RZ/A2M Group

Ethernet Driver

Introduction

This application note describes RZ/A2M Ethernet driver. This driver performs Ethernet frame transmission and reception using an Ethernet controller and an Ethernet controller DMA controller. In the remainder of this document, this module is called the Ethernet driver.

In order to use the Ethernet driver, assign input and output signals for Ethernet Controller to I/O Ports. Refer to section 4 Pin Setting in detail.

Target Devices

This API supports the following devices.

RZ/A2M

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Contents

1. Overview

The Ethernet driver uses an Ethernet controller (ETHERC) and an Ethernet controller DMA controller (EDMAC) to implement Ethernet frame transmission and reception. The Ethernet driver supports the following functions.

- MII (Media Independent Interface) and RMII (Reduced Media Independent Interface)
- An automatic negotiating function is used for the Ethernet PHY-LSI link.
- The link state is detected using the link signals output by the Ethernet PHY-LSI.
- The result of the automatic negotiation is acquired from the Ethernet PHY-LSI and the connection mode (full or half duplex, 10 or 100 Mbps transfer rate) is set in the ETHERC.

1.1 Ethernet driver

The Ethernet driver is implemented in a project and used as the API. Refer to “5 How to use” for details on implementing the module to the project.

1.2 Outline of the API

Table 1.1 lists the API functions included in the Ethernet driver.

Table 1.1 API Functions

Function	Contents
R_ETHER_Initial()	Initializes the Ethernet driver.
R_ETHER_Open_ZC2()	Applies a software reset to the ETHERC, EDMAC, and PHY-LSI, after which it starts PHY-LSI auto-negotiation and enables the link signal change interrupt.
R_ETHER_Close_ZC2()	Disables transmit and receive functionality on the ETHERC. Does not put the ETHERC and EDMAC into the module stop state.
R_ETHER_Read()	Receives data in the specified receive buffer.
R_ETHER_Read_ZC2()	Returns a pointer to the start address of the buffer that holds the receive data.
R_ETHER_Read_ZC2_BufRelease()	Releases the buffer read with the R_ETHER_Read_ZC2() function.
R_ETHER_Write()	Transmits data from the specified transmit buffer.
R_ETHER_Write_ZC2_GetBuf()	Returns a pointer to the start address of the write destination for transmit data.
R_ETHER_Write_ZC2_SetBuf()	Enables transmission of the transmit buffer data to the EDMAC.
R_ETHER_CheckLink_ZC()	Checks the link state of a physical Ethernet using the PHY management interface. If the PHY is connected to an appropriately initialized remote device with a cable, the Ethernet link state becomes link-up.
R_ETHER_LinkProcess()	Performs link signal change detected and magic packet detected interrupt handling.
R_ETHER_WakeOnLAN()	Switches the ETHERC setting from normal transmission and reception to magic packet detected operation.
R_ETHER_CheckWrite()	Verifies that data transmission has completed.
R_ETHER_Control()	Performs the processing that corresponds to a specified control code.
R_ETHER_GetVersion()	Returns the Ethernet driver version.

2. API Information

The API functions of the Ethernet driver adhere to the Renesas API naming standards.

2.1 Supported Toolchains

The operation of the Ethernet driver has been confirmed with the toolchain listed as C compiler in 6. Operation Confirmation Conditions.

2.2 Usage of Interrupt

EINT0 interrupt, or EINT1 interrupt corresponding to the channel number is enabled after specified argument to channel number and calling R_ETHER_Open_ZC2 function.

2.3 Header Files

All API calls and their supporting interface definitions are located in r_ether_rza2_if.h.

2.4 Integer Types

This project uses ANSI C99. These types are defined in stdint.h.

2.5 Configuration Overview

The configuration options in the Ethernet driver are specified by Smart Configurator. The setting value is automatically reflected in r_ether_drv_sc_cfg.h when adding modules to user project. The option names and setting values are listed in the table below.

Configuration options configured by Smart Configurator.	
Ethernet channel 0 Note: Default value = “Enabled”(1)	Enable or disable Ethernet channel 0. If set to “Disabled”(0), EHERC channel 0 is not used. If set to “Enabled”(1), EHERC channel 0 is used.
Ethernet channel 1 Note: Default value = “Enabled”(1)	Enable or disable Ethernet channel 1. If set to “Disabled”(0), EHERC channel 1 is not used. If set to “Enabled”(1), EHERC channel 1 is used.
Ethernet interface select. Note: Default value = “RMII”(1)	Sets the interface between ETHERC and the Ethernet PHY-LSI. If set to “MII”(0), MII (Media Independent Interface) is selected. If set to “RMII”(1), RMII (Reduced Media Independent Interface) is selected.
PHY-LSI address setting for ETHER0. Note: Default value = 7	Specify the PHY-LSI address used by ETHERC channel 0. Specify a value between 0 and 31.
PHY-LSI address setting for ETHER1. Note: Default value = 7	Specify the PHY-LSI address used by ETHERC channel 1. Specify a value between 0 and 31.
Loop count for write PIR register. Note: Default value = 8	Specify the loop count of software loop used for reading or writing in PHY-LSI. Set the number of loops according to the PHY-LSI to be used. Specify a value of 1 or greater.
Define the waiting time for reset completion of PHY-LSI Note: Default value = 0x00020000	Specify the loop count used for timeout processing of PHY-LSI reset completion wait. Set the number of loops according to the PHY-LSI to be used.
Link status read from LMON bit of ETHERC PSR register. Note: Default value = “(Link up/Link down)=(Falling/Rising)” (0)	Specify the polarity of the link signal output by the PHY-LSI. When “(Link up/Link down)=(Falling/Rising)”(0) is specified, link-up and link-down correspond respectively to the fall and rise of the LINKSTA signal. When “(Link up/Link down)=(Rising/Falling)” (1) is specified, link-up and link-down correspond respectively to the rise and fall of the LINKSTA signal.
LINKSTA signal for detect link status changes Note: Default value = “the PHY-LSI status register is used.”(0)	Specify whether or not to use the PHY-LSI status register instead of the LINKSTA signal when a change in the link status is detected. When “the PHY-LSI status register is used.”(0) is specified, the PHY-LSI status register is used. When “the LINKSTA signal is used.”(1) is specified, the LINKSTA signal is used.
Specify the difference between physical address and virtual address of the uncached RAM. Note: Default value = 0x02000000	Specify the difference between physical address and virtual address of the uncached RAM. If MMU is not used, specify 0. If physical address is same to virtual address, specify 0. If the start address of physical address of un-cached RAM is 0x80000000 and the start address of virtual address if un-cached RAM is 0x82000000, specify 0x02000000.

Configuration options in *r_ether_drv_sc_cfg.h*.	
#define ETHER_CFG_EMAC_RX_DESCRIPTORS Note: Default value = 8	Sets the number of receive descriptors. This must be set to a value 1 or greater
#define ETHER_CFG_EMAC_TX_DESCRIPTORS Note: Default value = 8	Sets the number of transmit descriptors. This must be set to a value 1 or greater
#define ETHER_CFG_BUFSIZE Note: Default value = 1,536	Specify the size of the transmit buffer or receive buffer. The buffer is aligned with 32-byte boundaries, so specify a value that is a multiple of 32 bytes.

2.6 Arguments

This section documents the enumerations, unions, and structures used as arguments to API functions. These are included in the r_ether_rza2_if.h header file along with the API function prototype declarations.

```
typedef enum
{
  CONTROL_SET_CALLBACK,            /* Callback function registration */
  CONTROL_SET_PROMISCUOUS_MODE,
                                   /* Promiscuous mode setting */
  CONTROL_SET_INT_HANDLER,         /* Interrupt handler function registration */
  CONTROL_POWER_ON,                /* Cancel ETHERC/EDMAC module stop */
  CONTROL_POWER_OFF,               /* Transition to ETHERC/EDMAC module stop */
  CONTROL_MULTICASTFRAME_FILTER,   /* Multicast frame filter setting */
  CONTROL_BROADCASTFRAME_FILTER    /* Broadcast frame filter continuous receive count setting */
} ether_cmd_t;

typedef union
{
   ether_cb_t           ether_callback;          /* Callback function pointer */
   ether_promiscuous_t  * p_ether_promiscuous;   /* Promiscuous mode setting */
   ether_cb_t           ether_int_hnd;           /* Interrupt handler function pointer */
   uint32_t             channel;                 /* ETHERC channel number */
   ether_multicast_t    * p_ether_multicast;
                                                 /* Multicast frame filter setting */
   ether_broadcast_t    * p_ether_broadcast;
                                                 /* Broadcast frame filter setting */
} ether_param_t;

typedef struct
{
  void   (*pcb_func)(void *);     /* Callback function pointer */
  void   (*pcb_int_hnd)(void *); /* Interrupt handler function pointer */
} ether_cb_t;

typedef enum
{
  ETHER_PROMISCUOUS_OFF,           /* ETHERC operates in standard mode */
  ETHER_PROMISCUOUS_ON             /* ETHERC operates in promiscuous mode */
} ether_promiscuous_bit_t;

typedef enum
{
    ETHER_MC_FILTER_OFF,           /* Disable multicast frame filter */
    ETHER_MC_FILTER_ON             /* Enable multicast frame filter */
} ether_mc_filter_t;

typedef struct
{
  uint32_t                 channel;     /* ETHERC channel */
  ether_promiscuous_bit_t  bit;         /* Promiscuous mode */
} ether_promiscuous_t;
```

```
typedef struct
{
   uint32_t             channel;     /* ETHERC channel */
   ether_mc_filter_t    flag;        /* Multicast frame filter setting */
} ether_multicast_t;

typedef struct
{
   uint32_t             channel;     /* ETHERC channel */
   uint32_t             counter;     /* Broadcast frame continuous receive count */
} ether_broadcast_t;

typedef enum
{
  ETHER_CB_EVENT_ID_WAKEON_LAN,      /* Magic packet detection */
  ETHER_CB_EVENT_ID_LINK_ON,         /* Link up detection */
  ETHER_CB_EVENT_ID_LINK_OFF         /* Link down detection */
} ether_cb_event_t;
typedef struct
{
  uint32_t            channel;       /* ETHERC channel */
  ether_cb_event_t    event_id;      /* Event code for callback function */
  uint32_t            status_ecsr;   /* ETHERC status register for interrupt handler */
  uint32_t            status_eesr;   /* ETHERC/EDMAC status register for interrupt handler */
} ether_cb_arg_t;
```

