Summary
This application note describes an example of settings for connecting the Ethernet controller of the SH7670, SH7671, SH7672, and SH7673.

Target Device
SH7670 MCU

Contents

1. Introduction ............................................................................................................................ 2
2. Description of the Sample Application .............................................................................. 3
3. Sample Program Listing ..................................................................................................... 3
4. References .......................................................................................................................... 40
1. Introduction

1.1 Specifications
- In this sample program, ten Ethernet frames are received. After the transmission of each frame is completed, transmission of the next proceeds.
- The frame transmission complete interrupt is used to judge whether frame transmission has been completed or not.

1.2 Module Used
- Ethernet controller (EtherC)
- Ethernet controller direct memory access controller (E-DMAC)
- Interrupt controller (INTC)
- I²C bus interface 3 (IIC3)
- Pin function controller (PFC)

1.3 Applicable Conditions

<table>
<thead>
<tr>
<th>MCU</th>
<th>SH7670</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating Frequency</td>
<td>Internal clock: 200 MHz</td>
</tr>
<tr>
<td></td>
<td>Bus clock: 66.6 MHz</td>
</tr>
<tr>
<td></td>
<td>Peripheral clock: 33.3 MHz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Environment</th>
<th>Renesas Electronics</th>
</tr>
</thead>
<tbody>
<tr>
<td>C Compiler</td>
<td>Renesas Electronics SuperH RISC engine Family</td>
</tr>
<tr>
<td></td>
<td>C/C++ compiler package Ver.9.01 Release 01</td>
</tr>
<tr>
<td>Compiler Options</td>
<td>Default setting in the High-performance Embedded Workshop</td>
</tr>
<tr>
<td></td>
<td>(-cpu=sh2afpu -fpu=single -debug -gbr=auto -global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1)</td>
</tr>
</tbody>
</table>

1.4 Related Application Notes
For more information, refer to the following application notes:
- SH7670 Group Example of Initialization
- SH7670 Group Example of Setting for Automatic Negotiation by Ethernet PHY-LSI
- SH7670 Group Example of Setting for Reception of Ethernet Frames
2. Description of the Sample Application

This sample application employs an Ethernet controller (EtherC) and a direct memory access controller for Ethernet controller (E-DMAC).

2.1 Operational Overview of Module Used

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

2.1.1 Overview of the EtherC

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

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

Figure 1 shows a configuration of the EtherC.
2.1.2 Overview of the EtherC Transmitter

The EtherC transmitter assembles the transmit data on the frame and outputs to MII when there is a transmit request from the E-DMAC. The data transmitted via the MII is transmitted to the lines by PHY-LSI. Figure 2 shows the state transition of the EtherC transmitter.

The following describes the flow of operations in transmission.

1. When the transmit enable (TE) bit of the EtherC mode register (ECMR) is set, the EtherC transmitter enters the idle state.
2. (A) When a request for transmission is issued by the transmitter E-DMAC while half-duplex transfer has been selected, the EtherC module attempts to detect a carrier. If it does not detect a carrier, the EtherC module sends the preamble to the RMII after a transmission delay equivalent to the time required by the frame interval. If a carrier is detected, the EtherC module waits until the carrier disappears and then sends the preamble to the RMII after a transmission delay equivalent to the time required by the frame interval.
   (B) Full-duplex transfer does not require carrier detection, so if this is selected, the preamble is sent as soon as the request for transmission is issued by the E-DMAC. In continuous transmission, however, the preamble is sent from the frame which has been transmitted at the last minute surely after a transmission delay equivalent to the time required by frame interval.
3. The EtherC transmitter sends the start frame delimiter (SFD), data, and cyclic redundancy check (CRC) code in sequence. At the end of transmission, the transmitter E-DMAC generates a frame transmission complete (TC) interrupt. If a collision occurs or the EtherC transmitter enters the carrier-not-detected state, an interrupt corresponding to the given state will be generated.
4. The EtherC transmitter enters the idle state and then, if there are more data for transmission, continues to transmit.
Notes: 1. Transmission retry processing includes both jam transmission that depends on collision detection and the adjustment of transmission intervals based on the back-off algorithm.
2. Transmission is retried only when data of 512 bits or less (including the preamble and SFD) is transmitted. When a collision is detected during the transmission of data greater than 512 bits, only jam is transmitted and transmission based on the back-off algorithm is not retried.

Figure 2  EtherC Transmitter State Transitions
2.1.3 Overview of the E-DMAC

This LSI includes a direct memory access controller (E-DMAC) directly connected to the Ethernet controller (EtherC). The E-DMAC transfers data for transmission and reception between transmit/receive FIFO in the E-DMAC and data storage location (transmit/receive buffer) specified by the user using DMA transfer. Directly writing data to or reading data from the transmit/receive FIFO by the CPU is not possible. During DMA transfer, the E-DMAC refers to information called transmit and receive descriptors (details to be described in the next section); these are placed in memory by the user. The E-DMAC reads the descriptor information before transmitting or receiving an Ethernet frame, and follows the descriptor in reading data for transmission from the transmission buffer or writing received data to the receiving buffer. By setting up a number of consecutive descriptors (a descriptor list), it is possible to execute the consecutive transfer of multiple Ethernet frames. This E-DMAC function lightens the load on the CPU and enables efficiency in data transfer control.

Figure 3 shows the configuration of the E-DMAC, and of the related descriptors and buffers.

The E-DMAC has the following features:
- Equipped with two independent on-chip DMACs for transmission and reception
- The load on the CPU is reduced by means of a descriptor management system
- Transmit/receive frame status information is indicated in descriptors
- Block transfer by using DMA (16-byte units) achieves efficient utilization of the system bus
- Supports one-frame/one-descriptor, one-frame/multi-frame (multi-buffer) operation (see section 2.1.5)
2.1.4 Overview of E-DMAC Descriptors

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

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

Figure 4 shows the relationship between a transmit descriptor and a transmit buffer.

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

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

![Figure 4](image-url)
2.1.6 Example of Setting Transmit Descriptors

Figure 5 shows an example (one frame/one descriptor) where three transmit descriptors and three areas of the transmit buffer are in use. In this case, a single frame is transmitted in response to a single request for transmission. The transmit descriptors are simplified in the figure, with only TD0 being shown. Numbers (1), (2), etc. in the figure indicate the sequence of execution.

The Settings are as follows.

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

![Figure 5: Relationship between Transmit Descriptor and Transmit Buffer](image-url)
2.1.7 Operation of the Sample Program

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

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

- **TFP = B'00 or B'10 (frame continuation):**
  - Writing back to the descriptor (to write 0 to the TACT bit) proceeds after the DMA transfer. The TACT bit of the next descriptor is then read.

