RX Family

EPTPC Light Module Firmware Integration Technology

Introduction

This document explains the PTP software driver light version (PTP light driver, EPTPC Light FIT module) based on the firmware integration technology (FIT). The PTP light driver is reduced the PTP (Precision Time Protocol) function defined by the IEEE1588-2008 specification [1] from the PTP driver (full version) [2], and focus on the enhanced standard Ethernet function such as the simple switch and multicast frame filter.

Target Device

This API supports the following device.

- RX64M Group
- RX71M Group
- RX72M Group
- RX72N Group

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

1.1 EPTPC Light FIT Module

The EPTPC FIT module can be used by being implemented in a project as an API. See section 2.14 Adding the FIT Module to Your Project for details on methods to implement this FIT module into a project.

1.2 Overview of the EPTPC Light FIT Module

This document explains the PTP software driver light version (hereafter PTP light driver) based on the firmware integration technology (FIT) and shows the usage example. The PTP light driver supports the simple switch and multicast frame filter functions applied to the Ethernet frame using the EPTPC peripheral module (EPTPC). This driver is subset of the PTP driver (full version) and has no PTP function.

1.3 Related documents

[1] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, Revision of IEEE Std 1588-2008, Mar 2008

[2] RX Family EPTPC Module Using Firmware Integration Technology, Rev.1.17, Document No. R01AN1943EJ0117, Nov 30, 2019

[3] RX Family Ethernet Module Using Firmware Integration Technology, Rev.1.20, Document No. R01AN2009EJ0120, Nov 22, 2019

[4] RX Family Ethernet Simple Switch Function Using Firmware Integration Technology Modules, Rev.1.11, Document No. R01AN3036EJ0111, Nov 11, 2016

[5] RX Family Ethernet Multicast Frame Filter Function Using Firmware Integration Technology Modules, Rev.1.11, Document No. R01AN3037EJ0111, Nov 11, 2016

[6] RX64M Group Renesas Starter Kit+ User's Manual For e^2 studio, Rev. 1.10, Document No. R20UT2593EG0110, Jun 25, 2015

[7] RX71M Group Renesas Starter Kit+ User's Manual, Rev. 1.00, Document No. R20UT3217EG0100, Jan 23, 2015

1.4 Hardware Structure

The Ethernet peripheral modules of the RX64M/71M/72M group are composed of the EPTPC, the PTP Host interface peripheral module (PTPEDMAC), dual channel Ethernet MAC ones (ETHERC (CH0), ETHERC (CH1)) and dual channel Ethernet Host interface ones (EDMAC (CH0), EDMAC (CH1)). The EPTPC is divided to PTP Frame Operation (CH0) part, PTP Frame Operation (CH1) part, Packet Relation Control part and Statistical Time Correction Algorithm part from their functionality. The EPTPC is also connected to the motor control timers (MTU3 and GPT peripheral modules) and the general ports (I/O ports) via ELC peripheral module to synchronous activation of multiple motors and output synchronous pulses.

Followings are the summary of the Ethernet peripheral modules and Figure 1.1 shows the related hardware's block diagram. This module only uses the enhanced standard Ethernet function.

1. Synchronous function (EPTPC and PTPEDMAC)

- Based on the IEEE1588-2008 Version2

- Time synchronous function issuing PTP messages (Ethernet frame[1] and UDP IPv4 format[2])

- Master and Slave, OC, BC, TC functionality

- Time deviation is corrected by the statistical correction method (Gradient prediction time correction algorithm)

- Timer event output (6CH, rise/fall edges, event flag auto clear)

- Motor control timer (MTU3, GPT) is started synchronously with the timer event via ELC

- Synchronous pulses are outputted connecting EPTPC to I/O ports via ELC

- Selectable PTP message operation (PTP module internal operation, CPU via PTPEDMAC, to other port)

2. Enhanced standard Ethernet function

- Possible to use the independent dual channels Ethernet
- HW switch (selectable Cut Through or Store & Forward internal frame propagation)
- HW multicast frame filter (all receive, all cancel, receive specific two frames)

[1] In case of RX64M Group, supports only Ethernet II frame format (not support IEEE802.3 frame format)

[2] Not support UDP IPv6

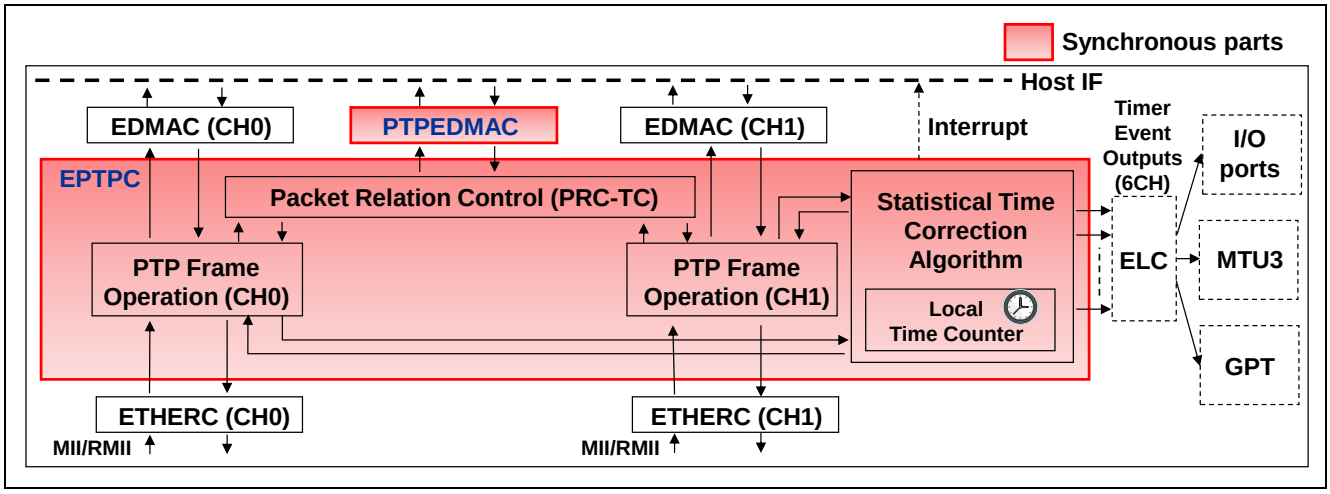


Figure 1.1 Hardware block diagram

1.5 Software Structure

The PTP light driver always should be used with Ether driver [3] and need to be combination with TCP/IP middle ware in case of applied to TCP/IP system. The PTP light driver set the simple switch and multicast frame filter functions depend on the requirement from upper layer software. Figure 1.2 shows the typical structure and functional overview of the software.

