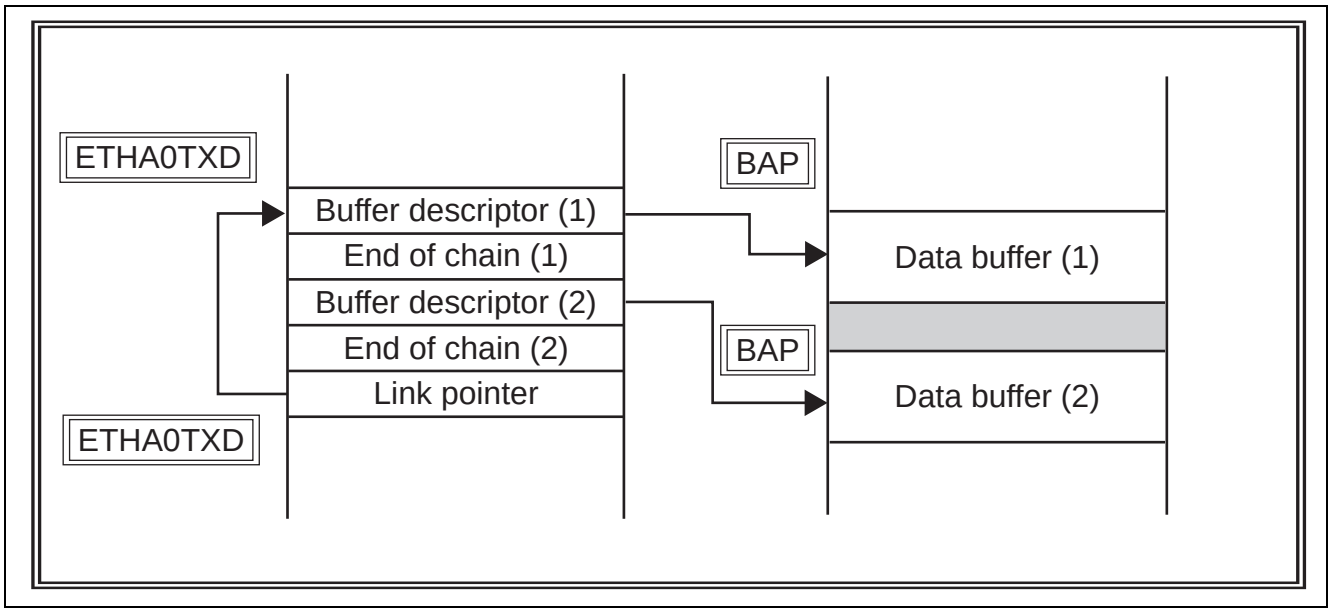


Figure 1.6 Example of Packet Transmission (3/4)

In this example, unlike the previous one, there is end-of-chain between buffer descriptors (1) and (2).

The higher-level application prepares buffer descriptor (1) and then starts transmission using ETHA0TXDP as buffer descriptor (1).

After the start of transmission, the Ethernet controller processes buffer descriptor (1), and with succeeding end-of-chain (1), halts the DMA. Then, it generates a TECI interrupt request.

Applications can process buffer descriptor (2) from the time the transmission starts until the TECI interrupt request occurs.

The Ethernet controller can process buffer descriptor (2) upon the TECI interrupt request. Thus, using this TECI interrupt request, it is possible to:

(a) Release the descriptor/buffer which causes the TECI request.
(b) Continue transmission if the next descriptor is already prepared.

(3) Multiple packets of data handled using multiple descriptors (example 3)

Two-packet data (A-n) and (B-n) is divided and stored separately in data buffers (A-1), (A-2), (B-1), (B-2) and (B-3). Using multiple descriptors, BAP (BaseAddressPointer) for each buffer descriptor (n) is specified.

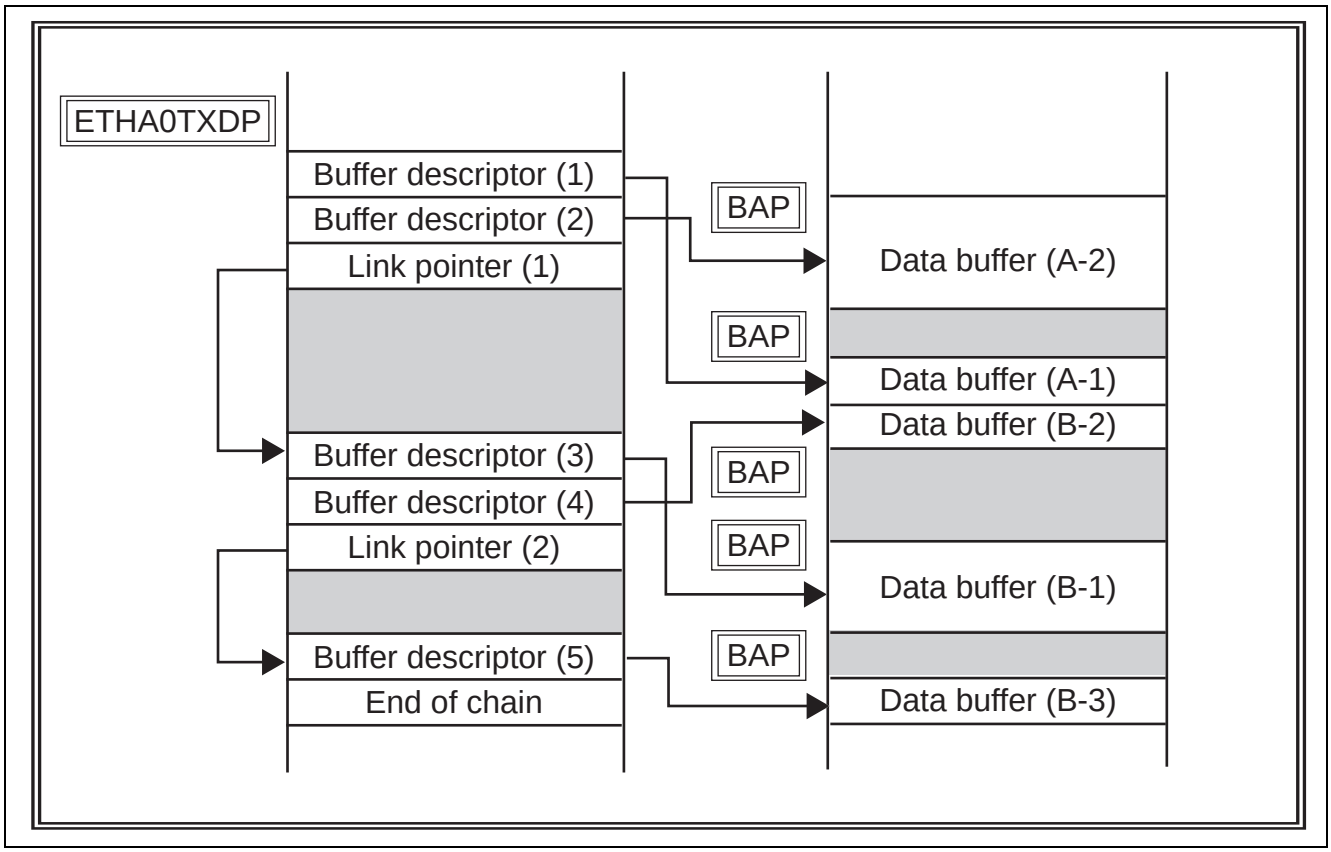


Figure 1.7 Example of Packet Transmission (4/4)

In this example, the control bits of buffer descriptor (n) are as follows, and the address (in the H-bus shared memory space) of buffer descriptor (1) is set in ETHA0TXDP.

- Buffer Descriptor (1)

When transfer starts:								⇒	When transfer ends:							
T	E	U	D	S	O	Status	Size		T	E	U	D	S	O	Status	Size
0	0	0	-	-	-	-	Send byte length		0	-	1	0/1	-	0/1	-	-

- Buffer Descriptor (2)

When transfer starts:								⇒	When transfer ends:							
T	E	U	D	S	O	Status	Size		T	E	U	D	S	O	Status	Size
0	1	0	-	-	-	-	Send byte length		0	-	1	0/1	-	0/1	-	-

- Link Pointer (1)

T	E	U	D	S	O	Status	Size
1	0	-	-	-	-	-	-

- Buffer Descriptor (3)

When transfer starts:								⇒	When transfer ends:							
T	E	U	D	S	O	Status	Size		T	E	U	D	S	O	Status	Size
0	0	0	-	-	-	-	Send byte length		0	-	1	0/1	-	0/1	-	-

- Buffer Descriptor (4)

When transfer starts:								⇒	**When transfer ends:**							
T	E	U	D	S	O	Status	Size		T	E	U	D	S	O	Status	Size
0	0	0	-	-	-	-	Send byte length		0	-	1	0/1	-	0/1	-	-

- Link Pointer (2)

T	E	U	D	S	O	Status	Size
1	0	-	-	-	-	-	-

- Buffer Descriptor (5)

When transfer starts:								⇒	**When transfer ends:**							
T	E	U	D	S	O	Status	Size		T	E	U	D	S	O	Status	Size
0	1	0	-	-	-	-	Send byte length		0	-	1	0/1	-	0/1	-	-

- End-of-Chain

T	E	U	D	S	O	Status	Size
1	1	-	-	-	-	-	-

1.3.6 Examples of Packet Reception

Ethernet reception involves using the receive descriptors (buffer descriptor, link pointer, and end-of-chain) and data buffers which hold receive packet data.

The packet reception steps are as follows.

(1) Preparing a receive buffer descriptor (CPU)

The CPU creates a receive buffer descriptor in the H-bus shared memory. At this time, the CPU clears the T bit (T = 1), E bit (E = 0), and U bit (U = 0) of the receive buffer descriptor. It sets the area size (in units of bytes) of the data buffer pointed to by this descriptor in the SIZE field. It also sets the address of this data buffer pointed to by this descriptor in the BAP field.

Because one packet can be stored using multiple descriptors, the data buffer size can be set smaller than the maximum Ethernet packet size.

(2) Setting the receive descriptor address and permitting reception (CPU)

The CPU sets the first receive buffer descriptor's address in the RXDP register. Then, it sets the ETHA0MODE.RXS bit (ETHA0MODE.RXS) to permit reception. This starts receive DMA transfer.

(3) Reading the buffer descriptor and storing receive data (Ethernet controller)

The Ethernet controller analyzes the descriptor, and via the DMA, transfers receive data in the FIFO to the data buffer area pointed to by this descriptor. When the data buffer area becomes full due to subsequent reception, the Ethernet controller reads the next descriptor to continue the same processing.

(4) Writing back to the buffer descriptor (Ethernet controller)

Upon completion of the receive DMA transfer, the Ethernet controller writes back the E bit, U bit, S bit, and receive status values, and also writes back the byte length of the transferred receive data back to the SIZE field.

If the receive packets are separately written to multiple data buffers, the Ethernet controller sets the E bit (E = 1) and U bit (U = 1) as the control bits of the last buffer descriptor. It sets the U bit (U = 1) and S bit (S = 1) and writes the receive status back to the Status field for the first buffer descriptor.

(5) Reporting the completion of reception (Ethernet controller)

Upon completion of the DMA transfer of the last frame data, the Ethernet controller generates an RXI interrupt request to inform that the DMA transfer has ended. Then, it starts processing the next descriptor.

(6) Detecting the end of reception (Ethernet controller)

If the descriptor read by the Ethernet controller is end-of-chain or the buffer descriptor (U = 1) with which the DMA transfer is completed, the Ethernet controller halts the receive DMA and generates an RECI interrupt request.

Remarks: The Ethernet controller has a capability to turn on/off the receive checksum function with the ETHA0TRANSCTL.RXCHKSMEN bit. If this function is used, the packet length plus two bytes is written back to the SIZE field. Thus, the transfer destination data buffer area should be large enough to hold data.

The following describes an example of packet reception which involves using the receive descriptors.

(1) One packet of data handled using multiple buffer descriptors (example 1)

One packet of data is divided and stored separately in data buffers (1), (2), and (3), and the data is handled using multiple descriptors.

BAP (BaseAddressPointer) for each buffer descriptor (n) specifies the address of associated data buffer (n).

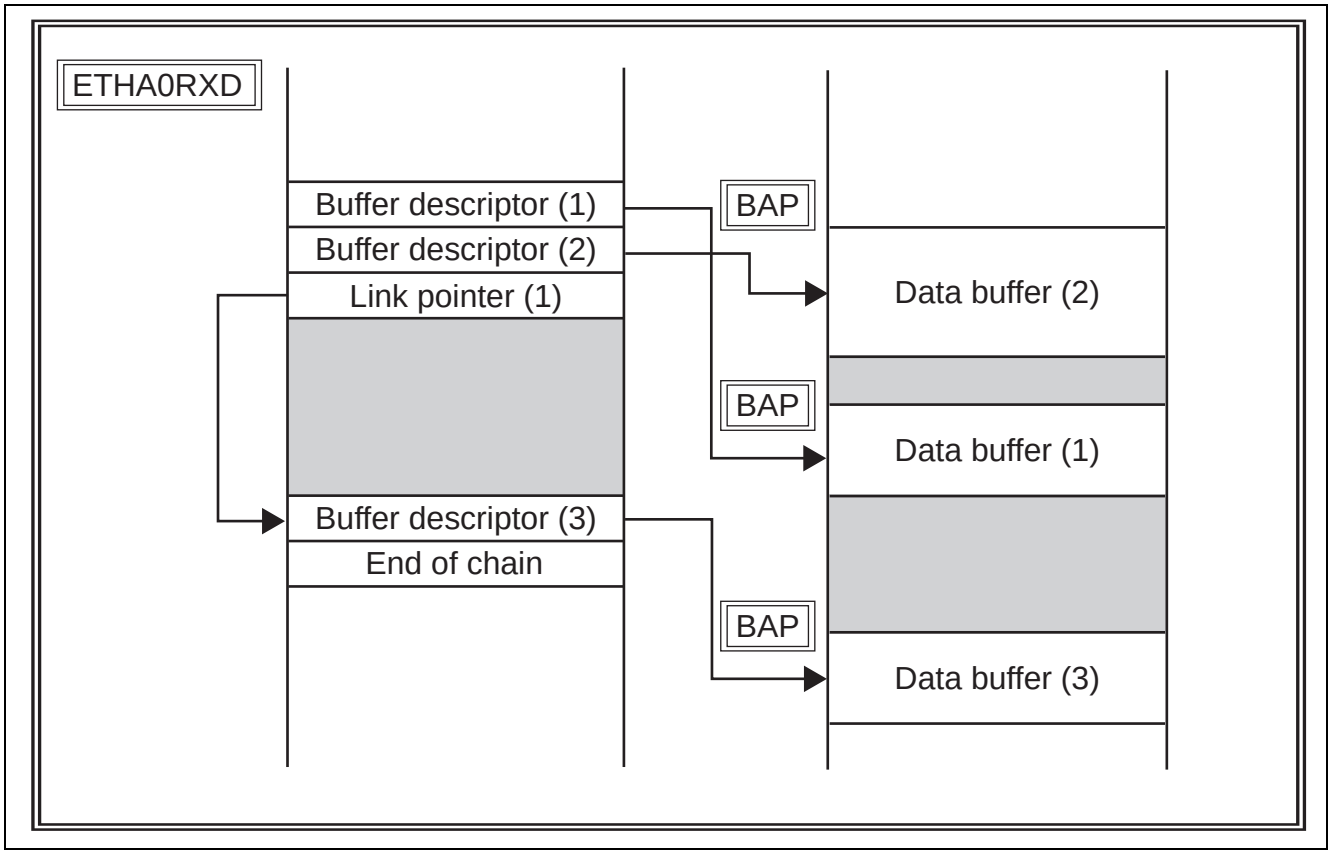


Figure 1.8 Example of Packet Reception (1/2)

In the above example, the control bits of buffer descriptor (n) are as follows, and the address (in the H-bus shared memory space) of buffer descriptor (1) is set in RXDP. After these settings, the ETHA0MODE.RXS bit is set, thereby starting receive DMA transfer.

- Buffer Descriptor (1)

When transfer starts:								⇒	When transfer ends:							
T	E	U	D	S	O	Status	Size		T	E	U	D	S	O	Status	Size
0	0	0	-	-	-	-	D.Buf(1) Size		0	0	1	0/1	1	0/1	Receive status	DMA transfer size

- Buffer Descriptor (2)

When transfer starts:								⇒	When transfer ends:							
T	E	U	D	S	O	Status	Size		T	E	U	D	S	O	Status	Size
0	0	0	-	-	-	-	D.Buf(2) Size		0	0	1	0/1	0	0/1	-	DMA transfer size

- Link Pointer (1)

T	E	U	D	S	O	Status	Size
1	0	-	-	-	-	-	-

- Buffer Descriptor (3)

When transfer starts:								⇒	When transfer ends:							
T	E	U	D	S	O	Status	Size		T	E	U	D	S	O	Status	Size
0	0	0	-	-	-	-	D.Buf(2) Size		0	1	1	0/1	0	0/1	-	DMA transfer size

- End-of-Chain

T	E	U	D	S	O	Status	Size
1	1	-	-	-	-	-	-

(2) One packet of data handled using a single descriptor (example 2)
One packet of data is handled using a single descriptor.
BAP (BaseAddressPointer) for buffer descriptor (n) specifies the address of associated data buffer (n).

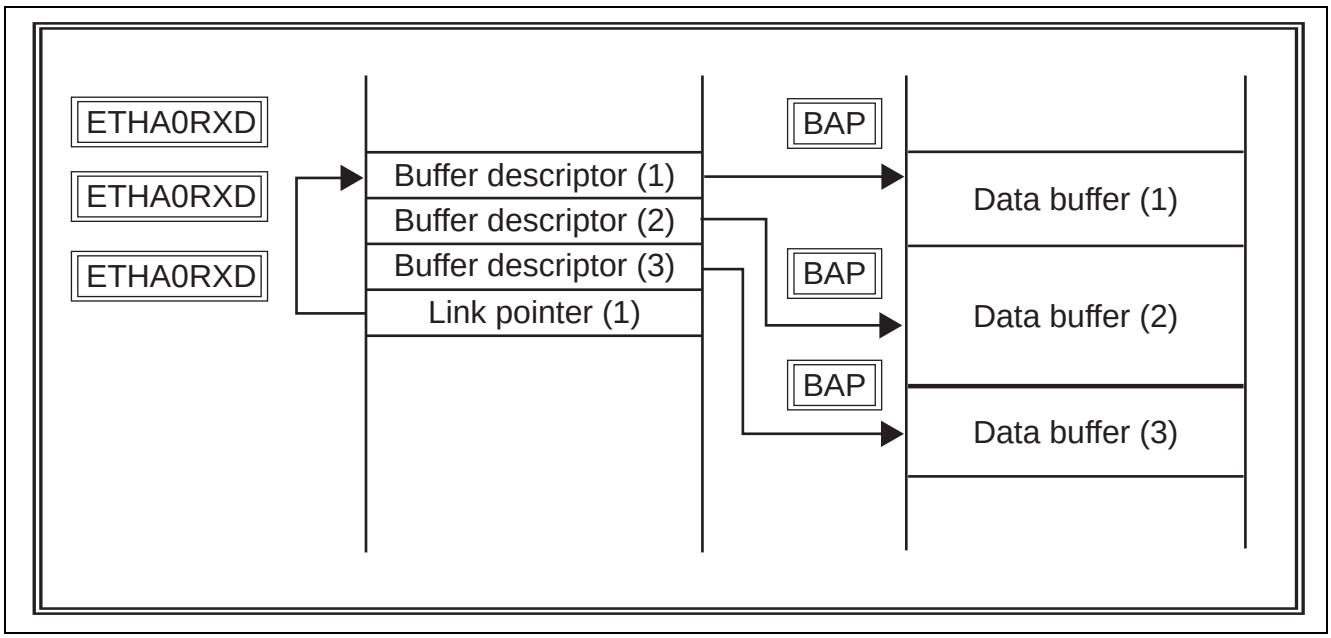


Figure 1.9 Example of Packet Reception (2/2)

In the above example, the control bits of buffer descriptor (n) are as follows, and the address (in the H-bus shared memory space) for each buffer descriptor whose transfer has completed (which is available after the transfer) is set in ETHA0RXDP.

- Buffer Descriptor (n)

When transfer starts:								⇒	When transfer ends:							
T	E	U	D	S	O	Status	Size		T	E	U	D	S	O	Status	Size
0	0	0	-	-	-	-	D.Buf(n) Size		0	1	1	0/1	1	0/1	Receive status	DMA transfer size

- Link Pointer (1)

T	E	U	D	S	O	Status	Size
1	0	-	-	-	-	-	-

1.3.7 Descriptor Structure for this Sample Driver

This subsection shows the descriptor structure for this sample driver.

There are five descriptor areas each containing a send descriptor in the H-bus shared memory space. Because they should be used as ring buffers, the last descriptor is a link pointer which indicates the location of the first descriptor.

There are also four descriptor areas each containing a receive descriptor in the H-bus shared memory space. Because they should be used as ring buffers, the last descriptor is a link pointer which indicates the location of the first descriptor.

Below are the send and receive descriptor areas for this sample driver.

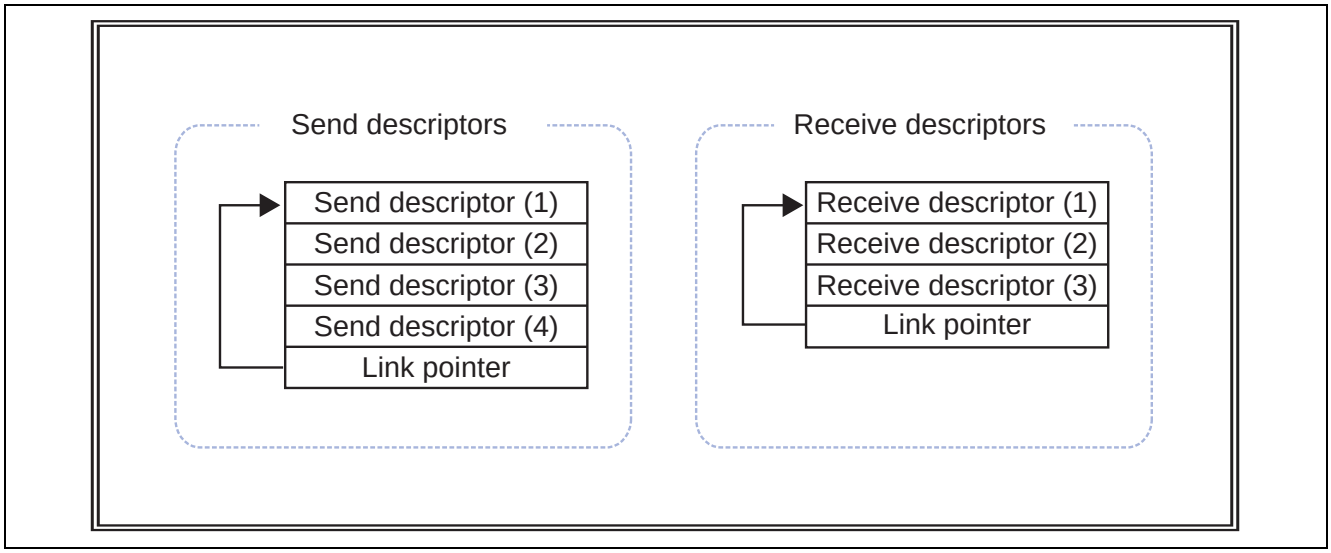


Figure 1.10 Descriptor Areas for the Sample Driver

The sample application including this sample driver uses the following descriptors which are allocated to the descriptor areas above.