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

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

Figure 6 shows an example of the flow of transmission (in the one-frame/one-descriptor and multiple-descriptor cases).
Supplementary Notes
EtherC/E-DMAC initialization: Both modules are software reset by setting the SWR bit in the EDMR to 1.
EtherC/E-DMAC settings: 1 is written to the TE bit in the ECMR and the TR bit in the EDTRR.
Initiation of transmission: This is done by the above action (writing 1 to the TE bit in the ECMR and TR bit in the EDTRR).
Reading of transmit descriptor: The E-DMAC automatically reads the transmit descriptor.
Transfer of data for transmission: The E-DMAC performs DMA transfer to write the data for transmission to the transmit FIFO.
Transmit descriptor write-back: The E-DMAC writes 0 to the TACT bit and writes the transmission state information to the transmit descriptor.

Figure 6 Sample Flow of Transmission
2.1.8 Procedure for Setting Module Used

This section describes an example of fundamental settings for reception of the Ethernet frames. Figures 7 and 8 show an example of flowchart for setting the reception of Ethernet frames.

- The EtherC and E-DMAC modules are reset by software. by writing the SWR bit in E-DMAC mode register. Access to all Ethernet-related registers is prohibited while the software reset is being executed (which takes 64 cycles of internal bus clock).
- Areas of transmit descriptors on memory is cleared.
- Areas of transmit buffers on memory is cleared.
- The entire transmit descriptor list is initialized.
  TD0: Bit TACT is set to 1 (valid).
  Bit TDLE is set to 1 in the last descriptor (and 0 in the others).
  Setting is not required for bit TFP because this item is written back by the E-DMAC.
  TD1: Bit RBL sets the maximum transmission byte length in the corresponding transmit buffer.
  Setting is not required for bit TDL because "this item is written back by the E-DMAC.
  TD2: Specifies the start address of the transmit buffer corresponding to each descriptor.
  The transmit buffer start address must be aligned with a longward boundary.
  When SDRAM is connected, it must be aligned with a 16-byte boundary.
  Padding area: This area is not used by the E-DMAC, but is freely available to the user.
- Initializes pointer variable that manages the current descriptor.
  The start address of the transmit descriptor list is set as the initial value.
- Selects whether or not the endian format is converted on data transfer by the E-DMAC.
  Descriptor length is set.
- Sets the start address of the transmit descriptor list.
  Lower-order bits are set as follows according to the specified descriptor length.
  16-byte boundary: TDLA[3:0] = 0000
  32-byte boundary: TDLA[4:0] = 00000
  64-byte boundary: TDLA[5:0] = 000000
  Actual memory areas are also allocated on corresponding boundaries.
- Specifies the threshold for data in the transmit FIFO at which the first transmission is started.
  In store-and-forward mode, this is set to H'00000000. Lowering the threshold value improves throughput, but this requires care to avoid an underflow of data for transmission.
- Sets the capacity of the transmit FIFO and receive FIFO.
  The setting is H'00000707, which selects the maximum capacity of 512 bytes.
- The setting of the TIS bit in this register determines notification or non-notification of the completion of write-back for each transmitted frame. The TIS bit is set to 1, selecting notification of the completion of write-back for each transmitted frame.
- Sets the gap between packets.
  The setting is H'14 which selects a gap of 96 bit periods.

Figure 7 Example of a Flowchart for Ethernet Settings (1)
**SH7670 Group Example of Setting for Transmission of Ethernet Frames**

- When automatic negotiation is enabled, wait until it is completed.

- Either full duplex or half duplex transfer method is obtained from the result of automatic negotiation by PHY-LSI.

- Set whether CRC error frame is received as an error or not.

- Magic Packet detection is enabled if required.

- Internal/external loopback is specified if required.

- Specify full duplex or half duplex mode
  - The result of automatic negotiation by the PHY-LSI is reflected.
  - The register is cleared to 0 by writing 1 to all of its bits

- The register is cleared to 0 by writing 1 to all of its bits

- Settings are made to the following bits: the link signal change interrupt enable, magic packet detection interrupt enable, and illegal carrier detection interrupt enable.
  - Set ECIIP in EESIPR to 1 when this interrupt is used.
  - Interrupts to all the bits in the EtherC/E-DMAC status register (EESR) are enabled.

- Priority level of the E-DMAC-related interrupt is set.
  - Bits 11 to 8 in the IPRC are set. The setting H'0 indicates the priority level 0 (requested mask), and H'F indicates the priority level 15 (the maximum level).

- Transmission of data is enabled by setting the TE bit in the ECMR to 1.

- Check that operations for the current descriptor are not in progress.
  - Use the descriptor management pointer for transmission to check that the TACT bit is cleared to 0 and the transmission has been completed or aborted.

- Data for transmission are set in buffer associated with the current descriptor.

- The setting to enable the transmission of data associated with the current descriptor is made.
  - The position within the frame for transmission is set in the TFP (the setting is three for one frame/one descriptor).
  - The data length of a transmit frame is set in the TDL.
  - The TACT bit is set to 1 (this bit is the last to be set)
  - The pointer is updated from the current descriptor to the next descriptor.
  - The TR bit in EDTRR is set to 1 initiating transmission

**Figure 8 Example of a Flowchart for Ethernet Settings (2)**
2.2 Operation of the Sample Program

This sample program employs the EtherC and the E-DMAC modules to transmit 10 Ethernet frames from the host personal computer at the other end. In this sample program, there are four transmit descriptors, and four areas of the transmit buffer each with 1,520 bytes.

Transmit descriptors are used in a ring structure. The frame transmit complete interrupt (TC) is used to determine when the transmission of one frame is completed and to start transmission of the next.

Data for transmission in the Ethernet frame (i.e., the frame with the exception of the preamble, start frame delimiter (SFD), and CRC section) needs to be prepared. The destination and source MAC addresses in the header must be changed to the MAC addresses of the devices in use. Note that the EtherC module does not check the source MAC address.

Figure 9 shows operating environment of the sample program, and figure 10 shows a format of the Ethernet frame.
2.3 Definition of Descriptors Used in the Sample Program

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

Figure 11 shows the definition of the transmit-descriptor structure in the sample program and an example of how the array of transmit descriptors is used.

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

**Figure 11** Definition of Transmit Descriptor and Usage Example of Transmit Descriptor Array
2.4 Sequence of Processing by the Sample Program

Figures 12 to 15 show the flow of processing in the sample program. Although descriptors and the various registers of the EtherC and E-DMAC modules are initially set up for reception, processing for reception is not performed.

