RXファミリ

R01AN2025JJ0124
Rev.1.24
Dec 28, 2018

USB Basic Host and Peripheral Driver 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、USB Basic Firmware モジュールについて説明します。本モジュールは USB 通信の H/W 制御を行います。以降、本モジュールを USB-BASIC-FW FIT モジュールと称します。

対象デバイス

RX63N/RX631 グループ
RX65N/RX651 グループ
RX64M グループ
RX71M グループ

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

関連ドキュメント

1. Universal Serial Bus Revision 2.0 specification
   【http://www.usb.org/developers/docs/】
2. RX63N/RX631 グループユーザーズマニュアル ハードウェア編 (ドキュメント No.R01UH0041)
3. RX64M グループユーザーズマニュアル ハードウェア編 (ドキュメント No.R01UH0377)
4. RX71M グループユーザーズマニュアル ハードウェア編 (ドキュメント No.R01UH0493)
5. RX65N/RX651 グループユーザーズマニュアル ハードウェア編（ドキュメント No. R01UH0590)
6. RX65N/RX651-2M グループユーザーズマニュアル ハードウェア編（ドキュメント No. R01UH0659)

ルネサス エレクトロニクスホームページ
【http://japan.renesas.com/】
USB デバイスページ
【http://japan.renesas.com/prod/usb/】

目次

1. 概要

USB-BASIC-FW FIT モジュールは、USB の H/W 制御を行います。USB-BASIC-FW FIT モジュールは Renesas が提供する 1 種類のサンプルデバイスクラスドライバと組み合わせることで動作します。
以下に本モジュールがサポートしている機能を以下に示します。

＜全般＞
- USB Host または USB Peripheral をサポート
- デバイスの接続／切断、サスペンド／レジューム、USB バスリセット処理を行う
- パイプ 0 でコントロール転送を行う
- パイプ 1～9 でバルク転送、インタラプト転送を行う
- 本ドライバはリアルタイム OS(FreeRTOS)を使用する RTOS 版(以降、「RTOS」と記載)とリアルタイム OS を使用しない Non-OS 版(以降、「Non-OS」と記載)をサポートしています。

＜ホスト機能＞
- Hi-speed/Full-Speed/Low-speed ファンクションデバイスとエニュメレーションを行う
(動作スピードはデバイスにより異なります)
- 転送エラー判定および転送リトライを行う

＜ペリフェラル機能＞
- USB1.1 / 2.0 / 3.0 ホストとエニュメレーションを行う

1.1 注意事項

1. 本アプリケーションノートは、USB 通信動作を保証するものではありません。システムに適用される場合は、お客様における動作検証は十分に実施いただきますようお願いします。
2. 本書内に記載されている「USB0 モジュール」および「USB1 モジュール」という用語は MCU ごとに示すモジュールが異なりますので、以下を参照してください。

	MCU	USB モジュール名
USB0 モジュール (開始アドレス:0xA0000)	RX63N/RX631	USBa モジュール
	RX64M	USBb モジュール
	RX65N/RX651	USBb モジュール
	RX71M	USBb モジュール
USB1 モジュール (開始アドレス:0xA0200 / 0xD0400)	RX63N/RX631	USBa モジュール
	RX64M	USBA モジュール
	RX71M	USBAa モジュール

[Note]

RX65N/RX651 は USB1 モジュールをサポートしていません。

1.2 制限事項

本モジュールには以下の制限事項があります。

1. USB Host モード時、USB Hub および USB Hub ダウンポートに接続した USB デバイスに対するサスペンド/レジュームには対応しておりません。
2. USB Host モード時、データ転送中のサスペンドはサポートしておりません。データ転送が完了したことを確認の上、サスペンドを実行して下さい。
3. マルチコンフィグレーションはサポートしておりません。
4. USB Host モードと USB Peripheral モードの同時動作はサポートしていません。
5. RX63N/RX631 をご使用の場合、DMA/DTC を使用できません。

6. USB Hub 使用時の DMA/DTC 転送は、USB デバイスが未接続状態の USB Hub に対し、最初に接続される USB デバイスに対してのみ DMA/DTC 転送を使ったデータ転送が行われます。その他の状態では、CPU 転送を使ったデータ転送が行われます。
7. 本ドライバでは、ドライバ内でサポートする各関数の引数に対し仕様外の値が指定された場合のエラー処理は行っていません。
8. Vendor Class の場合、USB Hub を使用することはできません。
9. 本ドライバは、D0FIFO/D1FIFO レジスタを使った CPU 転送をサポートしていません。
10. RX63N/RX631 をご使用の場合、本ドライバは RTOS をサポートしていません。

1.3 用語一覧

APL	:	Application program
CDP	:	Charging Downstream Port
DCP	:	Dedicated Charging Port
HBC	:	Host Battery Charging control
HCD	:	Host Control Driver for USB-BASIC-FW
HDCD	:	Host Device Class Driver (Device Driver and USB Class Driver)
HUBCD	:	Hub Class Driver
H/W	:	Renesas USB device
MGR	:	Peripheral Device State Manager for HCD
Non-OS	:	USB Driver for OS less
PBC	:	Peripheral Battery Charging control
PCD	:	Peripheral Control Driver for USB-BASIC-FW
PDCD	:	Peripheral Device Class Driver (Device Driver and USB Class Driver)
RSK	:	Renesas Starter Kits
RTOS	:	USB Driver for the real-time OS
USB-BASIC-FW	:	USB Basic Host and Peripheral Driver
スケジューラ	:	Non-OSでタスク動作を簡易的にスケジューリングするもの
タスク	:	処理の単位

1.4 USB-BASIC-FW FIT モジュール

本モジュールは、r_bsp を使用したプロジェクトに組み込む必要があります。プロジェクトに組み込み後、API を使用することで USB の H/W 制御を行います。

1.5 ソフトウェア構成

1.5.1 モジュール構成

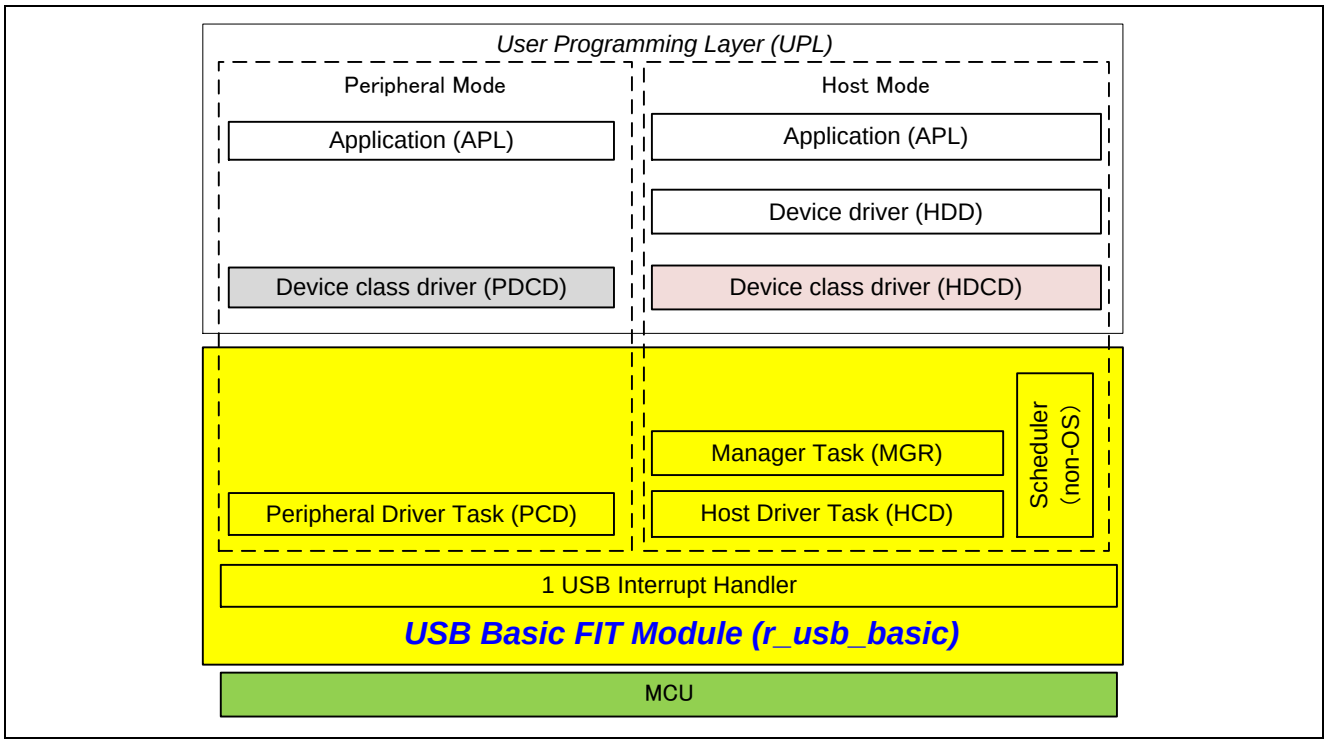

Figure 1-1 USB-BASIC-FWのモジュール構成図 (Non-OS)

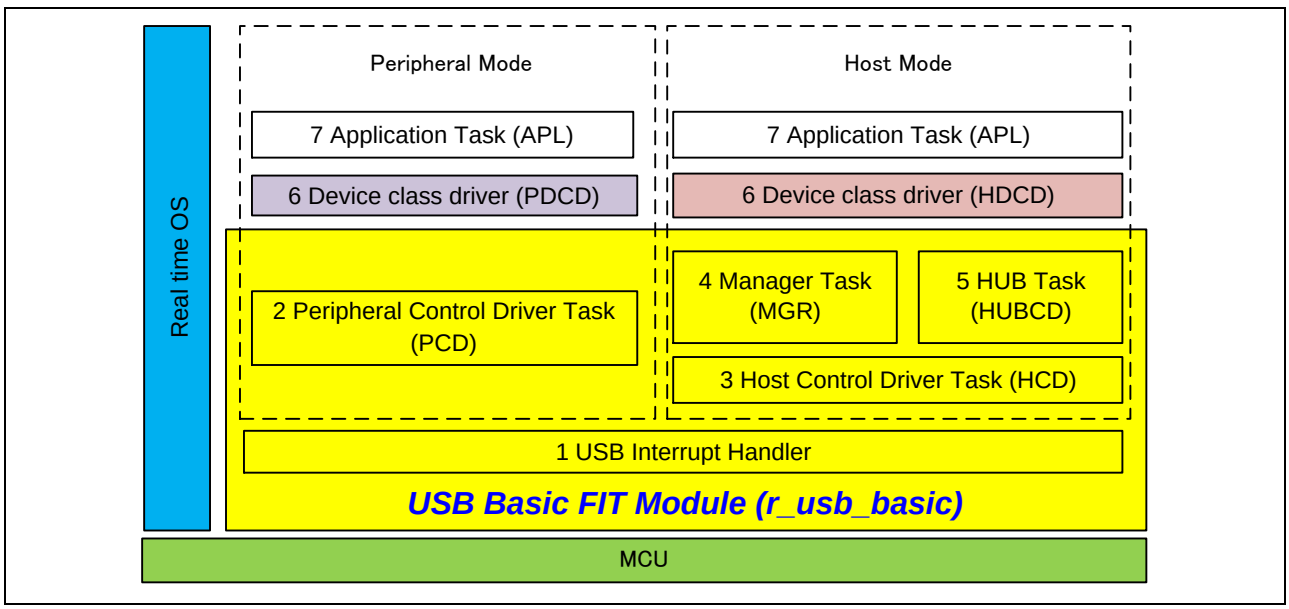

Figure 1-2 USB-BASIC-FWのモジュール構成図 (RTOS)

Table 1-1　モジュール機能概要

No	Module Name	Description
1	H/W Access Layer	H/W 制御
2	USB Interrupt Handler	USB 割り込みハンドラ
3	Peripheral Control Driver	ペリフェラルトランザクション管理
4	Host Control Driver	ホストトランザクション管理
5	Host Manager	デバイス状態管理 エニュメレーション処理 HCD/HUBCD 制御メッセージ判定
6	HUB Driver	HUB ダウンポートデバイス状態管理 HUB ダウンポートエニュメレーション
7	Device Class Driver	--
8	Device Driver	--
9	Application	システムにあわせてご用意ください

1.6　スケジューラ機能

本モジュール(Non-OS)は、スケジューラ機能を使用して各タスクや H/W の要求をタスクの優先順位にしたがって管理します。また優先順位が同じタスクに複数の要求が発生した場合は FIFO 構造で要求を実行します。タスク間の要求はメッセージの送受信で実現しています。

1.7　端子設定

USB FIT モジュールを使用するためには、マルチファンクションピンコントローラ（MPC）で周辺機能の入出力信号を端子に割り付ける（以下、端子設定と称す）必要があります。端子設定は、R_USB _Open 関数を呼び出す前に行ってください。

2. ペリフェラル

2.1 ペリフェラルコントロールドライバ（PCD）

2.1.1 基本機能

PCD は、H/W 制御用のプログラムです。PCD は PDCD から発行される要求を解析し、H/W の制御を行います。また、コールバック関数で制御結果を通知するとともに、H/W からの要求も解析し PDCD に通知します。PCD の機能を以下に示します。

1. Control 転送（ControlRead/ControlWrite/ No-data Control）
2. Data 転送（Bulk /Interrupt）および結果通知
3. データ転送の中断（全パイプ）
4. USB バスリセット信号検出およびリセットハンドシェイク結果通知
5. サスペンド/レジューム検出
6. VBUS 割り込みによるアタッチ/デタッチ検出
7. クロック停止（Low パワースリープモード）状態への H/W 制御および復帰

2.1.2 PCD に対する要求発行

PCD に対して H/W 制御要求を発行する場合およびデータ転送を行う場合は、API 関数を用います。
API 関数については、「**4. API**」の章を参照してください。

2.1.3 USB リクエスト

本ドライバは以下の標準リクエストをサポートしています。

1. GET_STATUS
2. GET_DESCRIPTOR
3. GET_CONFIGURATION
4. GET_INTERFACE
5. CLEAR_FEATURE
6. SET_FEATURE
7. SET_ADDRESS
8. SET_CONFIGURATION
9. SET_INTERFACE

本ドライバは上記以外のリクエストには STALL 応答します。
なお、本ドライバがデバイスクラスリクエスト又はベンダクラスリクエストを受信したときの処理方法については、「2.4 **クラスリクエスト**」を参照してください。

2.2 API 情報

本ドライバの API はルネサスの API の命名基準に従っています。

2.2.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- USB

2.2.2 ソフトウェアの要求

このドライバは以下のパッケージに依存しています。

- r_bsp
- r_dtc_rx (DTC 転送使用時)
- r_dmaca_rx (DMA 転送使用時)

2.2.3 動作確認環境

このドライバの動作確認環境を以下に示します。

Table 2-1 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 e^2 studio V.7.3.0
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 V.3.01.00
	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
リアルタイム OS	FreeRTOS V.10.0.0
エンディアン	リトルエンディアン / ビッグエンディアン
モジュールのリビジョン	Rev.1.24
使用ボード	Renesas Starter Kit for RX63N (Non-OS のみ) Renesas Starter Kit for RX64M Renesas Starter Kit for RX71M Renesas Starter Kit for RX65N, Renesas Starter Kit for RX65N-2MB
ホスト環境	下記の OS に接続し動作確認を行っています。 1. Windows® 7 2. Windows® 8.1 3. Windows® 10

2.2.4 使用する割り込みベクタ

このドライバが使用する割り込みベクタを以下に示します。

Table 2-2 使用する割り込みベクター一覧

デバイス	割り込みベクタ
RX63N RX631	USBI0 割り込み(ベクタ番号: 35) / USBR0 割り込み(ベクタ番号: 90) USB D0FIFO0 割り込み(ベクタ番号: 33) / USB D1FIFO0 割り込み(ベクタ番号: 34)

	USBI1 割り込み(ベクタ番号: 38) / USBR1 割り込み(ベクタ番号: 91) USB D0FIFO1 割り込み(ベクタ番号: 36) / USB D1FIFO1 割り込み(ベクタ番号: 37)
RX64M RX71M	USBI0(GROUPB)割り込み(ベクタ番号: 189, グループ割り込み要因番号 : 62) USB D0FIFO0 割り込み(ベクタ番号: 34) / USB D1FIFO0 割り込み(ベクタ番号: 35) USBR0 割り込み(ベクタ番号:90)
	USBAR 割り込み(ベクタ番号: 94) USB D0FIFO2 割り込み(ベクタ番号: 32) / USB D1FIFO2 割り込み(ベクタ番号: 33)
RX65N RX651	USBI0(GROUPB)割り込み(ベクタ番号: 185, グループ割り込み要因番号 : 62) USB D0FIFO0 割り込み(ベクタ番号: 34) / USB D1FIFO0 割り込み(ベクタ番号: 35) USBR0 割り込み(ベクタ番号:90)

2.2.5 タイマ

このドライバ(RTOS)は、RX MCU のタイマ(CMT)を使用しています。ユーザシステムにおいてタイマをご使用の場合は、このドライバが使用するタイマ以外のタイマをご使用いただきますようお願いします。

2.2.6 ヘッダファイル

すべての API 呼び出しとそれをサポートするインタフェース定義は r_usb_basic_if.h に記載されています。

2.2.7 整数型

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

2.2.8 コンパイル時の設定

コンパイル時の設定については、「8. **コンフィグレーション (r_usb_basic_config.h)**」の章を参照してください。

2.2.9 ROM / RAM サイズ

本ドライバの ROM/RAM サイズを以下に示します。

1. Non-OS

(1). RX64M, RX71M, RX65N/RX651

	引数チェック実施時	引数チェック非実施時
ROM サイズ	19.0K バイト (Note 3)	18.5K バイト (Note 4)
RAM サイズ	9.3K バイト	9.3K バイト

(2). RX63N/RX631

	引数チェック実施時	引数チェック非実施時
ROM サイズ	16.7K バイト (Note 3)	16.1K バイト (Note 4)
RAM サイズ	8.9K バイト	8.9K バイト

2. RTOS

	引数チェック実施時	引数チェック非実施時
ROM サイズ	34.4K バイト (Note 3)	33.9K バイト (Note 4)
RAM サイズ	18.9K バイト	18.9K バイト

[Note]

1. 上記のサイズには、BSP の ROM/RAM サイズが含まれています。
2. コンパイラの最適化オプションには、Default オプションが指定されています。
3. 「引数チェック実施時」の ROM サイズは、r_usb_basic_config.h ファイル内の USB_CFG_PARAM_CHECKING 定義に対し USB_CFG_ENABLE を指定した時の値です。
4. 「引数チェック非実施時」の ROM サイズは、r_usb_basic_config.h ファイル内の USB_CFG_PARAM_CHECKING 定義に対し USB_CFG_DISABLE を指定した時の値です。
5. RTOS には、リアルタイム OS のコードサイズが含まれています。

2.2.10 引数

API 関数の引数に使用される構造体については、「9. **構造体**」を参照してください。

2.2.11 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。

2.3 API 関数

API 関数の詳細については、「4. **API**」を参照してください。

2.4 クラスリクエスト

クラスリクエストを受信したときの処理方法については、「10. **クラスリクエスト**」を参照してください。

2.5 Descriptor

2.5.1 String Descriptor

この USB ドライバでは、String Descriptor については、各 String Descriptor を記述後、その String Descriptor を String Descriptor テーブルへ登録する必要があります。以下に String Descriptor の登録方法等を示します。

1. 各 String Descriptor を記述してください。各 String Descriptor の変数は、uint8_t*型で定義してください。

記述例)

```
uint8_t smp_str_descriptor0[] {
    0x04, /* Length */
    0x03, /* Descriptor type */
    0x09, 0x04 /* Language ID */
};
uint8_t smp_str_descriptor1[] =
{
    0x10, /* Length */
    0x03, /* Descriptor type */
    'R', 0x00,
    'E', 0x00,
    'N', 0x00,
    'E', 0x00,
    'S', 0x00,
    'A', 0x00,
    'S', 0x00
};
uint8_t smp_str_descriptor2[] =
{
    0x12, /* Length */
    0x03, /* Descriptor type */
    'C', 0x00,
    'D', 0x00,
    'C', 0x00,
    '_', 0x00,
    'D', 0x00,
    'E', 0x00,
    'M', 0x00,
    'O', 0x00
};
```

2. 上記で記述した各 String Descriptor の先頭アドレスを String Descriptor テーブルに設定してください。なお、String Descriptor テーブル用の変数は、uint8_t**型で定義してください。

[Note]

当該テーブル内での各 String Descriptor の設定箇所は、各 Descriptor 内に設定した Index 値 (iManufacurer, iConfiguration 等)によって決まります。

例えば、下記の場合、smp_str_descriptor1 には製造メーカーが記載されており、Device Descriptor 内の iManufacturer の値が"1"のため String Descriptor テーブル内の Index"1"の箇所に先頭アドレス "smp_str_descriptor1"を設定します。

```
/* String Descriptor テーブル */
uint8_t *smp_str_table[] =
{
    smp_str_descriptor0,    /* Index: 0 */
    smp_str_descriptor1,    /* Index: 1 */
    smp_str_descriptor2,    /* Index: 2 */
};
```

3. String Descriptor テーブルの先頭アドレスを usb_descriptor_t 構造体のメンバ pp_string に設定してください。usb_descriptor_t 構造体については、「**9.4. usb_descriptor_t構造体**」を参照してください。
4. usb_descriptor_t 構造体のメンバ num_string には、String Descriptor テーブルに設定した String Descriptor 数を設定してください。上記の例の場合、メンバ num_string には 3 を設定します。

2.5.2 その他の Descriptor

1. Device Descriptor、Configuration Descriptor および Qualifier Descriptor についてはドキュメント Universal Serial Bus Revision 2.0 specification(http://www.usb.org/developers/docs/)等をもとに作成してください。なお、各 Descriptor の変数は、uint8_t*型で定義してください。
2. 作成した各 Descriptor の先頭アドレスは、usb_descriptor_t 構造体の各メンバに登録してください。usb_descriptor_t 構造体については、「**9.4. usb_descriptor_t構造体**」を参照してください。

2.6 ペリフェラルバッテリチャージング制御 (PBC)

本ドライバは、PBC をサポートしています。

PBC は、対象デバイスを USB Battery Charging Specification Revision 1.2 で定義された Charging Port Detection(CPD)を動作させる際の H/W 制御用プログラムです。

CPD の結果は、R_USB_GetInformation 関数によって取得することができます。R_USB_GetInformation 関数については、「4.12 R_USB_GetInformation」を参照してください。

[Note]

