

# RX Family

R01AN2009EJ0100

Rev.1.00

Jul 18, 2014

## Ethernet Module Using Firmware Integration Technology

### Introduction

This application note describes an Ethernet module that uses Firmware Integration Technology (FIT). This module performs Ethernet/IEEE 802.3 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.

### Target Device

This API supports the following device.

RX64M

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

### Contents

1. Overview .....	2
2. API Information.....	3
3. API Functions .....	10
4. Appendices.....	32
5. Provided Modules.....	53
6. Reference Documents.....	53

## 1. Overview

The Ethernet FIT module uses an Ethernet controller (ETHERC) and an Ethernet controller DMA controller (EDMAC) to implement Ethernet/IEEE 802.3 frame transmission and reception. This 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.

### Limitations

The link signal output of the PHY-LSI must be connected to either the ET0\_LINKSTA pin or the ET1\_LINKSTA pin, depending on the channel used.

## 1.1 Ethernet FIT Module

This module is implemented in a project and used as the API. Refer to 2.9 Adding Driver to Your Project for details on implementing the module to the project.

## 1.2 Outline of the API

Table 1.1 lists the API Functions.

**Table 1.1 API Functions**

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

## 2. API Information

This driver API adheres to the Renesas API naming standards.

---

### 2.1 Hardware Requirements

---

This driver requires your MCU supports the following feature:

- ETHERC
- EDMAC

---

### 2.2 Software Requirements

---

This driver is dependent upon the following packages:

- r\_bsp

---

### 2.3 Supported Toolchains

---

This driver is tested and works with the following toolchain:

- Renesas RX Toolchain v.2.01.00

---

### 2.4 Header Files

---

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

---

### 2.5 Integer Types

---

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

## 2.6 Configuration Overview

The configuration options in this module are specified in `r_ether_rx_config.h`. The option names and setting values are listed in the table below.

### Configuration options in `r_ether_rx_config.h`

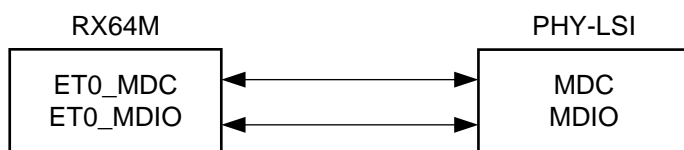
#define ETHER_CFG_MODE_SEL Note: Default value = 0	Sets the interface between ETHERC and the Ethernet PHY-LSI. If set to 0, MII (Media Independent Interface) is selected. If set to 1, RMII (Reduced Media Independent Interface) is selected.
#define ETHER_CFG_CH0_PHY_ADDRESS Note: Default value = 0	Specify the PHY-LSI address used by ETHERC channel 0. Specify a value between 0 and 15.
#define ETHER_CFG_CH1_PHY_ADDRESS Note: Default value = 1	Specify the PHY-LSI address used by ETHERC channel 1. Specify a value between 0 and 15.
#define ETHER_CFG_EMAC_RX_DESCRIPTOR 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_DESCRIPTOR 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_PRIORITY 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.
#define ETHER_CFG_CH0_PHY_ACCESS Note: Default value = 0* <sup>1</sup>	Specify the PHY access channel used by ETHERC channel 0. When 0 is specified, ETHERC0 is used for PHY register access.* <sup>2</sup> When 1 is specified, ETHERC1 is used for PHY register access.* <sup>3</sup>
#define ETHER_CFG_CH1_PHY_ACCESS Note: Default value = 0* <sup>1</sup>	Specify the PHY access channel used by ETHERC channel 1. When 0 is specified, ETHERC0 is used for PHY register access.* <sup>2</sup> When 1 is specified, ETHERC1 is used for PHY register access.* <sup>3</sup>
#define ETHER_CFG_PHY_MII_WAIT Note: Default value = 8	Specify the access timing for the MII or RMII register. Specify a value of 8 or greater.
#define ETHER_CFG_PHY_DELAY_RESET Note: Default value = 0x00020000	Specify the wait time for PHY-LSI reset completion.
#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_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.

Notes: 1. Refer to table 2.1 regarding settings for operating the Ethernet FIT module on the Renesas Starter Kit for RX64M (product number: R0K50564MSxxxBE).

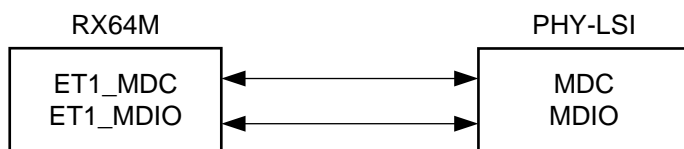
**Table 2.1 ETHER\_CFG\_CH0\_PHY\_ACCESS/ETHER\_CFG\_CH1\_PHY\_ACCESS Settings**

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

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



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



## 2.7 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 */
} 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_param_t;

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

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

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

typedef enum
{
    ETHER_CB_EVENT_ID_WAKEON_LAN, /* Magic packet detection */
    ETHER_CB_EVENT_ID_LINK_ON,    /* Link up detection */
    ETHER_CB_EVENT_ID_LINK_OFF    /* Link down detection */
} ether_cb_event_t;

typedef struct
{
    uint32_t            channel; /* ETHERC channel */
    ether_cb_event_t    event_id; /* Event code for callback function */
    uint32_t            status_ecsr; /* ETHERC status register for interrupt handler */
    uint32_t            status_eesr; /* ETHERC/EDMAC status register for interrupt handler */
} ether_cb_arg_t;
```

---

## 2.8 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_OTHER         /* Other error */
} ether_return_t;
```

## 2.9 Callback Function

### (1) Callback Function Called by API Function R\_ETHER\_LinkProcess

In this 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 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 2.2 are stored is passed as an argument. If the value of this argument is to be used outside the callback function, its value should be copied into, for example, a global variable.

**Table 2.2 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

This module calls an interrupt handler when an interrupt indicating a condition other than the following occurs.

- Magic packet detection
- Link signal change 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 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.



---

## 2.10 Adding Driver to Your Project

---

The FIT module must be added to each project in the e<sup>2</sup> Studio.

You can use the FIT plug-in to add the FIT module to your project, or the module can be added manually.

It is recommended to use the FIT plug-in as you can add the module to your project easily and also it will automatically update the include file paths for you.