For details on the automatic negotiation function phy_autonego, see the application note “SH7670 Example of Setting for Automatic Negotiation by Ethernet PHY-LSI (REJ06B0800)”. 

---

![Flowchart of the Sample Program](image)

Figure 12 Flow of Handling in the Sample Program (1)
**LAN open function**

**LAN close function**

**Start**

1. Release the EtherC/E-DMAC from module standby
2. Reset the EtherC/E-DMAC registers
3. Create descriptors
4. Set the MAC address
5. Obtain result of automatic negotiation
6. Success?
   - Yes: Set the EtherC/E-DMAC registers
   - No: Go back to step 6
7. Set interrupt priority of the E-DMAC to 0
8. Reset the EtherC/E-DMAC registers
9. Set the EtherC/E-DMAC module standby
10. Release the EtherC/E-DMAC from module standby

**Figure 13  Flow of Handling in the Sample Program (2)**
## EtherC/E-DMAC Reset Function

### lan_reg_reset

**START**
- Set the SWR bit in the E-DMAC mode register (EDMR)
- Wait for over 64 bus clock cycles

**END**

## Function for Initialization of Transmit/Receive Descriptor

### lan_desc_create

**START**
- Clear the descriptor area to 0
- Make initial settings for the transmit descriptor
- Make initial settings for the receive descriptor
- Initialize pointers for descriptor management
- Clear the transmit and receive buffers to 0

**END**

## Function for EtherC/E-DMAC Register Setting

### lan_reg_set

**START**
- Set the E-DMAC mode register (EDMR)
- Set the transmit descriptor list address register (TDLAR)
- Set the receive descriptor list address register (RDLAR)
- Set the transmit/receive status copy enable register (TRSCER)
- Set the transmit FIFO threshold register (TFTR)
- Set the FIFO depth register (FDR)
- Set the receiving method control register (RMCR)
- Set the E-DMAC operation control register (EDOCR)
- Set the transmit interrupt register (TRIMD)
- Set the EtherC mode register (ECMR)

**END**

---

**Figure 14** Flow of Handling in the Sample Program (3)
Figure 15 Flow of Handling in the Sample Program (4)
Interrupt function
INT_EDMAC_EINT0

START

Read and clear the EtherC/E-DMAC status register (EESR)

Transmission-related interrupts occur?

Yes

Transmit interrupt handling
lan_send_handler

No

Yes

Reception-related interrupts occur?

No

Receive interrupt handling
lan_recv_handler

No

Yes

EtherC-related interrupts occur?

No

Clear the EtherC/E-DMAC status register (EESR)

EtherC status interrupt handling
lan_etherc_handler

Yes

END

END

Function for transmit interrupt handling
lan_send_handler

START

Clear the flag indicating transmission

END

Figure 16 Flow of Handling in the Sample Program (5)
3. Sample Program Listing

3.1 Sample program list "main.c" (1)

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

#include "iodefine.h"
#include "defs.h"
#include "ether.h"

/* **** Prototype Declaration **** */
void main(void);
```
3.2 Sample program list "main.c" (2)

```c
/* **** Variable Declaration **** */
static unsigned char s_frame[] = {
  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, /* Destination MAC address */
  0x00, 0x01, 0x02, 0x03, 0x04, 0x05, /* Source MAC address (00:01:02:03:04:05) */
  0x08, 0x06, /* Type (ARP) */
  0x00, 0x01, /* +--H/W type= Ethernet */
  0x08, 0x00, /* +--Protocol type= IP */
  0x06, 0x04, /* +--HW/protocol address length */
  0x00, 0x01, /* +--+PCODE= request */
  0x00, 0x01, 0x02, 0x03, 0x04, 0x05, /* +--Source MAC address (00:01:02:03:04:05) */
  0x08, 0x06, /* +--Protocol type= IP */
  0x06, 0x04, /* +--HW/protocol address length */
  0x00, 0x01, /* +--+PCODE= request */
  0x00, 0x01, 0x02, 0x03, 0x04, 0x05, /* +--Source MAC address (00:01:02:03:04:05) */
  0x08, 0x06, /* +--Protocol type= IP */
  0x06, 0x04, /* +--HW/protocol address length */
  0x00, 0x01, /* +--+PCODE= request */
};

/*"FUNC COMMENT"*****************************************************************
* ID          :
* Outline     : Ethernet transmission sample program main function
*------------------------------------------------------------------------------
* Include     : #include "iodefine.h"
*------------------------------------------------------------------------------
* Declaration : void main(void)
*------------------------------------------------------------------------------
* Function : On-chip ethernet controller (EtherC) and the dynamic memory access controller
*             : (E-DMAC)for the ethernet controller is used to transmit Ethernet frame.
*             : RTL8201CP from REALTEK is used for the PHY module.
*             : Multiple planes of transmit scripter is used for continuous transmission.
*------------------------------------------------------------------------------
* Argument    : void
*------------------------------------------------------------------------------
* ReturnValue : void
*------------------------------------------------------------------------------
* Notice      : Mac address acquired from EEPROM is not reflected on the transmission frame.
"""FUNC COMMENT END"*****************************************************************/
void main(void)
{
  int i;
  int ret;

  /* ==== Ethernet initial setting ==== */
  ret = lan_open();
  if( ret == OPEN_OK ){
    /* ==== 10-frame transmission start ==== */
    for(i=0; i<10; i++){
      /* ----transmission ---- */
      ret = lan_send( s_frame, sizeof(s_frame) );
      if( ret != SEND_OK )
        break;
    }
    /* ==== Ethernet transmission and reception stop ==== */
    lan_close();
  }
  /* End of file */
```
3.3 Sample program list "ether.c" (1)

```c
#include "machine.h"
#include "string.h"
#include "iodefine.h"
#include "defs.h"
#include "phy.h"
#include "ether.h"
#include "slic.h"
```

---

**DISCLAIMER**

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

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

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

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

Renesas reserves the right, without notice, to make changes to this software and to discontinue the availability of this software.

By using this software, you agree to the additional terms and conditions found by accessing the following link:

http://www.renesas.com/disclaimer

Copyright (C) 2008(2010) Renesas Electronics Corporation. All rights reserved.

FILE COMMENT*********** Technical reference data ***********************

System Name : SH7671 Sample Program
File Name : ether.c
Abstract : Setting for Transmission of Ethernet Frames
Version : 1.00.01
Device : SH7671
Tool-Chain : High-performance Embedded Workshop (Ver.4.03.00).
: C/C++ compiler package for the SuperH RISC engine family
: (Ver.9.01 Release01).
OS : None
H/W Platform: M3A-HS71(CPU board)
Description :

History : Mar.05,2008 ver.1.00.00
: Apr.07,2010 ver.1.00.01 Changed the company name and device name

FILE COMMENT END**************************************************************************
3.4 Sample program list "ether.c" (2)

