RZ/A2M グループ

USB Basic Peripheral Driver

要旨

本アプリケーションノートでは、USB Basic Peripheral Driver Firmware について説明します。以降、本モジュールを USB-BASIC-F/W と称します。

動作確認デバイス

RZ/A2M

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

目次

1. 概要

USB-BASIC-F/W は、USB の H/W 制御を行います。USB-BASIC-F/W 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 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

＜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

- 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。
- 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 USB2.0 ファンクションモジュールのチャネル 0 およびチャネル 1 を示しています。

本書内の表記	RZ/A2M グループユーザーズマニュアル ハードウェア編（No.R01UH0403746JJ）の表記	開始アドレス
USB0 モジュール	USB2.0 ファンクションモジュール チャネル 0	0xE821 9000
USB1 モジュール	USB2.0 ファンクションモジュール チャネル 1	0xE821 B000

1.2 制限事項

本 F/W には以下の制限事項があります。

(1). 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。

(2). USB Host モードはサポートしていません。

(3). USB Hub 使用時の DMA 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 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。

(4). 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。

(5). Vendor Class の場合、USB Hub を使用することはできません。

(6). 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。

1.3 用語一覧

本資料で使用される用語と略語は以下のとおりです。

用語／略語	詳細
APL	Application program
CDP	Charging Downstream Port
DCP	Dedicated Charging Port
H/W	Renesas USB device RZ/A2Mグループ
MGR	Peripheral device state manager of HCD
PBC	Peripheral Battery Charging control
PCD	Peripheral control driver of USB-BASIC-F/W
PDCD	Peripheral device class driver (device driver and USB class driver)
USB	Universal Serial Bus
USB-BASIC-F/W	USB Basic Peripheral firmware for RZ/A2Mグループ
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 ソフトウェア構成

1.4.1 モジュール構成

USB-BASIC-F/W を構成するソフトウェアは“タスク”構造で作成されています。図 1.1 に USB-BASIC-F/W のタスク構成、表 1.1 にソフトウェア機能概要を示します。これらのタスクはメッセージシステムを使用したスケジューラを介して動作します。

USB-BASIC-F/W は、ペリフェラルドライバ（PCD）から構成されています。PDCD は USB-BASIC-F/W の一部ではなくクラスドライバです。

クラスやベンダ固有リクエストの発行が必要な場合をはじめ、通信速度、プログラム容量等を考慮する場合、さらにユーザインタフェースを個別に設定する場合には、お客様にてカスタマイズして頂く必要があります。

