RX Family

EPTPC Module Firmware Integration Technology

Introduction

This document explains a PTP software driver (PTP driver, EPTPC FIT module) based on the firmware integration technology (FIT). The PTP driver makes the time synchronization based on the PTP (Precision Time Protocol) defined by the IEEE1588-2008 specification [1].

Target Device

This API supports the following device.

- RX64M Group
- RX71M Group
- RX72M 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 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 FIT Module

This document explains a PTP software driver (hereafter PTP driver) based on the firmware integration technology (FIT) and shows the usage example. The PTP driver makes the time synchronization using the PTP function of the EPTPC peripheral module (EPTPC). The PTP driver also supports enhanced functions of the standard Ethernet such as the simple switch and multicast frame filter functions.

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 Ethernet Module Using Firmware Integration Technology, Rev.1.17, Document No. R01AN2009EJ0117, to be released.

[3] RX Family PTP Get Synchronous Time Using Firmware Integration Technology Modules, Rev.1.12, Document No. R01AN1983EJ0112, Mar 31, 2017

[4] RX Family PTP Timer Synchronous Start Using Firmware Integration Technology Modules, Rev.1.12, Document No. R01AN1984EJ0112, Mar 31, 2017

[5] RX Family PTP Synchronous Pulse Output Using Firmware Integration Technology Modules, Rev.1.12, Document No. R01AN2846EJ0112, Mar 31, 2017

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

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

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

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

1.4 Terms and Abbreviations

- IEEE1588

Specification makes the time synchronization in a communication network. In general, the communication network is specified the Ethernet. There are two versions which are IEEE1588-2002 (version1) and IEEE1588-2008 (version2), they do not have complete compatibilities each other. This document only attributes the IEEE1588-2008 (version2).

- PTP (Precision Time Protocol)

PTP means time synchronize protocol based on the IEEE1588.

- PTP message

The data format which is used in the PTP sequence. PTP messages are transmitted in the Ethernet frame (Layer2) or UDP packet (Layer3).

- Clock (Node)

Device whose functionality is time synchronization based on the IEEE1588.

- Local clock

Synchronize time of the each clock.

- Master

Master means the Clock issues the system standard time to other clocks.

- Slave

Slave means the Clock receives and corrects the system standard time to other clocks.

- OC (Ordinary Clock)

OC means the Clock has only one port and one local clock.

- BC (Boundary Clock)

BC means the Clock has more than two ports and common unique local clock. Each port has time synchronize function.

- TC (Transparent Clock)

TC means the Clock has more than two ports and corrects the frame propagation delay between ingress and egress ports.

- E2E (End to End)

Synchronize mode in which between a Master and multiple Slaves (or a Slave).

- P2P (Peer to Peer)

Synchronize mode in which between the specific two clocks.

- STCA (Statistical Time Correction Algorithm)

Correct offsetFromMaster[1] applied to statistical method to which estimates the tendencies of clock (time) deviation from the gradient calculated using sampled clock values with (worst-10 filter).

- BMC (Best Master Clock) algorithm

BMC algorithm determines the suitable master in the domain and composes of the data set comparison algorithm and the state decision one. Data set comparison algorithm decides which port[2] is better as master comparing the feature of each clock. State decision algorithm decides the next state of the port as the result of the data set comparison algorithm.

[1] Time difference between time on the Master and time on the Slave (refer to [1]).

[2] One port of the clock. If clock has only one port, port equals to clock.

1.5 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.

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

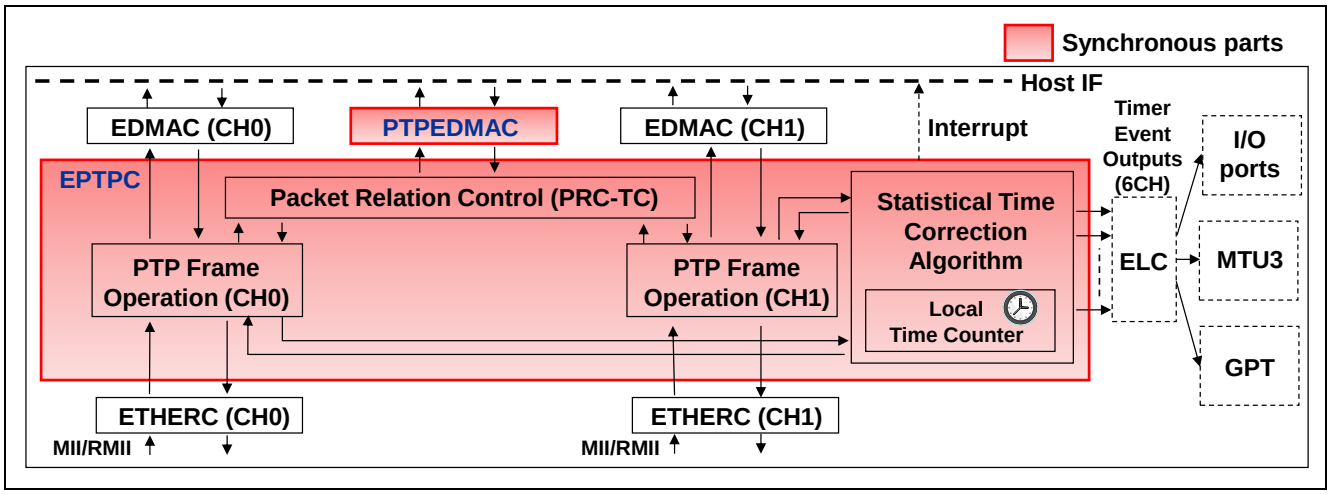


Figure 1.1 Hardware block diagram

1.6 Software Structure

The PTP driver always should be used with Ether drivers [2] and need to be combination with TCP/IP middle ware in case of applied to TCP/IP system. The PTP driver is also used with motor MTU3/GPT driver (or I/O ports driver) and an ELC driver in case of applied to motor control system (or PWM pulse output system). Figure 1.2 shows the typical structure and functional overview of the software.

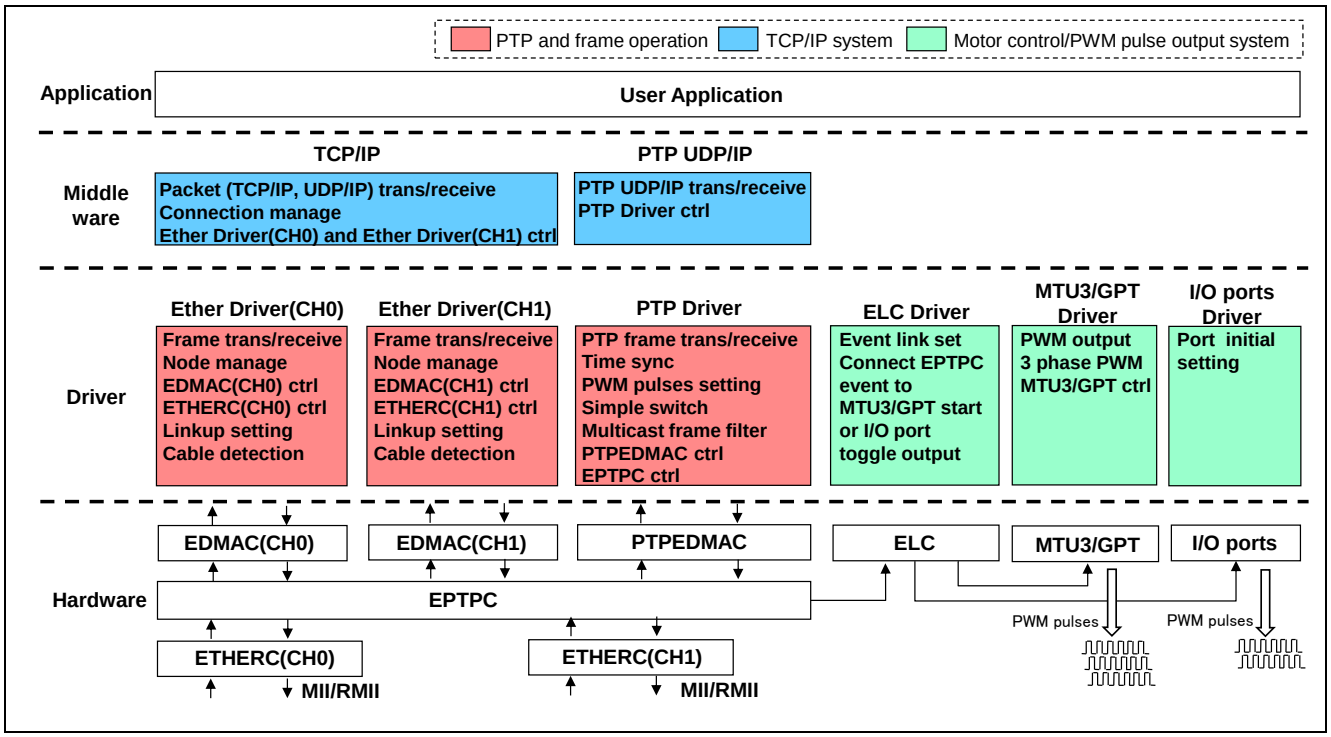


Figure 1.2 Software structure system example

1.7 File Structure

The PTP driver is composed of a PTP Host interface part and a PTP synchronization part. The PTP Host interface part transmits and receives PTP messages via the PTPEDMAC and those operation codes are described in the “r_ptpif.c”. The PTP part operates the time synchronization based on the PTP using the EPTPC and those operation codes are described in the “r_ptp.c”.

1.8 API Overview

The API functions of the PTP Host interface part shows Table 1.1 and the API functions PTP synchronization part shows Table 1.2.

Table 1.1 API Functions (PTP Host interface part)

Item	Contents
R_PTPIF_GetVersion()	Get PTP Host interface driver part version number.
R_PTPIF_Reset()	Reset PTPEDMAC.
R_PTPIF_Init()	Initialize resources of the PTP Host interface driver.
R_PTPIF_Open_ZC2()	Initialize PTP Host interface and peripheral modules.
R_PTPIF_LinkProcess()	Set PTP Host interface to transfer PTP messages.
R_PTPIF_CheckLink_ZC()	Check PTP Host interface communication status.
R_PTPIF_Close_ZC2()	Disable PTP Host interface peripheral modules.
R_PTPIF_Read()	Receive PTP message.
R_PTPIF_Write()	Transmit PTP message or standard Ethernet frame.
R_PTPIF_Read_ZC2()	Receive one PTP message frame or fragment. Set the allocated buffer pointer for received data.
R_PTPIF_Read_ZC2_BufRelease()	Release the receive buffer.
R_PTPIF_Write_ZC2_GetBuf()	Set the buffer pointer and allocate the buffer by transmit data size.
R_PTPIF_Write_ZC2_SetBuf()	Transmit one PTP message or one Ethernet frame or fragment. Set the descriptor for the transmit data.
R_PTPIF_RegMsgHndr()	Register a user function to the interrupt handler of PTPEDMAC.

Table 1.2 API Functions (PTP synchronization part)

Item	Contents
R_PTP_GetVersion()	Get PTP synchronization driver part version number.
R_PTP_Reset()	Reset EPTPC.
R_PTP_SetTran()	Set inter ports transfer mode
R_PTP_SetMCFilter()	Set Multicast frames (MC) filter (FFLTR).
R_PTP_SetExtPromiscuous()	Set/clear extended promiscuous mode.
R_PTP_Init()	Initialize EPTPC depends on the device configuration.
R_PTP_SubConfig()	Set the optional configuration of EPTPC. Set interval of getting worst10 values.
R_PTP_RegMINTHndr()	Register a user function to MINT interrupt handler of EPTPC.
R_PTP_RegTmrHndr()	Register a user function to timer interrupt handler of EPTPC.
R_PTP_ELC_Ind()	Set/clear ELC interrupt indication.
R_PTP_ELC_SetClr()	Set/clear ELC interrupt auto clear mode.
R_PTP_Tmr_Set()	Set and enable timer interrupt handler (timer event).
R_PTP_GetLcClk()	Get current local clock counter value.
R_PTP_SetLcClk()	Set local clock counter initial value.
R_PTP_ChkW10()	Wait worst10 values got and those values set as gradient limits.
R_PTP_GetW10()	Get current worst10 values.
R_PTP_SetGradLimit()	Set the gradient limit values.
R_PTP_GetMPortID()	Get Master port identity.
R_PTP_SetMPortID()	Set Master port identity.
R_PTP_GetSyncConfig()	Get synchronous configuration (SYRFL1R, SYRFL2R and SYCONFR).
R_PTP_SetSyncConfig()	Set synchronous configuration (SYRFL1R, SYRFL2R and SYCONFR).
R_PTP_GetSyncInfo()	Get current offsetFromMaster and meanPathDelay.
R_PTP_UpdClkID()	Update my clockIdentity.
R_PTP_UpdDomainNum()	Update domainNumber field of common message header.
R_PTP_UpdAnceFlags()	Update Announce message's flag fields.
R_PTP_UpdAnceMsgs()	Update Announce message's message fields.
R_PTP_UpdSyncAnceInterval()	Update transmission interval of Sync and Announce messages.
R_PTP_UpdDelayMsgInterval()	Update transmission interval and timeout values of delay messages.
R_PTP_Start()	Start synchronization.
R_PTP_Stop()	Stop synchronization.
R_PTP_SetPortState()	Set PTP port state.
R_PTP_GetSyncCH()	Get current synchronous channel.
R_PTP_SetInterrupt()	Enable EPTPC INFABT interrupt.
R_PTP_ChkInterrupt()	Check INFABT interrupt occurrence.
R_PTP_ClrInterrupt()	Clear INFABT interrupt occurrence flag.
R_PTP_DisableTmr()	Disable timer event interrupt.
R_PTP_SetSyncDet()	Set detect condition of change of synchronous state.
R_PTP_SetSynctout()	Set detect condition of sync message reception timeout.

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 “PTP Get Synchronous Time [3]”, “PTP Timer Synchronous Start [4]”, “PTP Synchronous Pulse Output [5]”, “Ethernet Simple Switch Function [6]” and “Ethernet Multicast Frame Filter Function [7]”.

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 synchronization based on the PTP and standard Ethernet enhanced function such as 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 kind of Clock (Node). Those resources need to the Ethernet MAC operations.

2.2.3 EDMAC Channel

Those resources need to the CPU Host interface of Ethernet frame operations. The driver uses the EDMAC (CH0), EDMAC (CH1) or both depend on the kind of Clock (Node) when standard frame is transferred. The driver also uses the PTPEDMAC when PTP frame is transferred.

