RX Family

EPTPC モジュール　Firmware Integration Technology

要旨

本アプリケーションノートでは、Firmware Integration Technology（FIT）を使用した PTP ソフトウェアドライバ（PTP ドライバ、EPTPC FIT モジュール）の内容を説明します。PTP ドライバは IEEE1588-2008 [1]で規定された Precision Time Protocol（PTP）に基づいて、時刻同期を行います。

動作確認デバイス

以下のデバイスがこのソフトウェアでサポートされています。

- RX64M グループ
- RX71M グループ
- RX72M グループ

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

目次

1. 概要

1.1 EPTPC FIT モジュールとは

本モジュールは API として、プロジェクトに組み込んで使用します。本モジュールの組み込み方については、「2.14 FIT モジュールの追加方法」を参照してください。

1.2 EPTPC FIT モジュールの概要

本アプリケーションノートでは、Firmware Integration Technology（FIT）に基づいた PTP ドライバと、その使用例を説明します。PTP ドライバは EPTPC 周辺モジュールを使用して PTP による時刻同期を行います。

1.3 関連ドキュメント

[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 ファミリ イーサネットモジュール Firmware Integration Technology, Rev.1.17, Document No. R01AN2009JJ0117, 発行予定
[3] RX ファミリ EPTPC：PTP 同期時刻取得サンプルプログラム, Rev.1.12, Document No. R01AN1983JJ0112, Mar 31, 2017
[4] RX ファミリ EPTPC：PTP タイマ同期開始サンプルプログラム, Rev.1.12, Document No. R01AN1984JJ0112, Mar 31, 2017
[5] RX ファミリ EPTPC：同期パルス出力サンプルプログラム, Rev.1.12, Document No. R01AN2846JJ0112, Mar 31, 2017
[6] RX ファミリ イーサネットコントローラ：簡易スイッチ動作例、Rev.1.11, Document No. R01AN3036JJ0111, Nov 11, 2016
[7] RX ファミリ イーサネットコントローラ：マルチキャストフレームフィルタ動作例、Rev.1.11, Document No. R01AN3037JJ0111, Nov 11, 2016
[8] Renesas Starter Kit+ for RX64M, ユーザーズマニュアル、Rev.1.20, Document No. R20UT2590JG0102, Jun 25, 2015
[9] Renesas Starter Kit+ for RX71M, ユーザーズマニュアル、Rev.1.00, Document No. R20UT3217JG0100, Jan 23, 2015

1.4 語彙・略語

- IEEE1588
  通信ネットワークにおける時刻同期の規格。通常、通信ネットワークとしてはイーサネットを使用します。IEEE1588-2002（バージョン 1）と IEEE1588-2008（バージョン 2）の 2 つのバージョンが存在し、互いに完全な互換性を持ってはいません。当アプリケーションノートでは IEEE1588-2008（バージョン 2）のみに準拠しています。
- PTP（Precision Time Protocol）
  PTP は IEEE1588 に定められている時刻同期のプロトコルです。
- PTP メッセージ
  PTP 手順で使用されるデータの形式です。PTP メッセージはイーサネットフレーム（レイヤ 2）もしくは UDP パケット（レイヤ 3）として送出されます。
- クロック（ノード）
  PTP では IEEE1588 規格に準じた時刻同期の機能を持つデバイスを意味します。
- ローカルクロック
  個々のクロックでの同期した時刻を意味します。
- マスタ
  マスタは、他のクロックに対して基準となる時刻を配信するクロックです。
- スレーブ
  スレーブはシステムの基準となる時刻を受け取り、自らをこれに合わせるクロックです。
- OC（Ordinary Clock）
  OC は 1 個のポートと、1 個のローカルクロックのみを有するクロックです。
- BC（Boundary Clock）
  BC は複数のポートと、それらで共用する 1 個のローカルクロックを持つクロックです。個々のポートはそれぞれ時刻同期機能を持っています。
- TC（Transparent Clock）
  TC は複数のポートを持ち、入力ポートから出力ポートへのフレームの伝播遅延を補正する働きを持つクロックです。
- E2E（End to End）
  1 個のマスタと 1 個もしくは複数のスレーブとの間で行われる同期モードを意味します。
- P2P（Peer to Peer）
  特定の 2 個のクロック間で行われる同期モードを意味します。
- STCA（Statistical Time Correction Algorithm）
  （ワースト 10 フィルタ）処理された複数の時刻値から時刻のずれる傾向（傾き値）を統計的に計算し、その結果を基に offsetFromMaster[1] を補正するアルゴリズムを意味しています。
- BMC (Best Master Clock) アルゴリズム
  BMC アルゴリズムは IEEE1588 のネットワーク内で最適なマスタを選択するアルゴリズムで、データセット比較アルゴリズムと状態決定アルゴリズムがあります。データセット比較アルゴリズムは個々のクロックの特性を比較によりマスタクロック（またはマスタポート[2]）を決定します。状態決定アルゴリズムはデータセット比較アルゴリズムの結果、遷移するクロックの状態を決定します。

[1] マスタの時刻とスレーブの時刻との間の時刻差異[1]

[2] クロックが持つ 1 個のポート

1.5　ハードウェアの構成

RX64M/71M/72M グループのイーサネットモジュールは、EPTPC、PTP 用イーサネットコントローラ用 DMA コントロートローラ（PTPEDMAC）、2 チャネルのイーサネットコントローラ（ETHERC（CH0）、ETHERC（CH1））、および 2 チャネルのイーサネットコントローラ用 DMA コントローラ（EDMAC（CH0）、EDMAC（CH1））で構成しています。EPTPC は、PTP 同期フレーム処理部（CH0）、PTP 同期フレーム処理部（CH1）、パケット中継部（PRC-TC）、および統計的クロック補正部で構成しています。また、EPTPC は、I/O ポートとモータ制御タイマ（MTU3 および GPT 周辺モジュール）に ELC 周辺モジュールを介して接続しており、PTP で時刻同期したパルスを出力できます。

イーサネットモジュールの概要を以下に示します。また、ハードウェアブロック図を図 1.1 に示します。

1. 時刻同期機能
・IEEE1588-2008 バージョン2 に準拠
・PTP メッセージ（イーサネットフレーム[1]とUDP/IPv4 フォーマット[2]）による時刻同期
・マスタまたはスレーブとして動作し、OC、BC、TC デバイスのいずれにも対応
・時刻偏差の統計的な補正（傾き予測時刻補正アルゴリズム）
・タイマイベント出力（6 チャネル、立ち上がり/立ち下がりエッジ、イベントフラグの自動クリア）
・ELC 経由のタイマイベントで、モータ制御タイマ（MTU3、GPT）の同期スタート
・EPTPCとELC経由で接続したI/OポートからPTPで時刻同期したパルスを出力
・PTP メッセージの処理を選択可能（PTP モジュール内部での処理、PTPEDMAC 経由のCPU 処理、他のポートを使用）
2. 標準イーサネット機能
・独立した2 チャネルのイーサネット動作
・ハードウェア簡易スイッチ（フレームの中継はカットスルー方式またはストア&フォワード方式から選択可能）
・ハードウェアマルチキャストフレームのフィルタリング（全て受信、全て受信しない、特定の2 種類のフレームのみ受信）

[1] RX64M グループではEthernet II フレームフォーマットのみサポートします（IEEE802.3 フレームフォーマットはサポートしません）。
[2] UDP IPv6 はサポートしません。

詳細に関しては「RX64M/71M グループユーザーズマニュアル ハードウェア編」を参照ください。

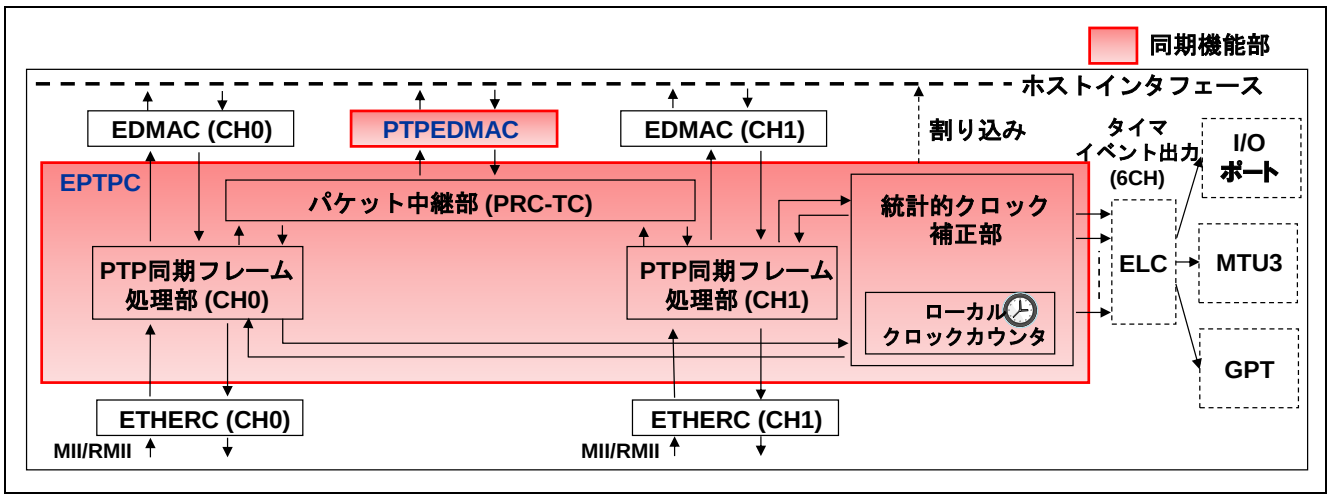


図 1.1　ハードウェアブロック図

1.6　ソフトウェアの構成

PTP ドライバは必ず Ether ドライバ[2]と組み合わせて使用し、TCP/IP システムに適用する場合は、TCP/IP ミドルウェアと組み合わせる必要があります。PTP ドライバはモータ制御システム（または PWM パルス出力システム）に適用する場合、MTU3/GPT ドライバ（または I/O ポートドライバ）と ELC ドライバと共に使用します。ソフトウェアの典型的な構成と機能の概要を図 1.2 に示します。

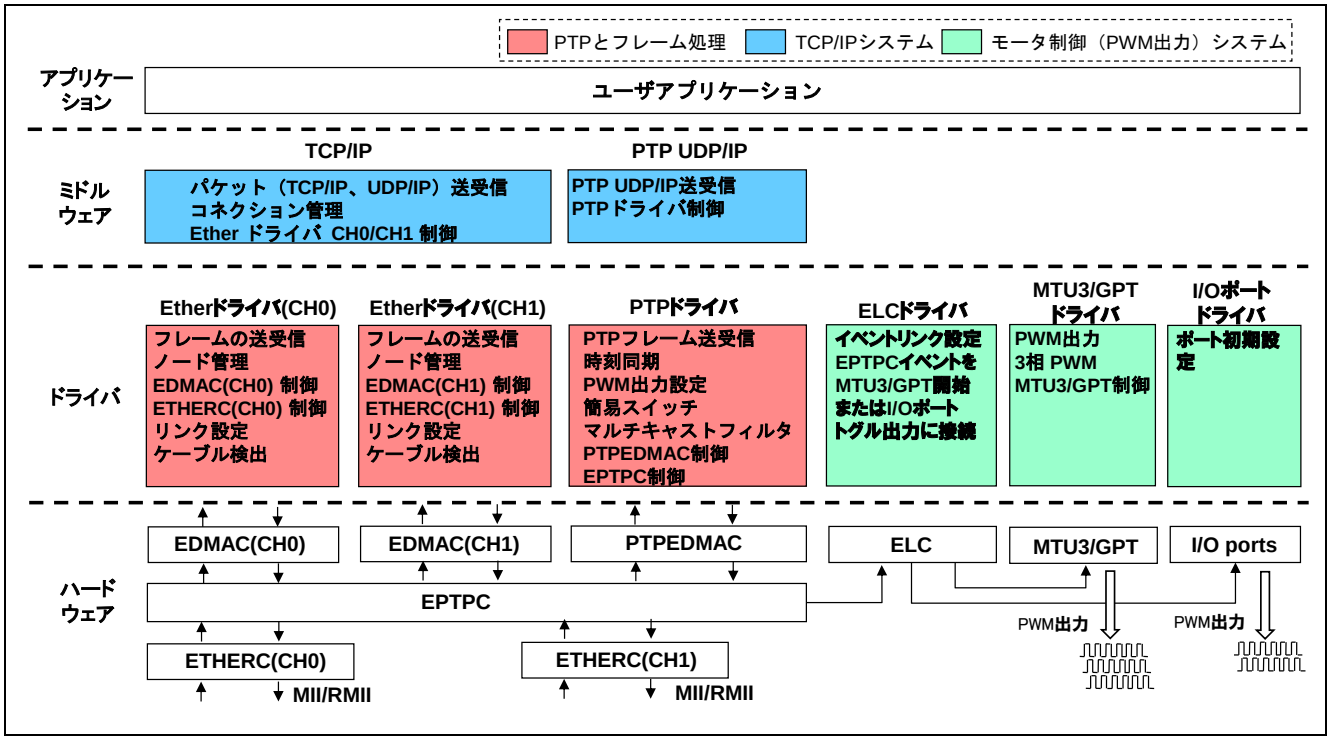


図 1.2　ソフトウェア構成例

1.7　ファイル構成

PTP ドライバは PTP ホストインタフェース部と PTP 時刻同期部で構成しています。PTP ホストインタフェース部は PTPEDMAC を介して PTP メッセージの送受信を行い、ソースコードは r_ptpif.c に記述しています。PTP 同期部は EPTPC を使用して PTP に準拠した時刻同期を行い、ソースコードは r_ptp.c に記述しています。

