

# RX ファミリ

イーサネットモジュール  
Firmware Integration Technology

R01AN2009JJ0100  
Rev.1.00  
2014.07.29

## 要旨

本アプリケーションノートは、Firmware Integration Technology (FIT) を使用したイーサネットモジュールについて説明します。本モジュールはイーサネットコントローラ、イーサネットコントローラ用 DMA コントローラを使用して、イーサネット / IEEE802.3 フレームの送受信を行います。以降、本モジュールをイーサネット FIT モジュールと称します。

## 対象デバイス

以下は、この API によってサポートできるデバイスの一覧です。

RX64M

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

## 目次

1. 概要 .....	2
2. API 情報.....	3
3. API 関数.....	10
4. 付録 .....	31
5. 提供するモジュール.....	52
6. 参考ドキュメント.....	52

## 1. 概要

イーサネット FIT モジュールは、イーサネットコントローラ（以降、ETHERC と呼称）とイーサネットコントローラ用 DMA コントローラ（以降、EDMAC と呼称）を使用し、イーサネット / IEEE802.3 フレームの送受信を行うための手段を提供します。以下に本モジュールがサポートしている機能を列挙します。

- MII（Media Independent Interface）および RMII（Reduced Media Independent Interface）に対応しています。
- イーサネット PHY-LSI のリンクには、自動交渉機能を用います。
- イーサネット PHY-LSI から出力されるリンク信号を用いて、リンク状態を検出します。
- イーサネット PHY-LSI からの自動交渉結果を取得し、接続モード（全二重モードまたは半二重モード、転送速度 10Mbps または 100Mbps）を ETHERC に設定します。

### 制限事項

- 使用するチャンネルに応じて、ET0\_LINKSTA 端子、もしくは ET1\_LINKSTA 端子に PHY-LSI から出力されるリンク信号を接続する必要があります。

## 1.1 イーサネット FIT モジュールとは

本モジュールは API として、プロジェクトに組み込んで使用します。本モジュールの組み込み方については、「2.10 モジュールの追加方法」を参照してください。

## 1.2 API の概要

表 1.1 に本モジュールに含まれる API 関数を示します。

表1.1 API 関数一覧

関数	関数説明
R_ETHER_Initial()	イーサネットドライバの初期化を行います。
R_ETHER_Open_ZC2()	ETHERC と EDMAC および PHY-LSI をソフトウェアリセットした後、PHY-LSI のオートネゴシエーションを開始してリンク信号変化割り込みを許可します。
R_ETHER_Close_ZC2()	ETHERC の送信、受信機能をディゼーブル状態とします。ETHERC、EDMAC をモジュールストップにしません。
R_ETHER_Read()	指定した受信バッファへデータを受信します。
R_ETHER_Read_ZC2()	受信データが格納されたバッファの先頭アドレスへのポインタを返します。
R_ETHER_Read_ZC2_BufRelease()	R_ETHER_Read_ZC2 関数で読み出したバッファを開放します。
R_ETHER_Write()	指定した送信バッファからデータを送信します。
R_ETHER_Write_ZC2_GetBuf()	送信データの書き込み先の先頭アドレスへのポインタが返されます。
R_ETHER_Write_ZC2_SetBuf()	EDMAC に送信バッファのデータの送信を許可します。
R_ETHER_CheckLink_ZC()	物理的なイーサネットのリンク状態を、PHY 管理インタフェースを使用してチェックします。PHY が適切に初期化されている相手デバイスとケーブルが接続されていれば、イーサネットのリンク状態がリンクアップとなります。
R_ETHER_LinkProcess()	リンク信号変化割り込み処理およびマジックパケット検出割り込み処理を行います。
R_ETHER_WakeOnLAN()	ETHERC の設定を通常の送受信動作からマジックパケット検出動作に切り替えます。
R_ETHER_CheckWrite()	データ送信が完了したことを確認します。
R_ETHER_Control()	コントロールコードに対応した処理を行います。
R_ETHER_GetVersion()	本モジュールのバージョン番号を返します。

---

## 2. API 情報

本モジュールの API はルネサスの API の命名基準に従っています。

---

### 2.1 ハードウェアの要求

---

ご使用になる MCU が以下の機能をサポートしている必要があります。

- ETHERC
- EDMAC

---

### 2.2 ソフトウェアの要求

---

このドライバは以下のパッケージに依存しています。

- Renesas Board Support Package (r\_bsp) v2.60 or higher

---

### 2.3 サポートされているツールチェーン

---

このドライバは下記ツールチェーンで動作確認を行っています。

- Renesas RX Toolchain v.2.01.00

---

### 2.4 ヘッドファイル

---

すべての API 呼び出しと使用されるインタフェース定義は `r_ether_rx_if.h` に記載しています。

---

### 2.5 整数型

---

このプロジェクトは ANSI C99 を使用しています。これらの型は `stdint.h` で定義されています。

## 2.6 コンパイル時の設定

本モジュールのコンフィギュレーションオプションの設定は、`r_ether_rx_config.h`で行います。

オプション名および設定値に関する説明を、下表に示します。

Configuration options in <i>r_ether_rx_config.h</i>	
#define ETHER_CFG_MODE_SEL 【注】デフォルト値は “0”	ETHERC とイーサネット PHY-LSI 間のインタフェースを設定してください。 “0” の場合、MII(Media Independent Interface)を選択します。 “1” の場合、RMII ( Reduced Media Independent Interface ) を選択します。
#define ETHER_CFG_CH0_PHY_ADDRESS 【注】デフォルト値は “0”	ETHERC チャンネル 0 が使用する PHY-LSI に割り当てられた PHY アドレスを設定してください。 “0” ~ “15” の範囲で設定してください。
#define ETHER_CFG_CH1_PHY_ADDRESS 【注】デフォルト値は “1”	ETHERC チャンネル 1 が使用する PHY-LSI に割り当てられた PHY アドレスを設定してください。 “0” ~ “15” の範囲で設定してください。
#define ETHER_CFG_EMAC_RX_DESCRIPTOR 【注】デフォルト値は “1”	受信ディスクリプタの数を設定してください。 “1” 以上の値を設定してください。
#define ETHER_CFG_EMAC_TX_DESCRIPTOR 【注】デフォルト値は “1”	送信ディスクリプタの数を設定してください。 “1” 以上の値を設定してください。
#define ETHER_CFG_BUFSIZE 【注】デフォルト値は “1536”	送信バッファ、受信バッファのサイズを設定してください。 バッファは 32 バイト境界で配置しますので、32 バイト単位の値を設定してください。
#define ETHER_CFG_AL1_INT_PRIORITY 【注】デフォルト値は “2”	グループ AL1 割り込みの優先レベルを設定してください。 “1” ~ “15” の範囲で設定してください。
#define ETHER_CFG_CH0_PHY_ACCESS 【注】デフォルト値は “1” * <sup>1</sup>	ETHERC チャンネル 0 が使用する PHY のアクセスチャネルを設定してください。 “0” の場合、PHY のレジスタアクセスは ETHERC0 を使用します。* <sup>2</sup> “1” の場合、PHY のレジスタアクセスは ETHERC1 を使用します。* <sup>3</sup>
#define ETHER_CFG_CH1_PHY_ACCESS 【注】デフォルト値は “1” * <sup>1</sup>	ETHERC チャンネル 1 が使用する PHY のアクセスチャネルを設定してください。 “0” の場合、PHY のレジスタアクセスは ETHERC0 を使用します。* <sup>2</sup> “1” の場合、PHY のレジスタアクセスは ETHERC1 を使用します。* <sup>3</sup>
#define ETHER_CFG_PHY_MII_WAIT 【注】デフォルト値は “8”	MII/RMII レジスタのアクセスタイミングを設定してください。 “8” 以上の値を設定してください。
#define ETHER_CFG_PHY_DELAY_RESET 【注】デフォルト値は “0x00020000”	PHY-LSI のリセット完了のウェイト時間を設定してください。

