RX Family

Ethernet Module Using Firmware Integration Technology

Introduction

This application note describes an Ethernet module that uses Firmware Integration Technology (FIT). This module 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 FIT module.

Pin setting in the Ethernet FIT module has been removed from Rev.1.11. In order to use the Ethernet FIT module, 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.

- RX64M
- RX71M
- RX65N
- RX72M
- RX72N
- RX66N

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

Target Compilers

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX
- IAR C/C++ Compiler for Renesas RX

For details of the confirmed operation contents of each compiler, refer to “6.2 Confirmed Operation Environment".

Related Documents

- Board Support Package Module Using Firmware Integration Technology (R01AN1685)

Contents

1. Overview

The Ethernet FIT module uses an Ethernet controller (ETHERC) and an Ethernet controller DMA controller (EDMAC) and a PHY management interface (PMGI)[*1] to implement Ethernet frame transmission and reception. The Ethernet FIT module 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.

Note 1. Use PMGI if ETHER_CFG_NON_BLOCKING is set to 1.

1.1 Ethernet FIT Module

The Ethernet FIT module is implemented in a project and used as the API. Refer to 2.11 Adding the FIT Module 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 FIT module.

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_WritePHY()	Write access to the registers in the PHY-LSI using the PHY management interface.
R_ETHER_ReadPHY()	Read access to the registers in the PHY-LSI using the PHY management interface.
R_ETHER_GetVersion()	Returns the Ethernet FIT module version.

1.3 Use Limit

The Ethernet FIT module has the following limitations.

- When using the Ethernet FIT module with RX64M, RX71M, RX72M, RX72N, RX66N, address 00000000h to 0000001Fh can not be used.

2. API Information

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

2.1 Hardware Requirements

This driver requires your MCU supports the following feature:

- ETHERC
- EDMAC
- PMGI[*1]

Note 1. Use PMGI if ETHER_CFG_NON_BLOCKING is set to 1.

2.2 Software Requirements

This driver is dependent upon the following packages:

- Renesas Board Support Package (r_bsp) Rev.5.20 or higher

2.3 Supported Toolchains

The operation of the Ethernet FIT module has been confirmed with the toolchain listed as C compiler in 6.2 Confirmed Operation Environment.

2.4 Usage of Interrupt Vector

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. Table 2.1 shows each interrupt vector that Ethernet FIT module uses.

Table 2.1 List of Usage of Interrupt Vectors

Device	Contents
RX64M RX71M RX72M RX72N	GROUPAL1 interrupt (Vector number. 113) • EINT0 interrupt [channel number 0] (group interrupt source number. 4) • EINT1 interrupt [channel number 1] (group interrupt source number. 5)
RX65N RX66N	GROUPAL1 interrupt (Vector number. 113) • EINT0 interrupt [channel number 0] (group interrupt source number. 4)
RX72M RX72N	• PMGI0 interrupt [channel number 0] (Vector number. 252[*1]) • PMGI1 interrupt [channel number 1] (Vector number. 253[*1])
RX66N	• PMGI0 interrupt [channel number 0] (Vector number. 252[*1])

Note 1. The interrupt vector numbers for software configurable interrupt A show the default values specified in the board support package FIT module (BSP module).

2.5 Header Files

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

2.6 Integer Types

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

2.7 Configuration Overview

The configuration options in the Ethernet FIT module are specified in r_ether_rx_config.h. When using Smart Configurator, the configuration option can be set on software component configuration screen. The setting value is automatically reflected in r_ether_rx_config.h when adding modules to user project. The option names and setting values are listed in the table below.

Configuration options in *r_ether_rx_config.h*	
#define ETHER_CFG_MODE_SEL Note: Default value = 0	Sets the interface between ETHERC and the Ethernet PHY-LSI. If set to 0, MII (Media Independent Interface) is selected. If set to 1, RMII (Reduced Media Independent Interface) is selected.
#define ETHER_CFG_CH0_PHY_ADDRESS Note: Default value = 0*6	Specify the PHY-LSI address used by ETHERC channel 0. Specify a value between 0 and 31.
#define ETHER_CFG_CH1_PHY_ADDRESS Note: Default value = 1*8	Specify the PHY-LSI address used by ETHERC channel 1. Specify a value between 0 and 31.
#define ETHER_CFG_EMAC_RX_DESCRIPTORS Note: Default value = 1	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 = 1	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.
#define ETHER_CFG_AL1_INT_PRIORTY Note: Default value = 2	Sets the priority level of the group AL1 interrupt. This must be set to a value in the range 1 to 15.*4
#define ETHER_CFG_CH0_PHY_ACCESS Note: Default value = 1*1*7	Specify the PHY access channel used by ETHERC channel 0. When 0 is specified, ETHERC0 is used for PHY register access.*2 When 1 is specified, ETHERC1 is used for PHY register access.*3
#define ETHER_CFG_CH1_PHY_ACCESS Note: Default value = 1*1*7	Specify the PHY access channel used by ETHERC channel 1. When 0 is specified, ETHERC0 is used for PHY register access.*2 When 1 is specified, ETHERC1 is used for PHY register access.*3
#define ETHER_CFG_PHY_MII_WAIT Note: Default value = 8	Specify the loop count of software loop used for read or write in PHY-LSI. Set the number of loops according to the PHY-LSI to be used. Specify a value of 1 or greater.
#define ETHER_CFG_PHY_DELAY_RESET 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.

#define ETHER_CFG_LINK_PRESENT Note: Default value = 0	Specify the polarity of the link signal output by the PHY-LSI. When 0 is specified, link-up and link-down correspond respectively to the fall and rise of the LINKSTA signal. When 1 is specified, link-up and link-down correspond respectively to the rise and fall of the LINKSTA signal.
#define ETHER_CFG_USE_LINKSTA Note: Default value = 1	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.*5 When 0 is specified, the PHY-LSI status register is used. When 1 is specified, the LINKSTA signal is used.
#define ETHER_CFG_USE_PHY_KSZ8041NL Note: Default value = 0	Specify whether or not the KSZ8041NL PHY-LSI from Micrel is used. When 0 is specified, the KSZ8041 is not used. When 1 is specified, the KSZ8041 is used.
#define ETHER_CFG_USE_PHY_ICS1894_32 Note: Default value = 0*10	Specify whether or not the ICS1894-32 PHY-LSI from Renesas Electronics is used. When 0 is specified, the ICS1894-32 is not used. When 1 is specified, the ICS1894-32 is used.
#define ETHER_CFG_NON_BLOCKING Note: Default value = 0*11	Specify whether or not to use non blocking for some API functions operation. When 0 is specified, the non-blocking mode is not used. When 1 is specified, the non-blocking mode is used.
#define ETHER_CFG_PMGI_CLOCK Note: Default value = 2500000*9	Specify the clock of the PHY Management Station. Specify a value in the range 97657 to 60000000
#define ETHER_CFG_PMGI_ENABLE_PREAMBLE Note: Default value = 0*9	PHY Management Station Preamble Control. When 0 is specified, include Preamble field When 1 is specified, not include Preamble field
#define ETHER_CFG_PMGI_HOLD_TIME Note: Default value = 0*9	Specify the Hold Time Adjustment of the PHY Management Station. Specify a value in the range 0 to 7
#define ETHER_CFG_PMGI_CAPTURE_TIME Note: Default value = 0*9	Define the Capture Time Adjustment of the PHY Management Station. Specify a value in the range 0 to 7
#define ETHER_CFG_PMGI_INT_PRIORTY Note: Default value = 2*9	Sets the priority level of the group PMGI interrupt. This must be set to a value in the range 1 to 15.

Notes: 1. Refer to Table 2.2 regarding settings for operating the Ethernet FIT module on the Renesas Starter Kit+ for RX64M (product number: R0K50564MSxxxBE). Or refer to Table 2.3 regarding settings for operating the Ethernet FIT module on the Renesas Starter Kit+ for RX71M (product number: R0K50571MSxxxBE)

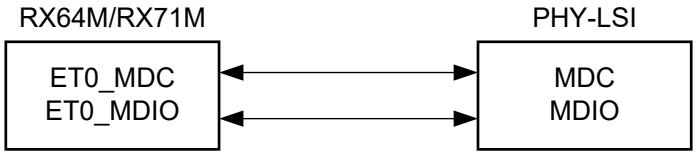
Table 2.2 ETHER_CFG_CH0_PHY_ACCESS/ETHER_CFG_CH1_PHY_ACCESS Settings 1

Short Pin J3	Short Pin J4	ETHER_CFG_CH0_PHY_ACCESS and ETHER_CFG_CH1_PHY_ACCESS Setting Values
1 and 2 shorted	1 and 2 shorted	0
		0
2 and 3 shorted	2 and 3 shorted	1
		1

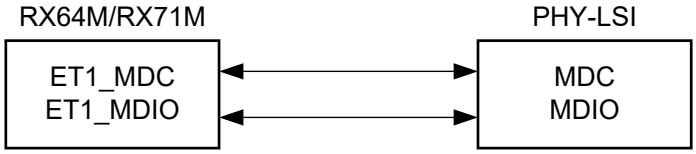
Table 2.3 ETHER_CFG_CH0_PHY_ACCESS/ETHER_CFG_CH1_PHY_ACCESS Settings 2

Short Pin J13	Short Pin J9	ETHER_CFG_CH0_PHY_ACCESS and ETHER_CFG_CH1_PHY_ACCESS Setting Values
1 and 2 shorted	1 and 2 shorted	0
		0
2 and 3 shorted	2 and 3 shorted	1
		1

2. Setting when ETHERC and PHY-LSI are connected as shown below.



3. Setting when ETHERC and PHY-LSI are connected as shown below.



4. This setting is valid only when the target microcontroller is the RX64M, RX71M, RX65N, RX72M, RX72N or RX66N.

5. This setting is valid for all channels when the target microcontroller is the RX64M, RX71M, RX72M or RX72N.

6. The default value is a numeric value based on the initial setting of Renesas Starter Kit+ for RX64M, Renesas Starter Kit+ for RX71M. When using Renesas Starter Kit+ for RX65N (product number. RTK500565NSxxxxxBE) or Renesas Starter Kit+ for RX65N-2MB (product number. RTK50565N2SxxxxxBE), set the value to 30. When using Renesas Starter Kit+ for RX72M (product number. RTK5572Mxxxxxxxxxx), set the value to 1.

7. The default value is a numeric value based on the initial setting of Renesas Starter Kit+ for RX64M, Renesas Starter Kit+ for RX71M or Renesas Starter Kit+ for RX72N. When using Renesas Starter Kit+ for RX65N, Renesas Starter Kit+ for RX65N-2MB or Renesas Starter Kit+ for RX72M set the value to 0.

8. The default value is a numeric value based on the initial setting of Renesas Starter Kit+ for RX64M, Renesas Starter Kit+ for RX71M or Renesas Starter Kit+ for RX72N. When using Renesas Starter Kit+ for RX72M (product number. RTK5572Mxxxxxxxxxx), set the value to 2.

9. These macros are only valid when ETHER_CFG_NON_BLOCKING == 1.

10. The PHY-LSI ICS1894-32 only support full-duplex mode. If you enable this option, prepare a device that support full-duplex mode as the communication partner.

11. This setting can be set to 1 only when the target microcontroller is the RX72M, RX72N, RX66N. Set the value to 0 when the target microcontroller is the RX64M, RX65N, RX71M.

2.8 Code Size

The code size when using the supported toolchain (see section 2.3) is assumed to be that when optimization level 2 and optimization for code size are used. The sizes of ROM (code and constants) and RAM (global data) are set in the configuration header file of the Ethernet FIT module and determined at build time by configuration options.

The values in the table below are confirmed under the following conditions.

Module Revision: r_ether_rx rev1.23

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.04.00

(Compiler option is default setting when using the Smart Configurator.)

GCC for Renesas RX 8.3.0.202104

(Compiler option is default setting when using the Smart Configurator.)

IAR C/C++ Compiler for Renesas RX version 4.20.1

(Compiler option is default setting of the integrated development environment.)

Configuration Options: Default settings

ROM, RAM and Stack Code Sizes

Device	Category	Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		Blocking*3	Non-blocking*4	Blocking	Non-blocking	Blocking	Non-blocking
RX72M*1	ROM	4,703 bytes	6,703 bytes	9,588 bytes	13,664 bytes	6,177 bytes	9,421 bytes
	RAM	6,281bytes	6,349bytes	6, 272bytes	6,392bytes	6,281 bytes	6,333bytes
	STACK*2	148 bytes	148 bytes	-		216 bytes	216 bytes
RX65N*1	ROM	4,577 bytes		9,312 bytes		5,674 bytes	
	RAM	3,146 bytes		3,200 bytes		3,146 bytes	
	STACK*2	148 bytes		-		168 bytes	

Note1: With the following settings: ETHER_CFG_EMAC_RX_DESCRIPTORS = 1, ETHER_CFG_EMAC_TX_DESCRIPTORS = 1, ETHER_CFG_BUFSIZE = 1536

Note2: The sizes of maxmum usage stack of Interrupts functions is included.