1.8 API の概要

PTP ホストインタフェース部の API 関数の一覧を表 1.1 に、PTP 時刻同期部の API 関数の一覧を表 1.2 に示します。

表 1.1 API 関数（PTP ホストインタフェース部）

関数	説明
R_PTPIF_GetVersion()	PTP ホストインタフェース部のドライババージョン番号の取得
R_PTPIF_Reset()	PTPEDMAC のリセット
R_PTPIF_Init()	PTP ホストインタフェースドライバのリソースの初期化
R_PTPIF_Open_ZC2()	PTP ホストインタフェースと周辺モジュールの初期化
R_PTPIF_LinkProcess()	PTP インタフェースの PTP メッセージ送受信設定
R_PTPIF_CheckLink_ZC()	PTP ホストインタフェースの状態確認
R_PTPIF_Close_ZC2()	PTP ホストインタフェース周辺モジュールの終了処理
R_PTPIF_Read()	PTP メッセージの受信
R_PTPIF_Write()	PTP メッセージまたは標準イーサネットフレームの送信
R_PTPIF_Read_ZC2()	PTP メッセージまたはその一部の受信。 受信データバッファの設定。
R_PTPIF_Read_ZC2_BufRelease()	受信データバッファの開放
R_PTPIF_Write_ZC2_GetBuf()	送信データバッファの設定
R_PTPIF_Write_ZC2_SetBuf()	PTP メッセージまたは標準イーサネットフレームの送信。 送信データのディスクリプタ設定。
R_PTPIF_RegMsgHndr()	PTPEDMAC の割り込みハンドラへのユーザ関数の登録

表 1.2　API 関数（PTP 時刻同期部）

関数	説明
R_PTP_GetVersion()	PTP 時刻同期部のドライババージョン番号の取得
R_PTP_Reset()	EPTPC のリセット
R_PTP_SetTran()	ポート間の転送モードの設定
R_PTP_SetMCFilter()	マルチキャストフレーム（MC）のフィルタ（FFLTR）設定
R_PTP_SetExtPromiscuous()	拡張プロミスキャスモードの設定と解除
R_PTP_Init()	デバイス構成に従った EPTPC の初期設定
R_PTP_SubConfig()	EPTPC のオプション構成の設定。（現バージョンはワースト 10 取得回数設定のみ）
R_PTP_RegMINTHndr()	EPTPC の MINT 割り込みハンドラへのユーザ関数の登録
R_PTP_RegTmrHndr()	EPTPC のタイマ割り込みハンドラへのユーザ関数の登録
R_PTP_ELC_Ind()	ELC 割り込み通知の設定と解除
R_PTP_ELC_SetClr()	ELC 割り込みの自動クリアモードの設定と解除
R_PTP_Tmr_Set()	タイマ割り込みハンドラ（タイマイベント）の設定と有効化
R_PTP_GetLcClk()	現在のローカルクロックカウンタの取得
R_PTP_SetLcClk()	ローカルクロックカウンタの初期値設定
R_PTP_ChkW10()	ワースト 10 の取得と傾き制限値設定の完了待ち（ハードウェアによる傾き上限値設定で使用）
R_PTP_GetW10()	現在のワースト 10 の取得（ソフトウェアによるワースト 10 取得で使用）
R_PTP_SetGradLimit()	傾き制限値の設定（ソフトウェアによる傾き上限値設定で使用）
R_PTP_GetMPortID()	マスタの PortIdentify の取得
R_PTP_SetMPortID()	マスタの PortIdentify の設定
R_PTP_GetSyncConfig()	PTP フレーム制御の構成（SYRFL1R、SYRFL2R、SYTRENR、SYCONFR）取得
R_PTP_SetSyncConfig()	PTP フレーム制御の構成（SYRFL1R、SYRFL2R、SYTRENR、SYCONFR）設定
R_PTP_GetSyncInfo()	offsetFromMaster と meanPathDelay の取得
R_PTP_UpdClkID()	自身の clockIdentity の更新設定
R_PTP_UpdDomainNum()	PTP メッセージヘッダの domainNumber フィールドの更新設定
R_PTP_UpdAnceFlags()	Announce メッセージのフラグフィールドの更新設定
R_PTP_UpdAnceMsgs()	Announce メッセージのメッセージフィールドの更新設定
R_PTP_UpdSyncAnceInterval()	Sync と Announce メッセージの送信間隔の更新設定
R_PTP_UpdDelayMsgInterval()	Delay メッセージの送信間隔とタイムアウト値の更新設定
R_PTP_Start()	同期開始
R_PTP_Stop()	同期終了
R_PTP_SetPortState()	PTP ポート状態の設定
R_PTP_GetSyncCH()	選択中の同期チャネルの取得
R_PTP_SetInterrupt()	EPTPC の INFABT 要因割り込みの有効化
R_PTP_ChkInterrupt()	INFABT 割り込み要因の確認
R_PTP_ClrInterrupt()	INFABT 割り込み要因検出フラグのクリア
R_PTP_DisableTmr()	タイマ割り込みの無効化
R_PTP_SetSyncDet()	同期状態変化の検出条件設定
R_PTP_SetSynctout()	Sync メッセージ受信タイムアウトの検出条件設定

2. API 情報

本 FIT モジュールは、下記の条件で動作を確認しています。

2.1 ハードウェアの要件

ご使用になる MCU が以下の機能をサポートしている必要があります。

- EPTPC
- ETHERC
- EDMAC

使用例は、「RX ファミリ EPTPC：PTP 同期時刻取得サンプルプログラム[3]」、「RX ファミリ EPTPC：PTP タイマ同期開始サンプルプログラム[4]」、「RX ファミリ EPTPC：同期パルス出力サンプルプログラム[5]」、「RX ファミリ イーサネットコントローラ：簡易スイッチ動作例 [6]」、「RX ファミリ イーサネットコントローラ：マルチキャストフレームフィルタ動作例[7]」に説明しています。

2.2 ハードウェアリソース要件

ドライバが必要とする周辺回路ハードウェアについて説明します。特に明記しない限り、周辺回路はドライバで制御します。ユーザアプリケーションから直接制御し、使用することはできません。

2.2.1 EPTPC チャネル

ドライバは EPTPC を使用します。この周辺モジュールは、PTP に準拠した時刻同期に加え、標準イーサの拡張期のである CH0 と CH1 間でのフレームの転送とマルチキャストフレームのフィルタリングに必要です。

2.2.2 ETHERC チャネル

ドライバはクロック（ノード）の種類に応じて、ETHERC（CH0）、ETHERC（CH1）、またはその両方を使用します。これらの周辺回路はイーサネット MAC 動作に必要です。

2.2.3 EDMAC チャネル

ドライバはクロック（ノード）の種類に応じて、EDMAC（CH0）、EDMAC（CH1）、またはその両方を使用します。これらの周辺回路は標準イーサネットフレームの送受信における CPU ホストインタフェースとして必要です。また、ドライバは PTP フレームの送受信に PTPEDMAC を使用します。

2.3 ソフトウェア要件

ドライバは以下のパッケージ（FIT モジュール）に依存しています。

- r_bsp
- r_ether_rx

2.4 制限事項

ドライバには以下の制限事項があります。

- TC として動作する場合、ドライバは TC 単独の動作となります。TC&OC の組み合わせで動作させる場合は、同期開始後、PTP メッセージの送受信設定を再度構成しなおしてください。
- BMC 処理には対応しません。

2.5 サポートされているツールチェイン

本 FIT モジュールは「4.6 動作確認環境」に示すツールチェーンで動作確認を行っています。

2.6　使用する割り込みベクタ

R_PTP_Init 関数と R_PTPIF_Open_ZC2 関数を実行すると EPTPC MINT 割り込みと PTPEDMAC PINT 割り込みがそれぞれ有効になります。

表 2.1 に本 FIT モジュールが使用する割り込みベクタを示します。

表 2.1　使用する割り込みベクター覧

デバイス	割り込みベクタ
RX64M RX71M	GROUPAL1 割り込み（ベクタ番号: 113） ● EPTPC MINT 割り込み（グループ割り込み要因番号：0） ● PTPEDMAC PINT 割り込み（グループ割り込み要因番号：1）

2.7　ヘッダファイル

API 呼び出しは、このドライバとともに提供されている r_ptp_rx_if.h または r_ptpif_rx_if.h のいずれか 1 つのファイルをインクルードすることで行われます。

2.8　整数型

このプロジェクトでは ANSI C99 を使用しています。整数型は stdint.h で定義されています。

2.9　コンパイル時の設定

ドライバの構成設定は *r_ptp_rx_config.h* で行います。オプション名と設定内容を以下の表に記載します。

構成設定	
#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	ドライバの PTP 時刻同期部の動作モード設定をします。有効にするチャネルと状態確認の方法を設定します。 • ビット 0 を 1 に設定すると、チャネル 0 が有効になります。 • ビット 1 を 1 に設定すると、チャネル 1 が有効になります。 ビット 0 とビット 1 の両方を 1 に設定した場合、チャネル 0 とチャネル 1 の両方が有効になります。 • ビット 4 を 1 に設定すると、状態の確認をソフトウェアのポーリングにより行います。このバージョンでは対応していません。 • ビット 5 を 1 に設定すると、状態の確認を EPTPC からの割り込みにより行います。このバージョンではこの値を設定してください。
#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	ドライバの PTP ホストインタフェース部の動作モード設定をします。有効にするチャネルと状態確認の方法を設定します。 • ビット 0 を 1 に設定すると、チャネル 0 が有効になります。 • ビット 1 を 1 に設定すると、チャネル 1 が有効になります。 ビット 0 とビット 1 の両方を 1 に設定した場合、チャネル 0 とチャネル 1 の両方が有効になります。

構成設定	
	• ビット 4 を 1 に設定すると、状態の確認をソフトウェアのポーリングにより行います。このバージョンでは対応していません。 ビット 5 を 1 に設定すると、状態の確認を EPTPC からの割り込みにより行います。このバージョンではこの値を設定してください。
#define PTPIF_CFG_NUM_RX_DESCRIPTORS - Default value = 4	受信ディスクリプタ数を設定します。 • 設定範囲は 1 から 8 です。
#define PTPIF_CFG_NUM_TX_DESCRIPTORS - Default value = 4	送信ディスクリプタ数を設定します。 • 設定範囲は 1 から 4 です。
#define PTPIF_CFG_BUFSIZE - Default value = 1536	PTPEDMAC で転送に使用するバッファサイズを設定します。 • 32 バイト単位の設定で、設定範囲は 64 から 1536 バイトです。
#define PTP_CFG_INTERRUPT_LEVEL - Default value = 2	EPTPC からの割り込み優先レベルを設定します。 • 設定範囲は 1 から 15 です。
#define PTPIF_CFG_INTERRUPT_LEVEL - Default value = 2	PTPEDMAC からの割り込み優先レベルを設定します。 • 設定範囲は 1 から 15 です。
#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	送信する PTP メッセージのフォーマット [1] を設定します。 • 0x00 を設定すると、Ethernet II フレームを使用します。 • 0x01 を設定すると、IEEE802.3 フレームを使用します。RX71M のみ対応です。 • 0x02 を設定すると、Ethernet II の UDP/IPv4 に準拠したパケットを使用します。 • 0x03 を設定すると、IEEE802.3 の UDP/IPv4 に準拠したパケットを使用します。RX71M のみ対応です。
#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	ドライバの同期モードを設定します。 • 0 を設定すると、傾き補正は行われません。 • 2 を設定すると、傾き補正を行い、ワースト 10 設定はハードウェアで実行します。推奨設定です。 • 3 を設定すると、傾き補正を行いますが、ワースト 10 設定はソフトウェアで設定する必要があります。
#define PTP_CFG_SYNC_TIMEOUT - Default value = 0x00000000	Sync メッセージ受信タイムアウト値を設定します。 ナノ秒単位で設定してください。 デフォルト値はタイムアウトを検出しない設定です。0 以外の値を設定した場合、タイムアウトが検出可能になります。
#define NUM_OF_TMR_CHANNEL - Default value = 6	タイマの全チャネル数を設定します。 • RX64M/71M では 6 を設定してください。
#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	ELC 経由の時刻同期イベントで開始する MTU3 チャネルを設定します。 • RX64M/71M では 0（デフォルト値）を設定してください。
#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	ELC 経由の時刻同期イベントで開始する GPT チャネルを設定します。 • RX64M/71M では 1（デフォルト値）を設定してください。
#define CURRENT_UTC_OFFSET - Default value = 0x0008	ローカルクロックカウンタの初期値を設定します。

構成設定	
#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	• currentUtcOffset を設定します。 デフォルト値は 8 秒です。 • PTP 時刻から NTP(Network Time Protocol)時刻へ変換するオフセットを設定します。 デフォルト値は 2,208,988,800 秒です。 • NTP 時刻を設定します。 デフォルト値は 3,668,572,800 秒(2016/03/31 08:00:00)です。 • PTP 時刻を設定します。 デフォルト値は 1,459,584,008 秒です。 • PTP_CFG_LCCLK_SEC_HI に秒の上位 16 ビットを設定します。 • PTP_CFG_LCCLK_SEC_LO に秒の下位 32 ビットを設定します。 • PTP_CFG_LCCLK_NANO にナノ秒を設定します。
#define PTP_CFG_TIMESTAMP_LATENCY - Default value = 0x03D4024E	ポート入出力時のタイムスタンプの遅延値を設定します。 デフォルト値は PHY インタフェースが MII、転送速度が 100Mbps、STCA 供給クロック周波数が 20MHz で、下記の値としています。 • 入力タイムスタンプ遅延値 デフォルト値は 980(0x3D4)ナノ秒です。 • 出力タイムスタンプ遅延値 デフォルト値は 590(0x24E)ナノ秒です。
#define PTP_CFG_LLC_CTL - Default value = 3	IEEE802.3 フォーマットの LLC-CTL フィールドを設定します。
#define PTP_CFG_PTP_VER_NUM - Default value = 0x02	PTP バージョンフィールドを設定します。 デフォルト値は IEEE1588-2008 (version 2)規格です。 • RX64M/71M では 0x02（デフォルト値）を設定してください。
#define PTP_CFG_DOMAIN_NUM - Default value = 0	ドメイン番号フィールドを設定します。 0 はデフォルトドメイン、1 から 3 は代替ドメイン、4 から 127 はユーザ定義、128 から 255 は予約。
#define PTP_CFG_ANNOUNCE_FLAG_FIELD - Default value = 0x00000000	Announce メッセージのフラグフィールドを設定します。 デフォルト値は全て”false”の設定です。 PTP profile Specific 2(b14) = false, PTP profile Specific 1(b13) = false, unicastFlag(b10) = false（RX64M/71M はユニキャスト PTP には対応しない）, 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	Sync メッセージのフラグフィールドを設定します。 デフォルト値は全て”false”の設定です。 PTP profile Specific 2(b14) = false, PTP profile Specific 1(b13) = false, unicastFlag(b10) = false（RX64M/71M はユニキャスト PTP には対応しない）,