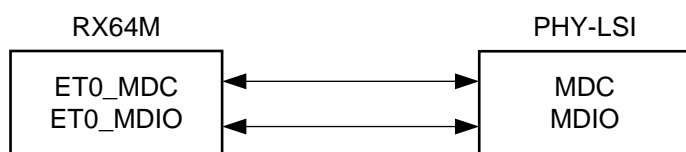
Configuration options in <i>r_ether_rx_config.h</i>	
#define ETHER_CFG_LINK_PRESENT 【注】デフォルト値は “0”	PHY-LSI から出力されるリンク信号の極性を設定してください。 “0” の場合、LINKSTA 信号の立ち下がり / 立ち上がりで、リンクアップ / リンクダウンとなります。 “1” の場合、LINKSTA 信号の立ち上がり / 立ち下がりで、リンクアップ / リンクダウンとなります。
#define ETHER_CFG_USE_PHY_KSZ8041NL 【注】デフォルト値は “0”	Micrel 社の PHY-LSI KSZ8041NL を使用するかどうかを設定してください。 “0” の場合、KSZ8041 を使用しません。 “1” の場合、KSZ8041 を使用します。

【注】 \*1 Renesas Starter Kit+ for RX64M( 製品型名 : R0K50564MSxxxBE )上でイーサネット FIT モジュールを 動かす場合の設定は表 2.1を参照ください。

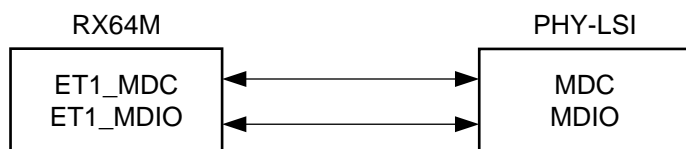
表2.1 ETHER\_CFG\_CH0\_PHY\_ACCESS / ETHER\_CFG\_CH1\_PHY\_ACCESS 設定

ショートピン J3	ショートピン J4	ETHER_CFG_CH0_PHY_ACCESS ETHER_CFG_CH1_PHY_ACCESS の設定値
1-2 間ショート	1-2 間ショート	0
		0
2-3 間ショート	2-3 間ショート	1
		1

\*2 ETHERC と PHY-LSI の接続が下記の場合の設定です。



\*3 ETHERC と PHY-LSI の接続が下記の場合の設定です。



## 2.7 引数

API 関数の引数である列挙体、共用体、構造体を示します。これらは API 関数のプロトタイプ宣言とともに `r_ether_rx_if.h` で記載されています。

```
typedef enum
{
    CONTROL_SET_CALLBACK,                /* コールバック関数の登録 */
    CONTROL_SET_PROMISCUOUS_MODE,        /* プロミスカスモード設定 */
    CONTROL_SET_INT_HANDLER,             /* 割り込みハンドラ関数の登録 */
    CONTROL_POWER_ON,                   /* ETHERC/EDMAC モジュールストップ解除 */
    CONTROL_POWER_OFF                   /* ETHERC/EDMAC モジュールストップ遷移 */
} ether_cmd_t;

typedef union
{
    ether_cb_t          ether_callback;   /* コールバック関数ポインタ */
    ether_promiscuous_t * p_ether_promiscuous;
    ether_cb_t          ether_int_hnd;    /* 割り込みハンドラ関数ポインタ */
    uint32_t            channel;          /* ETHERC のチャンネル番号 */
} ether_param_t;

typedef struct
{
    void    (*pcb_func)(void *);          /* コールバック関数ポインタ */
    void    (*pcb_int_hnd)(void *);       /* 割り込みハンドラ関数ポインタ */
} ether_cb_t;

typedef enum
{
    ETHER_PROMISCUOUS_OFF,                /* ETHERC は標準モード */
    ETHER_PROMISCUOUS_ON                 /* ETHERC はプロミスカスモード */
} ether_promiscuous_bit_t;

typedef struct
{
    uint32_t            channel;          /* ETHERC チャンネル */
    ether_promiscuous_bit_t bit;          /* プロミスカスモード */
} ether_promiscuous_t;

typedef enum
{
    ETHER_CB_EVENT_ID_WAKEON_LAN,         /* マジックパケット検出 */
    ETHER_CB_EVENT_ID_LINK_ON,           /* Link up 検出 */
    ETHER_CB_EVENT_ID_LINK_OFF           /* Link down 検出 */
} ether_cb_event_t;

typedef struct
{
    uint32_t            channel;          /* ETHERC チャンネル */
    ether_cb_event_t    event_id;         /* コールバック関数用イベントコード */
    uint32_t            status_ecsr;      /* 割り込みハンドラ関数用 ETHERC ステータスレジスタ */
    uint32_t            status_eesr;      /* 割り込みハンドラ関数用 */
                                           /* ETHERC/EDMAC ステータスレジスタ */
} ether_cb_arg_t;
```

## 2.8 戻り値

API 関数の戻り値を示します。この列挙型は API 関数のプロトタイプ宣言とともに `r_ether_rx_if.h` で記載されています。

```
typedef enum                                /* Ether API のエラーコード */
{
    ETHER_SUCCESS,                          /* 問題なく処理が終了した場合 */
    ETHER_ERR_INVALID_PTR,                  /* ポインタの値が、NULL もしくは FIT_NO_PTR の場合 */
    ETHER_ERR_INVALID_DATA,                 /* 引数のとり得る値が、範囲外の場合 */
    ETHER_ERR_INVALID_CHAN,                 /* 存在しないチャネルの場合 */
    ETHER_ERR_INVALID_ARG,                  /* 不正な引数の場合 */
    ETHER_ERR_LINK,                         /* オートネゴシエーション処理が完了しておらず受信が */
                                           /* 許可されていない場合 */
    ETHER_ERR_MPDE,                         /* マジックパケットの検出状態のため、 */
                                           /* 送信と受信が許可されていない場合 */
    ETHER_ERR_TACT,                         /* 送信バッファに空きがない場合 */
    ETHER_ERR_CHAN_OPEN,                    /* 他のアプリケーションが使用しているため */
                                           /* Ether を Open できない場合 */
    ETHER_ERR_OTHER                         /* その他エラー */
} ether_return_t;
```