Note3: The sizes when ETHER_CFG_NON_BLOCKING = 0.

Note4: The sizes when ETHER_CFG_NON_BLOCKING = 1.

2.9 Arguments

This section documents the enumerations, unions, and structures used as arguments to API functions. These are included in the r_ether_rx_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 */
  CONTROL_RECEIVE_DATA_PADDING,  /* Insert receive data padding */
  CONTROL_SET_PMGI_CALLBACK      /* Set PMGI callback */
} 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_cb_t        pmgi_callback;       /* PMGI callback function pointer */
   ether_recv_padding_t  * padding_param;
                        /* Parameters for inserting received data padding */
} ether_param_t;

typedef struct
{
  void    (*pcb_func)(void *);        /* Callback function pointer */
  void    (*pcb_int_hnd)(void *);     /* Interrupt handler function pointer */
  void    (*pcb_pmgi_hnd) (void *);  /* PGMI callback 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;

typedef struct
{
  uint32_t     channel;              /* ETHERC channel */
  uint8_t      position;             /* Padding insertion position */
  uint8_t      size;                 /* Padding insertion size */
}ether_recv_padding_t;

typedef enum
{
  OPEN_ZC2 = 0,                /* Executing R_ETHER_Open_ZC2 function */
  CHECKLINK_ZC,                /* Executing R_ETHER_CheckLink_ZC function */
  LINKPROCESS,                 /* Executing R_ETHER_LinkProcess function */
  WAKEONLAN,                   /* Executing R_ETHER_WakeOnLAN function */
  LINKPROCESS_OPEN_ZC2,        /* Executing R_ETHER_LinkProcess function */
  LINKPROCESS_CHECKLINK_ZC0,   /* Executing R_ETHER_LinkProcess function */
  LINKPROCESS_CHECKLINK_ZC1,   /* Executing R_ETHER_LinkProcess function */
  LINKPROCESS_CHECKLINK_ZC2,   /* Executing R_ETHER_LinkProcess function */
  WAKEONLAN_CHECKLINK_ZC,      /* Executing R_ETHER_WakeOnLAN function */
  WRITEPHY,                    /* Executing R_ETHER_WritePHY function */
  READPHY,                     /* Executing R_ETHER_ReadPHY function */
  PMGI_MODE_NUM                /* PMGI operation mode number */
}pmgi_mode_t;
```

```
typedef enum
{
  STEP0 = 0,                  /* PMGI operation step 0 */
  STEP1,                      /* PMGI operation step 1 */
  STEP2,                      /* PMGI operation step 2 */
  STEP3,                      /* PMGI operation step 3 */
  STEP4,                      /* PMGI operation step 4 */
  STEP5,                      /* PMGI operation step 5 */
  STEP6,                      /* PMGI operation step 6 */
  PMGI_STEP_NUM               /* PMGI operation step number */
}pmgi_step_t;

typedef enum
{
  PMGI_IDLE = 0,                  /* PMGI is idle */
  PMGI_RUNNING = 1,               /* PMGI is running */
  PMGI_COMPLETE = 2,              /* PMGI is complete */
  PMGI_ERROR = -1                 /* PMGI is error */
}pmgi_event_t;

typedef struct
{
  ether_return_t (* p_func)(uint32_t ether_channel);
                                  /* Type of function pointer array */
} st_pmgi_interrupt_func_t;

typedef struct
{
  bool            locked;         /* The flag of PMGI locked status */
  pmgi_event_t    event;          /* PMGI current operation status */
  pmgi_mode_t     mode;           /* PMGI operation mode */
  pmgi_step_t     step;           /* PMGI operation step */
  uint16_t        read_data;      /* The read value of PMGI register */
  uint32_t        reset_counter;  /* The counter of reading reset register */
  uint32_t        ether_channel;  /* ETHERC channel number */
}pmgi_param_t;

typedef struct
{
  uint32_t         channel;       /* ETHERC channel */
  pmgi_event_t     event;         /* Event code for callback function */
  pmgi_mode_t      mode;          /* PMGI operation mode */
  uint16_t         reg_data;      /* PHY register data for interrupt handler */
} pmgi_cb_arg_t;
```

2.10 Return Values

This section describes return values of API functions. This enumeration is located in r_ether_rx_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 or FIT_NO_PTR */
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_LOCKED,       /* When non-blocking mode is enabled, during PHY */
                        /* access. */
ETHER_ERR_OTHER         /* Other error */
} ether_return_t;
```

2.11 Callback Function

(1) Callback Function Called by API Function R_ETHER_LinkProcess

In the Ethernet FIT module, 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 2.9 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 2.4 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 2.4 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) Callback Function Called by EINT0/EINT1 Status Interrupts

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

- Magic Packet detection operation by the Ethernet FIT module
  - Link signal change detection*1
  - Magic packet detection
- Normal operation by the Ethernet FIT module
  - 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 2.9 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.

(3) Callback Function Called by PMGI interrupt

In the Ethernet FIT module, call the callback function when the non-blocking API function processing is completed.

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_PMGI_CALLBACK as the enumeration (the first argument) described in 2.9 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, the channel number for which API processing has been completed, the variable storing the constants shown in Table 2.5, the variable storing the constants shown in Table 2.6, and the PHY register read data are passed as arguments. When using the argument value outside the callback function, copy it to a variable such as a global variable.

Table 2.5 Argument List of the callback Function

Constant Definition	Description
OPEN_ZC2	Processing of the R_ETHER_Open_ZC2 function is complete.
CHECKLINK_ZC	Processing of the R_ETHER_CheckLink_ZC function is complete.
LINKPROCESS	Processing of the R_ETHER_LinkProcess function is complete.
WAKEONLAN	Processing of the R_ETHER_WakeOnLAN function is complete.
WRITEPHY	Processing of the R_ETHER_WritePHY function is complete.
READPHY	Processing of the R_ETHER_ReadPHY function is complete.

Table 2.6 Argument List of the callback Function

Constant Definition	Description
PMGI_COMPLETE	API function processing completed successfully. In the case of the R_ETHER_CheckLink_ZC function, a link up was detected.
PMGI_ERROR	API function processing ended abnormally. In the case of the R_ETHER_CheckLink_ZC function, a link down was detected.
PMGI_IDLE	API function processing completed successfully without PMGI operation. In the case of the R_ETHER_LinkProcess function, no PMGI operations are performed during function execution.

2.12 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (3) or (5) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the Smart Configurator in e^2 studio
By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: e^2 studio (R20AN0451)” for details.

(2) Adding the FIT module to your project using the FIT Configurator in e^2 studio
By using the FIT Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

(3) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: CS+ (R20AN0470)” for details.

(4) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to “RX Family Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

(5) Adding the FIT module to your project using the Smart Configurator in IAREW
By using the Smart Configurator Standalone version, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: IAREW (R20AN0535)” for details.

2.13 Ethernet Frame Format

The Ethernet FIT module supports the Ethernet II/IEEE 802.3 frame format.

2.13.1 Frame Format for Data Transmission and Reception

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

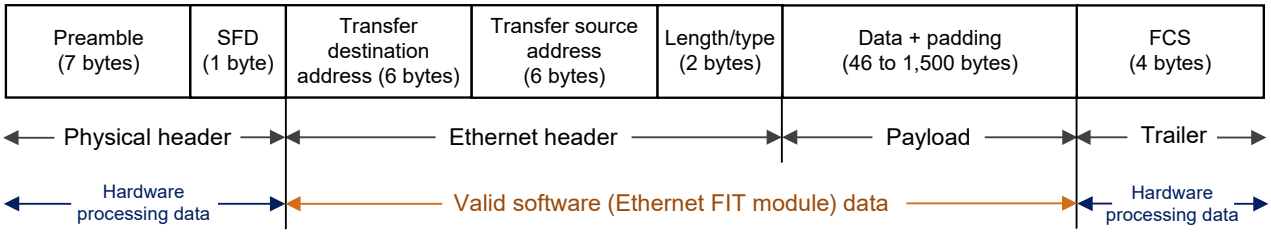


Figure 2.1 Ethernet II/IEEE 802.3 Frame Format

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).

2.13.2 PAUSE Frame Format

Table 2.2 shows the PAUSE frame format.

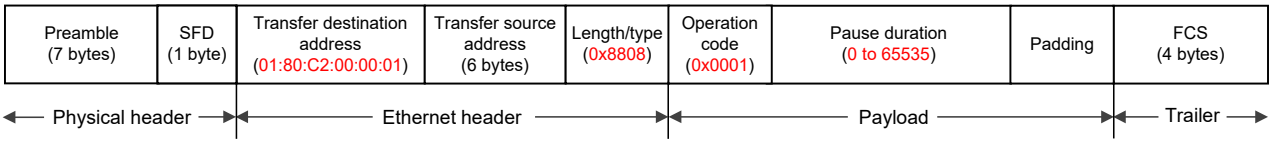


Figure 2.2 PAUSE Frame Format

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).

2.13.3 Magic Packet Frame Format

Table 2.3 shows the Magic Packet frame format.

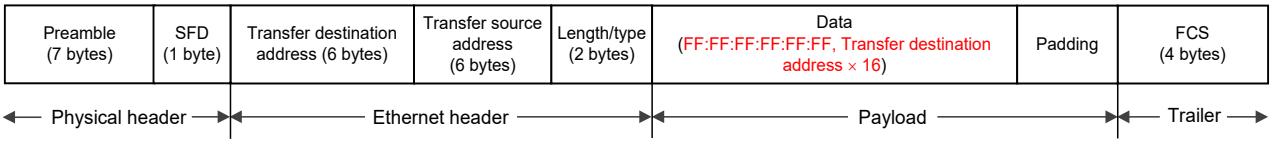


Figure 2.3 Magic Packet Frame Format

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.

2.14 “for”, “while” and “do while” statements

In this module, “for”, “while” and “do while” statements (loop processing) are used in processing to wait for register to be reflected and so on. For these loop processing, comments with “WAIT_LOOP” as a keyword are described. Therefore, if user incorporates fail-safe processing into loop processing, user can search the corresponding processing with “WAIT_LOOP”.

The following shows example of description.

```
while statement example :
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}

for statement example :
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}

do while statement example :
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /*
WAIT_LOOP */
```

3. API Functions

3.1 R_ETHER_Initial()

This function makes initial settings to the Ethernet FIT module.

Format

void R_ETHER_Initial(void);

Parameters

None

Return Values

None

Properties

Prototyped in r_ether_rx_if.h.

Description

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

Example

```
#include "platform.h"
#include "r_ether_rx_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 or the PMGI operation start */*
	/ normally when the non-blocking mode is enable*/*
ETHER_ERR_INVALID_CHAN	*/* Nonexistent channel number */*
ETHER_ERR_INVALID_PTR	*/* Value of the pointer is NULL or FIT_NO_PTR */*
ETHER_ERR_INVALID_DATA	*/* Value of the argument is out of range */*
ETHER_ERR_OTHER	*/* PHY-LSI initialization failed */*
	/ when the non-blocking mode is enabled */*
	/ and PMGI callback function is not registered */*
ETHER_ERR_LOCKED	*/* When PHY access is in progress when non-blocking mode is enabled */*

Properties

Prototyped in r_ether_rx_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.

When non-blocking mode is enabled, the processing result of the function is passed as an argument of the PMGI callback function.

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 "platform.h"
#include "r_ether_rx_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_rx_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.

Example

```
#include "platform.h"
#include "r_ether_rx_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 or FIT_NO_PTR */*
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_rx_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. Note that when hardware-based multicast frame filtering is enabled on the RX64M, RX71M, RX72M, RX72N or RX66N, multicast frames are discarded by the hardware and detection is not possible. For details, see section 6.1 EPTPC Light FIT Module.

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.

Example

```
#include <string.h>
#include "platform.h"
#include "r_ether_rx_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 FIT module.

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_rx_if.h.

Description

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

Example

```
#include <string.h>
#include "platform.h"
#include "r_ether_rx_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 FIT module.

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 or FIT_NO_PTR */*
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_rx_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.

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 "platform.h"
#include "r_ether_rx_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 FIT module.

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_rx_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.

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 "platform.h"
#include "r_ether_rx_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 FIT module.