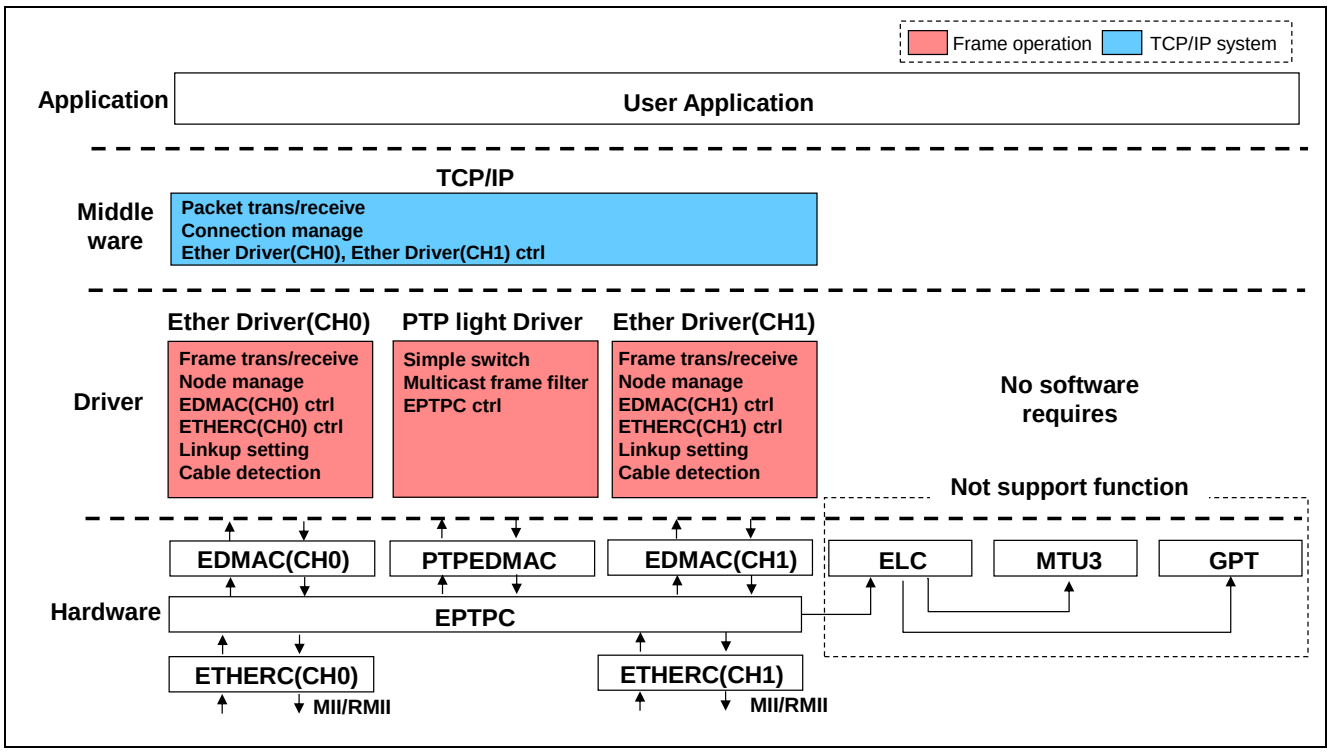


Figure 1.2 Software structure system example

1.6 File Structure

The PTP light driver is composed of a single source file whose name is “r_ptp_light.c”.

1.7 API Overview

The API functions of the PTP light driver show the Table 1.1.

Table 1.1 API Functions

Function	Contents
R_PTPL_GetVersion()	Get PTP light driver version number.
R_PTPL_Reset()	Reset EPTPC.
R_PTPL_SetTran()	Set inter ports transfer mode
R_PTPL_SetMCFilter()	Set Multicast frames (MC) filter (FFLTR).
R_PTPL_SetExtPromiscuous()	Set/clear extended promiscuous mode.
R_PTPL_Init()	Initialize EPTPC.
R_PTPL_RegMINTHndr()	Register a user function to MINT interrupt handler of EPTPC.
R_PTPL_GetSyncConfig()	Get PTP frame control configuration (SYRFL1R, SYRFL2R and SYCONFR).
R_PTPL_SetSyncConfig()	Set PTP frame control configuration (SYRFL1R, SYRFL2R and SYCONFR).
R_PTPL_SetInterrupt()	Enable EPTPC INFABT interrupt.
R_PTPL_ChkInterrupt()	Check INFABT interrupt occurrence.
R_PTPL_ClrInterrupt()	Clear INFABT interrupt occurrence flag.

2. API Information

This driver API follows the Renesas API naming standards.

2.1 Hardware Requirements

This driver requires your MCU supports the following feature:

- EPTPC
- ETHERC
- EDMAC

Those used examples are described by “Ethernet Simple Switch Function [4]” and “Ethernet Multicast Frame Filter Function [5]”.

2.2 Hardware Resource Requirements

This section details the hardware peripherals that this example requires. Unless explicitly stated, these resources must be reserved for the following driver, and the user cannot use them.

2.2.1 EPTPC Channel

The driver uses the EPTPC. This resource needs to the inter ports frame transfer between CH0 and CH1, and the reception filter of the multicast frame.

2.2.2 ETHERC Channel

The driver uses the ETHEC (CH0), ETHEC (CH1) or both depend on the system. Those resources need to the Ethernet MAC operations.

2.2.3 EDMAC Channel

The driver uses the EDMAC (CH0), EDMAC (CH1) or both. Those resources need to the CPU Host interface of standard Ethernet frame operations.

2.3 Software Requirements

This driver is dependent on the following packages (FIT modules):

- r_bsp
- r_ether_rx

2.4 Limitations

There are following limitations in this driver.

- Not support PTP time synchronization.
- Cannot use the PTP driver (full version) [2] simultaneously.
- Cannot receive and process the PTP message frames[1].

[1] Relay control is possible.

2.5 Supported Toolchains

This driver has been confirmed to work with the toolchain listed in 4.5 Confirmed Operation Environment.

2.6 Interrupt vector

The EPTPC MINT interrupt is enabled by executing the R_PTPL_Init function.

Table 2.1 lists the interrupt vector used in the EPTPC FIT Module.

Table 2.1 Interrupt Vector Used in the EPTPC FIT Module

Device	Interrupt Vector
RX64M, RX71M, RX72M and RX72N	GROUPAL1 interrupt (vector no.: 113) ● EPTPC MINT interrupt (group interrupt source no.: 0)

2.7 Header Files

All API calls are accessed by including a single file, *r_ptp_light_rx_if.h*, which is supplied with this driver's project code.

2.8 Integer Types

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

2.9 Configuration Overview

The configuration options in this driver are specified in *r_ptp_light_rx_config.h.* The option names and setting values are listed in the table below.