## 2.9 コールバック関数

### (1) API 関数 R\_ETHER\_LinkProcess から呼び出すコールバック関数

本モジュールでは、マジックパケットの検出、または、リンク信号変化の検出があったとき、コールバック関数を呼び出します。

コールバック関数の設定は、後述の関数 R\_ETHER\_Control を用いて、「2.7 引数」に記載の列挙体（第 1 引数）には、コントロールコード“CONTROL\_SET\_CALLBACK”を、構造体（第 2 引数）には、コールバック関数として登録したい関数のアドレスを設定して下さい。

コールバック関数が呼び出されるとき、検出があったチャンネル番号と表 2.2 に示す定数を格納した変数を、引数として渡します。引数の値をコールバック関数外で使用する場合は、グローバル変数などの変数にコピーしてください。

表2.2 コールバック関数の引数一覧

定数定義	意味
ETHER_CB_EVENT_ID_WAKEON_LAN	マジックパケットを検出した
ETHER_CB_EVENT_ID_LINK_ON	リンク信号変化（リンクアップ）を検出した
ETHER_CB_EVENT_ID_LINK_OFF	リンク信号変化（リンクダウン）を検出した

### (2) EINT0/EINT1 ステータス割り込みから呼び出す割り込みハンドラ関数

本モジュールでは、以下に示した内容以外の割り込みがあったとき、割り込みハンドラ関数を呼び出します。

- マジックパケットの検出
- リンク信号変化の検出

割り込みハンドラ関数の設定は、後述の関数 R\_ETHER\_Control を用いて、「2.7 引数」に記載の列挙体（第 1 引数）には、コントロールコード“CONTROL\_SET\_INT\_HANDLER”を、構造体（第 2 引数）には、割り込みハンドラ関数として登録したい関数のアドレスを設定して下さい。

割り込みハンドラ関数が呼び出されるとき、割り込みがあったチャンネル番号と ETHERC ステータスレジスタの値、ETHERC/EDMAC ステータスレジスタの値を格納した変数を、引数として渡します。引数の値をコールバック関数以外で使用する場合は、グローバル変数などの変数にコピーしてください。



## 2.10 モジュールの追加方法

本モジュールは、e<sup>2</sup> studio で、使用するプロジェクトごとに追加する必要があります。

プロジェクトへの追加方法は、FIT プラグインを使用する方法と、手動で追加する方法があります。

FIT プラグインを使用すると、簡単にプロジェクトに FIT モジュールを追加でき、また、自動的にインクルードファイルパスが更新されます。このため、プロジェクトへ FIT モジュールを追加する際は、FIT プラグインの使用を推奨します。

FIT プラグインを使用して FIT モジュールを追加する方法は、アプリケーションノート「e<sup>2</sup> studio に組み込む方法 Firmware Integration Technology (R01AN1723JU)」の「3. FIT プラグインを使用して FIT モジュールをプロジェクトに追加する方法」を参照してください。

FIT プラグインを使用せず手動で FIT モジュールを追加する方法は、以下の「2.10.1 イーサネット FIT モジュールの追加手順（手動で追加する場合）」を参照してください。

FIT モジュールを使用する場合、BSP FIT モジュールもプロジェクトに追加する必要があります。

BSP FIT モジュールの詳細については、アプリケーションノート「ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685JU)」を参照してください。

### 2.10.1 イーサネット FIT モジュールの追加手順（手動で追加する場合）

1. アプリケーションノートは、イーサネット FIT モジュールを含む ZIP ファイルパッケージにある r\_ether\_rx フォルダと一緒に配布されます。
2. 任意の場所に ZIP ファイルを解凍します。
3. ファイルブラウザウィンドウから、解凍したディレクトリを参照し、r\_ether\_rx フォルダを参照しておきます。
4. e<sup>2</sup> studio のワークスペースを立ち上げます。
5. e<sup>2</sup> studio のエクスプローラウィンドウで、本モジュールを追加するプロジェクトを選択します。
6. ブラウザウィンドウから r\_ether\_rx フォルダを選択した e<sup>2</sup> studio のプロジェクトの最上位層にドラッグ＆ドロップします。
7. モジュールファイルへパスを追加することによって、ソース探索/インクルードパスの更新を行います。
  - a. プロジェクトを右クリックし、プロパティ(R)を選択します。
  - b. 「C/C++ ビルド>>設定>> Compiler の下のソース」と選択します。
  - c. インクルードファイルディレクトリの追加ボタンをクリックし“ディレクトリパスの追加”ウィンドウが表示します。
  - d. ディレクトリパスに

```
"${workspace_loc}/${ProjName}/r_ether_rx"
"${workspace_loc}/${ProjName}/r_ether_rx/src"
```

を追加し、OK をクリックします。
  - e. 追加したディレクトリパスが一覧に表示されているのを確認し、適用(L)をクリックし、OK をクリックします。
8. r\_ether\_rx/ref/targets/rx64m ソースフォルダ内の r\_ether\_rx\_config\_reference.h を、プロジェクト内の r\_config フォルダにコピーします。
9. r\_ether\_rx\_config\_reference.h のファイル名を r\_ether\_config.h に修正します。
10. 必要に応じて、r\_ether\_rx\_config.h ファイルの設定を変更します。設定は「2.6 コンパイル時の設定」を参照してください。

### 3. API 関数

---

#### 3.1 R\_ETHER\_Initial()

---

イーサネット FIT モジュールの初期設定を行う関数です。

**Format**

void R\_ETHER\_Initial(void);

**Parameters**

なし

**Return Values**

なし

**Properties**

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

**Description**

イーサネット通信を開始するため、使用するメモリの初期化を行います。

**Reentrant**

- 不可

**Example**

「4.2 サンプルコード」を参照してください。

**Special Notes:**

R\_ETHER\_Open 関数よりも前で呼び出してください。

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

ETHER の API を使用する際に、最初に使用する関数です。

#### Format

```
ether_return_t R_ETHER_Open_ZC2(  
    uint32_t      channel      /* ETHERC のチャンネル番号 */  
    const uint8_t mac_addr[]   /* ETHERC の MAC アドレス */  
    uint8_t      pause        /* フロー制御機能の有効/無効 */  
);
```

#### Parameters

*channel*

ETHERC のチャンネル番号 (0、1) を指定します。

*mac\_addr*

ETHERC の MAC アドレスを指定します。

*pause*

PHY-LSI のレジスタ 4 (Auto-Negotiation Advertisement) のビット 10 (Pasuse) に設定する値を指定します。ユーザが使用する PHY-LSI が Pause 機能に対応している場合のみ ETHER\_FLAG\_ON の指定が可能です。この値はオートネゴシエーション時に相手側の PHY-LSI に引き渡されます。オートネゴシエーションの結果、自分の PHY-LSI と相手側の PHY-LSI の両方が Pasuse 機能に対応している場合はフロー制御が有効となります。