To add the FIT module using the plug-in, refer to chapter 2. “Adding FIT Modules to e<sup>2</sup> studio Projects Using FIT Plug-In” in the “Adding Firmware Integration Technology Modules to Projects” application note (R01AN1723EU).

To add the FIT module manually, refer to the following 2.10.1 “Adding the Ethernet FIT Module (when not using the plug-in)”.

When using the FIT module, the BSP FIT module also needs to be added. For details on the BSP FIT module, refer to the “Board Support Package Module Using Firmware Integration Technology” application note (R01AN1685EU).

### 2.10.1 Adding the Ethernet FIT module (when not using the plug-in)

1. This application note is distributed with a zip file package that includes the Ethernet FIT module in its own folder `r_ether_rx`.
2. Unzip the package into the location of your choice.
3. In a file browser window, browse to the directory where you unzipped the distribution package and locate the `r_ether_rx` folder.
4. Open your e<sup>2</sup> Studio workspace.
5. In the e<sup>2</sup> Studio project explorer window, select the project that you want to add the Ethernet FIT module to.
6. Drag and drop the `r_ether_rx` folder from the browser window into your e<sup>2</sup> Studio project at the top level of the project.
7. Update the source search/include paths for your project by adding the paths to the module files:
  - a. Navigate to the “Add directory path” control:
    - i. ‘project name’ → properties → C/C++ Build → Settings → Compiler → Source -Add (green + icon)
  - b. Add the following paths:
    - i. “\${workspace\_loc}/\${ProjName}/r\_ether\_rx”
    - ii. “\${workspace\_loc}/\${ProjName}/r\_ether\_rx/src”

Whether you used the plug-in or manually added the package to your project, it is necessary to configure the module for your application.

8. Locate the `r_ether_rx_config_reference.h` file in the `r_ether_rx/ref/targets/rx64m` source folder in your project and copy it to the `r_config` folder in your project.
9. Change the name of the copy in the `r_ether_rx_config_reference.h` to `r_ether_config.h`.
10. Make the required configuration settings by editing the `r_ether_rx_config.h` file. Refer to 2.6 “Compile Settings” for details.

### 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.

**Reentrant**

Function is not reentrant.

**Example**

Refer to 4.2, Sample Code.

**Special Notes:**

This function must be called before calling the R\_ETHER\_Open() function.

### 3.2 R\_ETHER\_Open\_ZC2()

When using the ETHERC 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 channel number (0, 1).

*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

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

#### 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.

#### Reentrant

Function is reentrant for different channels.

#### Example

Refer to 4.2, Sample Code.

**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 channel number (0, 1).

#### Return Values

<i>ETHER_SUCCESS</i>	<i>/* Processing completed successfully */</i>
<i>ETHER_ERR_INVALID_CHAN</i>	<i>/* Nonexistent channel number */</i>

#### 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.

#### Reentrant

Function is reentrant for different channels.

#### Example

Refer to 4.2, Sample Code.

#### 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 channel number (0, 1).

*\*\* pbuf*

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

#### Return Values

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

#### 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 any of following error occurs on a frame, this function recognizes that a receive framing error occurs in the frame.

- Receive FIFO overflow
- Receive residual-bit frame
- Receive too-long frame
- Receive too-short frame
- PHY-LSI receive error
- CRC error on received frame

The function discards the data in the descriptor where receive framing error occurred, clears the status and continues reading data.

#### Reentrant

Function is reentrant for different channels.

**Example**

Refer to 4.2, Sample Code.

**Special Notes:**

None

---

### 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 channel number (0, 1).

#### Return Values

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

#### 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.

#### Reentrant

Function is reentrant for different channels.

#### Example

Refer to 4.2, Sample Code.

#### Special Notes:

Execute this function after reading data by the R\_ETHER\_Read\_ZC2() function.



### 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 channel number (0, 1).

*\*\* 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

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

#### 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.

#### Reentrant

Function is reentrant for different channels.

#### Example

Refer to 4.2, Sample Code.

**Special Notes:**

None

### 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 channel number (0, 1).

*len*

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

#### Return Values

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

#### 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.

#### Reentrant

Function is reentrant for different channels.

#### Example

Refer to 4.2, Sample Code.

#### 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.

---

### 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 channel number (0, 1).

#### Return Values

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

#### 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. ETHER\_SUCCESS is returned when the link is up, and ETHER\_ERR\_OTHER when the link is down.

#### Reentrant

Function is reentrant for different channels.

#### Example

Refer to 4.2, Sample Code.

#### 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 channel number (0, 1).

#### 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.

- 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:
  - 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.

#### Reentrant

Function is reentrant for different channels.

#### Example

Refer to 4.2, Sample Code.

#### Special Notes:

Make sure to call this function periodically in loop processing within the normal processing routine. Note that Ethernet transmission and reception may not operate correctly, nor Ethernet driver may 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 channel number (0, 1).

#### Return Values

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

#### 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.

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

#### Reentrant

Function is reentrant for different channels.

#### Example

Refer to 4.2, Sample Code.

#### 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 channel number (0, 1).

#### Return Values

<i>ETHER_SUCCESS</i>	<i>/* Processing completed successfully */</i>
<i>ETHER_ERR_INVALID_CHAN</i>	<i>/* Nonexistent channel number */</i>

#### 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.

#### Reentrant

Function is reentrant for different channels.

#### Example

Refer to 4.2, Sample Code.

#### Special Notes:

This function should be called after transmit data has been written with the R\_ETHER\_SetBuf() function.

Note that it takes several tens of microseconds for data transmission to actually complete after the R\_ETHER\_SetBuf() function is called. Therefore, when using the R\_ETHER\_Close() function to shut down the Ethernet module following data transmission, call the R\_ETHER\_CheckWrite() function after calling the R\_ETHER\_SetBuf() function and, after waiting for data transmission to finish, call the R\_ETHER\_Close() function. Calling the R\_ETHER\_Close() 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 channel number (0, 1).

*\*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

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

#### 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 any of following error occurs on a frame, this function recognizes that a receive framing error occurs in the frame.

- Receive FIFO overflow
- Receive residual-bit frame
- Receive too-long frame
- Receive too-short frame
- PHY-LSI receive error
- CRC error on received frame

The function discards the data in the descriptor where receive framing error occurred, clears the status and continues reading data.