Configuration options	
#define PTPL_CFG_MODE **#define PTPL_CFG_MODE_CH0 (0x01)** **#define PTPL_CFG_MODE_CH1 (0x02)** **#define PTPL_CFG_MODE_POLL (0x10)** **#define PTPL_CFG_MODE_HWINT (0x20)** **- Default value = 0x23**	Specify the PTP light driver mode. Select the enable channels and the method of status check. - When bit0 is set to 1, channel 0 is enabled. - When bit1 is set to 1, channel 1 is enabled. If bit0 and bit1 are set, channel 0 and channel 1 are enabled. - When bit4 is set to 1, status checking is software polling. This is not supported in this version. - When bit5 is set to 1, status checking is hardware interrupt. Please set this value in this version.
#define PTPL_CFG_INTERRUPT_LEVEL **- Default value = 2**	Specifies interrupt priority levels of EPTPC interrupts. Specify the level between 1 and 15.

2.10 Parameters

This section details the data structures that are used with the driver's API functions. Those structures are located in *r_ptp_light_rx_if.h* and *r_ptp.c* as the prototype declarations of API functions.

2.10.1 Constant

```
/* Number of ports */
#define NUM_PORT (2) /* Set 2 in the RX64M/71M */
```

```
/* Inter ports transfer mode */
typedef enum
{
    ST_FOR = 0, /* Store and forward mode (legacy compatible) */
    CT_THR = 1  /* Cut through mode */
} TranMode;
```

```
/* Relay enable directions (bit map form) */
typedef enum
{
    ENAB_NO = 0x00, /* Prohibit relay */
    ENAB_01 = 0x01, /* Enable CH0 to CH1 */
    ENAB_10 = 0x02, /* Enable CH1 to CH0 */
    ENAB_BT = 0x03  /* Enable CH0 to CH1 and CH1 to CH0 */
} RelEnabDir;
```

```
/* Multicast(MC) frames filter setting */
typedef enum
{
    MC_REC_ALL = 0, /* Receive all MC frames (legacy compatible) */
    MC_REC_NO,      /* Do not receive MC frame */
    MC_REC_REG0,    /* Receive only the MC frame registered FMAC0R(U/L) */
    MC_REC_REG1,    /* Receive only the MC frame registered FMAC1R(U/L) */
} MCRecFil;
```

```
/* MINT interrupt register */
typedef enum
{
   MINT_FUNC_PRC = 0,  /* Interrupt from PRC-TC */
   MINT_FUNC_SYN0,     /* Interrupt from SYNFP0 */
   MINT_FUNC_SYN1,     /* Interrupt from SYNFP1 */
} MINT_Reg;
```

2.10.2 Data Type

```
/* Register access structure to SYNFP0 or SYNFP1 part of the EPTPC */
static volatile struct st_eptpc0 R_BSP_EVENACCESS_SFR *synfp[2] =
{
   &EPTPC0,
   &EPTPC1,
};
```

```
/* PTP part port related structure */
typedef struct
{
   uint8_t  macAddr[6];
   uint8_t  ipAddr[4];
} PTPLPort;
```

```
/* PTP part configuration structure (port information) */
typedef struct
{
   PTPLPort port[NUM_PORT];
} PTPLConfig;
```

2.11 Return Values

This section describes return values of API function. Those enumerations are located in *r_ptp_light_rx_if.h* as the prototype declarations of API functions.

```
/* PTP light driver return value */
typedef enum
{
   PTPL_ERR_TOUT = -3,  /* Timeout error */
   PTPL_ERR_PARAM = -2, /* Parameter error */
   PTPL_ERR = -1,       /* General error */
   PTPL_OK = 0
} ptpl_return_t;
```

2.12 Callback Function

In this module, the callback function Eptpc_isr is called when the EPTPC MINT interrupt occurs.

2.13 Code Size

The sizes of ROM, RAM and maximum stack usage associated with this module are listed below.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in “2.9 Configuration Overview”.

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

Module Revision: r_ptp_light_rx rev1.20

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00

(The option of “-lang = c99” is added to the default settings of the integrated development environment.)

GCC for Renesas RX 4.08.04.201902

(The option of “-std=gnu99” is added to the default settings of the integrated development environment.)

IAR C/C++ Compiler for Renesas RX version 4.12.1

(The default settings of the integrated development environment.)

Configuration Options: Default settings.

ROM, RAM and Stack Code Sizes

Device	Category	File	Memory Used		
			Renesas Compiler	GCC	IAR Compiler
RX64M	ROM	r_ptp_light.c	**1222 bytes**	**2543 bytes**	**1842 bytes**
	RAM	r_ptp_light.c	**22 bytes**	**24 bytes**	**22 bytes**
	STACK	r_ptp_light.c	**76 bytes**	-	**52 bytes**
RX72M	ROM	r_ptp_light.c	**1257 bytes**	**2591 bytes**	**1914 bytes**
	RAM	r_ptp_light.c	**22 bytes**	**24 bytes**	**22 bytes**
	STACK	r_ptp_light.c	**76 bytes**	-	**80 bytes**

2.14 Adding the FIT Module to Your Project

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

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

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

(3) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

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

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

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

Target devices describing “WAIT_LOOP”

- RX64M Group
- RX71M Group
- RX72M Group
- RX72N Group

3. API Functions

3.1 R_PTPL_GetVersion ()

This function returns the version number of PTP light driver.

Format

uint32_t R_PTPL_GetVersion(void);

Parameters

None

Return Values

RX_PTPL_VERSION_MAJOR (upper 16bit): Major version number
RX_PTPL_VERSION_MINO (lower 16bit): Minor version number

Properties

Prototyped in “r_ptp_light_rx_if.h”.

Description

Return major and minor version number of PTP light driver.

This driver version number is “1.13”.

- Upper 16 bit indicates major version number
  RX_PTP_LIGHT_VERSION_MAJOR: current value = H’1.

- Lower 16 bit indicates minor version number
  RX_PTP_LIGHT_VERSION_MINOR: current value = H’13.

Reentrant

Function is reentrant.

Example

Example showing this function being used.

```
#include <stdio.h>
#include "r_ptp_light_rx_if.h"

    uint32_t ptp_version;

    ptp_version = R_PTPL_GetVersion();

    printf("PTP light driver major version = %d\n", ptp_version >> 16u);
    printf("PTP light driver minor version = %d\n", ptp_version & 0xFFFF);
```

Special Notes

Return value itself will be change depend on the driver version.

3.2 R_PTPL_Reset ()

This function resets EPTPC.

Format

void R_PTPL_Reset(void);

Parameters

None

Return Values

None

Properties

Prototyped in “r_ptp_light_rx_if.h ".

Description

This function resets the EPTPC. The following operations are executed.

- Reset EPTPC
  Setting “1” RESET bit of the PTRSTR register.

- Wait reset complete
  More than 64 PCLKA cycles.
  To wait reset complete, loop operation of R_BSP_SoftwareDelay() is used.

- Release reset EPTPC
  Setting “0” RESET bit of the PTRSTR register.