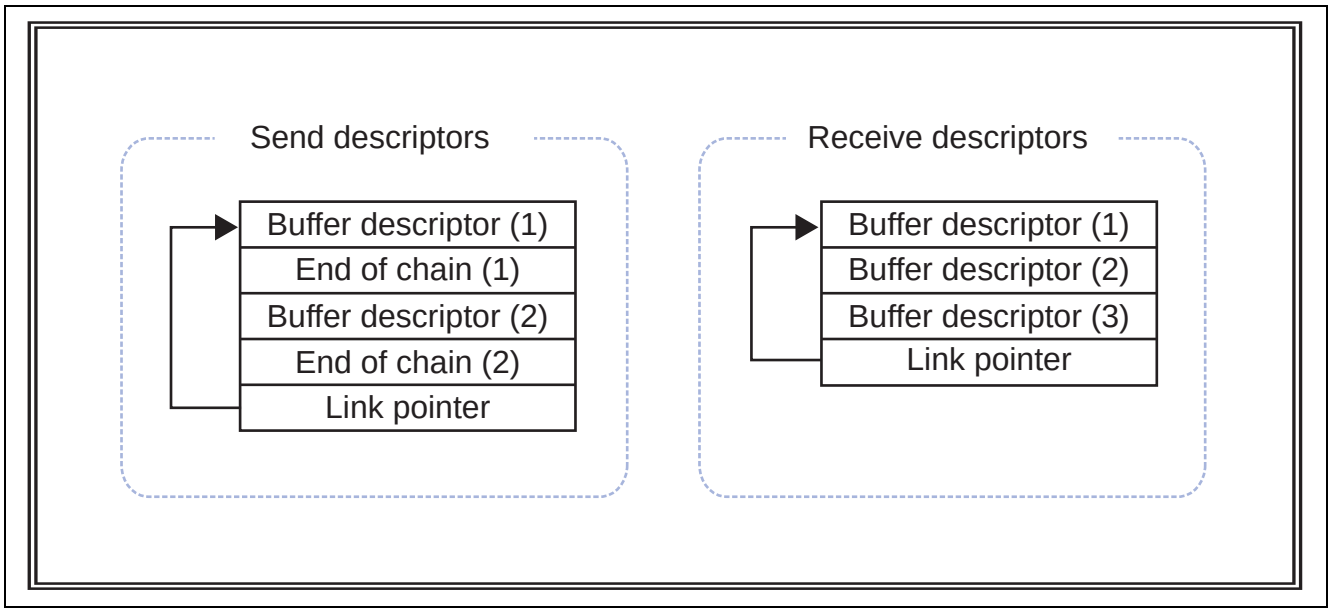


Figure 1.11 Descriptor Structure for the Sample Application

1.3.8 Data Buffer Structure for This Sample Driver

This subsection shows the data buffer structure for this sample driver.

There are four fixed-length send data buffers, indicated by the send descriptors, in the H-bus shared memory space.

There are also three fixed-length receive data buffers, indicated by the receive descriptors, in the H-bus shared memory space.

The figure below shows the relationships between the send and receive descriptors and data buffers.

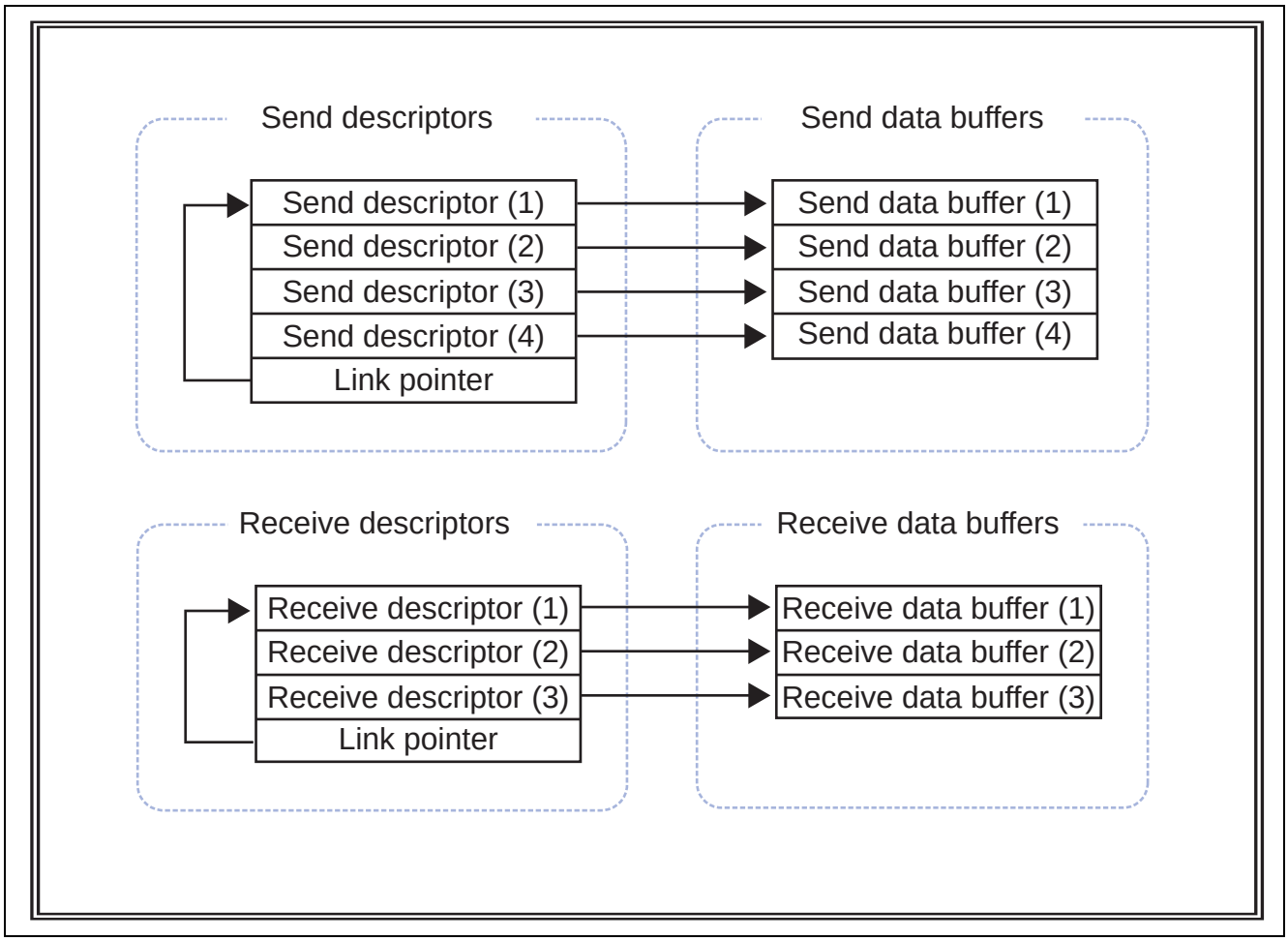


Figure 1.12 Relationships between the Send and Receive Descriptors and Data Buffers for the Sample Driver

The sample application including this sample driver uses the above data buffer areas as follows according to the sample application's descriptor structure.

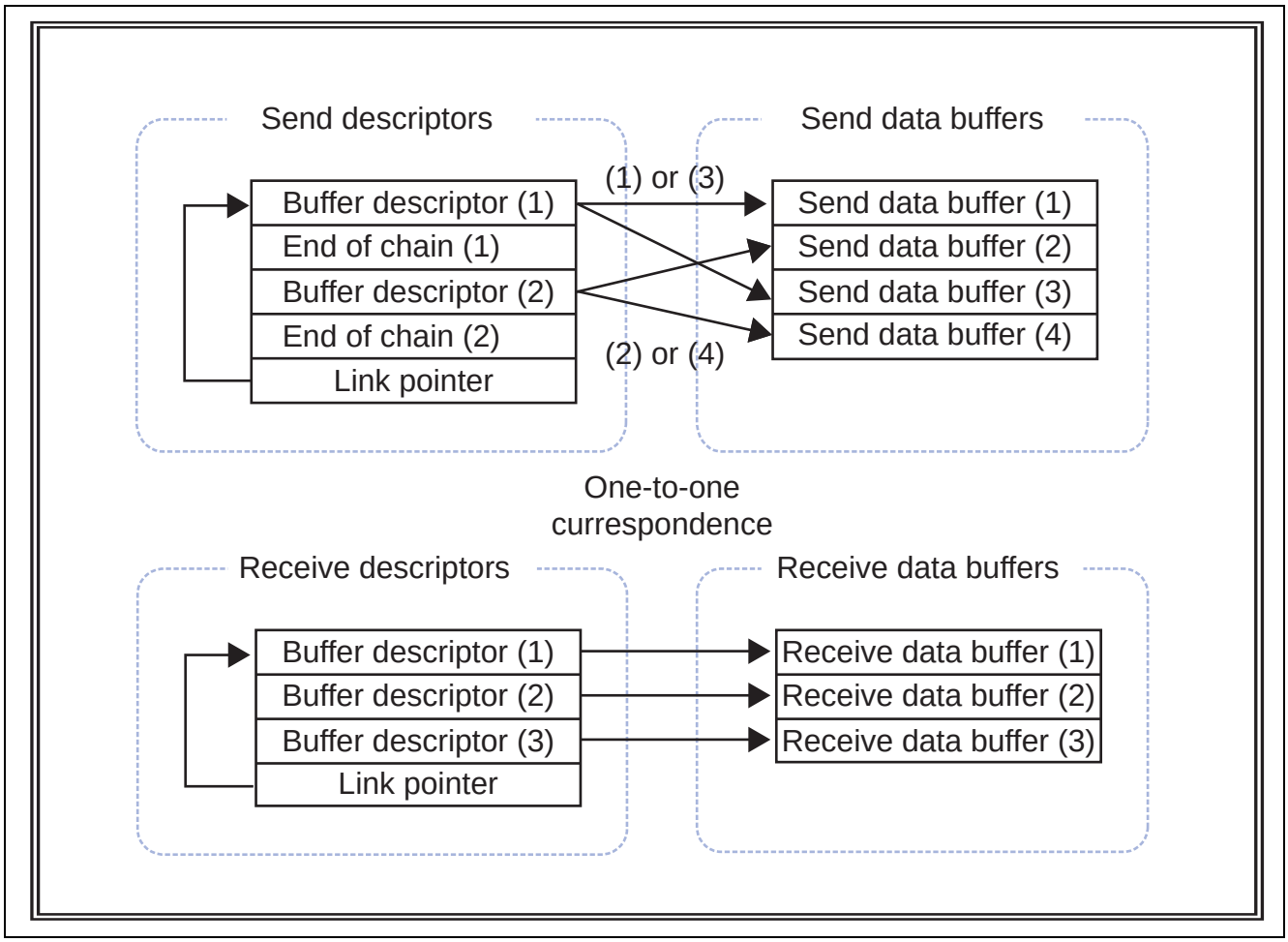


Figure 1.13 Relationships between the Descriptors and Data Buffers for the Sample Application

During send processing by the sample application, one-packet data is handled using a single data buffer, and only one packet is sent at a time through a single data buffer. Thus, a buffer descriptor and end-of-chain are combined into a set, and two of these sets are located at two consecutive addresses to implement a ring buffer structure.

During receive processing, one-packet data is handled using a single data buffer like data handled for send processing. The size of the receive data buffer is equal to the maximum packet size.

Remarks: The Ethernet controller automatically reads a buffer descriptor at address +8H if descriptors are allocated in consecutive address spaces.

1.3.9 H-Bus Shared Memory Space Address Mapping

This subsection describes how to map the descriptors and data buffers used for this sample driver to addresses.

All the descriptors, send data buffers, and receive data buffers treated by the Ethernet controller are referenced or transferred through the incorporated DMA controller which is dedicated for use by the Ethernet controller.

This dedicated DMA controller provides DMA capability via the Ethernet controller's internal system bus and serves as a DMA controller for sending and receiving data in only the H-bus shared memory.

It is therefore necessary to allocate the descriptors and data buffers in the H-bus shared memory space.

Remarks: For details about the dedicated DMA controller for the Ethernet controller, refer to the "V850E2/MN4 User's Manual, Hardware (R01UH0011EJ)."

(1) H-bus shared memory space mapping information

- μPD70F3512, μPD70F3514, μPD70F3515 (Note 1)
  F9800000H to F980FFFFH (64 Kbytes)

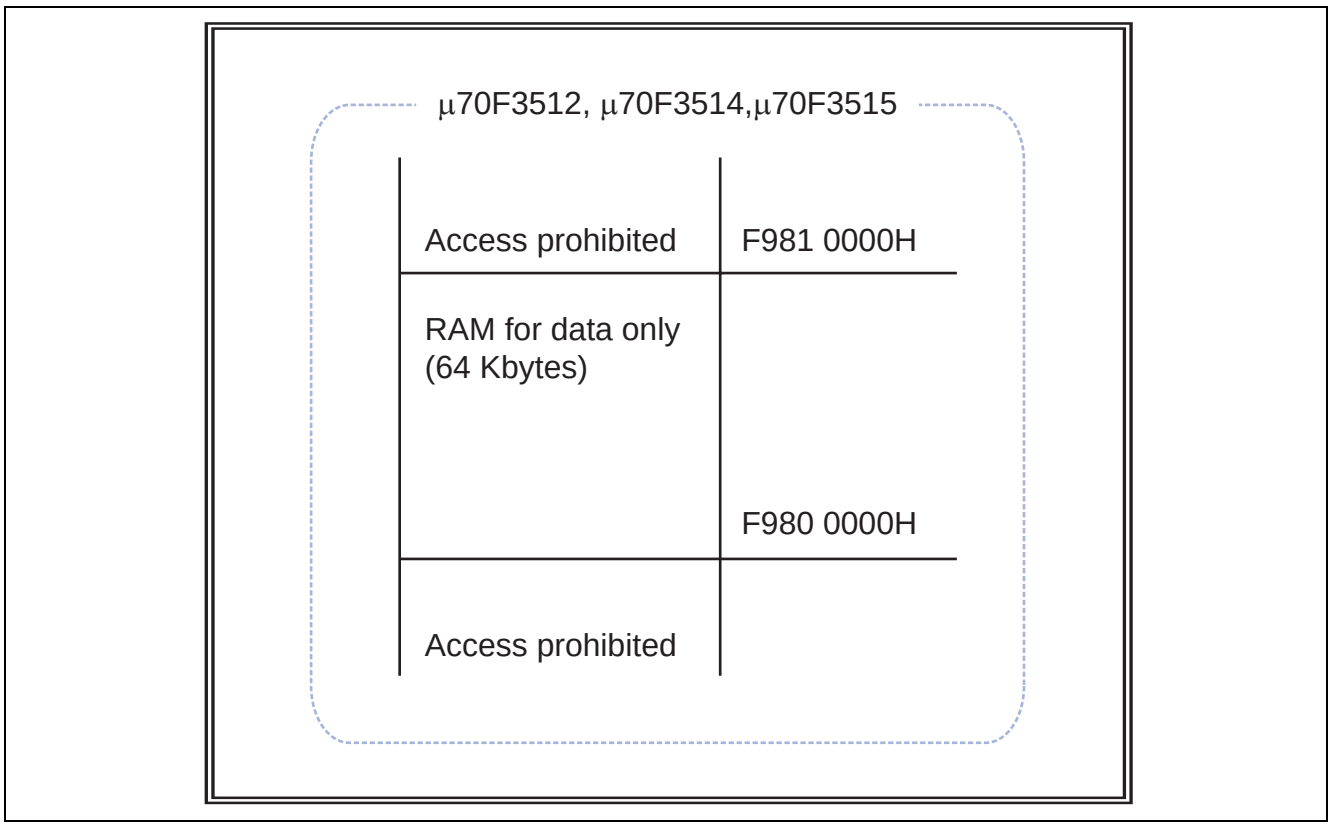


Figure 1.14 H-Bus Shared Memory

Note 1: μPD70F3510 does not have the Ethernet functions.

(2) Descriptor definitions

The following statement in source file \ether_driver.c defines the TX/RX descriptor.

List 1.2 Descriptor Area Allocation

```
UINT8 tx_descriptor[ (sizeof(CTRL_DESC) * TX_DESCRIPTOR_MAX) ];
UINT8 rx_descriptor[ (sizeof(CTRL_DESC) * RX_DESCRIPTOR_MAX) ];
```

The number of descriptors can vary depending on the TX_DESCRIPTOR_MAX and RX_DESCRIPTOR_MAX definitions.

TX_DESCRIPTOR_MAX and RX_DESCRIPTOR_MAX are defined in header file \ether_driver.h.

(3) Data buffer definitions

The following statement in source file \ether_driver.c defines the TX/RX data buffer.

List 1.3 Data Buffer Area Allocation

```
UINT8 tx_dma_buf[ TX_BUFFER_MAX ][ (FRAME_SIZE + DMA_SUM_SIZE) ];
UINT8 rx_dma_buf[ RX_BUFFER_MAX ][ (FRAME_SIZE + DMA_SUM_SIZE) ];
```

The data buffer is arranged as a two-dimensional array of values ([Number of data buffers] [Area size]).

The number of data buffers can vary depending on the TX_BUFFER_MAX and RX_BUFFER_MAX definitions.

TX_BUFFER_MAX and RX_BUFFER_MAX are defined in header file \ether_driver.h.

The area size can vary depending on the FRAME_SIZE + DMA_SUM_SIZE definition.

FRAME_SIZE + DMA_SUM_SIZE is defined as 1,518 bytes, 2 bytes in \ether_driver.h.

(4) Link directive

To allocate memory as expected, it is necessary to give information about program code and data allocation to the linker. This information is called a “link directive” and a file containing a link directive is called a “link directive file.”

Information about areas (“sections”) which contain programs and data is divided into types and attributes and provided as “segment” information whose address is determined. Information to be allocated in the H-bus shared memory space such as descriptors and data buffers defines the section name, area, and size of the H-bus shared memory space according to the link directive.

Remarks: The following three directives are specified as link directives.

- Segment directive
- Mapping directive
- Symbol directive

This sample uses the segment and mapping directives below.

List 1.4 Segment and Mapping Directives

```
ETH_MEMORY : !LOAD ?RW V0xf9800000 L0x10000 {
    eth_memory.data   = $PROGBITS    ?AW  eth_memory.data;
    eth_memory.bss    = $NOBITS      ?AW  eth_memory.bss;
};
```

This sample uses the symbol directive below.

List 1.5 Symbol Directive

```
__tp_TEXT @ %TP_SYMBOL;
__gp_DATA @ %GP_SYMBOL &__tp_TEXT {DATA ETH_MEMORY};
__ep_DATA @ %EP_SYMBOL;
```

These link directives are defined in link directive file \cnet.dir.

Remarks: For details about the link directives, refer to the “CubeSuite Ver. 1.40 User’s Manual: Coding for CX Compiler (R20UT0259EJ).”

(5) Allocation declaration

Using the area name defined by the link directive, the descriptors and data buffers are allocated in the H-bus shared memory space.

List 1.6 Descriptors and Data Buffer Allocation

```
#pragma section data "eth_memory"
UINT8 tx_descriptor[ (sizeof(CTRL_DESC) * TX_DESCRIPTOR_MAX) ];
UINT8 rx_descriptor[ (sizeof(CTRL_DESC) * RX_DESCRIPTOR_MAX) ];
UINT8 tx_dma_buf[ TX_BUFFER_MAX ][ (FRAME_SIZE + DMA_SUM_SIZE) ];
UINT8 rx_dma_buf[ RX_BUFFER_MAX ][ (FRAME_SIZE + DMA_SUM_SIZE) ];
#pragma section data "eth_memory"
```

Remarks: For information about coding with the #pragma directive, refer to the “CubeSuite Ver. 1.40 User’s Manual: Coding for CX Compiler (R20UT0259EJ).”

(6) Allocation information

The figure below shows the mapping configuration of the H-bus shared memory space in this sample driver.

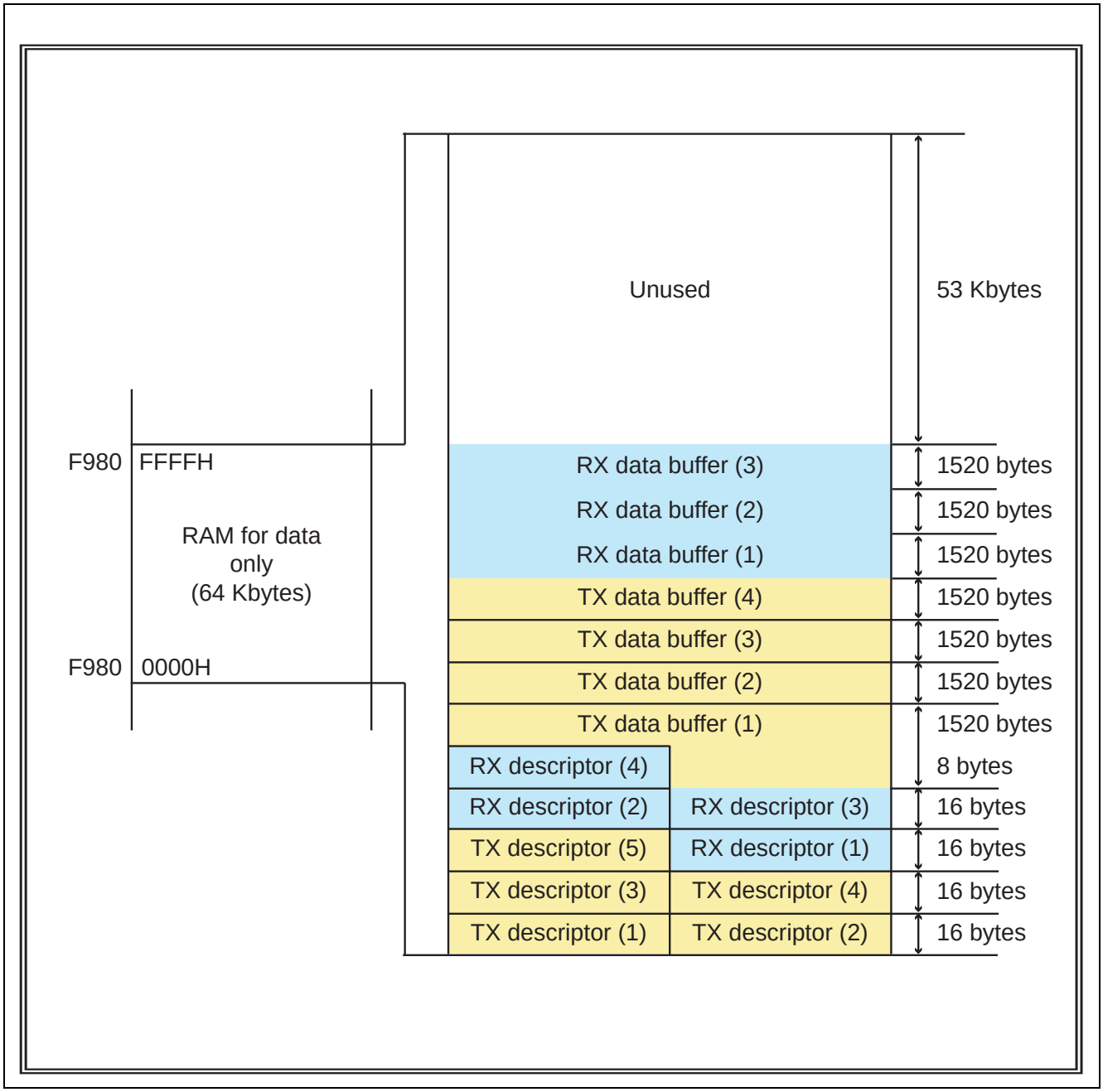


Figure 1.15 H-Bus Shared Memory Space Mapping

1.4 Function Specifications

This section describes the specifications for this sample driver's functions.

1.4.1 List of Functions

Below is a list of the processing programs.

Table 1.5 Functions

File name	ether_driver.c
Function name	**Functional overview**
void int_INTETHA0SRX_hdr(void)	Processes INTETMRQ interrupts.
void int_INTETHA0SCRXTX_hdr(void)	Processes INTETMRX and INTETMTX interrupts.
void int_INTETHA0FS_hdr(void)	Processes INTETMFS interrupts.
void int_INTETHA0TS_hdr(void)	Processes INTETMTS interrupts.
void int_INTETHA0RS_hdr(void)	Processes INTETMRS interrupts.
void int_INTETHA0MACV_hdr(void)	Processes INTETMOV interrupts.
void ether_initialize(UINT8*)	Initializes the Ethernet controller.
UINT16 phy_read_reg(UINT8)	Read-accesses the external PHY via the MII interface.
void phy_write_reg(UINT8 , UINT16)	Write-accesses the external PHY via the MII interface.
UINT16 ether_get_recv_len(void)	Acquires the length of a transferred packet written back by the Ethernet controller at the time of DMA transfer of the receive packet indicated by the receive descriptor.
UINT8* ether_recv(void)	Acquires the data buffer address transferred by the receive DMA. Prepares a descriptor for the next reception.
void ether_recv_start(CTRL_BUF*)	Passes a valid descriptor to the Ethernet controller to start reception if the receive DMA is halted.
UINT8* ether_get_send_buf(void)	Acquires a valid data buffer address for the send DMA.
void ether_set_buf(UINT8* , UINT16 , UINT16)	Sets the data buffer address holding the send data in a valid descriptor. Prepares for the next data setting.
void ether_send(void)	Sets EOC in a valid descriptor. Prepares for the next data setting.
void ether_send_start(CTRL_BUF*)	Passes a valid descriptor to the Ethernet controller to start transmission if the send DMA is halted.

The table below shows the user application functions required for this sample driver.