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	*/* the link status is link up or the operation starts normally when */*
	/ the non-blocking mode is enabled*/*
ETHER_ERR_OTHER	*/* the link status is link-down or the non-blocking mode is */*
	/ enabled and the interrupt handler function is not registered*/*
ETHER_ERR_INVALID_CHAN	*/* Nonexistent channel number */*
ETHER_ERR_LOCKED	*/* When PHY access is in progress when non-blocking mode is */*
	/ enabled */*

Properties

Prototyped in r_ether_rx_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. If non-blocking mode is disabled, ETHER_SUCCESS is returned when the link is up, and ETHER_ERR_OTHER when the link is down.

When non-blocking mode is enabled, the check result is passed as an argument of the interrupt handler function after the link status check is completed.

Example

```
#include "platform.h"
#include "r_ether_rx_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_rx_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 non-blocking mode is enabled, the processing result of the function is passed as an argument of the PMGI callback function.

- 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.

Example

```
#include "platform.h"
#include "r_ether_rx_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 or the operation starts*/*
	/ normally when the non-blocking mode is enable */*
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. Or the non-blocking mode is */*
	/ enabled and the interrupt handler function is not registered */*
ETHER_ERR_LOCKED	*/* When PHY access is in progress when non-blocking mode is */*
	/ enabled */*

Properties

Prototyped in r_ether_rx_if.h.

Description

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

If non-blocking call is disabled, 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.

When non-blocking mode is enabled, the processing result of the function is passed as an argument of the PMGI callback function.

Example

```
#include "platform.h"
#include "r_ether_rx_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)
  {
    R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_LPC_CGC_SWR);
    /*
     * Set the MCU in sleep mode as low power consumption mode when the MCU is
     * awaiting a Magic Packet detection.
    */
    SYSTEM.SBYCR.BIT.SSBY = 0;
    R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_LPC_CGC_SWR);

    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_rx_if.h.

Description

The R_ETHER_CheckWrite() function verifies that data was transmitted.

If the transmission completed, ETHER_SUCCESS is returned.

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 "platform.h"
#include "r_ether_rx_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 a 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 or FIT_NO_PTR */*
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_rx_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. Note that when hardware-based multicast frame filtering is enabled on the RX64M, RX71M, RX72M, RX72N or RX66N, multicast frames are discarded by the hardware and detection is not possible. For details, see section 6.1 EPTPC Light FIT Module.

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.

Example

```
#include "platform.h"
#include "r_ether_rx_if.h"
#include "r_ether_rx_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 FIT module.

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 or FIT_NO_PTR */*
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_rx_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.

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 "platform.h"
#include "r_ether_rx_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 FIT module.

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  contorl  /* 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_rx_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.
CONTROL_RECEIVE_DATA_PADDING	Specifies the parameters of receive data padding insert register (PRADIR). The second argument specifies the ETHERC channel, padding insert position and padding insert size.
CONTROL_SET_PMGI_CALLBACK	If non-blocking call is enabled, register a function to be called back after API function processing is completed. Registers the function specified with the second argument.

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_Contorl(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_Contorl(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_Contorl(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_Contorl(CONTROL_BROADCASTFRAME_FILTER, param);
```

To set the receive data insert padding)

```
ether_return_t          ret;
ether_param_t           param;
ether_recv_padding_t    pad_param;

pad_param.channel       = channel;
pad_param.position      = 0x3f;
pad_param.size          = 0x3;
param.padding_param     = &pad_param;

ret = R_ETHER_Contorl(CONTROL_RECEIVE_DATA_PADDING, param);
```

Registering an PMGI callback function)

```
void int_handler(void*);

ether_return_t    ret;
ether_param_t     param;
ether_cb_t        cb_func;

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

ret = R_ETHER_Contorl(CONTROL_SET_PMGI_CALLBACK, param);
```

PMGI callback function)

```
static pmgi_event_t   pmgi_event;
static pmgi_mode_t   pmgi_mode;
static uint16_t    phy_reg_data;

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

  p_arg = (pmgi_cb_arg_t *)p_param;

  if (ETHER_CANNEL_MAX > p_arg->channel)
  {
    pmgi_event    = p_arg->event;
    pmgi_mode     = p_arg->mode;
    pmgi_reg_data = p_arg->reg_data;
  }
}
```

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 FIT module 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_WritePHY()

The R_ETHER_WritePHY function uses the PHY management interface to write to registers in the PHY-LSI.

Format

```
ether_return_t R_ETHER_WritePHY(
      uint32_t channel,          /* ETHERC channel number */
      uint16_t address,          /* Register address of PHY-LSI to access */
      uint16_t data              /* Data to be written to PHY-LSI registers */
);
```

Parameters

channel

Specify the ETHERC / EDMAC channel number (0, 1). Be sure to specify channel number 0 for products with only 1 channel of ETHERC / EDMAC.

address

Specify the address of the PHY-LSI register to be accessed. For details, check the data sheet of the PHY-LSI to be used.

data

Specify the data to be written to the PHY-LSI register. For details, check the data sheet of the PHY-LSI to be used.

Return Values

ETHER_SUCCESS	*/ * When access is completed normally or when the operation start normally when*/ /* non-blocking mode is enabled * /*
ETHER_ERR_OTHER	*/ * When non-blocking mode is enabled and no interrupt handler function is*/ /* registered * /*
ETHER_ERR_INVALID_CHAN	*/ * For a nonexistent channel * /*
ETHER_ERR_LOCKED	*/ * When non-blocking mode is enabled and PHY is being accessed * /*

Properties

Prototyped in r_ether_rx_if.h.

Description

The R_ETHER_WritePHY function uses the PHY management interface to write access to registers in the PHY-LSI. If non-blocking mode is disabled, ETHER_SUCCESS is returned when write access is successfully completed.

When non-blocking mode is enabled, the callback function is executed after the write access is completed.

Example

```
#include "platform.h"
#include "r_ether_rx_if.h"
ether_return_t    ret;
uint32_t          channel;
uint16_t          address;
uint16_t          data;

channel = ETHER_CHANNEL_0;
address = PHY_REG_CONTROL;
data    = PHY_CONTROL_RESET;
ret = R_ETHER_WritePHY(channel, address, data);
```

Special Notes:

None.

3.16 R_ETHER_ReadPHY()

The R_ETHER_ReadPHY function uses the PHY management interface to access to the registers in the PHY-LSI.

Format

```
ether_return_t R_ETHER_ReadPHY(
      uint32_t channel,     /* ETHERC channel number */
      uint16_t address,     /* Register address of PHY-LSI to access */
      uint16_t *p_data      /* Pointer to the variable that stores the value */
                            /* of the read register*/
);
```

Parameters

channel

Specify the ETHERC / EDMAC channel number (0, 1). Be sure to specify channel number 0 for products with only 1 channel of ETHERC / EDMAC.

address

Specify the address of the PHY-LSI register to be accessed. For details, check the data sheet of the PHY-LSI to be used.

**p_data*

Specify the pointer of the variable to store the register value read from PHY-LSI. For details, check the data sheet of the PHY-LSI to be used.

Return Values

ETHER_SUCCESS	*/ * When access is completed normally or when the operation starts */*
	/ normally when non-blocking mode is enabled * /*
ETHER_ERR_OTHER	*/ * When non-blocking mode is enabled and no interrupt handler function */*
	/ is registered * /*
ETHER_ERR_INVALID_CHAN	*/ * For a nonexistent channel * /*
ETHER_ERR_LOCKED	*/ * When non-blocking mode is enabled and PHY is being accessed * /*

Properties

Prototyped in r_ether_rx_if.h.

Description

The R_ETHER_ReadPHY function uses the PHY management interface to read access to the registers in the PHY-LSI. When non-blocking mode is disabled, the register value read from the PHY-LSI is stored in the argument *p_data*. Also, ETHER_SUCCESS is returned when the read access is successfully completed.

When non-blocking mode is enabled, the read value is transferred as an argument of the callback function.

Example

```
#include "platform.h"
#include "r_ether_rx_if.h"
ether_return_t    ret;
uint32_t          channel;
uint16_t          address;
uint16_t          data;
channel = ETHER_CHANNEL_0;
address = PHY_REG_CONTROL;
ret = R_ETHER_ReadPHY(channel, address, &data);
```

Special Notes:

None.

3.17 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_rx_if.h.

Description

Returns the API version number.

Example

```
#include "platform.h"
#include "r_ether_rx_if.h"

uint32_t   version;