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
/* ==== Ether communication setting ==== */
#define LINK_CH (1) /* 0 or 1(default) */
#define NUM_CH  (2) /* The number of active channel */

/* ==== MAC address ==== */
/* Please change usr own vendor ID */
/* Followings are applied to Renesas vendor ID (= 74-90-50) as sample data */
#define MAC_ADDR_1H (0x00007490)
#define MAC_ADDR_1L (0x50007934)
#define MAC_ADDR_2H (0x00007490)
#define MAC_ADDR_2L (0x50007935)
static uint32_t mac_addr[2][2] = {{MAC_ADDR_1H, MAC_ADDR_1L},{MAC_ADDR_2H,
MAC_ADDR_2L}};

/* ==== IP address ==== */
#define IP_ADDR_1 (0x66676869)
#define IP_ADDR_2 (0x76777879)
static uint32_t ip_addr[2] = {IP_ADDR_1, IP_ADDR_2};

/* ==== MAC and IP address ==== */
static uint32_t my_mac_addr[2][2] = {{MAC_ADDR_1H, MAC_ADDR_1L},{MAC_ADDR_2H,
MAC_ADDR_2L}};
static uint32_t my_ip_addr[2] = {IP_ADDR_1, IP_ADDR_2};
static uint32_t ether_ch[] = {ETHER_CHANNEL_0, ETHER_CHANNEL_1};

/* ==== PTP message reception filter values ==== */
static uint32_t fil1 = 0x00000000; /* SYRFL1R */
```

```
static uint32_t fil2 = 0x00000000; /* SYREL2R */

/* ==== Standard frame received interrupt handler ==== */
extern void EINT_Trig_isr(void *ectrl);

#include "r_ptp_light_rx_if.h"
#include "r_ether_if.h"

    int32_t ch;
    ether_return_t ether_ret;    /* Ether driver return code */
    ether_param_t ectrl;         /* EDMAC and ETHERC control */
    ether_cb_t ecbt;             /* EDMAC callback function structure */
    ptpl_return_t ptpl_ret;      /* PTP light driver return code */

    PTPLConfig ptplc;            /* PTP part configuration structure */
    ether_promiscuous_t pcuous; /* promiscuous control */

    /* Initialize PTP configuration */
    memset(&ptplc, 0, sizeof(ptplc));

    for (ch = 0;ch < NUM_CH;ch++)
    { /* set mac address */
        ptplc.port[ch].macAddr[0] = (uint8_t)(my_mac_addr[ch][0] >> 8u);
        ptplc.port[ch].macAddr[1] = (uint8_t)(my_mac_addr[ch][0]);
        ptplc.port[ch].macAddr[2] = (uint8_t)(my_mac_addr[ch][1] >> 24u);
        ptplc.port[ch].macAddr[3] = (uint8_t)(my_mac_addr[ch][1] >> 16u);
        ptplc.port[ch].macAddr[4] = (uint8_t)(my_mac_addr[ch][1] >> 8u);
        ptplc.port[ch].macAddr[5] = (uint8_t)(my_mac_addr[ch][1]);

        /* set IP address */
        ptplc.port[ch].ipAddr[0] = (uint8_t)(my_ip_addr[ch] >> 24u);
        ptplc.port[ch].ipAddr[1] = (uint8_t)(my_ip_addr[ch] >> 16u);
        ptplc.port[ch].ipAddr[2] = (uint8_t)(my_ip_addr[ch] >> 8u);
        ptplc.port[ch].ipAddr[3] = (uint8_t)(my_ip_addr[ch]);
    }

    /* Initialize resources of the Ether driver */
    R_ETHER_Initial();

    /* Register trigger packet received event to EDMAC interrupt handler */
    ecbt.pcb_int_hnd = EINT_Trig_isr;
    ectrl. ether_int_hnd = ecbt;
    R_ETHER_Control(CONTROL_SET_INT_HANDLER, ectrl);

/* ==== Open standard Ether ==== */
#if (1 == LINK_CH)
    for (ch = LINK_CH;ch > (LINK_CH - NUM_CH);ch--)
#else  /* (0 == LINK_CH) */
    for (ch = 0;ch < NUM_CH;ch++)
#endif
    { /* Power on ether channel */
        ectrl.channel = ether_ch[ch];
        R_ETHER_Control(CONTROL_POWER_ON, ectrl);

        /* Initialize EDMAC interface and peripheral modules */
        ether_ret = R_ETHER_Open_ZC2(ch, (const uint8_t*)ptplc.port[ch].macAddr,
ETHER_FLAG_OFF);

        if (ETHER_SUCCESS != ether_ret)
        {
            goto Err_End;
```

```
        }
    }

    /* ==== Set PTP configuration ==== */
    /* Reset EPTPC */
    R_PTPL_Reset();

    /* Initialize EPTPC */
    ptpl_ret = R_PTPL_Init(&ptplc);
    if (PTPL_OK != ptpl_ret)
    {
       goto Err_End;
    }

#if (LINK_CH == 1)
    for (ch = LINK_CH;ch > (LINK_CH - NUM_CH);ch--)
#else  /* (LINK_CH == 0) */
    for (ch = 0;ch < NUM_CH;ch++)
#endif
    {
       /* Set promiscuous mode */
       pcuous.channel = ch;
       pcuous.bit = ETHER_PROMISCUOUS_ON;
       ectrl.p_ether_promiscuous = &pcuous;
       R_ETHER_Control(CONTROL_SET_PROMISCUOUS_MODE, ectrl);

       /* Clear extended promiscuous mode */
       R_PTPL_SetExtPromiscuous(ch, false);

       /* Set frame filter */
       R_PTPL_SetSyncConfig(ch, &fil1, &fil2, NULL, NULL);
    }

    /* ==== Link standard Ether ==== */
#if (LINK_CH == 1)
    for (ch = LINK_CH;ch > (LINK_CH - NUM_CH);ch--)
#else  /* (LINK_CH == 0) */
    for (ch = 0;ch < NUM_CH;ch++)
#endif
    {
        while (1)
        { /* Check EDMAC Host interface status */
           ether_ret = R_ETHER_CheckLink_ZC(ch);
           if (ETHER_SUCCESS == ether_ret)
           {
               break;
           }
       }

       /* Set EDMAC interface to transfer standard Ethernet frame */
       R_ETHER_LinkProcess(ch);
    }