構成設定	
	twoStepFlag(b9) = false（RX64M/71M は two step には対応しない） , alternateMasterFlag(b8) = false.
#define PTP_CFG_DELAY_REQ_FLAG_FIELD - Default value = 0x00000000	Delay_Req/Pdelay_Req メッセージのフラグフィールドを設定します。 デフォルト値は全”false”の設定です。 PTP profile Specific 2(b14) = false, PTP profile Specific 1(b13) = false, unicastFlag(b10) = false（RX64M/71M はユニキャスト PTP には対応しない）.
#define PTP_CFG_DELAY_RESP_FLAG_FIELD - Default value = 0x00000000	Delay_Resp/Pdelay_Resp メッセージのフラグフィールドを設定します。 デフォルト値は全”false”の設定です。 PTP profile Specific 2(b14) = false, PTP profile Specific 1(b13) = false, unicastFlag(b10) = false（RX64M/71M はユニキャスト PTP には対応しない）, twoStepFlag(b9) = false（RX64M/71M は two step には対応しない）.
#define PTP_CFG_CLK_ID #define CLK_ID_EUI48_BASE (0) #define CLK_ID_USR_DEFINE (1) - Default value = 0	PortIdentity の clockIdentity を設定します。 OUI フィールド（上位 3 バイト）と拡張フィールド（下位 5 バイト）を設定します 。 • 0 を設定すると、MAC アドレス（EUI-48）を基に clockIdentity を生成します。 • 1 を設定すると、ユーザ固有の値 [2] を設定します。
#define PTP_CFG_PORTID_PORT_NUM0 - Default value = 1 (Port0) #define PTP_CFG_PORTID_PORT_NUM1 - Default value = 2 (Port1)	PortIdentity のポート番号を設定します。 0x0000 と 0xFFFF は予約されています。
#define PTP_CFG_MTCID_U - Default value = 0x00000000 (high: 32 bit) #define PTP_CFG_MTCID_L - Default value = 0x00000000 (low: 32 bit)	マスタの clockIdentity を設定します。 parentDS.parentPortIdentity.clockIdentity フィールドに相当します。
#define PTP_CFG_MTPID - Default value = 0x0000	マスタのポート番号を設定します。 parentDS.parentPortIdentity.portNumber フィールドに相当します。
#define PTP_CFG_GM_PRIORITY10 - Default value = 0x00 (Port0) #define PTP_CFG_GM_PRIORITY11 - Default value = 0x00 (Port1)	grandmasterPriority1 の値を設定します。 parentDS.grandmasterPriority1 フィールドに相当します。 0 から 255 が設定可能で、値が小さいほど優先度は高くなります。
#define PTP_CFG_GM_PRIORITY20 - Default value = 0x00 (Port0) #define PTP_CFG_GM_PRIORITY21 - Default value = 0x00 (Port1)	grandmasterPriority2 の値を設定します。 parentDS.grandmasterPriority2 フィールドに相当します。 0 から 255 が設定可能で、値が小さいほど優先度は高くなります。
#define PTP_CFG_GM_CLK_QUALITY0 - Default value = 0xF821FFFF (Port0) #define PTP_CFG_GM_CLK_QUALITY1 - Default value = 0xF821FFFF (Port1)	grandmasterClockQuality の値を設定します。 parentDS.grandmasterClockQuality フィールドに相当します。 • b31 - b24: clockClass デフォルト値は 248(0xF8)で、スレーブのみのクロックは 255(0xFF)です。 • b23 - b16: clockAccuracy デフォルト値である 0x21 は 100 ナノ秒以下を示し、0x20 から 0x31 が設定可能です。

構成設定	
	• b15 - b0: offsetScaledLogVariance デフォルト値である 0xFFFF は、未計算であることを示します。
#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)	grandmasterIdentity の値を設定します。 parentDS.grandmasterIdentity フィールドに相当します。
#define PTP_CFG_CUR_UTC_OFFSET0 - Default value = 0x0008 (Port0) #define PTP_CFG_CUR_UTC_OFFSET - Default value = 0x0008 (Port1)	currentUtcOffset の値を設定します。 timePropertiesDS.currentUtcOffset フィールドに相当します。 デフォルト値はローカルクロックカウンタの初期値で定義した“CURRENT_UTC_OFFSET”の値です。
#define PTP_CFG_TIME_SOURCE0 - Default value = 0xA0 (Port0) #define PTP_CFG_TIME_SOURCE1 - Default value = 0xA0 (Port1)	timeSource の値を設定します。 timePropertiesDS.timeSource フィールドに相当します。 デフォルト値の timeSource は内蔵発振です。
#define PTP_CFG_STEPS_REMOVED0 - Default value = 0x0000 (Port0) #define PTP_CFG_STEPS_REMOVED1 - Default value = 0x0000 (Port1)	stepsRemoved の値を設定します。 currentDS.stepsRemoved フィールドに相当します。 デフォルト値は通過ノードはなしです。
#define PTP_CFG_PTP_EVENT_TOS0 - Default value = 0x00 (Port0) #define PTP_CFG_PTP_EVENT_TOS1 - Default value = 0x00 (Port1)	PTP event メッセージの TOS フィールドの値を設定します。 デフォルト値は best effort です。
#define PTP_CFG_PTP_GENERAL_TOS0 - Default value = 0x00 (Port0) #define PTP_CFG_PTP_GENERAL_TOS1 - Default value = 0x00 (Port1)	PTP general メッセージの TOS フィールドの値を設定します。 デフォルト値は best effort です。
#define PTP_CFG_PTP_PRIMARY_TTL0 - Default value = 0x80 (Port0) #define PTP_CFG_PTP_PRIMARY_TTL1 - Default value = 0x80 (Port1)	PTP primary メッセージの TTL フィールドの値を設定します。 デフォルト値は 128 です。
#define PTP_CFG_PTP_PDELAY_TTL0 - Default value = 0x01 (Port0) #define PTP_CFG_PTP_PDELAY_TTL1 - Default value = 0x01 (Port1)	PTP pdelay メッセージの TTL フィールドの値を設定します。 デフォルト値は 1 です。
#define PTP_CFG_LOG_ANNOUNCE_INTERVAL0 - Default value = 0x01 (Port0: 2sec interval) #define PTP_CFG_LOG_ANNOUNCE_INTERVAL1 - Default value = 0x01 (Port1: 2sec interval)	チャネル毎の Announce メッセージの送信間隔を設定します。 portDS.logAnnounceInterval フィールドに相当します。 送信間隔は秒単位で 2 を底とする対数で設定します。設定値が n のとき、送信間隔は 2^n 秒です。 0xF9（= -7）から 0x06（= 6）の値のみが設定可能で、この値は整数値として扱われます。
#define PTP_CFG_LOG_SYNC_INTERVAL0 - Default value = 0x00 (Port0: 1sec interval) #define PTP_CFG_LOG_SYNC_INTERVAL1 - Default value = 0x00 (Port1: 1sec interval)	チャネル毎の Sync メッセージの送信間隔を設定します。 portDS.logSyncInterval フィールドに相当します。 送信間隔は秒単位で 2 を底とする対数で設定します。設定値が n のとき、送信間隔は 2^n 秒です。 0xF9（= -7）から 0x06（= 6）の値のみが設定可能で、この値は整数値として扱われます。
#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)	チャネル毎の Delay_Req メッセージの平均送信間隔の最小値を設定します。 portDS.logMinDelayReqInterval フィールドに相当します。 平均送信間隔の最小値は秒単位で 2 を底とする対数で設定します。設定値が n のとき、送信間隔は 2^n 秒です。

構成設定	
	0xF9（= -7）から 0x06（= 6）の値のみが設定可能で、この値は整数値として扱われます。
#define PTP_CFG_LOG_MIN_PDELAY_REQ_INTERVAL 0 - Default value = 0x00 (Port0: 1sec interval) #define PTP_CFG_LOG_MIN_PDELAY_REQ_INTERVAL 1 - Default value = 0x00 (Port1: 1sec interval)	チャネル毎の Pdelay_Req メッセージの平均送信間隔の最小値を設定します。 portDS.logMinPdelayReqInterval フィールドに相当します。 平均送信間隔の最小値は秒単位で 2 を底とする対数で設定します。設定値が n のとき、送信間隔は 2^n 秒です。 0xF9（= -7）から 0x06（= 6）の値のみが設定可能で、この値は整数値として扱われます。

[1] RX64M グループでは Ethernet II フレームフォーマットのみがサポートされています。（IEEE802.3 フレームフォーマットはサポートされません。）

[2] 以下の定義を変更してください。
- PORTID_CLK_ID0_U: clockIdentity 上位（ポート 0）
- PORTID_CLK_ID0_L: clockIdentity 下位（ポート 0）
- PORTID_CLK_ID1_U: clockIdentity 上位（ポート 1）
- PORTID_CLK_ID1_L: clockIdentity 下位（ポート 1）

2.10 引数

ドライバの API 関数で使用しているデータ構造について説明します。これらの構造体は r_ptp_rx_if.h および r_ptpif_rx_if.h に API 関数のプロトタイプ宣言と共に定義しています。

2.10.1 定数

```
/* 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;
```

```
/* 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;
```

```
/* 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 データ型

(1) PTP データ型　構造体

```
/* 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) ドライバ制御　構造体

```
/* 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 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 戻り値

ドライバの API 関数の戻り値を示します。これらの戻り値は r_ptp_rx_if.h および r_ptpif_rx_if.h にプロトタイプ宣言と共に定義しています。

```
/* 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 コールバック関数

本モジュールでは、EPTPC MINT 割り込み発生時にコールバック関数である Eptpc_isr 関数、PTPEDMAC PINT 割り込み発生時にコールバック関数である PTPEDMAC PINT 関数を呼び出します。

2.13　コードサイズ

ROM (コードおよび定数) と RAM (グローバルデータ) のサイズは、ビルド時の「2.9 コンパイル時の設定」のコンフィギュレーションオプションによって決まります。

下表の値は下記条件で確認しています。

モジュールリビジョン: r_ptp_rx rev1.50

コンパイラバージョン: Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
(統合開発環境のデフォルト設定に"-lang = c99"オプションを追加)
GCC for Renesas RX 4.08.04.201801
(統合開発環境のデフォルト設定に" -std=gnu99"オプションを追加)
IAR C/C++ Compiler for Renesas RX version 4.12.1
(統合開発環境のデフォルト設定)

コンフィグレーションオプション: デフォルト設定

ROM、RAM およびスタックのコードサイズ

デバイス	分類	ファイル	使用メモリ		
			Renesas Compiler	GCC	IAR Compiler
RX64M	ROM	r_ptpif.c	**942 バイト**	**1750 バイト**	**1323 バイト**
		r_ptp.c	**5528 バイト**	**12060 バイト**	**9840** バイト
		Total	**6470 バイト**	**13810 バイト**	**11163** バイト
	RAM	r_ptpif.c	**3128 バイト**	**3128 バイト**	**3125** バイト
		r_ptp.c	**71 バイト**	**63 バイト**	**56** バイト
		Total	**3199 バイト**	**3191 バイト**	**3181** バイト
	STACK	r_ptpif.c	**64 バイト**	**-**	**48** バイト
		r_ptp.c	**80 バイト**	**-**	**52** バイト
		Total	**80 バイト**	**-**	**52** バイト
RX72M	ROM	r_ptpif.c	**962 バイト**	**1775 バイト**	**1399 バイト**
		r_ptp.c	**5481 バイト**	**12797 バイト**	**9998** バイト
		Total	**6443 バイト**	**14572 バイト**	**11397** バイト
	RAM	r_ptpif.c	**3128 バイト**	**3128 バイト**	**3125** バイト
		r_ptp.c	**71 バイト**	**63 バイト**	**56** バイト
		Total	**3199 バイト**	**3191 バイト**	**3181** バイト
	STACK	r_ptpif.c	**64 バイト**	**-**	**72** バイト
		r_ptp.c	**80 バイト**	**-**	**72** バイト
		Total	**80 バイト**	**-**	**72** バイト

2.14　FIT モジュールの追加方法

本モジュールは、使用するプロジェクトごとに追加する必要があります。ルネサスでは、Smart Configurator を使用した(1)、(3)の追加方法を推奨しています。ただし、Smart Configurator は、一部の RX デバイスのみサポートしています。サポートされていない RX デバイスについては(2)、(4)の方法を使用してください。

(1) e² studio 上で Smart Configurator を使用して FIT モジュールを追加する場合
e² studio の Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e² studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(2) e² studio 上で FIT Configurator を使用して FIT モジュールを追加する場合
e² studio の FIT Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加することができます。詳細は、アプリケーションノート「RX ファミリ e² studio に組み込む方法 Firmware Integration Technology (R01AN1723)」を参照してください。

(3) CS+上で Smart Configurator を使用して FIT モジュールを追加する場合
CS+上で、スタンドアロン版 Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e² studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(4) CS+上で FIT モジュールを追加する場合
CS+上で、手動でユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」を参照してください。

2.15　for 文、while 文、do while 文について

本モジュールでは、レジスタの反映待ち処理等で for 文、while 文、do while 文（ループ処理）を使用しています。これらループ処理には、「WAIT_LOOP」をキーワードとしたコメントを記述しています。そのため、ループ処理にユーザがフェイルセーフの処理を組み込む場合は、「WAIT_LOOP」で該当の処理を検索できます。

「WAIT_LOOP」を記述している対象デバイス

・RX64M グループ
・RX71M グループ
・RX72M グループ

3. API 関数

3.1 R_PTPIF_GetVersion ()

この関数は PTP ホストインタフェースドライバのバージョン番号を返します。

フォーマット

uint32_t R_PTPIF_GetVersion(void);

パラメータ

なし

戻り値

RX_PTPIF_VERSION_MAJOR（上位 16 ビット）：メジャーバージョン番号

RX_PTPIF_VERSION_MINOR（下位 16 ビット）：マイナーバージョン番号

プロパティ

r_ptpif_rx_if.h にプロトタイプ宣言しています。

説明

PTP ホストインタフェースドライバのメジャーバージョン番号とマイナーバージョン番号を返します。

今回のドライバのバージョン番号は「1.16」です。

- 上位 16 ビットはメジャーバージョン番号を示します。
  RX_PTPIF_VERSION_MAJOR: 値 = H'1.
- 下位 16 ビットはマイナーバージョン番号を示します。
  RX_PTPIF_VERSION_MINOR: 値 = H'16.