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:

- In case of TC, this driver behaves as the TC only operation. If you want to select the TC & OC combined operation, please reconfigure the transmission and reception setting of PTP messages after starting synchronization.
- BMC operation is not supported.

2.5 Supported Toolchains

This driver has been confirmed to work with the toolchain listed in 4.6 Confirmed Operation Environment.

2.6 Interrupt Vector

The EPTPC MINT interrupt and PTPEDMAC PINT are enabled by executing the R_PTP_Init function and R_PTPIF_Open_ZC2 respectively.

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	GROUPAL1 interrupt (vector no.: 113)
RX71M	● EPTPC MINT interrupt (group interrupt source no.: 0) ● PTPEDMAC PINT interrupt (group interrupt source no.: 1)

2.7 Header Files

All API calls are accessed by including a single file, *r_ptp_rx_if.h* and *r_ptpif_rx_if.h*, which are 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 module are specified in *r_ptp_rx_config.h*. The option names and setting values are listed in the table below.

Configuration options	

`#define PTP_CFG_MODE`
`#define PTP_MODE_CH0 (0x01)`
`#define PTP_MODE_CH1 (0x02)`
`#define PTP_MODE_POLL (0x10)`
`#define PTP_MODE_HWINT (0x20)`
`- Default value = 0x23`
	Specify the PTP synchronization part 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 PTPIF_CFG_MODE`
`#define PTPIF_MODE_CH0 (0x01)`
`#define PTPIF_MODE_CH1 (0x02)`
`#define PTPIF_MODE_POLL (0x10)`
`#define PTPIF_MODE_HWINT (0x20)`
`- Default value = 0x23`
	Specify the PTP Host interface driver mode. Select to 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.
	- 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 PTPIF_CFG_NUM_RX_DESCRIPTORS` `- Default value = 4`	Set the number of Rx descriptors.
`#define PTPIF_CFG_NUM_TX_DESCRIPTORS` `- Default value = 4`	Set the number of Tx descriptors.
`#define PTPIF_CFG_BUFSIZE` `- Default value = 1536`	Set the buffer size of the PTPEDMAC. - Set 32 byte units. Min 64 byte, Max 1536 byte.
`#define PTP_CFG_INTERRUPT_LEVEL` `- Default value = 2`	Specifies interrupt priority levels of EPTPC interrupts. Specify the level between 1 and 15.
`#define PTPIF_CFG_INTERRUPT_LEVEL` `- Default value = 2`	Specifies interrupt priority levels of PTPEDMAC interrupts. Specify the level between 1 and 15.

`#define PTP_CFG_MSG_FORM`
`#define PTP_MSG_FORM_ETH (0x00)`
`#define PTP_MSG_FORM_ETH_8023 (0x01)`
`#define PTP_MSG_FORM_UDP4 (0x02)`
`#define PTP_MSG_FORM_UDP4_8023 (0x03)`
`- Default value = 0`
	Specify the transmit PTP message format[1].
	- When this is set to 0x00, the format is Ethernet II frame.
	- When this is set to 0x01, the format is IEEE802.3 frame. RX71M only supports.
	- When this is set to 0x02, the format is UDP IPv4 Ethernet II frame based packet.
	- When this is set to 0x03, the format is UDP IPv4 IEEE802.3 frame based packet. RX71M only supports.

Configuration options	

`#define PTP_CFG_SYNC_MODE`
`#define PTP_SYNC_MODE1 (0x00)`
`#define PTP_SYNC_MODE2_HW (0x02)`
`#define PTP_SYNC_MODE2_SW (0x03)`
`- Default value = 2`
	Specify the synchronous mode of the PTP driver.
	- When this is set to 0, gradient correction is not applied.
	- When this is set to 2, gradient correction is applied and hardware worst10 setting. This is recommended setting.
	- When this is set to 3, gradient correction is applied and software worst10 setting.

`#define PTP_CFG_SYNC_TIMEOUT` `- Default value = 0x00000000`	Set Sync message reception timeout value. - Set nano second unit. Default value is no timeout detection. If more than 0 value is set, sync message reception timeout can be detected.
`#define NUM_OF_TMR_CHANNEL` `- Default value = 6`	Set total number of timer's channel. - Set 6 (default value) in the RX64M/71M.

`#define PTP_CFG_MTU3_OUTPUT`
`#define MTU3_PWM_OUTPUT_CH0 (0)`
`#define MTU3_PWM_OUTPUT_CH3 (3)`
`#define MTU3_PWM_OUTPUT_CH4 (4)`
`- Default value = 0`
	Select the MTU3 channel started by synchronous event via ELC.
	- Set 0 (default value) in the RX64M/71M.

`#define PTP_CFG_GPT_OUTPUT`
`#define GPT_PWM_OUTPUT_CH0 (0)`
`#define GPT_PWM_OUTPUT_CH1 (1)`
`#define GPT_PWM_OUTPUT_CH2 (2)`
`#define GPT_PWM_OUTPUT_CH3 (3)`
`- Default value = 1`
	Select the GPT channel started by synchronous event via ELC.
	- Set 1 (default value) in the RX64M/71M.

`#define CURRENT_UTC_OFFSET`
`- Default value = 0x0008`
`#define PTP2NTP_OFFSET`
`- Default value = 2208988800`
`#define NTP_SEC`
`- Default value = 3668572800`
`#define PTP_SEC`
`- Default value = (NTP_SEC - PTP2NTP_OFFSET + CURRENT_DS_UTC_OFFSET)`
`#define PTP_CFG_LCCLK_SEC_HI`
`- Default value = 0x0000`
`#define PTP_CFG_LCCLK_SEC_LO`
`- Default value = PTP_SEC`
`#define PTP_CFG_LCCLK_NANO`
`- Default value = 0x12345678`
	Set local clock counter initial value.
	- Set currentUtcOffset.
	Default value is 8 sec.
	- Set PTP time to NTP (Network Time Protocol) time conversion offset constant.
	Default value is 2,208,988,800 sec.
	- Set NTP sec.
	Default value is 3,668,572,800 sec (2016/03/31 08:00:00).
	- Set PTP sec.
	Default value is 1,459,584,008 sec.
	- Set second order high (16 bits) to PTP_CFG_LCCLK_SEC_HI.
	- Set second order low (32 bits) to PTP_CFG_LCCLK_SEC_LO.
	- Set nanosecond order to PTP_CFG_LCCLK_NANO.

`#define PTP_CFG_TIMESTAMP_LATENCY` `- Default value = 0x03D4024E`	Set timestamp latency value of ingress and egress ports. Default value is for MII, 100Mbps and STCA clock = 20MHz. - Ingress timestamp latency is 980(=0x3D4) nsec. - Egress timestamp latency is 590(=0x24E) nsec.
`#define PTP_CFG_LLC_CTL` `- Default value = 3`	Set LLC-CTL of IEEE802.3 format field value.
`#define PTP_CFG_PTP_VER_NUM` `- Default value = 0x02`	Set PTP version field value. Default value is IEEE1588-2008 (version2) spec. - Set 0x02 in the RX64M/71M.
`#define PTP_CFG_DOMAIN_NUM` `- Default value = 0`	Set domainNumber field value. 0 means "default domain", 1 to 3 mean "alternate domain", 4 to 127 mean "user defined" and 128 to 255 are reserved.

Configuration options	
#define PTP_CFG_ANNOUNCE_FLAG_FIELD **- Default value = 0x00000000**	Set Announce message's flag field value. In the following default setting, all flags are false. PTP profile Specific 2(b14) = false, PTP profile Specific 1(b13) = false, unicastFlag(b10) = false (not support unicast PTP in the RX64M/71M), alternateMasterFlag(b8) = false, frequencyTraceable(b5) = false, timeTraceable(b4) = false, ptpTimescale(b3) = false, currentUtcOffsetValid(b2) = false, leap59(b1) = false, leap61(b0) = false.
#define PTP_CFG_SYNC_FLAG_FIELD **- Default value = 0x00000000**	Set Sync message's flag field value. In the following default setting, all flags are false. PTP profile Specific 2(b14) = false, PTP profile Specific 1(b13) = false, unicastFlag(b10) = false (not support unicast PTP in the RX64M/71M), twoStepFlag(b9) = false (not support two step in the RX64M/71M), alternateMasterFlag(b8) = false.
#define PTP_CFG_DELAY_REQ_FLAG_FIELD **- Default value = 0x00000000**	Set Delay_Req/Pdelay_Req message's flag field value. In the following default setting, all flags are false. PTP profile Specific 2(b14) = false, PTP profile Specific 1(b13) = false, unicastFlag(b10) = false (not support unicast PTP in the RX64M/71M).
#define PTP_CFG_DELAY_RESP_FLAG_FIELD **- Default value = 0x00000000**	Set Delay_Resp/Pdelay_Resp message's flag field value. In the following default setting, all flags are false. PTP profile Specific 2(b14) = false, PTP profile Specific 1(b13) = false, unicastFlag(b10) = false (not support unicast PTP in the RX64M/71M), twoStepFlag(b9) = false (not support two step in the RX64M/71M).
#define PTP_CFG_CLK_ID **#define CLK_ID_EUI48_BASE (0)** **#define CLK_ID_USR_DEFINE (1)** **- Default value = 0**	Set PortIdentity clockIdentity value. OUI field (upper 3byte) and Extension field (lower 5byte). - When this is set to 0, create clockIdentity based on MAC (EUI-48) address. - When this is set to 1, set user specific value[2].
#define PTP_CFG_PORTID_PORT_NUM0 **- Default value = 1 (Port0)** **#define PTP_CFG_PORTID_PORT_NUM1** **- Default value = 2 (Port1)**	Set PortIdentity port number values (1, 2,,, N,). All 0 (0x0000) and all 1(0xFFFF) are reserved.
#define PTP_CFG_MTCID_U **- Default value = 0x00000000 (high: 32 bit)** **#define PTP_CFG_MTCID_L** **- Default value = 0x00000000 (low: 32 bit)**	Set master clockIdentity value. In general, equal to parentDS.parentPortIdentity.clockIdentity field.
#define PTP_CFG_MTPID **- Default value = 0x0000**	Set master port number value. In general, equal to parentDS.parentPortIdentity.portNumber field.

Configuration options	

`#define PTP_CFG_GM_PRIORITY10`
`- Default value = 0x00 (Port0)`
`#define PTP_CFG_GM_PRIORITY11`
`- Default value = 0x00 (Port1)`
	Set grandmasterPriority1 values.
	Equal to parentDS.grandmasterPriority1 field.
	From 0 to 255 can be set and lower value has higher priority.

`#define PTP_CFG_GM_PRIORITY20`
`- Default value = 0x00 (Port0)`
`#define PTP_CFG_GM_PRIORITY21`
`- Default value = 0x00 (Port1)`
	Set grandmasterPriority2 values.
	Equal to parentDS.grandmasterPriority2 field.
	From 0 to 255 can be set and lower value has higher priority.

`#define PTP_CFG_GM_CLK_QUALITY0`
`- Default value = 0xF821FFFF (Port0)`
`#define PTP_CFG_GM_CLK_QUALITY1`
`- Default value = 0xF821FFFF (Port1)`
	Set grandmasterClockQuality values.
	Equal to parentDS.grandmasterClockQuality field.
	- b31 to b24: clockClass
	Default value is 248 (= 0xF8) and slave only clock is 255.
	- b23 to b16: clockAccuracy
	Default value(=0x21)is within 100 nsec, from 0x20 to 0x31 can be set..
	- b15 to b0: offsetScaledLogVariance
	Default value(=0xFFFF) is not calculated yet.

`#define PTP_CFG_GM_CLK_ID0_U`
`- Default value = 0x00000000 (Port0, high: 32 bit)`
`#define PTP_CFG_GM_CLK_ID0_L`
`- Default value = 0x00000000 (Port0, low: 32 bit)`
`#define PTP_CFG_GM_CLK_ID1_U`
`- Default value = 0x00000000 (Port1, high: 32 bit)`
`#define PTP_CFG_GM_CLK_ID1_L`
`- Default value = 0x00000000 (Port1, low: 32 bit)`
	Set grandmasterIdentity values.
	Equal to parentDS.grandmasterIdentity field.

`#define PTP_CFG_CUR_UTC_OFFSET0`
`- Default value = 0x0008 (Port0)`
`#define PTP_CFG_CUR_UTC_OFFSET`
`- Default value = 0x0008 (Port1)`
	Set currentUtcOffset values.
	Equal to timePropertiesDS.currentUtcOffset field.
	Default value is “CURRENT_UTC_OFFSET” defined in the local clock counter initial value.

`#define PTP_CFG_TIME_SOURCE0`
`- Default value = 0xA0 (Port0)`
`#define PTP_CFG_TIME_SOURCE1`
`- Default value = 0xA0 (Port1)`
	Set timeSource values.
	Equal to timePropertiesDS.timeSource field.
	Default value is timesource is the internal oscillator.

`#define PTP_CFG_STEPS_REMOVED0`
`- Default value = 0x0000 (Port0)`
`#define PTP_CFG_STEPS_REMOVED1`
`- Default value = 0x0000 (Port1)`
	Set stepsRemoved values.
	Equal to currentDS.stepsRemoved field.
	Default value is no traversed.

`#define PTP_CFG_PTP_EVENT_TOS0`
`- Default value = 0x00 (Port0)`
`#define PTP_CFG_PTP_EVENT_TOS1`
`- Default value = 0x00 (Port1)`
	Set PTP event message's TOS field values.
	Default value is best effort.

`#define PTP_CFG_PTP_GENERAL_TOS0`
`- Default value = 0x00 (Port0)`
`#define PTP_CFG_PTP_GENERAL_TOS1`
`- Default value = 0x00 (Port1)`
	Set PTP general message's TOS field values.
	Default value is best effort.

`#define PTP_CFG_PTP_PRIMARY_TTL0`
`- Default value = 0x80 (Port0)`
`#define PTP_CFG_PTP_PRIMARY_TTL1`
`- Default value = 0x80 (Port1)`
	Set PTP-primary message's TTL field values.
	Default value is 128.

`#define PTP_CFG_PTP_PDELAY_TTL0`
`- Default value = 0x01 (Port0)`
`#define PTP_CFG_PTP_PDELAY_TTL1`
`- Default value = 0x01 (Port1)`
	Set PTP-pdelay message's TTL field values.
	Default value is 1.