version = R_ETHER_GetVersion();
```

Special Notes:

None.

4. Pin Setting

To use the Ethernet FIT module, input/output signals of the peripheral function have to be allocated to pins with the multi-function pin controller (MPC). 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 e^2 studio, the pin setting feature of the FIT configurator or 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 FIT configurator or the Smart Configurator. Pins are configured by calling the function defined in the source file. Refer to Table 4.1 for details.

Table 4.1 Function Output by the FIT Configurator

MCU Used	Option Selected	Function to be Output	Remarks
RX64M, RX71M, RX65N, RX72M, RX72N, RX66N	Channel 0 MII mode	R_ETHER_PinSet_ETHERC0_MII()	When Channel 0 is used in MII mode.
	Channel 0 RMII mode	R_ETHER_PinSet_ETHERC0_RMII()	When Channel 0 is used in RMII mode.
	Channel 1 MII mode	R_ETHER_PinSet_ETHERC1_MII()	When Channel 1 is used in MII mode.
	Channel 1 RMII mode	R_ETHER_PinSet_ETHERC1_RMII()	When Channel 1 is used in RMII mode.

4.1 Pin setting example for using RSK+RX64M/RSK+RX71M/RSK+RX72M

Table 4.3 and Table 4.4 shows pin setting example using RSK+RX64M or RSK+RX71M, Table 4.5 and shows pin setting example using RSK+RX72M. Note that channel number in need of pin setting are determined by use channel and configuration option specified in Table 4.2. Don’t set the parameters other than Table 4.2. Table 4.3, Table 4.4, Table 4.5 and Table 4.6 show the details of each channel’s Pins.

Table 4.2 Channel number in need of pin setting by use channel and configuration option

Use Channel	Setting of Configuration Option	Channel Number in Need of Pin Setting
Channel 0	ETHER_CFG_CH0_PHY_ACCESS (0) ETHER_CFG_CH1_PHY_ACCESS (0)	Channel 0
	ETHER_CFG_CH0_PHY_ACCESS (1) ETHER_CFG_CH1_PHY_ACCESS (1)	Channel 0 Channel 1
Channel 1	ETHER_CFG_CH0_PHY_ACCESS (0) ETHER_CFG_CH1_PHY_ACCESS (0)	Channel 0 Channel 1
	ETHER_CFG_CH0_PHY_ACCESS (1) ETHER_CFG_CH1_PHY_ACCESS (1)	Channel 1
Channel 0 Channel 1	ETHER_CFG_CH0_PHY_ACCESS (0) ETHER_CFG_CH1_PHY_ACCESS (0)	Channel 0 Channel 1
	ETHER_CFG_CH0_PHY_ACCESS (1) ETHER_CFG_CH1_PHY_ACCESS (1)	Channel 0 Channel 1

Table 4.3 Pin setting example for channel 0 of RSK+RX64M and RSK+RX71M

Case of Using MII Mode	Case of Using RMII Mode	I/O Port
ET0_TX_CLK		PC4
ET0_RX_CLK	REF50CK0	P76
ET0_TX_EN	RMII0_TXD_EN	P80
ET0_ETXD3		PC6
ET0_ETXD2		PC5
ET0_ETXD1	RMII0_TXD1	P82
ET0_ETXD0	RMII0_TXD0	P81
ET0_TX_ER		PC3
ET0_RX_DV		PC2
ET0_ERXD3		PC0
ET0_ERXD2		PC1
ET0_ERXD1	RMII0_RXD1	P74
ET0_ERXD0	RMII0_RXD0	P75
ET0_RX_ER	RMII0_RX_ER	P77
ET0_CRS	RMII0_CRS_DV	P83
ET0_COL		PC7
ET0_MDC		P72
ET0_MDIO		P71
ET0_LINKSTA		P34 *1
ET0_EXOUT		- *2
ET0_WOL		- *2

Notes: 1. Setting is not required if the setting of #define ETHER_CFG_USE_LINKSTA is 0.
Notes: 2. Setting is not required because these pin are not used in Ethernet FIT module.

Table 4.4 Pin setting example for channel 1 of RSK+RX64M and RSK+RX71M

Case of Using MII Mode	Case of Using RMII Mode	I/O Port
ET1_TX_CLK		PG2
ET1_RX_CLK	REF50CK1	PG0
ET1_TX_EN	RMII1_TXD_EN	P60
ET1_ETXD3		PG6
ET1_ETXD2		PG5
ET1_ETXD1	RMII1_TXD1	PG4
ET1_ETXD0	RMII1_TXD0	PG3
ET1_TX_ER		PG7
ET1_RX_DV		P90
ET1_ERXD3		P97
ET1_ERXD2		P96
ET1_ERXD1	RMII1_RXD1	P95
ET1_ERXD0	RMII1_RXD0	P94
ET1_RX_ER	RMII1_RX_ER	PG1
ET1_CRS	RMII1_CRS_DV	P92
ET1_COL		P91
ET1_MDC		P31
ET1_MDIO		P30
ET1_LINKSTA		P93 *1
ET1_EXOUT		- *2
ET1_WOL		- *2

Notes: 1. Setting is not required if the setting of #define ETHER_CFG_USE_LINKSTA is 0.
Notes: 2. Setting is not required because these pin are not used in Ethernet FIT module.

Table 4.5 Pin setting example for channel 0 of RSK+RX72M

Case of Using MII Mode	Case of Using RMII Mode	I/O Port
CLKOUT25M		PH7
ET0_TX_CLK		PM6
ET0_RX_CLK	REF50CK0	PL3
ET0_TX_EN	RMII0_TXD_EN	PL6
ET0_ETXD3		PM5
ET0_ETXD2		PM4
ET0_ETXD1	RMII0_TXD1	PL5
ET0_ETXD0	RMII0_TXD0	PL4
ET0_TX_ER		- *2
ET0_RX_DV		PK2
ET0_ERXD3		PK5
ET0_ERXD2		PK4
ET0_ERXD1	RMII0_RXD1	P74
ET0_ERXD0	RMII0_RXD0	P75
ET0_RX_ER	RMII0_RX_ER	PL2
ET0_CRS	RMII0_CRS_DV	PM7
ET0_COL		PK1
PMGI0_MDC *3		PK0
PMGI0_MDIO *3		PL7
ET0_MDC *4		PK0
ET0_MDIO *4		PL7
ET0_LINKSTA		P34 *1
ET0_EXOUT		- *2
ET0_WOL		- *2

Notes: 1. Setting is not required if the setting of #define ETHER_CFG_USE_LINKSTA is 0.
Notes: 2. Setting is not required because these pin are not used in Ethernet FIT module.
Notes: 3. Setting is not required if the setting of #define ETHER_CFG_NON_BLOCKING is 0.
Notes: 4. Setting is not required if the setting of #define ETHER_CFG_NON_BLOCKING is 1.

Table 4.6 Pin setting example for channel 1 of RSK+RX72M

Case of Using MII Mode	Case of Using RMII Mode	I/O Port
CLKOUT25M		PH7
ET1_TX_CLK		PN2
ET1_RX_CLK	REF50CK1	PQ4
ET1_TX_EN	RMII1_TXD_EN	PQ7
ET1_ETXD3		PN1
ET1_ETXD2		PN0
ET1_ETXD1	RMII1_TXD1	PQ6
ET1_ETXD0	RMII1_TXD0	PQ5
ET1_TX_ER		- *2
ET1_RX_DV		PQ2
ET1_ERXD3		PM3
ET1_ERXD2		PM2
ET1_ERXD1	RMII1_RXD1	PM1
ET1_ERXD0	RMII1_RXD0	PM0
ET1_RX_ER	RMII1_RX_ER	PN3
ET1_CRS	RMII1_CRS_DV	PQ0
ET1_COL		PQ1
PMGI0_MDC *3		PK0
PMGI0_MDIO *3		PL7
ET0_MDC *4		PK0
ET0_MDIO *4		PL7
ET1_LINKSTA		P84 *1
ET1_EXOUT		- *2
ET1_WOL		- *2