#### Reentrant

Function is reentrant for different channels.



**Example**

Refer to 4.2, Sample Code.

**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.

### 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 channel number (0, 1).

*\* 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

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

#### 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.

#### Reentrant

Function is reentrant for different channels.

**Example**

Refer to 4.2, Sample Code.

**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.

### 3.14 R\_ETHER\_Control()

The R\_ETHER\_Control() function performs the processing that corresponds to the control code.

#### Format

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

#### Parameters

*cmd*

Specifies the control code.

*control*

Specify the parameters according to the control code.

#### Return Values

*ETHER\_SUCCESS*                      /\* Processing completed successfully \*/  
*ETHER\_ERR\_INVALID\_CHAN*           /\* Nonexistent channel number \*/  
*ETHER\_ERR\_CHAN\_OPEN*            /\* Indicates the Ethernet cannot be opened because it is being used by another application \*/  
*ETHER\_ERR\_INVALID\_ARG*           /\* Invalid argument \*/

#### 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	Specifies the pins for ETHERC and cancels module stop for the ETHERC and EDMAC. The second argument specifies the ETHERC channel.
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.

**Reentrant**

Function is reentrant.

**Example**

To register a callback function.)

```
void callback(void*);

ether_return_t    ret;
ether_param_t     param;
ether_cb_t        cb_func;

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

ret = R_ETHER_Contorl(CONTROL_SET_CALLBACK, 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)

See Example under R\_ETHER\_Initial.

Transitioning ETHERC/EDMAC to module stop)

See Example under R\_ETHER\_CheckWrite.

#### Special Notes:

Register callback functions and interrupt handlers before calling the R\_ETHER\_Open function. It may not be possible to detect the first interrupt if the preceding are registered after the R\_ETHER\_Open 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.

---

### 3.15 R\_ETHER\_GetVersion()

---

This function returns the API version.

**Format**

```
uint32_t R_ETHER_GetVersion(void);
```

**Parameters**

None

**Return Values**

*Version number*

**Properties**

Prototyped in r\_ether\_rx\_if.h.

**Description**

Returns the API version number.

**Reentrant**

Function is reentrant for different channels.

**Example**

```
uint32_t    version;  
  
version = R_ETHER_GetVersion();
```

**Special Notes:**

This function is inlined using '#pragma inline'.

## 4. Appendices

### 4.1 Section Allocation

Table 4.1 shows a sample section allocation for this module.

**Table 4.1 Program Section Allocation**

Address	Device	Section	Description
0x00000000	Internal RAM	B_ETHERNET_BUFFERS_1	Transmit buffer and receive buffer area
		B_RX_DESC_1	Receive descriptor area
		B_TX_DESC_1	Transmit descriptor area
		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)
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 (C\$DEC, C\$BSEC, C\$VECT) of C\$* section
		D*	Initialization data area
		P*	Program area
		W*	Branch table area for switch statements
		L	String literal area
0xFFFFFFF80		EXCEPTVECT	Interrupt vector area
0xFFFFFFFFC		RESETVECT	Reset vector area

#### 4.1.1 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.



## 4.2 Sample Code

API usage examples are shown below.

The MAC addresses appearing in the code are examples for reference only. The customer's products should always use MAC addresses registered with the IEEE.

```
#include <stdint.h>

#include <machine.h>

#include "platform.h"

#include "r_ether_rx_if.h"

#include "r_ether_rx_config.h"

/*****
*****
Macro definitions
*****
*****/
#ifdef ETHER_DRIVER_TEST
    #define STATIC
#else /* ETHER_DRIVER_TEST */
    #define STATIC static
#endif /* ETHER_DRIVER_TEST */

/*
 * This program supports both the host and remote host.
 * Undefine the following macro when using the program of the host.
 * To use the program of the remote host, define the following macro.
 */

/* Specify the time-out period in ms to wait to receive data. */
#define NO_RECEIVE_TIMEOUT_MS (5000)

/*****
*****
Typedef definitions
*****
*****/
/* Type of sample application */
typedef enum
{
    ETHER_SAMPLE_NOTHING,
    ETHER_SAMPLE_TX_RX_WITHOUT_PAUSE_FRAME_NON_ZERO_COPY, /* Transmission and Reception without
pause frame */
    ETHER_SAMPLE_TX_RX_WITHOUT_PAUSE_FRAME_ZERO_COPY,      /* Transmission and Reception without
pause frame */
    ETHER_SAMPLE_TX_RX_WITH_PAUSE_FRAME,                  /* Transmission and Reception with pause frame */
    ETHER_SAMPLE_MAGIC_PACKET_DETECTION,                   /* Magic packet detection */
    ETHER_SAMPLE_INVALID,
} ether_sample_t;

/* Comparison result of data */
typedef enum
```

```

{
    DATA_COMPARE_OK = 0,                /* Comparison result is match */
    DATA_COMPARE_NG = -1                /* Comparison result is mismatch */
} data_compare_result_t;

/*****
*****
Imported global variables and functions (from other files)
*****
*****/

/*****
*****
Exported global variables (to be accessed by other files)
*****
*****/

/*****
*****
Private global variables and functions
*****
*****/
/*
 * This is used as 1-ms counter.
 */
STATIC volatile uint32_t cmt0_cmi0_count = 0;

static volatile uint8_t  pause_enable = ETHER_FLAG_OFF;
static volatile uint8_t  magic_packet_detect[ETHER_CHANNEL_MAX];
static volatile uint8_t  link_detect[ETHER_CHANNEL_MAX];

static volatile uint32_t g_status_ecsr[ETHER_CHANNEL_MAX];
static volatile uint32_t g_status_eesr[ETHER_CHANNEL_MAX];

/*
 *The MAC address included in the following sample codes is an example of the reference.
 *Please use the MAC address that was submitted to the IEEE always in your product.
 */

/*
 * MAC address set to the host (remote host)
 */
static const uint8_t mac_addr_src[6] =
{
    0x00,0x01,0x02,0x03,0x04,0x05
};

/* (1) Transmit data */
static uint8_t send_data[60] =
{
    0x0A,0x0B,0x0C,0x0D,0x0E,0x0F,                /* Destination MAC address */
    0x00,0x01,0x02,0x03,0x04,0x05,                /* Source MAC address */
    0x00,0x00,                /* The type field is not used. */
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* Use the first byte of the data field as
                                                         the packet ID. */
    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
};