Configuration options	
#define PTP_CFG_LOG_ANNOUNCE_INTERVAL0 - Default value = 0x01 (Port0: 2sec interval) **#define PTP_CFG_LOG_ANNOUNCE_INTERVAL1 - Default value = 0x01 (Port1: 2sec interval)**	Set announce message transmission interval value for each channel. Equals to portDS.logAnnounceInterval value. Transmission interval is power of 2 second set value. Assume set value is n, the transmission interval becomes 2^n sec. Only from 0xF9 (= -7) to 0x06 (=6) can be set and then they should be integer.
#define PTP_CFG_LOG_SYNC_INTERVAL0 **- Default value = 0x00 (Port0: 1sec interval)** **#define PTP_CFG_LOG_SYNC_INTERVAL1** **- Default value = 0x00 (Port1: 1sec interval)**	Set the sync message transmission interval value for each channel. Equals to portDS.logSyncInterval value. Transmission interval is power of 2 second set value. Assume set value is n, the transmission interval becomes 2^n sec. Only from 0xF9 (= -7) to 0x06 (=6) can be set and then they should be integer.
#define PTP_CFG_LOG_MIN_DELAY_REQ_INTERVAL0 - Default value = 0x00 (Port0: 1sec interval) **#define PTP_CFG_LOG_MIN_DELAY_REQ_INTERVAL1 - Default value = 0x00 (Port1: 1sec interval)**	Set the Delay_Req message minimum of mean transmission interval value for each channel. Equals to portDS.logMinDelayReqInterval value. Transmission interval is power of 2 second set value. Assume set value is n, the transmission interval becomes 2^n sec. Only from 0xF9 (= -7) to 0x06 (=6) can be set and then they are integer.
#define PTP_CFG_LOG_MIN_PDELAY_REQ_INTERVAL0 - Default value = 0x00 (Port0: 1sec interval) **#define PTP_CFG_LOG_MIN_PDELAY_REQ_INTERVAL1 - Default value = 0x00 (Port1: 1sec interval)**	Set the Pdelay_Req message minimum of mean transmission interval value for each channel. Equal to portDS.logMinPdelayReqInterval value. Transmission interval is power of 2 second set value. Assume set value is n, the transmission interval becomes 2^n sec. Only from 0xF9 (= -7) to 0x06 (=6) can be set and then they are integer.

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

[2] please change following definitions;
- PORTID_CLK_ID0_U for clockIdentity high (Port0)
- PORTID_CLK_ID0_L for clockIdentity low (Port0)
- PORTID_CLK_ID1_U for clockIdentity high (Port1)
- PORTID_CLK_ID1_L for clockIdentity low (Port1)

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_rx_if.h*, *r_ptpif_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 */
```

```
/* PTPEDMAC interrupt event */
typedef enum
{
    PTPIF_FUNC_READ = 0, /* Frame reception interrupt (FR) */
    PTPIF_FUNC_WRITE,    /* Frame transmission interrupt (TC) */
    PTPIF_FUNC_ERR,      /* Error interrupt (MACE, RFOF, RDE, TFUF, TDE, ADE and
RFCOF) */
} PTPIF_INTEVT;
```

```
/* 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;
```

```
/* Clock type and port number */
typedef enum
{
    PD_ORDINARY_CLOCK_PORT0 = 0, /* Ordinary Clock port0 */
    PD_ORDINARY_CLOCK_PORT1,     /* Ordinary Clock port1 */
    PD_BOUNDARY_CLOCK,           /* Boundary Clock */
    PD_TRANSPARENT_CLOCK,        /* Transparent Clock */
} PTPDevice;
```

```
/* Delay correction protocol */
typedef enum
{
    NP_P2P = 0, /* Peer to peer */
    NP_E2E,     /* End to end */
} DelayMechanism;
```

```
/* Master, Slave or Listening */
typedef enum
{ /* Those states are different from PTP state enumeration value */
    ST_MASTER = 0, /* Master state */
    ST_SLAVE,      /* Slave state */
    ST_LIST,       /* Listening state */
} PTPState;
```

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

```
/* Timer channel (bit map form) */
typedef enum
{
    INT_CYC0 = 0x01,
    INT_CYC1 = 0x02,
    INT_CYC2 = 0x04,
    INT_CYC3 = 0x08,
    INT_CYC4 = 0x10,
    INT_CYC5 = 0x20
} IntCycCh;
```

```
/* STCA mode and gradient setting */
typedef enum
{
    STCA_MODE1    = 0x00, /* Mode1 (not use STCA) */
    STCA_MODE2_HW = 0x02, /* Mode2 (use STCA) and HW gradient setting */
    STCA_MODE2_SW = 0x03, /* Mode2 (use STCA) and SW gradient setting */
} STCA_GRAD;
```

2.10.2 Data Type

(1) PTP data type structure

```
/* 48 bit unsigned integer */
typedef struct
{
    uint16_t hi;
    uint32_t lo;
} UInt48;
```

```
/* 64 bit signed integer */
typedef struct
{
    int32_t hi;
    uint32_t lo;
} Int64;
```

```
/* 64 bit unsigned integer */
typedef struct
{
    uint32_t hi;
    uint32_t lo;
} UInt64;
```

```
/* 64 bit scaled nano second unit expression */
typedef struct
{
    Int64 scaledNanoseconds;
} TimeInterval;
```

```
/* PTP message timestamp expression */
typedef struct
{
    UInt48 secondsField;
    uint32_t nanosecondsField;
} Timestamp;
```

(2) Driver control structure

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

```
/* PTPEDMAC interrupt handler type definition */
typedef void (*PTPIF_HNDLR)(uint8_t, uint32_t);
```

```
/* PTP port related structure (MAC address, IP address, PTP state and delay
mechanism) */
typedef struct
{
    uint8_t  macAddr[6];
    uint8_t  ipAddr[4];
    PTPState state;
    DelayMechanism delay;
} PTPPort;
```

```
/* PTP configuration structure (device and port information) */
typedef struct
{
    PTPDevice device;
    PTPPort port[NUM_PORT];
} PTPConfig;
```

```
/* Synchronous state change detection structure */
typedef struct
{
    UInt64 th_val; /* Threshold value of synchronous state or deviation */
    uint8_t times; /* Times of successive detection */
} SyncDet;
```

```
/* PTP sub configuration structure */
typedef struct
{
    UInt48 delay_asymmetry;
    uint8_t w10_times; /* Times of sync reception getting worst10 completed */
    SyncDet dev;       /* Condition of deviation state detection */
    SyncDet syn;       /* Condition of synchronous state detection */
} PTPSubConfig;
```

```
/* MINT interrupt handler user's function type definition */
typedef void (*MINT_HNDLR)(uint32_t);
```

```
/* Timer interrupt handler user's function type definition */
typedef void (*TMR_HNDLR)(uint32_t);
```

```
/* Announce flagField type structure */
#pragma bit_order left
typedef union
{
    uint32_t LONG;
    R_BSP_ATTRIB_STRUCT_BIT_ORDER_LEFT_14 (
      uint32_t Rsv1:17;
      uint32_t profileSpec2:1;
      uint32_t profileSpec1:1;
      uint32_t Rsv2:2;
```

```
      uint32_t unicastFlag:1;
      uint32_t Rsv3:1;
      uint32_t alternateMasterFlag:1;
      uint32_t Rsv4:2;
      uint32_t frequencyTraceable:1;
      uint32_t timeTraceable:1;
      uint32_t ptpTimescale:1;
      uint32_t currentUtcOffsetValid:1;
      uint32_t leap59:1;
      uint32_t leap61:1;
   ) BIT;
} AnceFlag;
```

```
/* PTP clock quality structure */
typedef struct
{
   uint8_t clockClass;
   uint8_t clockAccuracy;
   uint16_t offsetScaledLogVariance;
} ClkQuality;
```

```
/* Announce message field type structure */
typedef struct
{
   uint8_t grandmasterPriority1;
   uint8_t grandmasterPriority2;
   ClkQuality grandmasterClockQuality;
   int8_t *grandmasterIdentity;
} AnceMsg;
```

2.11 Return Values

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

```
/* PTP driver (PTP Host interface part) return value */
typedef enum
{
   PTPIF_ERR_PARAM = -6,     /* Parameter error */
   PTPIF_ERR_LEN = -5,       /* Data length error */
   PTPIF_ERR_TACT = -4,      /* No remaining transmit descriptor */
   PTPIF_ERR_NO_DATA = -3,   /* No data received */
   PTPIF_ERR_NOT_TRAN = -2,  /* Not transfer enabled */
   PTPIF_ERR = -1,           /* General error */
   PTPIF_OK = 0
} ptpif_return_t;
```

```
/* PTP driver (PTP synchronization part) return value */
typedef enum
{
   PTP_ERR_TOUT = -3,   /* Timeout error */
   PTP_ERR_PARAM = -2,  /* Parameter error */
   PTP_ERR = -1,        /* General error */
   PTP_OK = 0
} ptp_return_t;
```

2.12 Callback Function

In this module, the callback function Eptpc_isr is called when the EPTPC MINT interrupt occurs, and the callback function Ptpedmac_isr is called when the PTPEDMAC PINT 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_rx rev1.50

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 4.08.04.201801

(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_ptpif.c	942 bytes	1750 bytes	1323 bytes
		r_ptp.c	5528 bytes	12060 bytes	9840 bytes
		Total	6470 bytes	13810 bytes	11163 bytes
	RAM	r_ptpif.c	3128 bytes	3128 bytes	3125 bytes
		r_ptp.c	71 bytes	63 bytes	56 bytes
		Total	3199 bytes	3191 bytes	3181 bytes
	STACK	r_ptpif.c	64 bytes	-	48 bytes
		r_ptp.c	80 bytes	-	52 bytes
		Maximum	80 bytes	-	52 bytes
RX72M	ROM	r_ptpif.c	962 bytes	1775 bytes	1399 bytes
		r_ptp.c	5481 bytes	12797 bytes	9998 bytes
		Total	6443 bytes	14572 bytes	11397 bytes
	RAM	r_ptpif.c	3128 bytes	3128 bytes	3125 bytes
		r_ptp.c	71 bytes	63 bytes	56 bytes
		Total	3199 bytes	3191 bytes	3181 bytes
	STACK	r_ptpif.c	64 bytes	-	72 bytes
		r_ptp.c	80 bytes	-	72 bytes
		Maximum	80 bytes	-	72 bytes

[1] The sizes of maximum usage stack of interrupts functions is included.

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

3. API Functions

3.1 R_PTPIF_GetVersion ()

This function returns the version number of PTP Host interface driver.

Format

uint32_t R_PTPIF_GetVersion(void);

Parameters

None

Return Values

RX_PTPIF_VERSION_MAJOR (upper 16bit): Major version number
RX_PTPIF_VERSION_MINO (lower 16bit): Minor version number

Properties

Prototyped in “r_ptpif_rx_if.h”.

Description

Return major and minor version number of PTP Host interface driver.

This driver version number is “1.16”.

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

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

Reentrant

Function is reentrant.

Example

Example showing this function being used.

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

   uint32_t ptpif_version;

   ptpif_version = R_PTPIF_GetVersion();

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

Special Notes

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

3.2 R_PTPIF_Reset ()

This function resets PTPEDMAC.

Format

void R_PTPIF_Reset(void);

Parameters

None

Return Values

None

Properties

Prototyped in “r_ptpif_rx_if.h”.

Description

This function resets the PTPEDMAC[1]. Following operations are executed.

[1] Does not reset ETHERC

- Reset PTPEDMAC
  Setting “1” SWR bit of the EDMR register.

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

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptpif_rx_if.h"
#include "r_ptp_rx_if.h"
#include "r_ether_if.h"

   ether_return_t ether_ret; /* Ether driver return */
   ptp_return_t ptp_ret;     /* PTP synchronous driver return */
   ptpif_return_t ptpif_ret; /* PTP Host interface driver return */
   int32_t ch;

   PTPConfig ptpc;                          /* PTP device information */
   ether_param_t ectrl;               /* EDMAC and ETHERC control */
   ether_promiscuous_t pcuous;  /* promiscuous control */

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

   /* ==== Open standard Ether ==== */
   for (ch = 0;ch < NUM_CH;ch++)
   { /* 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*)&mac_addr[ch],
ETHER_FLAG_OFF);
      if (ETHER_SUCCESS != ether_ret)
      {
          goto Err_end;
```

```
    }
}

/* ==== Open PTP and PTP Host interface ==== */
/* Reset PTPEDMAC */
R_PTPIF_Reset();

/* Reset EPTPC */
R_PTP_Reset();

/* Initialize resources of the PTP driver */
R_PTPIF_Init();

/* Initialize EPTPC */
ptp_ret = R_PTP_Init(&ptpc);
if (PTP_OK != ptp_ret)
{
    goto Err_end;
}

/* Initialize PTP Host interface and peripheral modules */
ptpif_ret = R_PTPIF_Open_ZC2();
if (PTPIF_OK != ptpif_ret)
{
    goto Err_end;
}

/* Set PTP Host interface to transfer PTP message */
R_PTPIF_LinkProcess();

/* Check PTP Host interface status */
ptpif_ret = R_PTPIF_CheckLink_ZC();
if (PTPIF_OK != ptpif_ret)
{
    goto Err_end;
}

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

    /* Clear extended promiscuous mode */
    R_PTP_SetExtPromiscuous((uint8_t)ch, false);

    /* Clear promiscuous mode */
    pcuous.channel = ch;
    pcuous.bit = ETHER_PROMISCUOUS_ON;
    ectrl.p_ether_promiscuous = &pcuous;
    R_ETHER_Control(CONTROL_SET_PROMISCUOUS_MODE, ectrl);

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

```
/* Start synchronization */
while(1)
{
   ptp_ret = R_PTP_Start();
    if (PTP_OK == ptp_ret)
    {
       break;
    }
    else if (PTP_ERR_TOUT == ptp_ret)
    {
       ; /* continue */
    }
    else
    { /* any error occured */
       goto Err_end;
    }
}