リエントラント

関数はリエントラントです。

使用例

この関数の使用例を以下に示します。

```
 #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);
```

注意

ドライバのバージョンによって、戻り値は異なります。

3.2 R_PTPIF_Reset ()

この関数は PTPEDMAC をリセットします。

フォーマット

void R_PTPIF_Reset(void);

パラメータ

なし

戻り値

なし

プロパティ

r_ptpif_rx_if.h にプロトタイプ宣言しています。

説明

PTPEDMAC[1] をリセットします。動作の概要を以下に説明します。

[1]ETHERC はリセットしません。

- PTPEDMAC をリセットします。
  EDMR レジスタの SWR ビットに 1 を設定
- リセット動作の完了を待ちます。
  PCLKA で 64 サイクル以上
  リセット完了待ちで、R_BSP_SoftwareDelay 関数のループ処理を使用しています。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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 ==== */
```

注意

この関数は通常、初期化時、またはエラーからの復帰時に実行します。

標準イーサネット MAC のチャネル 0 に最初に（チャネル 1 より以前）アクセスする場合、関連ドキュメント[2]「RX ファミリ イーサネットモジュール Firmware Integration Technology」（= r_ether_rx）の r_ether_rx_config.h で定義されている設定を、デフォルト設定から変更する必要があります。

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

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

3.3 R_PTPIF_Init ()

この関数は PTP ドライバのリソースを初期化します。

フォーマット

int32_t R_PTPIF_Init(void);

パラメータ

なし

戻り値

なし

プロパティ

r_ptpif_rx_if.h にプロトタイプ宣言しています。

説明

PTP ホストインタフェースと周辺モジュールを初期化します。動作の概要を以下に説明します。

- 送信ディスクリプタおよび受信ディスクリプタを初期化します。
- PTPEDMAC バッファを初期化します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例は「3.2 R_PTPIF_Reset ()」を参照ください。

注意

この関数は ETHERC と標準 EDMAC を開始後、通常は 1 回だけ実行します。

3.4 R_PTPIF_Open_ZC2 ()

この関数は PTP ホストインタフェースと周辺モジュールを初期化します。

フォーマット

ptpif_return_t R_PTPIF_Open_ZC2(void);

パラメータ

なし

戻り値

PTPIF_OK: 処理は正常に終了しました。

PTPIF_ERR: エラーが発生しました。

プロパティ

r_ptpif_rx_if.h にプロトタイプ宣言しています。

説明

PTP ホストインタフェースと周辺モジュールを初期化します。次の動作が実行されます。

- PTP ホストインタフェースの転送フラグを無効に初期化します。
- すべての PTPEDMAC ステータスフラグをクリアします。
- PTPEDMAC 割り込みを設定します。
  ptpif_dev_start()を呼び出します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例は「3.2 R_PTPIF_Reset ()」を参照ください。

注意

この関数は ETHERC と標準 EDMAC を開始後、通常は 1 回だけ実行します。

3.5 R_PTPIF_LinkProcess ()

この関数は PTP ホストインタフェースが PTP メッセージを転送するように設定します。

フォーマット

void R_PTPIF_LinkProcess(void);

パラメータ

なし

戻り値

なし

プロパティ

r_ptpif_rx_if.h にプロトタイプ宣言しています

説明

PTP ホストインタフェースが PTP メッセージを転送するよう設定します。次の動作が実行されます。

- 受信ディスクリプタおよび送信ディスクリプタを初期化します。
  _R_PTPIF_InitDescriptors()を呼び出します。
- PTPEDMAC を初期化します。
  _R_PTPIF_ConfigEthernet()を呼び出します。
- PTPEDMAC 受信動作を有効に設定します。
- PTP ホストインタフェース転送フラグを有効に設定します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例は「3.2 R_PTPIF_Reset ()」を参照ください。

注意

この関数はイーサネット MAC（ETHERC）の状態が変更されるたびに 1 回だけ実行する必要があります。

3.6 R_PTPIF_CheckLink_ZC ()

この関数は PTP ホストインタフェースの状態を確認します。

フォーマット

ptpif_return_t R_PTPIF_CheckLink_ZC(void);

パラメータ

なし

戻り値

PTPIF_OK: 転送有効

PTPIF_ERR: 転送無効

プロパティ

r_ptpif_rx_if.h にプロトタイプ宣言しています

説明

PTP ホストインタフェースの転送の有効または無効を確認します。

リエントラント

この関数はリエントラントです。

使用例

この関数の使用例は「3.2 R_PTPIF_Reset ()」を参照ください。

注意

なし

3.7 R_PTPIF_Close_ZC2 ()

この関数は PTP ホストインタフェースを無効にします。

フォーマット

ptpif_return_t R_PTPIF_Close_ZC2(void);

パラメータ

なし

戻り値

PTPIF_OK: 処理は正常に終了しました。

PTPIF_ERR: 既に無効となっていました。

プロパティ

r_ptpif_rx_if.h にプロトタイプ宣言しています

説明

PTP ホストインタフェースを無効にします。次の動作が実行されます。

- PTPEDMAC 割り込みをクリアします。
  ptpif_dev_start()を呼び出します。
- PTP ホストインタフェース転送フラグを無効に設定します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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;
```

注意

　この関数は終了動作を行うもので、PTP ホストインタフェースの転送が有効の場合にのみ終了動作を行います。

3.8 R_PTPIF_Read ()

この関数は PTP メッセージを受信します。

フォーマット

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

パラメータ

ch – 受信ポートチャネル番号（0 または 1）

buf – 読み込みバッファのポインタ。PTPEDMAC のバッファサイズ（=PTPIF_CFG_BUFSIZE）以上の領域の割り当てが必要です。

戻り値

1 以上の数値: 受信データの数。

PTPIF_ERR: エラーが発生しました。

プロパティ

r_ptpif_rx_if.h にプロトタイプ宣言しています

説明

PTP メッセージを受信します。次の動作が実行されます。

- PTP メッセージまたはその一部を受信し、受信データを格納しているバッファのポインタを設定します。
  R_PTPIF_Read_ZC2()を呼び出します。
- 受信バッファを開放します。
  R_PTPIF_Read_ZC2_BufRelease()を呼び出します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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;
```

注意

この関数では標準ライブラリ<string.h>が使用されています。

ユーザデータバッファは PTPEDMAC のバッファサイズ（=EMACP_BUFSIZE）以上の領域を割り当てる必要があります。サイズが不足している場合、PTP ドライバによってバッファ領域を超える書き込みがされます。

この関数は「3.10 R_PTPIF_Read_ZC2 ()」および「3.11 R_PTPIF_Read_ZC2_BufRelease ()」と同時に使用することはできません。

標準イーサネットと PTP 処理は、この関数の実行前にオープンしリンクする必要があります。

3.9 R_PTPIF_Write ()

この関数は PTP メッセージまたは標準イーサネットフレームを送信します。

フォーマット

ptpif_return_t R_PTPIF_Write(uint8_t* buf, uint32_t size);

パラメータ

buf – 書き込みバッファのポインタ

size – 書き込みデータのサイズ

戻り値

PTPIF_OK: 処理は正常に終了しました。

PTPIF_ERR_LEN: データ長が範囲外でした。

PTPIF_ERR: エラーが発生しました（PTPIF_ERR_LEN 以外のエラー）。

プロパティ

r_ptpif_rx_if.h にプロトタイプ宣言しています

説明

PTP メッセージまたは標準イーサネットフレームを送信します。次の動作が実行されます。

- 送信データのバッファポインタを設定し、そのサイズ分を割り当てます。
  R_PTPIF_Write_ZC2_GetBuf()を呼び出します。
- PTP メッセージまたはその一部を送信し、送信データのディスクリプタを設定します。
  R_PTPIF_Write_ZC2_SetBuf()を呼び出します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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;
```

注意

この関数は PTP メッセージだけではなく、標準のイーサネットフレームの送信にも使用できます。

この関数では標準ライブラリ<string.h>が使用されています。

標準イーサネットと PTP 処理は、この関数の実行前にオープンしリンクする必要があります。

この関数は「3.12 R_PTPIF_Write_ZC2_GetBuf ()」および「3.13 R_PTPIF_Write_ZC2_SetBuf ()」と同時に使用することはできません。

3.10 R_PTPIF_Read_ZC2 ()

この関数は PTP メッセージまたはその一部を受信します。

フォーマット

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

パラメータ

ch – 受信ポートチャネル番号（0 または 1）

buf – 受信データを格納するバッファ

戻り値

1 以上の数値: 受信データの数。

PTPIF_ERR_NO_DATA: データは受信されていませんでした。

PTPIF_ERR_NOT_TRAN: 転送が無効な状態でした。

プロパティ

r_ptpif_rx_if.h にプロトタイプ宣言しています

説明

PTP メッセージまたはその一部を受信し、受信データを格納しているバッファのポインタを設定します。次の動作が実行されます。

- 転送が有効な状態であることを確認します。
- 受信データの有無を確認します。
- 受信ポートチャネルを取得します。
- 受信データが格納されているバッファのポインタを取得します。
- 受信データ長を取得します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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 */
    }
```

注意

標準イーサネットと PTP 処理は、この関数の実行前にオープンしリンクする必要があります。

PTP メッセージの受信では、通常、この関数は「3.11 R_PTPIF_Read_ZC2_BufRelease ()」とともに使用します。

この関数を「3.8 R_PTPIF_Read ()」と同時に使用することはできません。

3.11 R_PTPIF_Read_ZC2_BufRelease ()

この関数は受信データバッファを開放します。

フォーマット

ptpif_return_t R_PTPIF_Read_ZC2_BufRelease(void);

パラメータ

なし

戻り値

PTPIF_OK: 処理は正常に終了しました。

PTPIF_ERR_NOT_TRAN: 転送が無効な状態でした。

プロパティ

r_ptpif_rx_if.h にプロトタイプ宣言しています

説明

受信データバッファを開放します。次の動作が実行されます。

- 転送が有効な状態であることを確認します。
- 受信データの有無を確認します。
- ディスクリプタの現在位置を次のディスクリプタに移動します。
- 受信動作が無効（EDRRR.RR = 0）の場合、PTPEDMAC を受信できるように設定します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例は「3.10 R_PTPIF_Read_ZC2 ()」を参照ください。

注意

標準イーサネットと PTP 処理は、この関数の実行前にオープンしリンクする必要があります。

PTP メッセージを受け取る際に、通常、この関数は「3.10 R_PTPIF_Read_ZC2 ()」とともに使用します。

この関数を「3.8 R_PTPIF_Read ()」と同時に使用することはできません。

3.12 R_PTPIF_Write_ZC2_GetBuf ()

この関数は送信データのためにバッファポインタを設定し、そのサイズのバッファを確保します。

フォーマット

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

パラメータ

buf – 送信データを書きこむためのバッファ

size – 割り当てられたバッファサイズ

戻り値

PTPIF_OK: 処理は正常に終了しました。

PTPIF_ERR_NOT_TRAN: 転送が無効な状態でした。

PTPIF_ERR_TACT: 送信バッファのための領域に空きがありません。

プロパティ

r_ptpif_rx_if.h にプロトタイプ宣言しています

説明

送信データのためにバッファポインタを設定し、そのサイズのバッファを確保します。次の動作が実行されます。

- 転送が有効な状態であることを確認します。
- 送信バッファの空きを確認します。
- 送信データのバッファポインタを設定し、そのサイズのバッファを確保します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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 */
    }
```

注意

この関数は PTP メッセージだけではなく、標準イーサネットフレームの送信にも使用できます。

標準イーサネットと PTP 処理は、この関数の実行前にオープンしリンクする必要があります。

PTP メッセージを送信する際には、通常、この関数は「3.13 R_PTPIF_Write_ZC2_SetBuf ()」関数とともに使用します。

この関数を「3.9 R_PTPIF_Write ()」と同時に使用することはできません。

3.13 R_PTPIF_Write_ZC2_SetBuf ()

この関数は PTP メッセージ、イーサネットフレーム、またはこの一部を送信します。

フォーマット

ptpif_return_t R_PTPIF_Write_ZC2_SetBuf(uint32_t len);

パラメータ

len – 送信データ長

戻り値

PTPIF_OK: 処理は正常に終了しました。

PTPIF_ERR_NOT_TRAN: 転送が無効な状態でした。

PTPIF_ERR_LEN: データ長が範囲外でした。

プロパティ

r_ptpif_rx_if.h にプロトタイプ宣言しています

説明

PTP メッセージ、イーサネットフレーム、またはこの一部を送信し、送信データのディスクリプタを設定します。次の動作が実行されます。

- 転送が有効な状態であることを確認します。
- 送信データのサイズを確認します。
- 送信データのディスクリプタを設定します。
- 送信動作が無効（EDTRR.TR = 0）の場合、PTPEDMAC を送信できるように設定します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例は「3.12 R_PTPIF_Write_ZC2_GetBuf ()」を参照ください。

注意

この関数は PTP メッセージだけではなく、標準イーサネットフレームの送信にも使用できます。

標準イーサネットと PTP 処理は、この関数の実行前にオープンしリンクする必要があります。

PTP メッセージの送信では、通常、この関数は「3.12 R_PTPIF_Write_ZC2_GetBuf ()」関数とともに使用されます。

この関数を「3.9 R_PTPIF_Write ()」と同時に使用することはできません。

3.14 R_PTPIF_RegMsgHndr ()

この関数は PTPEDMAC の割り込みハンドラにユーザ関数を登録します。

フォーマット

ptpif_return_t R_PTPIF_RegMsgHndr(uint32_t event, PTPMSG_HNDLR func);

パラメータ

event - PTPEDMAC 割り込みイベント（PTPEDMAC.EESR レジスタ）
PTPIF_FUNC_READ: フレーム受信フラグ(FR) (EMACP_FR_INT = 0x0004000)
PTPIF_FUNC_WRITE: フレーム転送完了フラグ (TC) (EMACP_TC_INT = 0x0020000)
PTPIF_FUNC_ERR: エラーフラグ (MACE, RFOF, RDE, TFUF, TDE, ADE, RFCOF)
func – 登録されるユーザ関数