Pasuse 機能に対応していることをオートネゴシエーション時に相手側の PHY-LSI に伝達したい場合は、ETHER\_FLAG\_ON を、Pause 機能対応していない場合または対応していても使わない場合は、ETHER\_FLAG\_OFF を指定してください。

#### Return Values

ETHER_SUCCESS	/* 問題なく処理が完了した場合 */
ETHER_ERR_INVALID_CHAN	/* 存在しないチャンネルの場合 */
ETHER_ERR_INVALID_PTR	/* ポインタの値が、NULL もしくは FIT_NO_PTR の場合 */
ETHER_ERR_INVALID_DATA	/* 引数のとり得る値が、範囲外の場合 */
ETHER_ERR_OTHER	/* PHY-LSI の初期化に失敗した場合 */

#### Properties

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

#### Description

R\_ETHER\_Open\_ZC2 関数は ETHERC と EDMAC および PHY-LSI をソフトウェアリセットした後、PHY-LSI のオートネゴシエーションを開始し、リンク信号変化割り込みを許可します。

MAC アドレスは ETHERC の MAC アドレスレジスタを初期化するために使用されます。

#### Reentrant

- 異なるチャンネルからリエントラントは可能です。

#### Example

「4.2 サンプルコード」を参照してください。

#### Special Notes:

パワーオンリセット後に R\_ETHER\_Initial 関数を実行した後、および R\_ETHER\_Close\_ZC2 関数を実行した後は、必ず本関数を実行して戻り値が ETHER\_SUCCESS であることを確認した後、他の API をご使用ください。

---

### 3.3 R\_ETHER\_Close\_ZC2()

---

R\_ETHER\_Close\_ZC2関数はETHERCの送信、受信機能をディゼーブル状態にします。この関数はETHERC、EDMACをモジュールストップにしません。

**Format**

```
ether_return_t R_ETHER_Close_ZC2(  
    uint32_t      channel          /* ETHERC のチャンネル番号 */  
);
```

**Parameters**

*channel*

ETHERC のチャンネル番号 (0、1) を指定します。

**Return Values**

<i>ETHER_SUCCESS</i>	<i>/* 問題なく処理が完了した場合 */</i>
<i>ETHER_ERR_INVALID_CHAN</i>	<i>/* 存在しないチャンネルの場合 */</i>

**Properties**

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

**Description**

R\_ETHER\_Close\_ZC2 関数は ETHERC の送信、受信機能およびイーサネット割り込みをディゼーブル状態にします。ETHERC、EDMAC をモジュールストップにしません。  
本関数はイーサネット通信を終了する場合に実行してください。

**Reentrant**

- 異なるチャンネルからリエントラントは可能です。

**Example**

「4.2 サンプルコード」を参照してください。

**Special Notes:**

なし

### 3.4 R\_ETHER\_Read\_ZC2()

R\_ETHER\_Read\_ZC2 関数は受信データが格納されたバッファの先頭アドレスへのポインタを返します。

#### Format

```
int32_t R_ETHER_Read_ZC2(  
    uint32_t      channel    /* ETHERC のチャンネル番号 */  
    void          ** pbuf    /* 受信データが格納されたバッファポインタ */  
);
```

#### Parameters

*channel*

ETHERC のチャンネル番号 (0、1) を指定します。

*\*\* pbuf*

受信データが格納されたバッファの先頭アドレスへのポインタを返します。

#### Return Values

1 以上の値	/* 受信したバイト数 */
ETHER_NO_DATA	/* ゼロが返されたときは、データが受信されていません */
ETHER_ERR_INVALID_CHAN	/* 存在しないチャンネルの場合 */
ETHER_ERR_INVALID_PTR	/* ポインタの値が、NULL もしくは FIT_NO_PTR の場合 */
ETHER_ERR_LINK	/* オートネゴシエーション処理が完了しておらず受信が許可されていない場合 */
ETHER_ERR_MPDE	/* マジックパケットの検出状態のため、送信と受信が許可されていない場合 */

#### Properties

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

#### Description

受信データが格納されたバッファの先頭アドレスへのポインタはパラメータ pbuf に格納して返されます。返されたポインタを利用して、ゼロコピーで操作が行えます。

戻り値は受信されたバイト数を示しています。呼び出し時に、データが存在しないときには値 ETHER\_NO\_DATA が返されます。オートネゴシエーション処理が完了しておらず受信が許可されていないときには値 ETHER\_ERR\_LINK が返されます。マジックパケット検出状態となっているときには値 ETHER\_ERR\_MPDE が返されます。

EDMAC は R\_ETHER\_Read\_ZC2 関数とは独立して動作します。EDMAC は受信ディスクリプタで指定されたバッファにデータを読み込みます。EDMAC の受信ディスクリプタが指しているバッファはイーサネットドライバによって静的に割り当てられます。

受信 FIFO オーバフロー、端数ビットフレーム受信エラー、ロングフレーム受信エラー、ショートフレーム受信エラー、PHY-LSI 受信エラー、受信フレーム CRC エラーが発生したフレームは受信フレームエラーとなります。受信フレームエラーが発生したディスクリプタのデータは破棄され、ステータスをクリアして読み込みを続けます。

#### Reentrant

- 異なるチャンネルからリエントラントは可能です。

**Example**

「4.2 サンプルコード」を参照してください。

**Special Notes:**

なし

---

### 3.5 R\_ETHER\_Read\_ZC2\_BufRelease()

---

R\_ETHER\_Read\_ZC2\_BufRelease 関数は R\_ETHER\_Read\_ZC2 関数で読み出したバッファを開放します。

**Format**

```
int32_t R_ETHER_Read_ZC2_BufRelease(  
    uint32_t      channel          /* ETHERC のチャンネル番号を指定します */  
);
```

**Parameters**

*channel*

ETHERC のチャンネル番号 (0、1) を指定します。

**Return Values**

<i>ETHER_SUCCESS</i>	/* 問題なく処理が完了した場合 */
<i>ETHER_ERR_INVALID_CHAN</i>	/* 存在しないチャンネルの場合 */
<i>ETHER_ERR_LINK</i>	/* オートネゴシエーション処理が完了しておらず受信が */
	/* 許可されていない場合 */
<i>ETHER_ERR_MPDE</i>	/* マジックパケットの検出状態のため、 */
	/* 送信と受信が許可されていない場合 */