/* ==== Complete synchronization procedure ==== */
```

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 (before access CH1) such as this example, you need to change the configuration defined in the “r_ether_rx_config.h” of “RX Family Ethernet Module Using Firmware Integration Technology [2]”(= r_ether_rx) from default setting.

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

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

3.3 R_PTPIF_Init ()

This function initializes resources of the PTP Host interface driver.

Format

void R_PTPIF_Init(void);

Parameters

None

Return Values

None

Properties

Prototyped in “r_ptpif_rx_if.h”.

Description

This function initializes resources of the PTP Host interface driver. Following operations are executed.

- Initialize the transmit and receive descriptors

- Initialize the PTPEDMAC buffer

Reentrant

Function is not reentrant.

Example

This example is same as “3.2 R_PTPIF_Reset”.

Special Notes

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

3.4 R_PTPIF_Open_ZC2 ()

This function initializes PTP Host interface and peripheral modules.

Format

ptpif_return_t R_PTPIF_Open_ZC2(void);

Parameters

None

Return Values

PTPIF_OK: Processing completed successfully
PTPIF_ERR: Not implemented in this version

Properties

Prototyped in “r_ptpif_rx_if.h” and "r_ptpif_rx_private.h".

Description

This function initializes PTP Host interface and peripheral modules. Following operations are executed.

- Initialize PTP Host interface transfer to disable.
- Clear all PTPEDMAC status flags.
- Set interrupt from PTPEDMAC.
  Call ptpif\dev_start().

Reentrant

Function is not reentrant.

Example

This example is same as “3.2 R_PTPIF_Reset”.

Special Notes

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

3.5 R_PTPIF_LinkProcess ()

This function set PTP Host interface to transfer PTP message.

Format

void R_PTPIF_LinkProcess(void);

Parameters

None

Return Values

None

Properties

Prototyped in “r_ptpif_rx_if.h”.

Description

This function set PTP Host interface to transfer PTP message. Following operations are executed.

- Initialize the receive and transmit descriptors
  Call _R_PTPIF_InitDescriptors().

- Initialize PTPEDMAC
  Call _R_PTPIF_ConfigEthernet().

- Set PTPEDMAC receive operation to enable

- Set PTP Host interface transfer flag to enable

Reentrant

Function is not reentrant.

Example

This example is same as “3.2 R_PTPIF_Reset”.

Special Notes

This function needs to execute every time Ethernet MAC status changes.

3.6 R_PTPIF_CheckLink_ZC ()

This function checks PTP Host interface communication status.

Format

ptpif_return_t R_PTPIF_CheckLink_ZC(void);

Parameters

None

Return Values

PTPIF_OK: Transfer enable
PTPIF_ERR: Transfer disable

Properties

Prototyped in “r_ptpif_rx_if.h”.

Description

This function inquires PTP Host interface whether the transfer enable or not refer to PTP Host interface transfer flag.

Reentrant

Function is reentrant.

Example

This example is same as “3.2 R_PTPIF_Reset”.

Special Notes

None

3.7 R_PTPIF_Close_ZC2 ()

This function disables a PTP Host interface module.

Format

ptpif_return_t R_PTPIF_Close_ZC2(void);

Parameters

None

Return Values

PTPIF_OK: Processing completed successfully
PTPIF_ERR: Already closed

Properties

Prototyped in “r_ptpif_rx_if.h”.

Description

This function disables a PTP Host interface module. Following operations are executed.

- Clear PTPEDMAC interrupt
  Call ptpif_dev_start().

- Set PTP Host interface transfer flag to disable

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptpif_rx_if.h"
#include "r_ptp_rx_if.h"
#include "r_ether_if.h"

   int32_t ch;
   ether_param_t ectrl; /* EDMAC and ETHERC control */

   /* Synchronization based on the PTP */

   /* Stop synchronization */
   R_PTP_Stop();

   for (ch = 0;ch < NUM_CH;ch++)
   { /* Disable EDMAC Host interface */
      R_ETHER_Close_ZC2(ch);

      /* Power off ether channel */
      ectrl.channel = ether_ch[ch];
      R_ETHER_Control(CONTROL_POWER_OFF, ectrl);
   }

   /* Disable PTP Host interface */
   R_PTPIF_Close_ZC2();

   return;
```

Special Notes

This function is finalized operation and only relevant when PTP Host interface transfer is enabled.

3.8 R_PTPIF_Read ()

This function receives a PTP message.

Format

int32_t R_PTPIF_Read(uint32_t *ch, uint8_t* buf);

Parameters

ch - received port channel (0 or 1).
buf - read buffer pointer. Need to allocate more than buffer size of PTPEDMAC (=PTPIF_CFG_BUFSIZE).

Return Values

More than 0: Number of received data
PTPIF_ERR: Any error occurred

Properties

Prototyped in “r_ptpif_rx_if.h”.

Description

This function receives a PTP message. Following operations are executed.

- Receive one PTP message frame or fragment and set the allocated buffer pointer for received data
  Call R_PTPIF_Read_ZC2()

- Release the receive buffer
  Call R_PTPIF_Read_ZC2_BufRelease()

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include <string.h>
#include "r_ptpif_rx_if.h"
#include "r_ptp_rx_if.h"
#include "r_ether_if.h"

    int32_t r_size; /* read data size */
    uint32_t ch; /* read channel */
    uint8_t R_BUF[PTPIF_CFG_BUFSIZE]; /* read data buffer */

    /* Standard Ethernet open and link were completed */

    /* PTP open and link were completed */
    /* Ref 3.2 R_PTPIF_Reset() example for more detail information. */

    /* Receive PTP message */
    while (1)
    {
        r_size = R_PTPIF_Read(&ch, R_BUF);
        if (r_size > 0)
        {
            break; /* read complete */
        }
    }

   return;
```

Special Notes

Standard library <string.h> is used in this function.

User's data buffer is had to allocate more than buffer size of PTPEDMAC (=PTPIF_CFG_BUFSIZE). Otherwise, user's data buffer is overwritten by the PTP driver.

This function is prohibited to simultaneous use with "3.10 R_PTPIF_Read_ZC2" and "3.11 R_PTPIF_Read_ZC2_BufRelease".

Standard Ethernet and PTP operations should be opened and linked before executing this function.

3.9 R_PTPIF_Write ()

This function transmits a PTP message or a standard Ethernet frame.

Format

ptpif_return_t R_PTPIF_Write(uint8_t* buf, uint32_t size);

Parameters

buf - write buffer pointer.
size – write data size.

Return Values

PTPIF_OK: Processing completed successfully
PTPIF_ERR_LEN: Data length error
PTPIF_ERR: Any error occurred but for "PTPIF_ERR_LEN"

Properties

Prototyped in “r_ptpif_rx_if.h”.

Description

This function transmits a PTP message or a standard Ethernet frame. Following operations are executed.

- Set the buffer pointer and allocate the size for transmit data
  Call R_PTPIF_Write_ZC2_GetBuf()

- Transmit one PTP message frame or fragment and set the descriptor for the transmit data
  Call R_PTPIF_Write_ZC2_SetBuf()

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include <string.h>
#include "r_ptpif_rx_if.h"
#include "r_ptp_rx_if.h"
#include "r_ether_if.h"

    ptpif_return_t ret;
    int32_t w_size; /* write data size */
    uint8_t W_BUF[128]; /* write data buffer */

    /* Standard Ethernet open and link were completed */

    /* PTP open and link were completed */
    /* Ref 3.2 R_PTPIF_Reset() example for more detail information. */

   w_size = 128; /* set write data size */

    /* Transmit PTP message or Ethernet frame */
    ret = R_PTPIF_Write(W_BUF, w_size);
    if (PTPIF_OK != ret)
    {
        goto Err_end; /* error */
    }

    return;
```

Special Notes

This function is used not only PTP message but also standard Ethernet frame transmission.

Standard library <string.h> is used in this function.

Standard Ethernet and PTP should be opened and linked before executing this function.

This function is prohibited to simultaneous use with “3.12 R_PTPIF_Write_ZC2GetBuf” and “3.13 R_PTPIF_Write_ZC2_SetBuf”.

3.10 R_PTPIF_Read_ZC2 ()

This function receives one PTP message or fragment.

Format

int32_t R_PTPIF_Read_ZC2(uint32_t *ch, void* buf);

Parameters

ch - received port channel (0 or 1).
buf - received data store buffer.

Return Values

More than 0: Number of received data
PTPIF_ERR_NO_DATA: No data received
PTPIF_ERR_NOT_TRAN: Not transfer enabled

Properties

Prototyped in “r_ptpif_rx_if.h”.

Description

This function receives one PTP message or fragment and set the allocated buffer pointer for received data. Following operations are executed.

- Check transfer status

- Check received data existed or not

- Get received port channel

- Set the allocated buffer pointer for received data

- Get received data length

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

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

   int32_t ret;
   ptpif_return_t ptpif_ret;
   uint8_t* r_buf;
   uint8_t R_BUF[PTPIF_CFG_BUFSIZE]; /* read data buffer */

   if (NULL == buf)
   {
      goto Err_end; /* error */
   }

   /* Set the allocated buffer pointer for received data */
   ret = R_PTPIF_Read_ZC2(ch, (void **)&r_buf);
   if (0 < ret)
   {
      memcpy(R_BUF, r_buf, ret);

      /* Release the receive buffer */
      ptpif_ret = R_PTPIF_Read_ZC2_BufRelease();
      if (PTPIF_OK != ptpif_ret)
      {
```

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

Special Notes

Standard Ethernet and PTP operations should be opened and linked before executing this function.

This function usually used combination with “3.11 R_PTPIF_Read_ZC2_BufRelease” function in the PTP message receiving operation.

This function is prohibited to simultaneous use with “3.8 R_PTPIF_Read”.

3.11 R_PTPIF_Read_ZC2_BufRelease ()

This function releases a receive buffer.

Format

ptpif_return_t R_PTPIF_Read_ZC2_BufRelease(void);

Parameters

None

Return Values

PTPIF_OK: Processing completed successfully
PTPIF_ERR_NOT_TRAN: Not transfer enabled

Properties

Prototyped in “r_ptpif_rx_if.h”.

Description

This function releases a receive buffer. Following operations are executed.

- Check transfer status

- Check received data existed or not

- Move current descriptor to next one

- If receive operation of the PTPEDMAC was disabled (EDRRR.RR = 0), make PTPEDMAC to receive enable

Reentrant

Function is not reentrant.

Example

This example is same as “3.10 R_PTPIF_Read_ZC2”.

Special Notes

Standard Ethernet and PTP operations should be opened and linked before executing this function.

This function usually used combination with “3.10 R_PTPIF_Read_ZC2” function in the PTP message receiving operation.

This function is prohibited to simultaneous use with “3.8 R_PTPIF_Read”.

3.12 R_PTPIF_Write_ZC2_GetBuf ()

This function set a buffer pointer and allocate the buffer by transmit data size.

Format

ptpif_return_t R_PTPIF_Write_ZC2_GetBuf(void **buf, uint16_t *size);

Parameters

buf - write data store buffer.
size – allocated buffer size.

Return Values

PTPIF_OK: Processing completed successfully
PTPIF_ERR_NOT_TRAN: Not transfer enabled
PTPIF_ ERR_TACT: No remaining transmit descriptor

Properties

Prototyped in “r_ptpif_rx_if.h”.

Description

This function set a buffer pointer and allocate the buffer by transmit data size. Following operations are executed.

- Check transfer status

- Check remaining transmit buffer existed or not

- Set the buffer pointer and allocate the buffer by transmit data size

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include <string.h>
#include "r_ptpif_rx_if.h"
#include "r_ptp_rx_if.h"

   ptpif_return_t ret;
   uint8_t* w_buf;
   uint16_t w_size;

   if ((NULL == buf) || (NULL == size))
   {
      goto Err_end; /* error */
   }

   ret = R_PTPIF_Write_ZC2_GetBuf((void**)&w_buf, &w_size);
   if (PTPIF_OK == ret)
   {
      memcpy(w_buf, (uint8_t*)buf, size);
     R_PTPIF_Write_ZC2_SetBuf(size);
      return;
   }
   else
   {
      goto Err_end; /* error */
   }
```

Special Notes

This function is used not only PTP message but also standard Ethernet frame transmission.

Standard Ethernet and PTP operations should be opened and linked before executing this function.

This function usually used combination with “3.13 R_EPTPIF_Write_ZC2_SetBuf” function in the PTP message transmitting operation.

This function is prohibited to simultaneous use with “3.9 R_PTPIF_Write”.

3.13 R_PTPIF_Write_ZC2_SetBuf ()

This function transmits one PTP message or one Ethernet frame or fragment.

Format

ptpif_return_t R_PTPIF_Write_ZC2_SetBuf(uint32_t len);

Parameters

len - transmit data length.

Return Values

PTPIF_OK: Processing completed successfully
PTPIF_ERR_NOT_TRAN: Not transfer enabled
PTPIF_ERR_LEN: Data length error

Properties

Prototyped in “r_ptpif_rx_if.h”.

Description

This function transmits one PTP message or one Ethernet frame or fragment and set the descriptor for the transmit data. Following operations are executed.

- Check transfer status

- Check transmit frame data length

- Set the descriptor for the transmit data

- If transmit operation was disabled (EDTRR.TR = 0), make PTPEDMAC to transmit enable

Reentrant

Function is not reentrant.

Example

This example is same as “3.12 R_PTPIF_Write_ZC2_GetBuf”.

Special Notes

This function is used not only PTP message but also standard Ethernet frame transmission.

Standard Ethernet and PTP operations should be opened and linked before executing this function.

This function usually used combination with “3.12 R_PTPIF_Write_ZC2_GetBuf” function in the PTP message transmitting operation.

This function is prohibited to simultaneous use with “3.9 R_PTPIF_Write”.

3.14 R_PTPIF_RegMsgHndr ()

This function registers a user function to the interrupt handler of PTPEDMAC.

Format

ptpif_return_t R_PTPIF_RegMsgHndr(PTPIF_INTEVT event, PTPIF_HNDLR func);

Parameters

event - PTPEDMAC interrupt event.
PTPIF_FUNC_READ: Frame reception interrupt (FR) (EMACP_FR_INT = 0x0004000)
PTPIF_FUNC_WRITE: Frame transmission interrupt (TC) (EMACP_TC_INT = 0x0020000)
PTPIF_FUNC_ERR: Error interrupt (MACE, RFOF, RDE, TFUF, TDE, ADE and RFCOF)
func - register function.

Return Values

PTPIF_OK: Processing completed successfully
PTPIF_ERR: Failed to register

Properties

Prototyped in “r_ptpif_rx_if.h”.

Description

This function registers a user function to the interrupt handler of PTPEDMAC. Following operations are executed.

- If the event is frame receive complete event (=bit18), register PTPIF read message handler.

- If the event is frame transmit complete event (=bit21), register PTPIF write message handler.

- If the event is any error events (=bit8, bit16, bit17, bit19, bit20, bit23 and bit24), register PTPIF error message handler.