/* (2) ACK to (1) */
static uint8_t ack_data[60] =
{
    0x00,0x01,0x02,0x03,0x04,0x05, /* Destination MAC address */
    0x0A,0x0B,0x0C,0x0D,0x0E,0x0F, /* Source MAC address */
    0x00,0x00, /* The type field is not used. */
    0x00,'a','c','k', 0x00,0x00,0x00,0x00,0x00,0x00, /* Use the first byte of the data field as
                                                         the packet ID. */

    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
};

/* Notify the remote host that the host has detected a Magic Packet. */
static const uint8_t magic_packet_detect_notify[60] =
{
    0x0A,0x0B,0x0C,0x0D,0x0E,0x0F, /* Destination MAC address */
    0x00,0x01,0x02,0x03,0x04,0x05, /* Source MAC address */
    0x00,0x00, /* The type field is not used. */
    'd','e','t','e','c','t','e','d', 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,
    0x00,0x00,0x00,0x00,0x00,0x00
};

/* Notify the host that the remote host recognized that the host had received a Magic Packet. */
static const uint8_t magic_packet_detect_ack[60] =
{
    0x00,0x01,0x02,0x03,0x04,0x05, /* Destination MAC address */
    0x0A,0x0B,0x0C,0x0D,0x0E,0x0F, /* Source MAC address */
    0x00,0x00, /* The type field is not used. */
    'a','c','k', 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,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00
};

static void tx_rx_sample_non_zero_copy(uint32_t channel);
static void tx_rx_sample_zero_copy(uint32_t channel);

static void magic_packet_detection_sample(uint32_t channel);

static void callback_sample(void * pparam);
static void callback_wakeon_lan(uint32_t channel);
static void callback_link_on(uint32_t channel);
static void callback_link_off(uint32_t channel);
static void int_handler_sample(void *pparam);

```

```

static void    int_handler_0_sample(uint32_t status_ecsr, uint32_t status_eesr);
static void    int_handler_1_sample(uint32_t status_ecsr, uint32_t status_eesr);

static void    data_copy(uint8_t * pdst, const uint8_t * psrc, const uint16_t len);
static int8_t  data_compare(const uint8_t * pdst, const uint8_t * psrc, const uint16_t len);

STATIC void    cmt0_init(void);
#pragma interrupt (cmt0_cmi0_isr(vect = _VECT(_CMT0_CMI0), enable))
static void    cmt0_cmi0_isr(void);

/*****
*****
* Function Name: main
* Description   : The main loop
* Arguments    : none
* Return Value  : none
*****
*****/
void main(void)
{
    ether_sample_t  sample_app;
    uint32_t        channel;

    /* Select the channel of Ether to use */
    channel = ETHER_CHANNEL_0;

    /* Select the type of sample application */
    // sample_app = ETHER_SAMPLE_TX_RX_WITHOUT_PAUSE_FRAME_NON_ZERO_COPY;
    // sample_app = ETHER_SAMPLE_TX_RX_WITHOUT_PAUSE_FRAME_ZERO_COPY;

    // sample_app = ETHER_SAMPLE_TX_RX_WITH_PAUSE_FRAME;
    sample_app = ETHER_SAMPLE_MAGIC_PACKET_DETECTION;

    switch (sample_app)
    {
        case ETHER_SAMPLE_TX_RX_WITHOUT_PAUSE_FRAME_NON_ZERO_COPY: /* Transmission and Reception
                                                                    without pause frame */

            /*
             * When not using a pause frame, set the value of the pause_enable to
             * ETHER_FLAG_OFF before calling the R_ETHER_Open_ZC2 function.
             */
            pause_enable = ETHER_FLAG_OFF;
            tx_rx_sample_non_zero_copy(channel);
            break;

        case ETHER_SAMPLE_TX_RX_WITHOUT_PAUSE_FRAME_ZERO_COPY: /* Transmission and Reception
                                                                without pause frame */

            /*
             * When not using a pause frame, set the value of the pause_enable to
             * ETHER_FLAG_OFF before calling the R_ETHER_Open_ZC2 function.
             */
            pause_enable = ETHER_FLAG_OFF;
            tx_rx_sample_zero_copy(channel);
    }
}

```

```

        break;

    case ETHER_SAMPLE_TX_RX_WITH_PAUSE_FRAME: /* Transmission and Reception with pause frame */
        /*
         * When using a pause frame, set the value of the pause_enable to
         * ETHER_FLAG_ON before calling the R_ETHER_Open_ZC2 function.
         */
        pause_enable = ETHER_FLAG_ON;
        tx_rx_sample_zero_copy(channel);

        break;

    case ETHER_SAMPLE_MAGIC_PACKET_DETECTION: /* Magic packet detection */
        magic_packet_detection_sample(channel);

        break;
    default:
        break;
}

while (1)
{
    /* Do Nothing */
}

} /* End of function main() */

/*****
*****
* Function Name: tx_rx_sample_non_zero_copy
* Description   : Program example of transmission/reception at the host
* Arguments     : channel -
*                Ethernet channel number
* Return Value  : none
*****
*****/
static void tx_rx_sample_non_zero_copy(uint32_t channel)
{
    enum { WAIT_LINK_ON, DATA_SEND, ACK_RECEIVE };
    int32_t      ret;
    uint8_t      read_buffer[ETHER_CFG_BUFSIZE];
    uint8_t      status;
    int8_t       compare_result;

    ether_param_t param;
    ether_cb_t    cb_func;

    /* ---- Disable maskable interrupts ---- */
    R_BSP_InterruptsDisable();

    /* Initialization of the Ethernet driver */
    R_ETHER_Initial();

    /* ---- Enable maskable interrupts ---- */
    R_BSP_InterruptsEnable();

```

```
/* 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);

/* ETHERC/EDMAC power on */
param.channel = channel;
ret = R_ETHER_Control(CONTROL_POWER_ON, param);