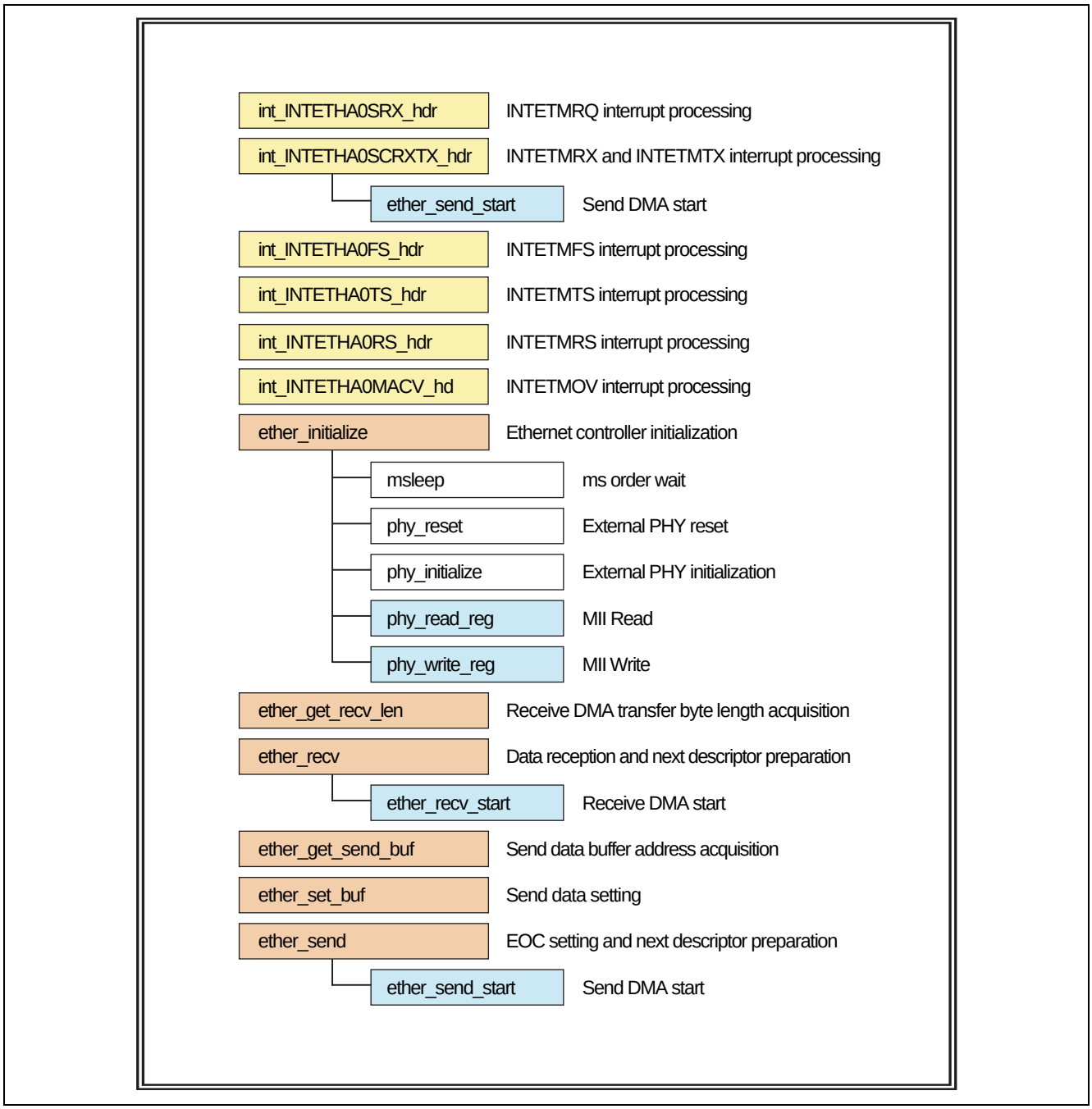
Table 1.6 User Functions Required for This Sample Driver

Function name	Functional overview
void phy_reset(void)	Resets the external PHY in the user system.
void phy_initialize(void)	Initializes the external PHY in the user system.
void msleep (unsigned int)	Processes wait in ms units.

1.4.2 Call Tree

The call tree for this sample driver is shown below.

List 1.7 Call Tree



1.4.3 Data Types

This subsection shows the data macros (such as the data types and return values) used in this sample driver.

Header file \types.h defines macro definitions for data types of various parameters used in this sample driver.

Below is a list of the data types.

Table 1.7 Data Types

Macro	Type	Meaning
UINT8	unsigned char	Unsigned 8-bit integer
UINT16	unsigned short	Unsigned 16-bit integer
UINT32	unsigned int	Unsigned 32-bit integer
INT8	signed char	Signed 8-bit integer
INT16	signed short	Signed 16-bit integer
INT32	signed int	Signed 32-bit integer
VUINT8	volatile unsigned char	Unsigned 8-bit integer (declared as volatile)
VUINT32	volatile unsigned int	Unsigned 32-bit integer (declared as volatile)

1.4.4 Data Structure

This subsection shows the data structures used by this sample driver.

(1) Storage classes

Header file \ether_driver.h defines the storage classes.

They are shown below.

Table 1.8 Descriptor-Type Storage Class

	typedef struct tst_ctrl_desc { } CTRL_DESC	
Members	UINT32 value	Low order word of a descriptor
	UINT8 * addr	High order word of a descriptor

Table 1.9 Storage Class for the Driver's Internal Information

	typedef struct tst_ctrl_buf { } CTRL_BUF	
Members	CTRL_DESC * tx_next	Beginning of a send active descriptor
	CTRL_DESC * tx_write	Send active descriptor
	UINT8 tx_next_buf	Send active buffer
	CTRL_DESC * rx_read	Receive active descriptor

1.4.5 Function Specifications

This subsection shows the specifications for this sample driver's functions.

(1) ether_initialize

Function name	**ether_initialize**
C language coding format	void ether_initialize(UINT8* mymac)
Parameters	UINT8* mymac: Pointer to the local MAC address set in the LSA1 and LSA2 registers mymac[0]:MSB - mymac[5]:LSB
Return value	None
Processing overview	This function enables the CPU integrated Ethernet controller and then initializes the MAC registers, FIFO registers, dedicated DMA registers for the Ethernet controller, and the external PHY via the MII interface. After the initialization, this function sets interrupts for the CPU-integrated Ethernet controller, triggers receive DMA operation, and completes processing.

List 1.8 ether_initialize Function (1/10)

```
void ether_initialize( UINT8* mymac )
{
   CTRL_BUF  * ctrl_buf;
   CTRL_DESC * ctrl_desc;
   UINT32      lp;
   UINT32      value;

   /* MAC register initialize  */

   /* MACC1 register   */
   ETHA0MACC1 = MACC1_PARF  |  /* control packet pass      */
            MACC1_CRCEN |  /* CRC enable               */
            MACC1_PADEN |  /* PAD enable               */
            MACC1_FULLD;   /* full duplex              */

   /* MACC2 register   */
   ETHA0MACC2 = MACC2_MCRST |  /* MAC reset               */
            MACC2_TFRST |  /* TX reset                 */
            MACC2_RFRST;   /* RX reset                 */
   msleep(1);                /* more necessary more than 5 clock than RXCLK,TXCLK*/
   value      = ETHA0MACC2 & (~(MACC2_MCRST | MACC2_TFRST | MACC2_RFRST));
   ETHA0MACC2 = value;
```

List 1.9 ether_initialize Function (2/10)

```
/* MIIC register    */
ETHA0MIIC = MIIC_MIRST;
msleep(1);              /* more necessary more than 5 clock than fEC */
ETHA0MIIC = MIIC_CLKS_50M;
msleep(1);              /* more necessary more than 5 clock than fEC */

/* PHY reset        */
phy_reset();

/* IPGT register    */
ETHA0IPGT = 0x00000013;   /* default  */

/* IPGR register    */
ETHA0IPGR = 0x00000E13;   /* default  */

/* CLRT register    */
ETHA0CLRT= 0x0000380F;   /* default  */

/* LMAX register    */
ETHA0LMAX = 0x00000600;   /* default  */

/* LSA1/LSA2 register   */
ETHA0LSA1 = ((UINT32)mymac[0]<< 8) | ((UINT32)mymac[1]);
ETHA0LSA2 = ((UINT32)mymac[2]<<24) | ((UINT32)mymac[3]<<16) |
        ((UINT32)mymac[4]<< 8) | ((UINT32)mymac[5]);
```

List 1.10 ether_initialize Function (3/10)

```
/* VLTP register    */
ETHA0VLTP = 0x00000000;       /* default  */

/* AFR register     */
ETHA0AFR = AFR_PRO;       /* PRO.mode */

/* HT1/HT2 register */
ETHA0HT1 = 0x00000000;    /* default  */
ETHA0HT2 = 0x00000000;    /* default  */

/* CAR1/CAR2 register   */
ETHA0CAR1 = 0x00000000;       /* default  */
ETHA0CAR2 = 0x00000000;       /* default  */

/* statistics counter clear */
ETHA0RBYT = 0x00000000;
ETHA0RPKT = 0x00000000;
ETHA0RFCS = 0x00000000;
ETHA0RMCA = 0x00000000;
ETHA0RBCA = 0x00000000;
ETHA0RXCF = 0x00000000;
ETHA0RXPF = 0x00000000;
ETHA0RXUO = 0x00000000;
ETHA0RALN = 0x00000000;
ETHA0RFLR = 0x00000000;
ETHA0RCDE = 0x00000000;
ETHA0RFCR = 0x00000000;
ETHA0RUND = 0x00000000;
ETHA0ROVR = 0x00000000;
ETHA0RFRG = 0x00000000;
ETHA0RJBR = 0x00000000;
ETHA0R64  = 0x00000000;
ETHA0R127 = 0x00000000;
ETHA0R255 = 0x00000000;
ETHA0R511 = 0x00000000;
ETHA0R1K  = 0x00000000;
ETHA0RMAX = 0x00000000;
```

List 1.11 ether_initialize Function (4/10)

```
    ETHA0RVBT = 0x00000000;
    ETHA0TBYT = 0x00000000;
    ETHA0TPKT = 0x00000000;
    ETHA0TFCS = 0x00000000;
    ETHA0TMCA = 0x00000000;
    ETHA0TBCA = 0x00000000;
    ETHA0TUCA = 0x00000000;
    ETHA0TXPF = 0x00000000;
    ETHA0TDFR = 0x00000000;
    ETHA0TXDF = 0x00000000;
    ETHA0TSCL = 0x00000000;
    ETHA0TMCL = 0x00000000;
    ETHA0TLCL = 0x00000000;
    ETHA0TXCL = 0x00000000;
    ETHA0TNCL = 0x00000000;
    ETHA0TCSE = 0x00000000;
    ETHA0TIME = 0x00000000;

    /* FIFO register initialize */

    /* RSTCNT register */
    ETHA0RSTCNT = RSTCONT_SFTRST;   /* soft reset */
    do
    {
       value = ETHA0RSTCNT;
    } while ( value & RSTCONT_SFTRST );

    /* MFFCONT register */
    ETHA0MFFCONT = MFFCONT_RXSDMA1 | /* fixed 1   */
                   MFFCONT_APS     | /* fixed 1   */
                   MFFCONT_APL     | /* fixed 1   */
                   MFFCONT_TXEN;     /* TX Enable */

    /* FLOWTHRESH register */
    ETHA0FLOWTH  = 0x06000200;  /* default */
```

List 1.12 ether_initialize Function (5/10)

```
/* PAUSETM register */
ETHA0PAUSETM = 0x7FFFFFFF;  /* default */

/* RXERSEL register */
ETHA0RXERSEL = RXERSEL_VLAN  | /* (VLAN packet recv)                */
               RXERSEL_USOP  | /* (undef.opecode control packet recv) */
               RXERSEL_DBNB  | /* (dribble nibble received)          */
               RXERSEL_RCRCE | /* (CRCerror)                         */
               RXERSEL_RXER  | /* (RXER)                             */
               RXERSEL_CEPS  | /* (False Carrier)                    */
               RXERSEL_REPS  | /* (preamble + SFD 1nibble packet)    */
               RXERSEL_PAIG;   /* (carrier error, preamble error)    */

/* TXABTCNT register */
ETHA0TXABTCNT = 0x00000001; /* clear */

/* RXABTCNT register */
ETHA0RXABTCNT = 0x00000001; /* clear */

/* STATUS register */
value = ETHA0FSTATUS;   /* clear */
value = ETHA0TXSTATUS;  /* clear */
value = ETHA0RXSTATUS;  /* clear */

/* DMA register initialize */

/* SFTRST register */
ETHA0SFTRST = SFTRST_SFTRST;        /* DMA soft reset */
do
{
   value = ETHA0SFTRST;
} while ( value & SFTRST_SFTRST );  /* Reset complete wait */
```

List 1.13 ether_initialize Function (6/10)

```
/* DMA stop */
value        = ETHA0TRANSCTL & ~(TRANSCTL_TXEN | TRANSCTL_RXEN);
ETHA0TRANSCTL = value;
do
{
   value     = ETHA0TRANSCTL;
} while( value & (TRANSCTL_RXEN_STA | TRANSCTL_TXEN_STA) );

/* PHY initialize */
phy_initialize();

/* DMACM register */
value      = ETHA0DMACM | DMACM_BURST16;    /* 16beat increment burst */
ETHA0DMACM = value;

/* TRANSCTL register */
/* case this bit is set , Rtransfer size of RX DMA is RXSTMONI.RBYn+2(sum) */
value        = ETHA0TRANSCTL | TRANSCTL_RXCHKSMEN; /* RX auto checksum */
ETHA0TRANSCTL = value;

/* interrupt factor */
value = ETHA0INTMS;     /* read clear */
```

List 1.14 ether_initialize Function (7/10)

```
/* descriptor setting */
ctrl_buf            = &g_ctrl_buf;
ctrl_desc           = (CTRL_DESC *)tx_descriptor;
ctrl_buf->tx_write  = ctrl_desc;
ctrl_buf->tx_next   = ctrl_desc;
ctrl_buf->tx_next_buf = 0;
for ( lp = 0; lp < TX_DESCRIPTOR_MAX; lp++ )
{
    if ( lp < (TX_DESCRIPTOR_MAX - 1) )
    {
        ctrl_desc->value = DESC_USED;
        ctrl_desc->addr  = 0;
        ctrl_desc        = ctrl_desc + 1;
    }
    else
    {
        ctrl_desc->value = DESC_LINK;
        ctrl_desc->addr  = (UINT8*)&tx_descriptor;
    }
}

ctrl_desc        = (CTRL_DESC *)rx_descriptor;
ctrl_buf->rx_read = ctrl_desc;
for ( lp = 0; lp < RX_DESCRIPTOR_MAX; lp++ )
{
    if ( lp < (RX_DESCRIPTOR_MAX-1) )
    {
        ctrl_desc->value = DESC_NONE + (FRAME_SIZE+DMA_SUM_SIZE);
        ctrl_desc->addr  = (UINT8*)&rx_dma_buf[lp];
        ctrl_desc        = ctrl_desc + 1;
    }
    else
    {
        ctrl_desc->value = DESC_LINK;
        ctrl_desc->addr  = (UINT8*)&rx_descriptor;
    }
}
```

List 1.15 ether_initialize Function (8/10)

```
/* interrupts setting */

/* FIFO status interrupt MASK*/
value = ETHA0FSTATMK & ~(
        FSTATUS_TACOF | /* TX abort count overflow     */
        FSTATUS_RACOF | /* RX abort count overflow     */
        FSTATUS_TSUP  | /* TX status update            */
        FSTATUS_TFNRTY| /* TX FIFO Abort               */
        FSTATUS_TFWE  | /* TX FIFO    Write Error      */
        FSTATUS_RFFE  | /* RX FIFO info Error          */
        FSTATUS_RSUP  | /* RXSTMONI update             */
        FSTATUS_RFWE  | /* RX FIFO Write Error         */
        FSTATUS_RFOF  | /* RX FIFO overflow            */
        FSTATUS_RFFLW | /* RX FIFO > FLOWTHRSH.FLOWTHR */
        FSTATUS_RFZP    /* RX FIFO > FLOWTHRSH.ZPTHR   */
       );
ETHA0FSTATMK = value;

/* TX status interrupt MASK */
value = ETHA0TXSTATMK & ~(
        TXSTATUS_TAB   | /* TX abort                  */
        TXSTATUS_TGNT  | /* transfer packet over LMAX  */
        TXSTATUS_LCOL  | /* late collision             */
        TXSTATUS_ECOL  | /* MAX collision over         */
        TXSTATUS_TEDFR | /* late transfer(surplus)     */
        TXSTATUS_TDFR  | /* late transfer              */
        TXSTATUS_TFLOR | /* length field > 1500        */
        TXSTATUS_TFLER | /* length field != data field */
        TXSTATUS_TCRCE   /* CRC Error                  */
       );
ETHA0TXSTATMK    =    value;
```

List 1.16 ether_initialize Function (9/10)

```
/* RX status interrupt MASK */
value = ETHA0RXSTATMK & ~(
        RXSTATUS_RLENE | /* RX packet length error        */
        RXSTATUS_VLAN  | /* VLAN packet received          */
        RXSTATUS_USOP  | /* undef.OCP received            */
        RXSTATUS_RPCF  | /* pause control packet recv     */
        RXSTATUS_RCFR  | /* control packet recv           */
        RXSTATUS_DBNB  | /* dribble nibble received       */
        RXSTATUS_RLOR  | /* length field > 1500           */
        RXSTATUS_RLER  | /* length field != data field    */
        RXSTATUS_RCRCE | /* receive CRC error             */
        RXSTATUS_RXER  | /* RXER                          */
        RXSTATUS_CEPS  | /* False Carrier                 */
        RXSTATUS_REPS  | /* preamble + SFD 1nibble packet */
        RXSTATUS_PAIG  | /* carrier error / preamble error */
        RXSTATUS_TXRX  | /* transmission of a message during the HDX reception */
        RXSTATUS_DVCF    /* packet receved                */
       );
ETHA0RXSTATMK    =    value;

/* DMA interrupt MASK       */
ETHA0INTMS = ~(
             INTMS_RBEMSK | /* RX data buffer access error */
             INTMS_RECMSK | /* RX DMA end of chain         */
             INTMS_RXMSK  | /* packet recv (DMA complete)  */
             INTMS_TBEMSK | /* TX data buffer access error */
             INTMS_TECMSK | /* TX DMA end of chain         */
             INTMS_TXMSK    /* packet send (DMA complete)  */
            );
```

List 1.17 ether_initialize Function (10/10)

```
    /* TX/RX enable */
    value       = ETHA0MACC1    | (MACC1_SRXEN);
    ETHA0MACC1    = value;
    value       = ETHA0MFFCONT  | (MFFCONT_RXEN  | MFFCONT_TXEN);
    ETHA0MFFCONT  = value;
    value       = ETHA0TRANSCTL | (TRANSCTL_RXEN | TRANSCTL_TXEN);
    ETHA0TRANSCTL = value;

    /* interrupt clear and enbale */
    ICETHA0SRX  = 0x000f;
    ICETHA0SCRX = 0x000f;
    ICETHA0SCTX = 0x000f;
    ICETHA0FS   = 0x000f;
    ICETHA0TS   = 0x000f;
    ICETHA0RS   = 0x000f;
    ICETHA0MAC  = 0x000f;

    /* RX start */
    ctrl_desc = ctrl_buf->rx_read; /* RX DMA descriptor start address */
    ether_recv_start( ctrl_buf );
}
```

(2) phy_read_reg

Function name	phy_read_reg
C language coding format	UINT16 phy_read_reg(UINT8 rReg)
Parameters	UINT8 rReg: Address of the PHY register to be handled
Return value	UINT16: Value read from the PHY register
Processing overview	This function provides read access to the external PHY register via the MII interface.

List 1.18 phy_read_reg

```
UINT16 phy_read_reg( UINT8 rReg )
{
    /* SCANA bit check */
    if (ETHA0MIND & MIND_SCANA )
    {
        ETHA0MCMD = 0;
    }

    /* BUSY bit check */
    while ( ETHA0MIND & MIND_BUSY );

    /* read PHY register */
    ETHA0MADR = (((UINT32)PHY_ADDRESS << 8) | ((UINT32)rReg) );

    ETHA0MCMD = MCMD_RSTAT;

    /* BUSY bit check */
    while ( ETHA0MIND & MIND_BUSY )
    {
        ;
    }
    return (UINT16)ETHA0MRDD;
}
```

(3) phy_write_reg

Function name	**phy_write_reg**
C language coding format	void phy_write_reg(UINT8 wReg , UINT16 wdata)
Parameters	UINT8 wReg：Address of the PHY register to be handled UINT16 wdata：Value to be written to the PHY register
Return value	None
Processing overview	This function provides write access to the external PHY register via the MII interface.

List 1.19 phy_write_reg

```
void phy_write_reg( UINT8 wReg , UINT16 wdata )
{
    /* SCANA bit check */
    if (ETHA0MIND & MIND_SCANA )
    {
        ETHA0MCMD    = 0;
    }

    /* BUSY bit check */
    while ( ETHA0MIND & MIND_BUSY ){
        ;
    }

    /* write PHY register */
    ETHA0MADR = (((UINT32)PHY_ADDRESS << 8) | ((UINT32)wReg));
    ETHA0MWTD = wdata;

    /* BUSY bit check */
    while ( ETHA0MIND & MIND_BUSY );
}
```

(4) ether_get_recv_len

Function name	ether_get_recv_len
C language coding format	UINT16 ether_get_recv_len(void)
Parameters	None
Return value	Transfer size written back to the descriptor. 0 is returned if the receive DMA has yet to be completed.
Processing overview	This function returns the transfer size written back to the descriptor if the descriptor is already transferred. It returns 0 if a data buffer access error or overflow occurs or if data buffer transfer is not yet completed. This function is not implemente because processing after the detection of either an access error or overflow depends on the system.

List 1.20 ether_get_recv_len

```
UINT16 ether_get_recv_len( void )
{
    CTRL_DESC * ctrl_desc;
    CTRL_BUF  * ctrl_buf;
    UINT16      len = 0;

    ctrl_buf  = &g_ctrl_buf;       /* driver's infomation structure */
    ctrl_desc = ctrl_buf->rx_read; /* RX descriptor start pointer */

    if((ctrl_desc->value & (DESC_USED | DESC_RINFO)) !=
       (DESC_USED | DESC_RINFO) ) /* check U & S bits */
    {
       if ( ctrl_desc->value & DESC_ERROR )
       {
          /* data buffer access error */
          DBG_PRINT("data buffer access error\r\n");
       }
       if ( ctrl_desc->value & DESC_OVER  )
       {
          /* data buffer overflow */
          DBG_PRINT("data buffer overflow\r\n");
       }
       __nop();
    }
    else
    {
       len = (UINT16)(ctrl_desc->value & 0x0000FFFF); /* length */
    }
    return len;
}
```

(5) ether_recv

Function name	ether_recv
C language coding format	UINT8* ether_recv(void)
Parameters	None
Return value	Address of the data buffer to which receive data is DMA-transferred.
Processing overview	This function returns the address of the data buffer to which receive data is DMA-transferred.For the next receive DMA, this function updates the descriptor information and calls the ether_recv_start() function. When doing this, if the descriptor is a link pointer, this function provides a link to the link destination address.

Link 1.21 ether_recv

```
UINT8* ether_recv( void )
{
    CTRL_DESC * ctrl_desc;
    CTRL_BUF  * ctrl_buf;
    UINT8     * recv_buf;

    ctrl_buf  = &g_ctrl_buf;       /* driver's infomation structure */
    ctrl_desc = ctrl_buf->rx_read; /* RX descriptor start pointer   */

    recv_buf         = (UINT8*)ctrl_desc->addr; /* RX data buffer address */
    ctrl_desc->value  = DESC_NONE + (FRAME_SIZE + DMA_SUM_SIZE);
    ctrl_buf->rx_read = ctrl_desc + 1;          /* next RX descriptor pointer++ */
    ctrl_desc         = ctrl_buf->rx_read;

    while ( ctrl_desc->value & DESC_LINK )       /* case LINK pointer, link to next */
    {
        ctrl_desc = (CTRL_DESC*)( ctrl_desc->addr );
    }

    ctrl_buf->rx_read = ctrl_desc; /* Next RX DMA descriptor pointer */
    ether_recv_start( ctrl_buf );     /* RX start                    */
    return recv_buf;
}
```

(6) ether_recv_start

Function name	ether_recv_start
C language coding format	void ether_recv_start(CTRL_BUF* ctrl_buf)
Parameters	CTRL_BUF* ctrl_buf: Pointer to CTRL_BUF*.
Return value	None
Processing overview	If ETHA0TRANSCTL.RXEN_STA indicates that receive operation has stopped, this function sets the valid descriptor in the ETHA0RXDP register and restarts the receive operation. If the descriptor is a link pointer, this function provides a link to the link destination address. If the descriptor has yet to be transferred, the receive operation cannot start.