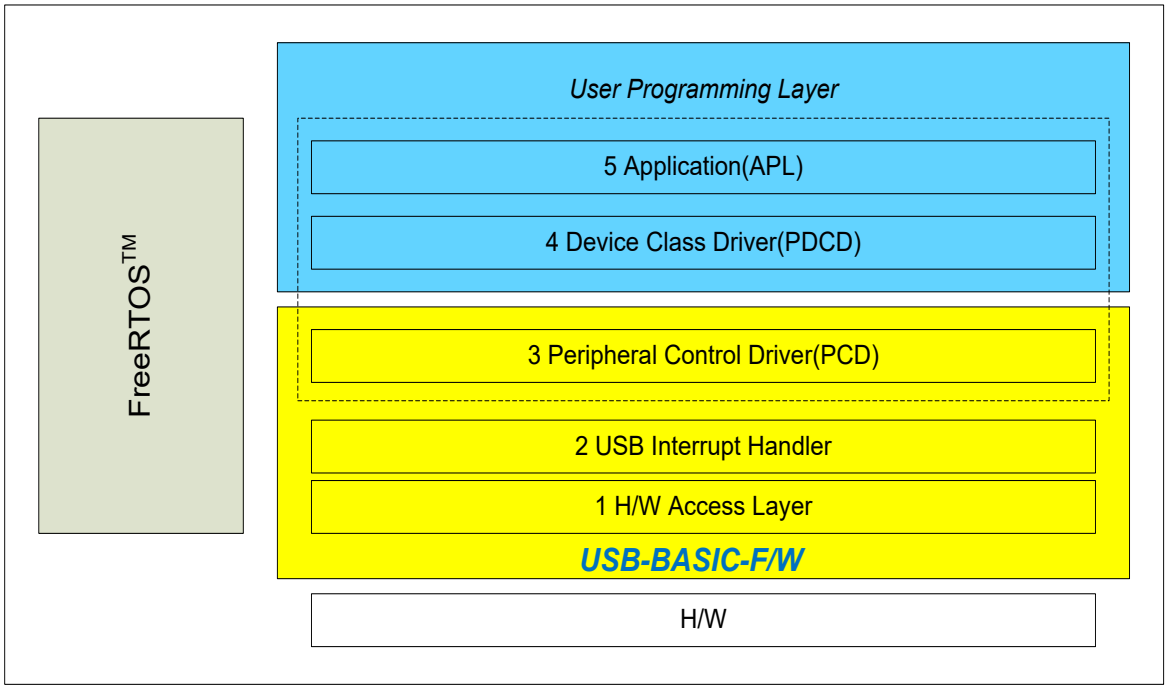

図 1.1 USB-BASIC-F/W のモジュール構成図

表 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	Device Class Driver	システムにあわせてご用意ください
5	Application	システムにあわせてご用意ください

1.5 スケジューラ機能

本モジュールは、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。

1.6 端子設定

USB モジュールを使用するためには、汎用入出力ポート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. 動作確認条件

本アプリケーションノートのサンプルコードは、下記の条件で動作を確認しています。

表 2.1 動作確認条件

項目	内容
使用マイコン	RZ/A2M
動作周波数（注）	CPU クロック（Iφ）：528MHz 画像処理クロック（Gφ）：264MHz 内部バスクロック（Bφ）：132MHz 周辺クロック 1（P1φ）：66MHz 周辺クロック 0（P0φ）：33MHz QSPI0_SPCLK：66MHz CKIO：132MHz
動作電圧	電源電圧（I/O）：3.3V 電源電圧（1.8/3.3V 切替 I/O（PVcc_SPI））：3.3V 電源電圧（内部）：1.2V
統合開発環境	e2 studio V7.8.0
C コンパイラ	GNU Arm Embedded Toolchain 6.3.1
	コンパイラオプション（ディレクトリパスの追加は除く） Release: -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Os -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -Wstack-usage=100 -fabi-version=0 Hardware Debug: -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Og -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -g3 -Wstack-usage=100 -fabi-version=0
動作モード	ブートモード 3 （シリアルフラッシュブート 3.3V 品）
ターミナルソフトの通信設定	• 通信速度：115200bps • データ長：8 ビット • パリティ：なし • ストップビット長：1 ビット • フロー制御：なし
使用ボード	RZ/A2M CPU ボード　RTK7921053C00000BE RZ/A2M SUB ボード　RTK79210XXB00000BE
使用デバイス （ボード上で使用する機能）	• シリアルフラッシュメモリ（SPI マルチ I/O バス空間に接続） メーカ名：Macronix 社、型名：MX25L51245GXD • RL78/G1C（USB 通信とシリアル通信を変換し、ホスト PC との通信に使用） • LED1

【注】 クロックモード 1（EXTAL 端子からの 24MHz のクロック入力）で使用時の動作周波数です。

3. ソフトウェア概要

3.1 ペリフェラルコントロールドライバ（PCD）

3.1.1 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

3.1.2 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。

3.1.3 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、「3.4 クラスリクエスト」を参照してください。

3.2 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。

3.2.1 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

3.2.2 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_dmaca_rz (DMA 転送使用時)