RX63N/RX631 および RX65N/RX651 は、PBC をサポートしていません。

PBC の処理フローを以下に示します。

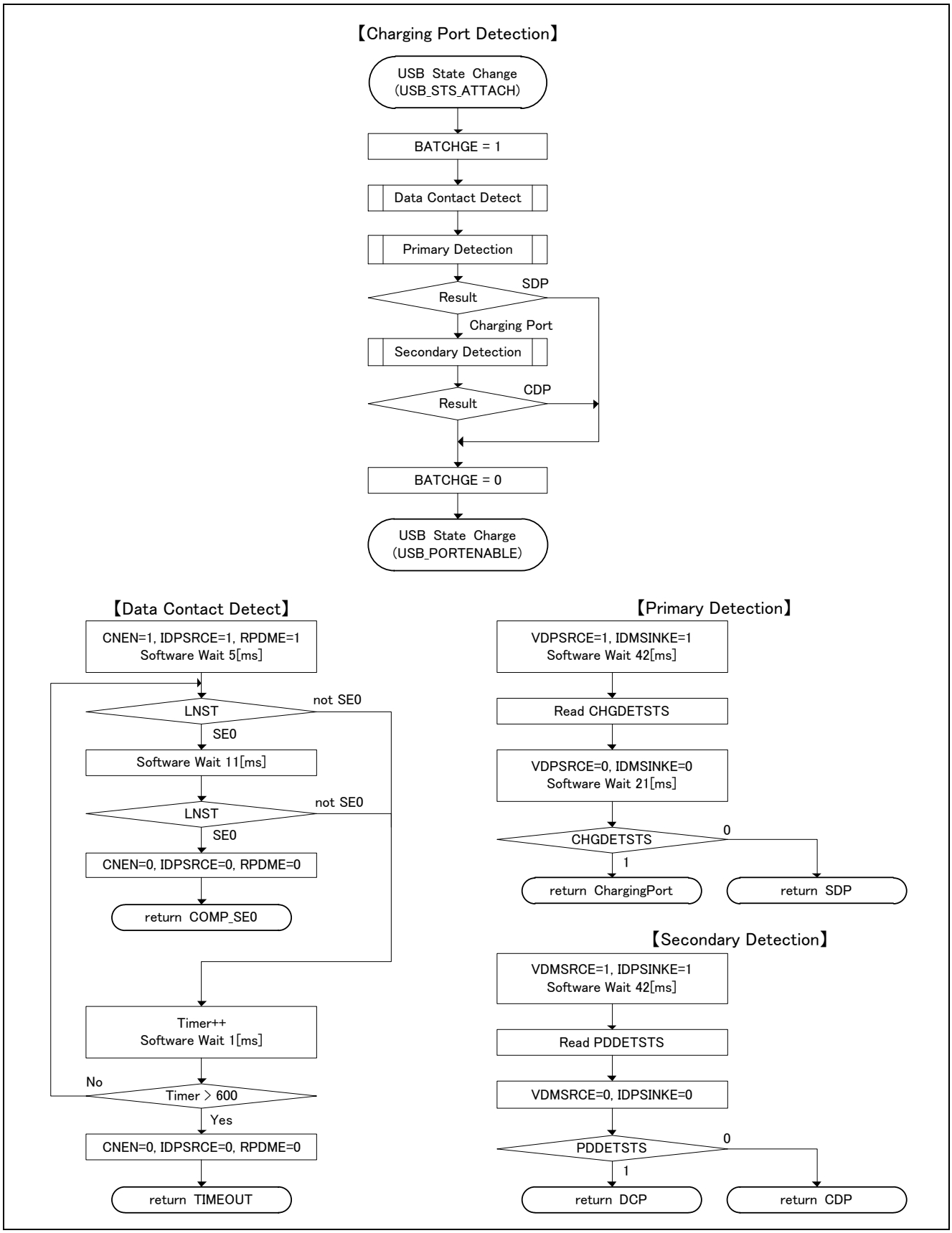

Figure 2-1　PBC フローチャート

3. ホスト

3.1 ホストコントロールドライバ（HCD）

3.1.1 基本機能

HCD は、H/W 制御用のプログラムです。HCD の機能を以下に示します。

1. Control 転送（ControlRead/ControlWrite/No-dataControl）および結果通知
2. Data 転送（Bulk /Interrupt）および結果通知
3. データ転送の中断（全パイプ）
4. USB 通信エラー判定および転送リトライ
5. USB バスリセット信号送出およびリセットハンドシェイク結果通知
6. サスペンド信号/レジューム信号送出
7. ATCH/DTCH 割り込みによるアタッチ/デタッチ検出

3.2 ホストマネージャ（MGR）

3.2.1 基本機能

MGR の機能を以下に示します。

1. HDCD の登録
2. 接続されたデバイスの状態管理
3. 接続されたデバイスのエニュメレーション
4. ディスクリプタからエンドポイント情報の検索

3.2.2 USB 標準リクエスト

MGR は、接続されたデバイスに対してエニュメレーションを行います。MGR が発行する USB 標準リクエストを以下に示します。

GET_DESCRIPTOR（Device Descriptor）
SET_ADDRESS
GET_DESCRIPTOR（Configuration Descriptor）
SET_CONFIGURATION

3.3 API 情報

本ドライバの API はルネサスの API の命名基準に従っています。

3.3.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- USB

3.3.2 ソフトウェアの要求

このドライバは以下のパッケージに依存しています。

- r_bsp
- r_dtc_rx (DTC 転送使用時)
- r_dmaca_rx (DMA 転送使用時)

3.3.3 動作確認環境

このドライバの動作確認環境を以下に示します。

Table 3-1 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 e^2 studio V.7.3.0
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 V.3.01.00
	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
リアルタイム OS	FreeRTOS V.10.0.0
エンディアン	リトルエンディアン / ビッグエンディアン
モジュールのリビジョン	Rev.1.24
使用ボード	Renesas Starter Kit for RX63N (Non-OS のみ) Renesas Starter Kit for RX64M Renesas Starter Kit for RX71M Renesas Starter Kit for RX65N, Renesas Starter Kit for RX65N-2MB

3.3.4 使用する割り込みベクタ

このドライバが使用する割り込みベクタを以下に示します。

Table 3-2 使用する割り込みベクター覧

デバイス	割り込みベクタ
RX63N RX631	USBI0 割り込み(ベクタ番号: 35) / USBR0 割り込み(ベクタ番号: 90) USB D0FIFO0 割り込み(ベクタ番号: 33) / USB D1FIFO0 割り込み(ベクタ番号: 34)
RX64M RX71M	USBI0(GROUPB)割り込み(ベクタ番号: 189, グループ割り込み要因番号：62) USB D0FIFO0 割り込み(ベクタ番号: 34) / USB D1FIFO0 割り込み(ベクタ番号: 35) USBR0 割り込み(ベクタ番号:90)
	USBAR 割り込み(ベクタ番号: 94) USB D0FIFO2 割り込み(ベクタ番号: 32) / USB D1FIFO2 割り込み(ベクタ番号: 33)

RX65N RX651	USBI0(GROUPB)割り込み(ベクタ番号: 185, グループ割り込み要因番号 : 62) USB D0FIFO0 割り込み(ベクタ番号: 34) / USB D1FIFO0 割り込み(ベクタ番号: 35) USBR0 割り込み(ベクタ番号:90)

3.3.5 タイマ

このドライバ(RTOS)は、RX MCU のタイマ(CMT)を使用しています。ユーザシステムにおいてタイマをご使用の場合は、このドライバが使用するタイマ以外のタイマをご使用いただきますようお願いします。

3.3.6 ヘッダファイル

すべての API 呼び出しとそれをサポートするインタフェース定義は r_usb_basic_if.h に記載しています。

3.3.7 整数型

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

3.3.8 コンパイル時の設定

コンパイル時の設定については、「8. **コンフィグレーション (r_usb_basic_config.h)**」の章を参照してください。

3.3.9 ROM / RAM サイズ

本ドライバの ROM/RAM サイズを以下に示します。

1. Non-OS

(1). RX64M, RX71M, RX65N/RX651

	引数チェック実施時	引数チェック非実施時
ROM サイズ	35.0K バイト (Note 3)	34.5K バイト (Note 4)
RAM サイズ	15.9K バイト	15.9K バイト

(2). RX63N/RX631

	引数チェック実施時	引数チェック非実施時
ROM サイズ	32.2K バイト (Note 3)	31.6K バイト (Note 4)
RAM サイズ	15.5K バイト	15.5K バイト

2. RTOS

	引数チェック実施時	引数チェック非実施時
ROM サイズ	46.7K バイト (Note 3)	46.2K バイト (Note 4)
RAM サイズ	33.7K バイト	33.7K バイト

[Note]

1. 上記のサイズには、BSP の ROM/RAM サイズが含まれています。
2. コンパイラの最適化オプションには、Default オプションが指定されています。

3. 「引数チェック実施時」の ROM サイズは、r_usb_basic_config.h ファイル内の USB_CFG_PARAM_CHECKING 定義に対し USB_CFG_ENABLE を指定した時の値です。
4. 「引数チェック非実施時」の ROM サイズは、r_usb_basic_config.h ファイル内の USB_CFG_PARAM_CHECKING 定義に対し USB_CFG_DISABLE を指定した時の値です。
5. RTOS には、リアルタイム OS のコードサイズが含まれています。

3.3.10 引数

API 関数の引数に使用される構造体については、「9. **構造体**」を参照してください。

3.3.11 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。

3.4 API 関数

API関数の詳細については、「4. **API**」を参照してください。

3.5 クラスリクエスト

クラスリクエストを受信したときの処理方法については、「10. **クラスリクエスト**」を参照してください。

3.6 ターゲットペリフェラルリスト(TPL)の設定方法

USB Host モードでは、TPL に Vendor ID(VID)と Product ID(PID)を登録すると、登録した VID と PID をもつ USB デバイスに対してのみ USB 通信を行います。

TPL への USB デバイスの登録は、コンフィグレーションファイル(r_usb_basic_config.h)内にあるTable 3-3の TPL 定義に対し、VID と PID をセットで指定してください。USB ドライバは、接続した USB デバイスの VID と PID が TPL に登録されているかどうかをチェックし、TPL に登録されていればその USB デバイスとの USB 通信を行います。TPL に登録されていなければその USB デバイスとの USB 通信は行いません。

なお、TPL に VID と PID を登録する必要がない場合、Table 3-3の TPL 定義に対し、USB_NOVENDOR と USB_NOPRODUCT を指定してください。USB_NOVENDOR と USB_NOPRODUCT が指定されている場合、USB ドライバは TPL の登録チェックを行いませんので、このチェックによる USB 通信不可の状態が発生することはありません。

Table 3-3 TPL 定義

TPL 定義名	内容
USB_CFG_TPLCNT	サポートする USB デバイスの数を指定してください。
USB_CFG_TPL	サポートする USB デバイスの VID と PID のセットを指定しください。 (必ず VID, PID の順番で記載してください。)
USB_CFG_HUB_TPLCNT	サポートする USB Hub の数を指定してください。
USB_CFG_HUB_TPL	サポートする USB Hub の VID と PID のセットを指定下さい。 (必ず VID, PID の順番で記載してください。)

== USB_TPL / USB_HUB_TPL への VID と PID の指定方法 ==

```
#define     USB_CFG_TPL       0x0011, 0x0022, 0x0033, 0x0044, 0x0055, 0x0066
                              VID  PID  VID  PID  VID  PID
                              USB デバイス 1  USB デバイス 2  USB デバイス 3

#define     USB_CFG_HUB_TPL   0x1111, 0x2222, 0x3333, 0x4444
                              VID  PID  VID  PID
                              USB Hub1  USB Hub2
```

記載例 1) 3 つの USB デバイスと 2 つの USB Hub を TPL に登録する場合

```
#define     USB_CFG_TPLCNT      3
#define     USB_CFG_TPL         0x0011, 0x0022, 0x0033, 0x0044, 0x0055, 0x0066
#define     USB_CFG_HUB_TPLCNT  2
#define     USB_CFG_HUB_TPL     0x1111, 0x2222, 0x3333, 0x4444
```

記載例 2) 3 つの USB デバイスを登録する場合 (USB Hub を使用しない)

```
#define     USB_CFG_TPLCNT      3
#define     USB_CFG_TPL         0x0011, 0x0022, 0x0033, 0x0044, 0x0055, 0x0066
#define     USB_CFG_HUB_TPLCNT  1
#define     USB_CFG_HUB_TPL     USB_NOVENDOR,USB_NOPRODUCT
```

記載例 3)　VID と PID を登録する必要が無い場合

```
#define     USB_CFG_TPLCNT      1
#define     USB_CFG_TPL         USB_NOVENDOR,USB_NOPRODUCT
#define     USB_CFG_HUB_TPLCNT  1
#define     USB_CFG_HUB_TPL     USB_NOVENDOR,USB_NOPRODUCT
```

[Note]

1. Table 3-3の TPL 定義に対し、USB_NOVENDOR と USB_NOPRODUCT を設定した場合でも USB_CFG_TPLCNT および USB_CFG_HUB_TPLCNT には、"1"を指定してください。
2. コンフィグレーションファイル(r_usb_basic_config.h)については、「8. **コンフィグレーション (r_usb_basic_config.h)**」の章を参照してください。

3.7　デバイスアドレスの割り当てについて

USB Host モード時､USB ドライバは､接続された USB デバイスに対しデバイスアドレスを割り当てます。

1. USB Hub を使用する場合

   USB Hub に対しデバイスアドレス値 1 が割り当てられ､Hub 下に接続した USB デバイスに対してデバイスアドレス値 2 以降が割り当てられます。

2. USB Hub を使用しない場合

   USB デバイス対しデバイスアドレス値 1 が割り当てられます。

[Note]

デバイスアドレスは、USB モジュール単位で割り当てられます。例えば、RX64M 等の複数の USB モジュールをサポートする MCU の場合で、USB0 モジュールと USB1 モジュールにそれぞれ USB デバイスを接続したときは、各 USB デバイスには、デバイスアドレス値 1 が割り当てられます。

3.8　ホストバッテリチャージング制御 (HBC)

本ドライバは、HBC をサポートしています。

HBC は､対象デバイスを USB Battery Charging Specification Revision 1.2 で定義された CDP または DCP 機能を動作させる際の H/W 制御用プログラムです。

本ドライバの VBUS ドライブ、アタッチ処理、デタッチ処理のタイミングでそれぞれに応じた処理を行います。また、PDDETINT 割り込み発生時に処理を行います。上位層からの制御の必要はありません。

Change Port Detection(CPD)の結果は、R_USB_GetInformation 関数によって取得することができます。R_USB_GetInformation 関数については、「4.12 R_USB_GetInformation」を参照してください。

[Note]

RX63N/RX631 および RX65N/RX651 は、HBC をサポートしていません。

HBC の処理フローを以下に示します。

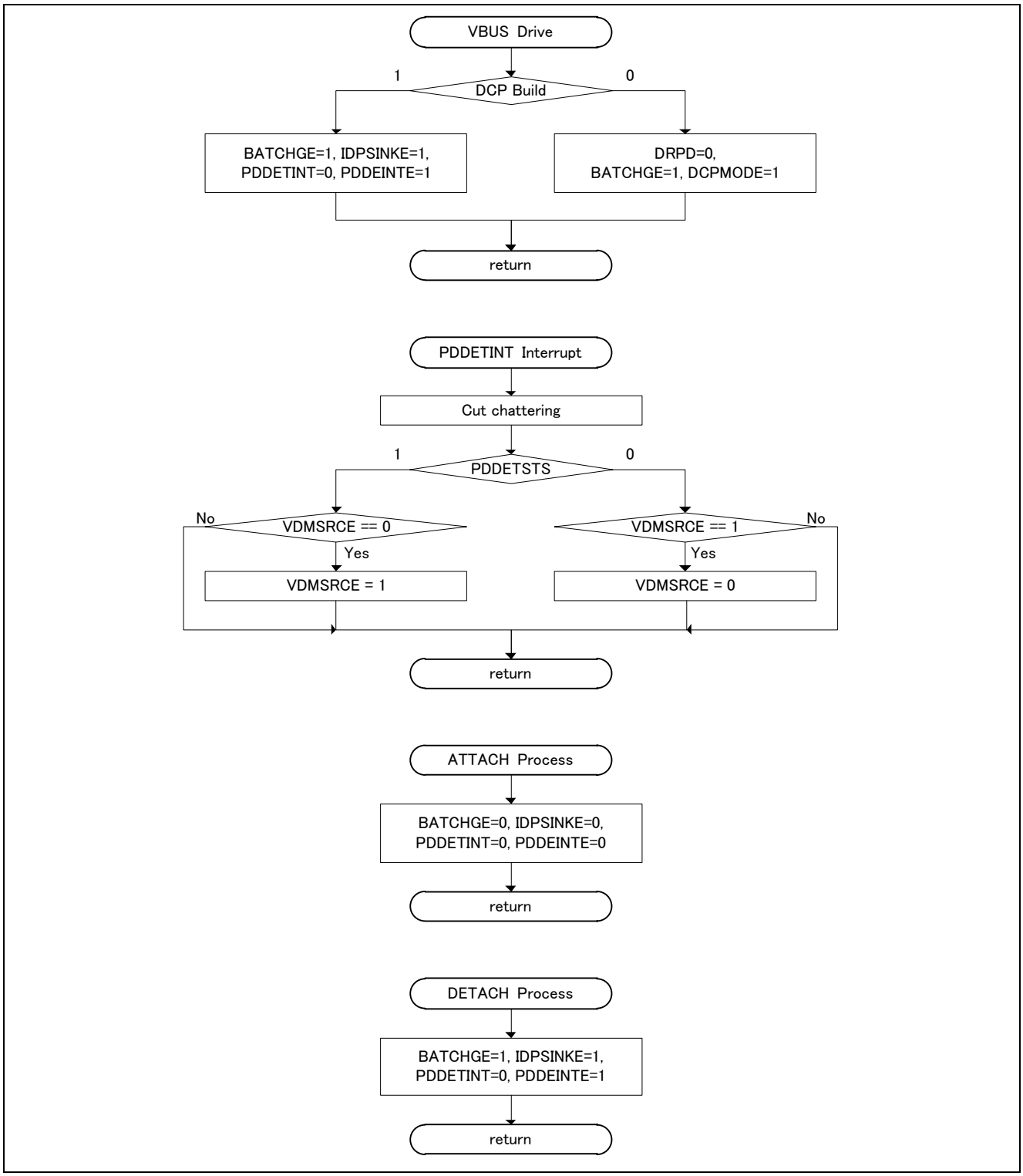

Figure 3-1 HBC フローチャート

4. API

Table4-1に API 関数一覧を示します。これらの API は各クラス共通で使用することができます。アプリケーションプログラムでは、下記の API をご使用ください。

Table4-1 API 一覧

API		説明
R_USB_Open()	(Note1, 5)	USB モジュール起動
R_USB_Close()	(Note1, 5)	USB モジュール停止
R_USB_GetVersion()		本モジュールのバージョン情報を取得
R_USB_Read()	(Note1, 5)	USB データリード要求
R_USB_Write()	(Note1, 5)	USB データライト要求
R_USB_Stop()	(Note1)	USB データリード/データライト停止処理
R_USB_Suspend()	(Note1, 5)	サスペンド要求
R_USB_Resume()	(Note1, 5)	レジューム要求
R_USB_GetEvent()	(Note1)	USB 関連の完了イベントを取得 (Non-OS のみ)
R_USB_Callback()	(Note1)	コールバック関数の登録 (RTOS のみ)
R_USB_VbusSetting()	(Note1, 5)	VBUS 供給開始/供給停止設定
R_USB_GetInformation()		USB デバイスについての情報を取得
R_USB_PipeRead()	(Note1, 5)	指定 PIPE からのデータリード要求
R_USB_PipeWrite()	(Note1, 5)	指定 PIPE へのデータライト要求
R_USB_PipeStop()	(Note1)	指定 PIPE に対するデータリード/データライト停止
R_USB_GetUsePipe()		使用 PIPE 番号を取得
R_USB_GetPipeInfo()		PIPE 情報を取得

[Note]

1. (Note1)の API 実行中、同じ USB モジュール上で、割り込み処理等により(Note1)の API が実行された場合、この USB ドライバは正常に動作しない場合があります。
2. Host Mass Storage Class では、上記の API 以外にデバイスクラス固有の API が用意されています。当該 API の詳細については、Host Mass Storage Class のドキュメント(Document number:R01AN2026)を参照してください。
3. Host Human Interface Device Class では、上記の API 以外にデバイスクラス固有の API が用意されています。当該 API の詳細については、Host Human Interface Device Class のドキュメント(Document number: R01AN2028)を参照してください。
4. USB_CFG_PARAM_CHECKING 定義に対し USB_CFG_DISABLE を指定した場合、引数チェック処理が行われないため、戻り値 USB_ERR_PARA は返されません。
   USB_CFG_PARAM_CHECKING 定義については、「8. **コンフィグレーション (r_usb_basic_config.h)**」を参照してください。
5. RTOS では、r_usb_basic_config.h ファイル内の USB_CFG_MULTI_TASK 定義に対し、USB_CFG_ENABLE を指定した場合､複数のタスクから上記の API をコールすることができます。「12.5.1 **セマフォ**」を参照してください。

4.1 R_USB_Open

USB モジュールの起動および USB ドライバの初期化を行います。(USB モジュールを使用する際に最初に使用する関数です。)

形式

usb_err_t R_USB_Open(usb_ctrl_t *p_ctrl, usb_cfg_t *p_cfg)

引数

p_ctrl	usb_ctrl_t 構造体領域へのポインタ
p_cfg	usb_cfg_t 構造体領域へのポインタ

戻り値

USB_SUCCESS	成功
USB_ERR_PARA	パラメータエラー
USB_ERR_BUSY	引数で指定された USB モジュールがすでに起動中

解説

引数(p_ctrl)に指定された USB モジュールの起動および USB ドライバの初期化処理を行います。

リエントラント

1. Non-OS の場合

   本 API は異なる USB モジュールに対して再入可能（リエントラント）です。

2. RTOS の場合

   本 API は再入可能（リエントラント）です。

補足

1. usb_ctrl_t 構造体については9.1章を、usb_cfg_t 構造体については9.3章を参照してください。
2. usb_ctrl_t 構造体のメンバ module には起動するモジュール番号(USB_IP0/USB_IP1)を指定してください。”USB_IP0”を指定すると USB0 モジュールが起動され、”USB_IP1”を指定すると USB1 モジュールが起動されます。なお、メンバ module に対し USB_IP0/USB_IP1 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。
3. ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合、メンバ module に対し USB_IP1 を指定しないでください。USB_IP1 を指定した場合は、戻り値に USB_ERR_PARA が返されます。
4. usb_ctrl_t 構造体のメンバ type に対しデバイスクラス種別(7章参照)を指定してください。なお、USB_HCDCC および USB_PCDCC を指定しないでください。USB_HCDCC または USB_PCDCC を指定した場合、USB_ERR_PARA が返されます。
5. usb_cfg_t 構造体のメンバ usb_mode には、USB Host として起動するときは"USB_HOST"を指定し、USB Peripheral として起動するときは"USB_PERI"を指定してください。なお、その指定がご使用になる USB モジュールでサポートしていない場合は、USB_ERR_PARA が返されます。
6. usb_cfg_t 構造体のメンバ usb_speed には、USB Speed(USB_HS / USB_FS)を指定してください。なお、指定したスピードがご使用になる USB モジュールでサポートしていない場合は、USB_ERR_PARA が返されます。
7. usb_cfg_t 構造体のメンバ p_usb_reg には、usb_descriptor_t 構造体へのポインタを指定してください。なお、本指定は、メンバ usb_mode に対し"USB_PERI"を指定した時にのみ有効な設定です。"USB_HOST"指定時、メンバ p_usb_reg に対する指定は無視されます。
8. いずれかの引数に対し 0(zero)を指定した場合、戻り値に USB_ERR_PARA が返されます。

9. RTOS の場合、セマフォ待ち状態に移行する場合があります。「12.5.1 **セマフォ**」を参照してください。
10. 以下の関数内で本 API をコールしないでください。
   (1). 割り込み関数
   (2). R_USB_Callback 関数で登録されたコールバック関数

使用例

1. USB Host モードの場合

```
void usb_host_application(void)
{
      usb_err_t       err;
      usb_ctrl_t      ctrl;
      usb_cfg_t       cfg;
                :
      ctrl.module = USB_IP0;
      ctrl.type = USB_HCDC;
      cfg.usb_mode = USB_HOST;
      cfg.usb_speed = USB_FS;
      err = R_USB_Open(&ctrl, &cfg);    /* Start USB module */
      if (USB_SUCCESS != err)
      {
            /* error */
      }
                :
}
```

2. USB Peripheral モードの場合

```
usb_descriptor_t smp_descriptor =
{
      g_device,
      g_config_f,
      g_config_h,
      g_qualifier,
      g_string
};
void    usb_peri_application(void)
{
      usb_err_t    err;
      usb_ctrl_t    ctrl;
      usb_cfg_t    cfg;
                              :
      ctrl.module = USB_IP1;
      ctrl.type = USB_PCDC;
      cfg.usb_mode = USB_PERI;
      cfg.usb_speed = USB_HS;
      cfg.p_usb_reg = &smp_descriptor;
      err = R_USB_Open(&ctrl, &cfg);    /* Start USB module */
      if (USB_SUCCESS != err)
      {
                  /* error */
      }
                              :
}
```

4.2 R_USB_Close

USB モジュールの停止

形式

usb_err_t R_USB_Close(usb_ctrl_t *p_ctrl)

引数

p_ctrl	usb_ctrl_t 構造体領域へのポインタ

戻り値

USB_SUCCESS	成功
USB_ERR_PARA	パラメータエラー
USB_ERR_NOT_OPEN	USB モジュールが Open されていない

解説

引数(p_ctrl)で指定された USB モジュールを停止します。メンバ module に USB_IP0 を指定すると USB0 モジュールが停止し、メンバ module に USB_IP1 を指定すると USB1 モジュールが停止します。

リエントラント

1. Non-OS の場合

   本 API は異なる USB モジュールに対して再入可能（リエントラント）です。

2. RTOS の場合

   本 API は再入可能（リエントラント）です。

補足

1. usb_ctrl_t 構造体のメンバ module には停止する USB モジュール番号(USB_IP0/USB_IP1)を指定してください。なお、メンバ module に対し USB_IP0/USB_IP1 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。
2. ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合、メンバ module に対し USB_IP1 を指定しないでください。USB_IP1 を指定した場合は、戻り値に USB_ERR_PARA が返されます。
3. 引数 p_ctrl に対し USB_NULL を指定した場合、戻り値に USB_ERR_PARA が返されます。
4. RTOS の場合、セマフォ待ち状態に移行する場合があります。「12.5.1 **セマフォ**」を参照してください。
5. 以下の関数内で本 API をコールしないでください。