List 1.22 ether_recv_start

```
void ether_recv_start( CTRL_BUF* ctrl_buf )
{
   CTRL_DESC * ctrl_desc;
   UINT32      value;

   if ( !(ETHA0TRANSCTL & TRANSCTL_RXEN_STA)) /* DMA stopped? */
   {
      ctrl_desc = ctrl_buf->rx_read;        /* RX descriptor start address */
      while ( ctrl_desc->value & DESC_LINK ) /* case LINK pointer , link to next */
      {
         ctrl_desc = (CTRL_DESC*)( ctrl_desc->addr );
      }
      ctrl_buf->rx_read = ctrl_desc;        /* RX DMA descriptor start pointer */

      ctrl_desc = ctrl_buf->rx_read;
      if ( !(ctrl_desc->value & DESC_USED))  /* completed ? */
      {
         ETHA0RXDP = (UINT32)ctrl_desc; /* set RX DMA start pointer */
         value     = ETHA0MODE | ETHMODE_RXS;
         ETHA0MODE = value;                  /* RX start */
      }
      else
      {
         __nop();
      }
   }
   else
   {
      __nop();
   }
}
```

(7) ether_get_send_buf

Function name	ether_get_send_buf
C language coding format	UINT8* ether_get_send_buf(void)
Parameters	None
Return value	Address of the data buffer which holds send data
Processing overview	This function returns the address of the data buffer which holds send data. It returns 0 if the descriptor which indicates the data buffer has yet to be transferred.

List 1.23 ether_get_send_buf

```
UINT8* ether_get_send_buf( void )
{
    CTRL_BUF  * ctrl_buf;
    CTRL_DESC * ctrl_desc;
    UINT8     * send_buf = 0;

    ctrl_buf  = &g_ctrl_buf;       /* driver's infomation structure */
    ctrl_desc = ctrl_buf->tx_next; /* TX descriptor start address   */

    if ( ctrl_desc->value & DESC_USED ) /* completed ? */
    {
        send_buf = (UINT8*)&tx_dma_buf[ctrl_buf->tx_next_buf];
    }
    else
    {
        send_buf = 0;                  /* buffer empty */
    }
    return send_buf;
}
```

(8) ether_set_buf

Function name	ether_set_buf
C language coding format	void ether_set_buf(UINT8* send_buf , UINT16 len , UINT16 last)
Parameters	UINT8* send_buf: Address of the data buffer which holds send data UINT16 len: Send data byte length UINT16 last: End-of-packet flag
Return value	None
Processing overview	This function sets both the data buffer address and send data byte length in the descriptor. If the descriptor is a link pointer, this function provides a link to the link destination. If the end of the packet is indicated, this function sets the E bit to 1. After setting the descriptor, it updates the next descriptor location and data buffer location.

List 1.24 ether_set_buf

```
void ether_set_buf( UINT8* send_buf , UINT16 len , UINT16 last )
{
    CTRL_BUF  * ctrl_buf;
    CTRL_DESC * ctrl_desc;

    ctrl_buf  = &g_ctrl_buf;        /* driver's infomation structure   */
    ctrl_desc = ctrl_buf->tx_write; /* TX DMA descriptor write address */

    while ( ctrl_desc->value == DESC_LINK )  /* case LINK pointer, link to next */
    {
        ctrl_desc = (CTRL_DESC*)( ctrl_desc->addr );
    }
    ctrl_buf->tx_write = ctrl_desc;     /* TX DMA descriptor write address */

    ctrl_desc->addr = (UINT8*)send_buf; /* TX data buffer address    */
    if (last == LAST_DATA)              /* LAST packet data ?        */
    {
        ctrl_desc->value = DESC_LAST + len; /* LAST packet = 40000000H */
    }
    else
    {
        ctrl_desc->value = len;         /* continues packet data    */
    }
    ctrl_buf->tx_write = ctrl_desc + 1; /* next TX descriptor write address */

    ctrl_buf->tx_next_buf++;            /* next TX data buffer address */
    if ( ctrl_buf->tx_next_buf > (TX_BUFFER_MAX-1))
    {                                   /* must not exceed (TX_BUFFER_MAX-1)*/
        ctrl_buf->tx_next_buf = 0;
    }
}
```

(9) ether_send

Function name	ether_send
C language coding format	void ether_send(void)
Parameters	None
Return value	None
Processing overview	This function sets the descriptor as end-of-chain and calls ether_send_start(). If the descriptor is a link pointer, it provides a link to the link destination. After setting end-of-chain, it corrects the next descriptor location.

List 1.25 ether_send

```
void ether_send( void )
{
    CTRL_BUF  * ctrl_buf;
    CTRL_DESC * ctrl_desc;

    ctrl_buf  = &g_ctrl_buf;        /* driver's infomation structure    */
    ctrl_desc = ctrl_buf->tx_write; /* TX DMA descriptor address        */

    /* EOC write address */
    while ( ctrl_desc->value == DESC_LINK ) /* case LINK pointer , link to next */
    {
        ctrl_desc = (CTRL_DESC*)( ctrl_desc->addr );
    }
    ctrl_buf->tx_write = ctrl_desc;       /* TX DMA descriptor EOC write address */

    /* make EOC */
    ctrl_desc->addr    = 0;               /* fixed 0                  */
    ctrl_desc->value   = DESC_EOC;        /* EOC = C0000000H          */
    ctrl_buf->tx_write = ctrl_desc + 1;   /* TX DMA descriptor write address */
    ether_send_start((CTRL_BUF *)ctrl_buf ); /* TX DMA start          */
}
```

(10) ether_send_start

Function name	ether_send_start
C language coding format	void ether_send_start(CTRL_BUF* ctrl_buf)
Parameters	CTRL_BUF* ctrl_buf: Pointer to CTRL_BUF
Return value	None
Processing overview	If ETHA0TRANSCTL.TXEN_STA indicates that send operation has stopped, this function sets the valid descriptor in the ETHA0TXDP register and restarts the send operation. After the start of the send operation, this function corrects the next descriptor location. If the descriptor is a link pointer, this function provides a link to the link destination address.

List 1.26 ether_send_start

```
void ether_send_start( CTRL_BUF* ctrl_buf )
{
   CTRL_DESC * ctrl_desc;
   UINT32      value;

   if ( !(ETHA0TRANSCTL & TRANSCTL_TXEN_STA)) /* DMA stopped?            */
   {
      ctrl_desc = ctrl_buf->tx_next;      /* TX descriptor address    */
      if ( ctrl_desc->value == DESC_USED )   /* check U bit             */
      {
         __nop();
      }
      else
      {
         ETHA0TXDP = (UINT32)ctrl_desc; /* TX descriptor address set*/
         value     = ETHA0MODE | ETHMODE_TXS;
         ETHA0MODE = value;            /* TX start                  */
                                 /* CPU->start of descriptor analysis */

         ctrl_desc = ctrl_buf->tx_write;         /* next TX buffer addres          */
         while ( ctrl_desc->value == DESC_LINK ) /* case LINK pointer, link to next */
         {
            ctrl_desc = (CTRL_DESC*)( ctrl_desc->addr );
         }
         ctrl_buf->tx_next = ctrl_desc;          /* next TX descriptor address      */
      }
   }
   else
   {
      __nop();
   }
}
```

(11) int_INTETHA0SRX_hdri

Function name	int_INTETHA0SRX_hdr
C language coding format	void int_INTETHA0SRX_hdr(void)
Parameters	None
Return value	None
Processing overview	This function processes Ethernet receive data ready interrupts. This sample driver does not handle INTETHA0SRX.

List 1.27 int_INTETHA0SRX_hdr

```
#ifdef CNET_NOSYS
#pragma interrupt INTETHA0SRX int_INTETHA0SRX_hdr
#endif
void int_INTETHA0SRX_hdr( void )
{
   ICETHA0SRX = ICETHA0SRX & 0xefff;   /* Interrupt factor clear */
}
```

Remarks: For information about coding with the #pragma directive, refer to the "CA850 Ver.3.20 User's Manual, C Compiler Package, C Language (U18513EJ)."

The interrupt request names are registered in the device file. These names appear in the "V850E2/MN4 User's Manual, Hardware (R01UH0011EJ)."

(12) int_INTETHA0SCRXTX_hdri

Function name	int_INTETHA0SCRXTX_hdr
C language coding format	void int_INTETHA0SCRXTX_hdr(void)
Parameters	None
Return value	None
Processing overview	This function processes the Ethernet packet reception interrupt (INTETHA0SCRX) and Ethernet packet send interrupt (INTETHA0SCTX). INTETHA0SCRX can be asserted due to any one of the following: · Interrupt (RXI) upon completion of packet reception (DMA) · Receive (DMA) end-of-chain interrupt (RECI) · Interrupt (RBEI) upon receive data buffer access error · Pause interrupt (RUPI) with the receive descriptor's Used bit INTETHA0SCTX can be asserted due to any one of the following: · Interrupt (TXI) upon completion of packet transmission (DMA) · Send (DMA) end-of-chain interrupt (TECI) · Interrupt (TBEI) upon send data buffer access error · Pause interrupt (TUPI) with the send descriptor's Used bit All the interrupt sources are reflected in the ETHA0INTMS register.

List 1.28 int_INTETHA0SCRXTX_hdr (1/2)

```
#ifdef CNET_NOSYS
#pragma  interrupt  INTETHA0SCRX  int_INTETHA0SCRXTX_hdr
#pragma  interrupt  INTETHA0SCTX  int_INTETHA0SCRXTX_hdr
#endif
void int_INTETHA0SCRXTX_hdr( void )
{
   CTRL_BUF  * ctrl_buf;
   CTRL_DESC * ctrl_desc;
   UINT32      int_sts;

   ctrl_buf    = &g_ctrl_buf;
   /*-------------------------*/
   /* interrupt factor        */
   /*-------------------------*/
   int_sts    = ETHA0INTMS;       /* Interrupt Register (RC)    */

   /* Interrupt factor clear */
   if( ICETHA0SCRX & 0x1000 )
   {
      ICETHA0SCRX = ICETHA0SCRX & 0xefff;
   }
   if( ICETHA0SCTX & 0x1000 )
   {
      ICETHA0SCTX = ICETHA0SCTX & 0xefff;
   }

   /*-------------------------*/
   /* RX interrupt analysis    */
   /*-------------------------*/
   if ( int_sts & INTMS_RUPI ) {   /* unused                    */
      DBG_PRINT((''  [int] RUPI\r\n''));
   }

   if ( int_sts & INTMS_RBEI )     /* RX data buffer access error */
   {                               /* RX stop , and TRANSCTL.RXEN_STA is cleared */
      DBG_PRINT((''  [int] RBEI\r\n''));
   }
```

List 1.29 int_INTETHA0SCRXTX_hdr (2/2)

```
    if ( int_sts & INTMS_RECI )     /* RX end of chain            */
    {                               /* RX stop , and TRANSCTL.RXEN_STA is cleared */
        DBG_PRINT((" [int] RECI\r\n"));
    }

    if ( int_sts & INTMS_RXI  )     /* RX DMA complete            */
    {
        DBG_PRINT((" [int] RXI\r\n"));
    }
    /*-------------------------*/
    /* TX interrupt analysis   */
    /*-------------------------*/
    if ( int_sts & INTMS_TUPI )     /* unused                     */
    {
        DBG_PRINT((" [int] TUPI\r\n"));
    }

    if ( int_sts & INTMS_TBEI )     /* TX data buffer access error    */
    {                               /* TX stop , and TRANSCTL.TXEN_STA is cleared */
        DBG_PRINT((" [int] TBEI\r\n"));
    }

    if ( int_sts & INTMS_TECI )     /* TX end of chain            */
    {                               /* TX stop , and TRANSCTL.TXEN_STA is cleared */
        DBG_PRINT((" [int] TECI\r\n"));
        DBG_PRINT((" LSTTXDP[%08X]\r\n",LSTTXDP));
                                    /* TX descriptor cleaers */
        ctrl_desc       = (CTRL_DESC*)ETHA0LSTTXDP;
        ctrl_desc->value = DESC_USED;
        ctrl_desc       = (CTRL_DESC*)(ETHA0LSTTXDP & 0x0028FFF0);
        ctrl_desc->value = DESC_USED;
        ctrl_desc->addr  = 0;
        ether_send_start( ctrl_buf );
    }

    if ( int_sts & INTMS_TXI )      /* TX DMA complete            */
    {
        DBG_PRINT((" [int] TXI\r\n"));
        DBG_PRINT((" LSTTXDP[%08X]\r\n",LSTTXDP));
    }
#ifndef CNET_NOSYS
    cnet_ready(&waitid);
#endif
}
```

(13) int_INTETHA0FS_hdr

Function name	int_INTETHA0FS_hdr
C language coding format	void int_INTETHA0FS_hdr(void)
Parameters	None
Return value	None
Processing overview	This function processes the Ethernet FIFO status interrupt (INTETHA0FS). At this time, this sample driver does nothing except analyze the interrupt source. This sample driver is not responsible for processing the analysis results because this processing depends on the system.

List 1.30 int_INTETHA0FS_hdr (1/2)

```
#ifdef CNET_NOSYS
#pragma interrupt INTETHA0FS int_INTETHA0FS_hdr
#endif
void int_INTETHA0FS_hdr( void )
{
    UINT32 int_sts;

    /*--------------------------*/
    /* interrupt factor         */
    /*--------------------------*/
    int_sts    = ETHA0FSTATUS;          /* FIFO status interrupt Register (RC)    */

    ICETHA0FS = ICETHA0FS & 0xefff;     /* Interrupt factor clear */

    /*--------------------------*/
    /* interrupt analysis       */
    /*--------------------------*/
    if ( int_sts & FSTATUS_TACOF )     /* TXABTCNT register overflow    */
    {
        DBG_PRINT((" [INTETMFS][Reg:TXABTCNT] overflow\r\n"));
    }

    if ( int_sts & FSTATUS_RACOF )     /* RXABTCNT register overflow    */
    {
        DBG_PRINT((" [INTETMFS][Reg:RXABTCNT] overflow\r\n"));
    }

    if ( int_sts & FSTATUS_TSUP )       /* update TXSTMONI1 and TXSTMONI2*/
    {
        DBG_PRINT((" [INTETMFS][Reg:TXSTMONI1/TXSTMONI2] update\r\n"));
        DBG_PRINT((" [INTETMFS][Reg:TXSTMONI1]:%08Xh\r\n",TXSTMONI1));
        DBG_PRINT((" [INTETMFS][Reg:TXSTMONI2]:%08Xh\r\n",TXSTMONI2));
    }
```

List 1.31 int_INTETHA0FS_hdr (2/2)

```
    if ( int_sts & FSTATUS_TFNRTY )     /* TX FAIL                      */
    {
        DBG_PRINT((" [INTETMFS]TX FIFO Abort\r\n"));
    }

    if ( int_sts & FSTATUS_TFWE )       /* TX FIFO write error          */
    {
        DBG_PRINT((" [INTETMFS]TX FIFO Write Error\r\n"));
    }

    if ( int_sts & FSTATUS_RFFE )       /* RX FIFO info Error           */
    {
        DBG_PRINT((" [INTETMFS]RX FIFO info Error\r\n"));
    }

    if ( int_sts & FSTATUS_RSUP )       /* RXSTMONI register update     */
    {
        DBG_PRINT((" [INTETMFS][Reg:RXSTMONI] update\r\n"));
        DBG_PRINT((" [INTETMFS][Reg:RXSTMONI]:%08Xh\r\n",RXSTMONI));
    }

    if ( int_sts & FSTATUS_RFWE )       /* received packet, under 4 byte */
    {
        DBG_PRINT((" [INTETMFS]RX FIFO Write Error\r\n"));
    }

    if ( int_sts & FSTATUS_RFOF )       /* RX FIFO overflow             */
    {
        DBG_PRINT((" [INTETMFS]RX FIFO overflow\r\n"));
    }

    if ( int_sts & FSTATUS_RFFLW )      /* FIFO > FLOWTHRSH.FLOWTHR     */
    {
        DBG_PRINT((" [INTETMFS]RX FIFO Capacity more than a set point\r\n"));
    }

    if ( int_sts & FSTATUS_RFZP )       /* FIFO > FLOWTHRSH.ZPTHR       */
    {
        DBG_PRINT((" [INTETMFS]RX FIFO capacity more than a set point\r\n"));
    }
#ifndef CNET_NOSYS
    cnet_ready(&waitid);
#endif
}
```

(14) int_INTETHA0TS_hdr

Function name	int_INTETHA0TS_hdr
C language coding format	void int_INTETHA0TS_hdr(void)
Parameters	None
Return value	None
Processing overview	This function processes the Ethernet send status interrupt (INTETHA0TS). At this time, this sample driver does nothing except analyze the interrupt source. This sample driver is not responsible for processing the analysis results because this processing depends on the system.

List 1.32 int_INTETHA0TS_hdr (1/2)

```
#ifdef CNET_NOSYS
#pragma interrupt INTETHA0TS int_INTETHA0TS_hdr
#endif
void int_INTETHA0TS_hdr( void )
{
   UINT32 int_sts;

   /*--------------------------*/
   /* interrupt factor        */
   /*--------------------------*/
   int_sts = ETHA0TXSTATUS;       /* TX status interrupt Register (RC)    */

   ICETHA0TS = ICETHA0TS & 0xefff; /* Interrupt factor clear */

   /*--------------------------*/
   /* interrupt analysis      */
   /*--------------------------*/
   if ( int_sts & TXSTATUS_TAB )   /* TX abort                    */
   {
      DBG_PRINT((" [INTETMTS]TX abort\r\n"));
   }

   if ( int_sts & TXSTATUS_TGNT )  /* TAB factor                  */
   {
      DBG_PRINT((" [INTETMTS]transfer packet over LMAX\r\n"));
   }

   if ( int_sts & TXSTATUS_LCOL )  /* TAB factor                  */
   {
      DBG_PRINT((" [INTETMTS]late collision\r\n"));
   }
```

List 1.33 int_INTETHA0TS_hdr (2/2)

```
    if ( int_sts & TXSTATUS_ECOL )  /* TAB factor                  */
    {
        DBG_PRINT((''  [INTETMTS]MAX collision over\r\n''));
    }

    if ( int_sts & TXSTATUS_TEDFR ) /* TAB factor                  */
    {
        DBG_PRINT((''  [INTETMTS]late transfer(surplus)\r\n''));
    }

    if ( int_sts & TXSTATUS_TDFR )  /* late transfer               */
    {
        DBG_PRINT((''  [INTETMTS]late transfer\r\n''));
    }

    if ( int_sts & TXSTATUS_TFLOR ) /* VLAN packet pause control frame  */
    {                               /* case MACC1.FLCHT = 0 ,not set    */
        DBG_PRINT((''  [INTETMTS]length field over 1500 byte\r\n''));
    }

    if ( int_sts & TXSTATUS_TFLER ) /* case MACC1.FLCHT = 0 ,not set*/
    {
        DBG_PRINT((''  [INTETMTS]length field != data field\r\n''));
    }

    if ( int_sts & TXSTATUS_TCRCE ) /* MACC1.PADEN = 0 , CRCEN = 0    */
    {
        DBG_PRINT((''  [INTETMTS]CRC error\r\n''));
    }
#ifndef CNET_NOSYS
    cnet_ready(&waitid);
#endif
}
```

(15) int_INTETHA0RS_hdr

Function name	int_INTETHA0RS_hdr
C language coding format	void int_INTETHA0RS_hdr(void)
Parameters	None
Return value	None
Processing overview	This function processes the Ethernet receive status interrupt (INTETHA0RS). At this time, this sample driver does nothing except analyze the interrupt source. This sample driver is not responsible for processing the analysis results because this processing depends on the system.

List 1.34 int_INTETHA0RS_hdr (1/3)

```
#ifdef CNET_NOSYS
#pragma interrupt INTETHA0RS int_INTETHA0RS_hdr
#endif
void int_INTETHA0RS_hdr( void )
{
   UINT32 int_sts;

   /*--------------------------*/
   /* interrupt factor        */
   /*--------------------------*/
   int_sts = ETHA0RXSTATUS;    /* RX status interrupt Register (RC)    */

   ICETHA0RS = ICETHA0RS & 0xefff; /* Interrupt factor clear */

   /*--------------------------*/
   /* interrupt analysis      */
   /*--------------------------*/
   if ( int_sts & RXSTATUS_RLENE ) /* RX packet length error       */
   {
      DBG_PRINT((" [INTETMRS]RLENE\r\n"));
   }

   if ( int_sts & RXSTATUS_VLAN )  /* VLAN packet received         */
   {                          /* case CRC error , not set     */
      DBG_PRINT((" [INTETMRS]VLAN\r\n"));
   }

   if ( int_sts & RXSTATUS_USOP )  /* undef.opecode control packet received */
   {                          /* case CRC error , not set     */
      DBG_PRINT((" [INTETMRS]USOP\r\n"));
   }
```

List 1.35 int_INTETMRS_hdr (2/3)

```
    if ( int_sts & RXSTATUS_RPCF )  /* pause control packet received*/
    {                           /* case CRC error , not set     */
       DBG_PRINT((" [INTETMRS]RPCF\r\n"));
    }

    if ( int_sts & RXSTATUS_RCFR )  /* control packet received      */
    {                           /* case CRC error , not set     */
       DBG_PRINT((" [INTETMRS]RCFR\r\n"));
    }

    if ( int_sts & RXSTATUS_DBNB )  /* dribble nibble received      */
    {
       DBG_PRINT((" [INTETMRS]DBNB\r\n"));
    }

    if ( int_sts & RXSTATUS_RLOR )  /* length field > 1500          */
    {                           /* case MACC1.FLCHT = 0 ,not set*/
       DBG_PRINT((" [INTETMRS]RLOR\r\n"));
    }

    if ( int_sts & RXSTATUS_RLER )  /* length field != data field   */
    {
       DBG_PRINT((" [INTETMRS]RLER\r\n"));
    }

    if ( int_sts & RXSTATUS_RCRCE ) /* receive CRC error            */
    {
       DBG_PRINT((" [INTETMRS]RCRCE\r\n"));
    }

    if ( int_sts & RXSTATUS_RXER )  /* RXER                         */
    {
       DBG_PRINT((" [INTETMRS]RXER\r\n"));
    }

    if ( int_sts & RXSTATUS_CEPS )  /* False Carrier                */
    {
       DBG_PRINT((" [INTETMRS]CEPS\r\n"));
    }

    if ( int_sts & RXSTATUS_REPS )  /* preamble + SFD 1nibble packet*/
    {
       DBG_PRINT((" [INTETMRS]REPS\r\n"));
    }
```

List 1.36 int_INTETMRS_hdr (3/3)

```
    if ( int_sts & RXSTATUS_PAIG )  /* carrier error, preamble error*/
    {
        DBG_PRINT((" [INTETMRS]PAIG\r\n"));
    }

    if ( int_sts & RXSTATUS_TXRX )  /* ransmission of a message during the HDX reception*/
    {
        DBG_PRINT((" [INTETMRS]TXRX\r\n"));
    }

    if ( int_sts & RXSTATUS_DVCF )  /* packet receved              */
    {
        DBG_PRINT((" [INTETMRS]DVCF\r\n"));
    }
#ifndef CNET_NOSYS
    cnet_ready(&waitid);
#endif
}
```

(16) int_INTETHA0MACV_hdr

Function name	int_INTETHA0MACV_hdr
C language coding format	void int_INTETHA0MACV_hdr(void)
Parameters	None
Return value	None
Processing overview	This function processes the Ethernet MAC interrupt (INTETMOV). At this time, this sample driver does nothing except analyze the interrupt source. This sample driver is not responsible for processing the analysis results because this processing depends on the system.