```c
/* **** Macro declaration **** */
#define DEVADDR_EEPROM 0     /* Depends on the EEPROM PIN setting */
#define ROMADDR_MAC 0    /* MAC address storage location in EEPROM */
#define DEFAULT_MAC_H 0x00010203 /* For debugging */
#define DEFAULT_MAC_L 0x00000405
#define MACSET_OK    0
#define MACSET_NG    -1

/* **** Prototype declaration **** */
void main(void);
void lan_send_handler( unsigned long status );
static void lan_desc_create( void );
static void lan_reg_reset( void );
static void lan_reg_set( int link );
static int lan_set_mac( void );

/* **** Variable declaration **** */
/* ---- Discriptor ---- */
#pragma section ETH_DESC      /* Allocate to 16-byte boundary */
static volatile TXRX_DESCRIPTOR_SET desc;  /* Descriptor area */
#pragma section
/* ---- Buffer ---- */
#pragma section ETH_BUFF      /* Allocate to 16-byte boundary */
static volatile TXRX_BUFFER_SET buf;   /* Transmit/receive buffer area */
#pragma section
/* ---- MAC address ---- */
static unsigned long my_macaddr_h;
static unsigned long my_macaddr_l;
/* ---- Others ---- */
static volatile int f_send = 0;    /* Transmitting flag */
```

### 3.5 Sample program list "ether.c" (3)

```c
/*""FUNC COMMENT""*******************************************************
* ID    :
* Outline  : Ethernet open function
*---------------------------------------------------------------
* Include : #include "iodefine.h"
*     : #include "phy.h"
*     : #include "ether.h"
*---------------------------------------------------------------
* Declaration : int lan_open(void)
*---------------------------------------------------------------
* Function   : Initialize E-DMAC, EtherC, PHY, and buffer memory
*     : Initialization required for Ethernet is done within the function, and
*     : transmit/receive is enabled. If this fails, an error is returned.
*---------------------------------------------------------------
* Argument   : void
*---------------------------------------------------------------
* ReturnValue  : OPEN_OK(0)  : Open successful
*     : OPEN_NG(-1) : Open failed
*---------------------------------------------------------------
* Notice   :
""FUNC COMMENT END""***************************************************/

int lan_open(void)
{
    int link;
    
    // Setting for using the DK30686 board
    PORT.PBCRL1.BIT.PB6MD = 1;
    PORT.PCCRH1.WORD = 0x0155; /* EtherC function */
    PORT.PCCRL1.WORD = 0x5555;
    PORT.PCCRL2.WORD = 0x5555;

    /* Release EtherC/E-DMAC module standby mode */
    CPG.STBCR4.BIT.MSTP40 = 0;

    /* EtherC,E-DMAC halted */
    lan_reg_reset();

    /* Buffer initialization */
    lan_desc_create();

    /* Acquire MAC address */
    lan_set_mac();

    /* EtherC,E-DMAC setting */
    link = phy_autonego(); /* Check duplex mode */
    if( link == NEGO_FAIL ){
        return OPEN_NG; /* OPEN failed */
    } else{
        lan_reg_set(link);
    }
    return OPEN_OK;
}
```
3.6 Sample program list "ether.c" (4)

```c
/*""FUNC COMMENT"************************************************************
* ID    : 
* Outline : Ethernet close function
*-----------------------------------------------------------------------------
* Include : #include "iodefine.h"
*     : #include "ether.h"
*-----------------------------------------------------------------------------
* Declaration : int lan_close(void)
*-----------------------------------------------------------------------------
* Function : EDMAC/EtherC EDMAC/EtherC halted
*     : Clock supply to EDMAC/EtherC stops
*-----------------------------------------------------------------------------
* Argument : void
*-----------------------------------------------------------------------------
* ReturnValue : int CLOSE_OK( 0) : Close successful
*     : CLOSE_NG(-1)  : Close failed
*-----------------------------------------------------------------------------
* Notice :
""FUNC COMMENT END"*********************************************************/

int lan_close( void )
{
    int i;

    /* ==== Reset EtherC,E-DMAC === */
    lan_reg_reset();
    /* ==== EtherC,E-DMAC halted === */
    CPG.STBCR4.BIT.MSTP40 = 1;
    /* ==== E-DMAC-related interrupts are disabled=== */
    INTC.IPR12.BIT._ETC = 0;
    return CLOSE_OK;
}

/*""FUNC COMMENT"************************************************************
* ID    : 
* Outline : Frame transmission function
*-----------------------------------------------------------------------------
* Include : #include "ether.h"
*     : "iodefine.h"
*-----------------------------------------------------------------------------
* Declaration : int lan_send( unsigned char *addr, int flen )
*-----------------------------------------------------------------------------
* Function : Specified frame is copied and transmitted to the buffer registered
*     : in the transmit descriptor. Wait processing continues until transmission
*     : is completed. For safety, EDMAC is periodically monitored.
*-----------------------------------------------------------------------------
* Argument : None
*-----------------------------------------------------------------------------
* ReturnValue : SEND_OK(0) : Registration successful
*     : SEND_NG(-1) : Registration failed
*-----------------------------------------------------------------------------
* Notice :
""FUNC COMMENT END"*********************************************************/
```
3.7 Sample program list "ether.c" (5)