   (1). 割り込み関数

   (2). R_USB_Callback 関数で登録されたコールバック関数

使用例

```
void      usr_application( void )
{
   usb_err_t       err;
   usb_ctrl_t      ctrl;
            :
   ctrl.module = USB_IP0
   err = R_USB_Close(&ctrl);              /* Stop USB module */
   if (USB_SUCCESS != err)
   {
     /* error */
   }
            :
}
```

4.3 R_USB_GetVersion

USB ドライバのバージョン情報を取得

形式

uint32_t R_USB_GetVersion(void)

引数

-- --

戻り値

バージョン番号

解説

USB ドライバのバージョン番号が返されます。

リエントラント

本 API は再入可能（リエントラント）です。

補足

--

使用例

```
void    usr_application( void )
{
  uint32_t    version;
          :
  version = R_USB_GetVersion();

}
```

4.4 R_USB_Read

USB データリード要求

形式

usb_err_t R_USB_Read(usb_ctrl_t *p_ctrl, uint8_t *p_buf, uint32_t size)

引数

p_ctrl	usb_ctrl_t 構造体領域へのポインタ
p_buf	リードデータを格納する領域へのポインタ
size	リード要求サイズ

戻り値

USB_SUCCESS	正常終了 (データリード要求完了)
USB_ERR_PARA	パラメータエラー
USB_ERR_BUSY	同じデバイスに対するデータ受信要求中
USB_ERR_NG	その他のエラー

解説

1. Bulk/Interrupt 転送の場合

(1). Non-OS の場合

USB データのリード(Bulk/Interrupt 転送)要求を行います。
リードしたデータは引数 p_buf が示す領域に格納されます。
データリードの完了は、R_USB_GetEvent 関数の戻り値(USB_STS_READ_COMPLETE)により確認することができます。
リード完了したデータのサイズは、R_USB_GetEvent 関数の戻り値(USB_STS_READ_COMPLETE)を確認後、usb_ctrl_t 構造体のメンバ size を参照してください。

(2). RTOS の場合

USB データのリード(Bulk/Interrupt 転送)要求を行います。
リードしたデータは引数 p_buf が示す領域に格納されます。
データリードの完了は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event:USB_STS_READ_COMPLETE)により確認することができます。
リード完了したデータのサイズは、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event:USB_STS_READ_COMPLETE)を確認後、usb_ctrl_t 構造体のメンバ size を参照してください。

2. Control 転送の場合

「10. **クラスリクエスト**」を参照してください。

リエントラント

1. Non-OS の場合

本 API は異なる USB モジュールに対して再入可能（リエントラント）です。

2. RTOS の場合

本 API は再入可能（リエントラント）です。

補足

1. この API はデータリード要求処理のみを行います。アプリケーションプログラムが、この API によりデータリード完了待ちになることはありません。

2. RTOS の場合、セマフォ待ち状態に移行する場合があります。「12.5.1 **セマフォ**」を参照してください。
3. 戻り値に USB_SUCCESS が返された場合、USB ドライバに対するデータリード要求を行ったのみで、まだ、データのリード処理は完了していません。
4. リード済みのデータがマックスパケットサイズの n 倍、かつリード要求サイズに満たない場合は、USB ドライバは、データ転送の途中であると判断するため、R_USB_GetEvent 関数の戻り値には、USB_STS_READ_COMPLETE がセットされません。(Non-OS の場合)
5. リード済みのデータがマックスパケットサイズの n 倍、かつリード要求サイズに満たない場合は、USB ドライバは、データ転送の途中であると判断するため、データ受信完了を通知するためのコールバック関数をコールしません。(RTOS の場合)
6. 本 API をコールする前に usb_ctrl_t 構造体のメンバ type に対しデバイスクラス種別(7章参照)を指定してください。なお、USB Host モードの場合は、アクセスする USB デバイスを識別するためメンバ module に USB モジュール番号(USB_IP0/USB_IP1)を、メンバ address にデバイスアドレスを指定してください。なお、メンバ module に対し USB_IP0/USB_IP1 以外を指定した場合やメンバ type に対しサポート対象外のデバイスクラス種別を指定した場合、戻り値に USB_ERR_PARA が返されます。
7. ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合、メンバ module に対し USB_IP1 を指定しないでください。USB_IP1 を指定した場合は、戻り値に USB_ERR_PARA が返されます。
8. 第 2 引数(p_buf)には、自動変数(スタック)領域へのポインタは指定しないでください。
9. 第 2 引数(p_buf)に指定する領域は、第 3 引数(size)で指定したサイズ以上の領域を確保してください。なお、DMA/DTC 転送によるデータリードを行う場合、以下に示すサイズを確保してください。

   (1). r_usb_basic_config.h 内の USB_CFG_CNTMD 定義に対し USB_CFG_CNTMDON を指定している場合 (USBA/USBAa モジュールを使用している場合)

   FIFO バッファサイズ×n 倍以上のサイズを確保してください。なお、FIFO バッファサイズについては、「12.4 **PIPEBUF レジスタの参照/変更について**」を参照してください。

   (2). r_usb_basic_config.h 内の USB_CFG_CNTMD 定義に対し USB_CFG_CNTMDOFF を指定している場合

   MaxPacketSize×n 倍のサイズを確保してください。

9. いずれかの引数に対し 0(zero)を指定した場合、戻り値に USB_ERR_PARA が返されます。
10. USB Host モード時、usb_ctrl_t 構造体のメンバ address の設定値が同じ値の R_USB_Read 関数を連続でコールすることはできません。連続で R_USB_Read 関数をコールした場合は、戻り値 USB_ERR_BUSY が返されます。メンバ address の設定値が同じ値の R_USB_Read 関数を再度コールする場合は、R_USB_GetEvent 関数からの戻り値 USB_STS_READ_COMPLETE を確認した後で R_USB_Read 関数をコールしてください。(Non-OS の場合)
11. USB Host モード時、usb_ctrl_t 構造体のメンバ address の設定値が同じ値の R_USB_Read 関数を連続でコールすることはできません。連続で R_USB_Read 関数をコールした場合は、戻り値 USB_ERR_BUSY が返されます。メンバ address の設定値が同じ値の R_USB_Read 関数を再度コールする場合は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event:USB_STS_READ_COMPLETE)を確認した後で、R_USB_Read 関数をコールしてください。(RTOS の場合)
12. USB Peripheral モード時、usb_ctrl_t 構造体のメンバ type の設定値が同じ値の R_USB_Read 関数を連続でコールすることはできません。連続で R_USB_Read 関数をコールした場合は、戻り値 USB_ERR_BUSY が返されます。メンバ type の設定値が同じ値の R_USB_Read 関数を再度コールする場合は、R_USB_GetEvent 関数からの戻り値 USB_STS_READ_COMPLETE を確認した後で R_USB_Read 関数をコールしてください。(Non-OS の場合)

13. USB Peripheral モード時、usb_ctrl_t 構造体のメンバ type の設定値が同じ値の R_USB_Read 関数を連続でコールすることはできません。連続で R_USB_Read 関数をコールした場合は、戻り値 USB_ERR_BUSY が返されます。メンバ type の設定値が同じ値の R_USB_Read 関数を再度コールする場合は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event:USB_STS_READ_COMPLETE)を確認後、R_USB_Read 関数をコールしてください。(RTOS の場合)
14. Vendor クラスの場合、R_USB_PipeRead 関数をご使用ください。
15. usb_ctrl_t 構造体のメンバ type に対し USB_PCDCC / USB_HMSC / USB_PMSC / USB_HVND / USB_PVND を指定した後、本 API をコールした場合、戻り値に USB_ERR_PARA が返されます。
16. Host Mass Storage Class の場合で、ストレージメディアに対するアクセスを行う場合は、FAT(File Allocation Table)の API を使用し、本 API は使用しないでください。
17. USB デバイスが CONFIGURED 状態の場合に、本 API をコールすることができます。CONFIGURED 以外の状態で本 API をコールすると戻り値に USB_ERR_NG が返されます。
18. 以下の関数内で本 API をコールしないでください。

(1). 割り込み関数

(2). R_USB_Callback 関数で登録されたコールバック関数

使用例

1. Non-OS

```
void    usb_application( void )
{
    usb_ctrl_t      ctrl;
                    :
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                    :
            case USB_STS_WRITE_COMPLETE:
                    :
                ctrl.module = USB_IP1
                ctrl.adderss = adr;
                ctrl.type = USB_HCDC;
                R_USB_Read(&ctrl, g_buf, DATA_LEN);
                    :
            break;
            case USB_STS_READ_COMPLETE:
                    :
            break;
                    :
        }
    }
}
```

2. RTOS

```
/* Callback function */
void usb_apl_callback (usb_ctrl_t *p_ctr, usb_hdl_t hdl, uint8_t is_request)
{
      USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}

void        usb_application_task( void )
{
    usb_ctrl_t      ctrl;
    usb_ctrl_t      *p_mess;
                :
    while(1)
    {
        USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
        ctrl = *p_mess;
        switch (ctrl.event)
        {
                        :
            case USB_STS_WRITE_COMPLETE:
                        :
                ctrl.module = USB_IP1
                ctrl.adderss = adr;
                ctrl.type = USB_HCDC;
                R_USB_Read(&ctrl, g_buf, DATA_LEN);
                        :
              break;
              case USB_STS_READ_COMPLETE:
                        :
              break;
                        :
          }
     }
}
```

4.5 R_USB_Write

USB データライト要求

形式

usb_err_t R_USB_Write(usb_ctrl_t *p_ctrl, uint8_t *p_buf, uint32_t size)

引数

p_ctrl	usb_ctrl_t 構造体領域へのポインタ
p_buf	ライトデータを格納した領域へのポインタ
size	ライトサイズ

戻り値

USB_SUCCESS	正常終了 (データライト要求完了)
USB_ERR_PARA	パラメータエラー
USB_ERR_BUSY	同じデバイスに対するデータライト要求中
USB_ERR_NG	その他のエラー

解説

1. Bulk/Interrupt 転送の場合

(1). Non-OS の場合

USB データのライト(Bulk/Interrupt 転送)要求を行います。
ライトするデータは引数 p_buf が示す領域に格納してください。
データライトの完了は、R_USB_GetEvent 関数の戻り値(USB_STS_WRITE_COMPLETE)により確認することができます。なお、NULL パケットの送信要求を行う場合は、第 3 引数(size)に対し USB_NULL(0)を指定してください。

(2). RTOS の場合

USB データのライト(Bulk/Interrupt 転送)要求を行います。
ライトするデータは引数 p_buf が示す領域に格納してください。
データライトの完了は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event: USB_STS_WRITE_COMPLETE)により確認することができます。なお、NULL パケットの送信要求を行う場合は、第 3 引数(size)に対し USB_NULL(0)を指定してください。

2. Control 転送の場合

「10. **クラスリクエスト**」を参照してください。

リエントラント

1. Non-OS の場合

本 API は異なる USB モジュールに対して再入可能（リエントラント）です。

2. RTOS の場合

本 API は再入可能（リエントラント）です。

補足

1. この API はデータライト要求処理のみを行います。アプリケーションプログラムが、この API によりデータライト完了待ちになることはありません。
2. RTOS の場合、セマフォ待ち状態に移行する場合があります。「12.5.1 **セマフォ**」を参照してください。

3. 戻り値に USB_SUCCESS が返された場合、USB ドライバに対するデータライト要求を行ったのみで、まだ、データのライト処理は完了していません。
4. usb_ctrl_t 構造体のメンバ(type)にデバイスクラス種別(7章参照)を指定した後で、本 API をコールしてください。なお、USB Host モードの場合は、アクセスする USB デバイスを識別するためメンバ module に USB モジュール番号(USB_IP0/USB_IP1)を、メンバ address に対してデバイスアドレスを指定してください。なお、メンバ module に対し USB_IP0/USB_IP1 以外を指定した場合やメンバ type に対しサポート対象外のデバイスクラス種別を指定した場合、戻り値に USB_ERR_PARA が返されます。
5. ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合、メンバ module に対し USB_IP1 を指定しないでください。USB_IP1 を指定した場合は、戻り値に USB_ERR_PARA が返されます。
6. 第 2 引数(p_buf)には、自動変数(スタック)領域へのポインタは指定しないでください。
7. 引数 p_ctrl に対し USB_NULL を指定した場合、戻り値に USB_ERR_PARA が返されます。
8. 引数 size に 0 以外を指定し、引数 buf に対し USB_NULL を指定した場合、戻り値に USB_ERR_PARA が返されます。
9. USB Host モード時、usb_ctrl_t 構造体のメンバ address の設定値が同じ値の R_USB_Write 関数を連続でコールすることはできません。連続で R_USB_Write 関数をコールした場合は、戻り値 USB_ERR_BUSY が返されます。address の設定値が同じ値の R_USB_Write 関数を再度コールする場合は、R_USB_GetEvent 関数からの戻り値 USB_STS_WRITE_COMPLETE を確認した後で R_USB_Write 関数をコールしてください。(Non-OS の場合)
10. USB Host モード時、usb_ctrl_t 構造体のメンバ address の設定値が同じ値の R_USB_Write 関数を連続でコールすることはできません。連続で R_USB_Write 関数をコールした場合は、戻り値 USB_ERR_BUSY が返されます。address の設定値が同じ値の R_USB_Write 関数を再度コールする場合は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event:USB_STS_WRITE_COMPLETE)を確認した後で R_USB_Write 関数をコールしてください。(RTOS の場合)
11. USB Peripheral モード時、usb_ctrl_t 構造体のメンバ type の設定値が同じ値の R_USB_Write 関数を連続でコールすることはできません。連続で R_USB_Write 関数をコールした場合は、戻り値 USB_ERR_BUSY が返されます。メンバ type の設定値が同じ値の R_USB_Write 関数を再度コールする場合は、R_USB_GetEvent 関数からの戻り値 USB_STS_WRITE_COMPLETE を確認した後で R_USB_Write 関数をコールしてください。(Non-OS の場合)
12. USB Peripheral モード時、usb_ctrl_t 構造体のメンバ type の設定値が同じ値の R_USB_Write 関数を連続でコールすることはできません。連続で R_USB_Write 関数をコールした場合は、戻り値 USB_ERR_BUSY が返されます。メンバ type の設定値が同じ値の R_USB_Write 関数を再度コールする場合は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event:USB_STS_WRITE_COMPLETE)を確認した後で R_USB_PipeWrite 関数をコールしてください。(RTOS の場合)
13. Vendor クラスの場合、R_USB_PipeWrite 関数をご使用ください。
14. usb_ctrl_t 構造体のメンバ type に対し USB_HCDCC/USB_HMSC/USB_PMSC/USB_HVND/USB_PVND を指定した後、本 API をコールした場合、戻り値に USB_ERR_PARA が返されます。
15. Host Mass Storage Class の場合で、ストレージメディアに対するアクセスを行う場合は、FAT(File Allocation Table)の API を使用し、本 API は使用しないでください。
16. USB デバイスが CONFIGURED 状態の場合に、本 API をコールすることができます。CONFIGURED 以外の状態で本 API をコールすると戻り値に USB_ERR_NG が返されます。
17. 以下の関数内で本 API をコールしないでください。
    (1). 割り込み関数
    (2). R_USB_Callback 関数で登録されたコールバック関数

使用例

1. Non-OS

```
void    usb_application( void )
{
    usb_ctrl_t    ctrl;
                  :
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                      :
            case USB_STS_READ_COMPLETE:
                      :
                ctrl.module = USB_IP0;
                ctrl.address = adr;
                ctrl.type = USB_HCDC;
                R_USB_Write(&ctrl, g_buf, size);
                      :
            break;
            case USB_STS_WRITE_COMPLETE:
                  :
            break;
                  :
        }
    }
}
```

2. RTOS

```
/* Callback function */
void usb_apl_callback (usb_ctrl_t *p_ctrl, usb_hdl_t hdl, uint8_t is_request)
{
    USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}

void    usb_application_task( void )
{
    usb_ctrl_t    ctrl;
    usb_ctrl_t    *p_mess;
        :
    while(1)
    {
        USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
        ctrl = *p_mess;
        switch (ctrl.event)
        {
                    :
            case USB_STS_READ_COMPLETE:
                    :
                ctrl.module = USB_IP0;
                ctrl.address = adr;
                ctrl.type = USB_HCDC;
                R_USB_Write(&ctrl, g_buf, size);
                    :
            break;
            case USB_STS_WRITE_COMPLETE:
                    :
            break;
                    :
        }
    }
}
```

4.6 R_USB_Stop

USB データのリード/ライト停止

形式

usb_err_t R_USB_Stop(usb_ctrl_t *p_ctrl, uint16_t type)

引数

p_ctrl	usb_ctrl_t 構造体領域へのポインタ
type	受信(USB_READ)または送信(USB_WRITE)

戻り値

USB_SUCCESS	正常終了 (停止完了)
USB_ERR_PARA	パラメータエラー
USB_ERR_NG	その他のエラー

解説

データリード/データライト転送を行っている場合、このデータ転送に対する停止を行います。
データリード要求を停止する場合、引数 type に対し USB_READ を指定し、データライト要求を停止する場合、引数 type に対し USB_WRITE を指定してください。

リエントラント

1. Non-OS の場合

本 API は異なる USB モジュールに対して再入可能（リエントラント）です。

2. RTOS の場合

本 API は再入可能（リエントラント）です。

補足

1. usb_ctrl_t 構造体のメンバ(type)にデバイスクラス種別を指定した後で、本 API をコールしてください。なお、USB Host モードの場合は、アクセスする USB デバイスを識別するためメンバ module に USB モジュール番号(USB_IP0/USB_IP1)を、メンバ address に対してデバイスアドレスを指定してください。なお、メンバ module に対し USB_IP0/USB_IP1 以外を指定した場合メンバ type に対しサポート対象外のデバイスクラス種別を指定した場合、戻り値に USB_ERR_PARA が返されます。
2. ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合、メンバ module に対し USB_IP1 を指定しないでください。USB_IP1 を指定した場合、戻り値に USB_ERR_PARA が返されます。
3. 引数 p_ctrl に対し USB_NULL を指定した場合、戻り値に USB_ERR_PARA が返されます。
4. 第 3 引数 type に対し USB_READ/USB_WRITE 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。なお、第 2 引数 type に USB_NULL を指定した場合、本ドライバは、USB_READ が指定された場合と同じ処理を行います。
5. usb_ctrl_t 構造体のメンバ(type)に USB_HCDCC を指定し、第 2 引数(type)に USB_WRITE を指定した場合は、戻り値に USB_ERR_PARA が返されます。
6. usb_ctrl_t 構造体のメンバ(type)に USB_PCDCC を指定し、第 2 引数(type)に USB_READ を指定した場合は、戻り値に USB_ERR_PARA が返されます。
7. USB Host モード時、データリード/ライト要求を停止できなかったときは、戻り値に USB_ERR_NG が返されます。
8. データリード/ライト停止が完了した後で R_USB_GetEvent 関数をコールすると戻り値 USB_STS_READ_COMPLETE/USB_STS_WRITE_COMPLETE が返されます。(Non-OS の場合)

9. USB ドライバはデータリードまたはデータライト停止が完了すると USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event)に USB_STS_READ_COMPLETE または USB_STS_WRITE_COMPLETE をセットします。(RTOS の場合)
10. usb_ctrl_t 構造体のメンバ type に対し USB_HMSC/USB_PMSC/USB_HVND/USB_PVND を指定した後、本 API をコールした場合、戻り値に USB_ERR_PARA が返されます。
11. Vendor クラスの場合、R_USB_PipeStop 関数をご使用ください。
12. Host Mass Storage Class の場合、本 API は使用しないでください。
13. USB デバイスが CONFIGURED 状態の場合に、本 API をコールすることができます。CONFIGURED 以外の状態で本 API をコールすると戻り値に USB_ERR_NG が返されます。
14. 以下の関数内で本 API をコールしないでください。

(1). 割り込み関数

(2). R_USB_Callback 関数で登録されたコールバック関数

使用例

1. Non-OS

```
void        usb_application( void )
{
      usb_ctrl_t       ctrl;
                              :
      while (1)
      {
            switch (R_USB_GetEvent(&ctrl))
            {
                              :
                  case USB_STS_DETACH:
                              :
                        ctrl.module = USB_IP1;
                        ctrl.address = adr;
                        ctrl.type = USB_HCDC;
                        R_USB_Stop(&ctrl, USB_READ );     /* Receive stop */
                        R_USB_Stop(&ctrl, USB_WRITE );    /* Send stop */
                              :
                  break;
                              :
            }
      }
}
```

2. RTOS

```
/* Callback function */
void usb_apl_callback (usb_ctrl_t *p_ctrl, usb_hdl_t hdl, uint8_t is_request)
{
      USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}

void          usb_application_task( void )
{
    usb_ctrl_t      ctrl;
    usb_ctrl_t      *p_mess;
                :
    while(1)
    {
                USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
                ctrl = *p_mess;
                switch (ctrl.event)
                {
                                :
                   case USB_STS_DETACH:
                                :
                        ctrl.module = USB_IP1;
                        ctrl.address = adr;
                        ctrl.type = USB_HCDC;
                        R_USB_Stop(&ctrl, USB_READ );     /* Receive stop */
                        R_USB_Stop(&ctrl, USB_WRITE );    /* Send stop */
                                :
                   break;
                                :
            }
     }
}
```

4.7 R_USB_Suspend

Suspend 信号送信

形式

usb_err_t R_USB_Suspend(usb_ctrl_t *p_ctrl)

引数

p_ctrl	usb_ctrl_t 構造体領域へのポインタ

戻り値

USB_SUCCESS	正常終了
USB_ERR_PARA	パラメータエラー
USB_ERR_BUSY	指定した USB モジュールに対するサスペンド要求中、またはその USB モジュールがすでにサスペンド状態中
USB_ERR_NG	その他のエラー

解説

1. Non-OS の場合

usb_ctrl_t 構造体のメンバ module に指定された USB モジュールから SUSPEND 信号を送信します。なお、Suspend 要求の完了は、R_USB_GetEvent 関数の戻り値(USB_STS_SUSPEND)から確認することができます。

2. RTOS の場合

usb_ctrl_t 構造体のメンバ module に指定された USB モジュールから SUSPEND 信号を送信します。なお、Suspend 要求の完了は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event: USB_STS_SUSPEND)により確認することができます。

リエントラント

1. Non-OS の場合

本 API は異なる USB モジュールに対して再入可能（リエントラント）です。

2. RTOS の場合

本 API は再入可能（リエントラント）です。

補足

1. この API は Suspend 信号送信要求のみを行います。アプリケーションプログラムが、この API により Suspend 信号送信完了待ちになることはありません。
2. RTOS の場合、セマフォ待ち状態に移行する場合があります。「12.5.1 **セマフォ**」を参照してください。
3. 本 API は、USB Host モード時にのみ使用することができます。USB Peripheral モード時に、本 API を使用した場合は、戻り値に USB_ERR_NG が返されます。
4. 本 API は、Selective Suspend 機能をサポートしておりません。
5. SUSPEND 信号を送信する USB モジュールの指定は、usb_ctrl_t 構造体のメンバ module に対し行ってください。メンバ(module)に対し USB_IP0/USB_IP1 を指定してください。なお、メンバ module に対し USB_IP0/USB_IP1 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。

6. ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合、メンバ module に対し USB_IP1 を指定しないでください。USB_IP1 を指定した場合は、戻り値に USB_ERR_PARA が返されます。
7. 引数 p_ctrl に対し USB_NULL を指定した場合、戻り値に USB_ERR_PARA が返されます。
8. USB デバイスが CONFIGURED または SUSPEND 以外の状態で本 API をコールすると戻り値に USB_ERR_NG が返されます。
9. 以下の関数内で本 API をコールしないでください。

(1). 割り込み関数

(2). R_USB_Callback 関数で登録されたコールバック関数

使用例

1. Non-OS

```
void usb_host_application( void )
{
    usb_ctrl_t ctrl;
                :
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                    :
            case USB_STS_NONE:
                    :
                ctrl.module = USB_IP0;
                R_USB_Suspend(&ctrl);
            break;
            case USB_STS_SUSPEND:
                :
            break;
                    :
        }
    }
}
```