/* Ether Open */
ret = R_ETHER_Open_ZC2(channel, mac_addr_src, pause_enable);

if (ETHER_SUCCESS == ret)
{
    /* Start transmitting data when switch SW1 is pressed. */
    while (SW_ACTIVE != SW1)
    {
        /* Do Nothing */
    }

    status = WAIT_LINK_ON; /* Initial state */

    while (1)
    {
        /*
         * The R_ETHER_LinkProcess function performs link signal change interrupt processing
         * and Magic Packet detection interrupt processing.
         * Make sure to call this function periodically in loop processing within the normal
         * processing routine. Note that Ethernet transmission and reception may not operate
         * correctly, nor Ethernet driver may enter Magic Packet detection mode correctly if
         * this function is not called.
         */
        link_detect[channel] = ETHER_FLAG_OFF; /* Initial value ... Undetection */
        R_ETHER_LinkProcess(channel);

        switch (status)
        {
            case WAIT_LINK_ON:
                if (ETHER_FLAG_ON_LINK_ON == link_detect[channel])
                {
                    /*
                     * As the time to start data communication after the link is up is
                     * different between the host and remote host, the remote host may not be
                     * able to receive data if the host successfully transmits data.
                     */
                    /*
                     * The host verifies an ACK from the remote host to make sure that the
                     * remote host has received the data. After verifying that the host
                     * received an ACK from the remote host, it transmits an ACK to the remote
                     * host to synchronize mutually.
                     */
                    status = DATA_SEND;
                }
            }
        }
    }
}
```

```

        break;

    case DATA_SEND:
        ret = R_ETHER_Write(channel, (void *)send_data, sizeof(send_data));
        if (ETHER_SUCCESS == ret)
        {
            status = ACK_RECEIVE;
        }
        break;

    case ACK_RECEIVE:
        /*
         * A function which received the pointer to the read_buffer copies data to the
         * memory area which is pointed by the poiter using the standard function memcpy.
         * However, it does not copy data beyond the area which is controlled
         * by the read_buffer[].
         * Thus, cast from type "uint8_t *" to "void *" can be safely done.
         */
        ret = R_ETHER_Read(channel, (void *)read_buffer);
        /* When there is data to receive */
        if (ret > ETHER_NO_DATA)
        {
            /* When the received data size from the remote host matches the expected
             data size */
            if (sizeof(ack_data) == ret)
            {
                compare_result = data_compare(ack_data, read_buffer, (uint16_t)ret);
                /* When the receive data matches the transmit data */
                if (DATA_COMPARE_OK == compare_result)
                {
                    break;
                }
            }
        }
        break;

    default:
        break;
    }
}

R_ETHER_Close_ZC2(channel);

} /* End of function tx_rx_sample() */

/*****
*****
* Function Name: tx_rx_sample_zero_copy
* Description  : Program example of transmission/reception at the host
* Arguments    : channel -
*                Ethernet channel number
* Return Value : none
*****
*****/

```

```
static void tx_rx_sample_zero_copy(uint32_t channel)
{
    enum { WAIT_LINK_ON, DATA_SEND, ACK_RECEIVE };
    int32_t      ret;

    uint8_t      * pwrite_buffer_address;
    uint8_t      * pread_buffer_address;
    uint16_t      buf_size;
    uint8_t      status;
    int8_t      compare_result;

    ether_param_t    param;
    ether_cb_t      cb_func;

    /* ---- Disable maskable interrupts ---- */
    R_BSP_InterruptsDisable();

    /* Initialization of the Ethernet driver */
    R_ETHER_Initial();

    /* ---- Enable maskable interrupts ---- */
    R_BSP_InterruptsEnable();

    /* 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);

    /* ETHERC/EDMAC power on */
    param.channel = channel;
    ret = R_ETHER_Control(CONTROL_POWER_ON, param);

    /* Ether Open */
    ret = R_ETHER_Open_ZC2(channel, mac_addr_src, pause_enable);

    if (ETHER_SUCCESS == ret)
    {
        /* Start transmitting data when switch SW1 is pressed. */
        while (SW_ACTIVE != SW1)
        {
            /* Do Nothing */
        }

        status = WAIT_LINK_ON;

        while (1)
        {
            /*
             * The R_ETHER_LinkProcess function performs link signal change interrupt processing
             * and Magic Packet detection interrupt processing.
             * Make sure to call this function periodically in loop processing within the normal
            */
        }
    }
}
```



```
* processing routine.
* Note that Ethernet transmission and reception may not operate correctly, nor
* Ethernet driver may enter Magic Packet detection mode correctly if this function is
* not called.
*/
link_detect[channel] = ETHER_FLAG_OFF;
R_ETHER_LinkProcess(channel);

switch (status)
{
    case WAIT_LINK_ON:
        if (ETHER_FLAG_ON_LINK_ON == link_detect[channel])
        {
            /*
             * As the time to start data communication after the link is up is
             * different between the host and remote host, the remote host may not be
             * able to receive data if the host successfully transmits data.
             *
             * The host verifies an ACK from the remote host to make sure that the
             * remote host has received the data.
             * After verifying that the host received an ACK from the remote host, it
             * transmits an ACK to the remote host to synchronize mutually.
             */
            status = DATA_SEND;
        }
        break;

    case DATA_SEND:

        /*
         * As the "&write_buffer_address" does not access the memory area which is
         * pointed by the pointer to pointer in a destination, cast from type "uint8_t
         * *" to "void *" can be safely done.
         */
        ret = R_ETHER_Write_ZC2_GetBuf(channel, (void **)&write_buffer_address, &buf_size);
        if (ETHER_SUCCESS == ret)
        {
            /* Write the transmit data to the transmit buffer. */
            /*
             * Write 60 to 1514 bytes of data in the transmit buffer (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.
             */
            data_copy(write_buffer_address, send_data, sizeof(send_data));
            R_ETHER_Write_ZC2_SetBuf(channel, sizeof(send_data));

            /*
             * Confirm that the transmission is completed.
             * Data written in the transmit buffer is transmitted by the EDMAC. Make
             * sure that the transmission is completed after writing data to the
             * transmit buffer.
             * If the R_ETHER_Close_ZC2 function is called to stop the Ethernet
             * communication before verifying that the transmission is completed, the
             * written data written may not be transmitted.
             */
            ret = R_ETHER_CheckWrite(channel);
            if (ETHER_SUCCESS == ret )
            {
                status = WAIT_LINK_ON;
            }
        }
        else
        {
            status = WAIT_LINK_ON;
        }
    }
}
```

```

        {
            /* Update the counter for time-out not to cause the time-out
               processing. */

            status = ACK_RECEIVE;
        }
    }
    break;

case ACK_RECEIVE:

    /*
     * As the "&pread_buffer_address" does not access the memory area which is
     * pointed by the pointer to pointer in a destination, cast from type "uint8_t
     * *" to "void *" can be safely done.
     */
    ret = R_ETHER_Read_ZC2(channel, (void **)&pread_buffer_address);
    /* When there is data to receive */
    if (ret > ETHER_NO_DATA)
    {
        /* When the received data size from the remote host matches the expected
           data size */
        if (sizeof(ack_data) == ret)
        {
            compare_result = data_compare(ack_data, pread_buffer_address, (uint16_t)ret);
            /* When the receive data matches the transmit data */
            if (DATA_COMPARE_OK == compare_result)
            {
                goto TX_RX_SAMPLE_END;
            }
        }
        /* When the received data size from the remote host does not match the
           expected data size */

        /* Release the receive buffer after reading the receive data. */
        R_ETHER_Read_ZC2_BufRelease(channel);
    }
    /* When there is no data to receive */

    break;

default:
    break;
}
}

TX_RX_SAMPLE_END:
    R_ETHER_Close_ZC2(channel);

} /* End of function tx_rx_sample() */

/*****
*****
* Function Name: magic_packet_detection_sample
* Description  : Magic Packet detection program example at the host

```

```

* Arguments      : channel -
*                  Ethernet channel number
* Return Value   : none
*****
*****/
static void magic_packet_detection_sample(uint32_t channel)
{
    enum { CHANGE_MAGIC_PACKET_MODE, CHECK_MAGIC_PACKET_DETECT, DETECT_NOTIFY_SEND, ACK_RECEIVE };
    uint32_t      time;
    int32_t       ret;

    uint8_t       * pwrite_buffer_address;
    uint8_t       * pread_buffer_address;
    uint16_t      buf_size;
    uint8_t       status;
    int8_t        compare_result;

    ether_param_t  param;
    ether_cb_t     cb_func;

    /* ---- Disable maskable interrupts ---- */
    R_BSP_InterruptsDisable();

    /* Initialization of the Ethernet driver */
    R_ETHER_Initial();

    /* ---- Enable maskable interrupts ---- */
    R_BSP_InterruptsEnable();

    /* 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);

    /* ETHERC/EDMAC power on */
    param.channel = channel;
    ret = R_ETHER_Control(CONTROL_POWER_ON, param);

    /* Ether Open */
    ret = R_ETHER_Open_ZC2(channel, mac_addr_src, pause_enable);
    if (ETHER_SUCCESS == ret)
    {
        /* Clear the Magic Packet detection flag. */
        magic_packet_detect[channel] = 0;

        /*
         * As the host notified the remote host that the host had detected a Magic Packet,
         * the Ethernet driver enters normal communication mode and performs auto-negotiation
         * again.
         * As the time to start data communication after the link is up is different between the

```

```
* host and remote host, the remote host may not be able to receive data if the host
* successfully transmits data.
*
* The host detects an ACK from the remote host to know that the remote host recognized
* that the host had detected a Magic Packet.
*/
status = CHANGE_MAGIC_PACKET_MODE;
while (1)
{
    /*
    * The R_ETHER_LinkProcess function performs link signal change interrupt processing
    * and Magic Packet detection interrupt processing.
    * Make sure to call this function periodically in loop processing within the normal
    * processing routine.
    * Note that Ethernet transmission and reception may not operate correctly, nor
    * Ethernet driver may enter Magic Packet detection mode correctly if this function is
    * not called.
    */
    link_detect[channel] = ETHER_FLAG_OFF;
    R_ETHER_LinkProcess(channel);

    switch (status)
    {
        case CHANGE_MAGIC_PACKET_MODE:
            /* Enter Magic Packet detection mode. */
            ret = R_ETHER_WakeOnLAN(channel);
            if (ETHER_SUCCESS == ret)
            {
                /* Enter sleep mode after verifying that the link status has not been
                changed. */
                if (ETHER_FLAG_OFF == link_detect[channel])
                {

                    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);

                    /* Turn on LED0 to indicate that the MCU enters sleep mode to await a
                    Magic Packet detection. */
                    LED0 = LED_ON;

                    wait();

                    status = CHECK_MAGIC_PACKET_DETECT;
                }
            }
            break;

        case CHECK_MAGIC_PACKET_DETECT:
            /*
            * The g_magic_packet_detect flag is set to 1 within the Magic Packet
            * detection interrupt callback function.
            */
            break;
    }
}
```

```

    * The MCU calles the R_ETHER_LinkProcess function is called after exiting
    * from sleep mode, and verifies the flag is set to 1 after the interrupt
    * processing is executed.
    */
    /* When a Magic Packet is detected */
    if (1 == magic_packet_detect[channel])
    {
        magic_packet_detect[channel] = 0;

        status = DETECT_NOTIFY_SEND;
        /* Initialize the timer for the reception time-out */
        cmt0_init();

        /* Turn on LED1 to indicate that the MCU exits sleep mode by detecting a
        Magic Packet. */
        LED1 = LED_ON;
    }
    /* When no Magic Packet is detected */
    else
    {
        status = CHANGE_MAGIC_PACKET_MODE;
        /* Turn on LED0 to indicate that the MCU enters Magic Packet detection
        mode again. */
        LED0 = LED_OFF;
    }
    break;

case DETECT_NOTIFY_SEND:

    /*
    * As the "&write_buffer_address" does not access the memory area which is
    * pointed by the pointer to pointer in a destination, cast from type "uint8_t
    * *" to "void *" can be safely done.
    */
    ret = R_ETHER_Write_ZC2_GetBuf(channel, (void **)&write_buffer_address, &buf_size);
    if (ETHER_SUCCESS == ret)
    {
        /* Write data to the transmit buffer to notify that a Magic Packet is
        detected. */
        /*
        * Write 60 to 1514 bytes of data in the transmit buffer (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.
        */
        data_copy(write_buffer_address, magic_packet_detect_notify,
        sizeof(magic_packet_detect_notify));
        R_ETHER_Write_ZC2_SetBuf(channel, sizeof(magic_packet_detect_notify));