    /* Start user operation */
```

Special Notes

This function is usually executed in the beginning of initialization sequence and recovered from any error.

If you access CH0 of the standard Ethernet MAC first[1] (before access CH1), you need to change the configuration defined in the “r_ether_rx_config.h” of “RX Family Ethernet Module Using Firmware Integration Technology [3]”(= r_ether_rx) from default setting.

[1] In this usage example, define “LINK_CH = 0”.

#define ETHER_CFG_CH0_PHY_ACCESS (0) /* default (1) */

#define ETHER_CFG_CH1_PHY_ACCESS (0) /* default (1) */

3.3 R_PTPL_SetTran ()

This function set inter ports transfer mode.

Format

ptpl_return_t R_PTPL_SetTran(TranMode *mode, RelEnabDir *dir);

Parameters

mode – Inter ports transfer mode.

dir - Relay enable directions (bit map form).

Return Values

PTPL_OK: Processing completed successfully
PTPL_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_light_rx_if.h”.

Description

This function set inter ports transfer mode. The following operations are executed.

- Set/clear relay enable directions
  CH0 to CH1 transfer enable or disable.
  CH1 to CH0 transfer enable or disable.

- Set transfer mode
  Select Store & forward or cut-through mode.

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptp_light_rx_if.h"
#include "r_ether_if.h"

/* reception frame MAC address: 01:00:5E:00:01:02 */
#define MAC_ADDR_H (0x00000100)
#define MAC_ADDR_L (0x5E000102)

   ptpl_return_t ret; /* PTP light driver return code */
   TranMode mode; /* Inter ports transfer mode */
   RelEnabDir dir; /* Relay direction */
   uint32_t fmac[2];

   /* Reset EPTPC */
   R_PTPL_Reset();

   /* ==== Clear extended promiscuous mode: CH0 ==== */
   R_PTPL_SetExtPromiscuous(0, false);

   /* Set transfer mode to cut-through mode */
   mode = CT_THR;

   /* Set relay enable both directions */
   dir = ENAB_BT;

   /* ==== Set inter ports transfer mode ==== */
   ret = R_PTPL_SetTran(&mode, &dir);
   if (PTPL_OK != ret)
```

```
{
    goto Err_end; /* error */
}

/* Set reception frame MAC address */
fmac[0] = ((MAC_ADDR_H << 8u) | (MAC_ADDR_L >> 24u));
fmac[1] = (MAC_ADDR_L & 0x00FFFFFF);

/* ==== Set Multicast (MC) frames filter (FFLTR): CH0 ==== */
/* SYNFP CH0, only receive FMAC1R(U/L) and update FMAC1R(U/L) */
ret = R_PTPL_SetMCFilter(0, MC_REC_REG1, fmac);
if (PTPL_OK != ret)
{
    goto Err_end; /* error */
}

/* Thereafter, frame propagates both directions and cut-through method */
/* Only receives FMAC1R(U/L) registered address frame */
```

Special Notes

This function is valid only the standard Ethernet frames. (not valid for PTP message frames)

If the argument (mode or dir) is NULL pointer, the value does not set.

3.4 R_PTPL_SetMCFilter ()

This function set multicast frames (MC) filter (FFLTR).

Format

ptpl_return_t R_PTPL_SetMCFilter(uint8_t ch, MCRecFil fil, uint32_t *fmac);

Parameters

ch – Sync unit channel (SYNFP0 or SYNFP1).

fil - Multicast(MC) frames filter setting.

fmac - Reception frame MAC address (register FMAC0R(U/L) or FMAC1R(U/L)).

Return Values

PTPL_OK: Processing completed successfully
PTPL_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_light_rx_if.h".

Description

This function set MC filter (FFLTR). The following operations are executed.

- Set reception mode
  All frames receive, no frame receives or only registered frame receive.

- Update reception frame register
  If *fmac* is set, update FMAC0R (U/L) or FMAC1R (U/L) depend on the argument of *fil*.

Reentrant

Function is not reentrant.

Example

Example is same as “3.3 R_PTPL_SetTran”.

Special Notes

This function is valid only the standard Ethernet frames. (not valid for PTP message frames) .

As for the PTP message frames, use the “3.9 R_PTPL_SetSyncConfig” function.

The MC filter does not relevant in the extended promiscuous mode. In case of applying the MC filter, please clear the extended promiscuous mode showed as the usage example of “3.3 R_PTPL_SetTran”.

If 3rd argument (fmac) is NULL pointer, neither FMAC0R (U/L) nor FMAC1R (U/L) is not updated.

3.5 R_PTPL_SetExtPromiscuous ()

This function set/clears extended promiscuous mode.

Format

ptpl_return_t R_PTPL_SetExtPromiscuous(uint8_t ch, bool is_set);

Parameters

ch - Sync unit channel (SYNFP0 or SYNFP1).

is_set - (true): set, (false): clear.

Return Values

PTPL_OK: Processing completed successfully
PTPL_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_light_rx_if.h".

Description

This function set/clears extended promiscuous mode.

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptp_light_rx_if.h"
#include "r_ether_if.h"

   /* Set full functionality used case */
   int32_t ch; /* Ethernet and SYNC unit channel */
   ether_promiscuous_t pcuous; /* promiscuous control */
   ether_param_t ectrl; /* ETHERC control */

   for (ch = 0; ch < 2; ch++)
   {
      /* ==== Set promiscuous mode (CH0 and CH1) ==== */
      pcuous.channel = ch;
      pcuous.bit = ETHER_PROMISCUOUS_ON;
      ectrl. ether_promiscuous_t = &pcuous;
      R_ETHER_Contorl(CONTROL_SET_PROMISCUOUS_MODE, ectrl);

      /* ==== Clear extended promiscuous mode (CH0 and CH1) ==== */
      R_PTPL_SetExtPromiscuous(ch, false);
   }

   /* Thereafter, frame operations are followed */
   /* - Unicast
        coincidence address is EDMAC and PRC-TC,
        and other frames is only PRC-TC
      - Multicast
        depends on multicast frame filter
      - Broadcast
        EDMAC and Packet relation control */
```

Special Notes

The result of this function setting (extended promiscuous mode setting), please refer to Sec 4.4.

3.6 R_PTPL_Init ()

This function initializes EPTPC depends on the device configuration.

Format

ptp_return_t R_PTPL_Init(PTPLConfig *tbl);

Parameters

tbl – PTP part configuration table.

Return Values

PTPL_OK: Processing completed successfully
PTPL_ERR: Any error occurred

Properties

Prototyped in “r_ptp_light_rx_if.h".

Description

This function initializes EPTPC depends on the device configuration. The following operations are executed.

- Case of RX72M device, disable bypass function.

- Set MAC and IP address

- Initialize PTP reception filters (SYRFL1R and SYRFL2R)

- Validate set values to SYNFP0 and SYNFP1.

- Initialize packet relation controller unit (PRC-TC)

- Clear the interrupt status and mask of EPTPC

- Set EPTPC interrupt
  Call ptp_dev_start().

Reentrant

Function is not reentrant.

Example

This example is same as “3.2 R_PTPL_Reset”.

Special Notes

This function usually executes only once after the ETHERC and standard EDMAC were opened.

To set parameter for each channel, “for” statement (loop processing) is used.

3.7 R_PTPL_RegMINTHndr ()

This function set MINT interrupt and registers a user function to MINT interrupt handler of EPTPC.

Format

void R_PTPL_RegMINTHndr(MINT_Reg reg, unit32_t event, MINT_HNDLR func);

Parameters

reg - MINT interrupt register.
MINT_FUNC_PRC: Interrupt from PRC-TC
MINT_FUNC_SYN0: Interrupt from SYNFP0
MINT_FUNC_SYN1: Interrupt from SYNFP1
event - interrupt elements.
func - register function.

Return Values

None

Properties

Prototyped in “r_ptp_light_rx_if.h”.

Description

This function set MINT interrupt and registers a user function to MINT interrupt handler of EPTPC. Following operations are executed.

- If the event is interrupt from PRC-TC, register the user function called from PRC-TC interrupt event.

- If the event is interrupt from SYNFP0/1, register the user function called from SYNFP0/1 interrupt event.

- Set or clear the MINT interrupt.

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptp_light_rx_if.h"

   /* user MINT handler function */
   void user_mint_func(uint32_t reg);

   /* register user function called by INTCHG (logMessageInterval updated)
event of SYNFP0 */
   R_PTPL_RegMINTHndr(MINT_FUNC_SYN0, 0x00000002, (MINT_HNDLR)user_mint_func);

   /* wait interrupt from SYNFP0 */

   /* interrupt occurred (call user_mint_func) */

   /* Update transmission interval of Delay_Req message to suitable interval */

   /* release registered user read PTP message function */
   R_PTPL_RegMINTHndr(MINT_FUNC_SYN0, 0x00000002, (MINT_HNDLR)NULL);

   return;
```