List 1.37 int_INTETHA0MACV_hdr (1/5)

```
#ifdef CNET_NOSYS
#pragma interrupt INTETHA0MAC int_INTETHA0MACV_hdr
#endif
void int_INTETHA0MACV_hdr( void )
{
   UINT32 int_sts_car1;
   UINT32 int_sts_car2;

   /*--------------------------*/
   /* interrupt factor        */
   /*--------------------------*/
   int_sts_car1 = ETHA0CAR1;   /* carry1 Register (RC)          */
   int_sts_car2 = ETHA0CAR2;   /* carry2 Register (RC)          */

   ICETHA0MAC = ICETHA0MAC & 0xefff;   /* Interrupt factor clear */

   /*--------------------------*/
   /* CAR1 interrupt analysis  */
   /*--------------------------*/
   if ( int_sts_car1 & CAR1_C1VT )
   {
      DBG_PRINT((" [INTETMOV][Reg:RVBT] overflow\r\n"));
   }
   if ( int_sts_car1 & CAR1_C1UT )
   {
      DBG_PRINT((" [INTETMOV][Reg:TUCA] overflow\r\n"));
   }
   if ( int_sts_car1 & CAR1_C1BT )
   {
      DBG_PRINT((" [INTETMOV][Reg:TBCA] overflow\r\n"));
   }
   if ( int_sts_car1 & CAR1_C1MT )
   {
      DBG_PRINT((" [INTETMOV][Reg:TMCA] overflow\r\n"));
   }
```

List 1.38 int_INTETHA0MACV_hdr (2/5)

```
    if ( int_sts_car1 & CAR1_C1PT )
    {
        DBG_PRINT((" [INTETMOV][Reg:TPCT] overflow\r\n"));
    }
    if ( int_sts_car1 & CAR1_C1TB )
    {
        DBG_PRINT((" [INTETMOV][Reg:TBYT] overflow\r\n"));
    }
    if ( int_sts_car1 & CAR1_C1MX )
    {
        DBG_PRINT((" [INTETMOV][Reg:RMAX] overflow\r\n"));
    }
    if ( int_sts_car1 & CAR1_C11K )
    {
        DBG_PRINT((" [INTETMOV][Reg:R1K] overflow\r\n"));
    }
    if ( int_sts_car1 & CAR1_C1FE )
    {
        DBG_PRINT((" [INTETMOV][Reg:R511] overflow\r\n"));
    }
    if ( int_sts_car1 & CAR1_C1TF )
    {
        DBG_PRINT((" [INTETMOV][Reg:R255] overflow\r\n"));
    }
    if ( int_sts_car1 & CAR1_C1OT )
    {
        DBG_PRINT((" [INTETMOV][Reg:R127] overflow\r\n"));
    }
    if ( int_sts_car1 & CAR1_C1SF )
    {
        DBG_PRINT((" [Reg:R64] overflow\r\n"));
    }
    if ( int_sts_car1 & CAR1_C1BR )
    {
        DBG_PRINT((" [Reg:RBCA] overflow\r\n"));
    }
    if ( int_sts_car1 & CAR1_C1MR )
    {
        DBG_PRINT((" [INTETMOV][Reg:RMCA] overflow\r\n"));
    }
    if ( int_sts_car1 & CAR1_C1PR )
    {
        DBG_PRINT((" [INTETMOV][Reg:RPKT] overflow\r\n"));
    }
```

List 1.39 int_INTETHA0MACV_hdr (3/5)

```
    if ( int_sts_car1 & CAR1_C1RB )
    {
        DBG_PRINT((" [INTETMOV][Reg:RBYT] overflow\r\n"));
    }

    /*--------------------------*/
    /* CAR2 interrupt analysis  */
    /*--------------------------*/
    if ( int_sts_car2 & CAR2_C2DV )
    {
        DBG_PRINT((" [INTETMOV]status vector overrun\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2IM )
    {
        DBG_PRINT((" [INTETMOV][Reg:TIME] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2CS )
    {
        DBG_PRINT((" [INTETMOV][Reg:TCSE] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2NC )
    {
        DBG_PRINT((" [INTETMOV][Reg:TNCL] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2XC )
    {
        DBG_PRINT((" [INTETMOV][Reg:TXCL] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2LC )
    {
        DBG_PRINT((" [INTETMOV][Reg:TLCL] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2MC )
    {
        DBG_PRINT((" [INTETMOV][Reg:TMCL] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2SC )
    {
        DBG_PRINT((" [INTETMOV][Reg:TSCL] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2XD )
    {
        DBG_PRINT((" [INTETMOV][Reg:TXDF] overflow\r\n"));
    }
```

List 1.40 int_INTETHA0MACV_hdr (4/5)

```
    if ( int_sts_car2 & CAR2_C2DF )
    {
        DBG_PRINT((" [INTETMOV][Reg:TDFR] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2XF )
    {
        DBG_PRINT((" [INTETMOV][Reg:TXPF] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2TE )
    {
        DBG_PRINT((" [INTETMOV][Reg:TFCS] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2JB )
    {
        DBG_PRINT((" [INTETMOV][Reg:RBJR] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2FG )
    {
        DBG_PRINT((" [INTETMOV][Reg:RFRG] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2OV )
    {
        DBG_PRINT((" [INTETMOV][Reg:ROVR] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2UN )
    {
        DBG_PRINT((" [INTETMOV][Reg:RUND] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2FC )
    {
        DBG_PRINT((" [INTETMOV][Reg:RFCR] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2CD )
    {
        DBG_PRINT((" [INTETMOV][Reg:RCDE] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2FO )
    {
        DBG_PRINT((" [INTETMOV][Reg:RFLR] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2AL )
    {
        DBG_PRINT((" [INTETMOV][Reg:RALN] overflow\r\n"));
    }
```

List 1.41 int_INTETHA0MACV_hdr (5/5)

```
    if ( int_sts_car2 & CAR2_C2UO )
    {
        DBG_PRINT("  [INTETMOV][Reg:RXUO] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2PF )
    {
        DBG_PRINT("  [INTETMOV][Reg:RXPF] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2CF )
    {
        DBG_PRINT("  [INTETMOV][Reg:RXCF] overflow\r\n"));
    }
    if ( int_sts_car2 & CAR2_C2RE )
    {
        DBG_PRINT("  [INTETMOV][Reg:RFCS] overflow\r\n"));
    }
#ifndef CNET_NOSYS
    cnet_ready(&waitid);
#endif
}
```

1.4.6 Example of Reception Processing

Below is an example of processing for receiving packets with this sample driver.

Overview:

The higher-level applications for the Ethernet driver, which include the IP stack, perform reception processing by polling.

When data is received, it is stored in a required portion of memory allocated from the memory space (CPU internal memory) managed by the application.

List 1.42 Example of Reception Processing

```
{
   struct pbuf * p = (struct pbuf *)0;
   u16_t        idx;
   static u16_t  recv_len;
   u8_t *       recv_buf;

#ifndef CNET_NOSYS
   sys_thread_disable_dispatch();
#endif    /* !CNET_NOSYS */

   /* Read Packet */
   while( (recv_len=ether_get_recv_len()) > 0 )                    (1)
   {
      p = pbuf_alloc(PBUF_LINK, (u16_t)recv_len , PBUF_POOL);      (2)
      if (p == (struct pbuf *)0)
      {
         /* Error */
         // not support
         DBG_PRINT(("PBUF ERROR\n"));
      }
      recv_buf = (u8_t*)ether_recv();                              (3)
      eth_memcpy( p->payload, recv_buf, recv_len );                (4)
```

(1) Obtain a receive DMA transfer byte length from the ether_get_recv_len() function upon completion of the receive DMA. If the receive DMA does not complete, this function returns 0.

(2) Reserve Allocate an amount of application memory equal to the transfer byte length returned by the ether_get_recv_len() function.

(3) Obtain the destination data buffer address (in the H-bus shared memory space) from the ether_recv() function for the receive DMA.

(4) Copy data for the DMA transfer size from the data buffer address (in the H-bus shared memory space) where receive data is stored to the application memory (CPU internal memory space). Then, pass the receive packet data to the higher-level application.

Remarks: When packets are sent or received, the applications should not directly handle packet data in the H-bus shared memory for purposes such as packet data generation and analysis. This is because doing so decreases the memory access speed.

1.4.7 Example of Transmission Processing

Below is an example of processing for sending packets with this sample driver.

Overview:

This sample driver sends an Ethernet frame when it is called from one of the higher-level applications for the Ethernet driver, such as the IP stack.

If the send data buffer is full, this sample driver waits indefinitely until the send data buffer has room. If it has room, this sample driver copies send data from the memory space (CPU internal memory) managed by the application to the send data buffer and invokes the Ethernet transmission process.

List 1.43 Example of Transmission Processing

```
{
   u16_t  len;
   u16_t  payload_len = 0;
   u8_t * send_buf;

#ifndef CNET_NOSYS
   sys_thread_disable_dispatch();
#endif    /* !CNET_NOSYS */

   send_buf=(u8_t*)ether_get_send_buf();                                          (1)
   while ( send_buf == 0 )
   {
      DBG_PRINT(("TX buffer empty , wait\r\n"));
   }

   while (p != (struct pbuf *)0)
   {
      len = p->len;
      eth_memcpy((void *)(((int)(send_buf)) + payload_len), (void *)(p->payload),(int)len);   (2)
      payload_len += len;                                                         (3)
      p = p->next;
   }

   ether_set_buf( send_buf, payload_len , LAST_DATA );                            (4)
   ether_send();                                                                  (5)
```

(1) Obtain the send data buffer address (in the H-bus shared memory space) of the send packet data from the ether_get_send_buf() function.
If the send data buffer is full, this function returns 0.

(2) Copy the data from the application memory (CPU internal memory) to the data buffer address (in the H-bus shared memory) returned by the ether_get_send_buf() function.

(3) payload_len indicates the send packet length in bytes.

(4) Pass the data buffer address (in the H-bus shared memory) of the send packet data and send byte length to the ether_set_buf() function.
At this time, set the last buffer flag in the descriptor for the data buffer address by specifying LAST_DATA for the third parameter.
To set split packets or multiple packets at the same time, specify CNTN_DATA for the third parameter and call the ether_set_buf() function as many times as necessary.(Note)

(5) Send the packet data with the ether_send() function.

Note: Like the sample code above, this sample driver handles the number of descriptors and data buffers and uses end-of-chain to release the descriptors after their use assuming that send packets are not continuous.

1.5 Definitions of V850E2/MN4 Ethernet Controller Bits

The lists below show the V850E2/MN4 Ethernet controller bit names defined by this sample driver.

These definitions are included in header file \ether_register.h.

Caution:

The bit names defined for this sample driver are different from those shown in the “V850E2/MN4 User’s Manual, Hardware (R01UH0011EJ).” When using the bit names, refer to this the user’s manual.

List 1.44 Register List (1/12)

Register name	BIT 31 (defined value)	BIT 30 (defined value)	BIT 29 (defined value)	BIT 28 (defined value)	BIT 27 (defined value)	BIT 26 (defined value)	BIT 25 (defined value)	BIT 24 (defined value)
	BIT 23 (defined value)	BIT 22 (defined value)	BIT 21 (defined value)	BIT 20 (defined value)	BIT 19 (defined value)	BIT 18 (defined value)	BIT 17 (defined value)	BIT 16 (defined value)
	BIT 15 (defined value)	BIT 14 (defined value)	BIT 13 (defined value)	BIT 12 (defined value)	BIT 11 (defined value)	BIT 10 (defined value)	BIT 9 (defined value)	BIT 8 (defined value)
	BIT 7 (defined value)	BIT 6 (defined value)	BIT 5 (defined value)	BIT 4 (defined value)	BIT 3 (defined value)	BIT 2 (defined value)	BIT 1 (defined value)	BIT 0 (defined value)
MIICTL								MIICTL_ MIIEN (0x01)
MACC1								
		MACC1_ MACLB (0x00004000ul)			MACC1_ TXFC (0x00000800ul)	MACC1_ RXFC (0x00000400ul)	MACC1_ SRXEN (0x00000200ul)	MACC1_ PARF (0x00000100ul)
	MACC1_ PUREP (0x00000080ul)	MACC1_ FLCHT (0x00000040ul)	MACC1_ NOBO (0x00000020ul)		MACC1_ CRCEN (0x00000008ul)	MACC1_ PADEN (0x00000004ul)	MACC1_ FULLD (0x00000002ul)	MACC1_ HUGEN (0x00000001ul)
MACC2								
						MACC2_ MCRST (0x00000400ul)	MACC2_ RFRST (0x00000200ul)	MACC2_ TFRST (0x00000100ul)
	MACC2_ BPNB (0x00000040ul)	MACC2_ APD (0x00000020ul)	MACC2_ VPD (0x00000010ul)					
IPGT								
					IPGT_IPGT_MSK (0x0000007Ful)			

List 1.45 Register List (2/12)

IPGR								
					IPGR_IPGR1_MSK (0x00007F00ul)			
					IPGR_IPGR2_MSK (0x0000007Ful)			
CLRT								
			CLRT_LCOL_MSK (0x00003F00ul)					
					CLRT_RETRY_MSK (0x0000000Ful)			
LMAX								
	LMAX_MAXF_MSK (0x0000FFFFul)							
LSA1								
	LSA1_LSA1_MSK (0x0000FFFFul)							
LSA2	LSA2_LSA2_MSK (0xFFFFFFFFul)							
PTVR								
	PTVR_PTCT_MSK (0x0000FFFFul)							
VLTP								
	VLTP_VLTP_MSK (0x0000FFFFul)							

List 1.46 Register List (3/12)

MIIC								
	MIIC _MIRST (0x00008000ul)							
					MIIC_CLKS_50M (0x00000008ul) MIIC_CLKS_33M (0x00000004ul)		PHYSEL (0x00000002ul)	
MCMD								
							MCMD_SCANC (0x00000002ul)	MCMD_RSTAT (0x00000001ul)
MADR								
							MADR_FIAD_MSK (0x00001F00ul)	
							MADR_RGAD_MSK (0x0000001Ful)	
MWTD								
	MWTD_CTLD_MSK (0x0000FFFFul)							
MRDD								
	MRDD_PRSD_MSK (0x0000FFFFul)							
MIND								
						MIND_NVALID (0x00000004ul)	MIND_SCANA (0x00000002ul)	MIND_BUSY (0x00000001ul)
AFR								
					AFR_PRO (0x00000008ul)	AFR_PRM (0x00000004ul)	AFR_AMC (0x00000002ul)	AFR_ABC (0x00000001ul)

List 1.47 Register List (4/12)

HT1								
HT2								
CAR1								
	CAR1_ C1VT (0x00008000ul)	CAR1_ C1UT (0x00004000ul)	CAR1_ C1BT (0x00002000ul)	CAR1_ C1MT (0x00001000ul)	CAR1_ C1PT (0x00000800ul)	CAR1_ C1TB (0x00000400ul)	CAR1_ C1MX (0x00000200ul)	CAR1_ C11K (0x00000100ul)
	CAR1_ C1FE (0x00000080ul)	CAR1_ C1TF (0x00000040ul)	CAR1_ C1OT (0x00000020ul)	CAR1_ C1SF (0x00000010ul)	CAR1_ C1BR (0x00000008ul)	CAR1_ C1MR (0x00000004ul)	CAR1_ C1PR (0x00000002ul)	CAR1_ C1RB (0x00000001ul)
CAR2	CAR2_ C2DV (0x80000000ul)							
		CAR2_ C2IM (0x00400000ul)	CAR2_ C2CS (0x00200000ul)	CAR2_ C2NC (0x00100000ul)	CAR2_ C2XC (0x00080000ul)	CAR2_ C2LC (0x00040000ul)	CAR2_ C2MC (0x00020000ul)	CAR2_ C2Sc (0x00010000ul)
	CAR2_ C2XD (0x00008000ul)	CAR2_ C2DF (0x00004000ul)	CAR2_ C2XF (0x00002000ul)	CAR2_ C2TE (0x00001000ul)	CAR2_ C2JB (0x00000800ul)	CAR2_ C2FG (0x00000400ul)	CAR2_ C2OV (0x00000200ul)	CAR2_ C2UN (0x00000100ul)
	CAR2_ C2FC (0x00000080ul)	CAR2_ C2CD (0x00000040ul)	CAR2_ C2FO (0x00000020ul)	CAR2_ C2AL (0x00000010ul)	CAR2_ C2UO (0x00000008ul)	CAR2_ C2PF (0x00000004ul)	CAR2_ C2CF (0x00000002ul)	CAR2_ C2RE (0x00000001ul)
CAM1								
	CAM1_ M1VT (0x00008000ul)	CAM1_ M1UT (0x00004000ul)	CAM1_ M1BT (0x00002000ul)	CAM1_ M1MT (0x00001000ul)	CAM1_ M1PT (0x00000800ul)	CAM1_ M1TB (0x00000400ul)	CAM1_ M1MX (0x00000200ul)	CAM1_ M11K (0x00000100ul)
	CAM1_ M1FE (0x00000080ul)	CAM1_ M1TF (0x00000040ul)	CAM1_ M1OT (0x00000020ul)	CAM1_ M1SF (0x00000010ul)	CAM1_ M1BR (0x00000008ul)	CAM1_ M1MR (0x00000004ul)	CAM1_ M1PR (0x00000002ul)	CAM1_ M1RB (0x00000001ul)

List 1.48 Register List (5/12)

CAM2	CAM2_ M2DV (0x80000000ul)							
		CAM2_ M2IM (0x00400000ul)	CAM2_ M2CS (0x00200000ul)	CAM2_ M2NC (0x00100000ul)	CAM2_ M2XC (0x00080000ul)	CAM2_ M2LC (0x00040000ul)	CAM2_ M2MC (0x00020000ul)	CAM2_ M2Sc (0x00010000ul)
	CAM2_ M2XD (0x00008000ul)	CAM2_ M2DF (0x00004000ul)	CAM2_ M2XF (0x00002000ul)	CAM2_ M2TE (0x00001000ul)	CAM2_ M2JB (0x00000800ul)	CAM2_ M2FG (0x00000400ul)	CAM2_ M2OV (0x00000200ul)	CAM2_ M2UN (0x00000100ul)
	CAM2_ M2FC (0x00000080ul)	CAM2_ M2CD (0x00000040ul)	CAM2_ M2FO (0x00000020ul)	CAM2_ M2AL (0x00000010ul)	CAM2_ M2UO (0x00000008ul)	CAM2_ M2PF (0x00000004ul)	CAM2_ M2CF (0x00000002ul)	CAM2_ M2RE (0x00000001ul)
RBYT								
RPKT								
RFCS								
RMCA								
RBCA								
RXPF								
RXUO								
RALNF								
RFLR								
RCDE								
RFCR								
RUND								
ROVR								
RFRG								
RJBR								
R64								
R127								
R255								
R511								
R1K								
RMAX								
RVBT								
TBYT								
TPKT								
TFCS								

List 1.49 Register List (6/12)

TUCA								
TXPF								
TDFR								
TXDF								
TSCL								
TMCL								
TLCL								
TXCL								
TNCL								
TCSE								
TIME								
TUCA								
MFFCONT	MFFCONT_ LOOPBACK (0x80000000ul)	MFFCONT_ RCSEL (0x40000000ul)						
						MFFCONT _FLOWCNT (0x00040000ul)	MFFCONT_ IVPAUSE (0x00020000ul)	MFFCONT_ ZEROPAUSE (0x00010000ul)
	MFFCONT _ RXSDMA1 (0x00008000ul)	MFFCONT_ RXSDMA0 (0x00004000ul)		MFFCONT_ ASOE (0x00001000ul)	MFFCONT_ APS (0x00000800ul)	MFFCONT _APL (0x00000400ul)	MFFCONT _RXTHRC (0x00000200ul)	MFFCONT _RXEN (0x00000100ul)
						MFFCONT _TABT (0x00000004ul)	MFFCONT _TXTHRC (0x00000002ul)	MFFCONT _TXEN (0x00000001ul)
RSTCNT								
								RSTCONT_ RFFLSH (0x00000100ul)
								RSTCONT_ TFFLSH (0x00000010ul)
								RSTCONT_ SFTRST (0x00000001ul)
FLOWTHRESH								
	FLOWTHRESH_FLOWTHR_MSK (0x07FF0000ul)							
	FLOWTHRESH_ZPTHR_MSK (0x000007FFul)							

List 1.50 Register List (7/12)

PAUSETM	PAUSETM_FLOWTHR_MSK (0xFFFF0000ul)							
	PAUSETM_ZPTHR_MSK (0x0000FFFFul)							
RXERSEL	RXERSEL_ RLENE (0x80000000ul)	RXERSEL_ VLAN (0x40000000ul)	RXERSEL_ USOP (0x20000000ul)	RXERSEL_ RPCF (0x10000000ul)	RXERSEL_ RCFR (0x08000000ul)	RXERSEL_ DBNB (0x04000000ul)		
		RXERSEL_ RLOR (0x00400000ul)	RXERSEL_ RLER (0x00200000ul)	RXERSEL_ RCRCE (0x00100000ul)	RXERSEL_ RXER (0x00080000ul)	RXERSEL_ CEPS (0x00040000ul)	RXERSEL_ REPS (0x00020000ul)	RXERSEL_ PAIG (0x00010000ul)
							RXERSEL_ TXRX (0x00000002ul)	RXERSEL_ DVCF (0x00000001ul)
TXSTMONI1								
				TXSTMONI1_ CSE (0x00100000ul)	TXSTMONI1_ TBP (0x00080000ul)	TXSTMONI1_ TPP (0x00040000ul)	TXSTMONI1_ TPCF (0x00020000ul)	TXSTMONI1_ TCFR (0x00010000ul)
	TXSTMONI1_TTBC_MSK (0x0000FFFFul)							
TXSTMONI2	TXSTMONI2_ TUDR (0x80000000ul)	TXSTMONI2_ TGNT (0x40000000ul)	TXSTMONI2_ LCOL (0x20000000ul)	TXSTMONI2_ ECOL (0x10000000ul)	TXSTMONI2_ TEDFR (0x08000000ul)	TXSTMONI2_ TDFR (0x04000000ul)	TXSTMONI2_ TBRO (0x02000000ul)	TXSTMONI2_ TMUL (0x01000000ul)
	TXSTMONI2_ TDONE (0x00800000ul)	TXSTMONI2_ TFLOR (0x00400000ul)	TXSTMONI2_ TFLER (0x00200000ul)	TXSTMONI2_ TCRCE (0x00100000ul)	TXSTMONI2_ TCBC_MSK (0x000F0000ul)			
	TXSTMONI2_TBYT_MSK (0x0000FFFFul)							
TXFINF1								
	TXFINF1_TPCNT_MSK (0x01FF0000ul)							
	TXFINF1_TREMAIN_MSK (0x00000FFFul)							
TXFINF2								
								TXFINF2_ TXSTOP (0x00000001ul)

List 1.51 Register List (8/12)

RXSTMONI	RXSTMONI_RLENE (0x80000000ul)	RXSTMONI_VLAN (0x40000000ul)	RXSTMONI_USOP (0x20000000ul)	RXSTMONI_RPCF (0x10000000ul)	RXSTMONI_RCFR (0x08000000ul)	RXSTMONI_DBNB (0x04000000ul)	RXSTMONI_RBRO (0x02000000ul)	RXSTMONI_RMUL (0x01000000ul)
	RXSTMONI_RXOK (0x00800000ul)	RXSTMONI_RLOR (0x00400000ul)	RXSTMONI_RLER (0x00200000ul)	RXSTMONI_RCRCE (0x00100000ul)	RXSTMONI_RXER (0x00080000ul)	RXSTMONI_CEPS (0x00040000ul)	RXSTMONI_REPS (0x00020000ul)	RXSTMONI_PAIG (0x00010000ul)
	RXSTMONI_RBYT_MSK (0x0000FFFFul)							
RXFINF1								
	RXSTMONI_RBYT_MSK (0x0000FFFFul)							
RXFINF2								
	RXFINF2_RPCNT_MSK (0x01FF0000ul)							
	RXFINF2_RREMAIN_MSK (0x00000FFFul)							
RXFINF3								
								RXFINF3_RXSTOP (0x00000001ul)
FSTATUS								FSTATUS_TACOF (0x01000000ul)
								FSTATUS_RACOF (0x00010000ul)
				FSTATUS_TSUP (0x00001000ul)	FSTATUS_TFNRTY (0x00000800ul)	FSTATUS_TFWE (0x00000400ul)		
	FSTATUS_RFFE (0x00000080ul)	FSTATUS_RSUP (0x00000040ul)		FSTATUS_RFWE (0x00000010ul)	FSTATUS_RFOF (0x00000008ul)		FSTATUS_RFFLW (0x00000002ul)	FSTATUS_RFZP (0x00000001ul)

List 1.52 Register List (9/12)

FSTATUS_MASK								FSTATUS_MASK_TACOF (0x01000000ul)
								FSTATUS_MASK_RACOF (0x00010000ul)
				FSTATUS_MASK_TSUP (0x00001000ul)	FSTATUS_MASK_TFNRTY (0x00000800ul)	FSTATUS_MASK_TFWE (0x00000400ul)		
	FSTATUS_MASK_RFFE (0x00000080ul)	FSTATUS_MASK_RSUP (0x00000040ul)		FSTATUS_MASK_RFWE (0x00000010ul)	FSTATUS_MASK_RFOF (0x00000008ul)		FSTATUS_MASK_FFLW (0x00000002ul)	FSTATUS_MASK_RFZP (0x00000001ul)
TXSTATUS								
								TXSTATUS_TAB (0x00010000ul)
	TXSTATUS_TGNT (0x00000080ul)	TXSTATUS_LCOL (0x00000040ul)	TXSTATUS_ECOL (0x00000020ul)	TXSTATUS_TEDFR (0x00000010ul)	TXSTATUS_TDFR (0x00000008ul)	TXSTATUS_TFLOR (0x00000004ul)	TXSTATUS_TFLER (0x00000002ul)	TXSTATUS_TCRCE (0x00000001ul)
TXSTATUS_MASK								
								TXSTATUS_MASK_TAB (0x00010000ul)
	TXSTATUS_MASK_TGNT (0x00000080ul)	TXSTATUS_MASK_LCOL (0x00000040ul)	TXSTATUS_MASK_ECOL (0x00000020ul)	TXSTATUS_MASK_EDFR (0x00000010ul)	TXSTATUS_MASK_TDFR (0x00000008ul)	TXSTATUS_MASK_TFLOR (0x00000004ul)	TXSTATUS_MASK_TFLER (0x00000002ul)	TXSTATUS_MASK_TCRCE (0x00000001ul)
RXSTATUS								
		RXSTATUS_RLENE (0x00004000ul)	RXSTATUS_VLAN (0x00002000ul)	RXSTATUS_USOP (0x00001000ul)	RXSTATUS_RPCF (0x00000800ul)	RXSTATUS_RCFR (0x00000400ul)	RXSTATUS_DBNB (0x00000200ul)	RXSTATUS_RLOR (0x00000100ul)
	RXSTATUS_RLER (0x00000080ul)	RXSTATUS_RCRCE (0x00000040ul)	RXSTATUS_RXER (0x00000020ul)	RXSTATUS_CEPS (0x00000010ul)	RXSTATUS_REPS (0x00000008ul)	RXSTATUS_PAIG (0x00000004ul)	RXSTATUS_TXRX (0x00000002ul)	RXSTATUS_DVCF (0x00000001ul)