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptpif_rx_if.h"
#include "r_ptp_rx_if.h"
#include "r_ether_if.h"

   /* user read PTP message function */
   void user_read_func(uint8_t ch, uint32_t type);

   /* user error function */
   void user_err_func(uint8_t ch, uint32_t sts);

   /* register user read PTP message function */
   R_PTPIF_RegMsgHndr(PTPIF_FUNC_READ, (PTPMSG_HNDLR)user_read_func);

   /* register user error function */
   R_PTPIF_RegMsgHndr(PTPIF_FUNC_ERR, (PTPMSG_HNDLR)user_err_func);

   /* wait interrupt from PTPEDMAC */

   /* interrupt occurred (call user_read_func or user_err_func) */

   /* relese registered user read PTP message function */
   R_PTPIF_RegMsgHndr(PTPIF_FUNC_READ, (PTPMSG_HNDLR)NULL);

   return;
```

Special Notes
This function defines read message, write message and error interrupt handlers.

Even if any function is already registered, the message handler is updated by the new register function.

PTPEDMAC interrupt handler type definitions are following.

- Read PTP message (PTPIF_FUNC_READ):
  user_read_func(uint8_t ch, uint32_t type);
  1st argument is read message channel (0 or 1). 2nd argument is a PTP message type.
  Typical operation is getting PTP message contents.

- Write PTP message (PTPIF_FUNC_WRITE):
  user_write_func(uint32_t res, uint32_t sts);
  1st argument is reserved and set 0. 2nd argument is status register value of the PTPEDMAC.
  Typical operation is checking the transmission error.

- Error event (PTPIF_FUNC_ERR):
  user_err_func(uint32_t res, uint32_t sts);
  1st argument is reserved and set 0. 2nd argument is status register value of the PTPEDMAC.
  Typical operation is checking the any status error.

3.15 R_PTP_GetVersion ()

This function returns the version number of PTP synchronization driver.

Format

uint32_t R_PTP_GetVersion(void);

Parameters

None

Return Values

RX_PTP_VERSION_MAJOR (upper 16bit): Major version number
RX_PTP_VERSION_MINO (lower 16bit): Minor version number

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

Return major and minor version number of PTP synchronization driver.

This driver version number is “1.16”.

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

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

Reentrant

Function is reentrant.

Example

Example showing this function being used.

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

   uint32_t ptp_version;

   ptp_version = R_PTP_GetVersion();

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

Special Notes

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

3.16 R_PTP_Reset ()

This function resets EPTPC.

Format

void R_PTP_Reset(void);

Parameters

None

Return Values

None

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function resets the EPTPC. 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.

- Wait reset release is completed
  More than 256 PCLKA cycles.
  To wait reset complete, loop operation of R_BSP_SoftwareDelay() is used.

Reentrant

Function is not reentrant.

Example

This example is same as “3.2 R_PTPIF_Reset”.

Special Notes

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

3.17 R_PTP_SetTran ()

This function set inter ports transfer mode.

Format

ptp_return_t R_PTP_SetTran(TranMode *mode, RelEnabDir *dir);

Parameters

mode – Inter ports transfer mode.

dir - Relay enable directions (bit map form).

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function set inter ports transfer mode. 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_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)

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

   /* Reset PTPEDMAC */
   R_PTPIF_Reset();

   /* Reset EPTPC */
   R_PTP_Reset();

   /* ==== Clear extended promiscuous mode: CH0 ==== */
   R_PTP_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_PTP_SetTran(&mode, &dir);
if (PTP_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_PTP_SetMCFilter(0, MC_REC_REG1, fmac);
if (PTP_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.

3.18 R_PTP_SetMCFilter ()

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

Format

ptp_return_t R_PTP_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

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function set MC filter (FFLTR). 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 “3.17 R_PTP_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 3.35 R_PTP_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 “3.17 R_PTP_SetTran”.

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

3.19 R_PTP_SetExtPromiscuous ()

This function set/clears extended promiscuous mode.

Format

ptp_return_t R_PTP_SetExtPromiscuous(uint8_t ch, bool is_set);

Parameters

ch - Sync unit channel (SYNFP0 or SYNFP1).

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

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_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_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 < NUM_CH;ch++)
   {
      /* ==== Clear extended promiscuous mode (CH0 and CH1) ==== */
      R_PTPL_SetExtPromiscuous((uint8_t)ch, false);

      /* ==== 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);
   }

   /* 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

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

3.20 R_PTP_Init ()

This function initializes EPTPC depends on the device configuration.

Format

ptp_return_t R_PTP_Init(PTPConfig *tbl);

Parameters

tbl – PTP configuration table.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error
PTP_ERR: Any error occurred

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

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

- Case of RX72M device, disable bypass function.

- Set device parameter

- Select synchronization frame processing unit (SYNFP0 or SYNFP1)

- Initialize the pointer to timer interrupt handler

- Initialize SYNFP0 and SYNFP1
  Call _R_PTP_Init_Param_SYNFP().

- Set PTP reception filters depend on the device type

- Validate set values to SYNFP0 and SYNFP1

- Initialize packet relation controller unit (PRC-TC) and statistical time correction algorithm unit (STCA)
  Call _R_PTP_Init_PRC_STCA().

- Clear EPTPC interrupt status

- Clear EPTPC interrupt mask

- Set EPTPC interrupt
  Call ptp_dev_start().

Reentrant

Function is not reentrant.

Example

This example is same as “3.2 R_PTPIF_Reset”.

Special Notes

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

To set parameter for each channel, “for” statements (loop processing) are used.

3.21 R_PTP_SubConfig ()

This function set the optional configuration of EPTPC.

Format

ptp_return_t R_PTP_SubConfig(uint8_t ch, PTPSubConfig *tbl);

Parameters

ch - Sync unit channel (SYNFP0 or SYNFP1).

tbl - PTP optional configuration table.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

Set the optional configuration of EPTPC.

Set interval of getting worst10 values. (Recommend 32 or more times sync message reception).

Reentrant

Function is reentrant.

Example

Example showing this function being used when getting worst10 values and set those values to the gradient limits.

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

    ptp_return_t ptp_ret;
    PTPConfig ptpc; /* PTP device information */
    PTPSubConfig ptpsubc; /* PTP optional configuration table */
#if (PTP_SYNC_MODE == PTP_SYNC_MODE2_SW)
    uint32_t p_lim[3]; /* plus gradient limit */
    uint32_t m_lim[3]; /* minus gradient limit */
#endif /* End of (PTP_SYNC_MODE == PTP_SYNC_MODE2_SW) */

    /* Set interval of getting worst10 values */
    ptpsubc.w10_times = 64; /* Set 64 sync reception times */
    ptp_ret = R_PTP_SubConfig(0, &ptpsubc);
    if (PTP_OK != ptp_ret)
    {
        goto Err_end;
    }

    /* ==== Start synchronization ==== */
    while(1)
    {
        ptp_ret = R_PTP_Start();
        if (PTP_OK == ptp_ret)
        {
            break;
        }
        else if (PTP_ERR_TOUT == ptp_ret)
        {
            ; /* continue */
        }
        else
```

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

#if (PTP_SYNC_MODE2_HW == PTP_CFG_SYNC_MODE)
    if ((ST_SLAVE == ptpc.port[0].state) || (ST_SLAVE == ptpc.port[1].state))
    { /* Slave port */
       if (ptpsubc.w10_times != 0)
       { /* Get current worst10 values */
           ptp_ret = R_PTP_ChkW10();
           if (ptp_ret != PTP_OK)
           { /* Timeout error */
              goto Err_end;
           }
       }
    }
#elif (PTP_SYNC_MODE2_SW == PTP_CFG_SYNC_MODE)
    if ((ST_SLAVE == ptpc.port[0].state) || (ST_SLAVE == ptpc.port[1].state)
    { /* Slave port */
        if (ptpsubc.w10_times != 0)
        { /* Get current worst10 values */
           ptp_ret = R_PTP_GetW10(p_lim, m_lim);
           if (ptp_ret != PTP_OK)
           { /* Timeout error */
               goto Err_end;
           }

           /* Get current worst10 values */
           ptp_ret = R_PTP_SetGradLimit(p_lim, m_lim);
           if (ptp_ret != PTP_OK)
           {
              goto Err_end;
           }
           printf("plus gradient limit high, mid, low = %8x, %8x, %8x\n",
p_lim[0], p_lim[1], p_lim[2]);
           printf("minus gradient limit high, mid, low = %8x, %8x, %8x\n",
m_lim[0], m_lim[1], m_lim[2]);
        }
    }
#else
    ; /* no operation */
#endif /* End of (PTP_CFG_SYNC_MODE) */
    /* Operation completed */
```

Special Notes

User's optional static configurations are possible to add this function.

This version only implements worst10 acquisition times setting.

3.22 R_PTP_RegMINTHndr ()

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

Format

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

Parameters

reg - MINT interrupt register.
MINT_FUNC_STCA: Interrupt from STCA
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_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 STCA, register the user function called from STCA interrupt event.

- 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_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_PTP_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_PTP_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.

3.23 R_PTP_RegTmrHndr ()

This function registers a user function to timer interrupt handler of EPTPC.

Format

ptp_return_t R_PTP_RegTmrHndr(IntCycCh ch, TMR_HNDLR func);

Parameters.

ch - timer interrupt channel.

func - user function registered.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function registers a user function to timer interrupt handler of EPTPC. Following operations are executed.

- Set or clear timer interrupt channel of the registered handler

- Register the user function to the interrupt handler

Reentrant

Function is not reentrant.

Example

Example showing this function being used. Following is MTU3 selected case as PWM output timer.

```
#include "r_ptpif_rx_if.h"
#include "r_ptp_rx_if.h"
#include "r_ether_if.h"

   int32_t ret;
   int32_t ch;
   ptp_return_t ptp_ret;
   ptpif_return_t ptpif_ret;
   SyncConfig scfg; /* Synchronous test information */
   PulseConfig pcfg; /* Output pulse information */
   PTPConfig ptpc;  /* PTP device information */
   IntCycCh cyc_ch; /* Output pulse channel */
   UInt64 start_time; /* Output pulse start time */

   /* Initialize MTU3 (CH0) */
   R_MTU3_Init(0);

   /* Initialize ELC */
   R_ELC_Init();

   /* Set PTP timer event to ELC */
   /* Timer MTU3, channel 3, falling edge */
   ret = R_ELC_Set_Timer_Event(0, 3, 1);
   if (ret != 0)
   {
      goto Err_end; /* error */
   }

   /* Enable PTP timer event */
```

```
R_ELC_Ctr_Timer_Event(1);

/* ==== Open PTP and PTP Host interface ==== */
/* Reset PTPEDMAC */
R_PTPIF_Reset();

/* Reset EPTPC */
R_PTP_Reset();

/* Initialize resources of the PTP driver */
R_PTPIF_Init();

/* Initialize EPTPC */
ptp_ret = R_PTP_Init(&ptpc);
if (PTP_OK != ptp_ret)
{
    goto Err_end; /* error */
}

/* Clear extended promiscuous mode (CH0 and CH1) */
for (ch = 0;ch < NUM_CH;ch++)
{
   R_PTP_SetExtPromiscuous((uint8_t)ch, false);
}

/* Register timer interrupt handler */
/* In this example, register disable timer event function */
/* Select CH3, registered function is R_PTP_DisableTmr() */
ptp_ret = R_PTP_RegTmrHndr(INT_CYC3, (TMR_HNDLR) R_PTP_DisableTmr);
if (PTP_OK != ptp_ret)
{
    goto Err_end; /* error */
}

/* Set ELC interrupt indication */
/* Select CH3, falling edge, set indication */
R_PTP_ELC_Ind(INT_CYC3, 1, 1);

/* Set ELC interrupt */
/* Select CH3, falling edge, set auto clear */
R_PTP_ELC_SetClr(INT_CYC3, 1, 1);

/* Set output pulse start time (timer event time) */
/* Initial value, LCIVRM 0x56FF7C08, LCIVRL 0x87654321 */
/* TMSTTR format converted, hi 0x14417BEC, lo 0x63ADCC00 */
/* CH3: 23 sec, 2222 nsec, hi 0x14417BF1, lo 0xEEFBC200 */
start_time.hi = 0x14417BF1;
start_time.lo = 0xEEFBC200;

/* Enable timer interrupt handler (timer event) */
/* cycle and pulse interval are infinite */
R_PTP_Tmr_Set(INT_CYC3, start_time, 0x3FFFFFFF, 0x1FFFFFFF);

/* Initialize PTP Host interface and peripheral modules */
ptpif_ret = R_PTPIF_Open_ZC2();
if (PTPIF_OK != ptpif_ret)
{
    goto Err_end; /* error */
}

/* Set PTP Host interface to transfer PTP message */
```

```
    R_PTPIF_LinkProcess();

    /* Check PTP Host interface status */
    ptpif_ret = R_PTPIF_CheckLink_ZC();
    if (PTPIF_OK != ptpif_ret)
    {
       goto Err_end; /* error */
    }

    /* Start synchronization */
    ptp_ret = R_PTP_Start();
    if (PTP_OK != ptp_ret)
    {
       goto Err_end; /* error */
    }
```

Special Notes

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

If 2nd argument(=func) is NULL, registered function is removed and that channel interrupt is disabled.

To register user function for each timer channel, “for” statement (loop processing) is used.

3.24 R_PTP_ELC_Ind ()

This function set/clears ELC interrupt indication.

Format

ptp_return_t R_PTP_ELC_Ind(IntCycCh ch, uint8_t edge, uint8_t set);

Parameters.

ch - timer interrupt channel.

edge - (0): rise, (1): fall.

set - (0): clear, (1): set.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function set/clears ELC interrupt indication. The indication is rise or fall edge.

Reentrant

Function is not reentrant.

Example

Example is same as “3.23 R_PTP_RegTmrHndr”.

Special Notes

None

3.25 R_PTP_ELC_SetClr ()

This function set/clears ELC interrupt auto clear mode.

Format

ptp_return_t R_PTP_ELC_SetClr(IntCycCh ch, uint8_t edge, uint8_t set);

Parameters.

ch - timer interrupt channel.

edge - (0): rise, (1): fall.

set - (0): clear, (1): set.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function set/clears ELC interrupt indication. The indication is rise or fall edge.

Reentrant

Function is not reentrant.

Example

Example is same as “3.23 R_PTP_RegTmrHndr”.

Special Notes

None

3.26 R_PTP_Tmr_Set ()

This function set and enables timer interrupt handler (timer event).

Format

ptp_return_t R_PTP_Tmr_Set(IntCycCh ch, UInt64 start, uint32_t cycle, uint32_t pulse);

Parameters.

ch - timer interrupt channel.

start - timer start time(ns).

cycle – pulse output cycle interval(ns).

pulse – pulse output pulse interval(ns).

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function set and enables timer interrupt handler (timer event). Following operations are executed.

- Set the indication timer interrupt to CPU

- Set event time, pulse output cycle interval and pulse output pulse interval to specified timer channel

- Enable timer event

Reentrant

Function is not reentrant.

Example

Example is same as “3.23 R_PTP_RegTmrHndr”.

Special Notes

Timer start time should be set after the local clock setting.

3.27 R_PTP_GetLcClk ()

This function gets current local clock counter value.

Format

ptp_return_t R_PTP_GetLcClk(Timestamp *clk);

Parameters.

clk – local clock.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR: Not implemented in this version

Properties

Prototyped in “r_ptp_rx_if.h” and "r_ptp_rx_private.h".

Description

This function gets current local clock counter value. Following operations are executed.

- Request local clock counter value

- Wait local clock counter loaded
  Call _R_PTP_Wait ()

- Save local clock counter
  The order of saving is nano sec, sec lo and sec high order.

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

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

   uint32_t localClockNsec; /* local clock counter nano sec field */
   uint32_t localClockSec;  /* local clock counter sec field */
   Timestamp lc_clk; /* local clock counter value */

   /* Get local clock */
   R_PTP_GetLcClk(&lc_clk);

   /* Save local clock */
   localClockSec = lc_clk.secondsField.lo;
   localClockNsec = lc_clk.nanosecondsField;

   printf("local clock (sec field) = %d\n", localClockSec);
   printf("local clock (nano sec field) = %d\n", localClockNSec);
```

Special Notes

Getting local counter value is added retardation in some degree by the reading operation.