2.7 Return Values

This section describes return values of API functions. This enumeration is located in r_ether_rza2_if.h as are the prototype declarations of API functions.

```
typedef enum                /* Error code of Ether API */
{
ETHER_SUCCESS,              /* Processing completed successfully */
ETHER_ERR_INVALID_PTR,      /* Value of the pointer is NULL */
ETHER_ERR_INVALID_DATA,     /* Value of the argument is out of range */
ETHER_ERR_INVALID_CHAN,     /* Nonexistent channel number */
ETHER_ERR_INVALID_ARG,      /* Invalid argument */
ETHER_ERR_LINK,             /* Auto-negotiation is not completed, and transmission/reception is not enabled. */
ETHER_ERR_MPDE,             /* As a Magic Packet is being detected, and transmission/reception is not enabled. */
ETHER_ERR_TACT,             /* Transmit buffer is not empty. */
ETHER_ERR_CHAN_OPEN,        /* Indicates the Ethernet cannot be opened because it is being used by another application */
ETHER_ERR_MC_FRAME,         /* Multicast frame detected when multicast frame filtering is enabled. */
ETHER_ERR_RECV_ENABLE,      /* Could not change setting because receive function is enabled. */
ETHER_ERR_OTHER             /* Other error */
} ether_return_t;
```

2.8 Callback Function

2.8.1 Callback Function Called by API Function R_ETHER_LinkProcess

In the Ethernet driver, a callback function is called when either a magic packet or a link signal change is detected.

To set up the callback function, use the function R_ETHER_Control(), which is described later in this document, and set the control code CONTROL_SET_CALLBACK as the enumeration (the first argument) described in 2.6 Arguments, and set the address of the function to be registered as the callback function in the structure (the second argument).

When the callback function is called, a variable in which the channel number for which the detection occurred and a constant shown in Table 2.1 are stored is passed as an argument. If the value of this argument is to be used outside the callback function, it's value should be copied into, for example, a global variable.

Table 2.1 Argument List of the callback Function

Constant Definition	Description
ETHER_CB_EVENT_ID_WAKEON_LAN	Detect magic packet
ETHER_CB_EVENT_ID_LINK_ON	Detect link signal change (link-up)
ETHER_CB_EVENT_ID_LINK_OFF	Detect link signal change (link-down)

2.8.2 Callback Function Called by EINT0/EINT1 Status Interrupts

The Ethernet driver calls an interrupt handler when an interrupt indicating a condition other than the following occurs.

- Magic Packet detection operation by the Ethernet driver
  - Link signal change detection*1
  - Magic packet detection
- Normal operation by the Ethernet driver
  - Link signal change detection*1
  - Frame receive detection or frame transmit end detection

To specify the interrupt handler, use the R_ETHER_Control function described below to set the control code "CONTROL_SET_INT_HANDLER" in the enumeration (first argument) shown in 2.6 Arguments, and set the function address of the interrupt handler to be registered in the structure (second argument).

When the interrupt handler function is called, variables in which are stored the number of the channel on which the interrupt occurred, the ETHERC status register value, and the ETHERC/EDMAC status register value are passed as arguments. To use the argument values in functions other than the callback function, copy them to global variables or the like.

Note 1. If the setting of #define ETHER_CFG_USE_LINKSTA is 0, the interrupt handler function is not called when a link signal change is detected.

2.9 Ethernet Frame Format

The Ethernet driver supports the Ethernet II/IEEE 802.3 frame format.

2.9.1 Frame Format for Data Transmission and Reception

Figure 2.1 shows the Ethernet II/IEEE 802.3 frame format.

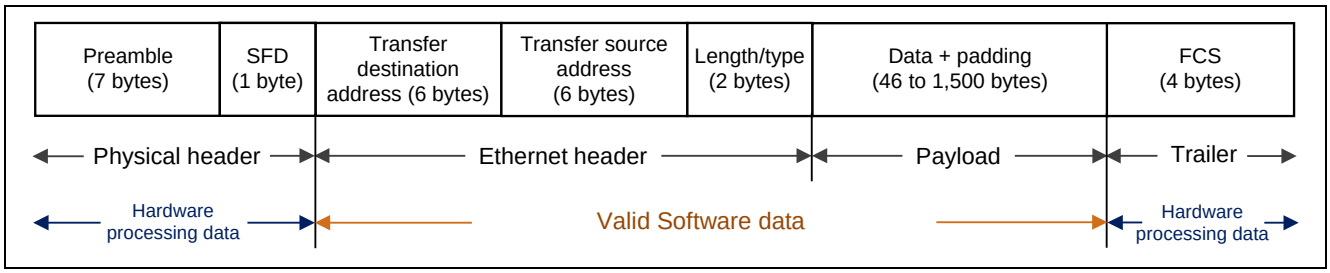


Figure 2.1 Ethernet II/IEEE 802.3 Frame Format

The preamble and SFD signal the start of an Ethernet frame. The FCS contains the CRC of the Ethernet frame and is calculated on the transmitting side. When data is received the CRC value of the frame is calculated in hardware, and the Ethernet frame is discarded if the values do not match.

When the hardware determines that the data is normal, the valid range of receive data is: (transmission destination address) + (transmission source address) + (length/type) + (data).

2.9.2 PAUSE Frame Format

Table 2.2 shows the PAUSE frame format.

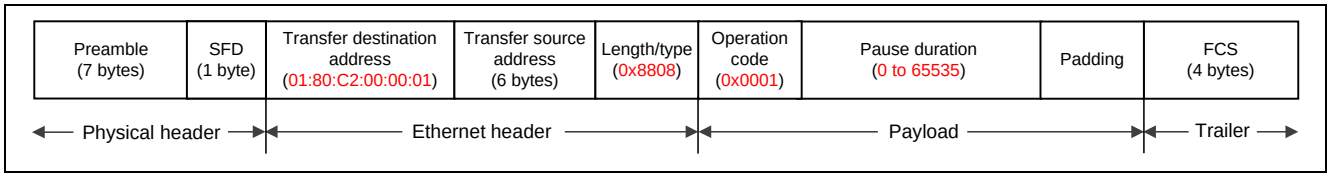


Figure 2.2 PAUSE Frame Format

The transmission destination address is specified as 01:80:C2:00:00:01 (a multicast address reserved for PAUSE frames). At the start of the payload the length/type is specified as 0x8808 and the operation code as 0x0001.

The pause duration in the payload is specified by the value of the automatic PAUSE (AP) bits in the automatic PAUSE frame setting register (APR), or the manual PAUSE time setting (MP) bits in the manual PAUSE frame setting register (MPR).

2.9.3 Magic Packet Frame Format

Table 2.3 shows the Magic Packet frame format.

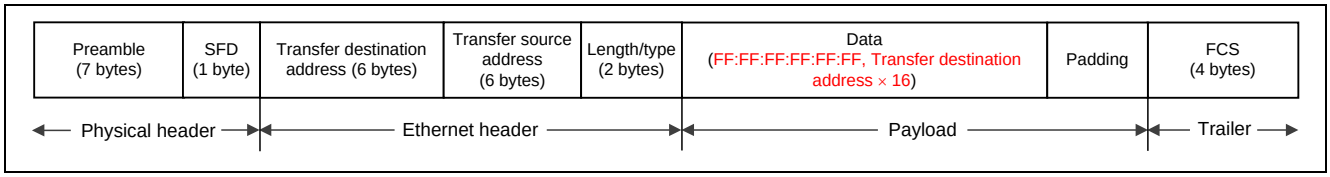


Figure 2.3 Magic Packet Frame Format

In a Magic Packet, the value FF:FF:FF:FF:FF:FF followed by the transmission destination address repeated 16 times is inserted somewhere in the Ethernet frame data.

3. API Functions

3.1 R_ETHER_Initial()

This function makes initial settings to the Ethernet driver.

Format

void R_ETHER_Initial(void);

Parameters

None

Return Values

None

Properties

Prototyped in r_ether_rza2_if.h.

Description

Initializes the memory to be used in order to start Ethernet communication.

Reentrant

Function is not reentrant.

Example

```
#include "r_ether_rza2_if.h"

void callback_sample(void*);
void int_handler_sample(void*);

ether_return         ret;
ether_param_t        param;
ether_cb_t           cb_func;

/* Ethernet channel number
 * ETHER_CHANNEL_0 = Ethernet channel number is 0
 * ETHER_CHANNEL_1 = Ethernet channel number is 1
 */
uint32_t             channel;

/* Initialize memory which ETHERC/EDMAC is used */
R_ETHER_Initial();

channel              = ETHER_CHANNEL_0
param.channel        = channel;

/* Set the callback function */
cb_func.pcb_func     = &callback_sample;
param.ether_callback = cb_func;
ret = R_ETHER_Control(CONTROL_SET_CALLBACK, param);

/* Set the interrupt handler */
cb_func.pcb_int_hnd = &int_handler_sample;
param.ether_int_hnd = cb_func;
ret = R_ETHER_Control(CONTROL_SET_INT_HANDLER, param);

/* Release ETHERC and EDMAC module stop, port settings using ETHERC */
ret = R_ETHER_Control(CONTROL_POWER_ON, param);
if(ETHER_SUCCESS == ret)
{
  /* Initialized successfully completed without ETHERC, EDMAC */
}
```

Special Notes:

This function must be called before calling the R_ETHER_Open_ZC2() function.

3.2 R_ETHER_Open_ZC2()

When using the ETHER API, this function is used first.