Notes: 1. Setting is not required if the setting of #define ETHER_CFG_USE_LINKSTA is 0.
Notes: 2. Setting is not required because these pin are not used in Ethernet FIT module.
Notes: 3. Setting is not required if the setting of #define ETHER_CFG_NON_BLOCKING is 0.
Notes: 4. Setting is not required if the setting of #define ETHER_CFG_NON_BLOCKING is 1.

4.2 Pin setting example for using RSK+RX65N/RSK+RX65N-2M

Table 4.7 shows pin setting example for using RSK+RX65N or RSK+RX65N-2M.

Table 4.7 Pin setting example for using RSK+RX65N or RSK+RX65N-2M

Case of Using MII Mode	Case of Using RMII Mode	I/O Port
ET0_TX_CLK		PC4
ET0_RX_CLK	REF50CK0	P76
ET0_TX_EN	RMII0_TXD_EN	P80
ET0_ETXD3		PC6
ET0_ETXD2		PC5
ET0_ETXD1	RMII0_TXD1	P82
ET0_ETXD0	RMII0_TXD0	P81
ET0_TX_ER		PC3
ET0_RX_DV		PC2
ET0_ERXD3		PC0
ET0_ERXD2		PC1
ET0_ERXD1	RMII0_RXD1	P74
ET0_ERXD0	RMII0_RXD0	P75
ET0_RX_ER	RMII0_RX_ER	P77
ET0_CRS	RMII0_CRS_DV	P83
ET0_COL		PC7
ET0_MDC		P72
ET0_MDIO		P71
ET0_LINKSTA *1		P54 (RSK+RX65N) *1 P34 (RSK+RX65N-2M) *1
ET0_EXOUT		- *2
ET0_WOL		- *2

Notes: 1. Setting is not required if the setting of #define ETHER_CFG_USE_LINKSTA is 0.

Notes: 2. Setting is not required because these pin are not used in Ethernet FIT module.

4.3 Pin setting example for using RSK+RX72N

Table 4.8 shows pin setting example for using RSK+RX72N.

Table 4.8 Pin setting example for using RSK+RX72N

Case of Using MII Mode	Case of Using RMII Mode	I/O Port
CLKOUT25M		PH7
ET1_TX_CLK		PN2
ET1_RX_CLK	REF50CK1	PQ4
ET1_TX_EN	RMII1_TXD_EN	PQ7
ET1_ETXD3		PN1
ET1_ETXD2		PN0
ET1_ETXD1	RMII1_TXD1	PQ6
ET1_ETXD0	RMII1_TXD0	PQ5
ET1_TX_ER		- *2
ET1_RX_DV		P90
ET1_ERXD3		P97
ET1_ERXD2		P96
ET1_ERXD1	RMII1_RXD1	P95
ET1_ERXD0	RMII1_RXD0	P94
ET1_RX_ER	RMII1_RX_ER	PN3
ET1_CRS	RMII1_CRS_DV	PQ0
ET1_COL		P91
PMGI1_MDC *3		P31
PMGI1_MDIO *3		P30
ET1_MDC *4		P31
ET1_MDIO *4		P30
ET1_LINKSTA *1		P93*1
ET1_EXOUT		- *2
ET1_WOL		- *2

Notes: 1. Setting is not required if the setting of #define ETHER_CFG_USE_LINKSTA is 0.
Notes: 2. Setting is not required because these pin are not used in Ethernet FIT module.
Notes: 3. Setting is not required if the setting of #define ETHER_CFG_NON_BLOCKING is 0.
Notes: 4. Setting is not required if the setting of #define ETHER_CFG_NON_BLOCKING is 1.

5. How to use

5.1 Section Allocation

Table 5.1 shows a sample section allocation for the Ethernet FIT module.

Table 5.1 Program Section Allocation

Address	Device	Section	Description
0x00000020	Internal RAM	SI	Interrupt stack area
		SU	User stack area
		B_1	Uninitialized data area of 1byte boundary
		R_1	Initialized data area of 1byte boundary (variable)
		B_2	Uninitialized data area of 2byte boundary
		R_2	Initialized data area of 2byte boundary (variable)
		B	Uninitialized data area of 4byte boundary
		R	Initialized data area of 4byte boundary (variable)
0x00010000		B_ETHERNET_BUFFERS*	Transmit buffer and receive buffer area
		B_RX_DESC*	Receive descriptor area
		B_TX_DESC*	Transmit descriptor area
0xFFFF8000	Internal ROM	C_1	Constant area of 1byte boundary
		C_2	Constant area of 2byte boundary
		C	Constant area of 4byte boundary
		C$*	Constant region (CDEC, CBSEC, C$VECT) of C$* section
		D*	Initialization data area
		P*	Program area
		W*	Branch table area for switch statements
		L	String literal area
0xFFFFFF80		EXCEPTVECT	Interrupt vector area
0xFFFFFFFC		RESETVECT	Reset vector area

5.1.1 GCC for Renesas RX section setting example

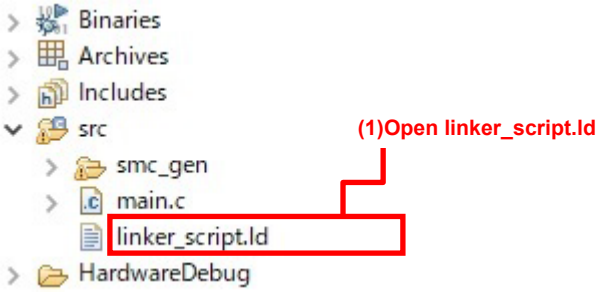
Edit the linker_script.ld file and add sections and symbols.

Add Sections and Symbols

Add the following code.

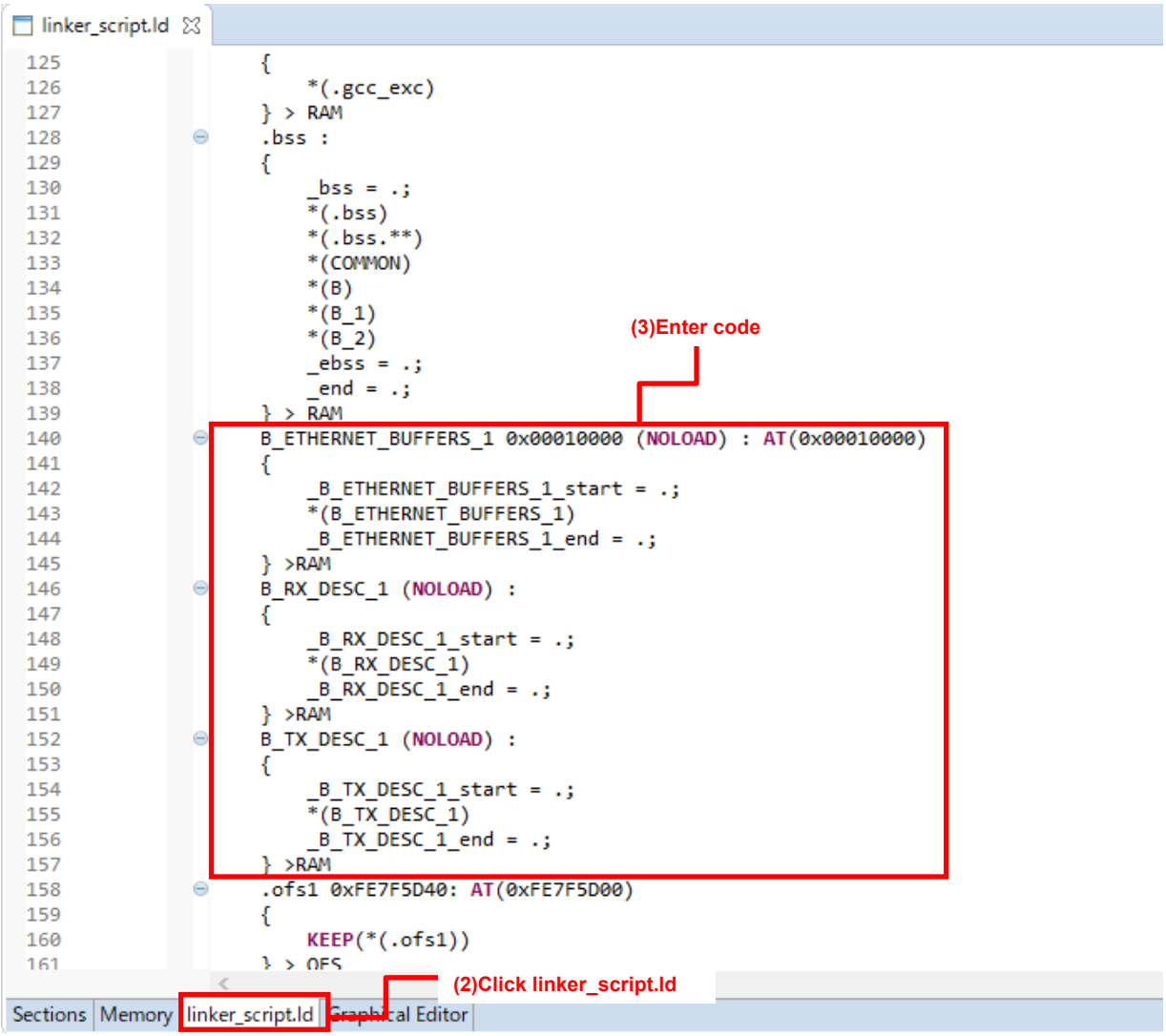
```
B_ETHERNET_BUFFERS_1 0x00010000 (NOLOAD) : AT(0x00010000)
{
      _B_ETHERNET_BUFFERS_1_start = .;
      *(B_ETHERNET_BUFFERS_1)
      _B_ETHERNET_BUFFERS_1_end = .;
} >RAM
B_RX_DESC_1 (NOLOAD) :
{
      _B_RX_DESC_1_start = .;
      *(B_RX_DESC_1)
      _B_RX_DESC_1_end = .;
} >RAM
B_TX_DESC_1 (NOLOAD) :
{
      _B_TX_DESC_1_start = .;
      *(B_TX_DESC_1)
      _B_TX_DESC_1_end = .;
} >RAM
```

(1) Open "linker_script.ld" from Project Explorer.



(2) Click linker_script.ld.

(3) Enter code.



5.1.2 IAR C/C++ Compiler for Renesas RX section setting example

Edit the icf file and add section settings.

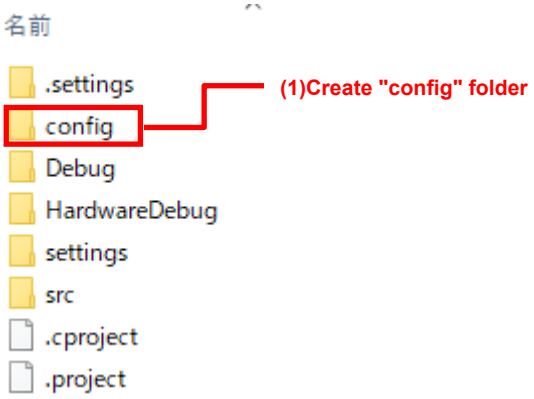
The icf file to be edited depends on the target device of the project, so please confirm and edit the upper 8 digits of the model name of the device to be used.

As an example, edit "lnkr5f565ne.icf" with RX65N (R5F565NEDDFC).

The following is an example of editing on the RX65N (R5F565NEDDFC).

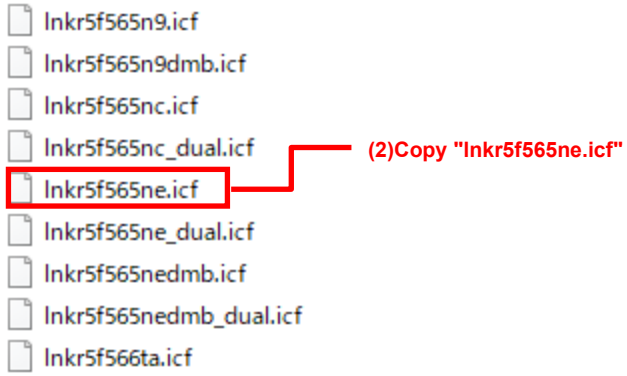
Add section settings.

(1) Create "config" folder in project folder.



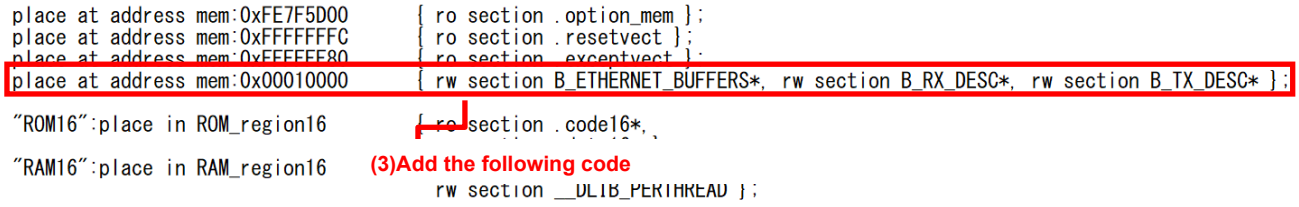
(2) Copy "lnkr5f565ne.icf" from "\rx\config" where IAR C/C++ Compiler for Renesas RX ("EWRX") is installed to the "config" folder in the project folder.

The installation default is "C: \Program Files (x86)\IAR Systems\Embedded Workbench 8.1".

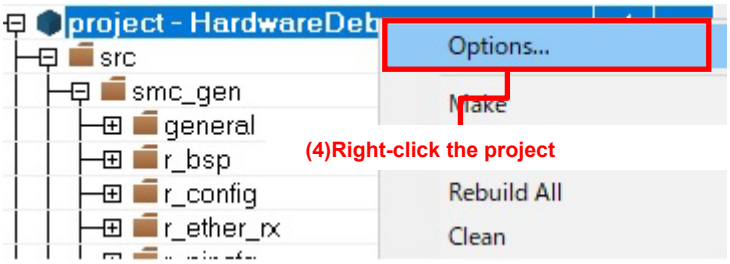


(3) Open the copied "lnkr5f565ne.icf" file and add the following code.