3.28 R_PTP_SetLcClk ()

This function set local clock counter value.

Format

ptp_return_t R_PTP_SetLcClk(Timestamp *clk);

Parameters.

clk – local clock.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR: Not implemented in this version

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function set local clock counter value. Following operations are executed.

- Set local clock counter value

- Load setting value to local clock counter

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptp_rx_if.h"

   Timestamp lc_clk; /* local clock counter value */

   /* Set local clock initial value */
   lc_clk.secondsField.hi = 0x0000;
   lc_clk.secondsField.lo = 0x56FF7C08; /* PTP time of Mar 31, 2016 */
   lc_clk.nanosecondsField = 0x12345678; /* 0x12345678 */

   /* Set local clock */
   R_PTP_SetLcClk(&lc_clk);
```

Special Notes

Master clock set the master time. If slave clock was informed any timescale in advance, it is recommended to set the initial time based on that timescale before starting synchronization.

3.29 R_PTP_ChkW10 ()

This function waits worst10 values got and those values set as gradient limits.

Format

void R_PTP_ChkW10(void);

Parameters.

None

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_TOUT: Timeout error

Properties

Prototyped in “r_ptp_rx_if.h” and "r_ptp_rx_private.h".

Description

This function waits worst10 values loaded and those values set as gradient limits.

Call _R_PTP_Wait_Ext (), the operation complete is checking “1” W10D bit of the STSR register.

To wait worst10 values loaded, “for” statement (loop processing) in the _R_PTP_Wait_Ext () is used.

Reentrant

Function is not reentrant.

Example

Example is same as “3.21 R_PTP_SubConfig”.

Special Notes

This function is executed when gradient correction is applied and hardware worst10 setting (PTP_CFG_SYNC_MODE = PTP_SYNC_MODE2_HW).

3.30 R_PTP_GetW10 ()

This function gets current worst10 values.

Format

ptp_return_t R_PTP_GetW10(uint32_t *p_w10, uint32_t *m_w10);

Parameters.

None

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_TOUT: Timeout error

Properties

Prototyped in “r_ptp_rx_if.h” and "r_ptp_rx_private.h".

Description

This function gets current worst10 value. Following operations are executed.

- Request current worst10 values got

- Wait gradient worst10 values got
  Call _R_PTP_Wait_Ext ()

- Request current worst10 values loaded

- Wait gradient worst10 values loaded
  Call _R_PTP_InfoChk ()

- Save current worst10 values
  Save PW10VR/MW10VR high, middle and low field.

Reentrant

Function is not reentrant.

Example

Example is same as “3.21 R_PTP_SubConfig”.

Special Notes

If the argument (p_w10 or n_w10) is NULL pointer, the value does not get.

This function is executed when gradient correction is applied and software worst10 setting (PTP_CFG_SYNC_MODE = PTP_SYNC_MODE2_SW).

To wait worst10 values loaded, “for” statement (loop processing) in the _R_PTP_Wait_Ext () and “do while” statement _R_PTP_InfoChk () are used.

3.31 R_PTP_SetGradLimit ()

This function sets the gradient limit values.

Format

ptp_return_t R_PTP_SetGradLimit(uint32_t *p_lim, uint32_t *m_lim);

Parameters.

p_lim - Plus (positive) gradient limit value.

m_lim - Minus (negative) gradient limit value.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR: Not implemented in this version

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function sets the gradient limit values.

Set PLIMITR/MLIMITR high, middle and low field.

Reentrant

Function is reentrant.

Example

Example is same as “3.21 R_PTP_SubConfig”.

Special Notes

If the argument (p_lim or n_lim) is NULL pointer, the value does not set.

This function is executed when gradient correction is applied and software worst10 setting (PTP_CFG_SYNC_MODE = PTP_SYNC_MODE2_SW).

3.32 R_PTP_GetMPortID ()

This function gets master port identity.

Format

ptp_return_t R_PTP_GetMPortID(uint8_t ch, uint32_t *clk, uint16_t *port);

Parameters.

ch - Sync unit channel (SYNFP0 or SYNFP1).

clk - Master clock identity field.

port - Master port number field.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function gets master port identity. Following operations are executed.

- Get master clock id (high field and low field)

- Get master port number field

Reentrant

Function is reentrant.

Example

Example showing this function being used.

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

   uint32_t curClkId[2];  /* Current clock identify field (0:hi, 1:lo) */
   uint16_t curPortId;    /* Current port number field */
   uint32_t anceClkId[2]; /* Announce message's clock identify field (0:hi,
1:lo) */
   uint16_t ancePortId;   /* Announce message's port number field */

   /* Get current master port identity of SYNFP0 */
   R_PTP_GetMPortID(0, curClkId, &curPortId);

   if ((curClkId[0] != anceClkId[0]) || (curClkId[1] != anceClkId[1])
      || (curPortId != ancePortId))
   { /* Update master port identity to SYNFP0 */
      /* Set master port identity */
      R_PTP_SetMPortID(0, anceClkId, &ancePortId);

       printf("master clock identity hi, lo = %8x, %8x\n", anceClkId[0],
anceClkId[1]);
       printf("master port number = %4x\n", ancePortId);
   }
```

Special Notes

If the argument (clk or port) is NULL pointer, the value does not get.

3.33 R_PTP_SetMPortID ()

This function set master port identity.

Format

ptp_return_t R_PTP_SetMPortID(uint8_t ch, uint32_t *clk, uint16_t *port);

Parameters.

ch - Sync unit channel (SYNFP0 or SYNFP1).

clk - Master clock identity field.

port - Master port number field.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function set master port id. Following operations are executed.

- Set master clock id (high field and low field)

- Set master port number field

- Validate set values to SYNFP0 or SYNFP1

Reentrant

Function is not reentrant.

Example

Example is same as “3.32 R_PTP_GetMPortID”.

Special Notes

None

3.34 R_PTP_GetSyncConfig ()

This function gets PTP synchronous configuration (SYRFL1R, SYRFL2R, SYTRENR and SYCONFR).

Format

ptp_return_t R_PTP_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 – SYTRENR.

conf – SYCONFR.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function gets PTP synchronous configuration (SYRFL1R, SYRFL2R, SYTRENR and SYCONFR).
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 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_rx_if.h"

    ptp_return_t ret;
    uint32_t fil1; /* Current SYRFL1R (CH0) value */
    uint32_t fil2; /* Current SYRFL2R (CH0) value */
    uint32_t tren; /* Current SYTRENR (CH0) value */

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

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

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

Special Notes

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

3.35 R_PTP_SetSyncConfig ()

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

Format

ptp_return_t R_PTP_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 – SYTRENR.

conf – SYCONFR.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function set PTP synchronous configuration (SYRFL1R, SYRFL2R, SYTRENR and SYCONFR).
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 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_rx_if.h"

   ptp_return_t ret;
   uint32_t fil1_val[2]; /* SYRFL1R setting value, 0:SYNFP0 1:SYNFP1 */
   uint32_t fil2_val[2]; /* SYRFL2R setting value, 0:SYNFP0 1:SYNFP1 */
   uint32_t conf_val[2]; /* SYCONFR setting value, 0:SYNFP0 1:SYNFP1 */

   /* ==== P2P TC and OC Slave (SYNFP0) setting example ==== */
   /* Sync, Follow_Up, Pdelay_Req, Pdelay_Resp and Pdelay_Resp_Follow_Up
messages are SYNFP0 operation and PRC-TC transfer */
   /* Announce message is PTPEDMAC and PRC-TC transfer */
   fil1_val[0] = 0x44400663;

   /* Management and Signaling messages are PTPEDMAC and PRC-TC transfer */
```

```
    fil2_val[0] = 0x20000033;

    /* P2P TC */
    conf_val[0] = 0x00100028;

    /* Set fil1_val, fil2_val and conf_val to SYNFP0. (Not set the SYTRENR) */
    ret = R_PTP_SetSyncConfig(0, &fil1_val[0], &fil2_val[0], NULL,
&conf_val[0]);
    if (ret != PTP_OK)
    {
        goto Err_end; /* error */
    }

    /* Announce, Sync and Follow_Up messages are PRC-TC transfer */
    /* Pdelay_Req, Pdelay_Resp and Pdelay_Resp_Follow_Up messages are SYNFP0
operation */
    fil1_val[1] = 0x44400222;

    /* Management and Signaling messages are PTPEDMAC and PRC-TC transfer */
    fil2_val[1] = 0x20000033;

    /* P2P TC */
    conf_val[1] = 0x00100028;

    /* Set fil1_val, fil2_val and conf_val to SYNFP1. (Not set the SYTRENR) */
    ret = R_PTP_SetSyncConfig(1, &fil1_val[1], &fil2_val[1], NULL,
&conf_val[1]);
    if (ret != PTP_OK)
    {
        goto Err_end; /* error */
    }

    /* Complete set synchronous configuration */
```

Special Notes

Need to update synchronous configuration every when clock is changed such as BMC algorithm.

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

3.36 R_PTP_GetSyncInfo ()

This function gets current offsetFromMaster and meanPathDelay.

Format

ptp_return_t R_PTP_GetSyncInfo(uint8_t ch, TimeInterval *ofm, TimeInterval *mpd);

Parameters.

ch - Sync unit channel (SYNFP0 or SYNFP1).

ofm – offsetFromMaster.

mpd - meanPathDelay.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h” and "r_ptp_rx_private.h".

Description

This function gets current offsetFromMaster and meanPathDelay. Following operations are executed.

- Get offsetFromMaster
  Over flow case, change largest positive value (=0x7FFFFFFF, 0xFFFFFFFF).
  Under flow case, change smallest minimum value (=0x80000000, 0x00000000).
  The other case, Convert scaled nano second expression.

- Get meanPathDelay
  Over flow case, change largest positive value (=0x7FFFFFFF, 0xFFFFFFFF).
  Under flow case, change smallest minimum value (=0x80000000, 0x00000000).
  The other case, Convert scaled nano second expression.

- Update currentDS.offsetFromMaster and meanPathDelay

Reentrant

Function is reentrant.

Example

Example showing this function being used.

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

    ptp_return_t ret;
    TimeInterval cur_ofm; /* Current offsetFromMaster */
    TimeInterval cur_mpd; /* Current meanPathDelay */
    uint32_t ch; /* Synchronous channel */

    /* Get current synchronous channel */
    ch = R_PTP_GetSyncCH();

    /* Get current offsetFromMaster and meanPathDelay of the synchronous channel
*/
    ret = R_PTP_GetSyncInfo(ch, &cur_ofm, &cur_mpd);
    if (ret != PTP_OK)
    {
        goto Err_end; /* error */
    }

    /* Complete get current offsetFromMaster and meanPathDelay of the current
synchronous SYNFP channel */
```

```
   printf("Current offsetFromMaster high, low = %8x, %8x\n",
(uint32_t)(cur_ofm.scaledNanoseconds.hi), cur_ofm.scaledNanoseconds.lo);
   printf("Current meanPathDelay high, low = %8x, %8x\n",
(uint32_t)( cur_mpd.scaledNanoseconds.hi), cur_mpd.scaledNanoseconds.lo);
```

Special Notes

If the argument (ofm or mpd) is NULL pointer, the value does not get.

3.37 R_PTP_UpdClkID ()

This function updates my clockIdentity.

Format

ptp_return_t R_PTP_UpdClkID(uint8_t ch, int8_t *id);

Parameters.

ch - Sync unit channel (SYNFP0 or SYNFP1).

id - clockIdentity.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function updates my clockIdentity value. Following operations are executed.

- Set my clock id (high field and low field) (SYCIDRU/L)

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptp_rx_if.h"

   /* User defined clockIdentity: FF-05-23-45-67-89-AB-CD */
   int8_t MyClkId[8] = {0xFF, 0x05, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD};

   /* ==== Complete PTP device configuration ==== */
   ex. Executing R_PTP_Init function

   /* Set my clockIdentity of SYNFP1 */
   R_PTP_UpdClkID(1, MyClkId);
```

Special Notes

Default value of clockIdentity is EUI-48 format. In case of selecting your own format, please execute this function after PTP device configuration.

3.38 R_PTP_UpdDomainNum ()

This function updates domainNumber field of common message header.

Format

ptp_return_t R_PTP_UpdDomainNum(uint8_t ch, uint8_t dnum);

Parameters.

ch - Sync unit channel (SYNFP0 or SYNFP1).

dnum – updating domainNumber value.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function updates domainNumber value of common message header. Following operations are executed.

- Set domainNumber field value (SYDOMR)

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptp_rx_if.h"

   /* 0: Default domain, 1 to 3: Alternate domain, 4 to 127: User defined */
   uint8_t dnum = 0x01;

   /* Set domainNumber field value of SYNFP1 */
    R_PTP_UpdDomainNum(1, dnum);
```

Special Notes

None

3.39 R_PTP_UpdAnceFlags ()

This function updates Announce message's flag fields.