3.2.3 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。

表 3.1 動作確認環境

項目	内容
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

3.2.4 使用する割り込み ID

このドライバが使用する割り込み ID を以下に示します。

表 3.2 使用する割り込み ID 一覧

USB2.0 ファンクション モジュール	要求要因名	割り込み ID	割り込み概要
チャネル 0	USBFI0	64	INTSTS0 レジスタ要因 割り込み
	USBFDMA00	65	DMA チャネル 0 トランザクション終了 割り込み
	USBFDMA01	66	DMA チャネル 1 トランザクション終了割り込み
	USBFDMAERR0	67	DMA 転送エラー割り込み（全 DMA チャネルで共通）
チャネル 1	USBFI1	69	INTSTS0 レジスタ要因 割り込み
	USBFDMA10	70	DMA チャネル 0 トランザクション終了 割り込み
	USBFDMA11	71	DMA チャネル 1 トランザクション終了割り込み
	USBFDMAERR1	72	DMA 転送エラー割り込み（全 DMA チャネルで共通）

3.2.5 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

3.2.6 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。

3.2.7 コンパイル時の設定

コンパイル時の設定については、「7. 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。

3.2.8 引数

API 関数の引数に使用される構造体については、「7.4. 構造体」を参照してください。

3.3 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。

3.4 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、「8. クラスリクエスト」を参照してください。

3.5 Descriptor

3.5.1 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 p_string に設定してください。usb_descriptor_t 構造体については、「7.8. 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 Descirptor 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。

3.5.2 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 Descirptor 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、「7.8. usb_descriptor_t 構造体」を参照してください。

3.6 ペリフェラルバッテリチャージング制御

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、「4.9 R_USB_GetInformation」を参照してください。