```c
int lan_send( unsigned char *addr, int flen )
{
    int i;
    int tlmso = 0;
    int t400ms = 0;
    /* ==== Check that data is not being transmitted ==== */
    while( desc.pSend_top->td0.BIT.TACT == 1 ){
        /* wait */
    }
    /* ==== Set the flag for transmission in process ==== */
    f_send = 1;           /* Clear by transmit-end interrupt */
    /* ==== Updated transmit descriptor ==== */
    memcpy( desc.pSend_top->td2.TBA, addr, flen ); /* Transmitted data */
    if( flen < 60 ){          /* Minimum frame of 60 bytes */
        memcpy( (desc.pSend_top->td2.TBA)+flen, 0, 60-flen ); /* Padding */
        flen = 60;
    }
    desc.pSend_top->td1.TDL = flen; /* Data length */
    desc.pSend_top->td0.BIT.TACT = 1; /* Transmission enabled */
    /* ==== Activate if transmission is stopped ==== */
    if( EDMAC.EDTRR.BIT.TR == 0 ){ /* Check by reading data */
        EDMAC.EDTRR.BIT.TR = 1;
    }
    /* ==== Check transmission completion ==== */
    while( f_send ){
        for( i=LOOP_100us; i>0; i-- ){ /* 100us wait */
        }
        /* ---- Check descriptor when 1 ms elapsed ---- */
        if( ++tlms > 10 ){ 
            tlmso = 0;
            if( desc.pSend_top->td0.BIT.TACT == 0 ){
                break;
            }
        }
        /* ---- If 400 ms has elapsed judge that EDMAC operation stopped ---- */
        if( ++t400ms > 4000 ){ 
            t400ms = 0;
            return SEND_NG;
        }
    }
    /* ==== Update current pointer ==== */
    desc.pSend_top = desc.pSend_top->pNext;
    return SEND_OK;
}
```
3.8 Sample program list “ether.c” (6)

```c
/*"FUNC COMMENT"****************************************************************************
* ID    :
* Outline : Descriptor configuration function
*-----------------------------------------------------------------------------
* Include : #include "ether.h"
*-----------------------------------------------------------------------------
* Declaration : static void lan_desc_create( void )
*-----------------------------------------------------------------------------
* Function : Initialize transmit/receive buffer required for Ethernet and
*     : initialize descriptor. One frame/one buffer is assumed.
*-----------------------------------------------------------------------------
* Argument : void
*-----------------------------------------------------------------------------
* ReturnValue : void
*-----------------------------------------------------------------------------
* Notice : 
"FUNC COMMENT END"************************************************************************/
static void lan_desc_create( void )
{
    int i;
    /* ===== Descriptor area configuration ===== */
    /* ---- Memory area ---- */
    memset(&desc, 0, sizeof(desc) );
    /* ---- Transmit descriptor ---- */
    for(i=0; i<NUM_OF_TX_DESCRIPTOR; i++){
        desc.send[i].td2.TBA = buf.send[i];  /* TD2 */
        desc.send[i].td1.TDL = 0;     /* TD1 */
        desc.send[i].td0.LONG= 0x30000000;  /* TD0:1frame/1buf, transmission disabled*/
        if( i != (NUM_OF_TX_DESCRIPTOR-1) ){   /* pNext */
            desc.send[i].pNext = &desc.send[i+1];
        }
    }
    desc.send[0].td0.BIT.TDLE = 1;   /* Set the last descriptor */
    /* ---- Receive descriptor ---- */
    for(i=0; i<NUM_OF_RX_DESCRIPTOR; i++){
        desc.recv[i].rd2.RBA = buf.recv[i];   /* RD2 */
        desc.recv[i].rd1.RBL = SIZE_OF_BUFFER;  /* RD1 */
        desc.recv[i].rd0.LONG= 0xb0000000;   /* RD0:1frame/1buf reception enabled */
        if( i != (NUM_OF_RX_DESCRIPTOR-1) ){   /* pNext */
            desc.recv[i].pNext = &desc.recv[i+1];
        }
    }
    desc.recv[0].rd0.BIT.RDLE = 1;
    /* ---- Buffer area configuration ---- */
    /* ---- Clear the area ---- */
    memset(&buf, 0, sizeof(buf) );
}
```

3.9 Sample program list "ether.c" (7)

```c
static void lan_reg_reset( void )
{
volatile int j = 100;      /* Wait for Bf 4 cycles */

    /* ---- Software reset ---- */
    EDMAC.EDMR.BIT.SWR = 1;
    /* ---- Secure reset period ---- */
    while(j--){
        /* Wait for Bf 4 cycles */
    }
}

static void lan_reg_set( void )
{
    volatile int j = 100;      /* Wait for Bf 4 cycles */

    /* ---- Software reset ---- */
    EDMAC.EDMR.BIT.SWR = 1;
    /* ---- Secure reset period ---- */
    while(j--){
        /* Wait for Bf 4 cycles */
    }
}
```

```c
static void lan_reg_set(int link)
{
    volatile int j = 100;      /* Wait for Bf 4 cycles */

    /* ---- Software reset ---- */
    EDMAC.EDMR.BIT.SWR = 1;
    /* ---- Secure reset period ---- */
    while(j--){
        /* Wait for Bf 4 cycles */
    }
}
```
3.10 Sample program list "ether.c" (8)

```c
static void lan_reg_set( int link )
{
    /* ==== EDMAC ==== */
    EDMAC.EDMR.LONG = 0x00000000;    /* Endian not changed (big endian) */
    /* descriptor length is 16 bytes */
    EDMAC.TDLAR = &desc.send[0];    /* Transmit descriptor start */
    EDMAC.RDLAR = &desc.recv[0];    /* Receive descriptor start */
    EDMAC.TRSCER.LONG = 0x00000000;    /* Copy all status to descriptor */
    EDMAC.TFTR = 0x00;       /* Transmit FIFO threshold: store&forward */
    EDMAC.FDR.BIT.TFD = 1;      /* Receive FIFO threshold: store&forward */
    EDMAC.RMCR.BIT.RNC = 1;      /* Transmit FIFO capacity of 512 bytes */
    EDMAC.RMCR.BIT.RNC = 1;      /* Receive FIFO capacity of 512 bytes */
    EDMAC.EDOCR.LONG = 0x00000000;    /* Continuous reception enabled */
    EDMAC.FCFTR.LONG = 0x00070000;    /* Operation continues on FIFO error */
    EDMAC.TRMD.BIT.TIS = 1;    /* Flow control threshold setting, disabled by EtherC */
    /* ==== EtherC ==== */
    EtherC.ECMR.LONG = 0x00000000;    /* Flow control disabled */
    /* CRC frame is recognized as an error */
    /* Magic Packet detection is disabled */
    /* Reception disabled */
    /* Transmission disabled */
    /* No internal loopback */
    /* No external loopback */
    /* Duplex mode (half-duplex mode) */
    if( link == FULL_TX || link == FULL_10M ){
        EtherC.ECMR.BIT.DM = 1;    /* Set to full-duplex mode */
        EtherC.ECMR.BIT.RE = 1;      /* Reception enabled */
        EtherC.ECMR.BIT.TE = 1;      /* Transmission enabled */
        if(EDMAC.EDRRR.BIT.RR == 0){
            EDMAC.EDRRR.BIT.RR = 0;    /* Reception disabled */
        }
    }
    if( link == FULL_TX || link == FULL_10M ){
        EtherC.MAHR = my_macaddr_h;     /* MAC address setting */
        EtherC.TABLE = my_macaddr_l;
        EtherC.RFLR = 0x000;      /* Maximum receive frame length of 1518 bytes */
        EtherC.IPGR = 0x14;       /* Gap between packets (96-bit period) */
        /* ---- Interrupt-related ---- */
        EDMAC.EESR.LONG = 0x47FFFDFF;
        /* Clear all status (clear by writing 1) */
        EDMAC.EESIPR.LONG = EDMAC_EESIPR_INI_SEND | EDMAC_EESIPR_INI_RECV |
        EDMAC_EESIPR_INI_EtherC;
        /* Transmit/receive enable setting */
        EtherC.ECSR.LONG = 0x00000017;   /* Transmit/receive and EtherC interrupts enabled */
        EtherC.ECSR.LONG = 0x00000017;   /* Transmit/receive and EtherC interrupts enabled */
        EtherC.ECSR.LONG = 0x00000017;   /* Transmit/receive and EtherC interrupts enabled */
        INTC.IPR12.BIT._ETC = 5; /* E-DMAC (EINT0) interrupt priority level */
        /* ==== Set to enable transmission/reception ==== */
        /* ---- EtherC ---- */
        EtherC.ECMR.BIT.BR = 1;         /* Reception enabled */
        EtherC.ECMR.BIT.TE = 1;         /* Transmission enabled */
        /* ---- E-DMAC ---- */
        if(EDMAC.EDRRR.BIT.RR == 0){
            EDMAC.EDRRR.BIT.RR = 0;    /* Reception disabled */
        }
    }
}
```
3.11 Sample program list "ether.c" (9)