Format

ptp_return_t R_PTP_UpdAnceFlags(uint8_t ch, AnceFlag *flags);

Parameters.

ch - Sync unit channel (SYNFP0 or SYNFP1).

flags - updating flag values.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function updates Announce message's flag field values. Following operations are executed.

- Set flag field values (ANFR)
  PTP profile Specific 2/1, unicastFlag, alternateMasterFlag, frequencyTraceable, timeTraceable, ptpTimescale, currentUtcOffsetValid, leap59 and leap61 fields.

- Validate set values to SYNFP0 or SYNFP1

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptp_rx_if.h"

   AnceFlag flags;

   /* PTP profile Specific 2(b14) = false, PTP profile Specific 1(b13) = false
*/
   flags.BIT.profileSpec2 = 0;
   flags.BIT.profileSpec1 = 0;

   /* unicastFlag(b10) = false (not support unicast PTP in the RX64M/71M) */
   flags.BIT.unicastFlag = 0;

   flags.BIT.alternateMasterFlag = 0; /* alternateMasterFlag(b8) = false */

   flags.BIT.frequencyTraceable = 0; /* frequencyTraceable(b5) = false */

   flags.BIT.timeTraceable = 1; /* timeTraceable(b4) = true */

   flags.BIT.ptpTimescale = 1; /* ptpTimescale(b3) = true */

   flags.BIT.currentUtcOffsetValid = 1; /* currentUtcOffsetValid(b2) = true */

   flags.BIT.leap59 = 0; /* leap59(b1) = false */

   flags.BIT.leap61 = 0; /* leap61(b0)= false */

   /* Update announce messages flags field of SYNFP1 */
   R_PTP_UpdAnceFlags(1, &flags);
```

Special Notes
None

3.40 R_PTP_UpdAnceMsgs ()

This function updates Announce message's message fields.

Format

ptp_return_t R_PTP_UpdAnceMsgs(uint8_t ch, AnceMsg *msgs);

Parameters.

ch - Sync unit channel (SYNFP0 or SYNFP1).

msgs - updating field values.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function updates Announce message's message field values. Following operations are executed.

- Set grandmasterPriority2/1 fields (GMPR)

- Set grandmasterClockQuality fields (GMCQR)

- Set grandmasterIdentity fields (GMIDRU/L)

- Validate set values to SYNFP0 or SYNFP1

Reentrant

Function is reentrant.

Example

Example showing this function being used.

```
#include "r_ptp_rx_if.h"

   AnceMsg msgs;

   /* grandmasterIdentity: FF-05-23-45-67-89-AB-CD */
   int8_t id[8] = {0xFF, 0x05, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD};

   /* grandmasterPriority1: set priority level 1 */
   msgs.grandmasterPriority1 = 0x01;

   /* grandmasterPriority2: set priority level 2 */
   msgs.grandmasterPriority2 = 0x02;

   /* grandmasterClockQuality */
   /* b31 to b24: clockClass, default value(=248, = 0xF8), 255 is slave only
clock */
   msgs.grandmasterClockQuality.clockClass = 0xFF;

   /* b23 to b16: clockAccuracy, default value(=0x21)is within 100 nsec, 0x20
to 0x31, 0x23 is within 1us */
   msgs.grandmasterClockQuality.clockAccuracy = 0x23;

   /* b15 to b0: offsetScaledLogVariance, default value(=0xFFFF) is not
calculated yet */
   msgs.grandmasterClockQuality.offsetScaledLogVariance = 0xFFFF;
```

```
    /* set grandmasterIdentity */
    msgs.grandmasterIdentity = id;

    /* Update announce messages flags field of SYNFP1 */
    R_PTP_UpdAnceMsgs(1, &msgs);
```

Special Notes
None

3.41 R_PTP_UpdSyncAnceInterval ()

This function updates transmission interval and logMessageInterval of Sync and Announce messages.

Format

ptp_return_t R_PTP_UpdSyncAnceInterval(uint8_t ch, int8_t *sync, int8_t *ance);

Parameters.

ch - Sync unit channel (SYNFP0 or SYNFP1).

sync - Sync transmission interval ($2^{interval}$ sec).

ance – Announce transmission interval ($2^{interval}$ sec).

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function updates interval and logMessageInterval of Sync and Announce messages. Following operations are executed.

- Update Sync and Announce transmission interval value.
  If argument(=sync or ance) is less than 2^{-7} case, set 7.8125 msec.
  If argument is more than 2^{6} case, set 64 sec.
  The other case, set the argument value to Sync and Announce transmission interval value.

- Validate set values to SYNFP0 or SYNFP1

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptp_rx_if.h"

   int8_t sync; /* Update Sync transmission interval */

   /* Sync transmission interval is 500msec (=2-1s) */
   sync = 0xFF; /* 0xFF = -1 */

   /* Update transmission interval of Sync (no update announce interval) */
   R_PTP_UpdMsgInterval (1, &sync, NULL);
```

Special Notes

If the argument (sync or ance) is NULL pointer, the value does not update.

3.42 R_PTP_UpdDelayMsgInterval ()

This function updates transmission interval, logMessageInterval and timeout values of Delay messages.

Format

ptp_return_t R_PTP_UpdDelayMsgInterval(uint8_t ch, int8_t *interval, uint32_t *tout);

Parameters.

ch - Sync unit channel (SYNFP0 or SYNFP1).

interval – Case of master: Delay_Resp logMessageInterval (2^(interval) sec).

Case of slave or listening: Delay_Req/Pdelay_Req transmission interval ($2^{interval}$ sec).

*tout - Delay_Resp/Pdelay_Resp receiving timeout (tout * 1024 nsec).*

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

In case of master state, this function updates logMessageInterval field of Delay_Resp. In case of slave or listening state, this function updates transmission interval and logMessageInterval field of Delay_Req/Pdelay_Req adjust to Delay_Resp's logMessageInterval field. This function also updates receiving timeout value of Delay_Resp/Pdelay_Resp. Following operations are executed.

- Refer Delay_Resp's logMessageInterval field value.

- Update Delay_Resp's logMessageInterval field when PTP port state is master.
  If argument(=interval) is less than 2^{-7} case, set 7.8125 msec.
  If argument is more than 2^{6} case, set 64 sec.
  The other case, set the argument value.

- Update Delay_Req/Pdelay_Req transmission interval value adjust to Delay_Resp's logMessageInterval field when PTP port state is slave or listening.
  Less than 2^{-7} case, set 7.8125 msec.
  More than 2^{6} case, set 64 sec.
  The other case, set Delay_Resp's logMessageInterval field to Delay_Req/Pdelay_Req transmission interval value.

- Update Delay_Resp/Pdelay_Resp receiving timeout value.

- Validate set values to SYNFP0 or SYNFP1

Reentrant

Function is not reentrant.

Example

Example showing this function being used when PTP state is slave.

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

   ptp_return_t ret;
   int8_t interval; /* Updated Delay_Req transmission interval */
   uint32_t tout; /* Update timeout value */
```

```
   /* Delay_Resp/Pdelay_Resp receiving timeout value is 10.24sec (=10^6 *
1024nsec) */
   tout = 0x989680;

   /* Update transmission interval and logMessageInterval field of
Delay_Req/Pdelay_Req, and timeout values of Delay_Resp/Pdelay_Resp */
   ret = R_PTP_UpdDelayMsgInterval (1, &interval, &tout);
   if (PTP_OK != ret)
   {
      goto Err_end; /* error */
   }

   printf("Delay_Req/Pdelay_Req transmission interval = %f s\n", 2^(dreq));
```

Special Notes

If the argument (interval or tout) is NULL pointer, the value does not update.

3.43 R_PTP_Start ()

This function starts synchronization.

Format

ptp_return_t R_PTP_Start(void);

Parameters.

None

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_TOUT: Timeout error
PTP_ERR: Any error occurred

Properties

Prototyped in “r_ptp_rx_if.h” and "r_ptp_rx_private.h".

Description

This function starts synchronization. Following operations are executed.

- Set PTP messages transmit or not depend on the clock. The setting of Table 3.1 is applied.

Table 3.1 Transmission Table of PTP Messages

Clock	OC and BC				TC	
Sync mode	P2P		E2E		P2P	E2E
Master or Slave	Master	Slave	Master	Slave		
Pdelay_Req	Trans	Trans	Not	Not	Trans	Not
Delay_Req	Not	Not	Not	Trans	Not	Not
Sync	Trans	Not	Trans	Not	Not	Not
Announce	Trans	Not	Trans	Not	Not	Not

- Case of E2E Slave, check the offsetFromMaster is updated or not.

If the offsetFromMaster was not updated, wait it will be updating.
Call _R_PTP_Wait ().

- Validate set values to SYNFP0 or SYNFP1

- Case of Slave, start Slave time synchronization (set SYNSTARTR.STR)

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptpif_rx_if.h"
#include "r_ptp_rx_if.h"

    ptpif_return_t ptpif_ret;
    ptp_return_t ptp_ret;
    int32_t i;
    PTPConfig ptpc; /* PTP device information */

    /* ==== Open and link standard Ether ==== */
    /* Ref 3.2 R_PTPIF_Reset() example for more detail information. */
```

```
/* ==== Open PTP and PTP Host interface ==== */
/* Reset PTPEDMAC */
R_PTPIF_Reset();

/* Reset EPTPC */
R_PTP_Reset();

/* Initialize resources of the PTP driver */
R_PTPIF_Init();

/* Initialize EPTPC */
ptp_ret = R_PTP_Init(&ptpc);
if (PTP_OK != ptp_ret)
{
    goto Err_end;
}

/* Initialize PTP Host interface and peripheral modules */
ptpif_ret = R_PTPIF_Open_ZC2();
if (PTPIF_OK != ptpif_ret)
{
    goto Err_end;
}

/* Set PTP Host interface to transfer PTP message */
R_PTPIF_LinkProcess();

/* Check PTP Host interface status */
ptpif_ret = R_PTPIF_CheckLink_ZC();
if (PTPIF_OK != ptpif_ret)
{
   goto Err_end;
}

/* Set PTP port sate to SLAVE state of SYNFP1. */
ptp_ret = R_PTP_SetPortState(1, ST_SLAVE);
if (ptp_ret != PTP_OK)
{
    goto Err_end; /* error */
}

/* Start synchronization */
ptp_ret = R_PTP_Start();
if (ptp_ret != PTP_OK)
{
    goto Err_end; /* error */
}

while(1)
{ /* Continue synchronization */
    if (g_stop_SyncFlag == 1)
   { /* Detect synchronization stop flag */
       /* Stop synchronization */
       R_PTP_Stop();
   }
}
```

Special Notes

This function need executes after the PTP configuration is set.

This function is valid when synchronization was not started.

To wait offsetFromMaster updated, “for” statement (loop processing) in the _R_PTP_Wait () is used.

3.44 R_PTP_Stop ()

This function stops synchronization.

Format

ptp_return_t R_PTP_Stop(void);

Parameters.

None

Return Values

PTP_OK: Processing completed successfully
PTP_ERR: Any error occurred

Properties

Prototyped in “r_ptp_rx_if.h” and "r_ptp_rx_private.h".

Description

This function stops synchronization. Following operations are executed.

- Stop PTP message transmission

- Set PTP messages transmission filter 1 (SYRFL1R)
  Set only announce message transfers to PTPEDMAC and the other messages are all canceled.

- Case of E2E Slave, check whether the offsetFromMaster is updated or not.
  If the offsetFromMaster was not updated, wait it will be updating.
  Call _R_PTP_Wait().

- Validate set values to SYNFP0 or SYNFP1

- Case of Slave, stop Slave time synchronization (clear SYNSTARTR.STR)

Reentrant

Function is not reentrant.

Example

Example is same as “3.43 R_PTP_Start”.

Special Notes

This function is valid when synchronization was started.

To finalize time synchronous operation, “for” statement (loop processing) in the _R_PTP_Wait () is used.

3.45 R_PTP_SetPortState ()

This function set PTP port state.

Format

ptp_return_t R_PTP_SetPortState (uint8_t ch, PTPState state);

Parameters.

ch - Sync unit channel (SYNFP0 or SYNFP1).

state - updating port state (ST_MASTER, ST_SLAVE or ST_LIST)

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function update PTP port state (ST_MASTER, ST_SLAVE or ST_LIST)

Reentrant

Function is reentrant.

Example

Example is same as “3.43 R_PTP_Start”.

Special Notes

None

3.46 R_PTP_GetSyncCH ()

This function gets current synchronous channel.

Format

uint32_t R_PTP_GetSyncCH(void);

Parameters

None

Return Values

Current synchronous channel.

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function gets current synchronous channel.

Reentrant

Function is reentrant.

Example

Example is same as “3.36 R_PTP_GetSyncCH”.

Special Notes

None

3.47 R_PTP_SetInterrupt ()

This function enables EPTPC INFABT interrupt.

Format

ptp_return_t R_PTP_SetInterrupt(uint8_t ch);

Parameters

ch - Sync unit channel (SYNFP0 or SYNFP1).

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_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_rx_if.h"
#include "r_ether_if.h"

   ptp_return_t ret; /* PTP driver return code */
   bool is_det; /* INABT interrupt detection flag */

   /* Standard Ethernet open and link were completed */

   /* PTP open was completed */

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

   while(1)
   {
       ret = R_PTP_ChkInterrupt(0, &is_det);
       if (PTP_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_PTP_Reset();
```

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

            /* Thereafter, please execute retrieve operation */

        }
    }
```

Special Notes
None

3.48 R_PTP_ChkInterrupt ()

This function checks INFABT interrupt occurrence.

Format

ptp_return_t R_PTP_ChkInterrupt(uint8_t ch, bool *is_det);

Parameters

ch - Sync unit channel (SYNFP0 or SYNFP1).

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_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 “3.47 R_PTP_SetInterrupt”.

Special Notes

None

3.49 R_PTP_ClrInterrupt ()

This function clears INFABT interrupt occurrence flag.

Format

ptp_return_t R_PTP_ClrInterrupt (uint8_t ch);

Parameters

ch - Sync unit channel (SYNFP0 or SYNFP1).

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_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 “3.47 R_PTP_SetInterrupt”.

Special Notes

None

3.50 R_PTP_DisableTmr ()

This function disables timer event interrupt.

Format

void R_PTP_DisableTmr (void);

Parameters

None

Return Values

None

Properties

Prototyped in “r_ptp_rx_if.h".

Description

This function disables timer event interrupt. The following operations are executed.

- Clear interrupt element bit of MIESR status register.

- Disable timer event interrupt.

Reentrant

Function is not reentrant.