List 1.53 Register List (10/12)

RXSTATUS_MASK								
		RXSTATUS_MASK_RLENE (0x00004000ul)	RXSTATUS_MASK_VLAN (0x00002000ul)	RXSTATUS_MASK_USOP (0x00001000ul)	RXSTATUS_MASK_RPCF (0x00000800ul)	RXSTATUS_MASK_RCFR (0x00000400ul)	RXSTATUS_MASK_DBNB (0x00000200ul)	RXSTATUS_MASK_RLOR (0x00000100ul)
	RXSTATUS_MASK_RLER (0x00000080ul)	RXSTATUS_MASK_RCRCE (0x00000040ul)	RXSTATUS_MASK_RXER (0x00000020ul)	RXSTATUS_MASK_CEPS (0x00000010ul)	RXSTATUS_MASK_REPS (0x00000008ul)	RXSTATUS_MASK_PAIG (0x00000004ul)	RXSTATUS_MASK_TXRX (0x00000002ul)	RXSTATUS_MASK_DVCF (0x00000001ul)
TXABTCNT								
	TXABTCNT_TABCNT_MSK (0x0000FFFFul)							
RXABTCNT								
	RXABTCNT_TABCNT_MSK (0x0000FFFFul)							
ETHMODE								
						ETHMODE_RXS (0x00040000ul)	ETHMODE_TXS (0x00020000ul)	
INTMS					INTMS_RUSMSK (0x08000000ul)	INTMS_RBEMSK (0x04000000ul)	INTMS_RECMSK (0x02000000ul)	INTMS_RXMSK (0x01000000ul)
					INTMS_RUPI (0x00080000ul)	INTMS_RBEI (0x00040000ul)	INTMS_RECI (0x00020000ul)	INTMS_RXI (0x00010000ul)
					INTMS_TUSMSK (0x00000800ul)	INTMS_TBEMSK (0x00000400ul)	INTMS_TECMSK (0x00000200ul)	INTMS_TXMSK (0x00000100ul)
					INTMS_TUPI (0x00000008ul)	INTMS_TBEI (0x00000004ul)	INTMS_TECI (0x00000002ul)	INTMS_TXI (0x00000001ul)

List 1.54 Register List (11/12)

TRANSCTL							TRANSCTL_RXEN_ RXEN_STA (0x02000000ul)	TRANSCTL_RXEN_ TXEN_STA (0x01000000ul)
							TRANSCTL_RXEN_ RXEN (0x00020000ul)	TRANSCTL_RXEN_ TXEN (0x00010000ul)
								TRANSCTL_RXEN_ RXCHKSMEN (0x00000001ul)
SFTRST								
								SFTRST_SFTRST (0x00000001ul)
DMACM								
						DMACM_BURST16 (0x00000700ul) DMACM_BURST8 (0x00000500ul) DMACM_BURST4 (0x00000300ul) DMACM_SINGLE (0x00000000ul)	
RXDP								

List 1.55 Register List (12/12)

LSTRXDP	
TXDP	
LSTTXDP	

2. Appendix

2.1 C-NET

The compact TCP/IP Library (C-NET) is a TCP/IP network library for the V850 Series of microcoontrollers. Because it uses only a small amount of the ROM/RAM, a network system can be established with only internal memory.

The C-NET offers application program interface (API) functions for various network services. With these API functions, the user can establish the following network services:

- Network communication via Transmission Control Protocol/Internet Protocol (TCP/IP)
- E-mail transmission via Simple Mail Transfer Protocol (SMTP)
- E-mail reception via Post Office Protocol 3 (POP3)
- Automatic acquisition of network information via Dynamic Host Configuration Protocol (DHCP)
- Hypertext transfer via HyperText Transfer Protocol (HTTP)

The figure below shows the position of the compact TCP/IP Library.

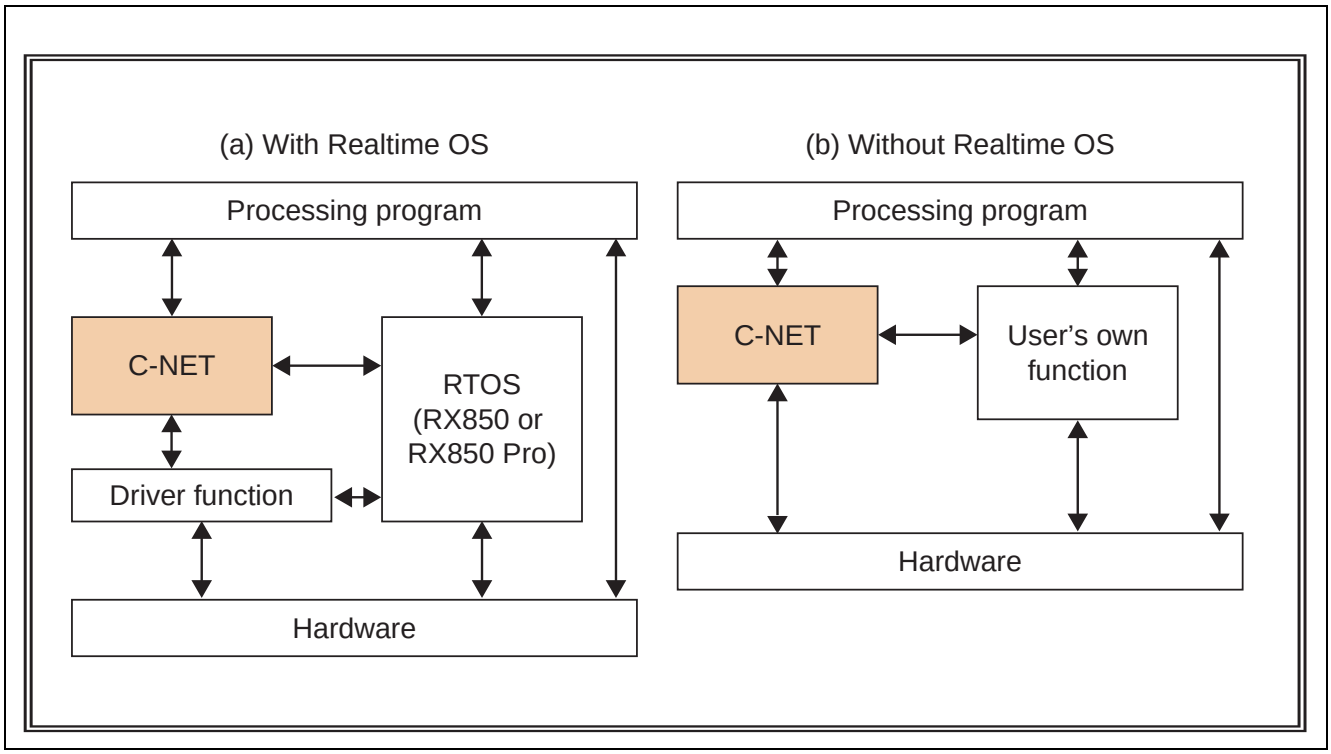


Figure 2.1 Position of the Compact TCP/IP Library

2.2 Initialization of the CPU Built-in Functions and Peripheral Circuits

To use the RTE-V850E2/MN4-EB-S, it is necessary to initialize the CPU built-in functions and peripheral circuits.

This sample driver initializes them with the following functions:

- Functions for basic CPU setting, H-bus initialization, peripheral circuit setting, and timer setting.
  sys_initialize: C-language function in the main.c file
  port_initialize: C-language function in the main.c file
  timer_initialize: C-language function in the main.c file

  They are described below.

List 2.1 Basic CPU Setting

```
void sys_initialize(void)
{
   /* special sequence to renew SFRCTL2 */
   /* the default of SFRCTL2 is 0x80    */
   CSCPCMD = 0xA5;                          (1)
   SFRCTL2 = 0x80;
   SFRCTL2 = 0x7F;
   SFRCTL2 = 0x80;

   /* Initialize H Bus */
   ETARCFG0  = 0x0000;                      (2)
   ETARADRS0 = 0xF0000000;
   ETARMASK0 = 0x1FFFFFFF;
   ETARCFG0  = ETARCFG0 | 0x0001;

   return;
}
```

(1) Set the special clock frequency control register (SFRCTL2).
Set the HCLK output enable bit to start Ethernet operation via the H bus.
The SFRCTL2 register is a specific register which is protected from being invalidly written by an uncontrollable program. The CSCPCMD register is a protection register to prevent invalid write operation on the specific register. A particular sequence should be followed for providing write access to the specific register.
For details about the specific register and protection register, refer to the “V850E2/MN4 User’s Manual, Hardware (R01UH0011EJ).”

(2) Initialize the H bus.
This initialization is required for accessing the H bus. For details about the initialization procedure, refer to the “V850E2/MN4 User’s Manual, Hardware (R01UH0011EJ).”

List 2.2 Peripheral Circuit Setting (1/2)

```
void port_initialize(void)
{
   unsigned short OutData;

   /* Ethernet */
   /*   P5[15:0] ETHER[15:0] */
   PFC5 = 0xffff;                                                    (1)
   PMC5 = 0xffff;
   PM5  = 0xfe83;

   /* Ethernet */
   /*   P6[1:0] ETHER[1:0] */
   PFC6  = PFC6  | 0x0003;                                           (1)
   PIPC6 = PIPC6 | 0x0002;
   PMC6  = PMC6  | 0x0003;
   PM6   = PM6   & 0xfffe;

   /* SMEMC */
   /*   P7[15:0] S_D[15:0] */
   PISA7 = 0xffff;                                                   (2)
   SetProtectReg(&PDSC7, 0xffff, &PPCMD7);
   PIPC7 = 0xffff;
   PMC7  = 0xffff;

   /* SMEMC */
   /*   P8[15:0] S_D[31:16] */
   PISA8 = 0xffff;                                                   (2)
   SetProtectReg(&PDSC8, 0xffff, &PPCMD8);
   PIPC8 = 0xffff;
   PMC8  = 0xffff;

   /* SMEMC */
   /*   P9[15:0] S_LUBE, S_LLBE, S_WR, S_RD, x, x, x, x */
   /*               x,      x,    x,    x, x, x, x, x */
   PIPC9 = PIPC9 | 0xf000;                                           (2)
   PMC9  = PMC9  | 0xf000;

   /* SMEMC */
   /*   P10[11:0] x, x, x,     x, x, x, S_WAIT,      x */
   /*             x, x, S_CS1, x, x, x, S_UUBE, S_ULBE */
   PISA10  = PISA10 | 0x0100;                                        (2)
   OutData = PODC10 | 0x0003;
   SetProtectReg(&PODC10, OutData, &PPCMD10);
   PIPC10  = PIPC10 | 0x0023;
   PMC10   = PMC10  | 0x0123;
```

List 2.3 Peripheral Circuit Setting (2/2)

```
    /* SMEMC */
    /*   P11[15:0] S_A[16:1] */
    PIPC11  = 0xffff;                                                    (2)
    PMC11   = 0xffff;

    /* SMEMC */
    /*   P12[9:0] x, x,       x, x, x, x, x, x */
    /*            S_A[24:23], x, S_A[21:17]    */
    PDSC12  = PDSC12 | 0x00df;                                           (2)
    PIPC12  = PIPC12 | 0x00df;
    PMC12   = PMC12  | 0x00df;

    return;
}
```

(1) Connect the input/output port to Ethernet.
Connect bits 0 to 15 in inout port P5 and bits 0 and 1 in inout port P6 to the Ethernet terminals.

(2) Set secondary memory controller SMSC.
Initialize the external memory interface and external SDRAM.

List 2.4 Timer Setting

```
void timer_initialize(void)
{
    /* Initialize TAUA0 ch0*/                                          (1)

    /* setting of CK0 */
    TAUA0TPS   = 0x000f;   /* PCLK/2^15 = 66MHz/32768 = 500us(/1cnt) */
    TAUA0CMOR0 = 0x0000;
    TAUA0CMUR0 = 0x0000;
    TAUA0CDR0  = 2;        /* 1ms */

    TAUA0TOE = TAUA0TOE | 0x0001;
    TAUA0TOM = TAUA0TOM & 0xfffe;
    TAUA0TOC = TAUA0TOC & 0xfffe;
    TAUA0TOL = TAUA0TOL & 0xfffe;
    TAUA0TDE = TAUA0TDE & 0xfffe;

    TAUA0TDM = TAUA0TDM & 0xfffe;
    TAUA0TDL = TAUA0TDL & 0xfffe;
    TAUA0TRE = TAUA0TRE & 0xfffe;
    TAUA0TRO = TAUA0TRO & 0xfffe;
    TAUA0TRC = TAUA0TRC & 0xfffe;
    TAUA0TME = TAUA0TME & 0xfffe;

    TAUA0RDE = TAUA0RDE & 0xfffe;
    TAUA0RDS = TAUA0RDS & 0xfffe;
    TAUA0RDM = TAUA0RDM & 0xfffe;
    TAUA0RDC = TAUA0RDC & 0xfffe;

    TAUA0TS  = TAUA0TE | 0x0001;

    ICTAUA0I0 = 0x0000;

    return;
}
```

(1) Initialize timer TAUA0.
When using the C-NET, interrupts with a 1-ms cycle are required for timer processing. The timer TAUA0 interrupt cycle is therefore set to 1 ms for this timer setting.

3. Confirmation of Sample Software Operation Capabilities

3.1 Software Configuration Including the Sample Applications

The configuration program and functional sample applications below are available.

	Overview
Setup function	
Configuring settings via the browser	Configures various settings via the Web browser when the RTE-V850E2/MN4-EB-S and PC are connected together via Ethernet.
Functional sample applications	
E-mail operation	Manipulates e-mail by using the POP3/SMTP function.
LED operation	Manipulates the LEDs installed on the RTE-V850E2/MN4-EB-S.

The procedure below describes how to use the applications.

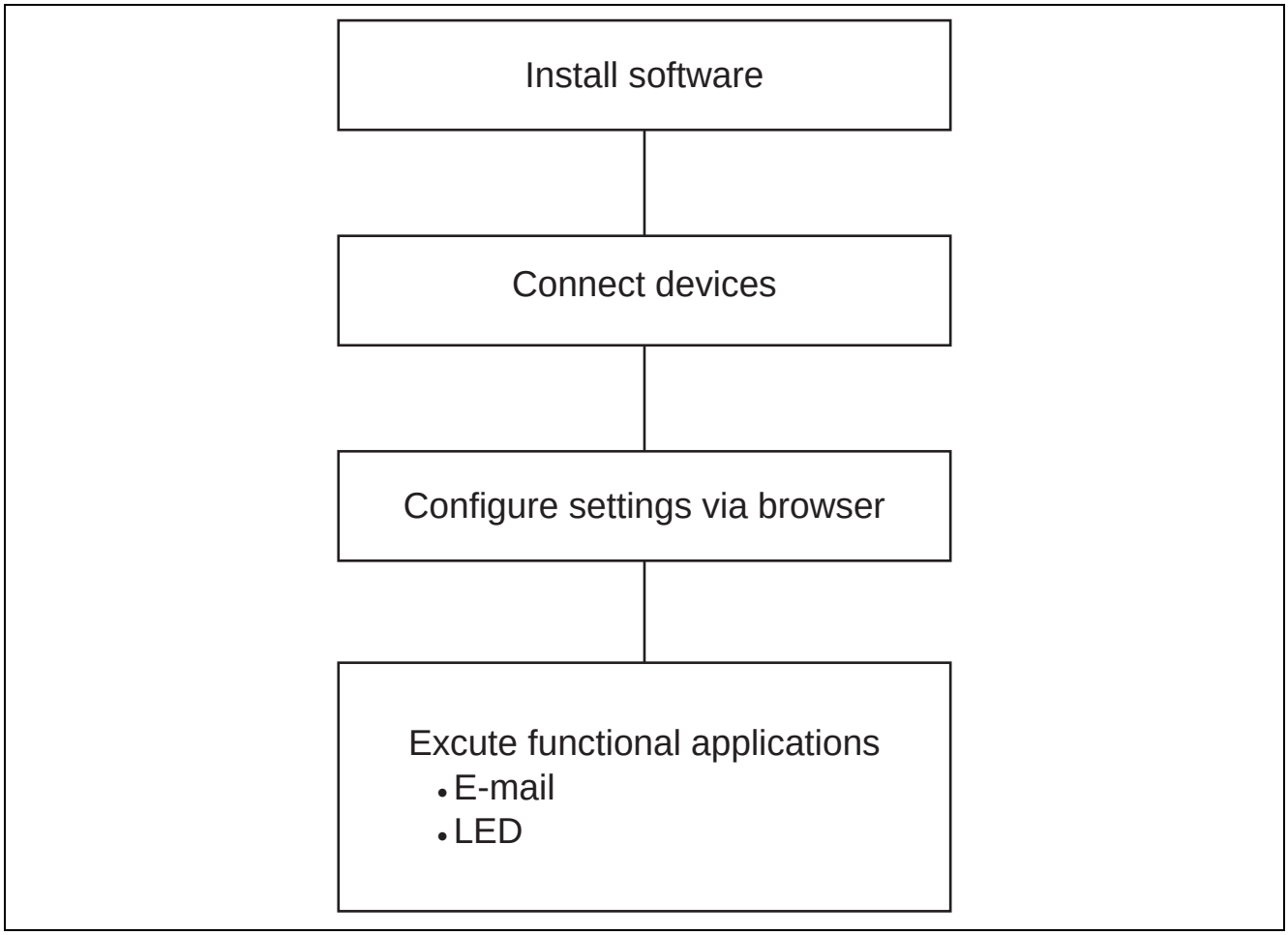


Figure 3.1 How to Use Applications

3.2 Preparation

3.2.1 Device Preparation

The host PC for evaluation should meet the following specifications:

<PC for Evaluation>

CPU	1 GHz or more (hyper-threading, multi-core CPU)
Memory	512 Mbytes or more (1 Gbyte or more recommended)
OS	Windows 2000 Professional Windows XP Professional Windows Vista
Web browser	Internet Explorer 6.0 or later
Ethernet	100BASE-TX/10BASE-T x 1 (used for sample AP execution) Network settings recommended: IP: 192.168.0.10 NetMask: 255.255.255.0
USB	USB (Rev. 1.1, 2.0)

3.2.2 Device Connection

Connect the host PC and RTE-V850E2/MN4-EB-S together after writing the executable module into the flash memory on the RTE-V850E2/MN4-EB-S. For information about building the executable module and storing the evaluation kit in the flash memory, refer to section 3.5, Creating an Executable Module and section 3.7, Storing the Built Object into Flash Memory.

(1) When the DIP switches and jumpers on the RTE-V850E2/MN4-EB-S are used, they should be in the factory-set positions.
Confirm their settings on the board.

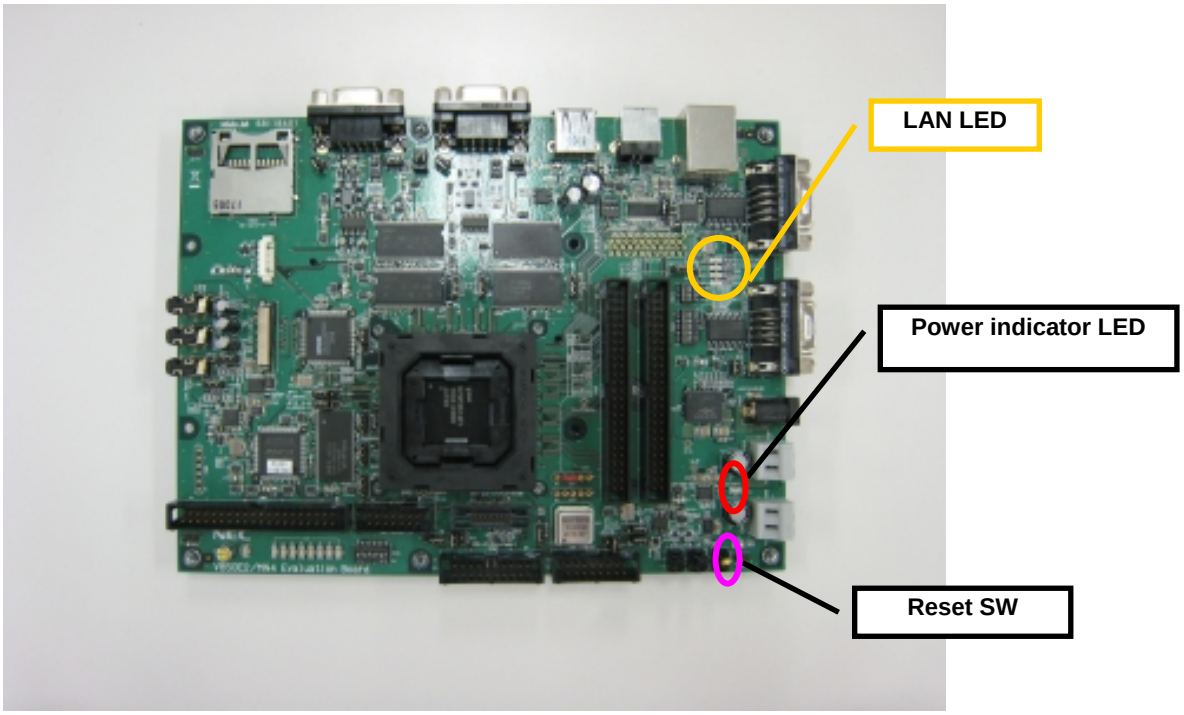
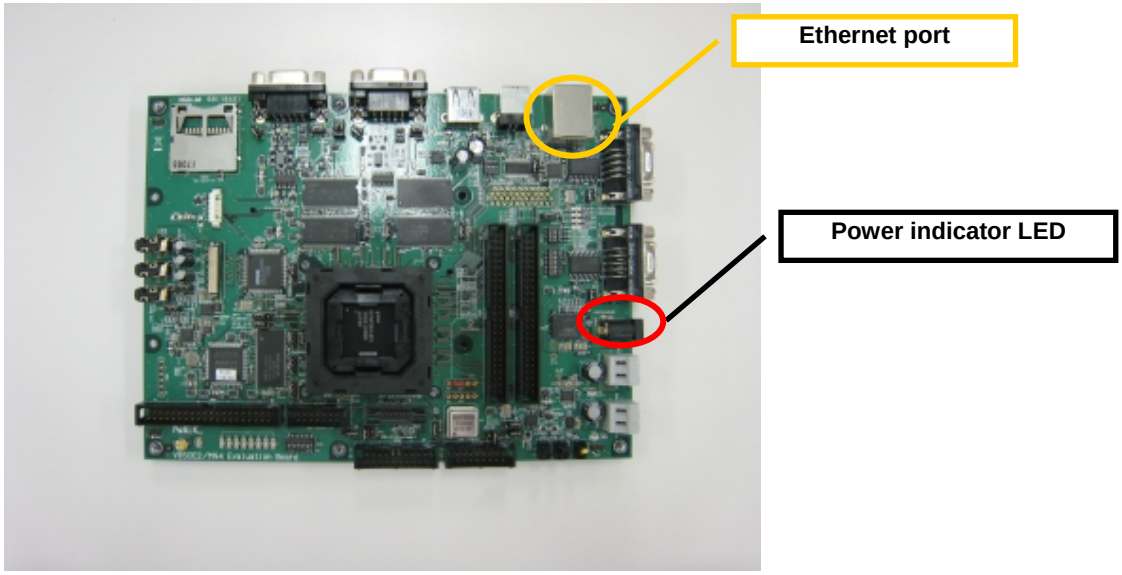


Photo: Switches and LEDs

(2) Connect the host PC and RTE-V850E2/MN4-EB-S Ethernet port together via a LAN cable (category 5) to use the HTTP server and e-mail functions over the network. Use a cross-wired LAN cable if the user wants to directly connect them bypassing the network hub.

(3) Connect the power source to the RTE-V850E2/MN4-EB-S. This device starts operating immediately after it is powered. (This device is not equipped with a power switch.)