2. RTOS

```
/* Callback function */
void usb_apl_callback (usb_ctrl_t *p_ctrl, usb_hdl_t hdl, uint8_t is_request)
{
      USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}

void usb_application_task( void )
{
    usb_ctrl_t      ctrl;
    usb_ctrl_t      *p_mess;
              :
    while(1)
    {
          USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
          ctrl = *p_mess;
          switch (ctrl.event)
          {
                        :
                    ctrl.module = USB_IP0;
                    R_USB_Suspend(&ctrl);
                break;
                case USB_STS_SUSPEND:
                            :
                break;
                            :
            }
       }
}
```

4.8 R_USB_Resume

RESUME 信号送信

形式

usb_err_t R_USB_Resume(usb_ctrl_t *p_ctrl)

引数

p_ctrl	usb_ctrl_t 構造体領域へのポインタ

戻り値

USB_SUCCESS	正常終了
USB_ERR_PARA	パラメータエラー
USB_ERR_BUSY	同じデバイスアドレスに対するレジューム要求中 (USB Host モード時のみ)
USB_ERR_NOT_SUSPEND	USB デバイスが SUSPEND 状態ではない

解説

1. Non-OS の場合

   usb_ctrl_t 構造体のメンバ module に指定された USB モジュールから RESUME 信号を送信します。
   なお、レジューム要求の完了は、R_USB_GetEvent 関数の戻り値(USB_STS_RESUME)から確認することができます。

2. RTOS の場合

   usb_ctrl_t 構造体のメンバ module に指定された USB モジュールから RESUME 信号を送信します。
   なお、レジューム要求の完了は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event: USB_STS_RESUME)により確認することができます。

リエントラント

1. Non-OS の場合

   本 API は異なる USB モジュールに対して再入可能（リエントラント）です。

2. RTOS の場合

   本 API は再入可能（リエントラント）です。

補足

1. この API は Resume 信号送信要求のみを行います。アプリケーションプログラムが、この API により Resume 信号送信完了待ちになることはありません。
2. RTOS の場合、セマフォ待ち状態に移行する場合があります。「12.5.1 **セマフォ**」を参照してください。
3. 本 API は、R_USB_Open 関数をコールした後(R_USB_Close 関数をコールする前)で呼び出してください。
4. USB Peripheral モードで、Feature Selector に DEVICE_REMOTE_WAKEUP が指定された SetFeature コマンドを受信した場合のみ、RemoteWakeup 信号送信用として本 API を使用することができます。なお、当該 SetFeature コマンドを受信する前に本 API をコールすると戻り値に USB_ERR_NG が返されます。

5. RESUME 信号を送信する USB モジュールの指定は、usb_ctrl_t 構造体のメンバ module に対し行ってください。メンバ module に対し USB_IP0/USB_IP1 を指定してください。なお、ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合、メンバ module に対し USB_IP1 を指定しないでください。USB_IP1 を指定した場合は、戻り値に USB_ERR_PARA が返されます。
6. 引数 p_ctrl に対し USB_NULL を指定した場合、戻り値に USB_ERR_PARA が返されます。
7. USB デバイスが SUSPEND 状態の場合に、本 API をコールすることができます。SUSPEND 状態以外の状態で本 API をコールすると戻り値に USB_ERR_NOT_SUSPEND が返されます。
8. 以下の関数内で本 API をコールしないでください。
   (1). 割り込み関数
   (2). R_USB_Callback 関数で登録されたコールバック関数

使用例

1. Non-OS

(1). In the case of USB Host mode

```
void  usb_host_application( void )
{
      usb_ctrl_t ctrl;
                              :
      while (1)
      {
          switch (R_USB_GetEvent( &ctrl ))
           {
                              :
                case USB_STS_NONE:

                     ctrl.module = USB_IP0;
                     R_USB_Resume( &ctrl );
                        :
                break;
                case USB_STS_RESUME:
                      :
                break;
                              :
            }
      }
}
```

(2). In the case of USB Peripheral mode

```
void  usb_peri_application( void )
{
      usb_ctrl_t ctrl;
                        :
      while (1)
      {
          switch (R_USB_GetEvent( &ctrl ))
           {
                        :
                case USB_STS_NONE:
                        :
                  R_USB_Resume(&ctrl);
                        :
                break;
                case USB_STS_RESUME:
                        :
                break;
                        :
           }
      }
}
```

2. RTOS

(1). In the case of USB Host mode

```
/* Callback function */
void usb_apl_callback (usb_ctrl_t *p_ctrl, usb_hdl_t hdl, uint8_t is_request)
{
      USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}

void          usb_application_task( void )
{
    usb_ctrl_t      ctrl;
    usb_ctrl_t      *p_mess;
              :
    while(1)
    {
          USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
          ctrl = *p_mess;
          switch (ctrl.event)
          {
                :
                  ctrl.module = USB_IP0;
                  R_USB_Resume( &ctrl );
                      :
              break;
              case USB_STS_RESUME:
                              :
              break;
                        :
            }
      }
}
```

(2). In the case of USB Peripheral mode

```
void        usb_peri_application( void )
{
    usb_ctrl_t      ctrl;
    usb_ctrl_t      *p_mess;
            :
    while(1)
    {
        USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
        ctrl = *p_mess;
        switch (ctrl.event)
        {
                    :
                R_USB_Resume(&ctrl);
                    :
            break;
            case USB_STS_RESUME:
                        :
            break;
                    :
         }
    }
}
```

4.9 R_USB_GetEvent

USB 関連の完了イベントを取得する。 (Non-OS のみ)

形式

uint16_t R_USB_GetEvent(usb_ctrl_t *p_ctrl)

引数

p_ctrl usb_ctrl_t 構造体領域へのポインタ

戻り値

-- USB 関連の完了イベントの値

解説

USB 関連の完了イベントを取得します。
USB Host モードの場合、usb_ctrl_t 構造体のメンバ address にイベントが完了した USB デバイスのデバイスアドレス値が設定されます。なお、USB Peripheral モードの場合、メンバ address には USB_NULL が指定されます。

リエントラント

本 API は再入不可能（リエントラント不可）です。

補足

1. 本 API は、R_USB_Open 関数をコールした後(R_USB_Close 関数をコールする前)で呼び出してください。

2. 本 API の戻り値である完了イベントの値の詳細については、「**5. コールバック関数 (RTOSのみ)**
USBドライバは、USBイベントが完了するとコールバック関数をコールします。コールバック関数は、ユーザアプリケーションプログラムとしてユーザが作成し、R_USB_Callback関数を使ってUSBドライバにそのコールバック関数を登録する必要があります。
USBドライバに登録するコールバック関数は、以下に示す引数および戻り値をサポートしてください。

```
引数    :  usb_ctrl_t   *p_ctrl      // usb_ctrl_t構造体領域へのポインタ
        :  usb_hdl_t    hdl          // USBイベントが完了したタスクハンドル
        :  uint8_t      is_request   //  クラスリクエスト受信フラグ(USB_ON / USB_OFF)
戻り値  :  void                      //  なし
```

[Note]

(1). 引数(p_ctrl)には、USB完了イベントのほか、そのイベントに応じた各種情報がUSBドライバによって設定されています。必ず、RTOSのAPI等を使ってアプリケーション用タスクへ当該引数の情報を通知してください。

(2). 引数(p_ctrl)のメンバeventが以下の場合、そのeventに関連するAPIをコールしたアプリケーションタスクのタスクハンドルが、引数(hdl)に設定されます。その他の場合、引数(hdl)にはUSB_NULLが設定されます。

a. USB_STS_READ_COMPLETE
b. USB_STS_WRITE_COMPLETE
c. USB_STS_REQUEST_COMPLETE (Note a)
d. USB_STS_SUSPEND (Note b)
e. USB_STS_RESUME (Note b)
f. USB_STS_MSC_CMD_COMPLETE

Note:

a. USB Peripheralモードで、ノーデータステータスステージをサポートするリクエストを受信した時、引数(hdl)にはUSB_NULLが設定されます。

b. USB Peripheralモードの場合、引数(hdl)にはUSB_NULLが設定されます。

(3). USB Peripheralモードで、クラスリクエストを受信した場合、引数(is_request)にはUSB_ONが設定されます。その他の場合、USB_OFFが設定されます。引数(is_request)がUSB_ONの時、引数(p_ctrl)のメンバsetupには、そのクラスリクエストに関する情報が設定されています。

記述例)

```
void usb_apl_callback (usb_ctrl_t *p_ctrl, usb_hdl_t hdl, uint8_t is_request)
{
    /* RTOSのAPIを使ってUSBイベントの情報をアプリケーションタスクに通知 */
    USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}
```

3. **R_USB_GetEvent関数の戻り値**」を参照してください。
4. 本 API をコールした時に、完了したイベントが無い場合は、戻り値に"USB_STS_NONE"が返されます。
5. 本 API をユーザアプリケーションプログラムのメインループからコールしてください。
6. 割り込み関数内で本 API をコールしないでください。

使用例

```
void usb_host_application( void )
{
    usb_ctrl_t      ctrl;
                        :
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                        :
            case USB_STS_CONFIGURED:
                        :
            break;
                        :
        }
    }
}
```

4.10 R_USB_Callback

USB 関連イベント完了時にコールされるコールバック関数の登録 (RTOS のみ)

形式

uint16_t R_USB_Callback(usb_callback_t *p_callback)

引数

p_callback コールバック関数へのポインタ

戻り値

-- なし

リエントラント

本 API は再入不可能（リエントラント不可）です。

解説

USB 関連のイベントが完了するとコールされるコールバック関数の登録を行います。
USB ドライバは、USB 関連のイベントが完了すると本 API により登録されたコールバック関数をコールします。

補足

1. 本 API は、R_USB_Open 関数をコールした後(R_USB_Close 関数をコールする前)で呼び出してください。
2. 本 API の引数に設定される USB イベントの値の詳細については、「6. **R_USB_GetEvent関数の戻り値 / USB完了イベントの取得**」を参照してください。
3. コールバック関数については、「**5. コールバック関数 (RTOSのみ)**」を参照してください。
4. 割り込み関数内で本 API をコールしないでください。

使用例

```
void usb_apl_callback (usb_ctrl_t *p_ctrl, usb_hdl_t hdl, uint8_t is_request)
{
    USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}

void usb_application_task(void)
{
    usb_ctrl_t    ctrl;
    usb_ctrl_t    *p_mess;
    usb_cfg_t     cfg;

    usb_pin_setting(); /* USB MCU pin setting */

    ctrl.module          = USE_USBIP;
    ctrl.type            = USB_PCDC;
    cfg.usb_speed     = USB_SUPPORT_SPEED;        /* USB_HS/USB_FS */
    cfg.p_usb_reg     = (usb_descriptor_t *)&usb_descriptor;
    R_USB_Open(&ctrl, &cfg);                      /* Initializes the USB module */

    R_USB_Callback(usb_apl_callback);

    while (1)
    {
        USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);

        ctrl = *p_mess;

        switch (ctrl.event)
        {
                :
        }
    }
}
```

4.11 R_USB_VbusSetting

VBUS 供給開始/供給停止設定

形式

usb_err_t R_USB_VbusSetting(usb_ctrl_t *p_ctrl, uint16_t state)

引数

p_ctrl	usb_ctrl_t 構造体領域へのポインタ
state	VBUS 供給開始/供給停止設定

戻り値

USB_SUCCESS	正常終了 (VBUS 供給開始/停止設定完了)
USB_ERR_PARA	パラメータエラー
USB_ERR_NG	その他のエラー

解説

VBUS 供給開始または供給停止設定をおこないます。

リエントラント

1. Non-OS の場合

   本 API は異なる USB モジュールに対して再入可能（リエントラント）です。

2. RTOS の場合

   本 API は再入可能（リエントラント）です。

補足

1. USB Host 用電源 IC の VBUS 出力が Low アサートか High アサートかの設定については、「8. **コンフィグレーション (r_usb_basic_config.h)**」章にある USB_CFG_VBUS 定義に対する設定を参照してください。
2. 第一引数(p_ctrl)のメンバ module には、VBUS 供給開始/供給停止設定をおこなうモジュール番号(USB_IP0/USB_IP1)を指定してください。”USB_IP0”を指定すると USB0 モジュールに対し設定処理が行われ、”USB_IP1”を指定すると USB1 モジュールに対する設定処理が行われます。なお、メンバ module に対し USB_IP0/USB_IP1 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。
3. ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合、メンバ module に対し USB_IP1 を指定しないでください。USB_IP1 を指定した場合は、戻り値に USB_ERR_PARA が返されます。
4. 第二引数には、"USB_ON"/"USB_OFF"を指定してください。VBUS 供給を開始する場合は、"USB_ON"を、VBUS 供給を停止する場合は、"USB_OFF"を指定してください。USB_ON/USB_OFF 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。なお、第 2 引数 state に USB_NULL を指定した場合、本ドライバは、USB_OFF が指定された場合と同じ処理を行います。
5. アプリケーションプログラムで VBUS の制御が必要になった場合のみ、本 API をご使用ください。(USB ドライバは初期化処理内で VBUS を ON にした後、VBUS を制御しません。)
6. 引数 p_ctrl に対し USB_NULL を指定した場合、戻り値に USB_ERR_PARA が返されます。
7. RTOS の場合、セマフォ待ち状態に移行する場合があります。「12.5.1 **セマフォ**」を参照してください。

8. 本 API は、USB Host モードの場合のみ処理されます。USB Peripheral モードの場合に、本 API をコールした場合は、戻り値に USB_ERR_NG が返されます。

9. 以下の関数内で本 API をコールしないでください。

   (1). 割り込み関数

   (2). R_USB_Callback 関数で登録されたコールバック関数

使用例

```
void usb_host_application( void )
{
    usb_ctrl_t ctrl;
            :
            :
    ctrl.module = USB_IP0;
    R_USB_VbusSetting(&ctrl, USB_ON ); /* VBUS 供給開始 */
            :
            :
    ctrl.module = USB_IP0;
    R_USB_VbusSetting(&ctrl, USB_OFF); /* VBUS 供給停止 */
            :
            :
}
```

4.12 R_USB_GetInformation

USB デバイスについての情報を取得

形式

usb_err_t R_USB_GetInformation(usb_ctrl_t *p_ctrl, usb_info_t *p_info)

引数

p_ctrl	usb_ctrl_t 構造体領域へのポインタ
p_info	usb_info_t 構造体領域へのポインタ

戻り値

USB_SUCCESS	正常終了
USB_ERR_PARA	パラメータエラー

解説

USB デバイスに関する情報を取得します。
取得情報については、「**9.6 usb_info_t構造体**」を参照してください。

リエントラント

本 API は再入可能（リエントラント）です。

補足

1. 本 API は、R_USB_Open 関数をコールした後(R_USB_Close 関数をコールする前)で呼び出してください。
2. USB Host モードの場合は、情報を取得する USB デバイスを識別するため usb_ctrl_t 構造体のメンバ module に USB モジュール番号(USB_IP0/USB_IP1)を、メンバ address にデバイスアドレスを指定してください。なお、メンバ module に対し USB_IP0/USB_IP1 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。
3. ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合、メンバ module に対し USB_IP1 を指定しないでください。USB_IP1 を指定した場合は、戻り値に USB_ERR_PARA が返されます。
4. USB Peripheral モードの場合、第一引数 p_ctrl に対し USB_NULL を指定してください。
5. USB Host モードの場合、第一引数 p_ctrl に対し USB_NULL を指定しないでください。USB_NULL を指定した場合、戻り値に USB_ERR_PARA が返されます。
6. 第二引数 p_info に対し USB_NULL を指定しないでください。USB_NULL を指定した場合、戻り値に USB_ERR_PARA が返されます。

使用例

a. USB Host モードの場合

```
void        usb_host_application( void )
{
      usb_ctrl_t ctrl;
      usb_info_t info;
            :
      ctrl.module = USB_IP0;
      ctrl.address = adr;
      R_USB_GetInformation(&ctrl, &info);
            :
}
```

b. USB Peripheral モードの場合

```
void usb_peri_application( void )
{
      usb_ctrl_t ctrl;
      usb_info_t info;
            :
      R_USB_GetInformation((usb_ctrl_t *)USB_NULL, &info);
            :
}
```

4.13 R_USB_PipeRead

指定 PIPE からのデータリード要求

形式

usb_err_t R_USB_PipeRead(usb_ctrl_t *p_ctrl, uint8_t *p_buf, uint32_t size)

引数

p_ctrl	usb_ctrl_t 構造体領域へのポインタ
p_buf	データを格納する領域へのポインタ
size	リード要求サイズ

戻り値

USB_SUCCESS	正常終了 (リード要求完了)
USB_ERR_PARA	パラメータエラー
USB_ERR_BUSY	指定 PIPE に対するデータ受信/送信要求中
USB_ERR_NG	その他のエラー

解説

1. Non-OS の場合

引数で指定した PIPE を使ったデータリード(Bulk/Interrupt 転送)要求を行います。
リードしたデータは引数 p_buf が示す領域に格納されます。
データリードの完了は、R_USB_GetEvent 関数の戻り値(USB_STS_READ_COMPLETE)から確認することができます。
受信したデータのサイズは、戻り値(USB_STS_READ_COMPLETE)を確認後、usb_ctrl_t 構造体のメンバ size を参照してください。

2. RTOS の場合

引数で指定した PIPE を使ったデータリード(Bulk/Interrupt 転送)要求を行います。
リードしたデータは引数 buf が示す領域に格納されます。
データリードの完了は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event:USB_STS_READ_COMPLETE)により確認することができます。
受信したデータのサイズは、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event:USB_STS_READ_COMPLETE)を確認後、usb_ctrl_t 構造体のメンバ size を参照してください。

リエントラント

1. Non-OS の場合

本 API は異なる USB モジュールに対して再入可能（リエントラント）です。

2. RTOS の場合

本 API は再入可能（リエントラント）です。

補足

1. この API はデータリード要求処理のみを行います。アプリケーションプログラムが、この API によりデータリード完了待ちになることはありません。
2. RTOS の場合、セマフォ待ち状態に移行する場合があります。「12.5.1 **セマフォ**」を参照してください。
3. 戻り値に USB_SUCCESS が返された場合、USB ドライバに対するデータリード要求を行ったのみで、まだ、データのリード処理は完了していません。

4. リード済みのデータサイズがマックスパケットサイズの n 倍、かつリード要求サイズに満たない場合は、USB ドライバはデータ転送の途中であると判断するため、R_USB_GetEvent 関数の戻り値には、USB_STS_READ_COMPLETE がセットされません。(Non-OS の場合)

5. リード済みのデータサイズがマックスパケットサイズの n 倍、かつリード要求サイズに満たない場合は、USB ドライバはデータ転送の途中であると判断するため、データ受信完了を通知するためのコールバック関数をコールしません。(RTOS の場合)

6. 本 API をコールする前に usb_ctrl_t 構造体のメンバ pipe に対し使用する PIPE 番号(USB_PIPE1 から USB_PIPE9)を指定してください。なお、USB Host モードの場合は、アクセスする USB デバイスを識別するためメンバ module に USB モジュール番号(USB_IP0/USB_IP1)を、メンバ address にデバイスアドレスを指定してください。なお、メンバ module に対し USB_IP0/USB_IP1 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。

7. ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合、メンバ module に対し USB_IP1 を指定しないでください。USB_IP1 を指定した場合は、戻り値に USB_ERR_PARA が返されます。

8. usb_ctrl_t 構造体のメンバ pipe に対し、USB_PIPE1 から USB_PIPE9 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。

9. 第 2 引数(p_buf)には、自動変数(スタック)領域へのポインタは指定しないでください。

8. 第 2 引数(p_buf)に指定する領域は､第 3 引数(size)で指定したサイズ以上の領域を指定してください。なお、DMA/DTC 転送によるデータリードを行う場合、以下に示すサイズを確保してください。

   (1). r_usb_basic_config.h 内の USB_CFG_CNTMD 定義に対し USB_CFG_CNTMDON を指定している場合 (USBA/USBAa モジュールを使用している場合)

   FIFO バッファサイズ×n 倍以上のサイズを確保してください。なお、FIFO バッファサイズについては、「12.4 **PIPEBUF レジスタの参照/変更について**」を参照してください。

   (2). r_usb_basic_config.h 内の USB_CFG_CNTMD 定義に対し USB_CFG_CNTMDOFF を指定している場合

   MaxPacketSize×n 倍のサイズを確保してください。

9. いずれかの引数に対し 0(zero)を指定した場合、戻り値に USB_ERR_PARA が返されます。

10. usb_ctrl_t 構造体のメンバ pipe の設定値が同じ値の R_USB_PipeRead 関数を連続でコールすることはできません。連続で R_USB_PipeRead 関数をコールした場合は、戻り値 USB_ERR_BUSY が返されます。メンバ pipe の設定値が同じ値の R_USB_PipeRead 関数を再度コールする場合は、R_USB_GetEvent 関数からの戻り値 USB_STS_READ_COMPLETE を確認した後で R_USB_PipeRead 関数をコールしてください。(Non-OS の場合)

11. usb_ctrl_t 構造体のメンバ pipe の設定値が同じ値の R_USB_PipeRead 関数を連続でコールすることはできません。連続で R_USB_PipeRead 関数をコールした場合は、戻り値 USB_ERR_BUSY が返されます。メンバ pipe の設定値が同じ値の R_USB_PipeRead 関数を再度コールする場合は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event:USB_STS_READ_COMPLETE)を確認後、R_USB_PipeRead 関数をコールしてください。(RTOS の場合)

12. CDC/HID クラスの Bulk/Interrupt 転送を行う場合は、R_USB_Read 関数を使用し、本 API は使用しないでください。また、Host Mass Storage クラスの MSC デバイスに対するデータアクセスを行う場合、FAT(File Allocation Table)の API を使用し、本 API は使用しないでください。

13. usb_ctrl_t 構造体のメンバ type に対する指定は行わないでください。メンバ type にデバイスクラス種別等を指定しても、その指定は無視されます。

14. Control 転送用のデータ転送を行う場合は、R_USB_Read 関数を使用し、本 API は使用しないでください。

15. 本 API を使用する場合、r_usb_basic_config.h ファイル内の USB_CFG_HVND_USE / USB_CFG_PVND_USE のいずれかの定義を有効にしてください。これらの定義が有効になっていない状態で本 API を使用した場合、戻り値に USB_ERR_NG が返されます。USB_CFG_HVND_USE / USB_CFG_PVND_CFG 定義については、「8. **コンフィグレーション (r_usb_basic_config.h)**」を参照してください。
16. USB デバイスが CONFIGURED 状態の場合に、本 API をコールすることができます。CONFIGURED 以外の状態で本 API をコールすると戻り値に USB_ERR_NG が返されます。
17. 以下の関数内で本 API をコールしないでください。(RTOS のみ)
    (1). 割り込み関数
    (2). R_USB_Callback 関数で登録されたコールバック関数

使用例

1. Non-OS

```
void        usb_application( void )
{
    usb_ctrl_t ctrl;
                    :
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                    :
            case USB_STS_WRITE_COMPLETE:
                    :
                ctrl.module = USB_IP1;
                ctrl.pipe = USB_PIPE1;
                R_USB_PipeRead(&ctrl, buf, size);
                    :
            break;
            case USB_STS_READ_COMPLETE:
                    :
            break;
                    :
        }
    }
}
```

2. RTOS

```
/* Callback Function */
void usb_apl_callback (usb_ctrl_t *p_ctrl, usb_hdl_t hdl, uint8_t is_request)
{
      USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}

/* Application Task */
void usb_application_task( void )
{
    usb_ctrl_t ctrl;
    usb_ctrl_t *p_mess;
              :
    while(1)
    {
          USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
          ctrl = *p_mess;
          switch (ctrl.event)
          {
                        :
              case USB_STS_WRITE_COMPLETE:
                        :
                  ctrl.module = USB_IP1;
                  ctrl.pipe = USB_PIPE1;
                  R_USB_PipeRead(&ctrl, buf, size);
                        :
              break;
              case USB_STS_READ_COMPLETE:
                        :
              break;
                        :
          }
      }
}
```

4.14 R_USB_PipeWrite

指定 PIPE へのデータライト要求

形式

usb_err_t R_USB_PipeWrite(usb_ctrl_t *p_ctrl, uint8_t *p_buf, uint32_t size)

引数

p_ctrl	usb_ctrl_t 構造体領域へのポインタ
p_buf	データを格納した領域へのポインタ
size	ライト要求サイズ

戻り値

USB_SUCCESS	正常終了(ライト要求完了)
USB_ERR_PARA	パラメータエラー
USB_ERR_BUSY	指定 PIPE に対するデータ送信/受信要求中
USB_ERR_NG	その他のエラー