Format

```
ether_return_t R_ETHER_Open_ZC2(
      uint32_t       channel      /* ETHERC channel number */
      const uint8_t  mac_addr[]   /* The MAC address of ETHERC */
      uint8_t        pause        /* Specifies whether flow control functionality is enabled or disabled. */
);
```

Parameters

channel

Specifies the ETHERC and EDMAC channel number (0 or 1). This value must be specified as 0 on products with only one ETHERC and EDMAC channel.

mac_addr

Specifies the MAC address of ETHERC.

pause

Specifies the value set in bit 10 (Pause) in register 4 (auto-negotiation advertisement) of the PHY-LSI. The setting ETHER_FLAG_ON is possible only when the user's PHY-LSI supports the pause function. This value is passed to the other PHY-LSI during auto-negotiation. Flow control is enabled if the auto-negotiation result indicates that both the local PHY-LSI and the other PHY-LSI support the pause function.

Specify ETHER_FLAG_ON to convey that the pause function is supported to the other PHY-LSI during auto-negotiation, and specify ETHER_FLAG_OFF if the pause function is not supported or will not be used even though it is supported.

Return Values

ETHER_SUCCESS	*/* Processing completed successfully */*
ETHER_ERR_INVALID_CHAN	*/* Nonexistent channel number */*
ETHER_ERR_INVALID_PTR	*/* Value of the pointer is NULL */*
ETHER_ERR_INVALID_DATA	*/* Value of the argument is out of range */*
ETHER_ERR_OTHER	*/* PHY-LSI initialization failed */*

Properties

Prototyped in r_ether_rza2_if.h.

Description

The R_ETHER_Open_ZC2() function resets the ETHERC, EDMAC and PHY-LSI by software, and starts PHY-LSI auto-negotiation to enable the link signal change interrupt.

The MAC address is used to initialize the ETHERC MAC address register.

Reentrant

Function is reentrant for different channels.

Example

The MAC address used in the sample code is assigned based on the vendor ID of Renesas Electronics Corporation. Customers developing products must use a MAC address obtained by submitting an application to the IEEE.

```
#include "r_ether_rza2_if.h"

ether_return      ret;

/* Source MAC Address */
static uint8_t    mac_addr_src[6] = {0x74,0x90,0x50,0x00,0x79,0x01};

/* Flow control function
 * ETHER_FLAG_ON  = Use flow control function
 * ETHER_FLAG_OFF = No use flow control function
 */
static volatile uint8_t  pause_enable = ETHER_FLAG_OFF;

/* Ethernet channel number
 * ETHER_CHANNEL_0 = Ethernet channel number is 0
 * ETHER_CHANNEL_1 = Ethernet channel number is 1
 */
uint32_t          channel;

channel = ETHER_CHANNEL_0;

/* Initialize ETHERC, EDMAC */
ret = R_ETHER_Open_ZC2(channel, mac_addr_src, pause_enable);
if(ETHER_SUCCESS == ret)
{
  while(1)
  {
    /* Check Link status when Initialized successfully completed */
    R_ETHER_LinkProcess(channel);
  }
}
```

Special Notes:

Either after the R_ETHER_Initial() function is called immediately following a power-on reset, or after the R_ETHER_Close_ZC2() function was called, applications should only use the other API functions after first calling this function and verifying that the return value is ETHER_SUCCESS.

3.3 R_ETHER_Close_ZC2()

The R_ETHER_Close_ZC2() function disables transmit and receive functionality on the ETHERC. This function does not put the ETHERC and EDMAC into the module stop state.

Format

```
ether_return_t R_ETHER_Close_ZC2(
      uint32_t channel          /* ETHERC channel number */
);
```

Parameters

channel

Specifies the ETHERC and EDMAC channel number (0 or 1). This value must be specified as 0 on products with only one ETHERC and EDMAC channel.

Return Values

ETHER_SUCCESS	*/* Processing completed successfully */*
ETHER_ERR_INVALID_CHAN	*/* Nonexistent channel number */*

Properties

Prototyped in r_ether_rza2_if.h.

Description

The R_ETHER_Close_ZC2() function disables transmit and receive functionality on the ETHERC and disables Ethernet interrupts. It does not put the ETHERC and EDMAC into the module stop state.

Execute this function to end the Ethernet communication.

Reentrant

Function is reentrant for different channels.

Example

```
#include "r_ether_rza2_if.h"

ether_return      ret;

/* Ethernet channel number
 * ETHER_CHANNEL_0 = Ethernet channel number is 0
 * ETHER_CHANNEL_1 = Ethernet channel number is 1
 */
uint32_t          channel;

channel = ETHER_CHANNEL_0;

/* Disable transmission and receive function */
ret = R_ETHER_Close_ZC2(channel);
if(ETHER_SUCCESS == ret)
{
  goto end;
}
```

Special Notes:

None

3.4 R_ETHER_Read_ZC2()

The R_ETHER_Read_ZC2() function returns a pointer to the starting address of the buffer storing the receive data.

Format

```
int32_t R_ETHER_Read_ZC2(
      uint32_t       channel      /* ETHERC channel number */
      void        ** pbuf         /* Pointer to buffer that holds the receive data */
);
```

Parameters

channel

Specifies the ETHERC and EDMAC channel number (0 or 1). This value must be specified as 0 on products with only one ETHERC and EDMAC channel.

** *pbuf*

Returns a pointer to the starting address of the buffer storing the receive data.

Return Values

A value of 1 or greater	*/* Returns the number of bytes received. */*
ETHER_NO_DATA	*/* A zero value indicates no data is received. */*
ETHER_ERR_INVALID_CHAN	*/* Nonexistent channel number */*
ETHER_ERR_INVALID_PTR	*/* Value of the pointer is NULL */*
ETHER_ERR_LINK	*/* Auto-negotiation is not completed, and reception is not enabled. */*
ETHER_ERR_MPDE	*/* As a Magic Packet is being detected, transmission and reception is not enabled. */*
ETHER_ERR_MC_FRAME	*/* Multicast frame detected when multicast frame filtering is enabled. */*

Properties

Prototyped in r_ether_rza2_if.h.

Description

The driver's buffer pointer to the starting address of the buffer storing the receive data is returned in the parameter pbuf. Returning the pointer allows the operation to be performed with zero-copy. Return value shows the number of received bytes. If there is no data available at the time of the call, ETHER_NO_DATA is returned. When auto-negotiation is not completed, and reception is not enabled, ETHER_ERR_LINK is returned. ETHER_ERR_MPDE is returned when a Magic Packet is being detected.

The EDMAC hardware operates independent of the R_ETHER_Read_ZC2() function and reads data into a buffer pointed by the EDMAC receive descriptor. The buffer pointed by the EDMAC receive descriptor is statically allocated by the driver.

When multicast frame filtering on the specified channel is enabled by the R_ETHER_Control function, the buffer is released immediately when a multicast frame is detected. Also, the value ETHER_ERR_MC_FRAME is returned.

Frames that generate a receive FIFO overflow, residual-bit frame receive error, long frame receive error, short frame receive error, PHY-LSI receive error, or receive frame CRC error are treated as receive frame errors. When a receive frame error occurs, the descriptor data is discarded, the status is cleared, and reading of data continues.

Reentrant

Function is reentrant for different channels.

Example

```
#include <string.h>
#include "r_ether_rza2_if.h"

ether_return     ret;
uint8_t        * pread_buffer_address;
uint8_t        * pbuf;

/* Ethernet channel number
 * ETHER_CHANNEL_0 = Ethernet channel number is 0
 * ETHER_CHANNEL_1 = Ethernet channel number is 1
 */
uint32_t          channel;

channel = ETHER_CHANNEL_0;

ret = R_ETHER_Read_ZC2(channel, (void **)&pread_buffer_address);
/* When there is data to receive */
if(ETHER_NO_DATA < ret)
{
  memcpy(pbuf, pread_buffer_address, (uint32_t)ret);

  /* Release the receive buffer after reading the receive data. */
  R_ETHER_Read_ZC2_BufRelease(channel);
}
```

Special Notes:

This function is used in combination with the R_ETHER_Read_ZC2_BufRelease function. Always call the R_ETHER_Read_ZC2 function and then the R_ETHER_Read_ZC2_BufRelease function in sequence. If the value ETHER_ERR_LINK is returned when this function is called, initialize the Ethernet driver.

3.5 R_ETHER_Read_ZC2_BufRelease()

The R_ETHER_Read_ZC2_BufRelease() function releases the buffer read by the R_ETHER_Read_ZC2() function.

Format

```
int32_t R_ETHER_Read_ZC2_BufRelease(
      uint32_t channel          /* Specifies the ETHERC channel number. */
);
```

Parameters

channel

Specifies the ETHERC and EDMAC channel number (0 or 1). This value must be specified as 0 on products with only one ETHERC and EDMAC channel.

Return Values

ETHER_SUCCESS	*/* Processing completed successfully */*
ETHER_ERR_INVALID_CHAN	*/* Nonexistent channel number */*
ETHER_ERR_LINK	*/* Auto-negotiation is not completed, and reception is not enabled. */*
ETHER_ERR_MPDE	*/* As a Magic Packet is being detected, transmission and reception is not enabled. */*

Properties

Prototyped in r_ether_rza2_if.h.

Description

The R_ETHER_Read_ZC2_BufRelease() function releases the buffer read by the R_ETHER_Read_ZC2() function.

Reentrant

Function is reentrant for different channels.

Example

```
#include <string.h>
#include "r_ether_rza2_if.h"

ether_return     ret;
uint8_t        * pread_buffer_address;
uint8_t        * pbuf;

/* Ethernet channel number
 * ETHER_CHANNEL_0 = Ethernet channel number is 0
 * ETHER_CHANNEL_1 = Ethernet channel number is 1
 */
uint32_t          channel;

channel = ETHER_CHANNEL_0;

ret = R_ETHER_Read_ZC2(channel, (void **)&pread_buffer_address);
/* When there is data to receive */
if(ETHER_NO_DATA < ret)
{
  memcpy(pbuf, pread_buffer_address, (uint32_t)ret);
  /* Release the receive buffer after reading the receive data. */
  R_ETHER_Read_ZC2_BufRelease(channel);
}
```

Special Notes:

Before calling this function, use the R_ETHER_Read_ZC2 function to read data. Call this function after a value of 1 or greater is returned.