戻り値

PTPIF_OK: 登録が完了しました。

PTPIF_ERR: 登録されませんでした。

プロパティ

r_ptpif_rx_if.h にプロトタイプ宣言しています

説明

PTPEDMAC の割り込みハンドラにユーザ関数を登録します。次の動作が実行されます。

- フレーム受信完了イベント（ビット 18）の場合、PTPIF メッセージ読み出しハンドラを登録します。
- フレーム送信完了イベント（ビット 21）の場合、PTPIF メッセージ書き込みハンドラを登録します。
- エラーイベント（ビット 8、ビット 16、ビット 17、ビット 19、ビット 20、ビット 23、ビット 24）の場合、PTPIF エラー通知ハンドラを登録します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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;
```

注意

この関数では、メッセージの読み出しと書き込み、及びエラー通知の割り込みハンドラを指定できます。

イベントに既に関数が登録されているときには、新たに指定された関数に置き換えられます。

PTPEDMAC 割り込みハンドラの型は次の通りです。

- PTP メッセージの読み出し:
  user_read_func(uint8_t ch, uint32_t type);
  第 1 引数は読み出しチャネル（0 または 1）で、第 2 引数は PTP メッセージの種類です。
  PTP メッセージの内容を読み出す処理を実行します。
- PTP メッセージの書き込み:
  user_write_func(uint32_t res, uint32_t sts);
  第 1 引数は予約で、0 を設定してください。第 2 引数は PTPEDMAC のステータスレジスタの内容です。
  送信時のエラーの確認を行います。
- PTPIF エラー通知:
  user_err_func(uint32_t res, uint32_t sts);
  第 1 引数は予約で、0 を設定してください。第 2 引数は PTPEDMAC のステータスレジスタの内容です。
  エラーの確認を行います。

3.15 R_PTP_GetVersion ()

この関数は PTP 時刻同期ドライバのバージョン番号を返します。

フォーマット

uint32_t R_PTP_GetVersion(void);

パラメータ

なし

戻り値

RX_PTP_VERSION_MAJOR（上位 16 ビット）：メジャーバージョン番号

RX_PTP_VERSION_MINOR（下位 16 ビット）：マイナーバージョン番号

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

PTP 時刻同期ドライバのメジャーバージョン番号とマイナーバージョン番号を返します。

今回のドライバのバージョン番号は「1.16」です。

- 上位 16 ビットはメジャーバージョン番号を示します。
  RX_PTP_VERSION_MAJOR: 値 = H'1.
- 下位 16 ビットはマイナーバージョン番号を示します。
  RX_PTP_VERSION_MINOR: 値 = H'16.

リエントラント

関数はリエントラントです。

使用例

この関数の使用例を以下に示します。

```
 #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);
```

注意

ドライバのバージョンによって、戻り値は異なります。

3.16 R_PTP_Reset ()

この関数は EPTPC をリセットします。

フォーマット

void R_PTP_Reset(void);

パラメータ

なし

戻り値

なし

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

EPTPC をリセットします。以下の動作を実行します。

- EPTPC をリセットします。
  PTRSTR レジスタの RESET ビットに 1 を書き込みます。
- リセット動作の完了を待ちます。
  PCLKA で 64 サイクル以上
  リセット完了待ちで、R_BSP_SoftwareDelay 関数のループ処理を使用しています。
- EPTPC へのリセットを解除します。
  PTRSTR レジスタの RESET ビットに 0 を書き込みます。
- リセット解除動作の完了を待ちます。
  PCLKA で 256 サイクル以上
  リセット完了待ちで、R_BSP_SoftwareDelay 関数のループ処理を使用しています。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例は「3.2 R_PTPIF_Reset ()」を参照ください。

注意

この関数は通常、初期化手順の最初、またはエラーからの復帰時に実行します。

3.17 R_PTP_SetTran ()

ポート間の転送モードを設定します。

フォーマット

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

パラメータ

mode – ポート間の転送モード

dir – 中継が行われる方向（ビットマップ形式）

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

ポート間の転送モードを設定します。次の動作が実行されます。

- 中継が行われる方向を設定します。
  CH0 から CH1 への転送の有効/無効の設定
  CH1 から CH0 への転送の有効/無効の設定
- 転送モードを設定します。
  ストア&フォワードモードまたはカットスルーモードを選択します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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 */
```

注意

この関数による設定は標準イーサネットフレームに対してのみ有効です。（PTP メッセージには有効ではありません。）

引数（mode または dir）が NULL ポインタのときには、その値は設定されません。

3.18 R_PTP_SetMCFilter ()

この関数はマルチキャスト（MC）フレームのフィルタ（FFLTR）の設定を行います。

フォーマット

ptp_return_t R_PTP_SetMCFilter(uint8_t ch, MCRecFil fil, uint32_t *fmac);

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

fil – マルチキャスト（MC）フレームのフィルタ設定

fmac – 受信フレームの MAC アドレス（FMAC0R(U/L)または FMAC1R(U/L)レジスタ）

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR: エラーが発生しました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

MC フィルタ（FFLTR）の設定を行います。以下の動作を実行します。

- 受信モードを設定します。
  全てのフレームを受信する、フレームを受信しない、または登録したフレームのみ受信のいずれかを設定します。
- 受信 MAC アドレスの更新
  引数 fmac を指定している場合、引数 fil に応じ、FMAC0R (U/L)レジスタまたは FMAC1R (U/L)レジスタを更新します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例は「3.17 R_PTP_SetTran ()」を参照ください。

注意

この関数による設定は標準イーサネットフレームにのみ有効です。PTP フレームには「3.35 R_PTP_SetSyncConfig ()」を使用ください。

マルチキャストフィルタ機能は拡張プロミスキャスモードでは無効です。マルチキャストフィルタ機能を使用する場合、「3.17 R_PTP_SetTran ()」の使用例に示すように拡張プロミスキャスモードをクリアしてください。

3 番目の引数（fmac）が NULL ポインタのときには、FMAC0R (U/L)レジスタと FMAC1R (U/L)レジスタはともに更新されません。

3.19 R_PTP_SetExtPromiscuous ()

拡張プロミスキャスモードを設定/解除します。

フォーマット

ptp_return_t R_PTP_SetExtPromiscuous(uint8_t ch, bool is_set);

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

is_set - (true): 設定、(false): クリア

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

拡張プロミスキャスモードの設定/クリアを行います。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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 */
```

注意

この関数による拡張プロミスキャスモード設定の結果に関しては、4.4 節を参照ください。

3.20　R_PTP_Init ()

この関数は EPTPC に対してデバイス設定に従った初期化を行います。

フォーマット

ptp_return_t R_PTP_Init(PTPConfig *tbl);

パラメータ

tbl – PTP 構成設定テーブル

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

PTP_ERR: エラーが発生しました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

EPTPC に対してデバイス設定に従った初期化を行います。次の動作が実行されます。

- RX72M の場合、バイパス機能を解除します。
- デバイスの動作パラメータを設定します。
- 同期フレーム処理部を選択します（SYNFP0 または SYNFP1）。
- タイマ割り込みハンドラへのポインタの初期化を行います。
- 同期フレーム処理部 SYNFP0 と SYNFP1 の初期化を行います。
  _R_PTP_Init_Param_SYNFP()を呼び出します。
- デバイス設定に従い PTP 受信フィルタの設定を行います。
- 同期フレーム処理部（SYNFP0 と SYNFP1）の設定値を有効にします。
- パケット中継部（PRC-TC）と統計的クロック補正部（STCA）の初期化を行います。
  _R_PTP_Init_PRC_STCA()を呼び出します。
- EPTPC の割り込みフラグと割り込みマスクをクリアします。
- EPTPC 割り込みの設定を行います。
  ptp_dev_start()を呼び出します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例は「3.2 R_PTPIF_Reset ()」を参照ください。

注意

この関数は ETHERC と標準 EDMAC を起動後、1 度だけ実行する必要があります。

チャネル毎のパラメータ設定で、for 文（ループ処理）を使用しています。

3.21 R_PTP_SubConfig ()

この関数は EPTPC のオプション設定を行います。

フォーマット

ptp_return_t R_PTP_SubConfig(uint8_t ch, PTPSubConfig *tbl);

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

tbl – PTP オプション構成テーブル

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

EPTPC のオプション設定を行います。

ワースト 10 を取得する回数を設定します。（32 回以上の sync メッセージ受信が推奨されます。）

リエントラント

この関数はリエントラントです。

使用例

ワースト 10 を取得し、これらを傾き制限値に設定する場合の使用例を以下に示します。

```
#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 */
```

注意

このバージョンではワースト 10 を取得する回数以外の静的なオプション設定には対応していません。

3.22 R_PTP_RegMINTHndr ()

この関数は EPTPC の MINT 割り込みを設定し、MINT 割り込みハンドラにユーザ関数を登録します。

フォーマット

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

パラメータ

reg – MINT 割り込みレジスタ
MINT_FUNC_STCA: STCA からの割り込み
MINT_FUNC_PRC: PRC-TC からの割り込み
MINT_FUNC_SYN0: SYNFP0 からの割り込み
MINT_FUNC_SYN1: SYNFP1 からの割り込み
event – 割り込み要因
func – 登録されるユーザ関数

戻り値

なし

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

EPTPC の MINT 割り込みを設定し、MINT 割り込みハンドラにユーザ関数を登録します。次の動作が実行されます。

- 割り込み要因が STCA の場合、STCA 割り込みで呼ばれる割り込みハンドラにユーザ関数を登録します。
- 割り込み要因が PRC-TC の場合、PRC-TC から呼ばれる割り込みハンドラにユーザ関数を登録します。
- 割り込み要因が SYNFP0/1 の場合、SYNFP0/1 から呼ばれる割り込みハンドラにユーザ関数を登録します。
- MINT 割り込みの設定または解除をします。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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;
```

注意

同じ要因の MINT 割り込みハンドラに関数を登録する場合、既に登録されているユーザ関数は更新されます。

引数（func）が NULL ポインタのときには、既に登録されているユーザ関数は解除され、同じ要因の MINT 割り込みは無効になります。

3.23 R_PTP_RegTmrHndr ()

この関数は EPTPC のタイマ割り込みハンドラにユーザ関数を登録します。

フォーマット

ptp_return_t R_PTP_RegTmrHndr(IntCycCh ch, TMR_HNDLR func);

パラメータ

ch – タイマ割り込みチャネル番号

func – 登録されるユーザ関数

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

EPTPC のタイマ割り込みハンドラにユーザ関数を登録します。次の動作が実行されます。

- 関数が登録されるハンドラのタイマ割り込みチャネルを設定します。
- 指定された割り込みハンドラにユーザ関数を登録します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。この例では PWM 出力タイマとして MTU3 を選択しています。

```
#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 */
}
```

注意

同じチャネルの割り込みハンドラに関数を登録する場合、既に登録されているユーザ関数は更新されます。

引数（func）が NULL ポインタのときには、既に登録されているユーザ関数は解除され、同じチャネルの割り込みは無効になります。

タイマ割り込みハンドラの設定で、for 文（ループ処理）を使用しています。

3.24 R_PTP_ELC_Ind ()

この関数は ELC 割り込み出力の設定/クリアを行います。

フォーマット

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

パラメータ

ch – タイマ割り込みチャネルの番号

edge – (0): 立ち上がりエッジ、(1): 立ち下がりエッジ

set – (0): クリア、(1): 設定

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR: エラーが発生しました。

プロパティ

プロトタイプは r_ptp_rx_if.h にプロトタイプ宣言しています

説明

ELC 割り込み出力の設定/クリアを行います。立ち上がりエッジと立ち下がりエッジでの割り込み出力を許可/禁止に設定できます。

リエントラント

関数はリエントラントです。

使用例

この関数の使用例は「3.23 R_PTP_RegTmrHndr ()」を参照ください。

注意

なし

3.25 R_PTP_ELC_SetClr ()

この関数は ELC 割り込みの自動クリアモードの設定/クリアを行います。

フォーマット

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

パラメータ

ch – タイマ割り込みチャネルの番号

edge – (0): 立ち上がりエッジ、(1): 立ち下がりエッジ

set – (0): クリア、(1): 設定

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR: エラーが発生しました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

ELC 割り込みの自動クリアモードの設定/クリアを行います。割り込みの自動クリアモードは立ち上がりエッジと立ち下がりエッジでの設定ができます。

リエントラント

関数はリエントラントではありません。

使用例

この関数の使用例は「3.23 R_PTP_RegTmrHndr ()」を参照ください。

注意

なし

3.26 R_PTP_Tmr_Set ()

この関数はタイマ割り込みハンドラ（タイマイベント）を有効にします。

フォーマット

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

パラメータ

ch – タイマ割り込みチャネル

start – タイマ開始時刻（ナノ秒）

cycle – パルス出力サイクル（ナノ秒）

pulse – パルス出力パルス幅（ナノ秒）

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR: エラーが発生しました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

タイマ割り込みハンドラ（タイマイベント）を有効にします。次の動作が実行されます。

- CPU へのタイマ割り込み出力を有効にします。
- 指定されたタイマチャネルにイベントタイム、パルス出力サイクル、出力パルス幅を設定します。
- タイマイベントを有効にします。

リエントラント

関数はリエントラントではありません。

使用例

この関数の使用例は「3.23 R_PTP_RegTmrHndr ()」を参照ください。

注意

タイマ開始時刻はローカルクロックの時刻以降の時刻を設定してください。

3.27 R_PTP_GetLcClk ()

この関数は現在のローカルクロックカウンタの値を取得します。

フォーマット

ptp_return_t R_PTP_GetLcClk(Timestamp *clk);

パラメータ

clk – ローカルクロックの値

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR: エラーが発生しました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

現在のローカルクロックカウンタ値を取得します。次の動作が実行されます。

- ローカルクロックカウンタ値を要求します。
- ローカルクロックカウンタ値がレジスタにロードされるのを待ちます。
  _R_PTP_Wait ()を呼び出します。
- ローカルクロックカウンタ値を格納します。
  格納される順序は、ナノ秒単位の 32 ビット、秒単位の下位 32 ビット、秒単位の上位 16 ビットとなります。