解説

1. Non-OS の場合

データライト(Bulk/Interrupt 転送)要求を行います。
ライトするデータは引数 p_buf が示す領域に格納してください。
データライトの完了は、R_USB_GetEvent 関数の戻り値(USB_STS_WRITE_COMPLETE)から確認することができます。
なお、NULL パケットの送信要求を行う場合は、第 3 引数(size)に対し USB_NULL(0)を指定してください。

2. RTOS の場合

データライト(Bulk/Interrupt 転送)要求を行います。
ライトするデータは引数 buf が示す領域に格納してください。
データライトの完了は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event: USB_STS_WRITE_COMPLETE)により確認することができます。
なお、NULL パケットの送信要求を行う場合は、第 3 引数(size)に対し USB_NULL(0)を指定してください。

リエントラント

1. Non-OS の場合

本 API は異なる USB モジュールに対して再入可能（リエントラント）です。

2. RTOS の場合

本 API は再入可能（リエントラント）です。

補足

1. この API はデータライト要求処理のみを行います。アプリケーションプログラムが、この API によりデータライト完了待ちになることはありません。
2. RTOS の場合、セマフォ待ち状態に移行する場合があります。「12.5.1 **セマフォ**」を参照してください。
3. 戻り値に USB_SUCCESS が返された場合、USB ドライバに対するデータライト要求を行ったのみで、まだ、データのライト処理は完了していません。

4. 本 API をコールする前に usb_ctrl_t 構造体のメンバ pipe に対し使用する PIPE 番号(USB_PIPE1 から USB_PIPE9)を指定してください。なお、USB Host モードの場合は、アクセスする USB デバイスを識別するためメンバ module に USB モジュール番号(USB_IP0/USB_IP1)を、メンバ address にデバイスアドレスを指定してください。なお、メンバ module に対し USB_IP0/USB_IP1 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。
5. ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合、メンバ module に対し USB_IP1 を指定しないでください。USB_IP1 を指定した場合は、戻り値に USB_ERR_PARA が返されます。
6. usb_ctrl_t 構造体のメンバ pipe に対し、USB_PIPE1 から USB_PIPE9 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。
7. 第 2 引数(p_buf)には、自動変数(スタック)領域へのポインタは指定しないでください。
8. 引数 p_ctrl または引数 buf に対し 0(zero)を指定した場合、戻り値に USB_ERR_PARA が返されます。
9. usb_ctrl_t 構造体のメンバ pipe の設定値が同じ値の R_USB_PipeWrite 関数を連続でコールすることはできません。連続で R_USB_PipeWrite 関数をコールした場合は、戻り値 USB_ERR_BUSY が返されます。メンバ pipe の設定値が同じ値の R_USB_PipeWrite 関数を再度コールする場合は、R_USB_GetEvent 関数からの戻り値 USB_STS_WRITE_COMPLETE を確認した後で R_USB_PipeWrite 関数をコールしてください。(Non-OS の場合)
10. usb_ctrl_t 構造体のメンバ pipe の設定値が同じ値の R_USB_PipeWrite 関数を連続でコールすることはできません。連続で R_USB_PipeWrite 関数をコールした場合は、戻り値 USB_ERR_BUSY が返されます。メンバ pipe の設定値が同じ値の R_USB_PipeWrite 関数を再度コールする場合は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event: USB_STS_WRITE_COMPLETE)を確認した後で R_USB_PipeWrite 関数をコールしてください。(RTOS の場合)
11. CDC/HID クラスの Bulk/Interrupt 転送を行う場合は、R_USB_Write 関数を使用し、本 API は使用しないでください。また、Host Mass Storage クラスの MSC デバイスに対するデータアクセスを行う場合、FAT(File Allocation Table)の API を使用し、本 API は使用しないでください。
12. usb_ctrl_t 構造体のメンバ type に対する指定は行わないでください。メンバ type にデバイスクラス種別等を指定しても、その指定は無視されます。
13. Control 転送用のデータ転送を行う場合は、R_USB_Write 関数を使用し、本 API は使用しないでください。
14. 本 API を使用する場合、r_usb_basic_config.h ファイル内の USB_CFG_HVND_USE / USB_CFG_PVND_CFG のいずれかの定義を有効にしてください。これらの定義が有効になっていない状態で本 API を使用した場合、戻り値に USB_ERR_NG が返されます。USB_CFG_HVND_USE / USB_CFG_PVND_CFG 定義については、「8. **コンフィグレーション (r_usb_basic_config.h)**」を参照してください。
15. USB デバイスが CONFIGURED 状態の場合に、本 API をコールすることができます。CONFIGURED 以外の状態で本 API をコールすると戻り値に USB_ERR_NG が返されます。
16. 以下の関数内で本 API をコールしないでください。(RTOS のみ)
    (1). 割り込み関数
    (2). R_USB_Callback 関数で登録されたコールバック関数

使用例

1. Non-OS

```
void       usb_application( void )
{
       usb_ctrl_t ctrl;
                    :
       while (1)
       {
            switch (R_USB_GetEvent(&ctrl))
            {
                                :
                 case USB_STS_READ_COMPLETE:
                                :
                      ctrl.moudle = USB_IP0;
                      ctrl.pipe = USB_PIPE2;
                      R_USB_PipeWrite(&ctrl, g_buf, size);
                                :
                 break;
                 case USB_STS_WRITE_COMPLETE:
                                :
                 break;
                                :
            }
       }
}
```

2. RTOS

```
/* Callback Function */
void usb_apl_callback (usb_ctrl_t *p_ctrl, usb_hdl_t hdl, uint8_t is_request)
{
      USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}

/* Application Task */
void usb_application_task( void )
{
    usb_ctrl_t ctrl;
    usb_ctrl_t *p_mess;
              :
    while(1)
    {
          USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
          ctrl = *p_mess;
          switch (ctrl.event)
          {
                              :
              case USB_STS_READ_COMPLETE:
                              :
                    ctrl.moudle = USB_IP0;
                    ctrl.pipe = USB_PIPE2;
                    R_USB_PipeWrite(&ctrl, g_buf, size);
                              :
              break;
              case USB_STS_WRITE_COMPLETE:
                              :
              break;
                              :
           }
      }
}
```

4.15 R_USB_PipeStop

指定 PIPE に対するデータリード/データライト停止

形式

usb_err_t R_USB_PipeStop(usb_ctrl_t *p_ctrl)

引数

p_ctrl	usb_ctrl_t 構造体領域へのポインタ

戻り値

USB_SUCCESS	正常終了 (停止要求完了)
USB_ERR_PARA	パラメータエラー
USB_ERR_NG	その他のエラー

解説

データリード/データライトの停止処理を行います。

リエントラント

1. Non-OS の場合

   本 API は異なる USB モジュールに対して再入可能（リエントラント）です。

2. RTOS の場合

   本 API は再入可能（リエントラント）です。

補足

1. 本 API をコールする前に usb_ctrl_t 構造体のメンバ pipe に対し PIPE 番号(USB_PIPE1 から USB_PIPE9)を指定してください。なお、USB Host モードの場合は、アクセスする USB デバイスを識別するためメンバ module に USB モジュール番号(USB_IP0/USB_IP1)を、メンバ address にデバイスアドレスを指定してください。なお、メンバ module に対し USB_IP0/USB_IP1 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。USB Peripheral モードの場合は、メンバ address および module に対する指定は不要です。指定した場合、これらの指定は無視されます。
2. ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合、メンバ module に対し USB_IP1 を指定しないでください。USB_IP1 を指定した場合は、戻り値に USB_ERR_PARA が返されます。
3. usb_ctrl_t 構造体のメンバ pipe に対し、USB_PIPE1 から USB_PIPE9 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。
4. USB Host モード時、データリード/ライト要求を停止できなかったときは、戻り値に USB_ERR_NG が返されます。
5. データリード/データライト停止が完了した後で R_USB_GetEvent 関数をコールすると戻り値 USB_STS_READ_COMPLETE/USB_STS_WRITE_COMPLETE が返されます。(Non-OS の場合)
6. USB ドライバはデータリードまたはデータライト停止が完了すると USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event)に USB_STS_READ_COMPLETE または USB_STS_WRITE_COMPLETE をセットします。(RTOS の場合)
7. usb_ctrl_t 構造体のメンバ type に対する指定は行わないでください。メンバ type にデバイスクラス種別等を指定しても、その指定は無視されます。
8. 本 API を使用する場合、r_usb_basic_config.h ファイル内の USB_CFG_HVND_USE / USB_CFG_PVND_CFG のいずれかの定義を有効にしてください。これらの定義が有効になっていない状態で本 API を使用した場合、戻り値に USB_ERR_NG が返されます。USB_CFG_HVND_USE /

USB_CFG_PVND_CFG 定義については、「8. **コンフィグレーション (r_usb_basic_config.h)**」を参照してください。

9. USB デバイスが CONFIGURED 状態の場合に、本 API をコールすることができます。CONFIGURED 以外の状態で本 API をコールすると戻り値に USB_ERR_NG が返されます。
10. 以下の関数内で本 API をコールしないでください。

(1). 割り込み関数

(2). R_USB_Callback 関数で登録されたコールバック関数

使用例

1. Non-OS

```
void      usb_application( void )
{
      usb_ctrl_t ctrl;
                        :
      while (1)
      {
            switch (R_USB_GetEvent(&ctrl))
            {
                        :
                  case USB_STS_DETACH:
                        :
                        ctrl.module = USB_IP0;
                        ctrl.pipe = USB_PIPE1;
                        R_USB_PipeStop( &ctrl );
                        :
                  break;
                        :
            }
      }
}
```

2. RTOS

```
/* Callback Function */
void usb_apl_callback (usb_ctrl_t *p_ctrl, usb_hdl_t hdl, uint8_t is_request)
{
      USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}
/* Application Task */
void          usb_application_task( void )
{
     usb_ctrl_t ctrl;
     usb_ctrl_t *p_mess;
               :
     while(1)
     {
          USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
          ctrl = *p_mess;
          switch (ctrl.event)
          {
                         :
               case USB_STS_DETACH:
                         :
                    ctrl.module = USB_IP0;
                    ctrl.pipe = USB_PIPE1;
                    R_USB_PipeStop( &ctrl );
                         :
               break;
                         :
           }
      }
}
```

4.16 R_USB_GetUsePipe

使用する PIPE 番号をビットマップにより取得

形式

usb_err_t R_USB_GetUsePipe(usb_ctrl_t *p_ctrl, uint16_t *p_pipe)

引数

p_ctrl	usb_ctrl_t 構造体領域へのポインタ
p_pipe	使用する PIPE 番号(ビットマップ情報)を格納する領域へのポインタ

戻り値

USB_SUCCESS	正常終了
USB_ERR_PARA	パラメータエラー
USB_ERR_NG	その他のエラー

解説

使用する PIPE 番号(初期化が完了している PIPE 番号)をビットマップ情報により取得します。ビットマップ情報は、引数(p_pipe)が示す領域に格納されます。usb_ctrl_t 構造体に指定された情報(メンバ module およびメンバ address)をもとに該当する USB デバイスの PIPE 情報を取得します。

ビットマップ情報が示す PIPE 番号とビット位置の関係は以下の通りです。

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
--	--	--	--	--	--	PIPE9	PIPE8	PIPE7	PIPE6	PIPE5	PIPE4	PIPE3	PIPE2	PIPE1	PIPE0
0	0	0	0	0	0	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	1

0:Not used, 1: Used

例えば、PIPE1、PIPE2 および PIPE8 の PIPE を使用している場合は、引数(p_pipe)が示す領域には、数値"0x0107"がセットされます。

リエントラント

本 API は再入可能（リエントラント）です。

補足

1. USB Host モードの場合、本 API をコールする前に usb_ctrl_t 構造体のメンバ address およびメンバ module に対し、Pipe 情報を取得したい USB デバイスのデバイスアドレスおよびその USB デバイスが接続された USB モジュール番号(USB_IP0/USB_IP1)を指定してください。なお、メンバ module に対し USB_IP0/USB_IP1 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。
2. ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合、メンバ module に対し USB_IP1 を指定しないでください。USB_IP1 を指定した場合は、戻り値に USB_ERR_PARA が返されます。
3. USB Peripheral モードの場合は、第一引数 p_ctrl に対し USB_NULL を指定してください。なお、USB Host モードの場合、第一引数 p_ctrl に対し USB_NULL を指定しないでください。USB_NULL を指定した場合、戻り値に USB_ERR_PARA が返されます。
4. ビットマップ情報の b0(PIPE0)には、必ず"1"がセットされます。
5. USB デバイスが CONFIGURED 状態の場合に、本 API をコールすることができます。CONFIGURED 以外の状態で本 API をコールすると戻り値に USB_ERR_NG が返されます。

使用例

1. **USB Host** モードの場合

```
void        usb_application( void )
{
    uint16_t usepipe;
    usb_ctrl_t ctrl;
                    :
    while (1)
    {
                    :
            case USB_STS_CONFIGURED:
                    :
                ctrl.module = USB_IP0;
                ctrl.address = adr;
                R_USB_GetUsePipe(&ctrl, &usepipe);
                    :
            break;
                    :
    }
}
```

2. **USB Pripheral** モードの場合

```
void        usb_application( void )
{
    uint16_t usepipe;
    usb_ctrl_t ctrl;

    while (1)
    {
                    :
            case USB_STS_CONFIGURED:
                    :
                R_USB_GetUsePipe((usb_ctrl_t *)USB_NULL, &usepipe);
                    :
            break;
                    :
    }
}
```

4.17 R_USB_GetPipeInfo

指定 PIPE の PIPE 情報を取得

形式

usb_err_t R_USB_GetPipeInfo(usb_ctrl_t *p_ctrl, usb_pipe_t *p_info)

引数

p_ctrl	usb_ctrl_t 構造体領域へのポインタ
p_info	usb_pipe_t 構造体領域へのポインタ

戻り値

USB_SUCCESS	正常終了
USB_ERR_PARA	パラメータエラー
USB_ERR_NG	その他のエラー

解説

引数(p_ctrl)のメンバ pipe に指定された PIPE の Endpoint 番号、転送タイプ、転送方向およびマックスパケットサイズの PIPE 情報を取得します。取得した PIPE 情報は、引数(p_info)が示す領域に格納されます。

リエントラント

本 API は再入可能（リエントラント）です。

補足

1. 本 API をコールする前に usb_ctrl_t 構造体のメンバ pipe に対し PIPE 番号(USB_PIPE1 から USB_PIPE9)を指定してください。メンバ pipe に対し USB_PIPE1 から USB_PIPE9 以外の値を指定した場合、戻り値に USB_ERR_PARA が返されます。
2. USB Host モードの場合、本 API をコールする前に usb_ctrl_t 構造体のメンバ address およびメンバ module に対し、Pipe 情報を取得したい USB デバイスのデバイスアドレスおよびその USB デバイスが接続された USB モジュール番号(USB_IP0/USB_IP1)を指定してください。なお、メンバ module に対し USB_IP0/USB_IP1 以外を指定した場合、戻り値に USB_ERR_PARA が返されます。
3. ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合、メンバ module に対し USB_IP1 を指定しないでください。USB_IP1 を指定した場合は、戻り値に USB_ERR_PARA が返されます。
4. いずれかの引数に対し 0(zero)を指定した場合、戻り値に USB_ERR_PARA が返されます。
5. USB Peripheral モードの場合は、メンバ address および module に対する設定は必要ありません。
6. usb_pipe_t 構造体については、「**9.5. usb_pipe_t構造体**」を参照してください。
7. USB デバイスが CONFIGURED 状態の場合に、本 API をコールすることができます。CONFIGURED 以外の状態で本 API をコールすると戻り値に USB_ERR_NG が返されます。

使用例

```
void usb_host_application( void )
{
    usb_pipe_t info;
    usb_ctrl_t ctrl;
                    :
    while (1)
    {
                    :
            case USB_STS_CONFIGURED:
                    :
                ctrl.pipe = USB_PIPE3;
                ctrl.module = USB_IP1;
                ctrl.address= address;
                R_USB_GetPipeInfo( &ctrl, &info );
                    :
            break;
                    :
    }
}
```

5. コールバック関数 (RTOS のみ)

USB ドライバは、USB イベントが完了するとコールバック関数をコールします。コールバック関数は、ユーザアプリケーションプログラムとしてユーザが作成し、R_USB_Callback 関数を使って USB ドライバにそのコールバック関数を登録する必要があります。

USB ドライバに登録するコールバック関数は、以下に示す引数および戻り値をサポートしてください。

引数	:	usb_ctrl_t	*p_ctrl	// usb_ctrl_t 構造体領域へのポインタ
	:	usb_hdl_t	hdl	// USB イベントが完了したタスクハンドル
	:	uint8_t	is_request	// クラスリクエスト受信フラグ(USB_ON / USB_OFF)
戻り値	:	void		// なし

[Note]

(4). 引数(p_ctrl)には、USB 完了イベントのほか、そのイベントに応じた各種情報が USB ドライバによって設定されています。必ず、RTOS の API 等を使ってアプリケーション用タスクへ当該引数の情報を通知してください。

(5). 引数(p_ctrl)のメンバ event が以下の場合、その event に関連する API をコールしたアプリケーションタスクのタスクハンドルが、引数(hdl)に設定されます。その他の場合、引数(hdl)には USB_NULL が設定されます。

g. USB_STS_READ_COMPLETE
h. USB_STS_WRITE_COMPLETE
i. USB_STS_REQUEST_COMPLETE (Note a)
j. USB_STS_SUSPEND (Note b)
k. USB_STS_RESUME (Note b)
l. USB_STS_MSC_CMD_COMPLETE

Note:

c. USB Peripheral モードで、ノーデータステータスステージをサポートするリクエストを受信した時、引数(hdl)には USB_NULL が設定されます。

d. USB Peripheral モードの場合、引数(hdl)には USB_NULL が設定されます。

(6). USB Peripheral モードで、クラスリクエストを受信した場合、引数(is_request)には USB_ON が設定されます。その他の場合、USB_OFF が設定されます。引数(is_request)が USB_ON の時、引数(p_ctrl)のメンバ setup には、そのクラスリクエストに関する情報が設定されています。

記述例)

```
void usb_apl_callback (usb_ctrl_t *p_ctrl, usb_hdl_t hdl, uint8_t is_request)
{
    /* RTOS の API を使って USB イベントの情報をアプリケーションタスクに通知 */
    USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}
```

6. R_USB_GetEvent 関数の戻り値 / USB 完了イベントの取得

(1). Non-OS

R_USB_GetEvent 関数の戻り値は以下のとおりです。アプリケーションプログラムでは R_USB_GetEvent 関数からの戻り値をトリガにして、各戻り値に対応したプログラムを記述してください。

(2). RTOS

USB イベントが完了すると USB ドライバから R_USB_Callback 関数により登録されたコールバック関数がコールされます。完了した USB イベント情報は、このコールバック関数の引数(usb_ctrl_t 構造体へのポインタ)のメンバ event に設定されています。アプリケーションプログラムでは、必ず、このコールバック関数を定義し、そのコールバック関数内で RTOS がサポートする API 等を使ってアプリケーションタスクに対し完了した USB イベントを通知してください。

戻り値	内容	Host	Peri
USB_STS_DEFAULT	USB デバイスが DEFAULT ステートに遷移した。	×	○
USB_STS_CONFIGURED	USB デバイスが CONFIGURED ステートに遷移した。	○	○
USB_STS_SUSPEND	USB デバイスが SUSPEND 状態に遷移した。	×	○
USB_STS_RESUME	USB デバイスが SUSPEND 状態から復帰した	○	○
USB_STS_DETACH	USB デバイスが USB Host から DETACH された。	○	○
USB_STS_REQUEST	USB デバイスが USB リクエスト(Setup)を受信した。	×	○
USB_STS_REQUEST_COMPLETE	USB リクエストデータの送受信が完了し、ステータスステージに遷移した。	○	○
USB_STS_READ_COMPLETE	USB データリード処理が完了した。	○	○
USB_STS_WRITE_COMPLETE	USB データライト処理が完了した。	○	○
USB_STS_BC	Battery Charging 機能をサポートした USB デバイスのアタッチを検出した。	○	×
USB_STS_OVERCURRENT	オーバーカレントを検出した。	○	×
USB_STS_NOT_SUPPORT	サポート対象外の USB デバイスが接続された。	○	×
USB_STS_NONE (Non-OS のみ)	USB 関連のイベントが無い状態。	○	○

○:サポート ×:非サポート

6.1 USB_STS_DEFAULT

USB デバイスのデバイスステートが Default ステートに遷移した状態を表します。

6.2 USB_STS_CONFIGURED

USB デバイスのデバイスステートが Configured ステートに遷移した状態を表します。USB Host モードの場合、usb_ctrl_t 構造体の以下メンバにも情報が設定されます。

module : Configured 状態に遷移した USB モジュールのモジュール番号 (USB Host モードのみ)
type : Configured 状態に遷移した USB デバイスのデバイスクラス種別 (USB Host モードのみ)
address : Configured 状態に遷移した USB デバイスのデバイスアドレス (USB Host モードのみ)

6.3 USB_STS_SUSPEND

USB デバイスのデバイスステートが SUSPEND ステートに遷移した状態を表します。

6.4 USB_STS_RESUME

SUSPEND 状態にある USB デバイスが RESUME 信号により SUSPEND 状態から復帰した状態を表します。

[Note]

USB Host モードの場合、HID デバイスからの RemoteWakeUp 信号により復帰した状態を表します。

6.5 USB_STS_DETACH

USB Host から USB デバイスが Detach された状態を表します。USB Host モードの場合、usb_ctrl_t 構造体の以下メンバにも情報が設定されます。

module : デタッチされた USB モジュールのモジュール番号(USB Host モードのみ)
address : デタッチした USB デバイスのデバイスアドレス (USB Host モードのみ)

6.6 USB_STS_REQUEST

USB デバイスが USB リクエスト(Setup)を受信した状態を表します。usb_ctrl_t 構造体の以下のメンバにも情報が設定されます。

setup : 受信した USB リクエスト情報(8 バイト)

[Note]

1. ノーデータコントロールステータスステージをサポートするリクエストを受信した状態で、R_USB_GetEvent 関数をコールしても戻り値 USB_STS_REQUEST は返されません。この場合の戻り値には USB_STS_REQUEST_COMPLETE が返されます。(Non-OS の場合)
2. ノーデータコントロールステータスステージをサポートするリクエストを受信した場合、メンバ event には、USB_STS_REQUEST はセットされず、USB_STS_REQUEST_COMPLETE がセットされます。(RTOS の場合)
3. メンバ setup に格納される USB リクエスト情報(8 バイト)については、「9.2. usb_setup_t構造体」を参照してください。

6.7 USB_STS_REQUEST_COMPLETE

コントロール転送のステータスステージが完了し、アイドルステージに遷移した状態を表します。このほか、usb_ctrl_t 構造体の以下メンバにも情報が設定されます。なお、usb_ctrl_t 構造体のメンバ setup には、直前のリクエスト情報が設定されています。

module : リクエストが完了した USB モジュールのモジュール番号(USB Host モードのみ)
address : リクエストが完了した USB デバイスのデバイスアドレス(USB Host モードのみ)
status : USB_ACK / USB_STALL のいずれかを設定

[Note]

ノーデータコントロールステータスステージをサポートするリクエストの場合、usb_ctrl_t 構造体のメンバ setup に受信した USB リクエスト情報(8 バイト)が設定されます。USB リクエスト情報(8 バイト)については、「**9.2 usb_setup_t構造体**」を参照してください。

6.8 USB_STS_READ_COMPLETE

R_USB_Read/R_USB_PipeRead 関数によるデータリードが完了した状態を表します。このほか、usb_ctrl_t 構造体の以下メンバにも情報が設定されます。

module : データリードが完了した USB モジュール番号 (USB Host モードのみ)
address : データリードが完了した USB デバイスのデバイスアドレス (USB Host モードのみ)
type : データリードが完了したデバイスクラス種別(R_USB_Read 関数使用時のみ設定)
size : リードしたデータサイズ
pipe : データリードが完了した PIPE 番号
status : リード完了エラー情報

[Note]

1. USB Host モードの場合、メンバ address にはデータリードが完了した USB デバイスのデバイスアドレスが設定され、メンバ module にはその USB デバイスが接続されている USB モジュール番号(USB_IP0 / USB_IP1)が設定されます。

2. R_USB_PipeRead 関数の場合は、メンバ pipe にデータリードが完了した PIPE 番号(USB_PIPE1 から USB_PIPE9)が設定されます。なお、R_USB_Read 関数の場合、メンバ pipe には USB_NULL が設定されます。
3. デバイスクラス種別については、「**7. デバイスクラス種別**」を参照してください。
4. メンバ status には、リード完了エラー情報が設定されます。このメンバに設定されるエラー情報は以下の通りです。