Special Notes

Registered user function is updated if register the same MINT interrupt handler.

If 3rd argument(=func) is NULL, registered function is removed and the MINT interrupt is disabled.

3.8 R_PTPL_GetSyncConfig ()

This function get PTP frame configuration (SYRFL1R, SYRFL2R, SYTRENR and SYCONFR).

Format

ptpl_return_t R_PTPL_GetSyncConfig(uint8_t ch, uint32_t *fil1, uint32_t *fil2, uint32_t *tren, uint32_t *conf);

Parameters

ch - Sync unit channel (SYNFP0 or SYNFP1).

fil1 - SYRFL1R.

fil2 - SYRFL2R.

tren[1] *– SYTRENR.*

conf[1] *– SYCONFR.*

[1]Those arguments are irrelevant to the standard Ethernet.

Return Values

PTPL_OK: Processing completed successfully
PTPL_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_light_rx_if.h".

Description

This function gets PTP frame control configuration (SYRFL1R, SYRFL2R, SYTRENR and SYCONFR). The following operations are executed.

- Get PTP messages reception filter 1 (SYRFL1R)
  Announce, Sync, Follow_Up, Delay_Req, Delay_Resp, Pdelay_Req, Pdelay_Resp and Pdelay_Resp_Follow_Up messages filter setting.

- Get PTP messages reception filter 2 (SYRFL2R)
  Management, Signaling and Illegal (un-defined) messages filter setting.

- Get PTP messages transmission enable (SYTRENR)
  Announce, Sync, Delay_Req and Pdelay_Req messages transmission setting.

- Get SYCONFR
  TC mode (E2E TC or P2P TC) and transmission interval setting.

Reentrant

Function is reentrant.

Example

Example showing this function being used.

```
#include <stdio.h>
#include "r_ptp_light_rx_if.h"

    ptpl_return_t ret; /* PTP light driver return code */
    uint32_t fil1; /* Current SYRFL1R (CH0) value */
    uint32_t fil2; /* Current SYRFL2R (CH0) value */

    /* Get SYRFL1R, SYRFL2R and SYTRENR from SYNFP0. (Not get the SYCONFR) */
    ret = R_PTPL_GetSyncConfig(0, &fil1, &fil2, NULL, NULL);
    if (PTPL_OK != ret)
    {
        goto Err_end; /* error */
    }
```

```
    printf("SYRFL1R (CH0) value = %8x\n", fil1);
    printf("SYRFL2R (CH0) value = %8x\n", fil2);
```

Special Notes

If the argument (fil1, fil2, tren or conf) is NULL pointer, the argument value does not get.

3.9 R_PTPL_SetSyncConfig ()

This function set PTP frame control configuration (SYRFL1R, SYRFL2R, SYTRENR and SYCONFR).

Format

ptpl_return_t R_PTPL_SetSyncConfig(uint8_t ch, uint32_t *fil1, uint32_t *fil2, uint32_t *tren, uint32_t *conf);

Parameters

ch - Sync unit channel (SYNFP0 or SYNFP1).

fil1 - SYRFL1R.

fil2 - SYRFL2R.

tren[1] *– SYTRENR.*

conf[1] *– SYCONFR.*

[1]Those arguments are irrelevant to the standard Ethernet.

Return Values

PTPL_OK: Processing completed successfully
PTPL_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_light_rx_if.h".

Description

This function set PTP synchronous configuration (SYRFL1R, SYRFL2R, SYTRENR and SYCONFR). The following operations are executed.

- Set PTP messages reception filter 1 (SYRFL1R)
  Announce, Sync, Follow_Up, Delay_Req, Delay_Resp, Pdelay_Req, Pdelay_Resp and Pdelay_Resp_Follow_Up messages filter setting.

- Set PTP messages reception filter 2 (SYRFL2R)
  Management, Signaling and Illegal (un-defined) messages filter setting.

- Set PTP messages transmission enable (SYTRENR)
  Announce, Sync, Delay_Req and Pdelay_Req messages transmission setting.

- Set SYCONFR
  TC mode (E2E TC or P2P TC) and transmission interval setting.

- Validate set values to SYNFP0 or SYNFP1

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptp_light_rx_if.h"

   ptpl_return_t ret; /* PTP light driver return code */
   uint32_t fil1; /* SYRFL1R setting value, 0:SYNFP0 1:SYNFP1 */
   uint32_t fil2; /* SYRFL2R setting value, 0:SYNFP0 1:SYNFP1 */

   /* Transfer all PTP messages to the other channel */
   fil1 = 0x22222222;

   fil2 = 0x00000022;
```

```
   /* Set fil1_val and fil2_val to SYNFP1. (Neither set the SYTRENR nor
SYCONFR) */
   ret = R_PTPL_SetSyncConfig(1, &fil1, &fil2, NULL, NULL);
   if (PTPL_OK != ret)
   {
      goto Err_end; /* error */
   }