**Properties**

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

**Description**

R\_ETHER\_Read\_ZC2\_BufRelease 関数は R\_ETHER\_Read\_ZC2 関数で読み出したバッファを開放します。

**Reentrant**

- 異なるチャンネルからリエントラントは可能です。

**Example**

「4.2 サンプルコード」を参照してください。

**Special Notes:**

本関数は R\_ETHER\_Read\_ZC2 関数でデータを読み出した後、実行してください。

### 3.6 R\_ETHER\_Write\_ZC2\_GetBuf()

R\_ETHER\_Write\_ZC2\_GetBuf 関数は送信データの書き込み先の先頭アドレスへのポインタが返されます。

#### Format

```
ether_return_t R_ETHER_Write_ZC2_GetBuf(
    uint32_t      channel      /* ETHERC のチャンネル番号 */
    void          ** pbuf      /* 送信データの書き込み先の先頭アドレスへのポインタ*/
    uint16_t      * pbuf_size  /* バッファに書き込み可能な上限サイズ */
);
```

#### Parameters

*channel*

ETHERC のチャンネル番号 (0、1) を指定します。

*\*\* pbuf*

送信データの書き込み先の先頭アドレスへのポインタが返されます。

*\* pbuf\_size*

バッファに書き込み可能な上限サイズが返されます。

#### Return Values

<i>ETHER_SUCCESS</i>	<i>/* 問題なく処理が完了した場合 */</i>
<i>ETHER_ERR_INVALID_CHAN</i>	<i>/* 存在しないチャンネルの場合 */</i>
<i>ETHER_ERR_INVALID_PTR</i>	<i>/* ポインタの値が、NULL もしくは FIT_NO_PTR の場合 */</i>
<i>ETHER_ERR_LINK</i>	<i>/* オートネゴシエーション処理が完了しておらず受信が */</i>
	<i>/* 許可されていない場合 */</i>
<i>ETHER_ERR_MPDE</i>	<i>/* マジックパケットの検出状態のため、 */</i>
	<i>/* 送信と受信が許可されていない場合 */</i>
<i>ETHER_ERR_TACT</i>	<i>/* 送信バッファに空きがない場合 */</i>

#### Properties

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

#### Description

送信データの書き込み先の先頭アドレスへのポインタはパラメータ pbuf に格納して返されます。またバッファに書き込み可能な上限サイズはパラメータ pbuf\_size に返されます。返されたポインタを利用して、ゼロコピーで操作が行えます。

戻り値は送信バッファ (pbuf) へ書き込みが可能であるか示しています。呼び出し時に、書き込みが可能などときには ETHER\_SUCCESS が返されます。オートネゴシエーション処理が完了しておらず送信が許可されていないときには値 ETHER\_ERR\_LINK が返されます。マジックパケット検出状態となっているときには値 ETHER\_ERR\_MPDE が返されます。送信バッファに空きがないときには値 ETHER\_ERR\_TACT が返されます。

EDMAC は R\_ETHER\_Write\_ZC2\_GetBuf 関数とは独立して動作します。EDMAC は送信ディスクリプタで指定されたバッファのデータを書き出します。EDMAC の送信ディスクリプタが指しているバッファはイーサネットドライバによって静的に割り当てられます。



**Reentrant**

- 異なるチャネルからリエントラントは可能です。

**Example**

「4.2 サンプルコード」を参照してください。

**Special Notes:**

なし

### 3.7 R\_ETHER\_Write\_ZC2\_SetBuf()

R\_ETHER\_Write\_ZC2\_SetBuf 関数は EDMAC に送信バッファのデータの送信を許可します。

#### Format

```
ether_return_t R_ETHER_Write_ZC2_SetBuf(
    uint32_t      channel      /* ETHERC のチャンネル番号 */
    const uint32_t len         /* イーサネットフレーム長から CRC の 4 バイトを
                                除いたサイズ (60 ~ 1514) */
);
```

#### Parameters

*channel*

ETHERC のチャンネル番号 (0、1) を指定します。

*len*

イーサネットフレーム長から CRC の 4 バイトを除いたサイズ (60 ~ 1514) を指定します。

#### Return Values

<i>ETHER_SUCCESS</i>	/* 問題なく処理が完了した場合 */
<i>ETHER_ERR_INVALID_CHAN</i>	/* 存在しないチャンネルの場合 */
<i>ETHER_ERR_INVALID_DATA</i>	/* 引数のとり得る値が、範囲外の場合 */
<i>ETHER_ERR_LINK</i>	/* オートネゴシエーション処理が完了しておらず受信が                                 許可されていない場合 */
<i>ETHER_ERR_MPDE</i>	/* マジックパケットの検出状態のため、                                 送信と受信が許可されていない場合 */

#### Properties

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

#### Description

本関数は 1 フレームの送信データの書き込みが完了した後、呼び出してください。  
 バッファ長に指定する値は、イーサネットフレームの最小値 64 バイトから CRC の 4 バイトを除いた 60 バイト以上かつイーサネットフレームの最大値 1518 バイトから CRC の 4 バイトを除いた 1514 バイト以下までの範囲としてください。

60 バイト未満のデータを送信する場合は、データを 0 パディングで埋めて 60 バイトとなるようにしてください。

戻り値は送信バッファに書き込んだデータの送信許可状態を示しています。呼び出し時に、送信バッファのデータの送信が許可されたときには *ETHER\_SUCCESS* が返されます。オートネゴシエーション処理が完了しておらず送信が許可されていないときには値 *ETHER\_ERR\_LINK* が返されます。マジックパケット検出状態となっているときには値 *ETHER\_ERR\_MPDE* が返されます。

#### Reentrant

- 異なるチャンネルからリエントラントは可能です。

#### Example

「4.2 サンプルコード」を参照してください。

#### Special Notes:

本関数は 1 フレームの送信データの書き込みが完了した後、呼び出してください。

60 バイト未満のデータを送信する場合は、データを 0 パディングで埋めて 60 バイトとなるようにしてください。

---

### 3.8 R\_ETHER\_CheckLink\_ZC()

---

R\_ETHER\_CheckLink\_ZC は物理的なイーサネットのリンク状態を、PHY 管理インタフェースを使用してチェックします。PHY が適切に初期化されている相手デバイスとケーブルが接続されていれば、イーサネットのリンク状態がリンクアップとなります。

**Format**

```
ether_return_t R_ETHER_CheckLink_ZC(  
    uint32_t      channel          /* ETHERC のチャンネル番号 */  
);
```

**Parameters**

*channel*

ETHERC のチャンネル番号 (0、1) を指定します。

**Return Values**

<i>ETHER_SUCCESS</i>	<i>/* リンク状態がリンクアップの場合 */</i>
<i>ETHER_ERR_OTHER</i>	<i>/* リンク状態がリンクダウンの場合 */</i>
<i>ETHER_ERR_INVALID_CHAN</i>	<i>/* 存在しないチャンネルの場合 */</i>

**Properties**

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

**Description**

R\_ETHER\_CheckLink\_ZC 関数はイーサネットのリンク状態を知るために PHY 管理インタフェースを使用します。この情報は PHY-LSI の Basic Status レジスタ (レジスタ 1) から読み出されます。リンク状態がリンクアップのときには ETHER\_SUCCESS が返され、リンク状態がリンクダウンのときには ETHER\_ERR\_OTHER が返されます。