USB_SUCCESS	:	データリード正常終了
USB_ERR_OVER	:	データ受信オーバー
USB_ERR_SHORT	:	データ受信ショート
USB_ERR_NG	:	データ受信失敗

(1). 受信要求サイズが MaxPacketSize×n 未満であるにも関わらず MaxPacketSize×n バイトのデータを受信した場合に USB_ERR_OVER が設定されます。

例えば、MaxPacketSize が 64 バイト、受信要求サイズに 510 バイト(MaxPacketSize×n 未満)を指定し、実際の受信データサイズが 512 バイト(MaxPacketSize×n)の場合に、USB_ERR_OVER が設定されます。

(2). 受信要求サイズが MaxPacketSize×n 未満で、その受信要求サイズ未満のデータを受信した場合に USB_ERR_SHORT が設定されます。

例えば、MaxPacketSize が 64 バイト、受信要求サイズに 510 バイトを指定し、実際の受信データサイズが 509 バイトの場合に USB_ERR_SHORT が設定されます。

(3). USB_SUCCESS または USB_ERR_SHORT の場合、メンバ size にリードしたデータサイズが設定されます。

6.9 USB_STS_WRITE_COMPLETE

R_USB_Write 関数によるデータライトが完了した状態を表します。このほか、usb_ctrl_t 構造体の以下メンバにも情報が設定されます。

module	:	データライトが完了した USB モジュール番号 (USB Host モードのみ)
address	:	データライトが完了した USB デバイスのデバイスアドレス (USB Host モードのみ)
type	:	データライトが完了したデバイスクラス種別(R_USB_Write 関数使用時のみ設定)
pipe	:	データライトが完了した PIPE 番号
status	:	ライト完了エラー情報

[Note]

1. R_USB_Write 関数の場合は、usb_ctrl_t 構造体のメンバ type にクラスタイプ種別が設定され、メンバ pipe には USB_NULL が設定されます。
2. R_USB_PipeWrite 関数の場合は、メンバ pipe にデータライトが完了した PIPE 番号(USB_PIPE1 から USB_PIPE9)が設定されます。なお、R_USB_Write 関数の場合、メンバ pipe には USB_NULL が設定されます。
3. デバイスクラス種別については、「**7. デバイスクラス種別**」を参照してください。
4. メンバ status には、ライト完了エラー情報が設定されます。このメンバに設定されるエラー情報は以下の通りです。

USB_SUCCESS	:	データライト正常終了
USB_ERR_NG	:	データ送信失敗

6.10 USB_STS_BC

Battery Charging 機能をサポートする USB Host/USB デバイスが接続された状態を表します。このほか、usb_ctrl_t 構造体の以下メンバにも情報が設定されます。

module : Battery Charging 機能をサポートする USB デバイスが接続された USB モジュール番号

(USB Host モードのみ)

6.11 USB_STS_OVERCURRENT

USB Host モード時、Overcurrent を検出した状態を表します。このほか、usb_ctrl_t 構造体の以下メンバにも情報が設定されます。

module : Overcurrent を検出した USB モジュール番号(USB_IP0 / USB_IP1)

6.12 USB_STS_NOT_SUPPORT

USB Host モード時、サポート対象外の USB デバイスが接続された状態を表します。

6.13 USB_STS_NONE (Non-OS のみ)

USB 関連のイベントが無い状態を表します。usb_ctrl_t 構造体の以下メンバにも情報が設定されます。

status : USB デバイスのステータス

7. デバイスクラス種別

usb_ctrl_t 構造体や usb_info_t 構造体のメンバ type に指定するデバイスクラス種別は以下のとおりです。お客様のシステムでサポートしているデバイスクラスを指定してください。

デバイスクラス種別	内容
USB_HCDC	Host Communication Device Class
USB_HCDCC	Host Communication Device Class (Control Class)
USB_HHID	Host Human Interface Device Class
USB_HMSC	Host Mass Storage Device Class
USB_PCDC	Peripheral Communication Device Class
USB_PCDCC	Peripheral Communication Device Class (Control Class)
USB_PHID	Peripheral Human Interface Device Class
USB_PMSC	Peripheral Mass Storage Device Class
USB_HVND	Host Vendor Class
USB_PVND	Peripheral Vendor Class

[Note]

1. Host Communication Device Class の場合で、Bulk 転送によるデータ通信を行う場合は、usb_ctrl_t 構造体のメンバ type に対し USB_HCDC を指定し、Interrupt 転送によるデータ通信を行う場合は、メンバ type に対し USB_HCDCC を指定してください。
2. Peripheral Communication Device Class の場合で、Bulk 転送によるデータ通信を行う場合は、usb_ctrl_t 構造体のメンバ type に対し USB_PCDC を指定し、Interrupt 転送によるデータ通信を行う場合は、メンバ type に対し USB_PCDCC を指定してください。

アプリケーションプログラムでは、USB_HMSC、USB_PMSC、USB_HVND および USB_PVND を usb_ctrl_t 構造体のメンバ(type)には指定しないでください。

8. コンフィグレーション (r_usb_basic_config.h)

8.1 USB Host/USB Peripheral モードの共通設定

USB Host / USB Peripheral モードのいずれの場合も下記の定義に対する指定を行ってください。

1. USB 動作モード設定

   USB モジュールの動作モード(Host / Peripheral)を USB_CFG_MODE 定義に対し指定してください。

   (1). USB Host モードの場合

   USB_CFG_MODE 定義に対し USB_CFG_HOST を指定してください。

```
#define     USB_CFG_MODE        USB_CFG_HOST
```

   (2). USB Peripheral モードの場合

   USB_CFG_MODE 定義に対し USB_CFG_PERI を指定してください。

```
#define     USB_CFG_MODE        USB_CFG_PERI
```

2. 引数チェック設定

   「**4. API**」に記載された各 API に対する引数チェックの実施/非実施を指定してください。

```
#define     USB_CFG_PARAM_CHECKING  USB_CFG_ENABLE    // 引数チェック実施
#define     USB_CFG_PARAM_CHECKING  USB_CFG_DISABLE   // 引数チェック非実施
```

3. デバイスクラス設定

   以下の定義のうち、お客様が使用する USB ドライバの定義を有効にしてください。なお、複数の定義を有効にすることはできません。有効にすることができる定義数は一つのみです。

```
#define     USB_CFG_HCDC_USE        // Host Communication Device Class
#define     USB_CFG_HHID_USE        // Host Human Interface Device Class
#define     USB_CFG_HMSC_USE        // Host Mass Storage Class
#define     USB_CFG_HVNDR_USE       // Host Vendor Class
#define     USB_CFG_PCDC_USE        // Peripheral Communication Device Class
#define     USB_CFG_PHID_USE        // Peripheral Human Interface Device Class
#define     USB_CFG_PMSC_USE        // Peripheral Mass Storage Class
#define     USB_CFG_PVNDR_USE       // Peripheral Vendor Class
```

4. DTC 使用設定

   DTC の使用/非使用を指定してください。

```
#define     USB_CFG_DTC         USB_CFG_ENABLE    // DTC 使用
#define     USB_CFG_DTC         USB_CFG_DISABLE   // DTC 非使用
```

   [Note]

   USB_CFG_DTC 定義に対し USB_CFG_ENABLE を指定した場合、下記5の USB_CFG_DMA 定義に対しては必ず USB_CFG_DISABLE を指定してください。

5. DMA 使用設定

   DMA の使用/非使用を指定してください。

```
#define     USB_CFG_DMA         USB_CFG_ENABLE    // DMA 使用
#define     USB_CFG_DMA         USB_CFG_DISABLE   // DMA 非使用
```

   [Note]

   (1). USB_CFG_DMA 定義に対し USB_CFG_ENABLE を指定した場合、上記4の USB_CFG_DTC 定義に対しては必ず USB_CFG_DISABLE を指定してください。

   (2). USB_CFG_DMA 定義に対し USB_CFG_ENABLE を指定した場合、下記6の定義において DMA Channel 番号を指定してください。

6. DMA Channel 設定

上記5の設定で USB_CFG_ENABLE を指定した場合、使用する DMA Channel 番号を指定してください。

```
#define     USB_CFG_USB0_DMA_TX     DMA Channel 番号    // USB0 モジュール用送信設定
#define     USB_CFG_USB0_DMA_RX     DMA Channel 番号    // USB0 モジュール用受信設定
#define     USB_CFG_USB1_DMA_TX     DMA Channel 番号    // USB1 モジュール用送信設定
#define     USB_CFG_USB1_DMA_RX     DMA Channel 番号    // USB1 モジュール用受信設定
```

[Note]

(1). DMA Channel 番号には、USB_CFG_CH0 から USB_CFG_CH7 を指定してください。なお、同じ DMA Channel 番号は指定しないでください。

(2). DMA 転送を使用しない場合は、DMA Channel 番号に USB_CFG_NOUSE を指定してください。

(3). USB Host Mass Storage Class の場合、必ず DMA 送受信に異なる DMA Channel 番号を指定してください。

指定例を以下に示します。

a. USB0 モジュールを使って USB データ送受信に DMA を使用する場合

```
#define     USB_CFG_USB0_DMA_TX     USB_CFG_CH0
#define     USB_CFG_USB0_DMA_RX     USB_CFG_CH3
```

[Note]

USB PIPE は、USB PIPE1 と USB PIPE2 を使用してください。

b. USB1 モジュールを使ってデータ送信用に DMA を使用し、データ受信用には DMA を使用しない場合

```
#define     USB_CFG_USB1_DMA_TX     USB_CFG_CH1
```

[Note]

送信用 USB PIPE(DMA 転送用)には、USB PIPE1/USB PIPE2 のいずれかを使用し、受信用 USB PIPE には USB PIPE3/USB PIPE4/USB PIPE5 のいずれかを使用してください。

c. USB0 モジュールを使ってデータ送信用に DMA を使用し、データ受信用には DMA を使用しない場合、および、USB1 モジュールを使ってデータ受信用に DMA を使用し、データ送信用には DMA を使用しない場合

```
#define     USB_CFG_USB0_DMA_TX     USB_CFG_CH1
#define     USB_CFG_USB1_DMA_RX     USB_CFG_CH2
```

[Note]

USB0 モジュールの送信用 USB PIPE(DMA 転送用)には、USB PIPE1/USB PIPE2 のいずれかを使用し、受信用 USB PIPE には USB PIPE3/USB PIPE4/USB PIPE5 のいずれかを使用してください。USB1 モジュールの受信用 USB PIPE(DMA 転送用)には、USB PIPE1/USB PIPE2 のいずれかを使用し、送信用 USB PIPE には USB PIPE3/USB PIPE4/USB PIPE5 のいずれかを使用してください。

7. Battery Charging(BC)機能設定

以下の定義に対し Battery Charging 機能の有効/無効を設定してください。Battery Charging 機能を使用する場合は、以下の定義に対し USB_CFG_ENABLE を指定してください。

```
#define     USB_CFG_BC          USB_CFG_ENABLE      // BC 機能を使用する
#define     USB_CFG_BC          USB_CFG_DISABLE     // BC 機能を使用しない。
```

[Note]

USBAa/USBA モジュール以外の USB モジュールの場合、この定義は無視されます。

8. PLL クロック周波数設定

以下の定義に対し PLL クロックソース周波数を指定してください。

```
#define     USB_CFG_CLKSEL     USB_CFG_24MHZ     // 24MHz 設定
#define     USB_CFG_CLKSEL     USB_CFG_20MHZ     // 20MHz 設定
#define     USB_CFG_CLKSEL     USB_CFG_OTHER     // 24MHz / 20MHz 以外の場合
```

[Note]

(1). USBAa/USBA モジュール以外の USB モジュールの場合、この定義は無視されます。

(2). USBAa/USBA モジュールは、RX71M/RX64M で使用されている USB モジュールです。

(3). XTAL 端子に 24MHz/20MHz 以外のクロックを入力する場合は、USB_CFG_CLKSEL 定義に対し USB_CFG_OTHER を指定してください。なお、USB_CFG_OTHER を指定した場合、USBAa/USBA モジュールは Classic(CL) only モードで動作します。CL only モードについては、RX71M/RX64M のハードウェアマニュアルを参照してください。

9. CPU バスウェイト設定

USBAa/USBA モジュール内にある BUSWAIT レジスタに設定する数値を USB_CFG_BUSWAIT に対し指定してください。

```
#define     USB_CFG_BUSWAIT     7          // 7 ウェイト設定
```

[Note]

(1). USB_CFG_BUSWAIT に指定する数値の算出については、RX71M/RX64M のハードウェアマニュアル内の BUSWAIT レジスタの章を参照してください。

(2). USBAa/USBA モジュール以外の USB モジュールの場合、この定義は無視されます。

(3). USBAa/USBA モジュールは、RX71M/RX64M で使用されている USB モジュールです。

10. 割り込み優先レベル設定

USB に関連する割り込みの割り込み優先レベルを USB_CFG_INTERRUPT_PRIORITY に対し指定してください。

```
#define     USB_CFG_INTERRUPT_PRIORITY     3     // 1(low) – 15(high)
```

11. マルチタスク設定 (RTOS のみ)

「4. **API**」に記載された API を複数のタスクからコールするかどうかを指定します。複数タスクからコールする場合は、USB_CFG_ENABLE を指定してください。

```
#define     USB_CFG_MULTIPLE_TASKS     USB_CFG_ENABLE     //  複数タスクからコールする
#define     USB_CFG_MULTIPLE_TASKS     USB_CFG_DISABLE    //  複数タスクからコールしない
```

[Note]

(1). Non-OS の場合は、この定義は無視されます。

8.2 USB Host モードの場合

USB モジュールを USB Host として動作させる場合は、ご使用のシステムに合わせて以下の定義を指定してください。

1. USB Host 用電源 IC 設定

ご使用の USB Host 用電源 IC の VBUS 出力が Low アサートか High アサートかを設定してください。Low アサートの場合は、以下の定義に対し USB_CFG_LOW を指定し、High アサートの場合は、以下の定義に対し USB_CFG_HIGH を指定してください。

```
#define     USB_CFG_VBUS     USB_CFG_HIGH     // High アサート
```

```
#define     USB_CFG_VBUS        USB_CFG_LOW     // Low アサート
```

2. Battery Charging(BC)機能使用時 USB ポート動作設定

以下の定義に対し、Dedicated Charging Port(DCP)の有効/無効を設定してください。BC 機能を Dedicated Charging Port(DCP)として機能させる場合､以下の定義に対し USB_CFG_ENABLE を指定してください。USB_CFG_DISABLE を指定した場合は、BC 機能は Charging Downstream Port(CDP)として機能します。

```
#define     USB_CFG_DCP         USB_CFG_ENABLE  // DCP 有効
#define     USB_CFG_DCP         USB_CFG_DISABLE // DCP 無効
```

[Note]

この定義に対し USB_CFG_ENABLE を指定する場合は、上記の USB_CFG_BC 定義に対して USB_CFG_ENABLE を指定してください。

3. Compliance Test モード設定

以下の定義に対し USB Embedded Host の Compliance Test 対応の有効/無効を指定してください。Compliance Test 実行時は、以下の定義に対し USB_CFG_ENABLE を指定してください。Compliance Test 実行時以外は、以下の定義に対し USB_CFG_DISABLE を指定してください。

```
#define     USB_CFG_COMPLIANCE  USB_CFG_ENABLE  // Compliance Test 対応
#define     USB_CFG_COMPLIANCE  USB_CFG_DISABLE // Compliance Test 非対応
```

4. Target Peripheral List (TPL)設定

以下の定義に対し接続する USB デバイス数および USB デバイスの VID, PID のセットを指定してください。TPL の設定方法については、「**3.6. ターゲットペリフェラルリスト(TPL)の設定方法**」を参照してください。

```
#define     USB_CFG_TPLCNT      接続する USB デバイス数を設定
#define     USB_CFG_TPL         接続する USB デバイスの VID と PID のセットを設定
```

5. USB Hub 用 Target Peripheral List(TPL)設定

以下の定義に対し必要に応じて接続する USB Hub 数および USB Hub の VID, PID のセットを指定してください。TPL の設定方法については、「**3.6. ターゲットペリフェラルリスト(TPL)の設定方法**」を参照してください。

```
#define     USB_CFG_HUB_TPLCNT      接続する USB Hub 数を設定
#define     USB_CFG_HUB_TPL         接続する USB Hub の VID と PID のセットを設定
```

6. Hi-speed Embedded Host Electrical Test 設定

以下の定義に対し Hi-speed Embedded Host Electrical Test 対応の有効/無効を指定してください。Hi-speed Embedded Host Electrical Test を実行する時は以下の定義に対し USB_CFG_ENABLE を指定してください。

```
#define     USB_CFG_ELECTRICAL  USB_CFG_ENABLE  // HS Electrical Test 対応
#define     USB_CFG_ELECTRICAL  USB_CFG_DISABLE // HS Electrical Test 非対応
```

[Note]

(1). この定義に対し USB_CFG_ENABLE を指定する場合は、上記3の USB_CFG_COMPLIANCE 定義に対して USB_CFG_ENABLE を設定してください。

(2). USBAa モジュール以外の USB モジュールの場合、この定義は無視されます。

8.3 USB Peripheral モードの場合

USB モジュールを USB Peripheral として動作させる場合は、ご使用のシステムに合わせて以下の定義を設定してください。

1. USB モジュール選択設定

USB_CFG_USE_USBIP 定義に対し、使用する USB モジュール番号を指定してください。USB0 モジュールを使用する場合は USB_CFG_USE_USBIP 定義に対し USB_CFG_IP0 を指定し、USB1 モジュールを使用する場合は USB_CFG_IP1 を指定してください。

```
#define     USB_CFG_USE_USBIP     USB_CFG_IP0     // USB0 モジュールを使用
#define     USB_CFG_USE_USBIP     USB_CFG_IP1     // USB1 モジュールを使用
```

[Note]

ご使用の MCU が USB モジュールを 1 つしかサポートしていない場合には、USB_CFG_USE_USBIP 定義に対し、USB_CFG_IP0 を指定してください。

8.4 その他の定義

r_usb_basic_config.h には、上記のほか、下記1から2の定義も記載されています。これらの定義に対しては推奨値が設定されているため、変更の必要が生じた時のみ変更してください。

1. DBLB ビット設定

USB モジュールのパイプコンフィグレーションレジスタ(PIPECFG)に DBLB ビットのセット/クリア指定を以下の定義により行います。

```
#define     USB_CFG_DBLB     USB_CFG_DBLBON      // DBLB ビットをセット
#define     USB_CFG_DBLB     USB_CFG_DBLBOFF     // DBLB ビットをクリア
```

2. CNTMD ビット設定(USBA/USBAa モジュールのみ)

USB モジュールのパイプコンフィグレーションレジスタ(PIPECFG)に CNTMD ビットのセット/クリア指定を以下の定義により行います。

```
#define     USB_CFG_CNTMD     USB_CFG_CNTMDON      // CNTMD ビットをセット
#define     USB_CFG_CNTMD     USB_CFG_CNTMDOFF     // CNTMD ビットをクリア
```

[Note]

(1). 上記の DBLB / CNTMD ビットの設定は、使用するすべてのパイプに対して行われます。したがって、このコンフィグレーションでは、これらのビットに対するパイプ固有の設定を行うことはできません。

(2). パイプコンフィグレーションレジスタ(PIPECFG)の詳細については、MCU のハードウェアマニュアルを参照してください。

(3). SHTNAK ビットは、必ずセットしてください。

9. 構造体

アプリケーションプログラムで使用する構造体について説明します。

9.1 usb_ctrl_t 構造体

usb_ctrl_t 構造体は、USB データ通信等で使用される構造体です。usb_ctrl_t 構造体はTable4-1にある API のうち R_USB_GetVersion を除くすべての API で使用されます。

```
typedef struct usb_ctrl {
    uint8_t         module;         /* Note 1 */
    uint8_t         address;        /* Note 2 */
    uint8_t         pipe;           /* Note 3 */
    uint8_t         type;           /* Note 4 */
    uint8_t         status;         /* Note 5 */
    uint8_t         event;          /* Note 6 */
    uint32_t        size;           /* Note 7 */
    usb_setup_t     setup;          /* Note 8 */
    void            *p_data;        /* Note 9 */
} usb_ctrl_t;
```

[Note]

1. メンバ(module)は、USB モジュール番号を指定するために使用されます。
2. メンバ(address)は、USB デバイスアドレスを指定するために使用されます。
3. メンバ(pipe)は、USB モジュールの PIPE 番号を指定するために使用されます。R_USB_PipeRead 関数や R_USB_PipeWrite 関数を使用する場合の PIPE 番号指定等で使用されます。
4. メンバ(type)は、デバイスクラス種別等を指定するために使用されます。
5. メンバ(status)には、USB デバイスのステートまたは USB リクエストコマンドの結果が格納されます。USB ドライバがこのメンバに対する設定を行いますので、アプリケーションプログラムでは usb_ctrl_t 構造体領域への初期化処理およびベンダクラスリクエストの ACK/STALL 応答処理を除いて、このメンバに対する書き込みは行わないでください。なお、ベンダクラスリクエストの ACK/STALL 応答処理については、「**10.2.5 クラスリクエストに対するACK/STALL応答処理**」を参照してください。
6. メンバ(event)は、USB ドライバが完了イベントを設定するために使用されます。(RTOS のみ)
7. メンバ(size)は、リードしたデータサイズを設定するために使用されます。USB ドライバがこのメンバに対して設定を行いますので、アプリケーションプログラムではこのメンバに対する書き込みは行わないでください。
8. メンバ(setup)は、クラスリクエストに関する情報を設定するために使用されます。
9. メンバ(p_data)は、上記以外の情報を設定するために使用されます。(RTOS のみ)

9.2 usb_setup_t 構造体

usb_setup_t 構造体は、USB クラスリクエストの送受信を行う場合に使用される構造体です。USB デバイスに対しクラスリクエストを送信する場合(USB Host モード時)は、usb_setup_t 構造体の各メンバに対し送信するクラスリクエスト情報を設定し、USB Host からのクラスリクエスト情報を取得する場合(USB Peripheral モード時)は、usb_setup_t 構造体の各メンバを参照します。

```
typedef struct usb_setup {
    uint16_t        type            /* Note 1 */
    uint16_t        value;          /* Note 2 */
    uint16_t        index;          /* Note 3 */
    uint16_t        length;         /* Note 4 */
} usb_setup_t;
```

[Note]

1. USB Host モード時、メンバ(type)に設定した値が USBREQ レジスタに設定され、USB Periphral モード時、USBREQ レジスタの値がメンバ(type)に設定されます。
2. USB Host モード時、メンバ(value)に設定した値が USBVAL レジスタに設定され、USB Periphral モード時、USBVAL レジスタの値がメンバ(value)に設定されます。
3. USB Host モード時、メンバ(index)に設定した値が USBINDX レジスタに設定され、USB Periphral モード時、USBINDX レジスタの値がメンバ(index)に設定されます。
4. USB Host モード時、メンバ(length)に設定した値が USBLENG レジスタに設定され、USB Periphral モード時、USBLENG レジスタの値がメンバ(length)に設定されます。
5. USBREQ, USBVAL, USBINDX および USBLENG レジスタについては MCU のユーザーズマニュアルを参照してください。

9.3 usb_cfg_t 構造体

usb_cfg_t 構造体は、使用する USB モジュールを USB Host として使用するかまたは USB Peripheral として使用するかの設定や USB スピードの設定等の情報を登録するための構造体です。Table4-1にある API のうち R_USB_Open 関数のみで使用されます。

```
typedef struct usb_cfg {
    uint8_t             usb_mode;       /* Note 1 */
    uint8_t             usb_speed;      /* Note 2 */
    usb_descriptor_t    *p_usb_reg;     /* Note 3 */
} usb_cfg_t;
```

[Note]

1. USB モジュールを USB Host として使用するか、または USB Peripheral として使用するかをこのメンバ(usb_mode)に指定してください。USB Host を指定する場合は"USB_HOST"を、USB Peripheral を指定する場合は"USB_PERI"をこのメンバに指定してください。
2. USB モジュールをどの USB スピードで使用するかを指定してください。Hi-speed を指定するときは"USB_HS"を、Full-speed を指定するときは"USB_FS"を指定してください。
3. このメンバ(p_usb_reg)には USB デバイスの usb_descriptor_t 型のポインタを指定してください。usb_descriptor_t 型については、「**9.4. usb_descriptor_t構造体**」を参照してください。なお、このメンバに対する指定は USB Peripheral モードの場合にのみ行ってください。USB Host モードの場合に、このメンバに対する指定を行ってもその指定は無視されます。