This function is used in combination with the R_ETHER_Read_ZC2_BufRelease function. Always call the R_ETHER_Read_ZC2 function and then the R_ETHER_Read_ZC2_BufRelease function in sequence. If the value ETHER_ERR_LINK is returned when this function is called, initialize the Ethernet driver.

3.6 R_ETHER_Write_ZC2_GetBuf()

The R_ETHER_Write_ZC2_GetBuf() function returns a pointer to the starting address of the transmit data destination.

Format

```
ether_return_t R_ETHER_Write_ZC2_GetBuf(
      uint32_t       channel     /* ETHERC channel number */
      void        ** pbuf        /* Pointer to the starting address of the transmit data destination */
      uint16_t    *  pbuf_size   /* The Maximum size to write to the buffer */
);
```

Parameters

channel

Specifies the ETHERC and EDMAC channel number (0 or 1). This value must be specified as 0 on products with only one ETHERC and EDMAC channel.

** *pbuf*

Returns a pointer to the starting address of the transmit data destination.

* *pbuf_size*

Returns the maximum size to write to the buffer.

Return Values

ETHER_SUCCESS	*/* Processing completed successfully */*
ETHER_ERR_INVALID_CHAN	*/* Nonexistent channel number */*
ETHER_ERR_INVALID_PTR	*/* Value of the pointer is NULL */*
ETHER_ERR_LINK	*/* Auto-negotiation is not completed, and reception is not enabled. */*
ETHER_ERR_MPDE	*/* As a Magic Packet is being detected, transmission and reception is not enabled. */*
ETHER_ERR_TACT	*/* Transmit buffer is not empty. */*

Properties

Prototyped in r_ether_rza2_if.h.

Description

The R_ETHER_Write_ZC2_GetBuf() function returns the parameter pbuf containing a pointer to the starting address of the transmit data destination. The function also returns the maximum size to write to the buffer to the parameter pbuf_size. Returning the pointer allows the operation to be performed with zero-copy.

Return values indicate if the transmit buffer (pbuf) is writable or not. ETHER_SUCCESS is returned when the buffer is writable at the time of the call. When auto-negotiation is not completed, and transmission is not enabled, ETHER_ERR_LINK is returned. ETHER_ERR_MPDE is returned when a Magic Packet is being detected. ETHER_ERR_TACT is returned when the transmit buffer is not empty.

The EDMAC hardware operates independent of the R_ETHER_Write_ZC2_GetBuf() function and writes data stored in a buffer pointed by the EDMAC transmit descriptor. The buffer pointed by the EDMAC transmit descriptor is statically allocated by the driver.

Reentrant

Function is reentrant for different channels.

Example

The MAC address used in the sample code is assigned based on the vendor ID of Renesas Electronics Corporation. Customers developing products must use a MAC address obtained by submitting an application to the IEEE.

```
#include <string.h>
#include "r_ether_rza2_if.h"

ether_return     ret;
uint8_t        * pwrite_buffer_address;
uint8_t        * pbuf;
uint16_t         buf_size;

/* Transmit data */
static uint8_t send_data[60] =
{
    0x74,0x90,0x50,0x00,0x79,0x02,          /* Destination MAC address          */
    0x74,0x90,0x50,0x00,0x79,0x01,          /* Source MAC address               */
    0x00,0x00,                              /* The type field is not used       */
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,    /* Data field     */
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00
};

/* Ethernet channel number
 * ETHER_CHANNEL_0 = Ethernet channel number is 0
 * ETHER_CHANNEL_1 = Ethernet channel number is 1
 */
uint32_t          channel;

channel = ETHER_CHANNEL_0;

ret = R_ETHER_Write_ZC2_GetBuf(channel, (void **)&pwrite_buffer_address,
&buf_size);
/* When transmission buffer is empty */
if(ETHER_SUCCESS == ret)
{
  /* Write the transmit data to the transmission buffer. */
  memcpy(pwrite_buffer_address, send_data, sizeof(send_data));

  R_ETHER_Write_ZC2_SetBuf(channel, sizeof(send_data));

  /* Verifying that the transmission is completed */
  ret = R_ETHER_CheckWrite(channel);
  if(ETHER_SUCCESS == ret)
  {
    /* Transmission is completed */
  }
}
```

Special Notes:

This function is used in combination with the R_ETHER_Write_ZC2_SetBuf function. Always call the R_ETHER_Write_ZC2_GetBuf function and then the R_ETHER_Write_ZC2_SetBuf function in sequence. If the value ETHER_ERR_LINK is returned when this function is called, initialize the Ethernet driver.

3.7 R_ETHER_Write_ZC2_SetBuf()

The R_ETHER_Write_ZC2_SetBuf() function enables the EDMAC to transmit the data in the transmit buffer.

Format

```
ether_return_t R_ETHER_Write_ZC2_SetBuf(
      uint32_t channel          /* ETHERC channel number */
      const uint32_t len        /* The size (60 to 1,514 bytes) which is the Ethernet frame length */
                                /* minus 4 bytes of CRC */
);
```

Parameters

channel

Specifies the ETHERC and EDMAC channel number (0 or 1). This value must be specified as 0 on products with only one ETHERC and EDMAC channel.

len

Specifies the size (60 to 1,514 bytes) which is the Ethernet frame length minus 4 bytes of CRC.

Return Values

ETHER_SUCCESS	*/* Processing completed successfully */*
ETHER_ERR_INVALID_CHAN	*/* Nonexistent channel number */*
ETHER_ERR_INVALID_DATA	*/* Value of the argument is out of range */*
ETHER_ERR_LINK	*/* Auto-negotiation is not completed, and reception is not enabled. */*
ETHER_ERR_MPDE	*/* As a Magic Packet is being detected, transmission and reception is not enabled. */*

Properties

Prototyped in r_ether_rza2_if.h.

Description

Call this function after writing one frame of transmit data is completed.

Set the buffer length to be not less than 60 bytes (64 bytes of the minimum Ethernet frame minus 4 bytes of CRC) and not more than 1,514 bytes (1,518 bytes of the maximum Ethernet frame minus 4 bytes of CRC).

To transmit data less than 60 bytes, make sure to pad the data with zero to be 60 bytes.

Return values indicate that the data written in the transmit buffer is enabled to be transmitted. ETHER_SUCCESS is returned when the data in the transmit buffer is enabled to be transmitted at the time of the call. When auto-negotiation is not completed, and transmission is not enabled, ETHER_ERR_LINK is returned. ETHER_ERR_MPDE is returned when a Magic Packet is being detected.

Reentrant

Function is reentrant for different channels.

Example

The MAC address used in the sample code is assigned based on the vendor ID of Renesas Electronics Corporation. Customers developing products must use a MAC address obtained by submitting an application to the IEEE.

```
#include <string.h>
#include "r_ether_rza2_if.h"

ether_return     ret;
uint8_t        * pwrite_buffer_address;
uint8_t        * pbuf;
uint16_t         buf_size;

/* Transmit data */
static uint8_t send_data[60] =
{
    0x74,0x90,0x50,0x00,0x79,0x02,          /* Destination MAC address              */
    0x74,0x90,0x50,0x00,0x79,0x01,          /* Source MAC address                   */
    0x00,0x00,                              /* The type field is not used           */
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,    /* Data field         */
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00
};

/* Ethernet channel number
 * ETHER_CHANNEL_0 = Ethernet channel number is 0
 * ETHER_CHANNEL_1 = Ethernet channel number is 1
 */
uint32_t          channel;

channel = ETHER_CHANNEL_0;

ret = R_ETHER_Write_ZC2_GetBuf(channel, (void **)&pwrite_buffer_address,
&buf_size);
/* When transmission buffer is empty */
if(ETHER_SUCCESS == ret)
{
  /* Write the transmit data to the transmission buffer. */
  memcpy(pwrite_buffer_address, send_data, sizeof(send_data));

  R_ETHER_Write_ZC2_SetBuf(channel, sizeof(send_data));

  /* Verifying that the transmission is completed */
  ret = R_ETHER_CheckWrite(channel);
  if(ETHER_SUCCESS == ret)
  {
    /* Transmission is completed */
  }
}
```

Special Notes:

- Call this function after writing one frame of transmit data is completed.
- To transmit data less than 60 bytes, make sure to pad the data with zero to be 60 bytes.
- Before calling this function, use the R_ETHER_Write_ZC2_GetBuf function to read data. Call this function after ETHER_SUCCESS is returned.
- This function is used in combination with the R_ETHER_Write_ZC2_GetBuf function. Always call the R_ETHER_Write_ZC2_GetBuf function and then the R_ETHER_Write_ZC2_SetBuf function in sequence. If the value ETHER_ERR_LINK is returned when this function is called, initialize the Ethernet driver.

3.8 R_ETHER_CheckLink_ZC()

The R_ETHER_CheckLink_ZC() function checks the status of the physical Ethernet link using PHY management interface. Ethernet link is up when the cable is connected to a peer device whose PHY is properly initialized.

Format

```
ether_return_t  R_ETHER_CheckLink_ZC(
      uint32_t channel          /* ETHERC channel number */
);
```

Parameters

channel

Specifies the ETHERC and EDMAC channel number (0 or 1). This value must be specified as 0 on products with only one ETHERC and EDMAC channel.

Return Values

ETHER_SUCCESS	*/* Link is up */*
ETHER_ERR_OTHER	*/* Link is down */*
ETHER_ERR_INVALID_CHAN	*/* Nonexistent channel number */*

Properties

Prototyped in r_ether_rza2_if.h.