PBC の処理フローを以下に示します。

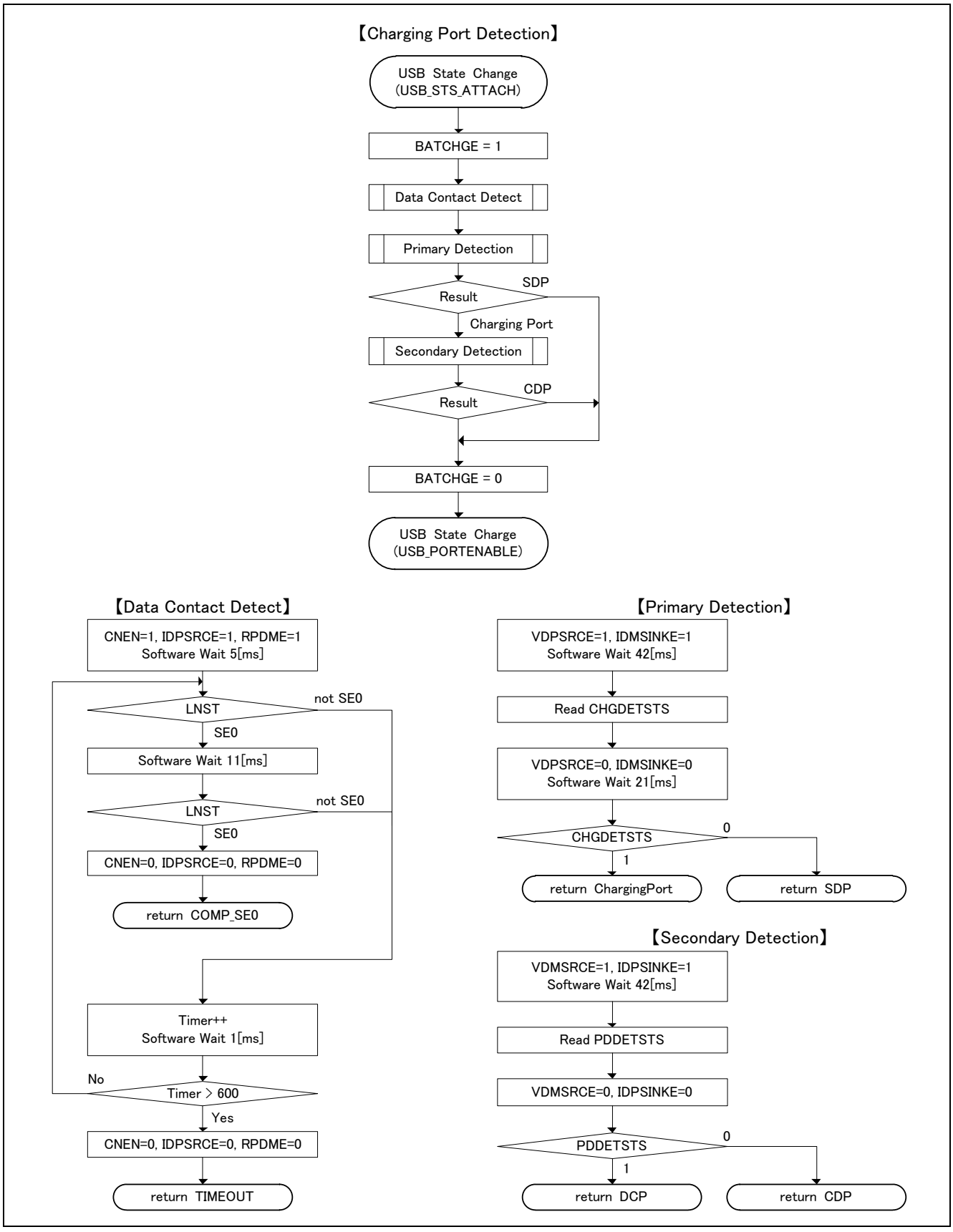

図 3.1　PBC フローチャート

4. API

表 4.1 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。

表 4.1 API 一覧

API		説明
R_USB_Open()	(Note1)	USB モジュール起動
R_USB_Close()	(Note1)	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)	USB データリード要求
R_USB_Write()	(Note1)	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_Resume()	(Note1)	レジューム要求
R_USB_GetEvent()	(Note1)	USB 関連の完了イベントを取得
R_USB_GetInformation()		USB デバイスについての情報を取得
R_USB_PipeRead()	(Note1)	指定 PIPE からのデータリード要求
R_USB_PipeWrite()	(Note1)	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。

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、「7. コンフィグレーション (r_usb_basic_config.h)」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

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。

補足

1. usb_ctrl_t 構造体については 7.5 章を、usb_cfg_t 構造体については 7.7 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(6 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。
5. usb_cfg_t 構造体のメンバ usb_mode には、"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。

使用例

```
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

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。

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。

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
   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 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. Control 転送の場合
   「8. クラスリクエスト」を参照してください。

リエントラント

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。

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. 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。データリードの完了は R_USB_GetEvent 関数の戻り値(USB_STS_READ_COMPLETE)により確認できます。
3. 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。
4. 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(6 章参照)を指定してください。
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. 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 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
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、「10.3 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。

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. 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。
10. Vendor クラスの場合、R_USB_PipeRead 関数をご使用ください。
11. 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。
12. 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_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_PCDC;
                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

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 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. Control 転送の場合

「8. クラスリクエスト」を参照してください。

リエントラント

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。

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. 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。データライトの完了は R_USB_GetEvent 関数の戻り値(USB_STS_WRITE_COMPLETE)により確認できます。
3. 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。
4. 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。
5. 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。
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. 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。
8. 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。
9. Vendor クラスの場合、R_USB_PipeWrite 関数をご使用ください。
10. 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。

11. 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_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_PCDC;
                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

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。

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。
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 2 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。
5. 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。
6. 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。
7. 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。
8. Vendor クラスの場合、R_USB_PipeStop 関数をご使用ください。
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。

使用例