**Reentrant**

- 異なるチャンネルからリエントラントは可能です。

**Example**

「4.2 サンプルコード」を参照してください。

**Special Notes:**

なし

### 3.9 R\_ETHER\_LinkProcess()

R\_ETHER\_LinkProcess 関数はリンク信号変化割り込み処理およびマジックパケット検出割り込み処理を行います。

**Format**

```
void R_ETHER_LinkProcess(  
    uint32_t channel /* ETHERC のチャンネル番号 */  
);
```

**Parameters**

*channel*

ETHERC のチャンネル番号 (0、1) を指定します。

**Return Values**

なし

**Properties**

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

**Description**

R\_ETHER\_LinkProcess 関数はリンク信号変化割り込み処理およびマジックパケット検出割り込み処理を行います。

- マジックパケット検出割り込みが発生していた場合
  - R\_ETHER\_Control 関数で登録したコールバック関数により、マジックパケットを検出したことを通知します。
- リンク信号変化 (リンク状態がリンクアップ) 割り込みが発生していた場合
  - ETHERC および EDMAC を初期化した後、オートネゴシエーション結果から全二重 / 半二重、リンク速度、フロー制御に関して適切なコンフィグレーションを決定して送受信機能を有効にします。
  - EDMAC のディスクリプタを初期状態にセットアップします。
  - R\_ETHER\_Control 関数で登録したコールバック関数により、リンク信号変化 (リンクアップ) を検出したことを通知します。
- リンク信号変化 (リンク状態がリンクダウン) 割り込みが発生していた場合
  - 送受信機能を無効にした後、R\_ETHER\_Control 関数で登録したコールバック関数により、リンク信号変化 (リンクダウン) を検出したことを通知します。

**Reentrant**

- 異なるチャンネルからリエントラントは可能です。

**Example**

「4.2 サンプルコード」を参照してください。

**Special Notes:**

本関数は必ず通常処理ルーチンで定期的に呼び出してください。本関数がコールされない場合、送受信およびマジックパケット検出モードへの変更が正常に動作致しませんのでご注意ください。

R\_ETHER\_Control 関数を用いて、コールバック関数を登録していない場合は、コールバック関数による通知はありません。

---

### 3.10 R\_ETHER\_WakeOnLAN()

---

R\_ETHER\_WakeOnLAN 関数は ETHERC の設定を通常を送受信動作からマジックパケット検出動作に切り替えます。

**Format**

```
ether_return_t R_ETHER_WakeOnLAN(  
    uint32_t channel /* ETHERC のチャンネル番号 */  
);
```

**Parameters**

*channel*

ETHERC のチャンネル番号 (0、1) を指定します。

**Return Values**

<i>ETHER_SUCCESS</i>	<i>/* 問題なく処理が完了した場合 */</i>
<i>ETHER_ERR_INVALID_CHAN</i>	<i>/* 存在しないチャンネルの場合 */</i>
<i>ETHER_ERR_LINK</i>	<i>/* オートネゴシエーション処理が完了しておらず受信が許可されていない場合 */</i>
<i>ETHER_ERR_OTHER</i>	<i>/* リンク状態がリンクダウンでマジックパケット検出動作に切り替えた場合 */</i>

**Properties**

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

**Description**

R\_ETHER\_WakeOnLAN 関数は ETHERC と EDMAC を初期化した後、ETHERC の設定をマジックパケット検出動作に切り替えます。

戻り値は ETHERC がマジックパケット検出動作への切り替えが成功したか否かを示しています。呼び出し時に、オートネゴシエーション処理が完了しておらず送受信が許可されていないときには値

ETHER\_ERR\_LINK が返されます。設定をマジックパケット検出動作に切り替えた後、リンク状態がリンクダウンとなっていたときには値 ETHER\_ERR\_OTHER が返されます。

**Reentrant**

- 異なるチャンネルからリエントラントは可能です。

**Example**

「4.2 サンプルコード」を参照してください。

**Special Notes:**

なし

---

### 3.11 R\_ETHER\_CheckWrite()

---

データ送信が完了したことを確認する関数です。

**Format**

```
ether_return_t R_ETHER_CheckWrite(  
    uint32_t      channel          /* ETHERC のチャンネル番号 */  
);
```

**Parameters**

*channel*

ETHERC のチャンネル番号 (0、1) を指定します。

**Return Values**

<i>ETHER_SUCCESS</i>	<i>/* 問題なく処理が完了した場合 */</i>
<i>ETHER_ERR_INVALID_CHAN</i>	<i>/* 存在しないチャンネルの場合 */</i>

**Properties**

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

**Description**

R\_ETHER\_CheckWrite 関数は、データが送信されたことを確認します。  
送信が完了した場合には、戻り値 ETHER\_SUCCESS を返します。

**Reentrant**

- 異なるチャンネルからリエントラントは可能です。

**Example**

「4.2 サンプルコード」を参照してください。

**Special Notes:**

本関数は、R\_ETHER\_SetBuf 関数で送信するデータを書込みした後、呼び出してください。  
また、R\_ETHER\_SetBuf 関数を呼び出した後、実際のデータ送信が完了するまでには数十  $\mu$ sec 必要になります。そのため、データ送信後に R\_ETHER\_Close 関数にて、イーサネットモジュールを終了する場合は、R\_ETHER\_Setbuf 関数を呼び出した後、本関数を呼び出し、データ送信が完了したことを待ってから R\_ETHER\_Close 関数を呼び出してください。本関数を呼び出さずに R\_ETHER\_Close 関数を呼び出した場合、データ送信が中断されることがあります。

### 3.12 R\_ETHER\_Read()

R\_ETHER\_Read 関数は指定した受信バッファヘデータを受信します。

#### Format

```
int32_t R_ETHER_Read(
    uint32_t channel /* ETHERC のチャンネル番号 */
    void * pbuf /* 受信データの保存先 */
);
```

#### Parameters

*channel*

ETHERC のチャンネル番号 (0、1) を指定します。

*\* pbuf*

受信バッファの (受信データの保存先) を指定します。

最大 1514 バイトの書込みがあります。本関数を呼び出す際には、1514 バイト確保した配列の先頭アドレスを指定してください。

#### Return Values

1 以上の値	/* 受信したバイト数 */
ETHER_NO_DATA	/* ゼロが返されたときは、データが受信されていません */
ETHER_ERR_INVALID_CHAN	/* 存在しないチャンネルの場合 */
ETHER_ERR_INVALID_PTR	/* ポインタの値が、NULL もしくは FIT_NO_PTR の場合 */
ETHER_ERR_LINK	/* オートネゴシエーション処理が完了しておらず受信が許可されていない場合 */
ETHER_ERR_MPDE	/* マジックパケットの検出状態のため、送信と受信が許可されていない場合 */

#### Properties

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

#### Description

指定した受信バッファに受信データを保存します。

戻り値は受信されたバイト数を示しています。呼び出し時に、データが存在しないときには値 ETHER\_NO\_DATA が返されます。オートネゴシエーション処理が完了しておらず受信が許可されていないときには値 ETHER\_ERR\_LINK が返されます。マジックパケット検出状態となっているときには値 ETHER\_ERR\_MPDE が返されます。