Description

The R_ETHER_CheckLink_ZC() function checks the status of the physical Ethernet link using PHY management interface. This information (status of Ethernet link) is read from the basic status register (register 1) of the PHY-LSI device. ETHER_SUCCESS is returned when the link is up, and ETHER_ERR_OTHER when the link is down.

Reentrant

Function is reentrant for different channels.

Example

```
#include "r_ether_rza2_if.h"

ether_return      ret;

/* Ethernet channel number
 * ETHER_CHANNEL_0 = Ethernet channel number is 0
 * ETHER_CHANNEL_1 = Ethernet channel number is 1
 */
uint32_t          channel;

channel = ETHER_CHANNEL_0;

ret = R_ETHER_CheckLink_ZC(channel);
if(ETHER_SUCCESS == ret)
{
  /* Link is up */
  LED1 = LED_ON;
}
else
{
  /* Link is down */
  LED1 = LED_OFF;
}
```

Special Notes:

None

3.9 R_ETHER_LinkProcess()

The R_ETHER_LinkProcess() function performs link signal change interrupt processing and Magic Packet detection interrupt processing.

Format

```
void  R_ETHER_LinkProcess(
      uint32_t channel      /* ETHERC channel number */
);
```

Parameters

channel

Specifies the ETHERC and EDMAC channel number (0 or 1). This value must be specified as 0 on products with only one ETHERC and EDMAC channel.

Return Values

None

Properties

Prototyped in r_ether_rza2_if.h.

Description

The R_ETHER_LinkProcess() function performs link signal change interrupt processing and Magic Packet detection interrupt processing. Note that link status change detection processing takes place but link signal change interrupt processing does not occur when ETHER_CFG_USE_LINKSTA is set to a value of 0.

- When a Magic Packet detection interrupt occurs:
  - The callback function registered by the function R_ETHER_Control() reports that a magic packet was detected.
- When a link signal change (link is up) interrupt occurs:
  - The descriptors and the contents of the transmit and receive buffers are erased.
  - After ETHERC and EDMAC are initialized, decide the appropriate configuration to support full-duplex/half-duplex, link speed, and flow control based on the auto-negotiation result, and then enable transmission and reception functionality.
  - EDMAC descriptor is set up to its initial status.
  - The callback function registered by the function R_ETHER_Control() reports that a link signal change (link is up) was detected.
- When a link signal change (link is down) interrupt occurs:
  - After the transmission and reception functions are disabled, the callback function registered by the function R_ETHER_Control() reports that a link signal change (link is down) was detected.
- When ETHER_CFG_USE_LINKSTA is set to a value of 0:
  - The PHY-LSI basic status register (register 1) is read to confirm the Ethernet link status. If a change in the link status is detected, the processing described below occurs.
  - If the link status has changed (link status is link up):
    - The descriptors and the contents of the transmit and receive buffers are erased.
    - After the ETHERC and EDMAC are initialized, the appropriate configuration of full-duplex/half-duplex, link speed, and flow control are determined based on the auto-negotiation result, and transmission and reception functionality are enabled.
    - The EDMAC descriptors are set to their initial status.
    - The callback function registered by the R_ETHER_Control function reports that a link status change (link up) was detected.
  - If the link status has changed (link status is link down):
    - After the transmission and reception functions are disabled, the callback function registered by the R_ETHER_Control function reports that a link status change (link down) was detected.

Reentrant

Function is reentrant for different channels.

Example

```
#include "r_ether_rza2_if.h"

/* Ethernet channel number
 * ETHER_CHANNEL_0 = Ethernet channel number is 0
 * ETHER_CHANNEL_1 = Ethernet channel number is 1
 */
uint32_t          channel;

channel = ETHER_CHANNEL_0;

while(1)
{
  /* Perform link signal change interrupt processing and
   * Magic Packet detection interrupt processing
   */
  R_ETHER_LinkProcess(channel);
}
```

Special Notes:

- If ETHER_CFG_USE_LINKSTA is set to a value of 1, either call this function periodically within the normal processing routine. Note that Ethernet transmission and reception may not operate correctly, and the Ethernet driver may not enter Magic Packet detection mode correctly, if this function is not called.
- If ETHER_CFG_USE_LINKSTA is set to a value of 0, either call this function periodically within the normal processing routine, or call it from an interrupt function that is processed when a periodically occurring interrupt source occurs. Note that Ethernet transmission and reception may not operate correctly, and the Ethernet driver may not enter Magic Packet detection mode correctly, if this function is not called.
- If no callback function was registered with the function R_ETHER_Control(), there will be no notification by a callback function.

3.10 R_ETHER_WakeOnLAN()

The R_ETHER_WakeOnLAN() function switches the ETHERC setting from normal transmission/reception to Magic Packet detection.

Format

```
ether_return_t  R_ETHER_WakeOnLAN(
      uint32_t channel      /* ETHERC channel number */
);
```

Parameters

channel

Specifies the ETHERC and EDMAC channel number (0 or 1). This value must be specified as 0 on products with only one ETHERC and EDMAC channel.

Return Values

ETHER_SUCCESS	*/* Processing completed successfully */*
ETHER_ERR_INVALID_CHAN	*/* Nonexistent channel number */*
ETHER_ERR_LINK	*/* Auto-negotiation is not completed, and reception is not enabled. */*
ETHER_ERR_OTHER	*/* A switch to magic packet detection was performed when the link state */*	*/* was link is down. */*

Properties

Prototyped in r_ether_rza2_if.h.

Description

The R_ETHER_WakeOnLAN() function initializes the ETHERC and EDMAC, and then switches the ETHERC to Magic Packet detection.

Return values indicate whether the ETHERC has been switched to Magic Packet detection or not. When auto-negotiation is not completed, and transmission/reception is not enabled, ETHER_ERR_LINK is returned. ETHER_ERR_OTHER is returned if the link is down after ETHERC is set to Magic Packet detection.

Reentrant

Function is reentrant for different channels.

Example

```
#include "r_ether_rza2_if.h"

ether_return      ret;

/* Ethernet channel number
 * ETHER_CHANNEL_0 = Ethernet channel number is 0
 * ETHER_CHANNEL_1 = Ethernet channel number is 1
 */
uint32_t          channel;

channel = ETHER_CHANNEL_0;

while(1)
{
  /* Perform link signal change interrupt processing and
   * Magic Packet detection interrupt processing
   */
  R_ETHER_LinkProcess(channel);

  /* Enter Magic Packet detection mode. */
  ret = R_ETHER_WakeOnLAN(channel);
  if(ETHER_SUCCESS == ret)
  {
    /*
     * Set the MCU in sleep mode as low power consumption mode when the MCU is
     * awaiting a Magic Packet detection.
    */
    R_LPM_DStandbyTransition( &LPM_SC_TABLE[config_no]);
    wait();
  }
}
```

Special Notes:

None

3.11 R_ETHER_CheckWrite()

The R_ETHER_CheckWrite() function verifies that data transmission has completed.

Format

```
ether_return_t  R_ETHER_CheckWrite(
      uint32_t channel      /* ETHERC channel number */
);
```

Parameters

channel

Specifies the ETHERC and EDMAC channel number (0 or 1). This value must be specified as 0 on products with only one ETHERC and EDMAC channel.

Return Values

ETHER_SUCCESS	*/* Processing completed successfully */*
ETHER_ERR_INVALID_CHAN	*/* Nonexistent channel number */*

Properties

Prototyped in r_ether_rza2_if.h.

Description

The R_ETHER_CheckWrite() function verifies that data was transmitted.

If the transmission completed, ETHER_SUCCESS is returned.

Reentrant

Function is reentrant for different channels.

Example

The MAC address used in the sample code is assigned based on the vendor ID of Renesas Electronics Corporation. Customers developing products must use a MAC address obtained by submitting an application to the IEEE.

```
#include <string.h>
#include "r_ether_rza2_if.h"

ether_return    ret;
uint8_t       * pwrite_buffer_address;
uint8_t       * pbuf;
uint16_t        buf_size;

/* Transmit data */
static uint8_t send_data[60] =
{
    0x74,0x90,0x50,0x00,0x79,0x02,          /* Destination MAC address              */
    0x74,0x90,0x50,0x00,0x79,0x01,          /* Source MAC address                   */
    0x00,0x00,                              /* The type field is not used           */
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,    /* Data field         */
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00
};

/* Ethernet channel number
 * ETHER_CHANNEL_0 = Ethernet channel number is 0
 * ETHER_CHANNEL_1 = Ethernet channel number is 1
 */
uint32_t          channel;

channel = ETHER_CHANNEL_0;

ret = R_ETHER_Write_ZC2_GetBuf(channel, (void **)&pwrite_buffer_address,
&buf_size);
/* When transmission buffer is empty */
if(ETHER_SUCCESS == ret)
{
  /* Write the transmit data to the transmission buffer. */
  memcpy(pwrite_buffer_address, send_data, sizeof(send_data));

  R_ETHER_Write_ZC2_SetBuf(channel, sizeof(send_data));

  /* Verifying that the transmission is completed */
  ret = R_ETHER_CheckWrite(channel);
  if(ETHER_SUCCESS == ret)
  {
    /* Transmission is completed */
  }
}
```

Special Notes:

- This function should be called after transmit data has been written with the R_ETHER_Write_ZC2_Setbuf() function.
- Note that it takes several tens of microseconds for data transmission to actually complete after the R_ETHER_Write_ZC2_Setbuf() function is called. Therefore, when using the R_ETHER_Close_ZC2() function to shut down the Ethernet module following data transmission, call the R_ETHER_CheckWrite() function after calling the R_ETHER_Write_ZC2_Setbuf() function and, after waiting for data transmission to finish, call the R_ETHER_Close_ZC2() function. Calling the R_ETHER_Close_ZC2() function without calling the R_ETHER_CheckWrite() function can cause data transmission to be cut off before it completes.

3.12 R_ETHER_Read()

The R_ETHER_Read() function receives data into the specified receive buffer.