```
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_PCDC;
                R_USB_Stop(&ctrl, USB_READ );   /* 受信停止*/
                R_USB_Stop(&ctrl, USB_WRITE );  /* 送信停止*/
                  :
            break;
                  :
        }
    }
}
```

4.7 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_NOT_SUSPEND	USB デバイスが SUSPEND 状態ではない

解説

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。

リエントラント

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。

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. 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。
3. 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。
4. 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。
5. 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。

使用例

1. HID(USB Peripheral)デバイスの場合

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

4.8 R_USB_GetEvent

USB 関連の完了イベントを取得する。

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 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. R_USB_GetEvent 関数の戻り値」を参照してください。
3. 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。
4. 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。

使用例

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

4.9 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、「7.10 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. 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、第 1 引数 p_ctrl に対し USB_NULL を指定してください。
4. 第 2 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

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

4.10 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

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、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。

リエントラント

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。

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. 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。データリードの完了は R_USB_GetEvent 関数の戻り値(USB_STS_READ_COMPLETE)により確認できます。
3. 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。
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。
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. 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 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
   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、「10.3 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。
11. 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。
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_Read 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_PVND_USE 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_PVND_CFG 定義については、「7. 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。

使用例

```
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;
                    :
        }
    }
}
```

4.11 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

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、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。

リエントラント

本 API は異なる PIPE に対して再入可能（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. 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。データライトの完了は R_USB_GetEvent 関数の戻り値(USB_STS_WRITE_COMPLETE)により確認できます。
3. 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。
4. 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。
5. 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。
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 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。
8. 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。
9. 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。
10. 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。

11. 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。
12. 本 API を使用する場合、r_usb_basic_config.h ファイル内の USB_CFG_PVND_CFG の定義を有効にしてください。こ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_PVND_CFG 定義については、「7. 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. 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_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;
                    :
        }
    }
}
```

4.12 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_BUSY	同じデバイスに対する停止要求中
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

本 API は異なる PIPE に対して再入可能（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 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. 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。
5. 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。
6. 本 API を使用する場合、r_usb_basic_config.h ファイル内の USB_CFG_PVND_CFG の定義を有効にしてください。こ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_PVND_CFG 定義については、「7. コンフィグレーション (r_usb_basic_config.h)」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_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;
                  :
        }
    }
}
```

4.13 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. 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。
2. USB Peripheral モードの場合は、第 1 引数 p_ctrl に対し USB_NULL を指定してください。
3. ビットマップ情報の b0(PIPE0)には、必ず"1"がセットされます。
4. 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_application( void )
{
    uint16_t     usepipe;
    usb_ctrl_t   ctrl;

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

4.14 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. 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、メンバ address および module に対する設定は必要ありません。
4. usb_pipe_t 構造体については、「7.9. usb_pipe_t 構造体」を参照してください。
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

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

5. R_USB_GetEvent 関数の戻り値

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。

戻り値	内容	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_NONE	USB 関連のイベントが無い状態。	○

○:サポート ×:非サポート

5.1 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で R_USB_GetEvent 関数をコールすると戻り値 USB_STS_DEFAULT が返されます。

5.2 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、アプリケーションプログラムが USB Host との USB データ通信を行える状態になった後で、R_USB_GetEvent 関数をコールすると戻り値 USB_STS_CONFIGURED が返されます。

5.3 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で R_USB_GetEvent 関数をコールすると戻り値 USB_STS_SUSPEND が返されます。

5.4 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で R_USB_GetEvent 関数をコールすると戻り値 USB_STS_RESUME が返されます。

5.5 USB_STS_DETACH

USB Host から USB デバイスを Detach した後で R_USB_GetEvent 関数をコールすると戻り値 USB_STS_DETACH が返されます。

5.6 USB_STS_REQUEST

USB デバイスが USB リクエスト(Setup)を受信した後で R_USB_GetEvent 関数をコールすると戻り値 USB_STS_REQUEST が返されます。この戻り値では、usb_ctrl_t 構造体の以下のメンバにも情報が設定されます。

メンバ	内容
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。
2. 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、「7.6. usb_setup_t 構造体」を参照してください。

5.7 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後で R_USB_GetEvent 関数をコールすると戻り値 USB_STS_REQUEST_COMPLETE が返されます。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。

メンバ	内容
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、「7.6 usb_setup_t 構造体」を参照してください。

5.8 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後で R_USB_GetEvent 関数をコールすると戻り値 USB_STS_READ_COMPLETE が返されます。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。

メンバ	内容
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. 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。
2. デバイスクラス種別については、「6. デバイスクラス種別」を参照してください。
3. 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。

ステータス	内容
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。

5.9 USB_STS_WRITE_COMPLETE

R_USB_Write 関数によるデータライトが完了した後で R_USB_GetEvent 関数をコールすると戻り値 USB_STS_WRITE_COMPLETE が返されます。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。

メンバ	内容
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、「6. 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。

ステータス	内容
USB_SUCCESS	データライト正常終了
USB_ERR_NG	データ送信失敗

5.10 USB_STS_NONE

USB 関連のイベントが無い状態で R_USB_GetEvent 関数をコールすると戻り値 USB_STS_NONE が返されます。この戻り値では、usb_ctrl_t 構造体の以下メンバにも情報が設定されます。

メンバ	内容
status	USB デバイスのステータス

6. 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_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_PVND	Peripheral Vendor Class

[Note]

1. 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。

7. コンフィグレーション

7.1 SmartConfigurator 設定(r_usbf_basic_drv_sc_cfg.h)

下記の定義に対する指定を SmartConfigurator で行ってください。

1. OS/非 OS 選択
   OSLess/FreeRTOS を選択してください。