```c
/*""FUNC COMMENT"************************************************************
* ID    :
* Outline : Transmit interrupt function
*-----------------------------------------------------------------------------
* Include : #include "iodefine.h"
*      : #include "ether.h"
*-----------------------------------------------------------------------------
* Declaration : void lan_send_handler( unsigned long status )
*-----------------------------------------------------------------------------
* Function : Interrupt handler related to transmission regarding EDMAC(EESR)
*      :
*-----------------------------------------------------------------------------
* Argument   : unsigned long status : I : EESR state (interrupt-enabled bits only)
*-----------------------------------------------------------------------------
* ReturnValue  : None
*-----------------------------------------------------------------------------
* Notice   :
*""FUNC COMMENT END"*********************************************************/

void lan_send_handler( unsigned long status )
{
    /* ==== Clear the flag for transmission in progress ==== */
    f_send = 0;
}

/*""FUNC COMMENT"************************************************************
* ID    :
* Outline : Interrupt handler related to reception regarding EDMAC (EESR)
*-----------------------------------------------------------------------------
* Include : #include "iodefine.h"
*      : #include "ether.h"
*-----------------------------------------------------------------------------
* Declaration : void lan_recv_handler( unsigned long status )
*-----------------------------------------------------------------------------
* Function : EDMAC(EESR) EESR state (interrupt-enabled bits only)
*      :
*-----------------------------------------------------------------------------
* Argument   : unsigned long status : I : EESR state (interrupt-enabled bits only)
*-----------------------------------------------------------------------------
* ReturnValue  : none
*-----------------------------------------------------------------------------
* Notice   : Nothing is done by this sample task
*""FUNC COMMENT END"*********************************************************/

void lan_recv_handler( unsigned long status )
{
}
```
3.12 Sample program list "ether.c" (10)

```c
/*""FUNC COMMENT""************************************************************
* ID    :
* Outline : EtherC interrupt function
*-----------------------------------------------------------------------------
* Include : #include "iodefine.h"
*      : #include "ether.h"
*-----------------------------------------------------------------------------
* Declaration : void lan_etherc_handler( unsigned long status )
*-----------------------------------------------------------------------------
* Function : Interrupt handler regarding EtherC(ECR)
*      :
*-----------------------------------------------------------------------------
* Argument : unsigned long status : I : ECSR state (interrupt-enabled bits only)
*-----------------------------------------------------------------------------
* ReturnValue : none
*-----------------------------------------------------------------------------
* Notice   : Nothing is done by this sample task
*""FUNC COMMENT END""*********************************************************/
void lan_etherc_handler( unsigned long status )
{
}

/*""FUNC COMMENT""************************************************************
* ID    :
* Outline : MAC address setting
*-----------------------------------------------------------------------------
* Include : #include "siic.h"
*-----------------------------------------------------------------------------
* Declaration : static int lan_set_mac( void )
*-----------------------------------------------------------------------------
* Function : Function to obtain MAC address from EEPROM
*-----------------------------------------------------------------------------
* Argument : None
*-----------------------------------------------------------------------------
* ReturnValue : MACSET_OK(0) : Acquisition successful
*      : MACSET_NG(-1) : Acquisition failed
*-----------------------------------------------------------------------------
* Notice   :
*""FUNC COMMENT END""*********************************************************/
```
3.13 Sample program list "ether.c" (11)

```c
static int lan_set_mac( void )
{
    volatile int ret, i;
    unsigned char buf[10];
    /* ==== EEPROM driver initial setting ==== */
    siic_Init_Driver();
    /* ==== Read data from EEPROM ==== */
    ret = siic_EepRomRW(DEVADDR_EEPROM, ROMADDR_MAC, 6, buf, SIIC_MODE_EEP_READ);
    if (ret < SIIC_OK) {
        /* ---- Read failed ---- */
        my_macaddr_h = DEFAULT_MAC_H;
        my_macaddr_l = DEFAULT_MAC_L;
        return MACSET_NG;
    }
    do{
        ret = siic_Chk_Eep();
        if( ret < SIIC_OK ){
            /* ---- Read failed ---- */
            my_macaddr_h = DEFAULT_MAC_H;
            my_macaddr_l = DEFAULT_MAC_L;
            return MACSET_NG;
        }
    }while( ret != SIIC_OK);
    /* ---- Read successful ---- */
    for(i=0; i<6; i++){
        if( buf[i] != 0xff ){
            break;
        }
    }
    if( i == 6 ){
        /* ---- Set the default value when EEPROM is not set ---- */
        my_macaddr_h = DEFAULT_MAC_H;
        my_macaddr_l = DEFAULT_MAC_L;
    }
    else{
        /* ---- Set the read address ---- */
        my_macaddr_h = buf[0];
        my_macaddr_h <<= 8;
        my_macaddr_h |= buf[1];
        my_macaddr_h <<= 8;
        my_macaddr_h |= buf[2];
        my_macaddr_h <<= 8;
        my_macaddr_h |= buf[3];
        my_macaddr_l <<= 8;
        my_macaddr_l |= buf[4];
        my_macaddr_l <<= 8;
        my_macaddr_l |= buf[5];
    }
    return MACSET_OK;
}
/* End of file */
```
3.14 Sample program list "ether.h" (1)