受信 FIFO オーバフロー、端数ビットフレーム受信エラー、ロングフレーム受信エラー、ショートフレーム受信エラー、PHY-LSI 受信エラー、受信フレーム CRC エラーが発生したフレームは受信フレームエラーとなります。受信フレームエラーが発生したディスクリプタのデータは破棄され、ステータスをクリアして読み込みを継続します。

#### Reentrant

- 異なるチャンネルからリエントラントは可能です。

#### Example

「4.2 サンプルコード」を参照してください。

**Special Notes:**

本関数は内部で R\_ETHER\_Read\_ZC2 関数および、R\_ETHER\_Read\_ZC2\_BufRelease 関数を呼び出しております。このため、EDMAC の受信ディスクリプタが指しているバッファと R\_ETHER\_Read 関数経由で指定した受信バッファの間でデータのコピーが行われます。(最大 1514 バイトの書き込みがありますので、指定する受信バッファは 1514 バイト確保してください。)

R\_ETHER\_Read 関数を使用する場合は R\_ETHER\_Read\_ZC2 関数および、R\_ETHER\_Read\_ZC2\_BufRelease 関数は使わないようにお願いいたします。

本関数では、標準関数 memcpy を使用するため、string.h をインクルードしています。



### 3.13 R\_ETHER\_Write()

R\_ETHER\_Write 関数は指定した送信バッファからデータを送信します。

#### Format

```
ether_return_t R_ETHER_Write(
    uint32_t      channel      /* ETHERC のチャンネル番号 */
    void          * pbuf       /* 送信バッファポインタ */
    const uint32_t len         /* イーサネットフレーム長から CRC の 4 バイトを */
                                /* 除いたサイズ ( 60 ~ 1514 ) */
);
```

#### Parameters

*channel*

ETHERC のチャンネル番号 ( 0, 1 ) を指定します。

*\* pbuf*

送信バッファ ( 送信データの書き込み先 ) を指定します。

*len*

イーサネットフレーム長から CRC の 4 バイトを除いたサイズ ( 60 ~ 1514 ) を指定します。

#### Return Values

<i>ETHER_SUCCESS</i>	/* 問題なく処理が完了した場合 */
<i>ETHER_ERR_INVALID_CHAN</i>	/* 存在しないチャンネルの場合 */
<i>ETHER_ERR_INVALID_DATA</i>	/* 引数のとり得る値が、範囲外の場合 */
<i>ETHER_ERR_INVALID_PTR</i>	/* ポインタの値が、NULL もしくは <i>ETHER_NO_PTR</i> の場合 */
<i>ETHER_ERR_LINK</i>	/* オートネゴシエーション処理が完了しておらず受信が */
	/* 許可されていない場合 */
<i>ETHER_ERR_MPDE</i>	/* マジックパケットの検出状態のため、 */
	/* 送信と受信が許可されていない場合 */
<i>ETHER_ERR_TACT</i>	/* 送信バッファに空きがない場合 */

#### Properties

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

#### Description

指定した送信バッファからデータを送信します。

バッファ長に指定する値は、イーサネットフレームの最小値 64 バイトから CRC の 4 バイトを除いた 60 バイト以上かつイーサネットフレームの最大値 1518 バイトから CRC の 4 バイトを除いた 1514 バイト以下までの範囲としてください。

60 バイト未満のデータを送信する場合は、データを 0 パディングで埋めて 60 バイトとなるようにしてください。

戻り値は送信バッファに書き込んだデータの送信許可状態を示しています。呼び出し時に、送信バッファのデータの送信が許可されたときには *ETHER\_SUCCESS* が返されます。オートネゴシエーション処理が完了しておらず送信が許可されていないときには値 *ETHER\_ERR\_LINK* が返されます。マジックパケット検出状態となっているときには値 *ETHER\_ERR\_MPDE* が返されます。送信バッファに空きがないときには値 *ETHER\_ERR\_TACT* が返されます。

#### Reentrant

- 異なるチャンネルからリエントラントは可能です。

**Example**

「4.2 サンプルコード」を参照してください。

**Special Notes:**

60 バイト未満のデータを送信する場合は、データを 0 パディングで埋めて 60 バイトとなるようにしてください。

本関数は内部で R\_ETHER\_Write\_ZC2\_GetBuf 関数および、R\_ETHER\_Write\_ZC2\_SetBuf 関数を呼び出しております。このため、EDMAC の送信ディスクリプタが指しているバッファと R\_ETHER\_Write 関数経由で指定した送信バッファの間でデータのコピーが行われます。

R\_ETHER\_Write 関数を使用する場合は R\_ETHER\_Write\_ZC2\_GetBuf 関数および、R\_ETHER\_Write\_ZC2\_SetBuf 関数は使わないようにお願いいたします。

本科数では、標準関数 memset、memcpy を使用するため、string.h をインクルードしています。

### 3.14 R\_ETHER\_Control()

コントロールコードに対応した処理を行う関数です。

#### Format

```
ether_return_t R_ETHER_Control(
    ether_cmd_t const    cmd          /* コントロールコード */
    ether_param_t const  control     /* コントロールコードに応じたパラメータ */
);
```

#### Parameters

*cmd*

コントロールコードを指定します。

*control*

コントロールコードに応じたパラメータを指定します。

#### Return Values

<i>ETHER_SUCCESS</i>	/* 問題なく処理が完了した場合 */
<i>ETHER_ERR_INVALID_CHAN</i>	/* 存在しないチャンネルの場合 */
<i>ETHER_ERR_CHAN_OPEN</i>	/* 他のアプリケーションが使用しているため */
	/* Ether を Open できない場合 */
<i>ETHER_ERR_INVALID_ARG</i>	/* 不正な引数の場合 */

#### Properties

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

#### Description

コントロールコードに対応した処理を行います。対応していないコントロールコードの場合、戻り値 *ETHER\_ERR\_INVALID\_ARG* を返します。

以下に、対応するコントロールコードを示します。

コントロールコード	概要
CONTROL_SET_CALLBACK	リンク信号変化割り込みがあったとき、もしくはマジックパケット検出割り込みがあったときにコールバックされる関数を登録します。 第2引数で指定した関数を登録します。
CONTROL_SET_PROMISCUOUS_MODE	ETHERC モードレジスタ (ECMR) のプロミスカスモードビット (PRM) を設定します。 第2引数には、PRM を設定する側の ETHERC のチャンネル番号および、PRM の値を格納している変数のアドレスを設定します。
CONTROL_SET_INT_HANDLER	EINT0/1 ステータス割り込みがあったときにコールバックされる関数を登録します。 第2引数で指定した関数を登録します。
CONTROL_POWER_ON	ETHERC 用の端子を設定し、ETHERC/EDMAC のモジュールストップを解除します。 第2引数に使用する ETHERC のチャンネルを指定します。
CONTROL_POWER_OFF	ETHERC/EDMAC のモジュールストップに遷移させます。 第2引数にモジュールストップに遷移させる ETHERC のチャンネルを指定します。