```
place at address mem:0x00010000      { rw section B_ETHERNET_BUFFERS*, rw
section B_RX_DESC*, rw section B_TX_DESC* };
```



(4) Open the project from EWRX, right-click the project in the workspace and open options.

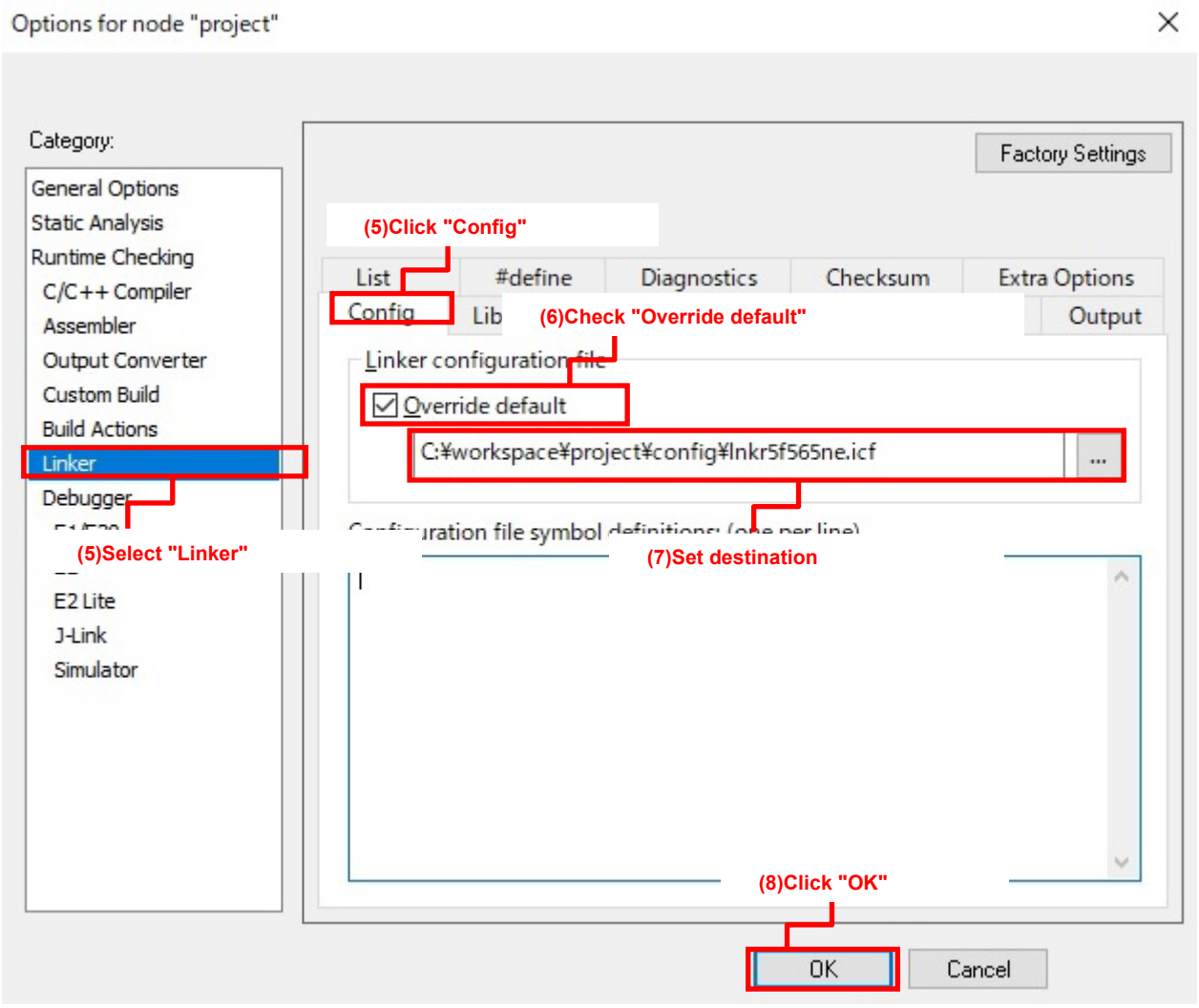


(5) Select "Category: Linker" and click "Config".

(6) Check "Override default".

(7) Set the file edited in step (3) as a reference destination.

(8) Click "OK".



5.1.3 Notes on Section Allocation

- Since the EDMAC mode register (EDMR) transmit/receive descriptor length bits (DL) are set to specify 16 bytes, sections must be allocated on 16-byte boundaries.
- Transmit buffer and receive buffer areas must be allocated on 32-byte boundaries.
- If Ethernet FIT module is installed in the user project by FIT configurator of e2 studio, section allocation is will be set automatically. Please change the setting according the user program.
- When using the Ethernet FIT module with RX64M, RX71M, RX72M, RX72N and RX66N, do not use addresses 0000 0000h to 0000 001Fh.

5.2 Ethernet FIT Module Initial Settings

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

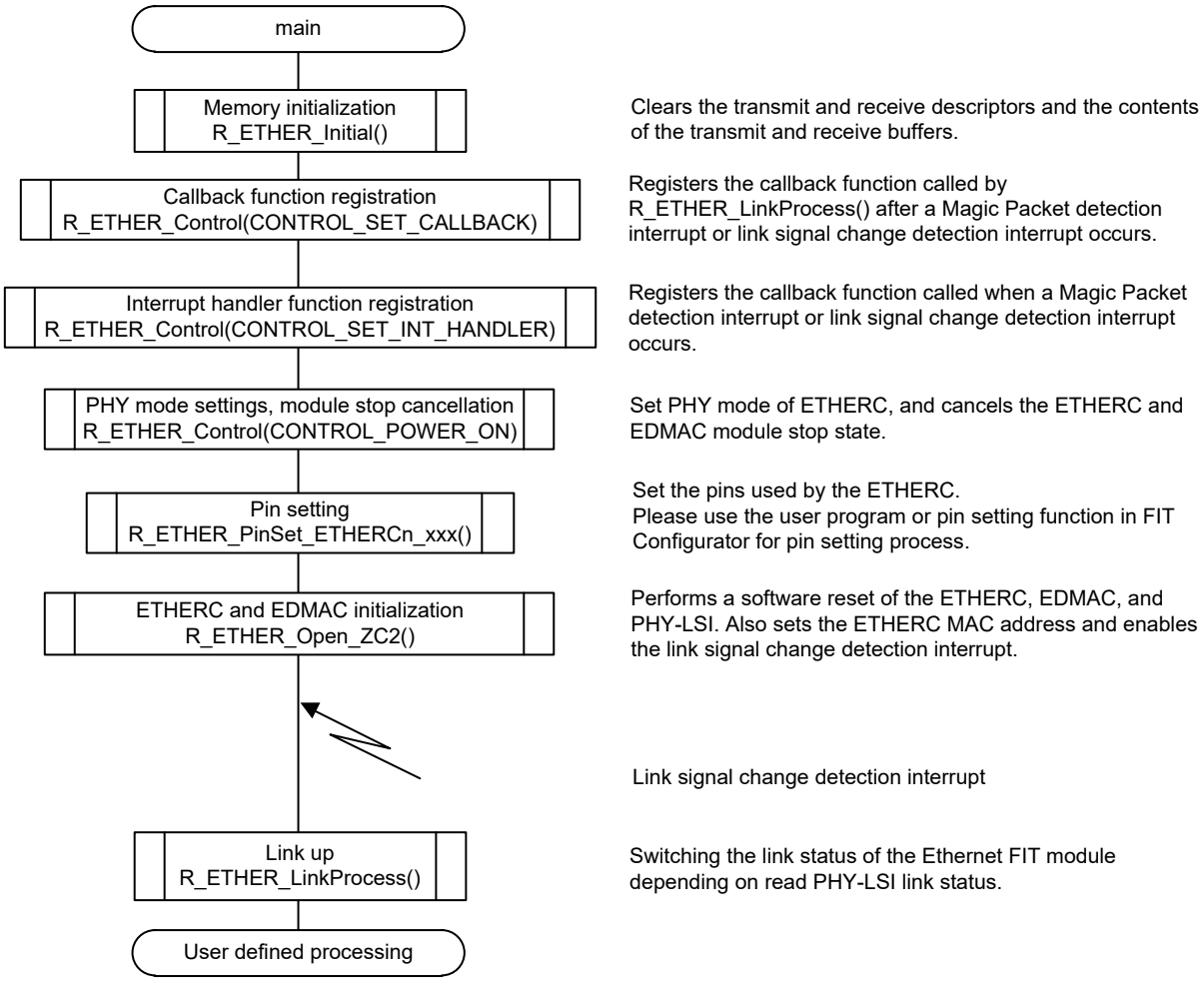


Figure 5.1 Flowchart of Ethernet FIT Module Initial Settings

5.2.1 Notes on Ethernet FIT Module 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.

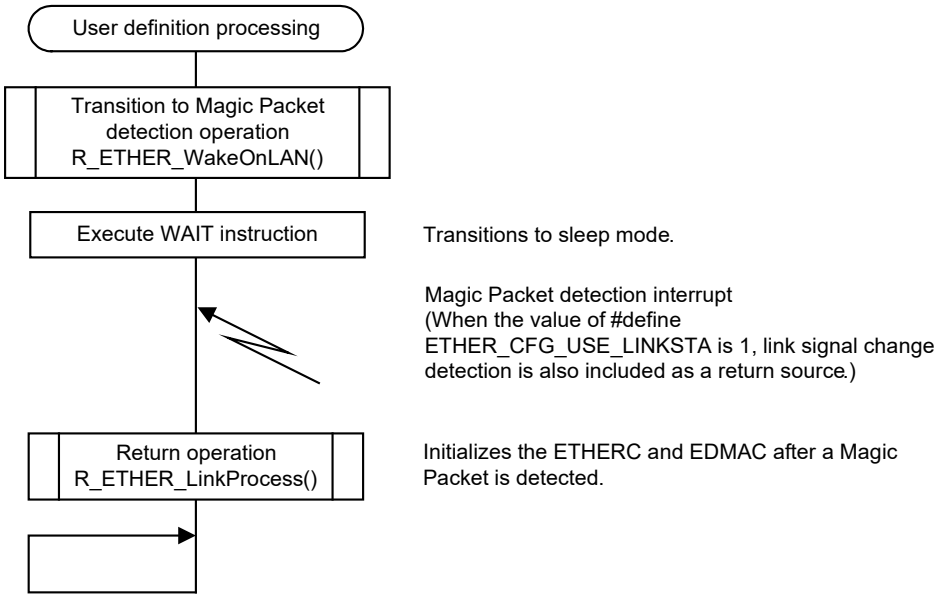


Figure 5.2 Flowchart of Magic Packet Detection Operation

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 WAIT 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.

5.4 Notes on Accessing MII/RMII Registers

When ETHER_CFG_NON_BLOCKING is set to 0, the MII/RMII register in PHY-LSI is accessed using PIR register. Serial data according to the MII/RMII management frame format is transmitted and received by controlling the ETn_MDC and ETn_MDIO pins with software.
Figure **5.3** shows the MII/RMII register access timing when accessing the MII/RMII registers in the PHY-LSI under the conditions shown in **Table 5.2**.

Table 5.2 Conditions for Accessing MII/RMII Registers in PHY-LSI

Item	Value
Microcomputer used	R5F565N9ADFB
C compiler	CC-RX V3.02
ICLK frequency	120MHz
PCLKA frequency	120MHz
Ether FIT Version	1.21
ETHER_CFG_PHY_MII_WAIT setting value	4
PHY implemented in RSK	DP83620

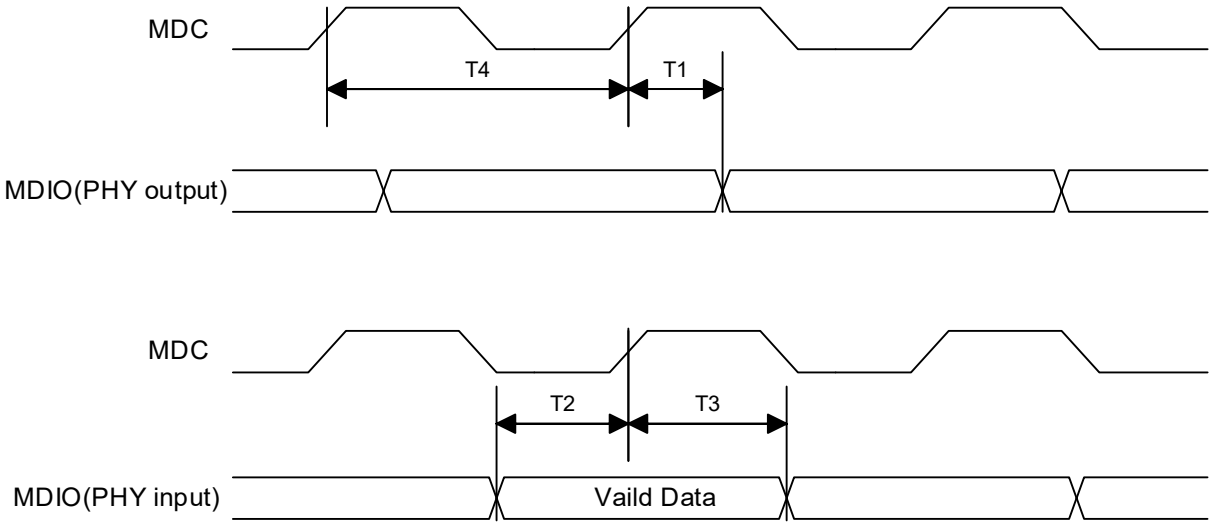


Figure 5.3 MII/RMII Register Access Timing

Table 5.3 shows the AC specifications of the input/output timing of MDC/MDIO in the PHY mounted on RSK and the measured values (reference) when the MII/RMII register is accessed under the conditions of Table 5.2.

Table 5.3 AC specifications and measurement values(reference) of MDC/MDIO input/output timing in PHY implemented in RSK

Item	Parameter	Min	Max	Measured Value (ref.)	Unit
MDC to MDIO (Output) Delay Time	T1	0	20	8	ns
MDIO (Input) to MDC Setup Time	T2	10	-	500	ns
MDIO (Input) to MDC Hold Time	T3	10	-	2300	ns
MDC Frenquency	T4	40	-	2840	ns

When ETHER_CFG_NON_BLOCKING is set to the value 1, PMGI is used to access the MII/RMII register in the PHY-LSI. Serial data according to the MII/RMII management frame format is sent and received from the PMGIn_MDC and PMGIn_MDIO pins. Figure 5.4 shows the MII/RMII register access timing when accessing the MII/RMII register in the PHY-LSI under the conditions shown in Table 5.4.

Table 5.4 Conditions for Accessing MII/RMII Registers in PHY-LSI

Item	Value
Microcomputer used	R5F572NNDDBD
C compiler	CC-RX V3.02
ICLK frequency	240MHz
PCLKA frequency	120MHz
ETHER_CFG_PMGI_CLOCK	2500000
ETHER_CFG_PMGI_HOLD_TIME	7
ETHER_CFG_PMGI_CAPTURE_TIME	0
Ether FIT Version	1.21
PHY implemented in RSK	KSZ8041NL

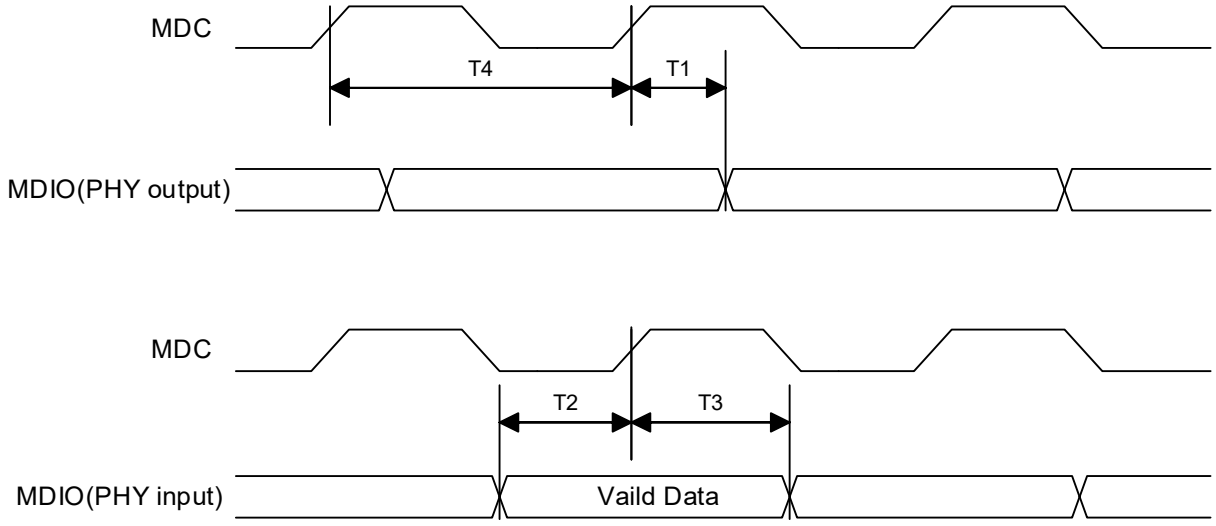


Figure 5.4 MII/RMII Register Access Timing

Table 5.5 shows the AC specifications of the input/output timing of MDC/MDIO in the PHY mounted on RSK and the measured values (reference) when the MII/RMII register is accessed under the conditions of Table 5.4.

Table 5.5 AC specifications and measurement values(reference) of MDC/MDIO input/output timing in PHY implemented in RSK

Item	Parameter	Min	Max	Measured Value (ref.)	Unit
MDC to MDIO (Output) Delay Time	T1	-	-	64	ns
MDIO (Input) to MDC Setup Time	T2	10	-	332	ns
MDIO (Input) to MDC Hold Time	T3	4	-	60	ns
MDC Frenquency	T4	-	-	399	ns

If you cannot meet the AC specifications of the PHY to be used under the set conditions, change the configuration option settings shown in Section 2.7 so that the MII/RMII register access timing can meet the AC specifications of the PHY.

5.5 How to Use API Function Called in Non-Blocking

Setting ETHER_CFG_NON_BLOCKING to a value of 1 makes R_ETHER_Open_ZC2, R_ETHER_CheckLink_ZC, R_ETHER_LinkProcess, R_ETHER_WakeOnLAN, R_ETHER_WritePHY, and R_ETHER_ReadPHY function calls non-blocking-call. The callback function is called when the processing of the API function called by the non-blocking-call is completed. Figures 5.5 to 5.6 show flowcharts of usage examples of API functions called by non-blocking-call.

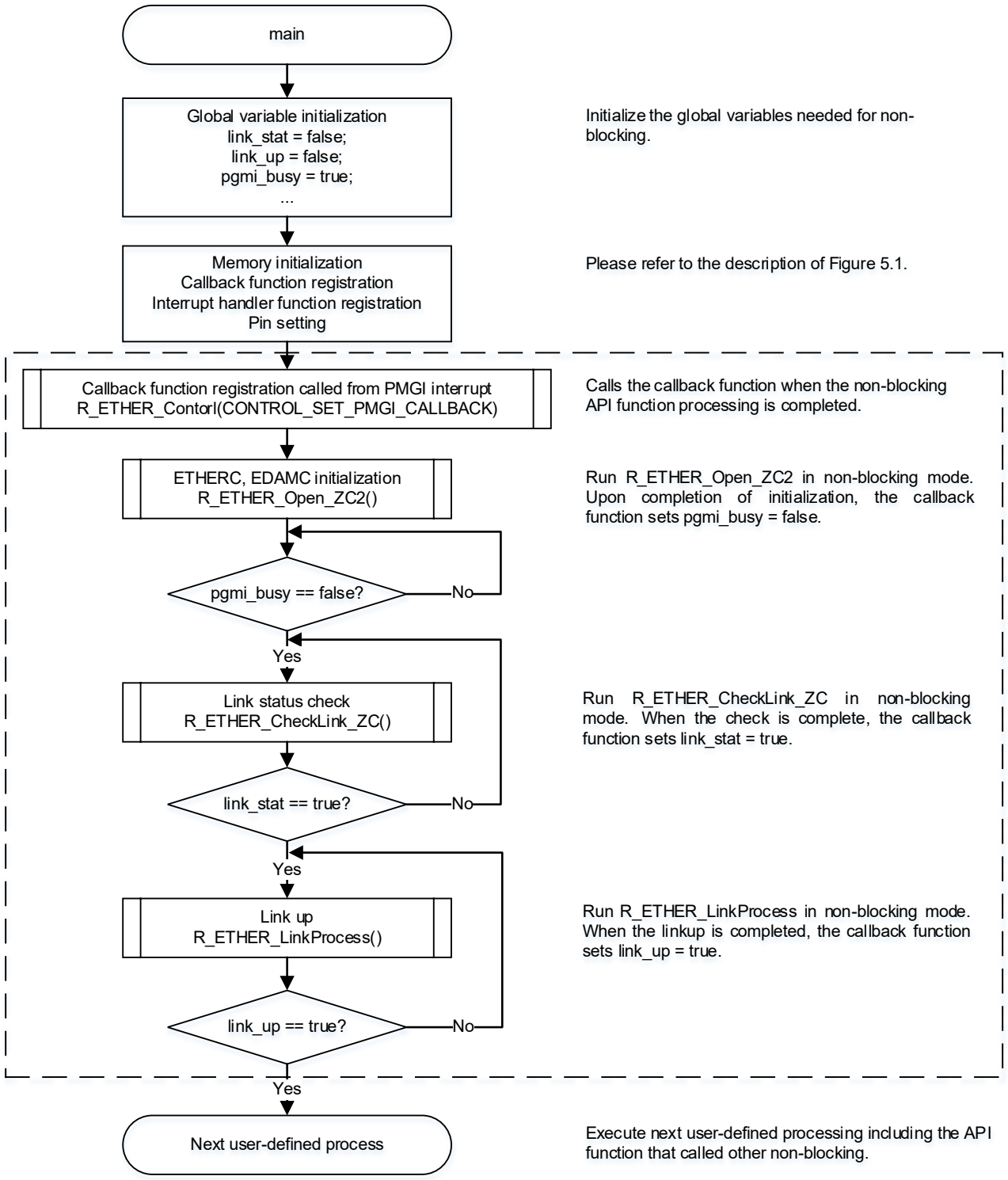


Figure 5.5 Usage Example of API Function Called by Non-Blocking-Call (1) - Main Routine

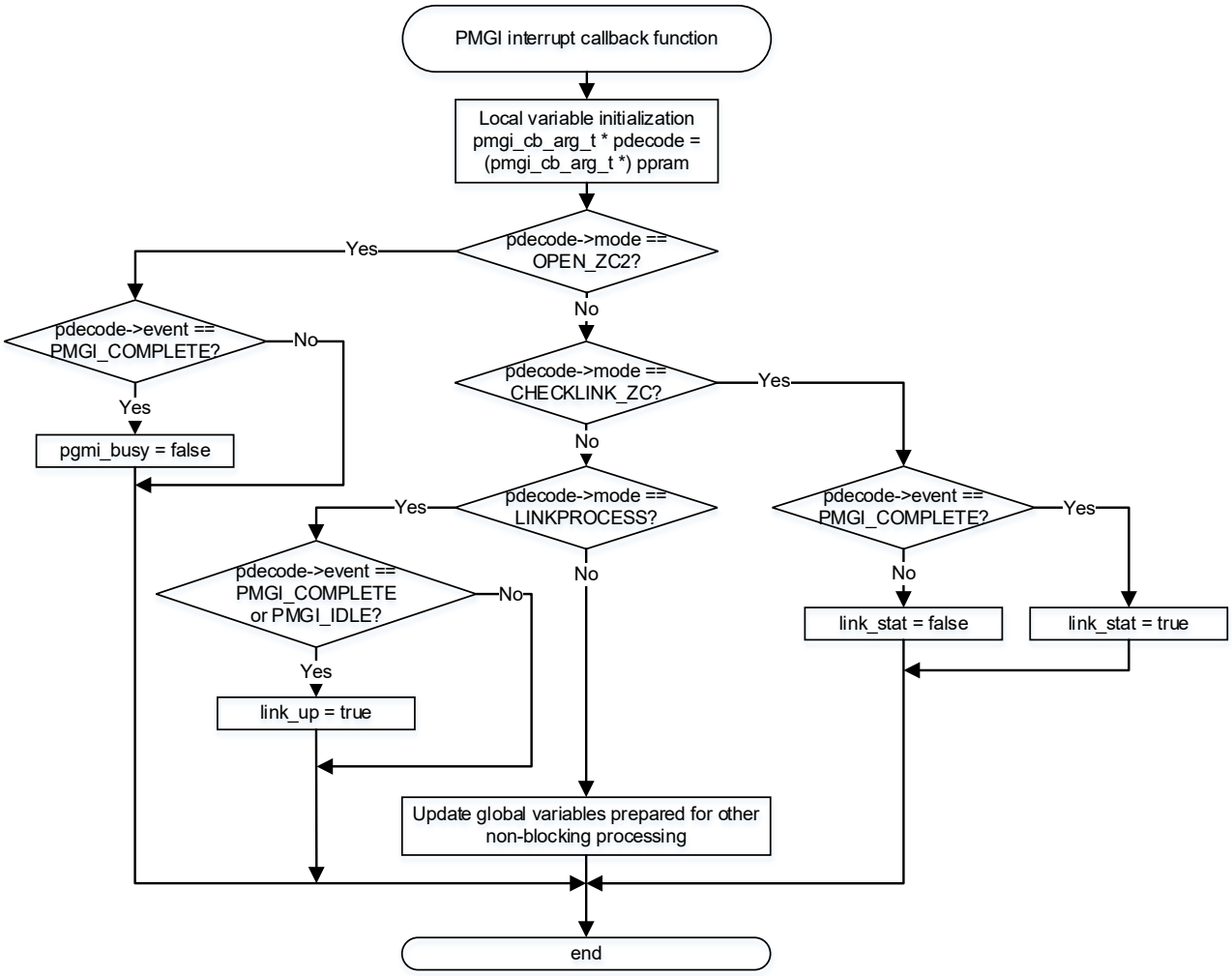


Figure 5.6 Usage Example of API Function Called by Non-Blocking-Call (2) - PMGI Interrupt Callback Function

Note: For PMGI callback functions, refer to 2.11 Callback Function.

6. Appendices

6.1 EPTPC Light FIT Module

Simple switching functionality and multicast frame filtering functionality can be implemented on the RX64M, RX71M, RX72M and RX72N by combining the Ethernet FIT module with the EPTPC Light FIT module.

(1) **Simple Switching**

When using a two-channel ETHERC, frame transfers between channels take place in hardware.

Channel 0 to channel 1, channel 1 to channel 0, or bidirectional can be selected as the transfer direction. Store and forward or cut through can be selected as the transfer method.

(2) **Multicast Frame Filtering**

Processing to receive or discard multicast frames received by the ETHERC is performed in hardware.

It is possible to receive all multicast frames, to receive no multicast frames, or to receive only multicast frames with a designated destination address (up to two addresses can be registered).

For details, refer to the EPTPC Light FIT module application note “RX Family: EPTPC Light Module Using Firmware Integration Technology Modules,” document No. R01AN3035

6.1.1 Usage Notes

When using the Ethernet FIT module and EPTPC Light FIT module together in combination, it is not possible at the same time to use the EPTPC FIT module (full version),*1 which provides time synchronization based on the IEEE 1588 specification.

When using simple switching or multicast frame filtering on the RX64M, RX71M, RX72M or RX72N select one of the following.

- When not using the IEEE 1588 time synchronization functionality:
  Select the EPTPC Light FIT module (module name: r_ptp_light_rx).
- When using the IEEE 1588 time synchronization functionality:
  Select EPTPC FIT module (full version) (module name: r_ptp_rx).

Note 1. RX Family: EPTPC Module Using Firmware Integration Technology, document No. R01AN1943

6.2 Confirmed Operation Environment

This section describes confirmed operation environment for the Ethernet FIT module.

Table 6.1 confirmed operation environment (Rev1.13)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 6.00.000
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian order	Big-endian/Little-endian
Board used	Renesas Starter Kit+ for RX64M (product number.R0K50564MSxxxBE) Renesas Starter Kit+ for RX65N (product number.RTK500565NSxxxxxBE) Renesas Starter Kit+ for RX65N-2MB (product number.RTK50565N2SxxxxxBE)

Table 6.2 confirmed operation environment (Rev1.15)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 6.2.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.08.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian order	Big-endian/Little-endian
Board used	Renesas Starter Kit+ for RX64M (product number.R0K50564MSxxxBE) Renesas Starter Kit+ for RX71M (product number.R0K50571MCxxxBE) Renesas Starter Kit+ for RX65N (product number.RTK500565NSxxxxxBE)