```c
#ifndef _ETHER_H
#define _ETHER_H

/* **** Macro definition **** */
#define NUM_OF_TX_DESCRIPTOR   4
#define NUM_OF_RX_DESCRIPTOR   4
#define NUM_OF_TX_BUFFER    4
#define NUM_OF_RX_BUFFER    4
#define SIZE_OF_BUFFER    1520    /* Must be an integral multiple of 16 */
#endif
```
### 3.15 Sample program list "ether.h" (2)

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td><code>#define OPEN_OK      0</code></td>
</tr>
<tr>
<td>56</td>
<td><code>#define OPEN_NG     -1</code></td>
</tr>
<tr>
<td>57</td>
<td><code>#define SEND_OK     0</code></td>
</tr>
<tr>
<td>58</td>
<td><code>#define SEND_NG     -1</code></td>
</tr>
<tr>
<td>59</td>
<td><code>#define CLOSE_OK    0</code></td>
</tr>
<tr>
<td>60</td>
<td><code>#define CLOSE_NG    -1</code></td>
</tr>
<tr>
<td>61</td>
<td><code>#define MIN_FRAME_SIZE    60</code></td>
</tr>
<tr>
<td>62</td>
<td><code>#define MAX_FRAME_SIZE    1514</code></td>
</tr>
<tr>
<td>63</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td><code>#define EDMAC_EESIPR_INI_SEND 0x40000000 : Write-back completed</code></td>
</tr>
<tr>
<td>65</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td><code>* 0x04000000 : Detect transmit suspended </code></td>
</tr>
<tr>
<td>67</td>
<td><code>* Not used 0x00200000 : Frame transmission completed*</code></td>
</tr>
<tr>
<td>68</td>
<td><code>* 0x00000000 : transmit FIFO underflow</code></td>
</tr>
<tr>
<td>69</td>
<td><code>* 0x00000000 : Carrier not detected</code></td>
</tr>
<tr>
<td>70</td>
<td><code>* 0x00000000 : Carrier lost detected</code></td>
</tr>
<tr>
<td>71</td>
<td><code>* 0x00000000 : Delayed collision detected</code></td>
</tr>
<tr>
<td>72</td>
<td>`* 0x00000000 : Transmit retry-over condition*</td>
</tr>
<tr>
<td>73</td>
<td><code>* 0x00000000 : Detect reception suspended */</code></td>
</tr>
<tr>
<td>74</td>
<td><code>#define EDMAC_EESIPR_INI_RECV 0x002000000 : Detect frame reception</code></td>
</tr>
<tr>
<td>75</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td><code>* 0x00000000 : Frame reception</code></td>
</tr>
<tr>
<td>77</td>
<td><code>* 0x00000000 : Receive FIFO overflow</code></td>
</tr>
<tr>
<td>78</td>
<td><code>* 0x00000000 : Residual bit frame reception</code></td>
</tr>
<tr>
<td>79</td>
<td><code>* 0x00000000 : Long frame reception</code></td>
</tr>
<tr>
<td>80</td>
<td><code>* 0x00000000 : Short frame reception</code></td>
</tr>
<tr>
<td>81</td>
<td><code>* 0x00000000 : PHY-LSI reception error</code></td>
</tr>
<tr>
<td>82</td>
<td><code>* 0x00000000 : Receive frame CRC error */</code></td>
</tr>
<tr>
<td>83</td>
<td><code>#define EDMAC_EESIPR_INI_EtherC 0x004000000 : EtherC status register */</code></td>
</tr>
<tr>
<td>84</td>
<td><code>#define EtherC_EESIPR_INI 0x000000004 : Ling signal change */</code></td>
</tr>
<tr>
<td>85</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td><code>/* **** Type definition **** */</code></td>
</tr>
<tr>
<td>87</td>
<td>`typedef union{</td>
</tr>
<tr>
<td>88</td>
<td>`    unsigned long LONG;</td>
</tr>
<tr>
<td>89</td>
<td>`    struct{</td>
</tr>
<tr>
<td>90</td>
<td>`      unsigned int TACT:1; /* Transmit descriptor enabled */</td>
</tr>
<tr>
<td>91</td>
<td>`      unsigned int TDLE:1; /* Transmit descriptor end */</td>
</tr>
<tr>
<td>92</td>
<td>`      unsigned int TFP :2; /* Location 1, 0 within transmit frame */</td>
</tr>
<tr>
<td>93</td>
<td>`      unsigned int TFE :1; /* Transmit frame error */</td>
</tr>
<tr>
<td>94</td>
<td>`      unsigned int reserved :23; /* TFS26 to 4 (reserved) */</td>
</tr>
<tr>
<td>95</td>
<td>`      unsigned int TFS3:1; /* No carrier is detected (CND bit in EESR) */</td>
</tr>
<tr>
<td>96</td>
<td>`      unsigned int TFS2:1; /* Carrier lost is detected (DLC bit in EESR) */</td>
</tr>
<tr>
<td>97</td>
<td>`      unsigned int TFS1:1; /* Delayed collision detected during transmission (CD bit in EESR) */</td>
</tr>
<tr>
<td>98</td>
<td><code>      unsigned int TFS0:1; /* Transmit retry over condition (TRO bit in EESR)*/</code></td>
</tr>
<tr>
<td>99</td>
<td>`</td>
</tr>
<tr>
<td>100</td>
<td>`</td>
</tr>
<tr>
<td>101</td>
<td>`    typedef struct{</td>
</tr>
<tr>
<td>102</td>
<td>`      unsigned short TDL; /* Transmit buffer data length */</td>
</tr>
<tr>
<td>103</td>
<td>`      unsigned short reserved;</td>
</tr>
<tr>
<td>104</td>
<td>`</td>
</tr>
</tbody>
</table>
### 3.16 Sample program list "ether.h" (3)

```c
typedef struct{
  unsigned char *TBA;    /* Address of transmit buffer */
}TD2;