**Reentrant**

- 可能

**Example**

コールバック関数を登録する場合)

```
void callback(void*);
```

```
ether_return_t    ret;  
ether_param_t    param;  
ether_cb_t       cb_func;
```

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

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

プロミスカスモードを設定する場合)

```
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);
```

割り込みハンドラ関数を登録する場合)

```
void int_handler(void*);
```

```
ether_return_t    ret;  
ether_param_t     param;  
ether_cb_t        cb_func;
```

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

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

割り込みハンドラ関数)

```
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_CHANNEL_MAX > p_arg->channel)  
    {  
        status_ecsr[p_arg->channel] = p_arg->status_ecsr;  
        status_eesr[p_arg->channel] = p_arg->status_eesr;  
    }  
}
```

ETHERC/EDMAC モジュールストップの解除)

R\_ETHER\_Initial の Example を参照。

ETHERC/EDMAC モジュールストップへの遷移)

R\_ETHER\_CheckWrite の Example を参照。

**Special Notes:**

コールバック関数の登録や割り込みハンドラ関数の登録は、R\_ETHER\_Open 関数を呼び出す前に登録してください。R\_ETHER\_Open 関数を呼び出してから登録した場合は、最初の割り込みを検出できない場合があります。

プロミスキャスモードを設定する場合、コントロールコードに CONTROL\_POWER\_ON を設定し、本関数を呼び出してから、設定してください。コントロールコードに CONTROL\_POWER\_ON を設定し、本関数を呼び出しせず、プロミスキャスモードを設定した場合は、意図した値が ETHERC モードレジスタに設定されません。

---

### 3.15 R\_ETHER\_GetVersion()

---

API のバージョンを返す関数です。

**Format**

uint32\_t R\_ETHER\_GetVersion(void);

**Parameters**

なし

**Return Values**

バージョン番号

**Properties**

r\_ether\_rx\_if.h にプロトタイプ宣言されています。

**Description**

本 API のバージョン番号を返します。

**Reentrant**

- 異なるチャネルからリエントラントは可能です。

**Example**

```
uint32_t  version;
```

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

**Special Notes:**

この関数は“#pragma inline”を使用してインライン化されています。

## 4. 付録

### 4.1 セクション配置

表 4.1に本モジュールのセクション配置例を示します。

表4.1 プログラムのセクション配置例

アドレス	デバイス	セクション	説明
0x00000000	内蔵 RAM	B_ETHERNET_BUFFERS_1	送信バッファおよび受信バッファ領域
		B_RX_DESC_1	受信ディスクリプタ領域
		B_TX_DESC_1	送信ディスクリプタ領域
		SI	割り込みスタック領域
		SU	ユーザスタック領域
		B_1	1byte 境界の未初期化データ領域
		R_1	1byte 境界の初期化データ領域（変数）
		B_2	2byte 境界の未初期化データ領域
		R_2	2byte 境界の初期化データ領域（変数）
		B	4byte 境界の未初期化データ領域
		R	4byte 境界の初期化データ領域（変数）
0x00120064	内蔵 ROM	OPT_MEMORY	オプション設定メモリ領域
0xFFFF8000		C_1	1byte 境界の定数領域
		C_2	2byte 境界の定数領域
		C	4byte 境界の定数領域
		C\$*	C\$*セクション（C\$DEC、C\$BSEC、C\$VECT）の定数領域
		D*	初期化データ領域
		P*	プログラム領域
		W*	switch 文分岐テーブル領域
		L	文字列リテラル領域
0xFFFFFFF80		EXCEPTVECT	割り込みベクタ領域
0xFFFFFFF8C		RESETVECT	リセットベクタ領域

#### 4.1.1 セクション配置の注意点

- 受信ディスクリプタ領域および送信ディスクリプタ領域は、EDAMC モードレジスタ（EDMR）の送受信ディスクリプタ長指定ビット（DL）を、16byte 設定にしているため、16byte 境界になるよう配置してください。
- 送信バッファおよび受信バッファ領域は 32byte 境界になるよう配置してください。

## 4.2 サンプルコード

以下に API の使用例を掲載します。

コードに含まれる MAC アドレスは参考例です。お客様の製品には必ず IEEE に申請した MAC アドレスを使用するようにしてください。

```
#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 pointer 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 "&pwrite_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 *)&pwrite_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(pwrite_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 )
{
    /* 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.
 * The MCU calls 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 "&pwrite_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 *)&pwrite_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(pwrite_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 -
* 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 -
*
* 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. */
CMT.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. 提供するモジュール

提供するモジュールは、ルネサス エレクトロニクスホームページから入手してください。

## 6. 参考ドキュメント

### ユーザーズマニュアル：ハードウェア

RX64M グループ ユーザーズマニュアル ハードウェア編 (ドキュメント No.R01UH0377JJ)  
(最新版をルネサス エレクトロニクスホームページから入手してください。)

### テクニカルアップデート/テクニカルニュース

(最新の情報をルネサス エレクトロニクスホームページから入手してください。)

### ユーザーズマニュアル：開発環境

RX ファミリ C/C++コンパイラパッケージ V.1.01 ユーザーズマニュアル Rev.1.00 (R20UT0570JJ0100)  
(最新版をルネサス エレクトロニクスホームページから入手してください。)

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ

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

お問い合わせ先

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

すべての商標および登録商標は、それぞれの所有者に帰属します。

## 改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2014.07.29	—	初版発行

## 製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

### 1. 未使用端子の処理

【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。

CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。

### 2. 電源投入時の処置

【注意】電源投入時は、製品の状態は不定です。

電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。

同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

### 3. リザーブアドレスのアクセス禁止

【注意】リザーブアドレスのアクセスを禁止します。

アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレスがあります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

### 4. クロックについて

【注意】リセット時は、クロックが安定した後、リセットを解除してください。

プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。

リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

### 5. 製品間の相違について

【注意】型名の異なる製品に変更する場合は、事前に問題ないことをご確認下さい。

同じグループのマイコンでも型名が違うと、内部メモリ、レイアウトパターンの相違などにより、特性が異なる場合があります。型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。

## ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したのですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。  
標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、家電、工作機械、パーソナル機器、産業用ロボット等  
高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置等  
当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じて、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問い合わせください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関し、当社は、一切その責任を負いません。
9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事情報に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注2. 本資料において使用されている「当社製品」とは、注1において定義された当社の開発、製造製品をいいます。



ルネサス エレクトロニクス株式会社

■営業お問い合わせ窓口

<http://www.renesas.com>

※営業お問い合わせ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社 〒100-0004 千代田区大手町2-6-2（日本ビル）

■技術的なお問い合わせおよび資料のご請求は下記へどうぞ。  
総合お問い合わせ窓口：<http://japan.renesas.com/contact/>