   /* Complete set synchronous configuration */
```

Special Notes

The relay enable bits (bit[4N+1] (N=1, 2,,, 7)) only can be set[1] and the other bits should be clear.

If the argument (fil1, fil2, tren or conf) is NULL pointer, the value does not set.

[1] This driver only supports relay the PTP message frame.

3.10 R_PTPL_SetInterrupt ()

This function enables EPTPC INFABT interrupt.

Format

ptpl_return_t R_PTPL_SetInterrupt(uint8_t ch);

Parameters

ch - Sync unit channel (SYNFP0 or SYNFP1).

Return Values

PTPL_OK: Processing completed successfully
PTPL_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_light_rx_if.h".

Description

This function enables EPTPC INFABT interrupt. The following operations are executed.

- Enable SYNFP interrupt to set EPTPC.MIEIPR register for each channel.

- Enable SYNFP interrupt to set EPTPC0/1.SYIPR.INFABT bit for each channel.

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptp_light_rx_if.h"
#include "r_ether_if.h"

   ptpl_return_t ret;   /* PTP light driver return code */
   bool is_det;         /* INFABT interrupt detection flag */

   /* Standard Ethernet open and link were completed */

   /* PTP open was completed */

   /* Enable EPTPC INFABT interrupt CH0 */
   ret = R_PTPL_SetInterrupt(0);
   if (PTPL_OK != ret)
   {
      goto Err_end; /* error */
   }

   while(1)
   {
       ret = R_PTPL_ChkInterrupt(0, &is_det);
       if (PTPL_OK != ret)
       {
          goto Err_end; /* error */
       }

       /* Check INFABT error */
       if (true == is_det)
       { /* INFABT error detected */
          /* stop standard Ether */
          R_ETHER_Close_ZC2(0);

          /* Reset EPTPC */
          R_PTPL_Reset();
```

```
            /* Clear INFABT interrupt flag */
            R_PTPL_ClrInterrupt(0);

            /* Thereafter, please execute retrieve operation */

        }
    }
```

Special Notes
None

3.11 R_PTPL_ChkInterrupt ()

This function checks INFABT interrupt occurrence.

Format

ptpl_return_t R_PTPL_ChkInterrupt(uint8_t ch, bool *is_det);

Parameters

ch - Sync unit channel (SYNFP0 or SYNFP1).

is_det – INFABT interrupt flag.

Return Values

PTPL_OK: Processing completed successfully
PTPL_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_light_rx_if.h".

Description

This function checks INFABT interrupt occurred or not?

Reentrant

Function is reentrant.

Example

Example showing this function being used. Example is same as “3.10 R_PTPL_SetInterrupt”.

Special Notes

None

3.12 R_PTPL_ClrInterrupt ()

This function clears INFABT interrupt occurrence flag.

Format

ptpl_return_t R_PTPL_ClrInterrupt (uint8_t ch);

Parameters

ch - Sync unit channel (SYNFP0 or SYNFP1).

Return Values

PTPL_OK: Processing completed successfully
PTPL_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_light_rx_if.h".

Description

This function clears INFABT interrupt occurrence flag.

Reentrant

Function is not reentrant.

Example

Example showing this function being used. Example is same as “3.10 R_PTPL_SetInterrupt”.

Special Notes

None

4. Appendices

4.1 Internal Functions

The PTP light driver calls the internal functions in their operations. The summary of the internal functions shows Table 4.1.

Table 4.1 Internal Functions

Item	Contents
_R_PTPL_Int_Syn0()	Set INFABT interrupt occurrence flag of SYNFP0.
_R_PTPL_Int_Syn1()	Set INFABT interrupt occurrence flag of SYNFP1.
_R_PTPL_Init_SYNFP()	Set SYNFP parameters by initial values.
_R_PTPL_Init_PRC()	Set PRC-TC parameters by initial values.

4.2 Related Ether Driver's API

The PTP light driver always should be used with the Ether driver. As for the information of the Ether driver's API to which the PTP light driver mandatory use, please refer to "RX Family Ethernet Module Using Firmware Integration Technology [3]". The typical usage of those API, describes Sec 3 as the example.

4.3 Additional Standard Ethernet Functionalities

The PTP light driver implements new functionalities but for the time synchronization based on the PTP. Those functionalities are simple switch[1] and MC (Multicast) frame filter. 3.3 R_PTPL_SetTran function sets the Simple switch and 3.4 R_PTPL_SetMCFilter function sets the MC frame filter. Summary of those functionalities shows followed.

[1] It means the inter ports transfer by hardware operation.

4.3.1 Simple switch (implemented PRC-TC part)

Store & forward or cut-throw transfer method is selectable. If cut-throw method is applied to the daisy chain topology which is common industrial network, it can reduce the inter ports transfer delay.
It is also possible to use as two independent networks by the network isolating function.

4.3.2 Multicast frame filter (implemented SYNFP0 and SYNFP1 part)

It is possible to enhance the total performance due to cancel irrelevant multicast frames. Even if the filter is set enabled, specific two frames can be received.
For PTP frames, the other PTP specific filters (SYRFL1R/2R) are implemented. As for those filters setting, please refer to 3.9 R_PTPL_SetSyncConfig function.

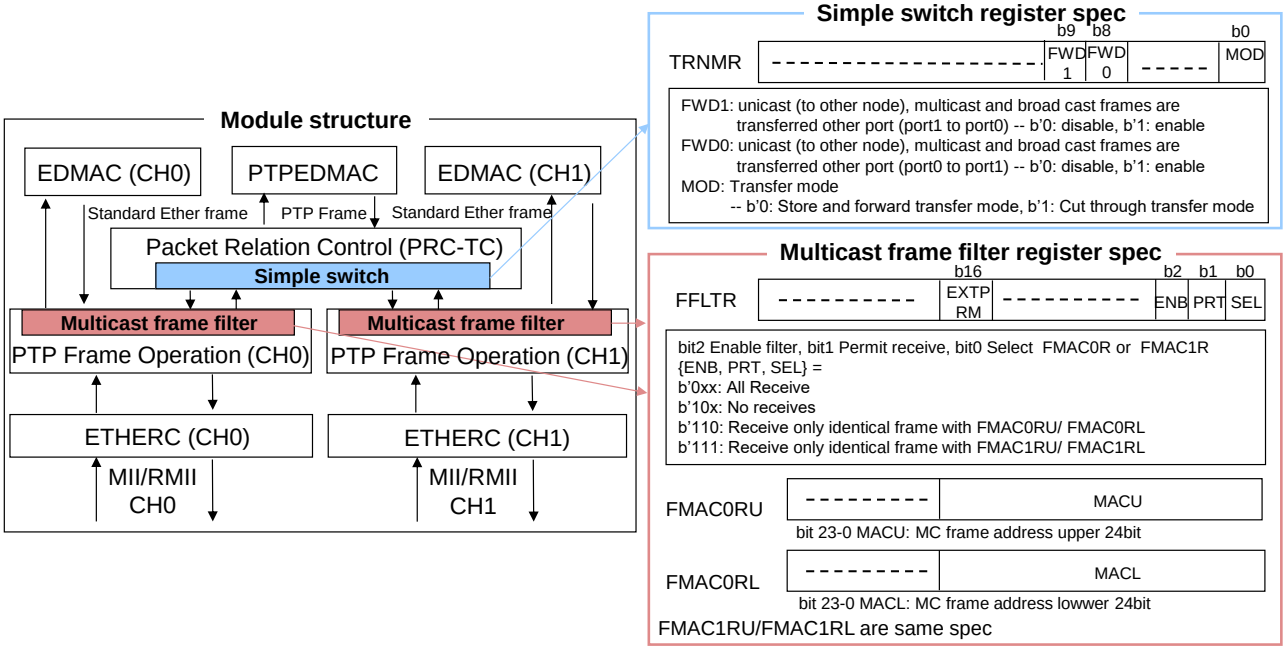
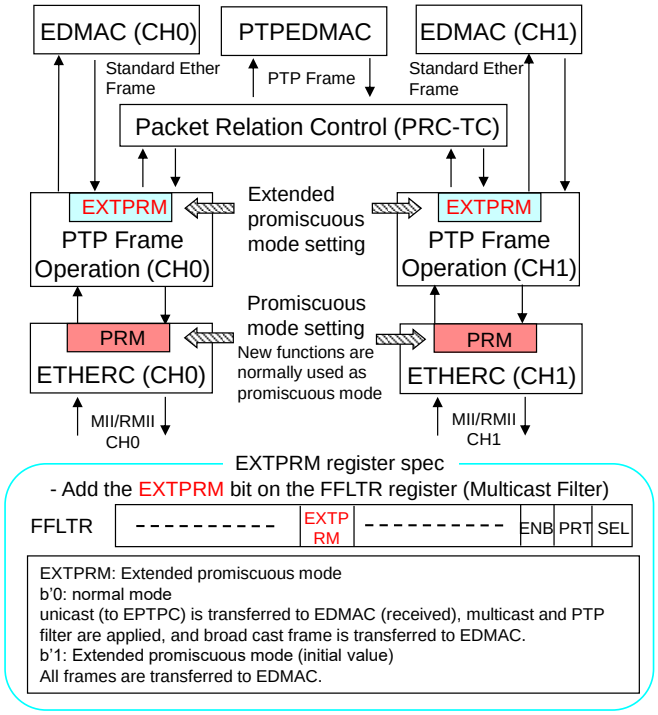