9.4 usb_descriptor_t 構造体

usb_descriptor_t 構造体は、Device Descriptor や Configuration Descriptor 等の Descriptor 情報を設定するための構造体です。この構造体に設定した Descriptor 情報は、USB Host との Enumeration 時にスタンダードリクエストの応答データとして USB Host に送信されます。この構造体は、R_USB_Open 関数の引数に設定されます。

```
typedef struct usb_descriptor {
    uint8_t     *p_device;      /* Note 1 */
    uint8_t     *p_config_f;    /* Note 2 */
    uint8_t     *p_config_h;    /* Note 3 */
    uint8_t     *p_qualifier;   /* Note 4 */
    uint8_t     **pp_string;    /* Note 5 */
    uint8_t     num_string;     /* Note 6 */
} usb_descriptor_t;
```

[Note]

1. メンバ(p_device)には、Device Descriptor を記載した領域の先頭アドレスを指定してください。

2. メンバ(p_config_f)には、Full-speed 用 Configuration Descriptor を記載した領域の先頭アドレスを指定してください。なお、Hi-speed の場合も、このメンバに対し Full-speed 用 Configuration Descriptor を記載した領域の先頭アドレスを指定してください。
3. メンバ(p_config_h)には、Hi-speed 用 Configuration Descriptor を記載した領域の先頭アドレスを指定してください。Full-speed の場合、USB_NULL を指定してください。
4. メンバ(p_qualifier)には、Qualifier Descriptor を記載した領域の先頭アドレスを指定してください。Full-speed の場合は、USB_NULL を指定してください。
5. メンバ(pp_string)には、String Descriptor テーブルの先頭アドレスを指定してください。String Descriptor テーブル内には各 String Descriptor を記載した領域の先頭アドレスを指定してください。

例1) Full-speed の場合

```
usb_descriptor_t usb_descriptor =
{
      smp_device,
      smp_config_f,
      USB_NULL,
      USB_NULL,
      smp_string,
      3,
};
```

例2) Hi-speed の場合

```
usb_descriptor_t usb_descriptor =
{
      smp_device,
      smp_config_f,
      smp_config_h,
      smp_qualifier,
      smp_string,
      3,
};
```

6. メンバ(num_string)には、String Descriptor テーブルに登録した String Descriptor 数を指定してください。

9.5 usb_pipe_t 構造体

usb_pipe_t 構造体には、USB PIPE(PIPE1 から PIPE9)に関する情報が USB ドライバによって設定されます。本構造体に設定された PIPE 情報の参照は、R_USB_GetPipeInfo 関数を使用してください。

```
typedef struct usb_pipe {
      uint8_t         ep;             /* Note 1 */
      uint8_t         type;           /* Note 2 */
      uint16_t        mxps;           /* Note 3 */
} usb_pipe_t;
```

[Note]

1. メンバ(ep)には、Endpoint 番号が設定されます。なお、最上位ビットには転送方向(IN/OUT)が設定されます。最上位ビットが"1"の場合は IN 方向、"0"の場合は OUT 方向を表しています。
2. メンバ(type)には、転送タイプ(Bulk/Interrupt)が設定されます。Bulk 転送の場合は、"USB_BULK"が設定され、Interrupt 転送の場合は、"USB_INT"が設定されます。
3. メンバ(mxps)には、マックスパケットサイズが設定されます。

9.6 usb_info_t 構造体

R_USB_GetInformation 関数をコールすることにより、usb_info_t 構造体には、USB デバイスに関する以下の情報が設定されます。

```
typedef struct usb_info {
    uint8_t     type;       /* Note 1 */
    uint8_t     speed;      /* Note 2 */
    uint8_t     status;     /* Note 3 */
    uint8_t     port;       /* Note 4 */
} usb_info_t;
```

[Note]

1. USB Host モード時、メンバ(type)には、接続されている USB デバイスのデバイスクラス種別が設定され、USB デバイスが接続されていない場合は、USB_NOT_CONNECT が設定されます。なお、USB Peripheral モード時のメンバ(type)にはサポートしているデバイスクラス種別が設定されます。デバイスクラス種別については、「**7. デバイスクラス種別**」を参照してください。(PCDC の場合、USB_PCDC が設定されます。)
2. メンバ(speed)には、USB スピード(USB_HS / USB_FS / USB_LS)が設定されます。USB Host モード時、USB デバイスが接続されていない場合は、USB_NOT_CONNECT が設定されます。
3. メンバ(status)には、USB デバイスの以下の状態が設定されます。

USB_STS_DEFAULT	:	Default 状態
USB_STS_ADDRESS	:	Address 状態 (Peripheral のみ)
USB_STS_CONFIGURED	:	Configured 状態
USB_STS_SUSPEND	:	Suspend 状態
USB_STS_DETACH	:	Detach 状態

4. メンバ(port)には、接続先の Battery Charging(BC)機能に関する以下の情報が設定されます。

USB_SDP	:	Standard Downstream Port
USB_CDP	:	Charging Downstream Port
USB_DCP	:	Dedicated Charging Port (USB Peripheral のみ)

9.7 usb_compliance_t 構造体

USB Compliance Test 実施時に使用される構造体です。この構造体には、USB に関する以下の情報が設定されます。

```
typedef struct usb_compliance {
    usb_ct_status_t     status;     /* Note 1 */
    uint16_t            vid;        /* Note 2 */
    uint16_t            pid;        /* Note 3 */
} usb_compliance_t;
```

[Note]

1. メンバ status には、接続した USB デバイスの以下の状態が設定されます。

USB_CT_ATTACH	:	USB デバイスのアタッチを検出
USB_CT_DETACH	:	USB デバイスのデタッチを検出
USB_CT_TPL	:	TPL に記載された USB デバイスのアタッチを検出
USB_CT_NOTTPL	:	TPL に記載されていない USB デバイスのアタッチを検出
USB_CT_HUB	:	USB Hub の接続を検出
USB_CT_OVRCUR	:	OverCurrent を検出
USB_CT_NORES	:	Control Read 転送に対する応答がない
USB_CT_SETUP_ERR	:	Setup Transaction Error が発生

2. メンバ vid には、接続した USB デバイスの Vendor ID が設定されます。
3. メンバ pid には、接続した USB デバイスの Product ID が設定されます。

10. クラスリクエスト

この章では、USB クラスリクエストの処理方法について説明します。なお、標準リクエストは USB ドライバによって処理されますので、アプリケーションプログラム内での標準リクエストの対応は不要です。

10.1 USB Host モードの場合

10.1.1 USB リクエスト(Setup)送信

R_USB_Write 関数をつかって USB デバイスに対し USB リクエストを送信します。以下に送信手順を示します。

1. usb_ctrl_t 構造体のメンバ type に USB_REQUEST を設定してください。
2. USB リクエスト(Setup:8 バイト)を usb_ctrl_t 構造体のメンバ setup の領域に設定してください。メンバ setup の設定内容については、「9.2 usb_setup_t構造体」を参照してください。
3. コントロールライトデータステージをサポートするリクエストの場合は、送信データをバッファに格納してください。コントロールリードデータステージをサポートするリクエストの場合は、USB デバイスからの受信データを格納するバッファを確保してください。なお、バッファは自動変数(スタック)領域には確保しないでください。
4. R_USB_Write 関数の第 2 引数にデータバッファの先頭アドレスを指定し、第 3 引数にデータサイズを指定してください。なお、ノーデータコントロールステータスステージをサポートするリクエストの場合は、第 2 引数と第 3 引数に USB_NULL を指定してください。
5. R_USB_Write 関数をコールしてください。

10.1.2 USB リクエスト完了

1. Non-OS の場合

USB リクエストの完了は、R_USB_GetEvent 関数の戻り値(USB_STS_REQUEST_COMPLETE)から確認することができます。コントロールリードデータステージをサポートするリクエストの場合は、R_USB_Write 関数の第 2 引数で指定した領域に受信データが格納されています。

2. RTOS の場合

USB リクエストの完了は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event:USB_STS_REQUEST_COMPLETE)により確認することができます。コントロールリードデータステージをサポートするリクエストの場合は、R_USB_Write 関数の第 2 引数で指定した領域に受信データが格納されています。

USB リクエストの結果は usb_ctrl_t 構造体のメンバ status から確認できます。メンバ status には以下が設定されます。

status	内容
USB_ACK	正常終了
USB_STALL	ストール

10.1.3 USB リクエスト処理記述例

(1). Non-OS の場合

```
void usr_application (void )
{
    usb_ctrl_t ctrl;
    switch( R_USB_GetEvent( &ctrl ) )
    {
        /* ctrl.setup へのリクエスト設定処理 */
                    :
        /* コントロールライトデータステージをサポートするリクエストの場合は、
           g_buf 領域へ送信データを設定 */
                    :
          ctrl.type = USB_REQUEST;
```

```
                R_USB_Write(&ctrl, g_buf, size); /* USB リクエスト(Setup ステージ)送信) */
                break;
            case USB_STS_REQUEST_COMPLETE: /* USB リクエスト完了 */
                if(USB_ACK == ctrl.status) /* USB リクエスト結果確認 */
                {
                    /* コントロールリードデータステージをサポートするリクエストの場合は、
                       g_buf 領域に受信データが格納されています。 */
                                    :
                }
                break;
        }
    }
```

(2). RTOS の場合

```
    void usr_application (void)
    {
        usb_ctrl_t ctrl;
        usb_ctrl_t *p_mess;
                :
        while(1)
        {
            USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
            ctrl = *p_mess;
            switch(ctrl.event)
            {
                    /* ctrl.setup へのリクエスト設定処理 */
                        :
                    /* コントロールライトデータステージをサポートするリクエストの場合は、
                       g_buf 領域へ送信データを設定 */
                            :
                    ctrl.type = USB_REQUEST;
                    R_USB_Write(&ctrl, g_buf, size); /* USB リクエスト(Setup ステージ)送信) */
                break;
                case USB_STS_REQUEST_COMPLETE: /* USB リクエスト完了 */
                    if(USB_ACK == ctrl.status) /* USB リクエスト結果確認 */
                    {
                        /* コントロールリードデータステージをサポートするリクエストの場合は、
                           g_buf 領域に受信データが格納されています。 */
                                        :
                    }
                break;
            }
    }
```

10.2 USB Peripheral モードの場合

10.2.1 USB リクエスト(Setup)

1. Non-OS の場合

USB Host から送信される USB リクエスト(Setup)の受信は、R_USB_GetEvent 関数の戻り値(USB_STS_REQUEST)から確認することができます。受信した USB リクエスト(Setup:8 バイト)の内容は、usb_ctrl_t 構造体のメンバ setup の領域に格納されています。メンバ setup の設定内容については、「9.2 usb_setup_t構造体」を参照してください。

[Note]

ノーデータステータスステージをサポートするリクエストを受信した時の R_USB_GetEvent 関数の戻り値は、USB_STS_REQUEST ではなく USB_STS_REQUEST_COMPLETE ですのでご注意ください。

2. RTOS の場合

USB Host から送信される USB リクエスト(Setup)の受信は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event: USB_STS_REQUEST)により確認することができます。受信した USB リクエスト(Setup:8 バイト)の内容は、usb_ctrl_t 構造体のメンバ setup の領域に格納されています。メンバ setup の設定内容については、「9.2 usb_setup_t構造体」を参照してください。

[Note]
ノーデータステータスステージをサポートするリクエストを受信した時の USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event)には、USB_STS_REQUEST ではなく USB_STS_REQUEST_COMPLETE が設定されていますのでご注意ください。

10.2.2 USB リクエストデータ

データステージのデータを受信する場合は、R_USB_Read 関数を使用し、データを USB Host へ送信する場合は、R_USB_Write 関数を使用します。以下に受信手順および送信手順を示します。

1. 受信手順

(1). usb_ctrl_t 構造体のメンバ type に USB_REQUEST を設定してください。
(2). R_USB_Read 関数の第 2 引数に受信データを格納する領域へのポインタを、第 3 引数に要求データサイズを指定してください。
(3). R_USB_Read 関数をコールしてください。

[Note]

(1). リクエストデータの受信完了は、R_USB_GetEvent 関数の戻り値 USB_STS_REQUEST_COMPLETE から確認できます。(Non-OS の場合)

(2). リクエストデータの受信完了は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event: USB_STS_REQUEST_COMPLETE)により確認できます。(RTOS の場合)

2. 送信手順

(1). usb_ctrl_t 構造体のメンバ type に USB_REQUEST を設定してください。
(2). データステージのデータをバッファに格納してください。R_USB_Write 関数の第 2 引数にそのバッファの先頭アドレスを指定し、第 3 引数に送信データサイズを指定してください。
(3). R_USB_Write 関数をコールしてください。

[Note]

(1). リクエストデータの送信完了は、R_USB_GetEvent 関数の戻り値 USB_STS_WRITE_COMPLETE から確認できます。この時、usb_ctrl_t 構造体のメンバ type が USB_REQUEST になっていることも確認してください。(Non-OS の場合)

(2). リクエストデータの送信完了は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event: USB_STS_WRITE_COMPLETE)により確認できます。この時、usb_ctrl_t 構造体のメンバ type が USB_REQUEST になっていることも確認してください。(RTOS の場合)

10.2.3 USB リクエスト結果

クラスごとのコンフィグレーションファイル(例: r_usb_pcdc_config.h など)内に定義されているクラスリクエスト設定定義(例: USB_CFG_PCDC_REQUEST など)に対し USB_CFG_ENABLE を設定している場合、この USB ドライバは、受信したクラスリクエストに対し、必ず ACK 応答します。

[Note]

ベンダクラスリクエストの場合は、USB ドライバがベンダクラスリクエストに対する ACK / STALL 応答の処理を行いません。アプリケーションプログラムからベンダクラスリクエストに対する ACK/STALL 応答を行う必要があります。ACK/STALL 応答の方法については、「**10.2.5 クラスリクエストに対する ACK/STALL応答処理**」を参照してください。

10.2.4 USB リクエスト処理記述例

1. コントロールリードデータステージをサポートするリクエストの場合

(1). Non-OS の場合

```
void usr_application (void)
{
    usb_ctrl_t ctrl;
    switch (R_USB_GetEvent(&ctrl))
    {
                    :
        case USB_REQUEST: /* USB リクエスト受信 */
            /* ctrl.setup の解析処理 */
                    :
            /* データ設定処理 */
                    :
            ctrl.type = USB_REQUEST;
            R_USB_Write(&ctrl, g_buf, size); /* データ(データステージ)送信要求 */
            break;
        case USB_STS_REQUEST_COMPLETE:
            if (USB_ACK == ctrl.status) /* USB リクエスト結果確認 */
            {
                    :
            }
            break;
    }
}
```

(2). RTOS の場合

```
void usr_application (void)
{
    usb_ctrl_t ctrl;
    usb_ctrl_t *p_mess;
        :
    while(1)
    {
        USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
        ctrl = *p_mess;
        switch (ctrl.event)
        {
                    :
            case USB_REQUEST: /* USB リクエスト受信 */
                /* ctrl.setup の解析処理 */
                    :
                /* データ設定処理 */
                    :
                ctrl.type = USB_REQUEST;
```

```
                R_USB_Write(&ctrl, g_buf, size); /* データ(データステージ)送信要求 */
            break;
            case USB_STS_REQUEST_COMPLETE:
                if (USB_ACK == ctrl.status) /* USB リクエスト結果確認 */
                {
                    :
                }
            break;
                    :
        }
    }
}
```

2. コントロールライトデータステージをサポートするリクエストの場合

(1). Non-OS の場合

```
void usr_application (void)
{
    usb_ctrl_t ctrl;
    switch (R_USB_GetEvent(&ctrl))
    {
                    :
        case USB_REQUEST: /* USB リクエスト受信 */
            /* ctrl.setup の解析処理 */
                    :
            ctrl.type = USB_REQUEST;
            R_USB_Read(&ctrl, g_buf, size); /* データ(データステージ)受信要求 */
            break;
        case USB_STS_REQUEST_COMPLETE:
                    :
            break;
    }
}
```

(2). RTOS の場合

```
void usr_application (void )
{
    usb_ctrl_t ctrl;
    usb_ctrl_t *p_mess;
            :
    while(1)
    {
        USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
        ctrl = *p_mess;
        switch (ctrl.event)
        {
                    :
            case USB_REQUEST: /* USB リクエスト受信 */
                /* ctrl.setup の解析処理 */
                    :
                ctrl.type = USB_REQUEST;
                R_USB_Read(&ctrl, g_buf, size); /* データ(データステージ)受信要求 */
            break;
            case USB_STS_REQUEST_COMPLETE:
                    :
            break;
                    :
        }
    }
```

```
}
```

3. ノーデータコントロールステータスステージをサポートするリクエストの場合

(1). Non-OS の場合

```
void usr_application (void)
{
    usb_ctrl_t ctrl;
    switch( R_USB_GetEvent( &ctrl ) )
    {
                    :
        case USB_STS_REQUEST_COMPLETE:
            /* ctrl.setup の解析処理 */
                    :
                    :
        break;
    }
}
```

(2). RTOS の場合

```
void usr_application (void)
{
    usb_ctrl_t ctrl;
    usb_ctrl_t *p_mess;
          :
    while(1)
    {
        USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
        ctrl = *p_mess;
        switch (ctrl.event)
        {
                    :
            case USB_STS_REQUEST_COMPLETE:
                /* ctrl.setup の解析処理 */
                    :
            break;
                    :
        }
    }
}
```

10.2.5 クラスリクエストに対する ACK/STALL 応答処理

1. Non-OS の場合

クラスリクエストに対し ACK / STALL 応答を行う必要がある場合は、usb_ctrl_t 構造体のメンバ type に対し USB_REQUEST を、メンバ status に対し USB_ACK / USB_STALL を設定し、R_USB_Write 関数をコールしてください。R_USB_Write 関数の第二引数および第三引数には、ともに USB_NULL を設定してください。なお、ACK / STALL の送信完了は、R_USB_GetEvent 関数の戻り値 USB_STS_REQUEST_COMPLETE により確認できます。この時、usb_ctrl_t 構造体のメンバ type が USB_REQUEST になっていることも確認してください。

2. RTOS の場合

クラスリクエストに対し ACK / STALL 応答を行う必要がある場合は、usb_ctrl_t 構造体のメンバ type に対し USB_REQUEST を、メンバ status に対し USB_ACK / USB_STALL を設定し、R_USB_Write 関数をコールしてください。R_USB_Write 関数の第二引数および第三引数には、ともに USB_NULL を設定してください。なお、ACK / STALL の送信完了は、USB ドライバに登録したコールバック関数の引

数(usb_ctrl_t 構造体のメンバ event: USB_STS_REQUEST_COMPLETE)により確認できます。この時、usb_ctrl_t 構造体のメンバ type が USB_REQUEST になっていることも確認してください。

1. STALL 応答の処理例

(1). Non-OS の場合

```
void usr_application (void )
{
    usb_ctrl_t ctrl;
    switch( R_USB_GetEvent( &ctrl ) )
    {
                    :
        case USB_STS_REQUEST:
            /* ctrl.setup の解析処理 */
                    :
            ctrl.type = USB_REQUEST:
            ctrl.status = USB_STALL;
            R_USB_Write(&ctrl, (uint8_t *)USB_NULL, (uint32_t)USB_NULL);
        break;
        case USB_STS_REQUEST_COMPLETE:
            if( USB_REQUEST == ctrl.type )
            {
                    :
            }
        break;
    }
}
```

(2). RTOS の場合

```
void usr_application_task (void )
{
    usb_ctrl_t ctrl;
    usb_ctrl_t *p_mess;
        :
    while(1)
    {
        USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
        ctrl = *p_mess;
        switch (ctrl.event)
        {
                    :
            case USB_STS_REQUEST:
                /* ctrl.setup の解析処理 */
                    :
                ctrl.type = USB_REQUEST:
                ctrl.status = USB_STALL;
                R_USB_Write(&ctrl, (uint8_t *)USB_NULL, (uint32_t)USB_NULL);
            break;
            case USB_STS_REQUEST_COMPLETE:
                if( USB_REQUEST == ctrl.type )
                {
                        :
                }
            break;
                    :
        }
    }
}
```

2. ACK 応答の処理例

(1). Non-OS の場合

```
void usr_application (void )
{
    usb_ctrl_t ctrl;
    switch( R_USB_GetEvent( &ctrl ) )
    {
                    :
        case USB_STS_REQUEST:
            /* ctrl.setup の解析処理 */
                    :
            ctrl.type = USB_REQUEST:
            ctrl.status = USB_ACK;
            R_USB_Write(&ctrl, (uint8_t *)USB_NULL, (uint32_t)USB_NULL);
            break;
        case USB_STS_REQUEST_COMPLETE:
            if( USB_REQUEST == ctrl.type )
            {
                        :
            }
            break;
    }
}
```

(2). RTOS の場合

```
void usr_application (void )
{
    usb_ctrl_t ctrl;
    usb_ctrl_t *p_mess;
        :
    while(1)
    {
        USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
        ctrl = *p_mess;
        switch (ctrl.event)
        {
                    :
            case USB_STS_REQUEST:
                /* ctrl.setup の解析処理 */
                    :
                ctrl.type = USB_REQUEST:
                ctrl.status = USB_ACK;
                R_USB_Write(&ctrl, (uint8_t *)USB_NULL, (uint32_t)USB_NULL);
            break;
            case USB_STS_REQUEST_COMPLETE:
                if( USB_REQUEST == ctrl.type )
                {
                        :
                }
            break;
                        :
        }
    }
}
```

11. DMA/DTC 転送

11.1 基本仕様

USB-BASIC-FW FIT モジュールの DMA/DTC 転送サンプルプログラムの仕様を以下に示します。DMA/DTC 転送が可能な PIPE 番号は PIPE1 と PIPE2 です。**Table11-1**に DMA/DTC 設定仕様を示します。

Table11-1 DMA/DTC Setting Specifications

説明	備考
使用 FIFO ポート	D0FIFO ポート、D1FIFO ポート
転送モード	ブロック転送モード
チェイン転送	禁止
アドレスモード	フルアドレスモード
リードスキップ	禁止
アクセスビット幅（MBW）	4 バイト転送：32 ビット幅(USBA/USBAa モジュール使用時) 2 バイト転送：16 ビット幅(USBb モジュール使用時)
USB 転送タイプ	BULK 転送
転送終了	受信方向：BRDY 割り込み 送信方向：D0FIFO/D1FIFO 割り込み, BEMP 割り込み

[Note]

本ドライバは DMA 転送と DTC 転送を同時に使用することはできません。

11.2 注意事項

11.2.1 DTC 転送について

DTC 転送を行う場合、「RX Family DTC モジュール」アプリケーションノート(Document No.R01AN1819)内の「R_DTC_Open」の章に記載された「Special Notes」を参照してください。

11.2.2 データ受信バッファ領域のサイズについて

受信したデータを格納するバッファは以下のサイズを確保してください。

1. r_usb_basic_config.h 内の USB_CFG_CNTMD 定義に対し USB_CFG_CNTMDON を指定している場合 (USBA/USBAa モジュールを使用している場合)

   FIFO バッファサイズ×n 倍以上のサイズを確保してください。なお、FIFO バッファサイズについては、「12.4 **PIPEBUF レジスタの参照/変更について**」を参照してください。

2. r_usb_basic_config.h 内の USB_CFG_CNTMD 定義に対し USB_CFG_CNTMDOFF を指定している場合

   MaxPacketSize×n 倍のサイズを確保してください。

11.2.3 USB PIPE について

DMA/DTC 転送で使用される USB PIPE は PIPE1 と PIPE2 のみです。DMA/DTC 転送用に PIPE1 と PIPE2 以外の USB PIPE を指定した場合、DMA/DTC によるデータ転送は行われません。

なお、DMA/DTC 転送と CPU 転送を組み合わせてデータ転送を行う場合、DMA/DTC 転送には、PIPE1/PIPE2 を指定し、CPU 転送には、PIPE3/PIPE4/PIPE5 を指定してください。

[Note]

USB PIPE は、各デバイスクラスのコンフィグレーションファイルで指定してください。

11.2.4 DMA/DTC 用初期化関数について

以下の DMA/DTC 初期化関数はアプリケーションプログラムからコールしてください。

転送	関数
DTC	R_DTC_Open
DMA	R_DMACA_Init R_DMACA_Open

[Note]

R_DMACA_Open 関数の引数には、以下の定義を指定してください。以下の定義の詳細については、「8. コンフィグレーション **(r_usb_basic_config.h)**」を参照してください。