リエントラント

関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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);
```

注意

ローカルクロックカウンタ値の取得では、読み出動作による遅延が発生します。

3.28 R_PTP_SetLcClk ()

この関数はローカルクロックカウンタに値を設定します。

フォーマット

ptp_return_t R_PTP_SetLcClk(Timestamp *clk);

パラメータ

clk – ローカルクロックの値

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR: エラーが発生しました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

ローカルクロックカウンタに値を設定します。次の動作が実行されます。

- ローカルクロックカウンタ値を設定します。
- 設定した値をローカルクロックカウンタにロードします。

リエントラント

関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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);
```

注意

マスタの場合はマスタの時刻を設定してください。同期開始前に時刻基準を知ることができるシステムでは、スレーブの場合も時刻基準に従った時刻を初期値として設定してください。

3.29 R_PTP_ChkW10 ()

この関数はワースト 10 の取得を待ち、この値を傾き制限値として設定します。

フォーマット

void R_PTP_ChkW10(void);

パラメータ

なし

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_TOUT: タイムアウトエラーが発生しました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

ワースト 10 の取得を待ち、この値を傾き制限値として設定します。

_R_PTP_Wait_Ext ()を呼び出し、STSR レジスタの W10D ビットが 1 になるまで待ちます。

ワースト 10 の取得待ちの_R_PTP_Wait_Ext 関数内で、for 文（ループ処理）を使用しています。

リエントラント

関数はリエントラントではありません。

使用例

この関数の使用例は「3.21 R_PTP_SubConfig ()」を参照ください。

注意

この関数は傾き補正が適用され、ハードウェアによるワースト 10 設定が行われるとき（PTP_CFG_SYNC_MODE = PTP_SYNC_MODE2_HW）にのみ実行してください。

3.30 R_PTP_GetW10 ()

この関数は現在のワースト 10 を取得します。

フォーマット

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

パラメータ

p_w10 – プラス（正）側の傾きワースト 10

m_w10 – マイナス（負）側の傾きワースト 10

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_TOUT: タイムアウトエラーが発生しました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

現在の傾きワースト 10 を取得します。次の動作が実行されます。

- 現在の傾きワースト 10 の取得を要求します。
- 傾きワースト 10 の取得を待ちます。
  _R_PTP_Wait_Ext ()を呼び出します。
- 現在の傾きワースト 10 のロードを要求します。
- 傾きワースト 10 がロードされるのを待ちます。
  _R_PTP_InfoChk ()を呼び出します。
- 現在の傾きワースト 10 を格納します。
  PW10VR/MW10VR レジスタの上位、中位、下位フィールドの値を格納します。

リエントラント

関数はリエントラントではありません。

使用例

この関数の使用例は「3.21 R_PTP_SubConfig ()」を参照ください。

注意

引数（p_w10 もしくは n_w10）が NULL ポインタの場合には、値は返しません。

この関数は傾き補正が適用され、ソフトウェアによるワースト 10 設定が行われるとき（PTP_CFG_SYNC_MODE = PTP_SYNC_MODE2_SW）にのみ使用してください。

ワースト 10 の取得待ちの_R_PTP_Wait_Ext 関数と_R_PTP_InfoChk 関数内で、それぞれ、for 文（ループ処理）と do while 文を使用しています。

3.31 R_PTP_SetGradLimit ()

この関数は傾き制限値を設定します。

フォーマット

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

パラメータ

p_lim – プラス（正）側の傾き制限値

m_lim – マイナス（負）側の傾き制限値

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR: 現バージョンでは未実装です。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

傾き制限値を設定します。

PLIMITR/MLIMITR レジスタの上位、中位、下位フィールドに値を格納します。

リエントラント

関数はリエントラントです。

使用例

この関数の使用例は「3.21 R_PTP_SubConfig ()」を参照ください。

注意

引数（p_lim もしくは n_lim）が NULL ポインタの場合には、値は設定されません。

この関数は傾き補正が適用され、ソフトウェアによるワースト 10 設定が行われるとき（PTP_CFG_SYNC_MODE = PTP_SYNC_MODE2_SW）にのみ実行されます。

3.32 R_PTP_GetMPortID ()

この関数はマスタの PortIdentify を取得します。

フォーマット

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

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

clk – マスタの clockIdentity フィールド

port – マスタのポート番号フィールド

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

マスタの PortIdentity を取得します。次の動作が実行されます。

- マスタの clockIdentity（上位フィールドおよび下位フィールド）を読み出します。
- マスタのポート番号フィールドを読み出します。

リエントラント

関数はリエントラントです。

使用例

この関数の使用例を以下に示します。

```
#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);
   }
```

注意

引数（clk または port）が NULL ポインタであれば、その値は返しません。

3.33 R_PTP_SetMPortID ()

この関数はマスタの PortIdentity を設定します。

フォーマット

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

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

clk – マスタの clockIdentity フィールド

port – マスタのポート番号フィールド

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

マスタの PortIdentity を設定します。次の動作が実行されます。

- マスタの clockIdentity（上位フィールドおよび下位フィールド）を設定します。
- マスタのポート番号フィールドを設定します。
- SYNFP0 もしくは SYNFP1 に対して、設定した値を有効にします。

リエントラント

関数はリエントラントではありません。

使用例

この関数の使用例は「3.32 R_PTP_GetMPortID ()」を参照ください。

注意

なし

3.34 R_PTP_GetSyncConfig ()

この関数は PTP 同期構成（SYRFL1R、SYRFL2R、SYTRENR および SYCONFR）の取得を行います。

フォーマット

ptp_return_t R_PTP_GetSyncConfig(uint8_t ch, uint32_t *fil1, uint32_t *fil2, uint32_t *tren, uint32_t *conf);

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

fil1 –SYRFL1R

fil2 –SYRFL2R

tren – SYTRENR.

conf –SYCONFR

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

PTP 同期の設定（SYRFL1R、SYRFL2R、SYTRENR および SYCONFR）を行います。次の動作が実行されます。

- SYNFP 受信フィルタレジスタ 1（SYRFL1R）の値を取得します。
  次の各メッセージの受信フィルタ設定（Announce、Sync、Follow_Up、Delay_Req、Delay_Resp、Pdelay_Req、Pdelay_Resp および Pdelay_Resp_Follow_Up）
- SYNFP 受信フィルタレジスタ 2（SYRFL2R）の値を取得します。
  Management、Signaling、およびイリーガルの各メッセージのフィルタ設定
- SYNFP 送信許可レジスタ（SYTRENR）の値を取得します。
  次の各メッセージの送信フィルタ設定（Announce、Sync、Delay_Req および Pdelay_Req）
- SYNFP 動作設定レジスタ（SYCONFR）の値を取得します。
  TC モード（E2E TC または P2P TC）および送信間隔の設定

リエントラント

関数はリエントラントです。

使用例

この関数の使用例を以下に示します。

```
#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);
```

注意

引数（fil1、fil2、tren または conf）が NULL ポインタのときには、その値は返しません。

3.35 R_PTP_SetSyncConfig ()

この関数は PTP 同期構成（SYRFL1R、SYRFL2R、SYTRENR および SYCONFR）の設定を行います。

フォーマット

ptp_return_t R_PTP_SetSyncConfig(uint8_t ch, uint32_t *fil1, uint32_t *fil2, uint32_t *tren, uint32_t *conf);

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

fil1 –SYRFL1R

fil2 –SYRFL2R

tren – SYTRENR.

conf –SYCONFR

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

PTP 同期の設定（SYRFL1R、SYRFL2R、SYTRENR および SYCONFR）を行います。次の動作が実行されます。

- SYNFP 受信フィルタレジスタ 1（SYRFL1R）の値を設定します。
  次の各メッセージのフィルタ設定（Announce、Sync、Follow_Up、Delay_Req、Delay_Resp、Pdelay_Req、Pdelay_Resp および Pdelay_Resp_Follow_Up）
- SYNFP 受信フィルタレジスタ 2（SYRFL2R）の値を設定します。
  Management、Signaling、およびイリーガルの各メッセージのフィルタ設定
- SYNFP 送信許可レジスタ（SYTRENR）の値を設定します。
  次の各メッセージの送信フィルタ設定（Announce、Sync、Delay_Req および Pdelay_Req）
- SYNFP 動作設定レジスタ（SYCONFR）の値を設定します。
  TC モード（E2E TC または P2P TC）および送信間隔の設定
- SYNFP0 もしくは SYNFP1 に対して、設定した値を有効にします。

リエントラント

関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
  #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 */
```

注意

BMC 処理の結果、クロックが変更になったときには、毎回、同期設定を行う必要があります。

引数（fil1、fil2、tren または conf）が NULL ポインタのときには、その値は設定されません。

3.36 R_PTP_GetSyncInfo ()

この関数は、現在の offsetFromMaster と meanPathDelay の値を取得します。

フォーマット

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

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

ofm – offsetFromMaster

mpd – meanPathDelay

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

現在の offsetFromMaster と meanPathDelay の値を取得します。次の動作が実行されます。

- offsetFromMaster の値を取得します。
  オーバフロー時には正の最大値（=0x7FFFFFFF、0xFFFFFFFF）に変更されます。
  アンダフロー時には最小値（=0x80000000、0x00000000）に変更されます。
  その他の場合は、scaledNanosecond 単位 [1] に変換されます。
- meanPathDelay の値を取得します。
  オーバフロー時には正の最大値（=0x7FFFFFFF、0xFFFFFFFF）に変更されます。
  アンダフロー時には最小値（=0x80000000、0x00000000）に変更されます。
  その他の場合は、scaledNanosecond 単位 [1] に変換されます。
- meanPathDelay の値を取得します。
  currentDS.offsetFromMaster と currentDS. meanPathDelay の値を更新します。

[1]IEEE1588 規格[1]で定義された 2^{16} 倍するとナノ秒になる単位。

リエントラント

この関数はリエントラントです。

使用例

この関数の使用例を以下に示します。

```
#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);
```

注意

引数（ofm または mpd）が NULL ポインタの場合には、その値は返しません。

3.37 R_PTP_UpdClkID ()

この関数は自身の clockIdentity の値を更新します。

フォーマット

ptp_return_t R_PTP_UpdClkID(uint8_t ch, int8_t *id);

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

id – clockIdentity

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

clockIdentity の値を更新します。次の動作が実行されます。

- 自身の clockIdentity（上位フィールドおよび下位フィールド、SYCIDRU/L）を設定します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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);
```

注意

デフォルト設定では EUI-48 形式の clockIdentity を生成します。独自の clockidentity を使用する場合、PTP 構成情報の設定が完了した後、この関数を実行してください。

3.38 R_PTP_UpdDomainNum ()

この関数は PTP メッセージヘッダの domainNumber フィールドを更新します。

フォーマット

ptp_return_t R_PTP_UpdDomainNum(uint8_t ch, uint8_t dnum);

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

dnum – 更新する domainNumber の値

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

PTP メッセージヘッダの domainNumber フィールドを更新します。次の動作が実行されます。

- domainNumber フィールドの値を設定します（SYDOMR）。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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);
```

注意

なし

3.39 R_PTP_UpdAnceFlags ()

この関数は Announce メッセージのフラグフィールドを更新します。

フォーマット

ptp_return_t R_PTP_UpdAnceFlags(uint8_t ch, AnceFlag *flags);

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

flags – 更新するフラグの値

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

Announce メッセージのフラグフィールドを更新します。次の動作が実行されます。

- フラグフィールドの値を設定します（ANFR）。
  次のフラグのフィールド設定（PTP profile Specific 2/1, unicastFlag, alternateMasterFlag, frequencyTraceable, timeTraceable, ptpTimescale, currentUtcOffsetValid, leap59 および leap61）
- SYNFP0 もしくは SYNFP1 に対して、設定した値を有効にします。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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);
```

注意

なし

3.40 R_PTP_UpdAnceMsgs ()

この関数は Announce メッセージのメッセージフィールドを更新します。

フォーマット

ptp_return_t R_PTP_UpdAnceMsgs(uint8_t ch, AnceMsg *msgs);

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

msgs – 更新するフィールドの値

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

Announce メッセージのメッセージフィールドを更新します。次の動作が実行されます。

- grandmasterPriority2/1 フィールドの値を設定します (GMPR)。
- grandmasterClockQuality フィールドの値を設定します (GMCQR)。
- grandmasterIdentity フィールドの値を設定します (GMIDRU/L)。
- SYNFP0 もしくは SYNFP1 に対して、設定した値を有効にします。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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);
```

注意

なし

3.41 R_PTP_UpdSyncAnceInterval ()

この関数は Sync と Announce メッセージの logMessageInterval とタイムアウト値を更新します。

フォーマット

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

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

sync – Sync メッセージ送信間隔（$2^{interval}$ 秒）

ance – Announce メッセージ送信間隔（$2^{interval}$ 秒）

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

Sync と Announce メッセージの logMessageInterval とタイムアウト値を更新します。次の動作が実行されます。

- Sync と Announce 送信間隔の値を更新します。
  引数（sync または ance）が 2^{-7} より小さい場合、7.8125 ミリ秒に設定します。
  引数（sync または ance）が 2^{6} より大きい場合、64 秒に設定します。
  その他の場合、引数（sync）の値を Sync 送信間隔に、引数（ance）の値を Announce 送信間隔に、それぞれ設定します。
- SYNFP0 もしくは SYNFP1 に対して、設定した値を有効にします。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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);
```

注意

引数（sync または ance）が NULL ポインタの場合には、その値は更新されません。

3.42 R_PTP_UpdDelayMsgInterval ()

この関数は Delay メッセージの送信間隔、logMessageInterval およびタイムアウト値を更新します。

フォーマット

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

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

interval – マスタの場合、Delay_Resq の logMessageInterval フィールド（$2^{interval}$ 秒）

スレーブまたはリスニングの場合、Delay_Req/Pdelay_Req 送信間隔（$2^{interval}$ 秒）

tout – Delay_Resp/Pdelay_Resp 受信タイムアウト（tout * 1024 ナノ秒）

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

PTP ポート状態がマスタの場合、Delay_Resp メッセージの logMessageInterval フィールドの値を更新します。PTP ポート状態がスレーブまたはリスニングの場合、Delay_Req/Pdelay_Req メッセージの送信間隔と logMessageInterval フィールドの値 を受信した Delay_Resp メッセージの logMessageInterval フィールドの値に更新します。また、Delay_Resp/Pdelay_Resp 受信タイムアウト値を更新します。次の動作が実行されます。