Figure 4.1 Additional standard Ethernet functionalities

4.4 Compatibility with Existing Devices

The Ethernet modules of RX64M/71M keep compatibility with existing devices of the Renesas product such as SH7216, RX63N and so on. The compatibility is supported combination with promiscuous mode (PRM) and extended promiscuous mode (EXTPRM). The promiscuous mode can be set using the R_ETHER_Control function with control code equal to “CONTROL_SET_PROMISCUOUS_MODE”, in detail refer to 3.5 R_PTPL_SetExtPromiscuous. As for the setting of those modes, please refer to “RX Family Ethernet Module Using Firmware Integration Technology [3]”. The promiscuous mode can be set using the R_PTPL_SetExtPromiscuous API described in the Sec 3. The result of those settings shows Figure 4.2.



Frame propagation list depends on the register setting.

b’1: set, b’0 clear

[1]Omits the PTP frame case

PRM	EXTPRM	Frame type	Destination
b’0	b’0	unicast	coincidence address: EDMAC others: canceled
		multicast	depends on the multicast frame filter
		broadcast	EDMAC and PRC-TC
b’0	b’1	unicast	coincidence address: EDMAC others: canceled
		multicast	EDMAC
		broadcast	EDMAC
b’1	b’0	unicast	coincidence address: EDMAC others: PRC-TC
		multicast	depends on the multicast frame filter
		broadcast	EDMAC and PRC-TC
b’1	b’1	unicast	coincidence address: EDMAC others: EDMAC
		multicast	EDMAC
		broadcast	EDMAC

Initial value: full compatible existing devices (ex.SH7216, RX63N)

Full functionality used case

Figure 4.2 Promiscuous and extended promiscuous mode setting

4.5 Confirmed Operation Environment

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

Table 4.2 Confirmed Operation Environment (Rev1.12)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version V7.2.0 IAR Embedded Workbench for Renesas RX 4.11.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201801 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.11.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.1.12
Board used	Renesas Starter Kit+ for RX64M Renesas Starter Kit+ for RX71M

Table 4.3 Confirmed Operation Environment (Rev1.13)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version V7.2.0 IAR Embedded Workbench for Renesas RX 4.12.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201801 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.1.13
Board used	Renesas Starter Kit+ for RX64M Renesas Starter Kit+ for RX71M Renesas Starter Kit+ for RX72M

Table 4.4 Confirmed Operation Environment (Rev1.14)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version V7.5.0 Renesas Electronics e^2 studio Version V7.4.0 (RX72N only) IAR Embedded Workbench for Renesas RX 4.12.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.1.14
Board used	Renesas Starter Kit+ for RX64M Renesas Starter Kit+ for RX71M Renesas Starter Kit+ for RX72M

4.6 Troubleshooting

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

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

- Using CS+:

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

- Using e^2 studio:

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

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

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

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

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

A: The setting in the file “r_ptp_light_rx_config.h” may be wrong. Check the file “r_ptp_light_rx_config.h”. If there is a wrong setting, set the correct value for that. Refer to 2.9 Configuration Overview for details.

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.1.10 (R01UH0377EJ)
RX71M Group User's Manual: Hardware Rev.1.10 (R01UH0493EJ)
RX72M Group User's Manual: Hardware Rev.1.00 (R01UH0804EJ)
RX72N Group User's Manual: Hardware Rev.1.00 (R01UH0824EJ)
The latest versions 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 CC-RX User's Manual (R20UT3248)
The latest information can be downloaded from the Renesas Electronics website.

Related Technical Updates

This module reflects the content of the following technical updates.

TN-RX*-A125A/E

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Oct 30, 2015	**—**	First edition issued.
1.10	Mar 31, 2016	**—**	Data structures changed.
1.11	Nov 11, 2016	**11**	Corrected the internal operation of getting version function.
		30	Corrected frame propagation list (Fig 4.2).
1.12	Jul 31, 2019	**—**	Added support for GNUC and ICCRX.
		—	Changed MINT interrupt handler operation.
		6, 22	Added registering user function to MINT interrupt handler.
		7	Added the explanation of using interrupt vector.
		9	Added MINT_Reg constant type.
		9	Added the explanation of callback function.
		10	Updated code size and adapted to GNUC and ICCRX.
		10	Updated how to add the FIT module to your project.
		11	Added the explanation about comment of loop operation.
		13	Changed using R_BSP_SoftwareDelay function to wait reset completion.
		13	Added reset release waiting operation in the R_PTPL_Reset function.
		21	Added the “WAIT_LOOP” comments of parameter setting for each channel.
		33	Added “Confirmed Operation Environment” section.
		33	Added “Troubleshooting” section.
1.13	Aug 31, 2019	**—**	Supported RX72M device.
		—	Added Bypass setting.
1.14	Nov 30, 2019	**—**	Supported RX72N device.

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

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

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

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

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

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

Notice

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

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

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

(Rev.4.0-1 November 2017)

Corporate Headquarters

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

Trademarks

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

Contact information

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