Example

Example is same as “3.23 R_PTP_RegTmrHndr”. That example explains registration method of this function to timer event handler. When 1st time timer event occurs, this function disables the following timer events.

Special Notes

None

3.51 R_PTP_SetSyncDet ()

This function set detect condition of change of synchronous state.

Format

ptp_return_t R_PTP_SetSyncDet (SyncDet *dev, SyncDet *syn, bool is_enab);

Parameters.

dev - Condition of deviation state detection.

syn - Condition of synchronous state detection.

is_enb – Alarm enable or not: (1) enable, (0) disable.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function set detect condition of change of synchronous state. The following operations are executed.

- Check the detect conditions of successive time occurrence of state changing.

- Set detect condition of deviation sate.

- Set detect condition of synchronous sate.

- Set or clear alarm indication.

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

```
#include "r_ptp_rx_if.h"

   SyncDet dev, syn;

   /* Register user interrupt handler for STCA SYNC and SYNCOUT */
   R_PTP_RegMINTHndr(MINT_FUNC_STCA, 0x00000003, (MINT_HNDLR)user_mint_func);

   /* Set detect condition of deviation state */
   dev.th_val.hi = 0x00000000;
   dev.th_val.lo = 0x00000500; /* 0x500 ns */
   dev.times = 1;

   /* Set detect condition of synchronous state */
   syn.th_val.hi = 0x00000000;
   syn.th_val.lo = 0x00000020; /* 0x20 ns */
   syn.times = 1;

   R_PTP_SetSyncDet(&dev, &syn, true);

   /* wait interrupt from STCA */

   /* interrupt occurred (call user_mint_func) */
```

Special Notes

None

3.52 R_PTP_SetSynctout ()

This function set detect condition of sync message reception timeout.

Format

ptp_return_t R_PTP_SetSynctout (uint32_t tout, bool is_enab);

Parameters.

tout - Sync message reception timeout value (1024ns unit).

is_enb – Alarm enable or not: (1) enable, (0) disable.

Return Values

PTP_OK: Processing completed successfully
PTP_ERR_PARAM: Parameter error

Properties

Prototyped in “r_ptp_rx_if.h”.

Description

This function set detect condition of sync message reception timeout. The following operations are executed.

- Set timeout value.

- Set or clear alarm indication.

Reentrant

Function is not reentrant.

Example

Example showing this function being used.

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

   ptp_return_t ret;
   uint32_t tout;

   /* Register user interrupt handler for STCA SYNTOUT */
   R_PTP_RegMINTHndr(MINT_FUNC_STCA, 0x00000008, (MINT_HNDLR)user_mint_func);

   /* Set detect condition of sync message reception timeout */
   tout = 0x00200000; /* approx. 2s */
   ret = R_PTP_SetSynctout(tout, true);
   if (PTPL_OK != ret)
   {
      goto Err_end; /* error */
   }
   printf("Set sync reception timeout : timeout value = %d\n", tout);

   /* wait interrupt from STCA */

   /* interrupt occurred (call user_mint_func) */
```

Special Notes

None

4. Appendices

4.1 Internal Functions

The PTP 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_PTPIF_InitDescriptors()	Initialize descriptors and buffers of PTPEDMAC.
_R_PTPIF_ConfigEthernet()	Initialize PTPEDMAC.
_R_PTP_Wait()	Wait PTP event operation completed. (timeout 100msec)
_R_PTP_Wait_Ext()	Wait PTP event operation completed. (timeout 400sec)
_R_PTP_InfoChk()	Wait information retention of GETINFOR.
_R_PTP_Int_STCA()	Interrupt handler of STCA.
_R_PTP_Int_PRC()	Interrupt handler of PRC-TC.
_R_PTP_Int_SYNFP0()	Interrupt handler and set INFABT flag of SYNFP0.
_R_PTP_Int_SYNFP1()	Interrupt handler and set INFABT flag of SYNFP1.
_R_PTP_Init_SYNFP()	Set SYNFP parameters by initial values.
_R_PTP_Init_PRC_STCA()	Set PRC-TC and STCA parameters by initial values.

4.2 Related Ether Driver's API

The PTP driver always should be used with the Ether driver. As for the information of the Ether driver's API to which the PTP driver mandatory use, please refer to "RX Family Ethernet Module Using Firmware Integration Technology [2]". The typical usage of those API, describes Sec 3 as the example.

4.3 Additional Standard Ethernet Functionalities

The PTP driver implements new functionalities but for the time synchronization based on the PTP. Those functionalities are simple switch[1] and MC (Multicast) frame filter. 3.17 R_PTP_SetTran function sets the simple switch and 3.18 R_PTP_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 3.35 R_PTP_SetSyncConfig function.

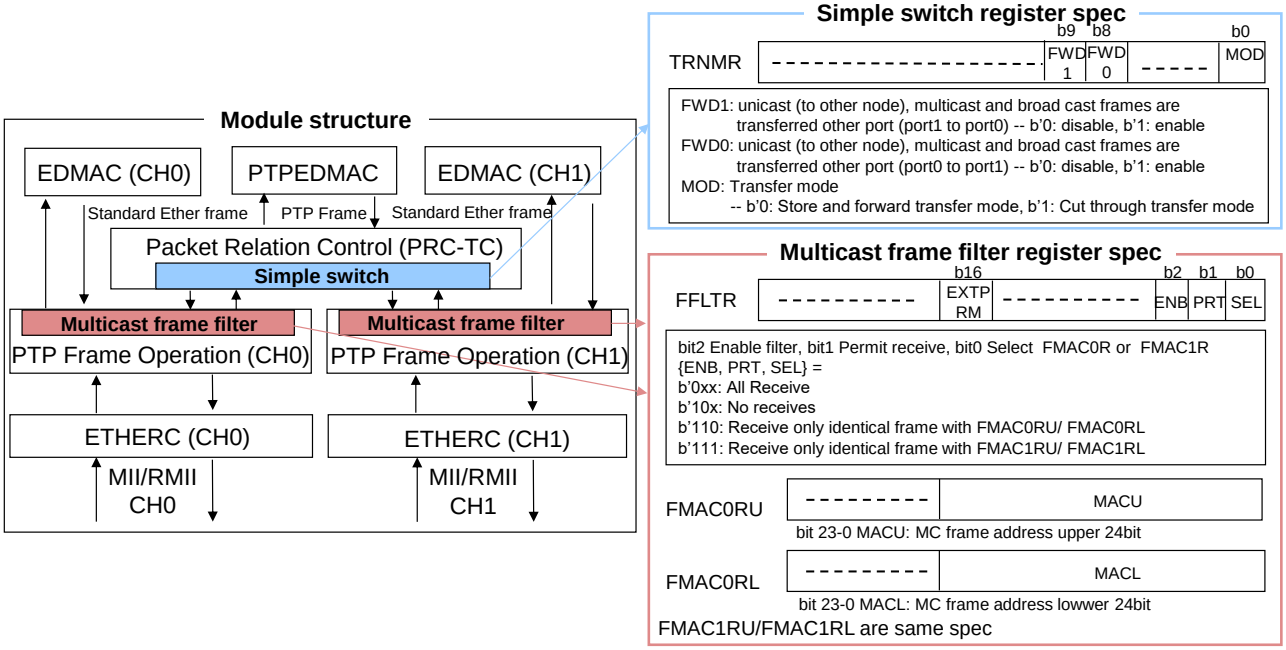
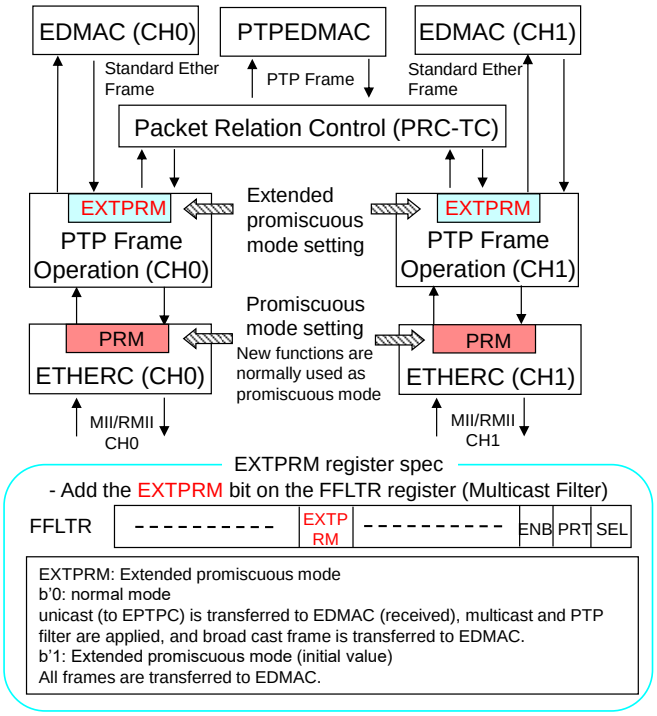


Figure 4.1 Additional standard Ethernet functionalities

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 3.19 R_PTP_SetExtPromiscuous. As for the setting of those modes, please refer to “RX Family Ethernet Module Using Firmware Integration Technology [2]”. The promiscuous mode can be set using the R_PTP_SetExtPromiscuous API described in the 3.19. 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 Section Allocation

Table 4.2 shows a sample section allocation for PTP driver. It also shows of the section allocation for Ether driver combination with PTP driver.

Table 4.2 Section Allocation Example

Address	Device	Section	Description
0x00000020	Internal RAM	SI	Interrupt stack area
		SU	User stack area
		B_1	Uninitialized data area of 1byte boundary
		R_1	Initialized data area of 1byte boundary (variable)
		B_2	Uninitialized data area of 2byte boundary
		R_2	Initialized data area of 2byte boundary (variable)
		B	Uninitialized data area of 4byte boundary
		R	Initialized data area of 4byte boundary (variable)
0x00010000		B_ETHERNET_BUFFERS_1	Transmit buffer and receive buffer area of standard Ethernet frames
		B_RX_DESC_1	Receive descriptor area of standard Ethernet frames
		B_TX_DESC_1	Transmit descriptor area of standard Ethernet frames
0x00018000		B_PTPIF_BUFFER_1	Transmit buffer and receive buffer area of PTP message frames
		B_PTPIF_RX_DESC_1	Receive descriptor area of PTP message frames
		B_PTPIF_TX_DESC_1	Transmit descriptor area of PTP message frames
0xFFF80000	Internal ROM	C_1	Constant area of 1byte boundary
		C_2	Constant area of 2byte boundary
		C	Constant area of 4byte boundary
		C$*	Constant region (CDEC, CBSEC, C$VECT) of C$* section
		D_1	Initialization data area of 1byte boundary
		D_2	Initialization data area of 2byte boundary
		D	Initialization data area of 4byte boundary
		P	Program area
		W_1	Branch table area for switch statements of 1byte boundary
		W_2	Branch table area for switch statements of 2byte boundary
		W	Branch table area for switch statements of 4byte boundary
		L	String literal area
0xFFFFFF80		EXCEPTVECT	Interrupt vector area
0xFFFFFFFC		RESETVECT	Reset vector area

4.5.1 Notes on Section Allocation

- Since the EDMAC mode register (EDMR) transmit/receive descriptor length bits (DL) are set to specify 16 bytes, sections must be allocated on 16-byte boundaries[1].
- Transmit buffer and receive buffer areas must be allocated on 32-byte boundaries[1].

[1]Both of PTP message frames and standard Ethernet frames need to allocate

4.6 Confirmed Operation Environment

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

Table 4.3 Confirmed Operation Environment

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.1.16
Board used	Renesas Starter Kit+ for RX64M Renesas Starter Kit+ for RX71M Renesas Starter Kit+ for RX72M

4.7 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_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_rx_config.h” may be wrong. Check the file “r_ptp_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)
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

Rev.	Date	Description	
		Page	**Summary**
1.00	Jul 10, 2014	**—**	First edition issued.
1.01	Aug 10, 2014	**—**	State transit function and worst10 setting added.
1.02	Dec 31, 2014	**—**	Supported RX71M device and added BMC function. Changed modules name (eliminate "_api").
1.10	Mar 31, 2016	**—**	Data structures changed.
1.11	May 15, 2016	**—**	Section information added.
1.12	Nov 11, 2016	**19, 40**	Corrected the internal operation of getting version functions.
		49	Corrected the array index deviation of ptp_tmr_isr.
		79	Corrected wait operation to information retention of GETINFOR and added it as the internal function (_R_PTP_InfoChk).
		80	Corrected frame propagation list (Fig 4.2).
1.13	Mar 31, 2017	**—**	Corrected BC and P2P TC setting.
		48, 81, 84	Modified listening state operations of R_PTP_Init, R_PTP_Start and R_PTP_Stop functions.
		51	Added registering user function to MINT interrupt handler.
		—	Changed MINT interrupt handler operation.
		—	Changed TC&OC combined operation to TC only operation.
		72	Added my clockIdentity setting function.
		73	Added domainnumber field updating function.
		74, 76	Added announce message fields updating function.
		78, 79	Added and modified transmission interval setting function.
		81	Added offsetFromMaster updating wait function when device state is P2P slave.
		85	Added PTP port state setting function.
1.14	Apr 30, 2017	**60**	Changed the description of special notes.
		70	Corrected underflow operation of R_PTP_GetSyncInfo function.
		78, 79	Corrected logMessageInterval out of range judgment.
1.15	Jul 31, 2019	**—**	Added support for GNUC and ICCRX.
		—	Added “WAIT_LOOP” comments for every loop operation.
		10	Added the explanation of using interrupt vectors.
		20	Added the explanation of callback function.
		20	Updated code size and adapted to GNUC and ICCRX.
		21	Updated how to add the FIT module to your project.
		21	Added the explanation about comment of loop operation.
		24, 45	Changed using R_BSP_SoftwareDelay function to wait reset completion.
		38	Corrected RACT bit setting sequence in the R_PTPIF_Read_ZC2_BufRelease function.
		45	Added reset release waiting operation in the R_PTP_Reset function.
		63	Corrected to wait loading worst10 value by calling _R_PTP_InfoChk function.
		96	Changed allocation of sections.
		97	Added “Confirmed Operation Environment” section.
		97	Added “Troubleshooting” section.
1.16	Aug 31, 2019	**—**	Supported RX72M device.
		—	Added Bypass mode clear setting.
		18	Added structure to detect synchronous state change.
		18	Added members that detect state deviation and synchronous status to PTPSubConfig structure.

94	Added R_PTP_SetSyncDet function which set detect condition of change of synchronous sate.
95	Added R_PTP_SetSynctout function which set detect condition of sync message reception timeout.

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/.