        /*
        * Confirm that the transmission is completed.
        * Data written in the transmit buffer is transmitted by the EDMAC. Make
        * sure that the transmission is completed after writing data to the
        * transmit buffer.
        * If the R_ETHER_Close_ZC2 function is called to stop the Ethernet
        * communication before verifying that the transmission is completed, the
        * written data written may not be transmitted.
        */
    }

```

```

        ret = R_ETHER_CheckWrite(channel);
        if (ETHER_SUCCESS == ret )
        {
            status = ACK_RECEIVE;
            /* Update the counter for time-out not to cause the time-out processing. */
            time    = cmt0_cmi0_count;
        }
    }

break;

case ACK_RECEIVE:

    /*
     * As the "&pread_buffer_address" does not access the memory area which is
     * pointed by the pointer to pointer in a destination, cast from type "uint8_t
     * *" to "void *" can be safely done.
     */
    ret = R_ETHER_Read_ZC2(channel, (void **)&pread_buffer_address);
    /* When there is data to receive */
    if (ret > ETHER_NO_DATA)
    {
        /* When the receive data size matches the ACK size to notify a Magic
         Packet is detected */
        if (sizeof(magic_packet_detect_ack) == ret)
        {
            /* Compare the data. */
            /*
             * As the "ret" value defined by type int32_t is between 0 and 0xFFFF,
             * cast to type uint16_t can be safely done.
             */
            compare_result = data_compare(magic_packet_detect_ack,
            pread_buffer_address, (uint16_t)ret);
            /* When the receive data matches the ACK to notify that a Magic Packet
             is detected */
            if (DATA_COMPARE_OK == compare_result)
            {
                /* Turn on LED2 to indicate that the host received an ACK that
                 responds to the Magic Packet detection notice from the host. */
                LED2 = LED_ON;
            }
        }

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

        goto MAGIC_PACKET_DETECT_END;
    }

    /* When there is no data to receive */
    else
    {
        /* When the reception time-out error occurs, retransmit the data that
         notified a Magic Packet detection. */
        if ((time + NO_RECEIVE_TIMEOUT_MS) < cmt0_cmi0_count)
        {
            status = DETECT_NOTIFY_SEND;
        }
    }
}

```

```

        }
        break;

        default:
        break;
    }
}
}

MAGIC_PACKET_DETECT_END:
    /* Turn on LED3 to indicate that a program to detect a Magic Packet detection terminated. */
    LED3 = LED_ON;

    R_ETHER_Close_ZC2(channel);

} /* End of function magic_packet_detection_sample() */

/*****
*****
* Function Name: callback_sample
* Description   : Sample of the callback function
* Arguments    : pparam -
*
* Return Value : none
*****
*****/
static void callback_sample(void * pparam)
{
    ether_cb_arg_t    * pdecode;
    uint32_t          channel;

    pdecode = (ether_cb_arg_t *)pparam;
    channel = pdecode->channel;                /* Get Ethernet channel number */

    switch (pdecode->event_id)
    {
        /* Callback function that notifies user to have detected magic packet. */
        case ETHER_CB_EVENT_ID_WAKEON_LAN:
            callback_wakeon_lan(channel);
            break;

        /* Callback function that notifies user to have become Link up. */
        case ETHER_CB_EVENT_ID_LINK_ON:
            callback_link_on(channel);
            break;

        /* Callback function that notifies user to have become Link down. */
        case ETHER_CB_EVENT_ID_LINK_OFF:
            callback_link_off(channel);
            break;

        default:
            break;
    }
} /* End of function callback_sample() */

```

```

/*****
*****
* Function Name: callback_wakeon_lan
* Description   :
* Arguments    : channel -
*               Ethernet channel number
* Return Value : none
*****
*****/
static void callback_wakeon_lan(uint32_t channel)
{
    if (ETHER_CHANNEL_MAX > channel)
    {
        magic_packet_detect[channel] = 1;

        /* Please add necessary processing when magic packet is detected. */
    }
} /* End of function callback_wakeon_lan() */

/*****
*****
* Function Name: callback_link_on
* Description   :
* Arguments    : channel -
*               Ethernet channel number
* Return Value : none
*****
*****/
static void callback_link_on(uint32_t channel)
{
    if (ETHER_CHANNEL_MAX > channel)
    {
        link_detect[channel] = ETHER_FLAG_ON_LINK_ON;

        /* Please add necessary processing when becoming Link up. */
    }
} /* End of function callback_link_on() */

/*****
*****
* Function Name: callback_link_off
* Description   :
* Arguments    : channel -
*               Ethernet channel number
* Return Value : none
*****
*****/
static void callback_link_off(uint32_t channel)
{
    if (ETHER_CHANNEL_MAX > channel)
    {
        link_detect[channel] = ETHER_FLAG_ON_LINK_OFF;

        /* Please add necessary processing when becoming Link down. */
    }
} /* End of function ether_cb_link_off() */

/*****
*****/

```



```

*****
* Function Name: int_handler_sample
* Description   :
* Arguments    : channel -
*               Ethernet channel number
* Return Value : none
*****
*****/
static void int_handler_sample(void *pparam)
{
    ether_cb_arg_t    * pdecode;

    pdecode = (ether_cb_arg_t *)pparam;
    if (ETHER_CHANNEL_0 == pdecode->channel)
    {
        int_handler_0_sample(pdecode->status_ecsr, pdecode->status_eesr);
    }
    else if (ETHER_CHANNEL_1 == pdecode->channel)
    {
        int_handler_1_sample(pdecode->status_ecsr, pdecode->status_eesr);
    }
    else
    {
        /* Do Nothing */
    }
} /* End of function int_handler_sample() */

/*****
*****
* Function Name: int_handler_0_sample
* Description   :
* Arguments    : status_ecsr -
*               status_eesr -
*               status_eesr -
*               status_eesr -
* Return Value : none
*****
*****/
static void int_handler_0_sample(uint32_t status_ecsr, uint32_t status_eesr)
{
    g_status_ecsr[ETHER_CHANNEL_0] = status_ecsr;
    g_status_eesr[ETHER_CHANNEL_0] = status_eesr;

} /* End of function int_handler_0_sample() */

/*****
*****
* Function Name: int_handler_1_sample
* Description   :
* Arguments    : status_ecsr -
*               status_eesr -
*               status_eesr -
*               status_eesr -
* Return Value : none
*****
*****/
static void int_handler_1_sample(uint32_t status_ecsr, uint32_t status_eesr)
{

```