USB_CFG_USB0_DMA_TX, USB_CFG_USB0_DMA_RX
USB_CFG_USB1_DMA_TX, USB_CFG_USB1_DMA_RX

例)

```
R_DMACA_Init();
R_DMACA_Open(USB_CFG_USB1_DMA_TX);
R_DMACA_Open(USB_CFG_USB1_DMA_RX);
```

12. 注意事項

12.1 Vendor ID について

Device Descriptor に記載する Vendor ID は、必ずお客様用の Vendor ID をご使用いただきますようお願いします。

12.2 Compliance Test 対応について

USB Compliance Test では、LCD 等の表示機器に USB デバイスに関する情報を表示する必要があります。コンフィグレーションファイル(r_usb_basic_config.h)内の USB_CFG_COMPLIANCE 定義に対し USB_CFG_ENABLE を指定した時、USB ドライバは下記の関数(usb_compliance_disp)をコールします。アプリケーションプログラム内でこの関数を定義し、この関数内に USB デバイスに関する表示処理等を記述いただきますようお願いします。

関数名 : void usb_compliance_disp
引数 : usb_compliance_t * USB 情報を格納する構造体へのポインタ

[Note]

1. USB デバイスに関する情報を引数が示す領域に USB ドライバが設定し、usb_compliance_disp 関数をコールします。
2. usb_compliance_t 構造体については、「**9.7 usb_compliance_t構造体**」を参照してください。
3. r_usb_basic_config.h 内の USB_CFG_COMPLIANCE 定義に対し USB_CFG_ENABLE を指定した時、USB デバイス用と USB Hub 用の TPL 定義に対し Vendor ID と Product ID の登録が必要です。TPL 定義については、「**3.6 ターゲットペリフェラルリスト(TPL)の設定方法**」を参照してください。
4. usb_compliance_disp 関数の参考プログラム例は、「**14.1 usb_compliance_disp関数**」を参照してください。

12.3 Hi-speed Embedded Host Electrical Test について

Hi-speed Embedded Host Electrical Test を実施する場合、USB Protocol and Electrical Test Tool が必要になります。なお、当該 Test を実施する場合、r_usb_basic_config.h ファイル内の USB_CFG_ELECTRICAL 定義に対し、USB_CFG_ENABLE を定義してください。当該定義については、「**8. コンフィグレーション(r_usb_basic_config.h)**」を参照してください。

12.4 PIPEBUF レジスタの参照/変更について

USBA / USBAa モジュールがサポートしている PIPEBUF レジスタの BUFSIZE ビットおよび BUFNMB ビットに対しては推奨値が設定されています。これらのビットに対する参照/変更を行う場合は、USB ドライバ内の以下の変数を参照/変更いただきますようお願いします。

デバイスクラス	ファイル名	変数名
Host Communication Device Class	r_usb_hcdc_driver.c	g_usb_hcdc_eptbl
Host Human I/F Device Class	r_usb_hhid_driver.c	g_usb_hhid_eptbl
Host Mass Storage Class	r_usb_hmsc_driver.c	g_usb_hmsc_eptbl
Peripheral Communication Device Class Peripheral Human I/F Device Class Periphral Mass Storage Class	r_usb_peptable.c	g_usb_eptbl

12.5 RTOS

12.5.1 セマフォ

1. r_usb_basic_config.h ファイル内の USB_CFG_MULTI_TASK 定義に対し、USB_CFG_ENABLE を指定した場合、以下の API はセマフォ獲得処理を行います。もし、他のタスクによってセマフォが獲得されている時に、以下の API をコールした場合、その API をコールしたタスクはセマフォ待ち状態に移行します。

   R_USB_Open / R_USB_Close / R_USB_Read / R_USB_Write / R_USB_Suspend

   R_USB_Resume / R_USB_VbusSetting / R_USB_PipeRead / R_USB_PipeWrite

2. 本 USB ドライバは USB モジュールごとに以下のセマフォを確保しています。

   SemaphoreHandle_t g_usb_semaphore_hdl[USB_NUM_USBIP];

[Note]

USB0 モジュールには、g_usb_semaphore_hdl[0]のセマフォが使用され、USB1 モジュールには、g_usb_semaphore_hdl[1]が使用されます。

12.5.2 タスク優先度

USB ドライバ用タスクに指定されている優先度は 8 から 11 です。アプリケーションタスクの優先度には 0 から 7 の値を指定してください。

13. アプリケーションプログラムの作成方法

本章では、このドキュメントに記載された API を使ったアプリケーションプログラムの作成方法について説明します。なお、アプリケーションプログラムは、このドキュメントに記載された API を使ってアプリケーションプログラム開発を行ってください。

13.1 コンフィグレーション

お客様のシステムにあわせて"r_config"フォルダ内にある各コンフィグレーションファイルの設定をお願いします。コンフィグレーションファイルの設定については、「8. **コンフィグレーション (r_usb_basic_config.h)**」を参照してください。

13.2 Descriptor の作成

USB Peripheral モードの場合、お客様のシステムに合わせた Descriptor の作成が必要です。作成した Descriptor は、usb_descriptor_t 構造体の各メンバに登録してください。なお、USB Host モードの場合、Descriptor の作成は不要です。

13.3 アプリケーションプログラム作成

13.3.1 インクルード

以下のファイルをアプリケーションプログラム内でインクルードしてください。

1. r_usb_basic_if.h (必ずインクルードしてください。)
2. r_usb_xxxxx_if.h (使用する USB デバイスクラスで用意されている I/F ファイルです。)
3. Host Mass Storage Class 用のアプリケーションプログラムの場合、FAT 用のヘッダファイルをインクルードしてください。
4. その他､アプリケーションプログラム内で使用するドライバ関連のヘッダファイルをインクルードしてください。

13.3.2 初期化処理

1. USB 端子設定

USB コントローラを使用するためには、USB の入出力端子を設定する必要があります。以下に、設定が必要な USB 端子を示します。必要に応じて以下の端子に対する設定を行ってください。

Table13-1 USB Peripheral モード時の USB 入出力端子設定

端子名	入出力	機能
USB_VBUS	入力	USB 用 VBUS 端子
USB_DPUPE	出力	プルアップ抵抗制御信号端子

Table13-2 USB Host モード時の USB 入出力端子設定

端子名	入出力	機能
USB_VBUSEN	出力	USB 用 VBUS 出力許可端子
USB_OVRCURA	入力	USB 用オーバーカレント検出端子

[Note]

(1). MCU のユーザーズマニュアルを参照し、ご使用のボードに合わせて端子設定を行ってください。

(2). USB_DPUPE 端子は RX63N/RX631 のみでサポートされている端子です。

(3). RX63N/RX631 をご使用の場合、必要に応じて USB_DPRPD および USB_DRPD 端子に対する設定を行ってください。

2. DMA/DTC 関連初期化

DMA/DTC 転送を行う場合は、DMA/DTC 初期化関数をコールしてください。

転送	関数
DTC	R_DTC_Open
DMA	R_DMACA_Init R_DMACA_Open

[Note]

(1). DMA/DTC 転送を行う場合は、r_usb_basic_config.h ファイルの設定が必要です。「**8. コンフィグレーション (r_usb_basic_config.h)**」を参照してください。

(2). DMA 転送を行う場合、R_DMACA_Open 関数の引数に対し、使用する DMA Channel 番号を指定する必要があります。なお、当該引数には、r_usb_basic_config.h ファイル内で指定した以下のいずれかの定義を指定してください。

DMA Channel 番号	内容
USB_CFG_USB0_DMA_TX	USB0 モジュール用送信設定
USB_CFG_USB0_DMA_RX	USB0 モジュール用受信設定
USB_CFG_USB1_DMA_TX	USB1 モジュール用送信設定
USB_CFG_USB1_DMA_RX	USB1 モジュール用受信設定

記述例 1) USB0 モジュールを使って DMA 送信設定を行う場合

```
R_DMACA_Open(USB_CFG_USB0_DMA_TX);
```

※ 受信用 USB PIPE には PIPE3 から PIPE5 を指定してください。

記述例 2) USB1 モジュールを使って DMA 受信設定を行う場合

```
R_DMACA_Open(USB_CFG_USB1_DMA_RX);
```

※ 送信用 USB PIPE には PIPE3 から PIPE5 を指定してください。

記述例 3) USB1 モジュールを使って DMA 送受信設定を行う場合

```
R_DMACA_Open(USB_CFG_USB1_DMA_TX);
R_DMACA_Open(USB_CFG_USB1_DMA_RX);
```

※ USB PIPE1 と PIPE2 以外の USB PIPE を指定しないでください。

(3). DMA/DTC 転送で使用可能な USB PIPE は PIPE1 と PIPE2 のみです。本ドライバは、USB PIPE3 から PIPE5 を使用した場合の DMA/DTC 転送はサポートしていません。

(4). 使用する USB PIPE 番号の指定は各デバイスクラスのコンフィグレーションファイルで行います。

3. USB 関連初期化

R_USB_Open 関数をコールし、使用する USB モジュール(HW)および USB ドライバ(SW)の初期化を行ってください。

4. コールバック関数の作成および登録 (RTOS のみ)

R_USB_Callback 関数により登録するコールバック関数を作成してください。作成後、R_USB_Callback 関数を使って当該コールバック関数を USB ドライバに登録してください。

コールバック関数の引数には、USB 完了イベントのほか、そのイベントに応じた各種情報が USB ドライバによって設定されています。必ず、RTOS の API 等を使ってアプリケーション用タスクへ当該引数の情報を通知してください。

記述例)

```
void usb_apl_callback (usb_ctrl_t *p_ctrl, usb_hdl_t hdl, uint8_t is_request)
{
    /* RTOS の API 使って USB イベントの情報をアプリケーションタスクに通知 */
    USB_APL_SND_MSG(USB_APL_MBX, (usb_msg_t *)p_ctrl);
}
```

13.3.3 Descriptor の作成 (USB Peripheral のみ)

USB Peripheral モードの場合は、お客様のシステムに応じた Descriptor を作成してください。Descriptor については「**2.5 Descriptor**」を参照してください。USB Host モードの場合は、Descriptor を作成する必要はありません。

13.3.4 メインルーチン / アプリケーションタスクの作成

(1). Non-OS

メインルーチンは、メインループ形式で記述してください。そのメインループ内では必ず R_USB_GetEvent 関数をコールしてください。USB 関連の完了イベントは、R_USB_GetEvent 関数の戻り値から取得できます。アプリケーションプログラムでは、戻り値をトリガに、各戻り値に応じたプログラムを記述してください。

[Note]

a. R_USB_Read/R_USB_Write/R_USB_PipeRead/R_USB_PipeWrite 関数による USB データ通信は、R_USB_GetEvent 関数からの戻り値 USB_STS_CONFIGURED を確認した後で行ってください。

b. Host Mass Storage クラスで、MSC デバイスに対するファイルアクセスを行う場合は FAT がサポートしている API をご使用ください。

(2). RTOS

アプリケーションプログラム用タスクは、ループ形式で記述してください。そのメインループ内では必ずコールバック関数から通知される情報(USB 完了イベントなど)を取得するための RTOS 用 API をコールしてください。アプリケーションタスクでは取得した USB 完了イベントをトリガにして、各 USB 完了イベントに応じたプログラムを記述してください。

[Note]

a. USB デバイスが CONFIGURED 状態の時に、R_USB_Read / R_USB_Write / R_USB_PipeRead / R_USB_PipeWrite 関数による USB データ通信を行ってください。USB デバイスが CONFIGURED 状態かどうかの確認は、USB ドライバに登録したコールバック関数の引数(usb_ctrl_t 構造体のメンバ event:USB_STS_CONFIGURED)により確認することができます。

b. Host Mass Storage クラスで、MSC デバイスに対するファイルアクセスを行う場合は FAT がサポートしている API をご使用ください。

13.3.5 リアルタイム OS への登録 (RTOS のみ)

以下をリアルタイム OS に登録してください。

1. アプリケーションプログラムタスク
2. アプリケーションタスクやコールバック関数で使用しているリアルタイム OS の機能

[Note]

アプリケーションプログラムタスクの優先度は、7 以下の優先度値を指定してください。

13.3.6 アプリケーションプログラム記述例

(1). Non-OS

a. CPU 転送の場合

```
#include "r_usb_basic_if.h"
#include "r_usb_pcdc_if.h"

void      usb_peri_application( void )
{
      usb_ctrl_t    ctrl;
      usb_cfg_t    cfg;

      /* USB 端子設定 */
      usb_pin_setting();

      /* 初期化処理 */
      ctrl.module = USB_IP1; /* 使用する USB モジュールを指定 */
      cfg.usb_mode = USB_PERI; /* USB Host か USB Peri かを指定 */
      cfg.usb_speed = USB_HS; /* USB スピードを指定 */
      cfg.p_usb_reg = &smp_descriptor; /* Descriptor テーブルの先頭アドレスを指定 */
      R_USB_Open( &ctrl, &cfg );

      /* メインルーチン */
      while(1)
      {
            switch( R_USB_GetEvent( &ctrl ) )
            {
                  case USB_STS_CONFIGURED:
                  case USB_STS_WRITE_COMPLETE:
                        ctrl.type = USB_PCDC;
                        R_USB_Read( &ctrl, g_buf, 64 );
                        break;
                  case USB_STS_READ_COMPLETE:
                        ctrl.type = USB_PCDC;
                        R_USB_Write( &ctrl, g_buf, ctrl.size );
                        break;
                  default:
                        break;
            }
      }
}
```

b. DMA 転送の場合

```
#include "r_usb_basic_if.h"
#include "r_usb_pcdc_if.h"

void      usb_peri_application( void )
{
      usb_ctrl_t    ctrl;
      usb_cfg_t     cfg;

      /* USB 端子設定 */
      usb_pin_setting();

      /* DMA 初期化処理 */
      R_DMACA_Init();
      R_DMACA_Open(USB_CFG_USB0_DMA_TX);
      R_DMACA_Open(USB_CFG_USB0_DMA_RX);

      /* 初期化処理 */
      ctrl.module = USB_IP0; /* 使用する USB モジュールを指定 */
      cfg.usb_mode = USB_PERI; /* USB Host か USB Peri かを指定 */
      cfg.usb_speed = USB_HS; /* USB スピードを指定 */
      cfg.p_usb_reg = &smp_descriptor; /* Descriptor テーブルの先頭アドレスを指定 */
      R_USB_Open( &ctrl, &cfg );

      /* メインルーチン */
      while(1)
      {
            switch( R_USB_GetEvent( &ctrl ) )
            {
                  case USB_STS_CONFIGURED:
                  case USB_STS_WRITE_COMPLETE:
                        ctrl.type = USB_PCDC;
                        R_USB_Read( &ctrl, g_buf, 64 );
                        break;
                  case USB_STS_READ_COMPLETE:
                        ctrl.type = USB_PCDC;
                        R_USB_Write( &ctrl, g_buf, ctrl.size );
                        break;
                  default:
                        break;
            }
      }
}
```

c. DTC 転送の場合

```
#include "r_usb_basic_if.h"
#include "r_usb_pcdc_if.h"

void      usb_peri_application( void )
{
      usb_ctrl_t    ctrl;
      usb_cfg_t     cfg;

      /* USB 端子設定 */
      usb_pin_setting();

      /* DTC 初期化処理 */
      R_DTC_Open();

      /* 初期化処理 */
      ctrl.module = USB_IP0; /* 使用する USB モジュールを指定 */
      cfg.usb_mode = USB_PERI; /* USB Host か USB Peri かを指定 */
      cfg.usb_speed = USB_HS; /* USB スピードを指定 */
      cfg.p_usb_reg = &smp_descriptor; /* Descriptor テーブルの先頭アドレスを指定 */
      R_USB_Open( &ctrl, &cfg );

      /* メインルーチン */
      while(1)
      {
            switch( R_USB_GetEvent( &ctrl ) )
            {
                  case USB_STS_CONFIGURED:
                  case USB_STS_WRITE_COMPLETE:
                        ctrl.type = USB_PCDC;
                        R_USB_Read( &ctrl, g_buf, 64 );
                        break;
                  case USB_STS_READ_COMPLETE:
                        ctrl.type = USB_PCDC;
                        R_USB_Write( &ctrl, g_buf, ctrl.size );
                        break;
                  default:
                        break;
            }
      }
}
```

(2). RTOS

a. CPU 転送の場合

```
#include "r_usb_basic_if.h"
#include "r_usb_pcdc_if.h"

/* コールバック関数 */
void usb_apl_callback(usb_ctrl_t *p_ctrl, usb_hdl_t hdl, uint8_t is_request)
{
    /* USB 完了イベントをアプリケーションプログラム用タスクへ通知 */
    USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}

void    usb_apl_task( void )
{
    usb_ctrl_t  ctrl;
    usb_ctrl_t  *p_mess;
    usb_cfg_t   cfg;

    /* USB 端子設定 */
    usb_pin_setting();

    /* 初期化処理 */
    ctrl.module = USB_IP1; /* 使用する USB モジュールを指定 */
    cfg.usb_mode = USB_PERI; /* USB Host か USB Peri かを指定 */
    cfg.usb_speed = USB_HS; /* USB スピードを指定 */
    cfg.p_usb_reg = &smp_descriptor; /* Descriptor テーブルの先頭アドレスを指定 */
    R_USB_Open(&ctrl, &cfg);

    /* コールバック関数の登録 */
    R_USB_Callback(usb_apl_callback);

    /* メインルーチン */
    while(1)
    {
        /* コールバック関数より通知された USB 完了イベントを取得 */
        USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t *)&p_mess);

        ctrl = *p_mess;
        switch(ctrl.event)
        {
            case USB_STS_CONFIGURED:
            case USB_STS_WRITE_COMPLETE:
                ctrl.type = USB_PCDC;
                R_USB_Read( &ctrl, g_buf, 64 );
            break;
            case USB_STS_READ_COMPLETE:
                ctrl.type = USB_PCDC;
                R_USB_Write( &ctrl, g_buf, ctrl.size );
            break;
            default:
            break;
        }
    }
}
```

b. DMA 転送の場合

```
#include "r_usb_basic_if.h"
#include "r_usb_pcdc_if.h"

/* コールバック関数 */
void usb_apl_callback(usb_ctrl_t *p_ctrl, usb_hdl_t hdl, uint8_t is_request) /* コールバック関数 */
{
    /* USB 完了イベントをアプリケーションプログラム用タスクへ通知 */
    USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}

void    usb_application_task(void)
{
    usb_ctrl_t  ctrl;
    usb_ctrl_t  *p_mess;
    usb_cfg_t   cfg;

    /* USB 端子設定 */
    usb_pin_setting();

    /* DMA 初期化処理 */
    R_DMACA_Init();
    R_DMACA_Open(USB_CFG_USB0_DMA_TX);
    R_DMACA_Open(USB_CFG_USB0_DMA_RX);

    /* 初期化処理 */
    ctrl.module = USB_IP0; /* 使用する USB モジュールを指定 */
    cfg.usb_mode = USB_PERI; /* USB Host か USB Peri かを指定 */
    cfg.usb_speed = USB_HS; /* USB スピードを指定 */
    cfg.p_usb_reg = &smp_descriptor; /* Descriptor テーブルの先頭アドレスを指定 */
    R_USB_Open( &ctrl, &cfg );

    /* コールバック関数の登録 */
    R_USB_Callback(usb_apl_callback);

    /* メインルーチン */
    while(1)
    {
        /* コールバック関数より通知された USB 完了イベントを取得 */
        USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);

        ctrl = *p_mess;
        switch (ctrl.event)
        {
            case USB_STS_CONFIGURED:
            case USB_STS_WRITE_COMPLETE:
                ctrl.type = USB_PCDC;
                R_USB_Read( &ctrl, g_buf, 64 );
            break;
            case USB_STS_READ_COMPLETE:
                ctrl.type = USB_PCDC;
                R_USB_Write( &ctrl, g_buf, ctrl.size );
                break;
            default:
            break;
        }
    }
}
```

c. DTC 転送の場合

```
#include "r_usb_basic_if.h"
#include "r_usb_pcdc_if.h"

/* コールバック関数 */
void usb_apl_callback(usb_ctrl_t *p_ctrl, usb_hdl_t hdl, uint8_t is_request) /* コールバック関数 */
{
    /* USB 完了イベントをアプリケーションプログラム用タスクへ通知 */
    USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}

void    usb_application_task(void)
{
    usb_ctrl_t  ctrl;
    usb_ctrl_t  *p_mess;
    usb_cfg_t   cfg;

    /* USB 端子設定 */
    usb_pin_setting();

    /* DTC 初期化処理 */
    R_DTC_Open();

    /* 初期化処理 */
    ctrl.module = USB_IP0; /* 使用する USB モジュールを指定 */
    cfg.usb_mode = USB_PERI; /* USB Host か USB Peri かを指定 */
    cfg.usb_speed = USB_HS; /* USB スピードを指定 */
    cfg.p_usb_reg = &smp_descriptor; /* Descriptor テーブルの先頭アドレスを指定 */
    R_USB_Open( &ctrl, &cfg );

    /* コールバック関数の登録 */
    R_USB_Callback(usb_apl_callback);

    /* メインルーチン */
    while(1)
    {
        /* コールバック関数より通知された USB 完了イベントを取得 */
        USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);

        ctrl = *p_mess;

        switch (ctrl.event)
        {
            case USB_STS_CONFIGURED:
            case USB_STS_WRITE_COMPLETE:
                ctrl.type = USB_PCDC;
                R_USB_Read( &ctrl, g_buf, 64 );
            break;
            case USB_STS_READ_COMPLETE:
                ctrl.type = USB_PCDC;
                R_USB_Write( &ctrl, g_buf, ctrl.size );
            break;
            default:
            break;
        }
    }
}
```

14. 参考プログラム例

14.1 usb_compliance_disp 関数

```
void usb_compliance_disp (usb_compliance_t *p_info)
{
    uint8_t                 disp_data[32];

    disp_data = (usb_comp_disp_t*)param;

    switch(p_info->status)
    {
        case USB_CT_ATTACH:         /* Device Attach Detection */
            display("ATTACH ");
        break;

        case USB_CT_DETACH:         /* Device Detach Detection */
            display("DETTACH");
        break;
        case USB_CT_TPL:            /* TPL device connect */
            sprintf(disp_data,"TPL PID:%04x VID:%04x",p_info->pid, p_info->vid);
            display(disp_data);
        break;

        case USB_CT_NOTTPL:         /* Not TPL device connect */
            sprintf(disp_data,"NOTPL PID:%04x VID:%04x",p_info->pid, p_info->vid);
            display(disp_data);
        break;

        case USB_CT_HUB:            /* USB Hub connect */
            display("Hub");
        break;

        case USB_CT_NOTRESP:        /* Response Time out for Control Read Transfer */
            display("Not response");
        break;

        default:
        break;
    }
```

[Note]

上記関数内にある display 関数は、表示機器に文字列を表示するための関数で、お客様にご用意いただく関数です。

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/inquiry

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Aug 1, 2014	—	初版発行
1.10	Dec 26, 2014	—	1. 対象デバイスに RX71M を追加。 2. Host モード時、USB デバイスの複数接続をサポート。
1.11	Sep 30, 2015	—	対象デバイスに RX63N と RX631 を追加。
1.20	Sep 30, 2016	—	1. 対象デバイスに RX65N/RX651 を追加 2. DMA 転送をサポート 3. USB Host and Peripheral Interface Driver アプリケーションノート(ドキュメント No.R01AN3293JJ)に対応
1.21	Mar 31, 2017	—	1. Technical Update(発行番号: TN-RX*-A172A/J)に対応しました。 2. 以下の章を追加しました。 (1). 2.5 Descriptor (2). 3.6 ターゲットペリフェラルリストの設定方法 (3). 3.7 デバイスアドレスの割り当て (4). 5. R_USB_GetEvent 関数の戻り値 (5). 6. デバイス種別 (6). 7. コンフィグレーション (7). 8. 構造体 (8). 9. クラスリクエスト (9). 11. 注意事項 (10).13. 参考プログラム例 3. 以下の章を削除しました。 "Hub", "non-OS スケジューラ"
1.22	Sep 30, 2017	—	RX65N/RX651-2M をサポート
1.23	Mar 31, 2018	—	1. Smart Configurator に対応しました。 2. usb_descriptor_t 構造体にメンバ num_string を追加しました。
1.24	Dec 28, 2018	—	RTOS をサポート

製品ご使用上の注意事項

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

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】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、将来の機能拡張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。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、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、内部ROM、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、ノイズ輻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)

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口　http://www.renesas.com

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

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：https://www.renesas.com/contact/