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

/* ==== xxxxxxxxxxxxxxx ==== */

typedef union{
  unsigned long LONG;
  struct{
    unsigned int RACT:1;     /* Receive descriptor enabled */
    unsigned int RDLE:1;     /* End of receive descriptor*/
    unsigned int RFP :2;     /* Location 1,0 within receive frame */
    unsigned int RFE :1;     /* Receive frame error */
    unsigned int reserved1:17;   /* TFS26 to 10: reserved */
    unsigned int RFS9:1;      /* Receive FIFO overflow (RFOF bit in EESR)*/
    unsigned int reserved2:1;    /* Reserved */
    unsigned int RFS7:1;      /* Receive multicast frames (RMAF bit in EESR)*/
    unsigned int reserved3:1;   /* Reserved */
    unsigned int reserved4:1;   /* Reserved */
    unsigned int RFS4:1;      /* Residual bits frame receive error (RRF bit in EESR)*/
    unsigned int RFS3:1;      /* Long frame receive error (RTLE bit in EESR)*/
    unsigned int RFS2:1;      /* Short frame receive error (RTSF bit in EESR)*/
    unsigned int RFS1:1;      /* PHY-LSI receive error (PRE bit in EESR)*/
    unsigned int RFS0:1;      /* Receive frame CRC error detected (CERF bit in EESCR)*/
  }BIT;
}RD0;

typedef struct{
  unsigned short RBL;        /* Receive buffer length */
  unsigned short RDL;        /* Receive data length */
}RD1;

typedef struct{
  unsigned char *RBA;        /* Receive buffer address */
}RD2;

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

/*== The whole transmit/receive descriptors (must be allocated in 16-byte boundaries) ==*/
typedef struct{
  EDMAC_SEND_DESC send[NUM_OF_TX_DESCRIPTOR];
  EDMAC_RECV_DESC recv[NUM_OF_RX_DESCRIPTOR];
  EDMAC_SEND_DESC *pSend_top;  /* Registration location of transmit descriptors */
  EDMAC_RECV_DESC *pRecv_end;  /* Registration location and reception end of transmit
descriptors */
}TXRX_DESCRIPTOR_SET;
```
3.17 Sample program list "ether.h" (4)

```c
/* ==== Transmit/receive buffers (must be allocated in 16-byte boundaries) ==== */
/* ---- Definition of all transmit/receive buffer areas ---- */
typedef struct{
    unsigned char send[NUM_OF_TX_BUFFER][SIZE_OF_BUFFER];
    unsigned char recv[NUM_OF_RX_BUFFER][SIZE_OF_BUFFER];
} TXRX_BUFFER_SET;

/* **** Prototype Declaration **** */
int lan_open( void );
int lan_close( void );
int lan_send( unsigned char *addr, int flen );

#endif
/* End of File */
```
### 3.18 Sample program list "intprg_eth.c" (1)

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

** This software is supplied by Renesas Electronics Corporation and is only **
** intended for use with Renesas products. No other uses are authorized. **
** This software is owned by Renesas Electronics Corporation and is protected under **
** all applicable laws, including copyright laws. **
** THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES **
** REGARDING THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY, **
** INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A **
** PARTICULAR PURPOSE AND NON-INFRINGEMENT. ALL SUCH WARRANTIES ARE EXPRESSLY **
** DISCLAIMED. **
** TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS **
** ELECTRONICS CORPORATION NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE **
** FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES **
** FOR ANY REASON RELATED TO THIS SOFTWARE, EVEN IF RENESAS OR ITS **
** AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. **
** Renesas reserves the right, without notice, to make changes to this **
** software and to discontinue the availability of this software. **
** By using this software, you agree to the additional terms and **
** conditions found by accessing the following link: **
** http://www.renesas.com/disclaimer **
****************************************************************************
** Copyright (C) 2007(2010) Renesas Electronics Corporation. All rights reserved. **
** "FILE COMMENT"**************** Technical reference data ***********************
** System Name : SH7671 Sample Program **
** File Name : intprg_eth.c **
** Abstract : interrupt entry function **
** Version : 1.00.01 **
** Device : SH7671 **
** Tool-Chain : High-performance Embedded Workshop (Ver.4.03.00). **
** : C/C++ compiler package for the SuperH RISC engine family **
** : (Ver.9.01 Release01). **
** OS : None **
** H/W Platform: M3A-HS71(CPU board) **
** Description : **
****************************************************************************
** History : Sep.18,2007 ver.1.00.00 **
** : May 10,2010 ver.1.00.01 Changed the company name and device name **
** "FILE COMMENT END"******************************************************************
```
3.19 Sample program list "intprg_eth.c" (2)

```c
// 171 ETC EINT0
void INT_ETC_EINT0(void)
{
    unsigned long stat_edmac;
    unsigned long stat_EtherC;

    /* ---- Clear the interrupt request flag ---- */
    stat_edmac = EDMAC.EESR.LONG & EDMAC.EESIPR.LONG;
    EDMAC.EESR.LONG = stat_edmac;
    /* Targets are restricted to allowed interrupts */

    if(stat_edmac & EDMAC_EESIPR_INI_SEND ){
        lan_send_handler(stat_edmac & EDMAC_EESIPR_INI_SEND);
    }

    /* ==== Transmission-related ==== */
    if( stat_edmac & EDMAC_EESIPR_INI_RECV ){
        lan_recv_handler( stat_edmac & EDMAC_EESIPR_INI_RECV );
    }

    /* ==== EtherC-related ==== */
    if( stat_edmac & EDMAC_EESIPR_INI_EtherC ){
        /* ---- Clear the interrupt request flag ---- */
        stat_EtherC = EtherC.ECSR.LONG & EtherC.ECSIPR.LONG;
        EtherC.ECSR.LONG = stat_EtherC;
        lan_etherc_handler(stat_EtherC);
    }
}
```

(omitted)
4. References

- Software Manual
  SH-2A/SH2A-FPU Software Manual Rev. 3.00
  The latest version of the software manual can be downloaded from the Renesas Electronics website.

- Hardware Manual
  SH7670 Group Hardware Manual Rev. 2.00
  The latest version of the hardware user's manual can be downloaded from the Renesas Electronics website.
Website and Support

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

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

All trademarks and registered trademarks are the property of their respective owners.
## Revision Record

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Description</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Dec.24.08</td>
<td>—</td>
<td></td>
<td>First edition issued</td>
</tr>
<tr>
<td>1.01</td>
<td>Oct.15.10</td>
<td>—</td>
<td></td>
<td>Changed the sample program ( AC Switching Characteristics are removed )</td>
</tr>
</tbody>
</table>
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.
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 or by 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 specifying 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 products 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" without 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.

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, heating 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 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 products" means any product developed or manufactured by or for Renesas Electronics.