```

    g_status_ecsr[ETHER_CHANNEL_1] = status_ecsr;
    g_status_eesr[ETHER_CHANNEL_1] = status_eesr;

} /* End of function int_handler_1_sample() */

/*****
*****
* Function Name: data_copy
* Description   : Data copy function
* Arguments    : pdst -
*                Data copy destination pointer
*                psrc -
*                Data copy source pointer
*                len -
*                Data length to copy
* Return Value : none
*****
*****/
static void data_copy(uint8_t * pdst, const uint8_t * psrc, const uint16_t len)
{
    uint16_t    j;

    for (j = 0; j < len; j++)
    {
        *pdst = *psrc;
        pdst++;
        psrc++;
    }
} /* End of function data_copy() */

/*****
*****
* Function Name: data_compare
* Description   : Data compare function
*                The data_compare function compares the number of data indicated by len using
pointers dst and src.
* Arguments    : pdst -
*                Data compare source pointer
*                psrc -
*                Data compare destination pointer
*                len -
*                Data length to compare
* Return Value : DATA_COMPARE_OK -
*                Comparison result is match
*                DATA_COMPARE_NG -
*                Comparison result is mismatch
*****
*****/
static int8_t data_compare(const uint8_t * pdst, const uint8_t * psrc, const uint16_t len)
{
    uint16_t    j;

    for (j = 0; j < len; j++)
    {
        if (*psrc != *pdst)
        {
            return DATA_COMPARE_NG;
        }
    }
}

```

```

        psrc++;
        pdst++;
    }
    return DATA_COMPARE_OK;
} /* End of function data_compare() */

/*****
*****
* Function Name: cmt0_init
* Description   : Initialize the timer.
* Arguments    : none
* Return Value  : none
* Limitations   : This sample program assumes the timer operating frequency as 48 MHz.
                  Changing the operating frequency affects the assumed timer cycle (1 ms), make
sure to review
                  the settings when changing the operating frequency.
*****
*****/
STATIC void cmt0_init(void)
{
    SYSTEM.PRCR.WORD = 0xA502; /* protect off */

    /* Enable compare match timer 0. */
    MSTP(CMT0) = 0;

    SYSTEM.PRCR.WORD = 0xA500; /* protect on */

    /* Interrupt on compare match. */
    CMT0.CMCR.BIT.CMIE = 1;

    /* Set the compare match value. */
    CMT0.CMCOR = ((40000000L / 1000) - 1) / 8; /* 1ms@40MHz */
    // CMT0.CMCOR = ((48000000L / 1000) - 1) / 8; /* 1ms@48MHz */
    // CMT0.CMCOR = ((60000000L / 1000) - 1) / 8; /* 1ms@60MHz */

    /* Divide the PCLK by 8. */
    CMT0.CMCR.BIT.CKS = 0;

    /* Enable the interrupt... */
    _IEN(_CMT0_CMI0) = 1;

    /* ...and set its priority to the application defined kernel priority. */
    _IPR(_CMT0_CMI0) = 1;

    /* Start the timer. */
    CMT0.CMSTR0.BIT.STR0 = 1;
} /* End of function cmt0_init() */

/*****
*****
* Function Name: cmt0_cmi0_isr
* Description   : 1-ms timer interrupt processing
* Arguments    : none
* Return Value  : none
*****
*****/
static void cmt0_cmi0_isr(void)

```

```
{  
    cmt0_cmi0_count++;  
  
#ifdef  ETHER_DRIVER_TEST  
    if (0 == (cmt0_cmi0_count & 0x00000001))  
    {  
        LED3 = LED_ON;  
    }  
    else  
    {  
        LED3 = LED_OFF;  
    }  
#endif  /*ETHER_DRIVER_TEST*/  
  
} /* End of function cmt0_cmi0_isr() */  
  
/* End of file */
```

## 5. Provided Modules

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

## 6. Reference Documents

User's Manual: Hardware

RX64M Group User's Manual: Hardware Rev.1.00 ( )

(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 Package V.1.01 User's Manual Rev.1.00 (R20UT0570EJ010)

(The latest version can be downloaded from the Renesas Electronics website.)

**Website and Support**

Renesas Electronics Website

<http://www.renesas.com/>

Inquiries

<http://www.renesas.com/contact/>

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

**Revision History**

Rev.	Date	Description	
		Page	Summary
1.00	Jul 18, 2014	—	First edition issued

## General Precautions in the Handling of MPU/MCU Products

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

### 1. Handling of Unused Pins

Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

- The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

### 2. Processing at Power-on

The state of the product is undefined at the moment when power is supplied.

- The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.

In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.

In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

### 3. Prohibition of Access to Reserved Addresses

Access to reserved addresses is prohibited.

- The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

### 4. Clock Signals

After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

- When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

### 5. Differences between Products

Before changing from one product to another, i.e. to a product with a different type number, confirm that the change will not lead to problems.

- The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the 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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
  2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
  3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
  4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
  5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended 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; and industrial robots etc.

"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

Renesas Electronics products are neither intended nor 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 damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
  6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
  7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
  8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
  9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
  10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
  11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
  12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.
- (Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.
- (Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.



### SALES OFFICES

### Renesas Electronics Corporation

<http://www.renesas.com>

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

#### **Renesas Electronics America Inc.**

2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.  
Tel: +1-408-588-6000, Fax: +1-408-588-6130

#### **Renesas Electronics Canada Limited**

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

#### **Renesas Electronics Europe Limited**

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

#### **Renesas Electronics Europe GmbH**

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

#### **Renesas Electronics (China) Co., Ltd.**

Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China  
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

#### **Renesas Electronics (Shanghai) Co., Ltd.**

Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333  
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

#### **Renesas Electronics Hong Kong Limited**

Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong  
Tel: +852-2265-6688, Fax: +852 2886-9022/9044

#### **Renesas Electronics Taiwan Co., Ltd.**

13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan  
Tel: +886-2-8175-9500, Fax: +886 2-8175-9670

#### **Renesas Electronics Singapore Pte. Ltd.**

80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949  
Tel: +65-6213-0200, Fax: +65-6213-0300

#### **Renesas Electronics Malaysia Sdn.Bhd.**

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

#### **Renesas Electronics Korea Co., Ltd.**

12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea  
Tel: +82-2-558-3737, Fax: +82-2-558-5141