- 受信した Delay_Resp メッセージの logMessageInterval フィールドの値を参照します。
- PTP ポート状態がマスタの場合、Delay_Resp メッセージの logMessageInterval フィールドの値 を更新します。
  引数（interval）が 2^{-7} より小さい場合、7.8125 ミリ秒に設定します。
  引数（interval）が 2^{6} より大きい場合、64 秒に設定します。
  その他の場合、引数（interval）の値を設定します。
- PTP ポート状態がスレーブまたはリスニングの場合、Delay_Req/Pdelay_Req 送信間隔の値を、Delay_Resp メッセージの logMessageInterval フィールドの値に更新します。
  2^{-7} より小さい場合、7.8125 ミリ秒に設定します。
  2^{6} より大きい場合、64 秒に設定します。
  その他の場合、Delay_Resp メッセージの logMessageInterval フィールドの値を Delay_Req/Pdelay_Req の送信間隔に設定します。
- Delay_Resp/Pdelay_Resp 受信タイムアウトの値を更新します。
- SYNFP0 もしくは SYNFP1 に対して、設定した値を有効にします。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。（PTP ポート状態がスレーブの場合）

```
#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));
```

注意

引数（interval または tout）が NULL ポインタの場合には、その値は更新されません。

3.43　R_PTP_Start ()

この関数は同期動作を開始します。

フォーマット

ptp_return_t R_PTP_Start(void);

パラメータ

なし

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_TOUT: タイムアウトエラーが発生しました。

PTP_ERR: エラーが発生しました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

同期動作を開始します。次の動作が実行されます。

- クロックの種類に応じて、PTP メッセージ送信、もしくは送信なしに設定します。設定内容は表 3.1 のようになります。

表 3.1　PTP メッセージ送信テーブル

クロック	OCとBC				TC	
同期モード	P2P		E2E		P2P	E2E
マスタ/スレーブ	マスタ	スレーブ	マスタ	スレーブ		
Pdelay_Req	送信	送信	なし	なし	送信	なし
Delay_Req	なし	なし	なし	送信	なし	送信 または なし
Sync	送信	なし	送信	なし	なし	なし
Announce	送信	なし	送信	なし	なし	なし

- E2E スレーブの場合、offsetFromMaster 値の更新の有無を確認します。
  offsetFromMaster 値が更新されていないときには、その更新を待ちます。
  _R_PTP_Wait ()を呼び出しします。
- SYNFP0 もしくは SYNFP1 に対して、設定した値を有効にします。
- スレーブの場合、時刻同期を開始します（SYNSTARTR.STR を設定）。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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();
     }
  }
```

注意

この関数は PTP 構成情報の設定完了後に実行してください。

この関数は時刻同期が開始されていないときにのみ有効です。

offsetFromMaster の更新待ちの_R_PTP_Wait 関数内で、for 文（ループ処理）を使用しています。

3.44 R_PTP_Stop ()

この関数は同期動作を終了します。

フォーマット

ptp_return_t R_PTP_Stop(void);

パラメータ

なし

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR: エラーが発生しました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

同期動作を停止します。次の動作が実行されます。

- PTP メッセージ送信を停止します。
- SYNFP 受信フィルタレジスタ 1（SYRFL1R）を設定します。
  Announce メッセージのみを PTPEDMAC に転送し、他のメッセージはすべて破棄するように設定します。
- E2E スレーブの場合、offsetFromMaster 値が更新されるか否かを確認します。
  offsetFromMaster 値が更新されていないときには、その更新を待ちます。
  _R_PTP_Wait ()を呼び出します。
- SYNFP0 もしくは SYNFP1 に対して、設定した値を有効にします。
- スレーブの場合、時刻同期を停止します（SYNSTARTR.STR をクリア）。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例は「3.43 R_PTP_Start ()」を参照ください。

注意

この関数は時刻同期が開始されているときにのみ有効です。

時刻同期処理の停止処理で、_R_PTP_Wait 関数を呼び、その内部で for 文（ループ処理）を使用しています。

3.45 R_PTP_SetPortState ()

この関数は PTP ポート状態を設定します。

フォーマット

ptp_return_t R_PTP_SetPortState (uint8_t ch, PTPState state);

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

state – 更新するポート状態（ST_MASTER、ST_SLAVE、ST_LIST）

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

PTP ポート状態（ST_MASTER、ST_SLAVE、ST_LIST）を更新します。

リエントラント

この関数はリエントラントです。

使用例

この関数の使用例は「3.43 R_PTP_Start ()」を参照ください。

注意

なし

3.46　R_PTP_GetSyncCH ()

この関数は選択中の同期チャネルの取得を行います。

フォーマット

ptp_return_t R_PTP_GetSyncCH (void);

パラメータ

なし

戻り値

現在、選択中の同期チャネル。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

この関数は現在、選択中の同期チャネルの取得を行います。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例は「3.36 R_PTP_GetSyncInfo ()」を参照ください。

注意

なし

3.47 R_PTP_SetInterrupt ()

EPTPC の INFABT（制御情報異常検出フラグ）要因割り込みを有効にします。

フォーマット

ptp_return_t R_PTP_SetInterrupt (uint8_t ch);

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

EPTPC の INFABT（制御情報異常検出フラグ）要因割り込みを有効にします。以下の動作を実行します。

- SYNFP0/1 からの割り込みを EPTPC の MIEIPR レジスタの設定で有効にする。
- SYNFP0/1.SYSR の INFABT 要因割り込みを EPTPC0/1 の SYIPR レジスタの設定で有効にする。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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 */
        }
    }
```

注意

なし

3.48 R_PTP_ChkInterrupt ()

EPTPC の INFABT 割り込み要因の確認を行います。

フォーマット

ptp_return_t R_PTP_ChkInterrupt (uint8_t ch, bool *is_det);

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

is_det – INFABT 割り込み要因発生フラグ

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

EPTPC の INFABT 割り込み要因の確認を行います。

リエントラント

この関数はリエントラントです。

使用例

この関数の使用例は「3.47 R_PTP_SetInterrupt ()」を参照ください。

注意

なし

3.49 R_PTP_ClrInterrupt ()

EPTPC の INFABT 割り込み要因検出フラグのクリアを行います。

フォーマット

ptp_return_t R_PTP_ClrInterrupt (uint8_t ch);

パラメータ

ch – 同期フレーム処理部のチャネル番号（SYNFP0 または SYNFP1）

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

EPTPC の INFABT 割り込み要因検出フラグのクリアを行います。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例は「3.47 R_PTP_SetInterrupt ()」を参照ください。

注意

なし

3.50 R_PTP_DisableTmr ()

この関数はタイマ割り込みを無効にします。

フォーマット

void R_PTP_DisableTmr (void);

パラメータ

なし

戻り値

なし

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

EPTPC のタイマ割り込みを無効にします。次の動作が実行されます。

- MIESR レジスタの割り込み要因をクリアする。
- 有効であったチャネルのタイマ割り込みを無効にする。

リエントラント

関数はリエントラントではありません。

使用例

この関数の使用例は「3.23 R_PTP_RegTmrHndr ()」を参照ください。使用例では、この関数をタイマ割り込みハンドラに登録する方法を説明しています。最初のタイマイベントが発生した時に、この関数が以降のタイマ割り込みを無効にします。

注意

なし。

3.51 R_PTP_SetSyncDet ()

この関数は同期状態が変化した場合の検出条件を設定します。

フォーマット

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

パラメータ

dev – 同期外れ状態の検出条件

syn – 同期状態の検出条件

is_enb – アラーム通知の有効または無効：(1) 有効、(0) 無効

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

同期状態が変化した場合の検出条件を設定します。次の動作が実行されます。

- 変化した状態が連続何回続いた場合に検出するかの回数の範囲確認をします。
- 同期外れ状態の検出条件を設定します。
- 状態の検出条件を設定します。
- アラーム通知の有効または無効を設定します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#include "r_ptp_rx_if.h"

   SyncDet dev, syn;

   /* Register user interrupt handler for STCA 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) */
```

注意

なし。

3.52 R_PTP_SetSynctout ()

この関数は Sync メッセージ受信タイムアウトの検出条件を設定します。

フォーマット

ptp_return_t R_PTP_SetSynctout (uint32_t tout, bool is_enab);

パラメータ

tout – Sync メッセージ受信タイムアウト値（1024ns 単位）

is_enb – アラーム通知の有効または無効：(1) 有効、(0) 無効

戻り値

PTP_OK: 処理は正常に終了しました。

PTP_ERR_PARAM: パラメータに誤りがありました。

プロパティ

r_ptp_rx_if.h にプロトタイプ宣言しています

説明

Sync メッセージ受信タイムアウトの検出条件を設定します。次の動作が実行されます。

- タイムアウトの検出条件を設定します。
- アラーム通知の有効または無効を設定します。

リエントラント

この関数はリエントラントではありません。

使用例

この関数の使用例を以下に示します。