Format

```
int32_t  R_ETHER_Read(
      uint32_t     channel      /* ETHERC channel number */
      void       * pbuf         /* The receive buffer (to store the receive data) */
);
```

Parameters

channel

Specifies the ETHERC and EDMAC channel number (0 or 1). This value must be specified as 0 on products with only one ETHERC and EDMAC channel.

** pbuf*

Specifies the receive buffer (to store the receive data).
The maximum write size is 1,514 bytes. When calling this function, specify the start address of an array with a capacity of 1,514 bytes.

Return Values

A value of 1 or greater	*/* Returns the number of bytes received. */*
ETHER_NO_DATA	*/* A zero value indicates no data is received. */*
ETHER_ERR_INVALID_CHAN	*/* Nonexistent channel number */*
ETHER_ERR_INVALID_PTR	*/* Value of the pointer is NULL */*
ETHER_ERR_LINK	*/* Auto-negotiation is not completed, and reception is not enabled. */*
ETHER_ERR_MPDE	*/* As a Magic Packet is being detected, transmission and reception is not enabled. */*
ETHER_ERR_MC_FRAME	*/* Multicast frame detected when multicast frame filtering is enabled. */*

Properties

Prototyped in r_ether_rza2_if.h.

Description

This function stores the receive data in the specified receive buffer.

Return values indicate the number of bytes received. If there is no data available at the time of the call, ETHER_NO_DATA is returned. When auto-negotiation is not completed, and reception is not enabled, ETHER_ERR_LINK is returned. ETHER_ERR_MPDE is returned when a Magic Packet is being detected.

When multicast frame filtering on the specified channel is enabled by the R_ETHER_Control function, the buffer is released immediately when a multicast frame is detected. Also, the value ETHER_ERR_MC_FRAME is returned.

Frames that generate a receive FIFO overflow, residual-bit frame receive error, long frame receive error, short frame receive error, PHY-LSI receive error, or receive frame CRC error are treated as receive frame errors. When a receive frame error occurs, the descriptor data is discarded, the status is cleared, and reading of data continues.

Reentrant

Function is reentrant for different channels.

Example

```
#include "r_ether_rza2_if.h"
#include "r_ether_rza2_config.h"

ether_return    ret;
uint8_t         read_buffer[ETHER_BUFSIZE];

/* Ethernet channel number
 * ETHER_CHANNEL_0 = Ethernet channel number is 0
 * ETHER_CHANNEL_1 = Ethernet channel number is 1
 */
uint32_t          channel;

channel = ETHER_CHANNEL_0;

ret = R_ETHER_Read(channel, (void *)read_buffer);
if(ETHER_NO_DATA < ret)
{
  /* Reading the receive data is completed */
}
```

Special Notes:

- As this function calls the R_ETHER_Read_ZC2() function and the R_ETHER_Read_ZC2_BufRelease() function internally, data is copied between the buffer pointed by the EDMAC receive descriptor and the receive buffer specified by the R_ETHER_Read() function. (The maximum write size is 1,514 bytes, so set aside a space of 1,514 bytes for the specified receive buffer.)
- Make sure not to use the R_ETHER_Read_ZC2() function and R_ETHER_Read_ZC2_BufRelease() function when using the R_ETHER_Read() function.
- This function uses the standard function memcpy, so string.h is included.
- If the value ETHER_ERR_LINK is returned when this function is called, initialize the Ethernet driver.

3.13 R_ETHER_Write()

The R_ETHER_Write() function transmits the data from the specified transmit buffer.

Format

```
ether_return_t  R_ETHER_Write(
      uint32_t       channel  /* ETHERC channel number */
      void         * pbuf     /* Transmit buffer pointer */
      const uint32_t len      /* The size (60 to 1,514 bytes) which is the Ethernet frame length */
                              /* minus 4 bytes of CRC */
);
```

Parameters

channel

Specifies the ETHERC and EDMAC channel number (0 or 1). This value must be specified as 0 on products with only one ETHERC and EDMAC channel.

** pbuf*

Specifies the transmit data (the destination for the transmit data to be written).

len

Specifies the size (60 to 1,514 bytes) which is the Ethernet frame length minus 4 bytes of CRC.

Return Values

ETHER_SUCCESS	*/* Processing completed successfully */*
ETHER_ERR_INVALID_CHAN	*/* Nonexistent channel number */*
ETHER_ERR_INVALID_DATA	*/* Value of the argument is out of range */*
ETHER_ERR_INVALID_PTR	*/* Value of the pointer is NULL */*
ETHER_ERR_LINK	*/* Auto-negotiation is not completed, and reception is not enabled. */*
ETHER_ERR_MPDE	*/* As a Magic Packet is being detected, transmission and reception is not enabled. */*
ETHER_ERR_TACT	*/* Transmit buffer is not empty. */*

Properties

Prototyped in r_ether_rza2_if.h.

Description

This function transmits data from the specified transmit buffer.

Set the buffer length to be not less than 60 bytes (64 bytes of the minimum Ethernet frame minus 4 bytes of CRC) and not more than 1,514 bytes (1,518 bytes of the maximum Ethernet frame minus 4 bytes of CRC).

To transmit data less than 60 bytes, make sure to pad the data with zero to be 60 bytes.

Return values indicate that the data written in the transmit buffer is enabled to be transmitted. ETHER_SUCCESS is returned when the data in the transmit buffer is enabled to transmit at the time of the call. When auto-negotiation is not completed, and transmission is not enabled, ETHER_ERR_LINK is returned. ETHER_ERR_MPDE is returned when a Magic Packet is being detected. The value ETHER_ERR_TACT is returned if there is no free space in the transmit buffer.

Reentrant

Function is reentrant for different channels.

Example

The MAC address used in the sample code is assigned based on the vendor ID of Renesas Electronics Corporation. Customers developing products must use a MAC address obtained by submitting an application to the IEEE.

```
#include "r_ether_rza2_if.h"

ether_return    ret;

/* Transmit data */
static uint8_t send_data[60] =
{
    0x74,0x90,0x50,0x00,0x79,0x02,          /* Destination MAC address          */
    0x74,0x90,0x50,0x00,0x79,0x01,          /* Source MAC address               */
    0x00,0x00,                              /* The type field is not used       */
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,    /* Data field     */
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00
};

/* Ethernet channel number
 * ETHER_CHANNEL_0 = Ethernet channel number is 0
 * ETHER_CHANNEL_1 = Ethernet channel number is 1
 */
uint32_t          channel;

channel = ETHER_CHANNEL_0;

ret = R_ETHER_Write(channel, (void *)send_data, sizeof(send_data));
if (ETHER_SUCCESS == ret)
{
  /* Transmission is completed */
}
```

Special Notes:

- To transmit data less than 60 bytes, make sure to pad the data with zero to be 60 bytes.
- As this function calls the R_ETHER_Write_ZC2_GetBuf() function and the R_ETHER_Write_ZC2_SetBuf() function internally, data is copied between the buffer pointed by the EDMAC transmit descriptor and the transmit buffer specified by the R_ETHER_Write() function.
- Make sure not to use the R_ETHER_Write_ZC2_GetBuf() function and R_ETHER_Write_ZC2_SetBuf() function when using the R_ETHER_Write() function.
- This function uses the standard functions memset and memcpy, so string.h is included.
- If the value ETHER_ERR_LINK is returned when this function is called, initialize the Ethernet driver.

3.14 R_ETHER_Control()

The R_ETHER_Control() function performs the processing that corresponds to the control code.

Format

```
ether_return_t R_ETHER_Control(
      ether_cmd_t const     cmd       /* Control code */
      ether_param_t const   control   /* Parameters according to the control code */
);
```

Parameters

cmd

Specifies the control code.

control

Specify the parameters according to the control code.

Return Values

ETHER_SUCCESS */* Processing completed successfully */*
ETHER_ERR_INVALID_CHAN */* Nonexistent channel number */*
ETHER_ERR_CHAN_OPEN */* Indicates the Ethernet cannot be opened because it is being used by another application */*
ETHER_ERR_INVALID_ARG */* Invalid argument */*
ETHER_ERR_RECV_ENABLE */* ETHERC receive function enabled */*

Properties

Prototyped in r_ether_rza2_if.h.

Description

Performs the processing that corresponds to the control code. The value ETHER_ERR_INVALID_ARG is returned if the control code is not supported.

The table below lists the corresponding control codes.

Control Code	Description
CONTROL_SET_CALLBACK	Registers a function to be called by callback when a link signal change interrupt occurs or a magic packet is detected. Registers the function specified with the second argument.
CONTROL_SET_PROMISCUOUS_MODE	Set the promiscuous mode bit (PRM) in the ETHERC mode register (ECMR). The second argument specifies the ETHERC channel number of the side on which PRM is to be set and the address of the variable storing the PRM value.
CONTROL_SET_INT_HANDLER	Registers the function that is called when an EINT0 or EINT1 status interrupt occurs. Registers the function specified with the second argument.
CONTROL_POWER_ON	Cancels module stop for the ETHERC and EDMAC. The second argument specifies the ETHERC channel for the cancel module stop.
CONTROL_POWER_OFF	Transitions the ETHERC and EDMAC to the module stop state. The second argument specifies the ETHERC channel for the transition to module stop.
CONTROL_MULTICASTFRAME_FILTER	Enables functionality that reads descriptor information, detects multicast frames, and discards those frames (multicast frame filtering). Specify the setting value for multicast frame filtering functionality with the second argument.
CONTROL_BROADCASTFRAME_FILTER	Specifies the number of broadcast frames that can be received continuously by the ETHERC. When more than the specified number of broadcast frames are received by the ETHERC, the additional broadcast frames are discarded. Specify the ETHERC channel number and the number of broadcast frames that can be received continuously by the ETHERC with the second argument. This function is disabled when the number of broadcast frames is specified as 0.

Reentrant

Function is reentrant.