```
#define     BSP_CFG_RTOS_USED   0     // OSLess
#define     BSP_CFG_RTOS_USED   1     // FreeRTOS
```

2. LPM 選択
   LPM 使用/非使用を選択してください。

```
#define     USE_LPW     0     // LPM 非使用
#define     USE_LPW     1     // LPM 使用
```

3. USB モジュール選択設定
   使用する USB モジュール番号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
```

7.2 USB Peripheral Basic 設定(r_usb_basic_config.h)

下記の定義に対する指定を行ってください。

1. 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
```

2. 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_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
```

3. 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、下記 4 の定義において DMA Channel 番号を指定してください。

4. DMA Channel 設定

上記 3 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_CH1 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。

5. CPU バスウェイト設定

SYSCFG1 レジスタの BWAIT ビットに設定する数値を 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、ハードウェアマニュアル内の USB2.0 ファンクションモジュール SYSCFG1 レジスタ BWAIT ビットを参照してください。

6. 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)
```

7. 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. クラスリクエスト設定

受信したクラスリクエストのサポート/非サポートを設定します。USB_CFG_ENABLE(サポート)を指定した場合は、USB ドライバはクラスリクエストの受信をアプリケーションプログラムに通知します。USB_CFG_DISABLE(非サポート)を指定した場合は、USB ドライバはクラスリクエストに対し STALL 応答します。

```
#define USB_CFG_CLASS_REQUEST      USB_CFG_ENABLE      // サポート
#define USB_CFG_CLASS_REQUEST      USB_CFG_DISABLE     // 非サポート
```

[Note]

(1). クラスリクエストの受信は R_USB_GetEvent 関数の戻り値 USB_STS_REQUST により確認できます。
(2). Mass Storage Class のクラスリクエスト GetMaxLun に対しては、USB_CFG_DISABLE を指定しても値"1"を返します。

7.3 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

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。

8. 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。

8.1 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 構造体は表 4-1 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 */
    uint16_t     status;      /* Note 5 */
    uint32_t     size;        /* Note 6 */
    usb_setup_t  setup;       /* Note 7 */
} 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、「8.1.5 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(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。
7. 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。

8.2 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 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 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 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 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 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。

8.3 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 Peripheral として使用する設定や 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。表 4-1 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 Peripheral として使用するため、このメンバ(usb_mode)に"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、「7.8. 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。

8.4 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      **p_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(p_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。

8.5 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。

8.6 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 Peripheral モード時はメンバ(type)に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、「6. 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。
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。

ステータス	内容
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。

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

8.7 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。

ステータス	内容
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。

9. 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。

9.1 USB Peripheral モードの場合

9.1.1 USB リクエスト(Setup)

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、「7.6 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。

9.1.2 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]

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。

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]

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。

9.1.3 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、「8.1.5 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。

9.1.4 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