3.2.3 Configuring Settings via the Browser Using the HTTP Server

Configure the network parameters via the Web browser on the Ethernet-connected host PC. With this browser, the user can check or change the settings below.

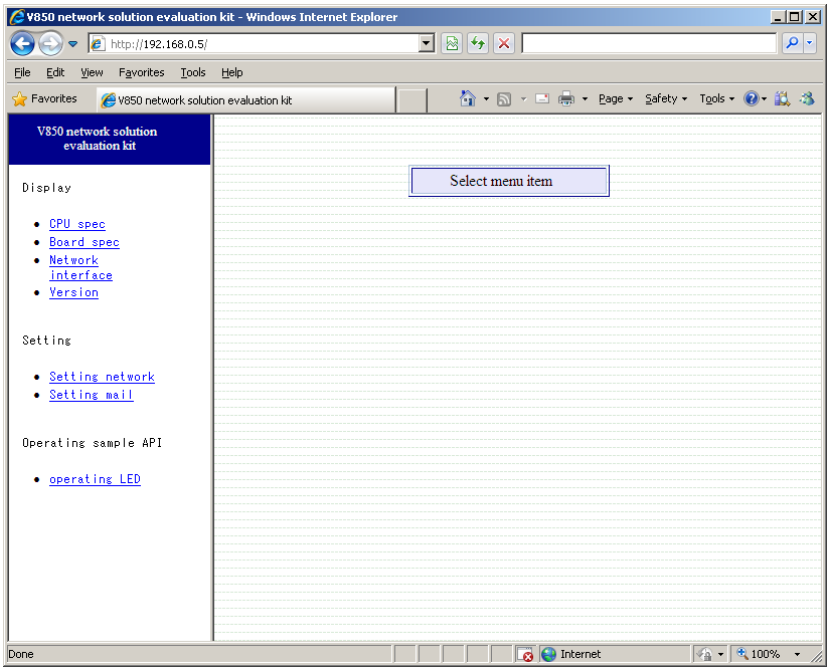
Item	Parameter	Reference/update
IP address	Fixed IP	Reference/update
Subnet mask	Subnet mask value	Reference/update
Subnet mask	Subnet mask value	Reference/update
Gateway	Gateway address value	Reference/update
MAC address	MAC address written into board flash memory	Reference
E-mail address	E-mail address	Reference/update
E-mail SMTP server	E-mail SMTP server address	Reference/update
E-mail POP3 server	E-mail POP3 server address	Reference/update
Version of library	Version number	Reference

(1) Connect the RTE-V850E2/MN4-EB-S to Ethernet using the LAN cable.
Also, connect the PC for using the Web browser to Ethernet.

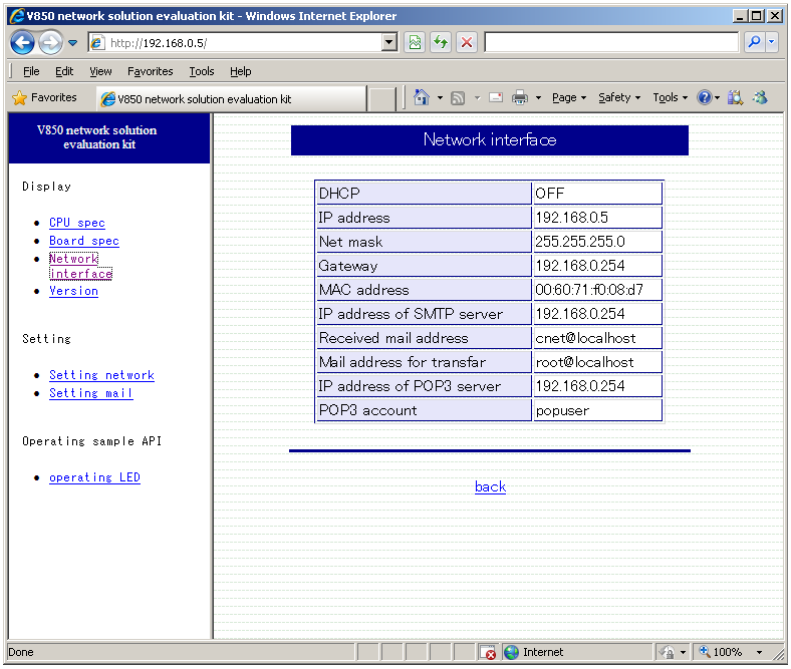
(2) Power up the RTE-V850E2/MN4-EB-S. This boots the program automatically so that the HTTP server starts.

(3) Start the Web browser on the host PC to connect to the HTTP server on the RTE-V850E2/MN4-EB-S. In the browser's address field, enter the address in the format below. The following is the default address.
http://192.168.0.5 / (Default)

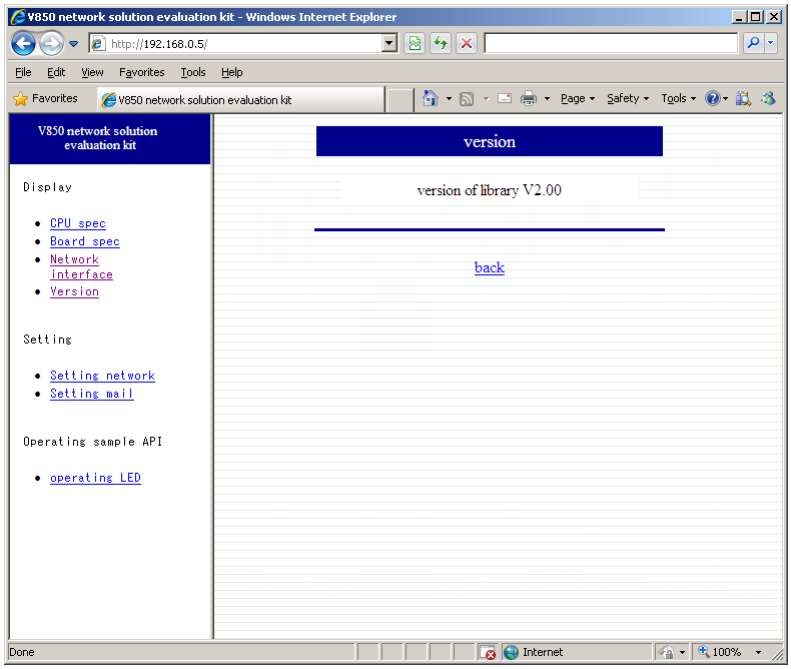
(4) Display the initial Web page on the RTE-V850E2/MN4-EB-S HTTP server.



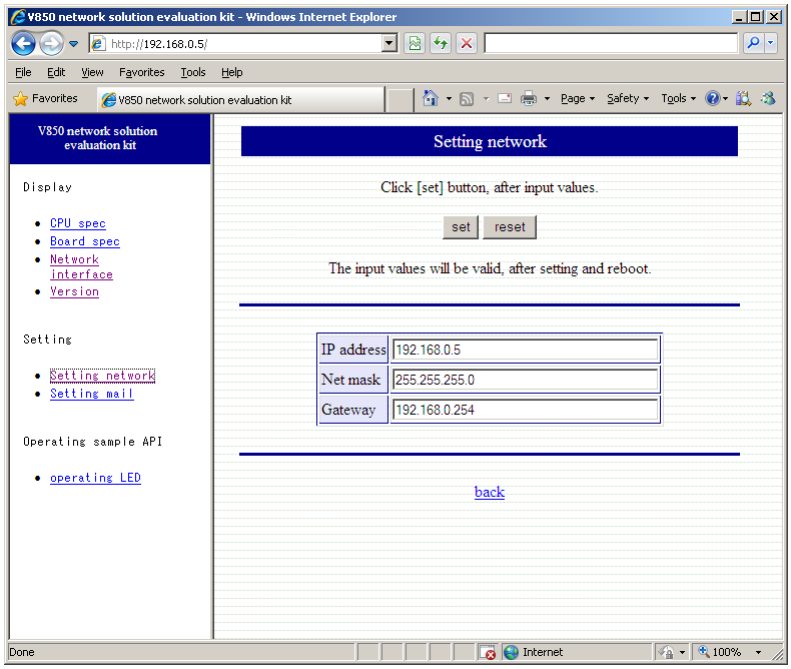
(5) Select [Display] – [Network interface] to display the current settings.



(6) Select [Display] – [Version] to display the version information.

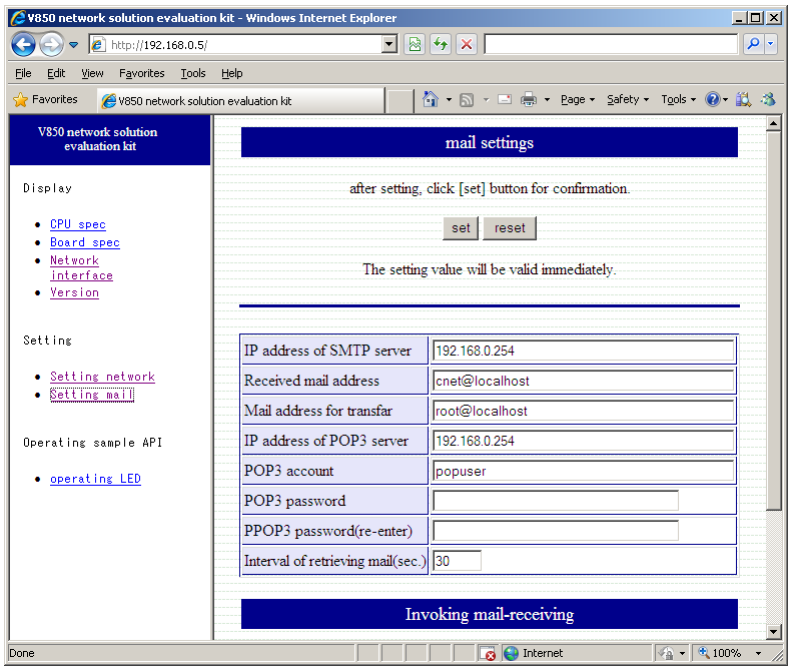


(7) Select [Settings] – [Setting network] to configure the IP address.



The settings configured on this page cannot be written into the flash memory. Changes made to these settings after they are configured do not take effect.

(8) Select [Settings] – [Setting mail] to configure the e-mail function of the functional sample application. Press the [set] button and the settings will be immediately applied.



— Specify the IP addresses of the SMTP server and POP3 server in dot-decimal form.
(Example: 192.168.0.1)
— Specify the e-mail reception address at which e-mail arrives when it is received automatically by the RTE-V850E2/MN4-EB-S.
— Specify the e-mail transfer address to which e-mail is automatically forwarded.
— Specify the POP3 account and password for the POP3 server at which e-mail arrives when it is received automatically by the RTE-V850E2/MN4-EB-S.
— Specify the e-mail reception interval in seconds at which e-mail reception by the POP3 server is checked. The longer the interval, the longer the period from the time e-mail is sent to the specified address until the e-mail sample application starts operation. Specifying a smaller number will shorten the time e-mail is sent until the e-mail sample application starts operation, but will result in a higher e-mail server load. Be careful not to place excessive load on the e-mail server.

3.3 Sample Applications

Functional sample applications are attached to confirm the RTE-V850E2/MN4-EB-S operation. Source codes are also attached to help create the user's own applications.

When these sample applications are executed, the network settings should already be configured. To configure them, refer to section 3.2.3.

3.3.1 E-mail Manipulation Application

[Overview]:

This sample application uses the e-mail functions (POP3 and SMTP).

(1) Using the POP function, this application checks whether e-mail arrives at the server (POP3 server).
(2) If it does, this application receives the e-mail through the POP3 function.
(3) This application forwards the e-mail through the SMTP function if requested by the e-mail contents. (E-mail forwarding to particular addresses)
(4) This application repeats steps (1) through (3) at preset intervals.

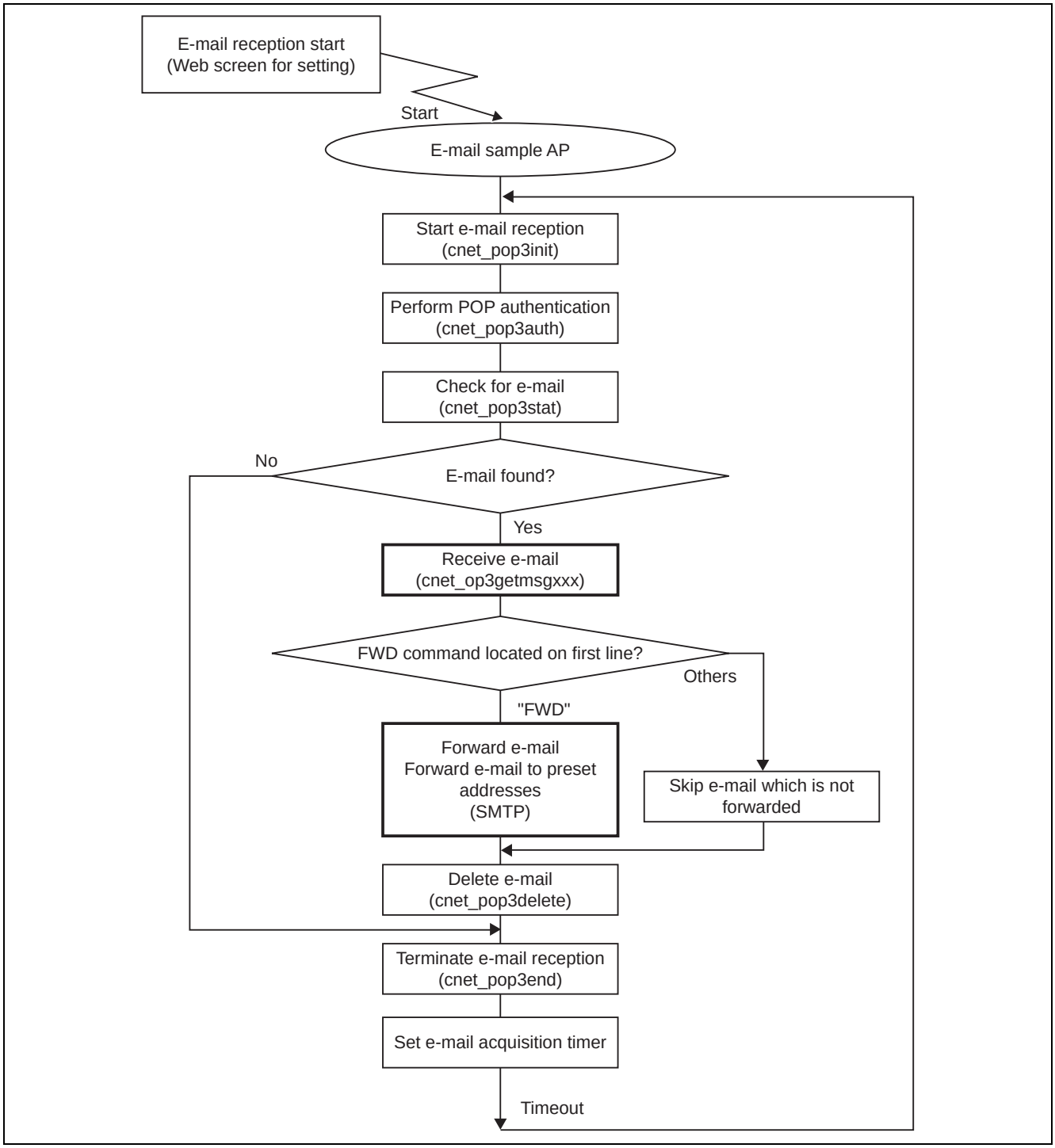
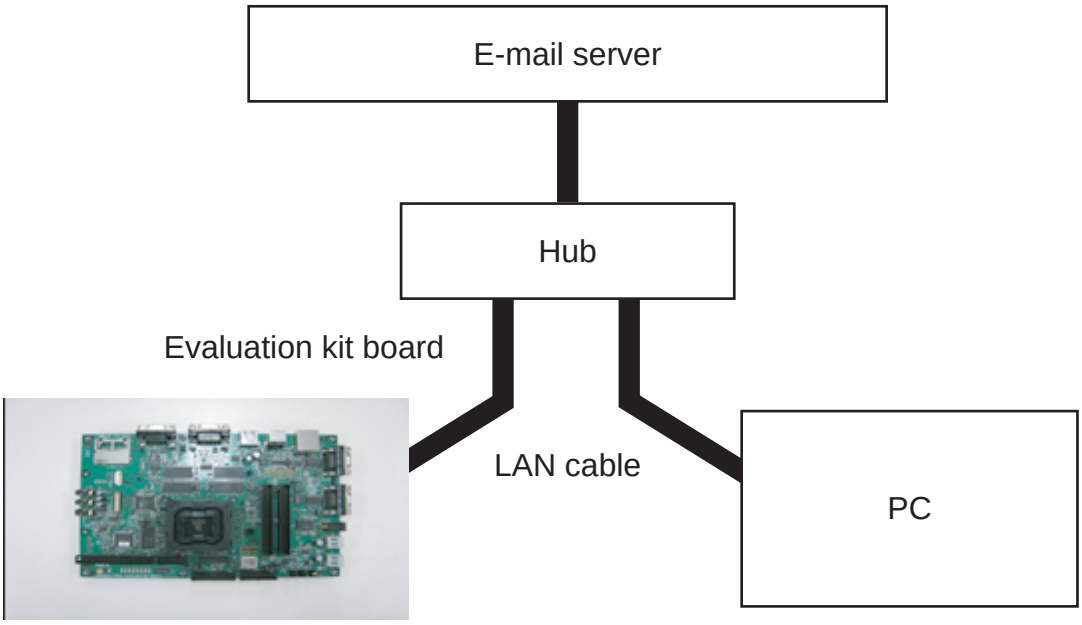


Figure 3.2 E-mail Reception

[Devices Required]:

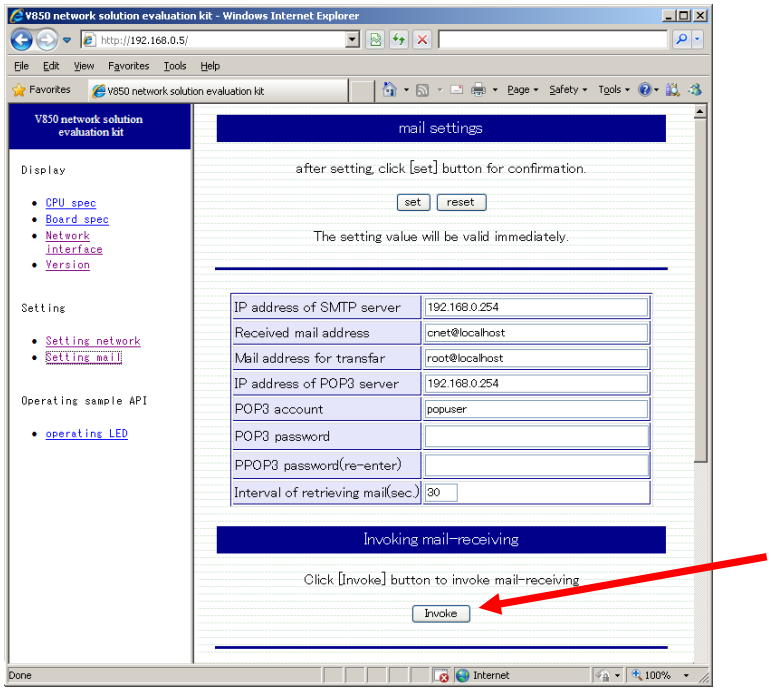
(1) RTE-V850E2/MN4-EB-S
(2) USB cable
Used for supplying power to the PC.
(3) LAN cable
Used for connecting the RTE-V850E2/MN4-EB-S to Ethernet.
(4) PC for using the Web browser

[Connection Example]



[Setup]:

(1) Connect the RTE-V850E2/MN4-EB-S to Ethernet via the LAN cable. Also, connect the PC for using the Web browser to Ethernet.
(2) Configure the e-mail address and SMTP/POP server settings on the Web browser.
(3) Invoking mail-receiving.
In the [Setting mail] screen displayed on the browser, click the [Invoke] button under [Invoking mail-receiving].



Processing to check whether e-mail arrives at the POP3 server gets started.

(4) Check the settings if the connection to the POP3 server fails due to an incorrect IP address, account, or password for the POP3 server.
(5) From a PC or other device, send e-mail to the user specified with the account for the POP3 server.
This e-mail should be in the format below.
To give a command to the e-mail manipulation application, specify the command by left-justifying it on the first line of the text as shown below.

(6)

To: ******@***.***.jp	← Specified e-mail address of the POP3 user
Subject: ******	← Arbitrary information (up to 10 characters)
(First line)	←Command to be given to the e-mail manipulation application
(Second line)	
:	←Information to be forwarded in e-mail
(nth line: Max. 10 lines)	

- Command to be given to the e-mail manipulation application
  FWD
  This command forwards the second and subsequent lines to the specified e-mail destination address.
  (Others)
  If FWD is not specified on the first line of the text in an e-mail, this e-mail will be deleted without being sent after it is received.
  The RTE-V850E2/MN4-EB-S receives e-mail from the POP3 server. This received e-mail may be forwarded depending on the command included in it.

3.3.2 LED Control Application

[Overview]:

This sample application turns on or off the LEDs.

Using the Web browser, provide on/off control of the LEDs at the bottom of the RTE-V850E2/MN4-EB-S.