Example

To register a callback function.)

```
void callback(void*);

ether_return_t    ret;
ether_param_t     param;
ether_cb_t        cb_func;

cb_func.pcb_func     = &callback;
param.ether_callback = cb_func;

ret = R_ETHER_Control(CONTROL_SET_CALBACK, param);
```

To set up promiscuous mode)

```
ether_return         ret;
ether_param_t        param;
ether_promiscuous_t  promiscuous;

promiscuous.channel        = ETHER_CHANNEL_0;
promiscuous.bit            = ETHER_PROMISCUOUS_ON;
param.p_ether_promiscuous  = &promiscuous;

ret = R_ETHER_Control(CONTROL_SET_PROMISCUOUS_MODE, param);
```

Registering an interrupt handler function)

```
void int_handler(void*);

ether_return_t    ret;
ether_param_t     param;
ether_cb_t        cb_func;

cb_func. pcb_int_hnd = &int_handler;
param.ether_callback = cb_func;

ret = R_ETHER_Control(CONTROL_SET_INT_HANDLER, param);
```

Interrupt handler function)

```
static uint32_t   status_ecsr[2];
static uint32_t   status_eesr[2];

void int_handler(void * p_param)
{
  ether_cb_arg_t  *p_arg;

  p_arg = (ether_cb_arg_t *)p_param;

  if (ETHER_CANNEL_MAX > p_arg->channel)
  {
    status_ecsr[p_arg->channel] = p_arg->status_ecsr;
    status_eesr[p_arg->channel] = p_arg->status_eesr;
  }
}
```

Canceling ETHERC/EDMAC module stop)

```
ether_return_t    ret;
ether_param_t     param;

param.channel = channel;
ret = R_ETHER_Control(CONTROL_POWER_ON, param);
```

Transitioning ETHERC/EDMAC to module stop)

```
ether_return_t    ret;
ether_param_t     param;

param.channel = channel;
ret = R_ETHER_Control(CONTROL_POWER_OFF, param);
```

To enable or disable multicast frame filtering)

```
ether_return_t       ret;
ether_param_t        param;
ether_multicast_t    multicast;

multicast.channel       = channel;
multicast.flag          = ETHER_MC_FILTER_ON;
param.p_ether_multicast = &multicast;

ret = R_ETHER_Control(CONTROL_MULTICASTFRAME_FILTER, param);
```

To set the continuous receive count for broadcast frame filtering)

```
ether_return_t       ret;
ether_param_t        param;
ether_broadcast_t    broadcast;

broadcast.channel       = channel;
broadcast.counter       = 10;
param.p_ether_broadcast = &broadcast;

ret = R_ETHER_Control(CONTROL_BROADCASTFRAME_FILTER, param);
```

Special Notes:

Register callback functions and interrupt handlers before calling the R_ETHER_Open_ZC2() function. It may not be possible to detect the first interrupt if the preceding are registered after the R_ETHER_Open_ZC2() function is called.

Specify promiscuous mode after setting the control code to CONTROL_POWER_ON and calling this function. The intended value will not be stored in the ETHERC mode register if the promiscuous mode setting is specified without first setting the control code to CONTROL_POWER_ON and calling this function.

Multicast frame filtering and broadcast frame filtering settings cannot be made while the receive functionality of the ETHERC is enabled. Make these settings before calling the R_ETHER_LinkProcess function. After the R_ETHER_LinkProcess function is called, the receive functionality is enabled when the Ethernet driver enters link up status, so calling this function with CONTROL_MULTICASTFRAME_FILTER or CONTROL_BROADCASTFRAME_FILTER set as the control code causes ETHER_ERR_RECV_ENABLE to be returned, and the settings have no effect.

3.15 R_ETHER_GetVersion()

This function returns the API version.

Format

uint32_t R_ETHER_GetVersion(void);

Parameters

None

Return Values

Version number

Properties

Prototyped in r_ether_rza2_if.h.

Description

Returns the API version number.

Reentrant

Function is reentrant for different channels.

Example

```
#include "r_ether_rza2_if.h"

uint32_t   version;

version = R_ETHER_GetVersion();
```

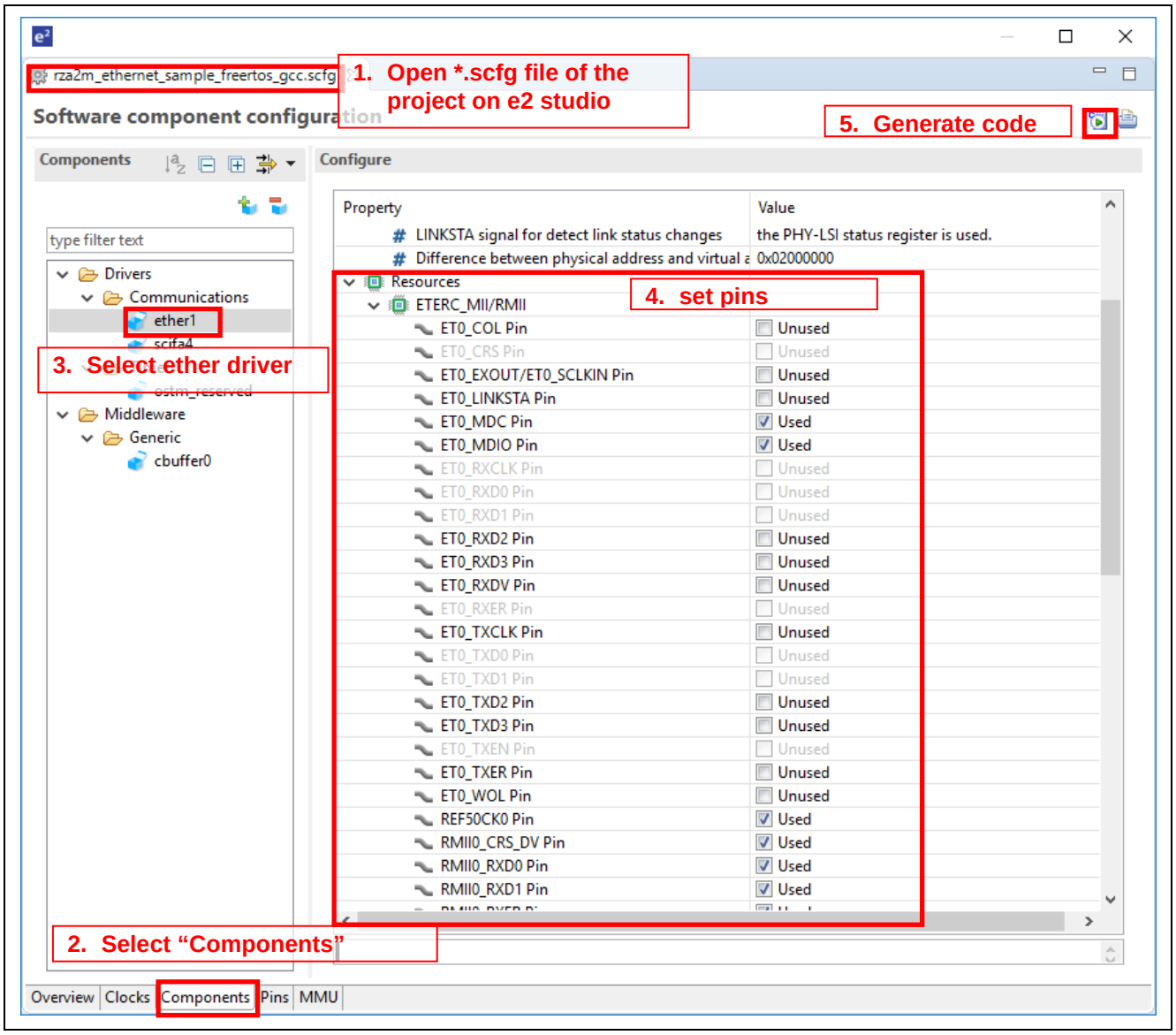
Special Notes:

This function is inline using ‘#pragma inline’.

4. Pin Setting

To use the Ethernet driver, input/output signals of the peripheral function have to be allocated to pins. This pin allocation is referred to as “pin setting” in this document. Please perform the pin setting before calling the R_ETHER_Open_ZC2 function.

When performing the pin setting in the e² studio, the pin setting feature of the Smart Configurator can be used. When using the pin setting feature, a source file is generated according to the option selected in the Pin Setting window in the Smart Configurator. Pins are configured by calling the function defined in the source file.



5. How to use

5.1 Section Allocation

Table 5.1 shows a sample section allocation for the Ethernet driver.

Table 5.1 Program Section Allocation

Memory	Section	Description
Cached RAM	.data	Variables with initial value area
	.bss	Variables without initial value area
Uncached RAM	UNCACHED_BSS	Descriptor area
	_ETHERNET_BUFFERS	Communication buffer area
ROM	.text	Program code area
	.rodata	Constant data area

5.1.1 Notes on Section Allocation

Set MMU_UNCACHED_DIFF (default 0x02000000) macro in r_ether\src\r_ether_rza2.c to the difference between the start addresses of cached RAM (assumed as 0x80000000) and uncached RAM (assumed as 0x82000000).

5.2 Ethernet driver Initial Settings

Figure 5.1 is a flowchart of the routine for making initial settings to the Ethernet driver.