Table 6.3 confirmed operation environment (Rev1.16)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0 IAR Embedded Workbench for Renesas RX 4.10.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201803 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.10.01 Compiler option: The default settings of the integrated development environment.
Endian order	Big-endian/Little-endian
Board used	Renesas Starter Kit+ for RX65N (product number.RTK500565Nxxxxxx)

Table 6.4 confirmed operation environment (Rev1.17)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.4.0 IAR Embedded Workbench for Renesas RX 4.12.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.01 Compiler option: The default settings of the integrated development environment.
Endian order	Big-endian/Little-endian
Board used	Renesas Starter Kit+ for RX72M (product number.RTK5572Mxxxxxxxxxx)

Table 6.5 confirmed operation environment (Rev1.20)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.4.0 IAR Embedded Workbench for Renesas RX 4.12.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.01 Compiler option: The default settings of the integrated development environment.
Endian order	Big-endian/Little-endian
Board used	Renesas Starter Kit+ for RX72N (product number.RTK5572Nxxxxxxxxxx)

Table 6.6 confirmed operation environment (Rev1.21)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.8.0 IAR Embedded Workbench for Renesas RX 4.14.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.02.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.202002 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.14.01 Compiler option: The default settings of the integrated development environment.
Endian order	Big-endian/Little-endian
Board used	Renesas Starter Kit+ for RX72N (product number.RTK5572Nxxxxxxxxxx)

Table 6.7 confirmed operation environment (Rev1.22)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.8.0 IAR Embedded Workbench for Renesas RX 4.20.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.03.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202104 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.20.1 Compiler option: The default settings of the integrated development environment.
Endian order	Big-endian/Little-endian
Board used	Renesas Starter Kit+ for RX65N (product number.RTK500565Nxxxxxx)

Table 6.8 confirmed operation environment (Rev1.23)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.8.0 IAR Embedded Workbench for Renesas RX 4.20.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.04.00 Compiler option: Default setting when using the Smart Configurator.
	GCC for Renesas RX 8.3.0.202104 Compiler option: Default setting when using the Smart Configurator.
	IAR C/C++ Compiler for Renesas RX version 4.20.1 Compiler option: The default settings of the integrated development environment.
Endian order	Big-endian/Little-endian
Board used	Renesas Starter Kit+ for RX65N (product number.RTK500565Nxxxxxx)

Table 6.9 confirmed operation environment (Rev1.24)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2025-01 IAR Embedded Workbench for Renesas RX 5.10.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.07.00 Compiler option: Default setting when using the Smart Configurator.
	GCC for Renesas RX 8.3.0.202411 Compiler option: Default setting when using the Smart Configurator.
	IAR C/C++ Compiler for Renesas RX version 5.10.1 Compiler option: The default settings of the integrated development environment.
Endian order	Big-endian/Little-endian
Board used	-

6.3 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the error: Could not open source file “platform.h”.

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following documents:

- Using CS+:

  Application note “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)”

- Using e^2 studio:

  Application note “Adding Firmware Integration Technology Modules to Projects (R01AN1723)”

When using a FIT module, the board support package FIT module (BSP module) must also be added to the project. For this, refer to the application note “Board Support Package Module Using Firmware Integration Technology (R01AN1685)”.

(2) Q: I have added the FIT module to the project and built it. Then I got the error: This MCU is not supported by the current r_ether_rx module.

A: The FIT module you added may not support the target device chosen in the user project. Check the supported devices of added FIT modules.

(3) Q: I have added the FIT module to the project and built it. Then I got an error for when the configuration setting is wrong.

A: The setting in the file “r_ether_rx_config.h” may be wrong. Check the file “r_ether_rx_config.h”. If there is a wrong setting, set the correct value for that. Refer to 2.7 Configuration Overview for details.

(4) Q: Data transmission and reception is not started.

A: The pin setting may not be performed correctly. When using this FIT module, the pin setting must be performed. Refer to 4 Pin Setting for details.

7. Provided Modules

The module provided can be downloaded from the Renesas Electronics website.

8. Ethernet FIT Module Usage Notes

Keep the following points in mind when using the Ethernet FIT module.

- If broken frames or noise on an external line cause detection of a frame error during reception by the ETHERC and EPTPC on the RX64M, RX71M, RX72M, RX72N or RX66N, proper reception may no not be possible even if subsequent received frames are normal. For details, refer to the technical updates and application notes listed below.
- Notes on Using Ethernet Controller (Technical Notification No. TN-RX*-A125A/E)
- RX Family Retrieve Recommend Operation of INFABT Occurrence in The Ethernet Controller (Doc No. R01AN2604)
- If the EDMR.SWR bit is set to 1 while data transfer is being performed by EDMAC0, EDMAC1, or PTPEDMAC in RX64M / RX71M / RX72M/ RX72N/ RX66N, data at address 0000 0000h to 0000 001Fh may be destroyed. Please refer to the following technical update for details.
- Notes on software reset of the DMA controller (EDMAC) for the RX64M group and RX71M group Ethernet controller (Technical Update No. TN-RX * -A0212A / J)

9. Reference Documents

User's Manual: Hardware
RX64M Group User's Manual: Hardware (Doc No. R01UH0377)
RX71M Group User's Manual: Hardware (Doc No. R01UH0493)
RX65N Group, RX651 Group User's Manual: Hardware (Doc No. R01UH0590)
RX72M Group User's Manual: Hardware (Doc No. R01UH0804)
RX72N Group User's Manual: Hardware (Doc No. R01UH0824)
RX66N Group User's Manual: Hardware (Doc No. R01UH0825)
(The latest version can be downloaded from the Renesas Electronics website.)

Technical Update/Technical News
(The latest information can be downloaded from the Renesas Electronics website.)

User's Manual: Development Tools
RX Family C/C++ Compiler, Assembler, Optimizing Linkage Editor Compiler Package (R20UT0570)
(The latest version can be downloaded from the Renesas Electronics website.)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jul 29, 2014	—	First edition issued
1.01	Jan 28, 2015	1	RX71M added to Target Devices
		5	Notes 1 to 3 of 2.6, Configuration Overview, amended Table 2.2 added
		9	Steps 7 and 8 of 2.10.1, Adding the Ethernet FIT Module, amended
		10	Special Notes of 3.1, R_ETHER_Initial(), amended
		23	Special Notes of 3.11, R_ETHER_CheckWrite(), amended
		30	Special Notes of 3.14, R_ETHER_Control(), amended
		32	Address 0x00120064 deleted from table 4.1, Program Section Allocation
		53	RX71M Group User's Manual: Hardware added to 6, Reference Documents Information regarding development environment user's manual amended
1.02	Mar 27, 2015	—	R_ETHER_LinkProcess() in r_ether_rx.c amended
1.10	Mar 15, 2016	1	RX63N added to Target Devices
		3	Limitations deleted under Overview
		5	#define ETHER_CFG_EINT_INT_PRIORITY added to 2.6, Configuration Overview
		6, 7	Notes 4 to 8 on #define ETHER_CFG_USE_LINKSTA added to 2.6, Configuration Overview
		7	2.7 Code Size added
		8	Description of ether_cmd_t in 2.8, Arguments, amended
		10	Description of ether_return_t in 2.9, Return Values, amended
		11	Description of (2) Callback Function Called by EINT0/EINT1 Status Interrupts in 2.10, Callback Function, amended. Note 1. added
			Description of 2.11, Adding the FIT Module, amended
		12	2.12, Ethernet Frame Format, added