```
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. 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

```
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;
    }
}
```

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

```
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;
    }
}
```

9.1.5 クラスリクエストに対する ACK/STALL 応答処理

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。

1. STALL 応答の処理例

```
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. ACK 応答の処理例

```
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;
    }
}
```

10. DMA 転送

10.1 基本仕様

USB-BASIC-F/W の DMA 転送サンプルプログラムの仕様を以下に示します。DMA 転送が可能な PIPE 番号は PIPE1 と PIPE2 です。表 9.1 に DMA 設定仕様を示します。

表 9.1　DMA 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	シングル転送モード
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
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

10.2 注意事項

10.2.1 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
   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、「10.3 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。

10.2.2 USB PIPE について

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

なお、DMA 転送と CPU 転送を組み合わせてデータ転送を行う場合、DMA 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. 注意事項

11.1 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。

11.2 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、「7.11 usb_compliance_t 構造体」を参照してください。
3. usb_compliance_disp 関数の参考プログラム例は、「11.1 usb_compliance_disp 関数」を参照してください。

11.3 PIPEBUF レジスタの参照/変更について

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
Peripheral Communication Device Class Peripheral Human I/F Device Class Periphral Mass Storage Class	r_usb_peptable.c	g_usb_eptbl

11.4 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。

11.5 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、「7. コンフィグレーション (r_usb_basic_config.h)」を参照してください。

11.6 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。

12. アプリケーションプログラム作成

12.1 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. 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。

12.2 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。

表 10.1 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

[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. DMA 関連初期化

DMA 転送を行う場合は、DMA 初期化関数をコールしてください。

転送	関数
DMA	R_DMACA_Init R_DMACA_Open

[Note]

(1). DMA 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。「7. 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 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 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。

12.3 Descriptor の作成

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「3.5 Descriptor」を参照してください。

12.4 メインルーチン

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]

1. 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。

12.5 アプリケーションプログラム記述例（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 Peri 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;
        }
    }
}
```

12.6 アプリケーションプログラム記述例（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 Peri 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;
        }
    }
}
```

13. 参考プログラム例

13.1 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。

14. 参考ドキュメント

ユーザーズマニュアル：ハードウェア

RZ/A2M グループ ユーザーズマニュアル ハードウェア編
（最新版をルネサス エレクトロニクスホームページから入手してください。）

RTK7921053C00000BE（RZ/A2M CPU ボード）ユーザーズマニュアル
（最新版をルネサス エレクトロニクスホームページから入手してください。）

RTK79210XXB00000BE（RZ/A2M SUB ボード）ユーザーズマニュアル
（最新版をルネサス エレクトロニクスホームページから入手してください。）

Arm Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C
（最新版を Arm ホームページから入手してください。）

Arm Cortex™-A9 Technical Reference Manual Revision: r4p1
（最新版を Arm ホームページから入手してください。）

Arm Generic Interrupt Controller Architecture Specification - Architecture version2.0
（最新版を Arm ホームページから入手してください。）

Arm CoreLink™ Level 2 Cache Controller L2C-310 Technical Reference Manual Revision: r3p3
（最新版を Arm ホームページから入手してください。）

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

ユーザーズマニュアル：統合開発

統合開発環境 e2 studio のユーザーズマニュアルは、ルネサス エレクトロニクスホームページから入手してください。
（最新版をルネサス エレクトロニクスホームページから入手してください。）

ホームページとサポート窓口

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

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

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2019 年 9 月 30 日	ー	初版
1.10	2019 年 12 月 17 日		表 2.1　動作確認条件 コンパイラオプション"-mthumb-interwork"を削除 FreeRTOS/OSLess 双方のサポート
1.20	2020 年 6 月 30 日	P44	LPM 設定追加

製品ご使用上の注意事項

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

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

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

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

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

(Rev.4.0-1　2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

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

www.renesas.com/contact/

商標について

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