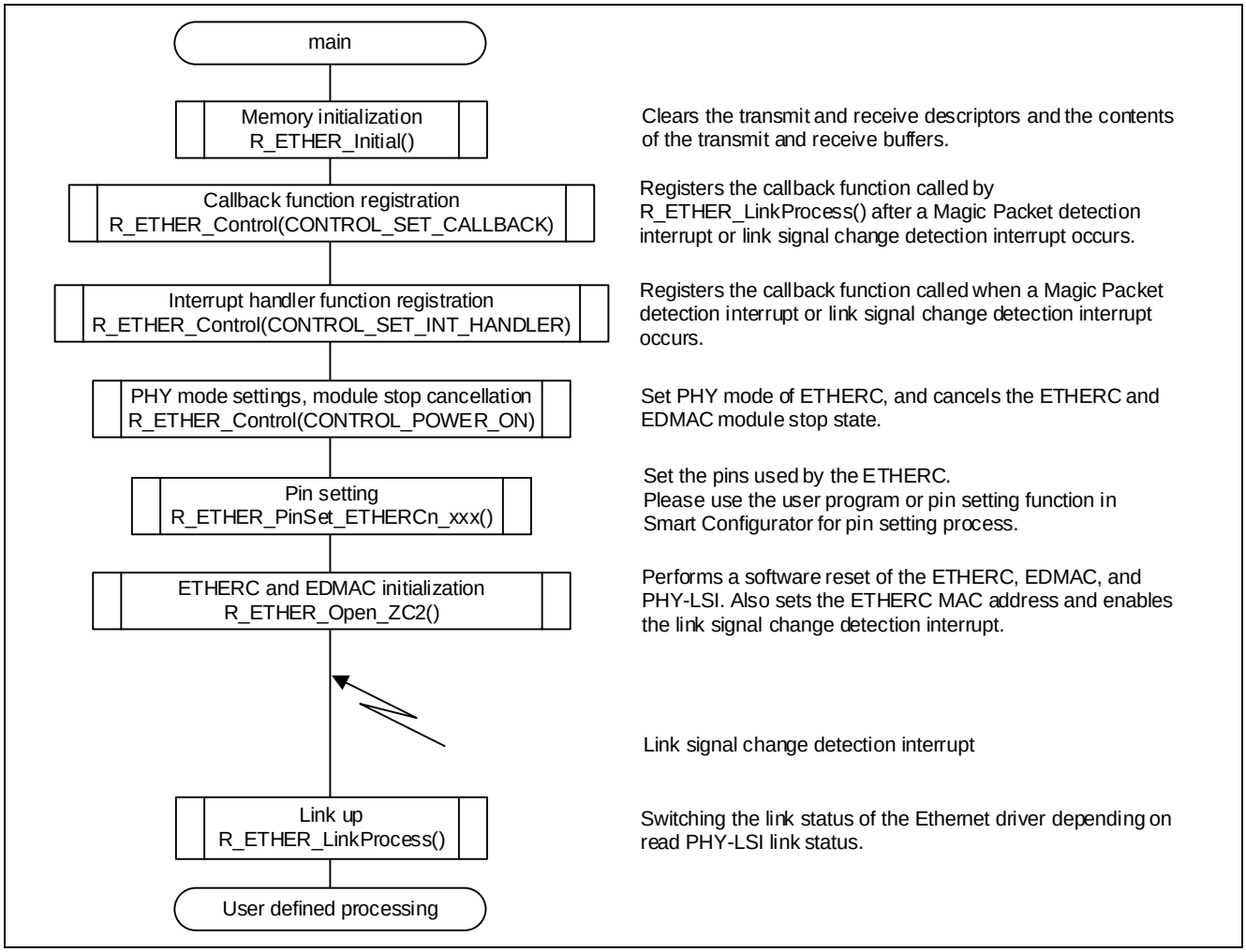


Figure 5.1 Flowchart of Ethernet driver Initial Settings

5.2.1 Notes on Ethernet driver Initial Settings

Calling the R_ETHER_Initial function clears the memory contents for all channels.

5.3 Magic Packet Detection Operation

Figure 5.2 is a flowchart showing the processing whereby the ETHERC and EDMAC are initialized when a Magic Packet is detected, following the transition to Magic Packet detection operation mode.

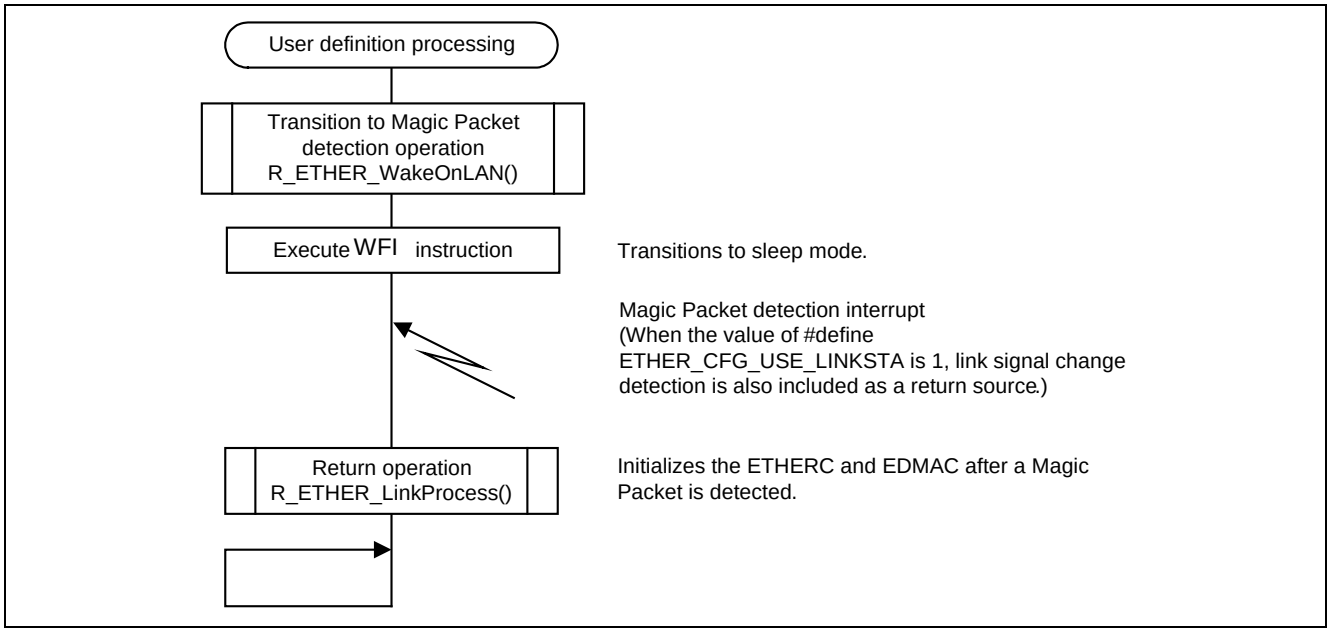


Figure 5.2 Flowchart of Magic Packet Detection Operation

5.3.1 Notes on Magic Packet Detection Operation

- Do not transition the ETHERC or EDMAC to the module stop state after switching to Magic Packet detection operation. Doing so will make it impossible to the CPU to recover from sleep mode following a WFI instruction, because the ETHERC will be unable to detect Magic Packets.
- When a Magic Packet is detected, there will be data from the previously received broadcast frame, etc., in the receive FIFO, and the ETHERC will receive notifications of receive status, etc. Therefore, call the R_ETHER_LinkProcess function to initialize the ETHERC and EDMAC.
- When the value of #define ETHER_CFG_USE_LINKSTA is set to 1, the interrupt handler function is called when a change in the link signal is detected. Therefore, if the CPU was in sleep mode when the link signal change was detected, it will return to normal operation regardless of whether or not a Magic Packet is detected.

6. Operation Confirmation Conditions

The sample code of this application note has been confirmed to operate under the following conditions.

Table 6.1 Peripheral device used(1/2)

Peripheral device	Usage
MCU used	RZ/A2M
Operating frequency[MHz] (Note)	CPU Clock (Iφ) : 528MHz Image processing clock (Gφ) : 264MHz Internal Bus Clock (Bφ) : 132MHz Peripheral Clock 1 (P1φ) : 66MHz Peripheral Clock 0 (P0φ) : 33MHz QSPI0_SPCLK : 66MHz CKIO : 132MHz
Operating voltage	Power supply voltage (I/O): 3.3 V Power supply voltage (either 1.8V or 3.3V I/O (PVcc_SPI)) : 3.3V Power supply voltage (internal): 1.2 V
Integrated development environment	e2 studio V7.4.0
C compiler	"GNU Arm Embedded Tool chain 6-2017-q2-update" compiler options(except directory path) Release: -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Os -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -Wstack-usage=100 -fabi-version=0 Hardware Debug: -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Og -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -g3 -Wstack-usage=100 -fabi-version=0

Note: The operating frequency used in clock mode 1 (Clock input of 24MHz from EXTAL pin)

7. How to Import the Driver

7.1 e² studio

Please refer to the RZ/A2M Smart Configurator User's Guide: e² studio R20AN0583EJ for details on how to import drivers into projects in e2 studio using the Smart Configurator tool.

7.2 For Projects created outside e² studio

This section describes how to import the driver into your project.

Generally, there are two steps in any IDE:

1) Copy the driver to the location in the source tree that you require for your project.

2) Add the link to where you copied your driver to the compiler.

Other required drivers, e.g. r_cbuffer, must be imported similarly.

Table 7.1 Peripheral device used(2/2)

Operation mode	Boot mode 3 (Serial Flash boot 3.3V)
Terminal software communication settings	Communication speed: 115200bps Data length: 8 bits Parity: None Stop bits: 1 bit Flow control: None
Board to be used	RZ/A2M CPU board (RTK7921053C00000BE) RZ/A2M SUB board (RTK79210XXB00000BE)
Device (functionality to be used on the board)	Serial flash memory allocated to SPI multi-I/O bus space (channel 0) Manufacturer : Macronix Inc. Model Name : MX25L51245GXD RL78/G1C (This device communications the host PC by convert USB Communication and Serial Communication.)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Dec 28, 2018	—	First edition issued
1.01	Apr 15, 2019	6	Switched the customize method from editing a header file to use Smart Configurator.
		52	The version of e^2 studio is changed to v7.4.0.
		—	Supported the configuration by Smart Configurator
1.10	May 17, 2019	52	Table 6.1 Peripheral device used(1/2) Remove compiler option "-mthumb-interwork"
		53	Added “7.How to Import the Driver”

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time 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 time 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 time 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 time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance 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 the 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.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is 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. Additionally, 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.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. 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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving 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, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, 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 and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall 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. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, 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.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.