		13 to 43	Description of API functions in 3., API Functions, amended
		45	4.2, Ethernet FIT Module Initial Settings, added
		46	4.3, EPTPC Light FIT Module, added
		47	4.4, Magic Packet Detection Operation, added
		—	4.2, Sample Code, deleted
		48	6., Ethernet FIT Module Usage Notes, added
1.11	Oct 1, 2016	—	Pin setting in the Ethernet FIT module has been removed for support pin setting function of e2 studio.
		1	RX65N added to Target Devices
		8, 9	Notes 5, 7 and 8 amended
		43	Description in 3.14, R_ETHER_Control(), amended
		48	Description of Figure 4.1 in 4.2, Ethernet FIT Module Initial Settings, amended
		49	4.3, Ethernet FIT Module Pin Settings, added
		52	5, Appendices, added

1.12	Nov 11, 2016	Program	The module is updated to fix the software issue. Description: When R_ETHER_LinkProcess function is called, there are cases when link up/link down are not processed successfully. Conditions: ETHER_CFG_USE_LINKSTA is set to a value of 0. Corrective action: Please use the Ethernet FIT module Rev1.12.
1.13	Oct 01, 2017	—	Supported RX65N-2MB version.
		54	Move 2.3 Operating Condition to 6.2 Operation Confirmation Environment
		6	2.4 Usage of Interrupt Vector, fixed
		9	Notes 7 and 8 amended
		14	2.12 Adding the FIT Module, amended
		47	Move 4.3 Ethernet FIT Module Pin Setting to 4. Pin Setting
		49, 50	4.1.2 Pin Setting example for using RSK+RX63N and 4.1.3 Pin Setting example for using RSK+RX65N/RSK+RX65N-2M
		51	5.1 Section Allocation, amended
		52	5.2 Ethernet FIT Module Initial Setting, amended
		55	6.3 Troubleshooting, added
1.14	Jan 08, 2018	—	Supported setting function of configuration option using GUI on Smart Configurator
		6	2.7 Configuration Overview, fixed
		46	Table 4.1, fixed
		55	9. Reference Documents, amended
1.15	May 07, 2018	Program	The module is updated to fix the software issue. Description: When R_ETHER_Read_ZC2 function or R_ETHER_Read function is called, there is case when the Ethernet frame cannot be received normally. Conditions: R_ETHER_Read_ZC2 function or R_ETHER_Read function is called in the interrupt function. Corrective action: Please use the Ethernet FIT module Rev1.15. The following function is changed by this correction. R_ETHER_LinkProcess function Corresponding Tool News number : R20TS0307

		Program	The module is updated to fix the software issue. Description: When R_ETHER_LinkProcess function is called, there is case when link up processing is not completed normally. Conditions: When R_ETHER_LinkProcess function is called, PHY auto-negotiation is not completed. Corrective action: Please use the Ethernet FIT module Rev1.15. The following function is changed by this correction. R_ETHER_LinkProcess function Corresponding Tool News number : R20TS0307
		Program	The module is updated to fix the software issue. Description: When R_ETHER_Read_ZC2 function or R_ETHER_Read function is called, there is case when execution of function is not completed. Conditions: R_ETHER_LinkProcess function is called in the interrupt function. Corrective action: Please use the Ethernet FIT module Rev1.15. The following function is changed by this correction. R_ETHER_Read_ZC2 function Corresponding Tool News number : R20TS0307
		9	2.8 Code Size, amended
		54	6.2 Confirmed Operation Environment, amended
1.16	May 20, 2019	-	Update the following compilers: GCC for Renesas RX IAR C/C++ Compiler for Renesas RX
		1	Added Target Compilers.
		1	Deleted R01AN1723, R01AN1826, R20AN0451 from Related Documents.
		5	Added revision of dependent r_bsp module in 2.2 Software Requirements.
		8	2.8 Code Size, amended.
		47	R_ETHER_GetVersion function, deleted special notes.
		53-56	5.1.1 and 5.1.2, added.
		61	Added Table 6.3 Operation Confirmation Environment (Ver. 1.16).
1.17	Jul 30,2019	-	Supported RX72M version.
		5	1.3 Use Limit, added.
		6	Table 2.1 List of Usage of Interrupt Vectors, amended.
		9	Note 5, Note6, Note 7, Note 8, amended.
			Note 9, added.
			2.8 Code Size, amended.
		16	2.14 “for”, “while” and “do while” statements, added.
		17-46	Delete “Reentrant” item on the API description page.

		47-50	Table 4.5 Pin setting example for channel 0 of RX72M and Table 4.6 Pin setting example for channel 1 of RX72M, added.
		53	5.1 Section Allocation, amended.
		58	5.1.3 Notes on Section Allocation, amended.
		60	6.1 EPTPC Light FIT Module add RX72M.
		62	6.2 Confirmed Operation Environment, amended.
		65	8 Ethernet FIT Module Usage Notes, amended.
			9 Reference Documents, amended.
		Program	Ethernet FIT module fixed due to software failure Description: If an Ethernet frame is received after buffer open processing in the R_ETHER_Read_ZC2_BufRelease function or R_ETHER_Read function, the following phenomenon may
			occur. (1) There are cases where the received Ethernet frame can not be read. (2) The received error frame may be read as a normal Ethernet frame. Conditions: An Ethernet frame is received after buffer open processing in the R_ETHER_Read_ZC2_BufRelease function or R_ETHER_Read function. Corrective action: Please use Ethernet FIT module Rev1.17. The following functions have been changed by this correction. R_ETHER_Read_ZC2_BufRelease Function Correspondence tool news number: R20TS0447
1.20	Nov 22,2019	-	Supported RX72N and RX66N version.
		5	Table 1.1 API Functions, amended.
		6	Table 2.1 List of Usage of Interrupt Vectors, amended.
		8	Configuration options in r_ether_rx_config.h, amended.
		10	Note 4, Note 5, Note6, Note 7, Note 8, amended. Note 9, added.
		11	2.8 Code Size, amended.
		16-17	2.11 Callback Function, amended.
		23	3.2 R_ETHER_Open_ZC2(), amended.
		34	3.8 R_ETHER_CheckLink_ZC(), amended.
		36	3.9 R_ETHER_LinkProcess(), amended.
		38	3.10 R_ETHER_WakeOnLAN(), amended.
		46-49	3.14 R_ETHER_Control(), amened.
		51	3.15 R_ETHER_WritePHY(), added.
		52	3.16 R_ETHER_ReadPHY(), added.
		-	4.1.2 Pin setting example for using RSK+RX63N, deleted.
		59	4.1.3 Pin setting example for using RSK+RX72N, added.
		70	Table 6.5 confirmed operation environment (Rev1.20), added.
		72	8 Ethernet FIT Module Usage Notes, amended.
			9 Reference Documents, amended.

		Program	Ethernet FIT module fixed due to software failure Description: When one receive descriptor is set, Ethernet frames may not be received after the buffer release processing in the “R_ETHER_Read_ZC2_BufRelease” function or “R_ETHER_Read” function. Conditions: Use the R_ETHER_Read_ZC2_BufRelease function or R_ETHER_Read function to receive an Ethernet frame. Corrective action: Please use Ethernet FIT module Rev1.20. The following functions are changed by this modification. R_ETHER_Read_ZC2 R_ETHER_Read_ZC2_BufRelease function Corresponding tool news number: R20TS0481
		Program	Ethernet FIT module fixed due to software failure Description: When one transmission descriptor is set, the Ethernet frame may not be transmitted after the transmission start processing in the “R_ETHER_Write_ZC2_SetBuf” function or the “R_ETHER_Write” function. Conditions: Use the R_ETHER_Read_ZC2_SetBuf function or R_ETHER_Write function to transmit an Ethernet frame. Corrective action: Please use Ethernet FIT module Rev1.20. The following functions are changed by this modification. R_ETHER_Write_ZC2_GetBuf function Corresponding tool news number: R20TS0481
1.21	Sep 10,2020	10	Note 10 amended
		18	2.12 Adding the FIT Module to Your Project, amended.
		56-57	Table 4.5 and 4.6, amended.
		59	Table 4.8, amended.
		68	5.4 Notes on Accessing MII/RMII Registers, added.
		70	5.5 How to Use API Function Called in Non-Blocking, added.
		75	Table 6.6 confirmed operation environment (Rev1.21), added.
1.22	Nov 21,2021	9, 10	2.7 Compile Time Settings, amended.
1.23	Mar 01,2022	62-66	5.1 Section Allocation, amended.
1.24	Mar 20,2025	76	Table 6.6 confirmed operation environment (Rev1.24), added.
		Program	Changed the disclaimer in program sources

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 be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. 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.
6. 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.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. 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.5.0-1 October 2020)

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/.