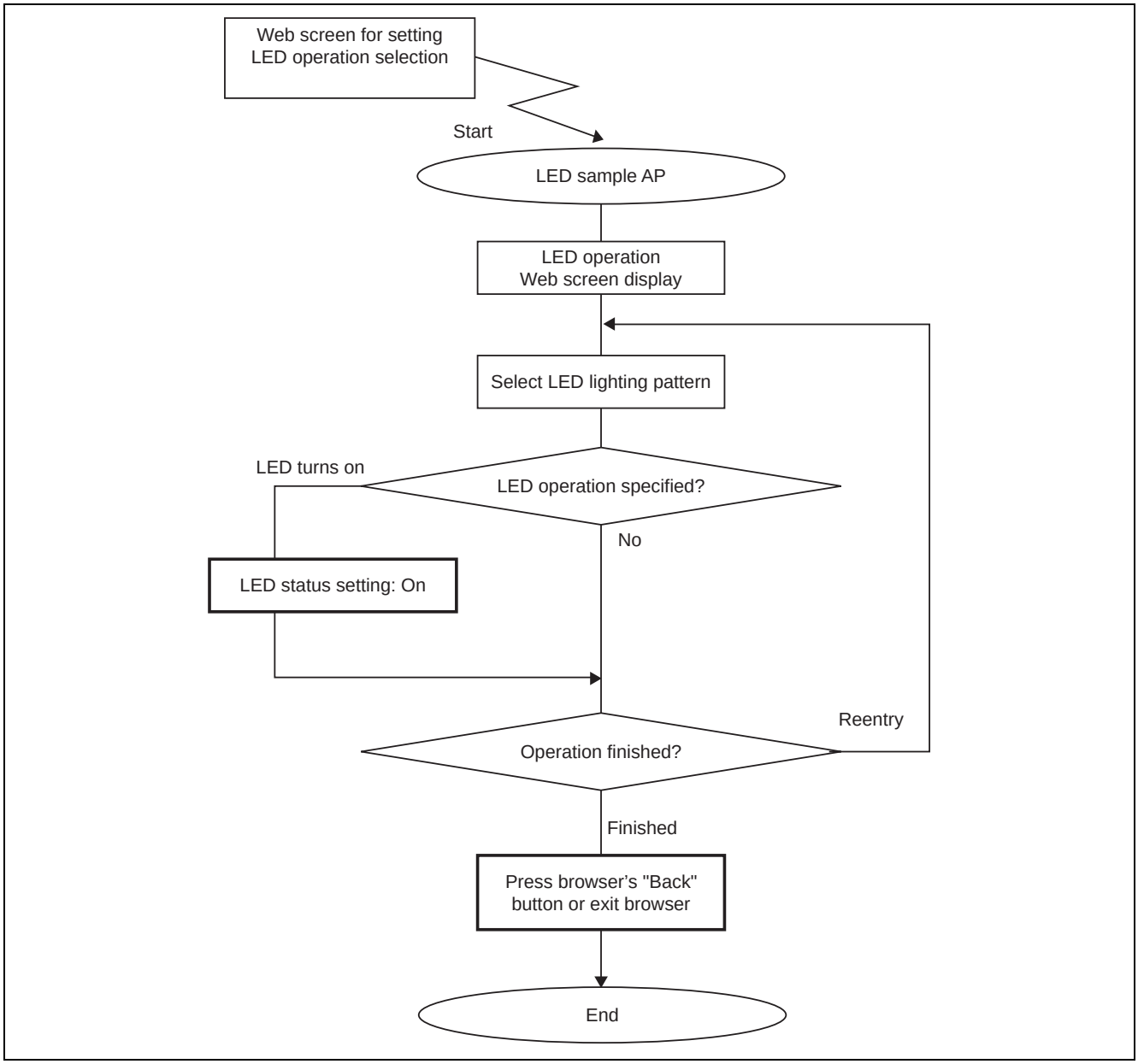
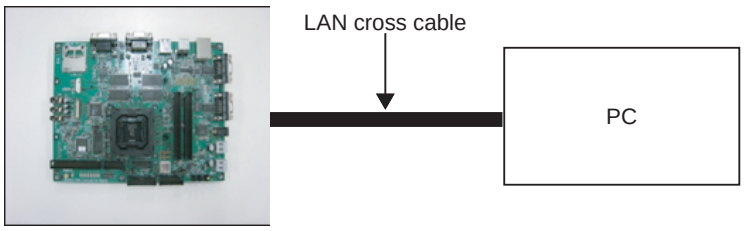


Figure 3.3 Control of LED Operation

[Devices Required]:

(1) RTE-V850E2/MN4-EB-S
(2) LAN cable
Connect the RTE-V850E2/MN4-EB-S to Ethernet.
(3) PC for using the Web browser

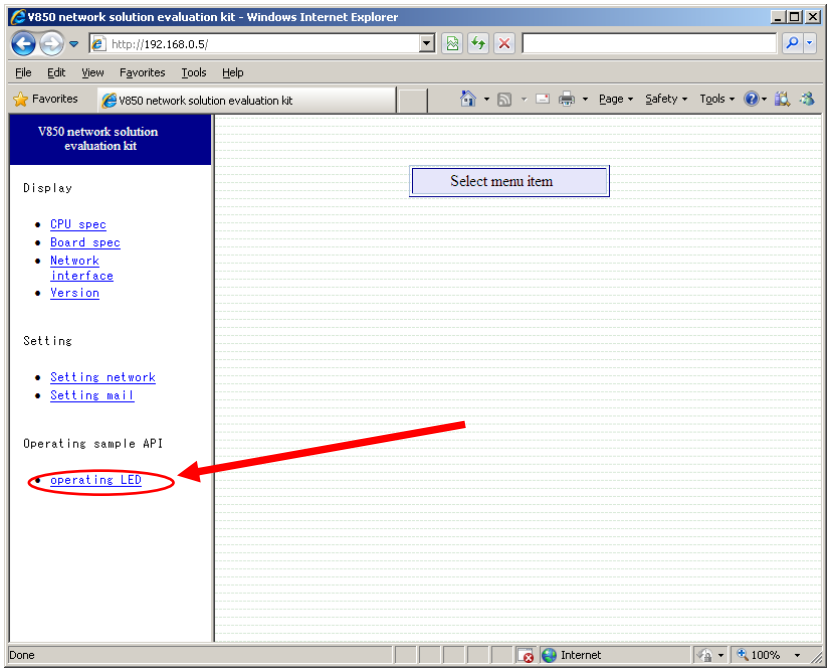
[Connection Example]



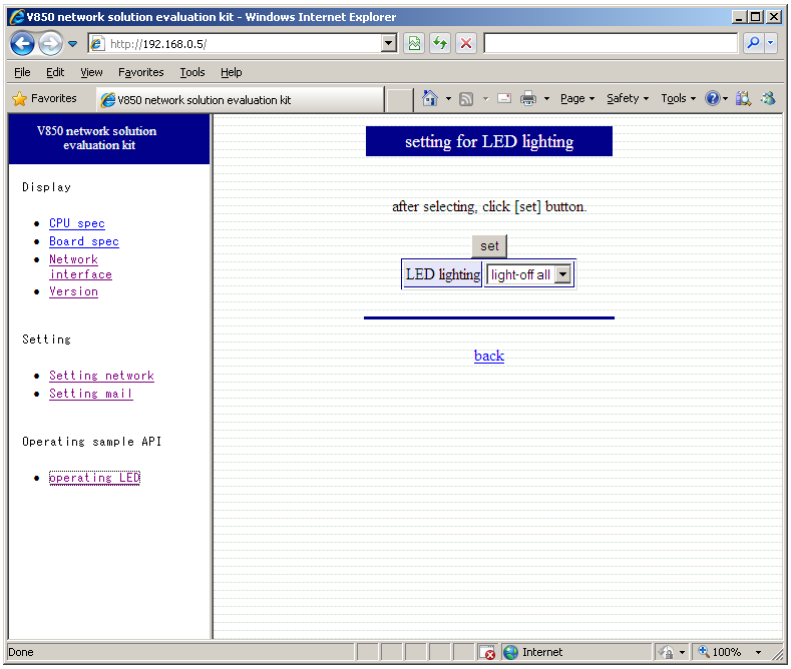
[Setup]:

(1) Connect the RTE-V850E2/MN4-EB-S to Ethernet via the LAN cable. Also, connect the PC for using the Web browser to Ethernet.

(2) Start the Web browser on the PC to connect to the HTTP server on the RTE-V850E2/MN4-EB-S. In the browser's address field, enter the address in the following format:
http://192.168.0.5/ (Default)



(3) Display the LED operation screen.
Select [Operating sample API] – [Operating LED] to display the LED operation screen.



(4) Using the on-screen pulldown menu, select an LED lighting pattern.
(5) Press the [set] button.
(6) The RTE-V850E2/MN4-EB-S's LEDs are set to the selected on/off status.
This status is retained until LED control is provided again. It is reset to off by resetting the RTE-V850E2/MN4-EB-S or turning the power off.
(7) To continue operation, repeat steps (4) through (6).
(8) To finish, exit the browser or press its [back] button to exit the 7Seg.operation screen. The LED status does not change even when the sample application terminates.

3.4 Creating a Web Application

3.4.1 Overview

Below is an example procedure for creating a Web application.

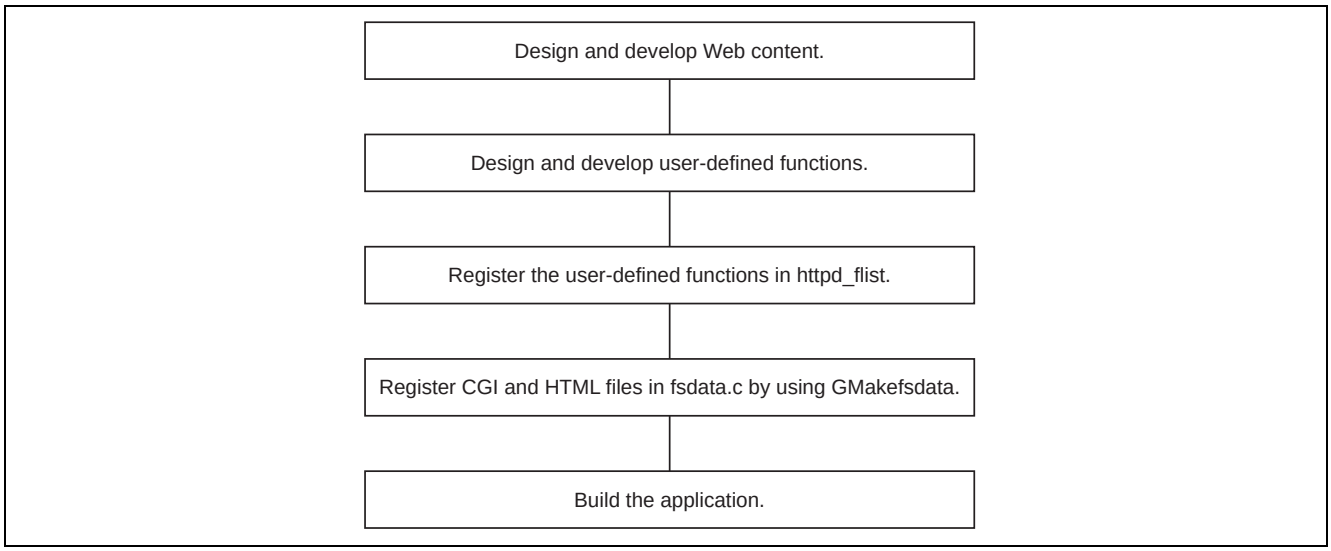


Figure 3.4 Procedure for Creating Web Application

The configuration for creating a Web application is as follows.

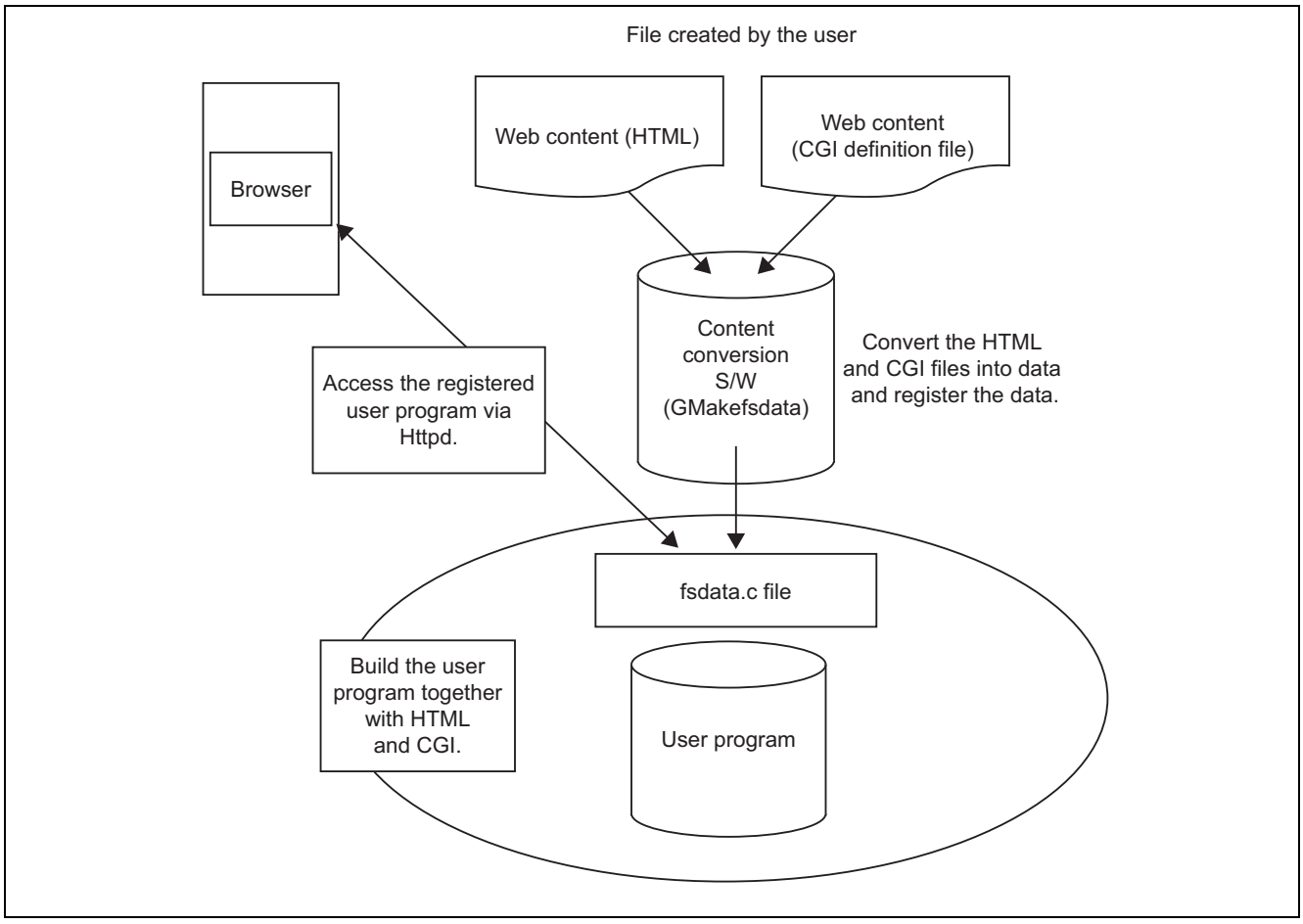


Figure 3.5 Configuration for Creating Web Application

3.4.2 CGI Specifications

CGI Coding Conventions

The following conventions apply to the development of CGI scripts:

- One line corresponds to a unit operation.
- The first character on each line specifies the type of operation.

  i Indicates that the file specified after “i” is output.

  t Indicates that the data specified after “t” is output.

  c Indicates that a function is executed with the function ID and parameters which are specified after “c.”

  * Sends 204 (No Contents).

  # Indicates a comment and does not process data.

  . (period)

  Terminates an HTML statement. (</BODY></HTML>\r\n)

  Others Do not process data.

3.4.3 Definition of Methods

(1) GET method

[Functionality]

This method requests the displaying of HTML, CGI, and image data.

[Rules]

If there is only “/” specified, this method displays index.html.

If the specified HTML or CGI file is missing, this method displays 404.html.

If a file under /cgi/ is specified, the file is considered to be a CGI file.

[Sample code]

GET /index.html HTTP/1.0

GET /test.html?func1=param1 HTTP/1.0

(2) POST method

[Functionality]

This method passes parameters to CGI.

[Rules]

If a file under /cgi/ is not specified, this method returns error 405.

If the specified CGI file is missing, this method displays 404.html.

3.4.4 Creating a User-Defined Function

To create a function which processes data via CGI, specify the association between the function ID and function in the httpd_flist structure. This function is called by specifying the function ID by means of CGI, data passed by the POST method, etc.

(1) User-defined function

[Functionality]

A user-defined function is called by specifying the function ID. To specify this ID, use the GET method parameter, data passed by the POST method, or CGI “c” (function execution) command.

This user-defined function can be included in httpdfunc.c or any other file.

(2) Format for a user-defined function

Each user-defined function includes three arguments. The number and types of arguments are fixed.

First argument	int type	Passes an ID available for the HTTP server to manage information.
Second argument	const char* type	Passes a function ID.
Third argument	const char* type	Passes the specified parameter for CGI or received data for the GET or POST method.

Format for cnet_httpd_datacmn()

cnet_httpd_datacmn() is a function which is always called after the data received with the GET or POST method has been processed. This function includes one argument. However, this function is not called if no parameter is specified in the GET method.

First argument	int type	Passes an ID available for the HTTP server to manage information.

For an example of an implemented user-defined function, refer to httpdfunc.c which is attached as a sample application.

(3) Registering user-defined functions

```
Register user-defined functions in httpdfunc.c.
/* Declaration of structure cnt_finfo included in cnet_httpd.h */
typedef struct {
    const char *fn;                                          /* Function ID*/
    void      (*fi)(int id, const char *name, const char *value);    /* Function pointer*/
} cnet_finfo;
=== httpdfunc.c ============================================================
/*Definition of user-defined functions included in httpdfunc.c and sample httpd_flist*/
static void
a(int id, const char *name, const char *value)
{
    ~
}

static void
b(int id, const char *name, const char *value)
{
    ~
}

static void
c(int id, const char *name, const char *value)
{
    ~
}

static void
func1(int id, const char *name, const char *value)
{
    ~
}

const cnet_finfo httpd_flist[] = {
{"a", a},
{"b", b},
{"c", c},
{"func1", func1},
{NULL, NULL}
};
============================================================================
```

Specify the user-defined function IDs and function pointers in the httpd_flist structure. The end of registration should be indicated by specifying the combination of “NULL, NULL” as the function ID and function pointer. Thus, be sure to add “NULL, NULL” to the end of the code.

[Sample code]

- Sample code for the POST method
  POST /cgi/test.cgi HTTP/1.0
  Content-Length: 12
  func1=param1
- Sample code for CGI
  c func1 param1
- Function call
  func1(id, “func1”, “param1”);

For information about using the CGI methods and calling a user-defined function, refer to the attached sample program of the Web application.

3.4.5 Using the Web File Registration Tool (GMakefsdata)

Before the HTTP server handles a file, be sure to register the file in fsdata.c by using Gmakefsdata.exe below.

- Tool operation environment requirement
  OS: Windows 2000 Professional or Windows XP Professional

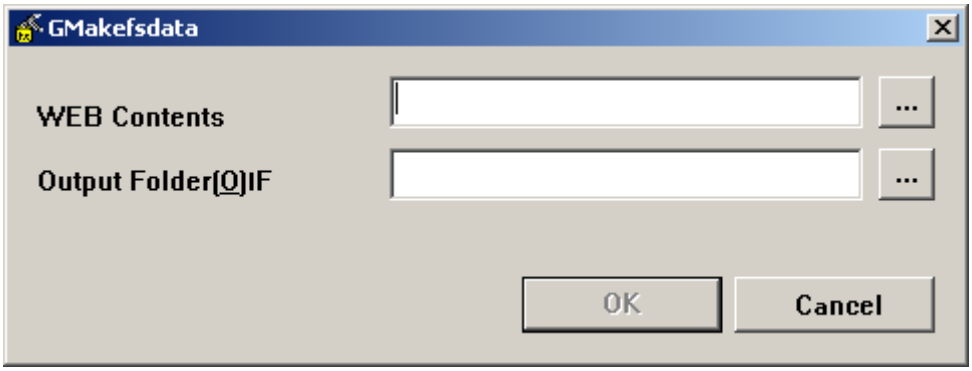
The procedure below describes how to register the file.

(1) Create folder fs at an arbitrary location.
→ The operating environment for the attached sample already includes \fs and \cgi.

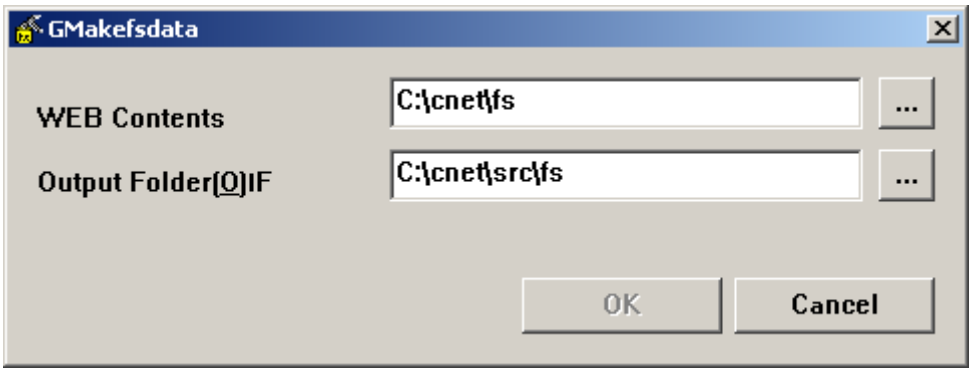
```
\cnet
        \bin
                GMakefsdata.exe          .....File registration tool
        \fs
                (HTML file)
                \cgi
                        (CGI file)
                \gif
                        (GIF file)
        \src
                \fs
                        fsdata.c         .....Content data file generated
```

(2) Create HTML, CGI, image, and other files under fs to make up the Web page.
All the files can be included in folders at any hierarchical level below fs except the CGI files which should be under fs\cgi.

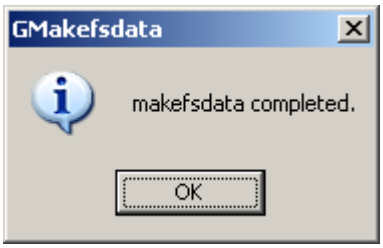
(3) After creating all the files under folder fs, execute GMakefsdata.exe. This displays the screen below.



(4) Specify the location of created folder fs and the output destination for fsdata.c, and press the OK button. Pressing the OK button starts content file registration.



(5) After the content file registration, fsdata.c is created at the specified output destination and the screen below appears. Press the OK button to terminate GMakefsdata.

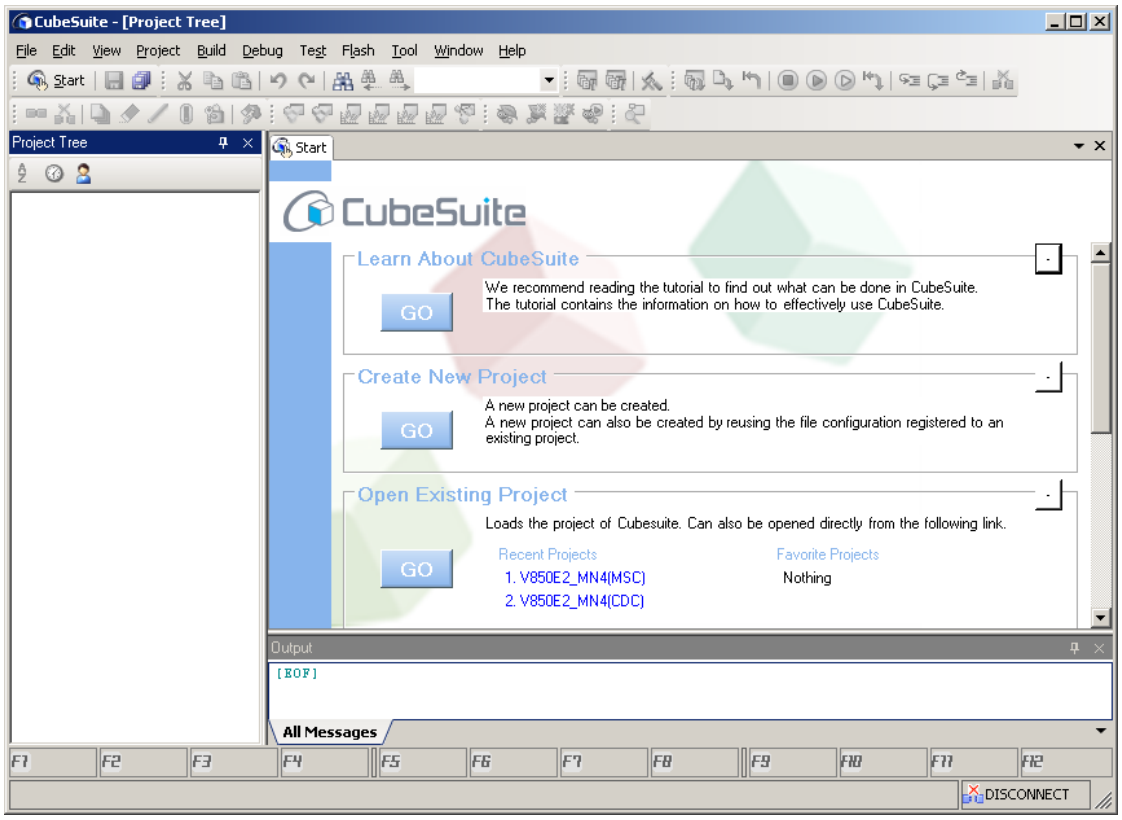


(6) After you finish, rebuild the files.

3.5 Creating an Executable Module

3.5.1 Building a Program

(1) Start CubeSuite.



(2) The explanation here is based on a sample project. The sample projects are located in the folder below.
C: \CNET_V2\Program Files\Renesas Electronics Tools\smp850e\cnet\
V850E2MN4_EvBoard\OSless\CNET ……Folder for holding created user programs

(3) Build objects related to the user program.
Select “File” → “Open.” Select the cnet.cspj file in the CNET folder as the project file for CubeSuite. This opens the cnet project.
Before building the objects, set the MAC address when the cnet project is open.
Select “File” → “main.c” from the project tree. Change the MAC address in L.45 to the MAC address of the RTE-V850E2/MN4-EB-S.
As an example, suppose the MAC address is “01:23:45:67:89:AB.”

```
const struct macdata     macaddr =
{
{0x01,0x23,0x45,0x67,0x89,0xab},// my MAC
{0x00,0x00}
};
```

The MAC address is a unique number assigned to the RTE-V850E2/MN4-EB-S. Use this MAC address, not an arbitrary MAC address.

(4) The objects related to the user programs are created after the build is performed.

3.5.2 Executable Module Generated

A load module file is created after the successful build.

(In this example, this file is c:\CNET_V2\Program Files\Renesas Electronics Tools\smp850e\cnet\V850E2MN4_EvBoard\OSless\CNET\DefaultBuild\cnet.lmf)

3.6 Executing the Built Objects

Execute the generated objects using CubeSuite.

After changing the MAC address in main.c, select “Build” → “Build Project” to perform the build. Then, select “Debug” → “Download to Debug Tool” to start the debugger. Open the cnet.lmf file which is generated after the build from the debugger. This downloads the created load module and enables debugging. Set breakpoints as required and confirm operations.

For details, refer to the “RTE-V850E2/MN4-EB-S Hardware Specifications.”

3.7 Storing the Built Objects into Flash Memory

The created ROM image should be written into the RTE-V850E2/MN4-EB-S’s flash memory.

When the cnet.lmf file is downloaded with the debugger as described in section 3.6, the ROM image is stored into the RTE-V850E2/MN4-EB-S’s flash memory’s flash memory. Thus, the stored program can be executed simply by starting the RTE-V850E2/MN4-EB-S alone. For details, refer to the “RTE-V850E2/MN4-EB-S Hardware Specifications.”

Website and Support

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

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

Revision Record

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Jan 31, 2012	—	First edition issued

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.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
   - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
   - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
   - "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

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: +49-211-65030, 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.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
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 1.1