```
#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) */
```

注意

なし。

4.　付録

4.1　内部関数

PTP ドライバは動作時に次の内部関数を呼び出しています。表 4.1 はその一覧表です。

表 4.1　内部関数

関数	説明
_R_PTPIF_InitDescriptors()	PTPEDMAC のディスクリプタとバッファの初期化
_R_PTPIF_ConfigEthernet()	PTPEDMAC の初期設定
_R_PTP_Wait()	PTP 動作完了イベント待ち（タイムアウト 100 ミリ秒）
_R_PTP_Wait_Ext()	PTP 動作完了イベント待ち（タイムアウト 400 秒）
_R_PTP_InfoChk()	GETINFOR レジスタの統計情報保持完了待ち
_R_PTP_Int_STCA()	STCA の割り込みハンドラ
_R_PTP_Int_PRC()	PRC-TC の割り込みハンドラ
_R_PTPL_Int_Syn0()	SYNFP0 の割り込みハンドラと INFABT 割り込みフラグを設定
_R_PTPL_Int_Syn1()	SYNFP1 の割り込みハンドラと INFABT 割り込みフラグを設定
_R_PTPL_Init_SYNFP()	SYNFP のパラメータを初期値で設定
_R_PTP_Init_PRC_STCA()	PRC-TC と STCA のパラメータを初期値で設定

4.2　関連する Ether ドライバの API

PTP ドライバは Ether ドライバと組み合わせて使用してください。PTP ドライバが使用する Ether ドライバの API の情報に関しては、「RX ファミリ イーサネットモジュール Firmware Integration Technology」（関連ドキュメント[2]）を参照してください。これらの API の典型的な使い方は、実例として 3 章に記載しています。

4.3　標準イーサネットの拡張機能

PTP ドライバには PTP に基づいた時刻同期のほかに、簡易スイッチ機能[1]とマルチキャストフレームフィルタ機能に対応しています。「3.17 R_PTP_SetTran ()」は簡易スイッチを、「3.18 R_PTP_SetMCFilter ()」はマルチキャストフレームフィルタ機能の設定を行うものです。これら機能の概要を以下に説明します。

[1] これはハードウェアによるポート間の転送を意味します。

4.3.1　簡易スイッチ（PRC-TC に実装）

ストア&フォワードまたはカットスルー転送方式を選択し使用できます。特に産業用ネットワークで一般的なディジーチェーン接続にカットスルー転送方式を適用した場合、ポート間の転送における遅延を短縮でき、有効な機能となります。

また、ネットワークをチャネル毎に分離することで、独立した 2 個のネットワークとして使用することもできます。

4.3.2　マルチキャストフレームフィルタ（SYNFP0 および SYNFP1 に実装）

不要なマルチキャストフレームの受信を省略することで、システムとしての性能を向上させることができます。また、フィルタを有効にした場合でも、特定の 2 種類のフレームを受信することができます。

PTP フレームには、固有のフィルタ（SYRFL1R/2R）を別に実装しています。このフィルタの設定に関しては「3.35 R_PTP_SetSyncConfig ()」を参照ください。

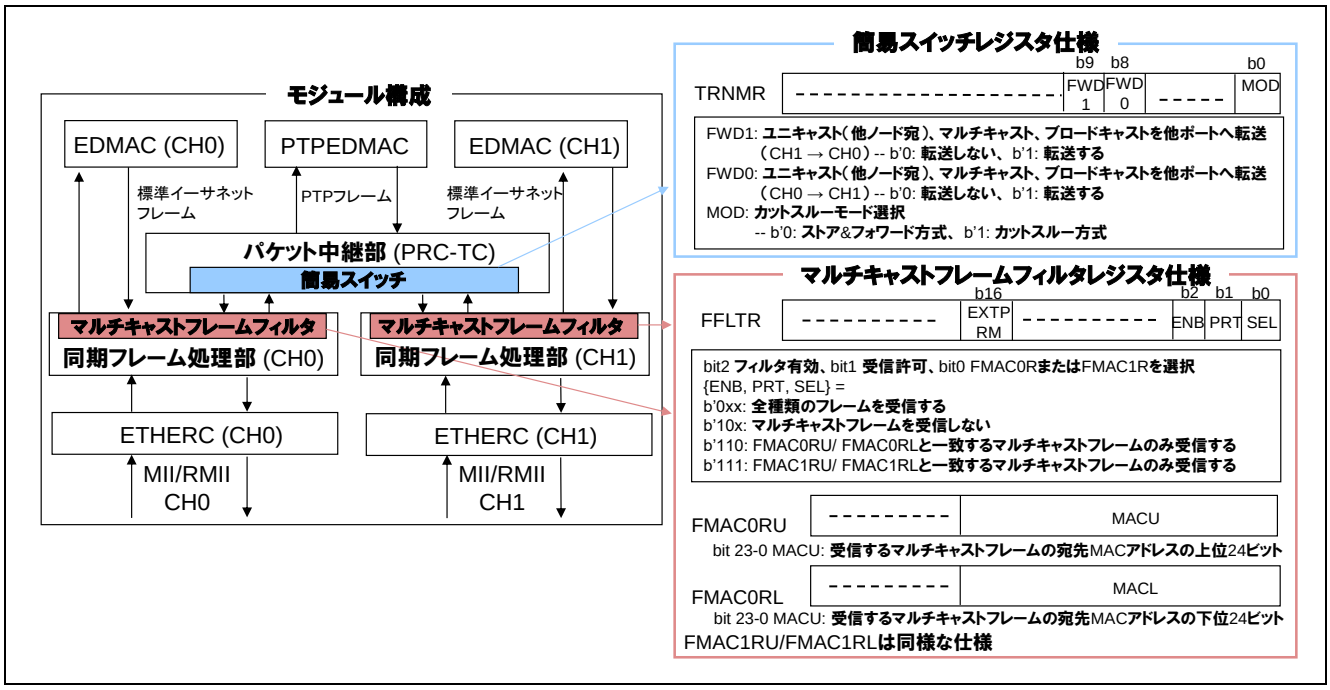
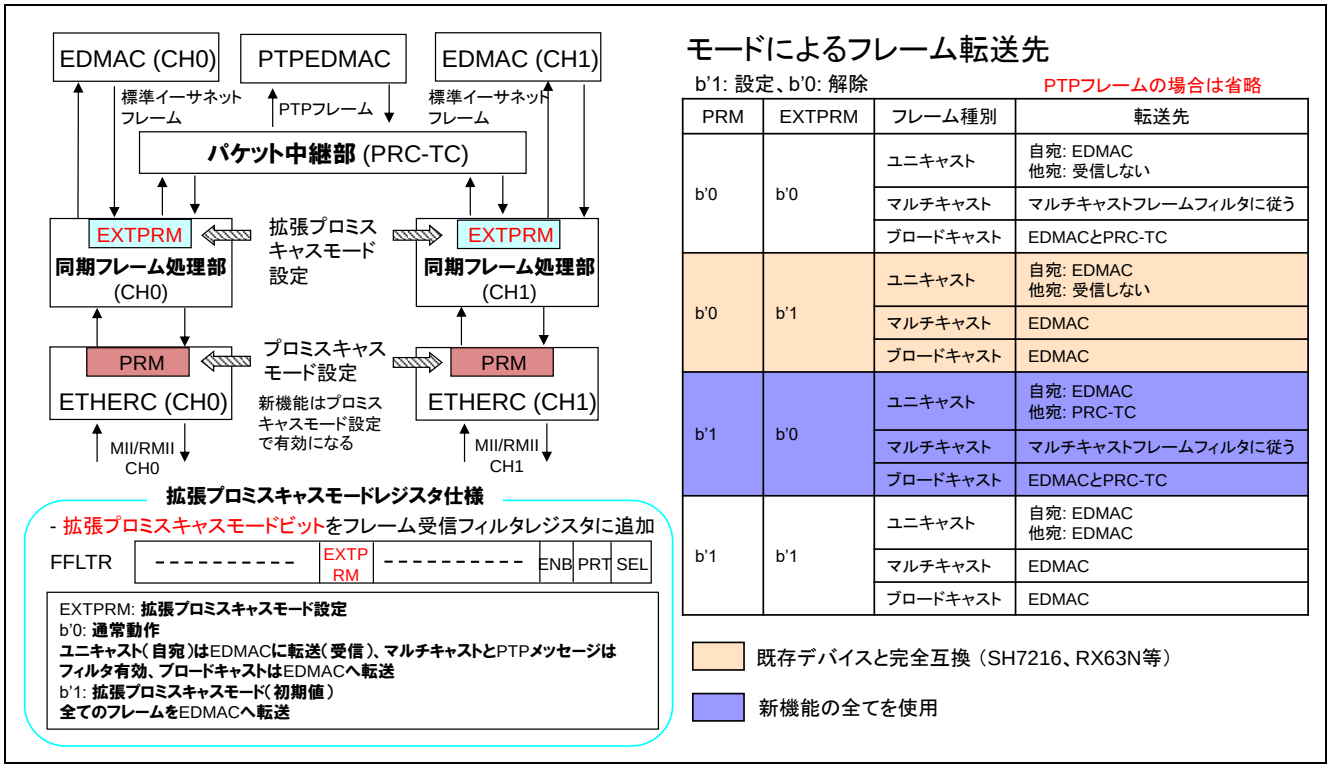


図 4.1 標準イーサネットの拡張機能

4.4　既存のデバイスとの互換性

RX64M/71M のイーサネットモジュールは SH7216 や RX63N 等の既存のルネサス製品と互換性があります。互換性はプロミスキャスモード（PRM）と拡張プロミスキャスモード（EXTPRM）の組み合わせで対応しています。プロミスキャスモードは R_ETHER_Control 関数のコントロールコード CONTROL_SET_PROMISCUOUS_MODE で設定することができます。詳細は「3.19 R_PTP_SetExtPromiscuous ()」を参照してください。これらモードの設定に関しては、「RX ファミリ イーサネットモジュール Firmware Integration Technology」（関連ドキュメント[2]）を参照ください。拡張プロミスキャスモードは「3.19 R_PTP_SetExtPromiscuous ()」で設定します。図 4.2 に設定の組み合わせを示します。



モードによるフレーム転送先

b'1: 設定、b'0: 解除　　PTPフレームの場合は省略

PRM	EXTPRM	フレーム種別	転送先
b'0	b'0	ユニキャスト	自宛: EDMAC 他宛: 受信しない
		マルチキャスト	マルチキャストフレームフィルタに従う
		ブロードキャスト	EDMACとPRC-TC
b'0	b'1	ユニキャスト	自宛: EDMAC 他宛: 受信しない
		マルチキャスト	EDMAC
		ブロードキャスト	EDMAC
b'1	b'0	ユニキャスト	自宛: EDMAC 他宛: PRC-TC
		マルチキャスト	マルチキャストフレームフィルタに従う
		ブロードキャスト	EDMACとPRC-TC
b'1	b'1	ユニキャスト	自宛: EDMAC 他宛: EDMAC
		マルチキャスト	EDMAC
		ブロードキャスト	EDMAC

既存デバイスと完全互換（SH7216、RX63N等）

新機能の全てを使用

図 4.2　プロミスキャスモードと拡張プロミスキャスモードの設定

4.5 セクション配置

表 4.2 に PTP ドライバのセクション配置例を示します。PTP ドライバと組み合わせて使用する Ether ドライバのセクション配置例も示します。

表 4.2 セクション配置例

アドレス	デバイス	セクション	説明
0x00000020	内蔵 RAM	SI	割り込みスタック領域
		SU	ユーザスタック領域
		B_1	1 バイト境界の未初期化データ領域
		R_1	1 バイト境界の初期化データ領域（変数）
		B_2	2 バイト境界の未初期化データ領域
		R_2	2 バイト境界の初期化データ領域（変数）
		B	4 バイト境界の未初期化データ領域
		R	4 バイト境界の初期化データ領域（変数）
0x00010000		B_ETHERNET_BUFFERS_1	標準イーサネットフレームの送信バッファおよび受信バッファ領域
		B_RX_DESC_1	標準イーサネットフレームの受信ディスクリプタ領域
		B_TX_DESC_1	標準イーサネットフレームの送信ディスクリプタ領域
0x00018000		B_PTPIF_BUFFER_1	PTP メッセージフレームの送信バッファおよび受信バッファ領域
		B_PTPIF_RX_DESC_1	PTP メッセージフレームの受信ディスクリプタ領域
		B_PTPIF_TX_DESC_1	PTP メッセージフレームの送信ディスクリプタ領域
0xFFF80000	内蔵 ROM	C_1	1 バイト境界の定数領域
		C_2	2 バイト境界の定数領域
		C	4 バイト境界の定数領域
		C$*	C$*セクション（C$DEC、C$BSEC、C$VECT）の定数領域
		D_1	1 バイト境界の初期化データ領域
		D_2	2 バイト境界の初期化データ領域
		D	4 バイト境界の初期化データ領域
		P	プログラム領域
		W_1	1 バイト境界の switch 文分岐テーブル領域
		W_2	2 バイト境界の switch 文分岐テーブル領域
		W	4 バイト境界の switch 文分岐テーブル領域
		L	文字列リテラル領域
0xFFFFFF80		EXCEPTVECT	割り込みベクタ領域
0xFFFFFFFC		RESETVECT	リセットベクタ領域

4.5.1 セクション配置の注意点

- 受信ディスクリプタ領域および送信ディスクリプタ領域は、EDMAC モードレジスタ（EDMR）の送受信ディスクリプタ長指定ビット（DL）を、16byte 設定にしているため、16byte 境界になるよう配置してください[1]。
- 送信バッファおよび受信バッファ領域は 32byte 境界になるよう配置してください[1]。

[1] PTP メッセージフレーム、標準イーサネットフレームともに配置が必要です。

4.6 動作確認環境

本 FIT モジュールの動作確認環境を以下に示します。

表 4.3 動作確認環境

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e2 studio V7.2.0 IAR Embedded Workbench for Renesas RX 4.12.1
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.01.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 4.08.04.201801 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.1 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.1.16
使用ボード	Renesas Starter Kit+ for RX64M Renesas Starter Kit+ for RX71M Renesas Starter Kit+ for RX72M

4.7 トラブルシューティング

(1) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「Could not open source file "platform.h"」エラーが発生します。

A：FIT モジュールがプロジェクトに正しく追加されていない可能性があります。プロジェクトへの追加方法をご確認ください。

- CS+を使用している場合
  アプリケーションノート RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」
- e2 studio を使用している場合
  アプリケーションノート RX ファミリ e2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)」

また、本 FIT モジュールを使用する場合、ボードサポートパッケージ FIT モジュール(BSP モジュール)もプロジェクトに追加する必要があります。BSP モジュールの追加方法は、アプリケーションノート「ボードサポートパッケージモジュール(R01AN1685)」を参照してください。

(2) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「This MCU is not supported by the current [r_ptp_rx] module.」エラーが発生します。

A：追加した FIT モジュールがユーザプロジェクトのターゲットデバイスに対応していない可能性があります。追加した FIT モジュールの対象デバイスを確認してください。

(3) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行するとコンフィグ設定が間違っているエラーが発生します。

A：“r_ptp_rx_config.h”ファイルの設定値が間違っている可能性があります。
“r_ptp_rx_config.h”ファイルを確認して正しい値を設定してください。詳細は「2.9 コンパイル時の設定」を参照してください。

5. 提供されているモジュール

提供されているモジュールは、ルネサス エレクトロニクスのウェブサイトからダウンロードすることができます。

6. 参考ドキュメント

ユーザーズマニュアル: ハードウェア
RX64M グループユーザーズマニュアル ハードウェア編 Rev.1.10（R01UH0377JJ）
RX71M グループユーザーズマニュアル ハードウェア編 Rev.1.10（R01UH0493JJ）
RX72M グループユーザーズマニュアル ハードウェア編 Rev.1.00（R01UH0804JJ）
最新版はルネサス エレクトロニクスのウェブサイトからダウンロードできます。

テクニカルアップデート／テクニカルニュース
（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：開発環境
RX ファミリ CC-RX コンパイラ ユーザーズマニュアル（R20UT3248）
（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデートの対応について

本モジュールは以下のテクニカルアップデートの内容を反映しています。

- TN-RX*-A125A/J

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2014.07.10	—	初版発行
1.01	2014.12.22	—	状態遷移関数とワースト 10 設定の追加
1.02	2014.12.31	—	RX71M 対応と BMC 機能の追加 モジュール名を修正（”_api”を削除）
1.10	2016.03.31	—	データ構造変更
1.11	2016.05.15	—	セクション情報の追加
1.12	2016.11.11	19, 41	バージョン取得関数の内部処理を修正
		51	ptp_tmr_isr 構造体の範囲外アクセスを修正
		82	統計情報保持完了待ち処理の修正し、その処理を内部関数として追加（_R_PTP_InfoChk 関数）
		83	モードによるフレーム転送先を修正（図 4.2 の記述）
1.13	2017.03.31	—	BC と P2P TC の動作設定を修正
		49, 82, 85	R_PTP_Init 関数、R_PTP_Start 関数、R_PTP_Stop 関数のリスニング状態の処理を変更
		52	ユーザ関数を MINT 割り込みハンドラに登録する関数を追加
		—	MINT 割り込みハンドラ処理を変更
		—	TC&OC の組み合わせ動作を TC 単独動作に変更
		73	自身の clockIdentity 設定関数を追加
		74	domainNumber フィールドの更新関数を追加
		75, 77	Announce メッセージのフィールド更新関数を追加
		79, 80	送信間隔設定関数の追加と変更
		82	P2P スレーブの場合、offsetFromMaster 更新待ち処理を追加
		86	PTP ポート状態の設定関数を追加
1.14	2017.04.30	61	R_PTP_SetLcClk 関数の注意書きを変更
		71	R_PTP_GetSyncInfo 関数のアンダフロー処理を修正
		79, 80	logMessageInterval の範囲外判定を修正
1.15	2019.07.31	—	GNUC と ICCRX に対応
		—	各ループ処理部分に「WAIT_LOOP」コメントを追加
		9	使用する割り込みベクタの説明を追加
		19	コールバック関数の説明を追加
		19	コードサイズの更新と GNUX と ICCRX の場合を追加
		20	FIT モジュールの追加方法を更新
		20	ループ処理のコメントに関する説明を追加
		22, 44	リセット完了待ちで R_BSP_SoftwareDelay 関数を使用するように変更
		37	R_PTPIF_Read_ZC2_BufRelease 関数の RACT ビット設定順序を修正
		44	R_PTP_Reset 関数にリセット解除待ち動作を追加
		64	ワースト 10 のロード待ち処理で_R_PTP_InfoChk 関数を呼ぶ記述に修正
		97	セクション配置を変更
		98	「動作確認環境」の節を追加
		98	「トラブルシューティング」の節を追加
1.16	2019.08.31	—	RX72M に対応
		—	バイパスモードの解除設定を追加
		17	同期状態検出用の構造体を追加

		17	同期外れと同期状態検出メンバを PTPSubConfig 構造体に追加
		95	同期状態変化の検出条件を設定する R_PTP_SetSyncDet 関数を追加
		97	Sync メッセージ受信タイムアウトの検出条件を設定する R_PTP_SetSynctout 関数を追加

製品ご使用上の注意事項

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

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

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

(Rev.4.0-1　2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。