RE01 1500KBグループ

R01AN4944JJ0100
Rev.1.00
Sep 30, 2019

USB Basic Peripheral Driver Specification

要旨

本書では、CMSIS software package の USB Basic Peripheral Driver (以下、USB ドライバ)の仕様を説明します。

動作確認デバイス

RE01 1500KB グループ

目次

1. 概要

本ドライバは、USB の H/W 制御を行い、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。以下に本ドライバがサポートしている機能を以下に示します。

- CMSIS Driver 仕様をサポートしています。
- 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
- 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。

1.2 制限事項

本ドライバ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. マルチインターフェースをサポートしていません。
3. 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。
4. 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

APL	:	Application program
H/W	:	Renesas USB device
Non-OS	:	USB Driver for OS less
PCD	:	Peripheral Control Driver for USB-BASIC-FW
PDCD	:	Peripheral Device Class Driver (Device Driver and USB Class Driver)
USB-BASIC-FW	:	USB Basic Peripheral Driver

1.4 ソフトウェア構成

1.4.1 モジュール構成

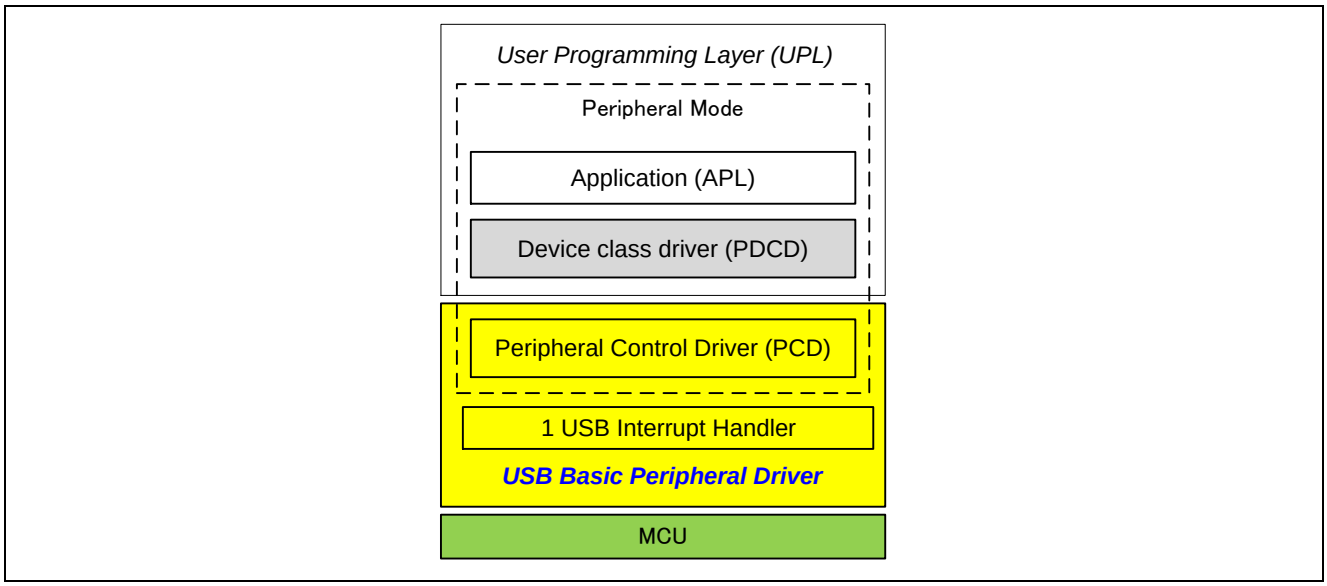

Figure 1-1 USB ドライバ ソフトウェア構成図

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
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

1.5 端子設定

本ドライバ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。

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

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、「**3. API**」を参照してください。

2.2 API 関数

API 関数については、「**3. API**」を参照してください。

2.3 USB リクエスト処理

USB リクエストの処理手順については、「8.3 Enumeration (リクエスト)処理」を参照してください。

2.4 Descriptor

2.4.1 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 */
};
```

2.4.2 その他の Descriptor

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。

2.4.3 Descriptor の送信

USB Host から送信される GetDescriptor コマンドを受信したら、GetDescriptor コマンドを解析し、Descriptor Type を取得してください。その取得した Decriptor Type に該当する Descriptor を ARM_USBD_EndpointTransfer 関数(API)を使って USB Host に送信します。

記述例) Device Descriptor の場合

```
ARM_USBD_EndpointTransfer ((USB_EP_IN | USB_EP0), g_device_descriptor, 18);
```

1. 第 1 引数には、Endpoint 番号 0 を指定しください。
2. 第 2 引数には、該当の Descirptor を格納した領域の先頭アドレスを指定してください。String Descriptor の場合は、String Descriptor テーブルの先頭アドレスを指定してください。
3. 第 3 引数には、該当の Descriptor のバイト数を指定してください。

3. API

3.1 API 一覧

Table3-1に API 関数一覧を示します。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。

Table3-1 API 一覧

API	説明
ARM_USBD_Initialize	USB ドライバの初期化
ARM_USBD_Uninitialize	USB ドライバの解放
ARM_USBD_PowerControl	USB モジュールストップビットのクリアまたはセット処理
ARM_USBD_DeviceConnect	D+ラインプルアップ許可設定
ARM_USBD_DeviceDisConnect	D+ラインプルアップ禁止設定
ARM_USBD_DeviceGetState	USB デバイス状態取得
ARM_USBD_DeviceRemoteWakeup	RemoteWakeup 設定
ARM_USBD_ReadSetupPacket	Setup パケット読み出し
ARM_USBD_EndpointConfigure	指定 Endpoint のコンフィグレーション
ARM_USBD_EndpointUnconfigure	指定 Endpoint のコンフィグレーション解除
ARM_USBD_EndpointStall	指定 Endpoint の Stall 許可/禁止
ARM_USBD_EndpointTransfer	指定 Endpoint のデータ転送要求
ARM_USBD_EndpointTransferGetResult	指定 Endpoint の受信データサイズ取得
ARM_USBD_EndpointTransferAbort	指定 Endpoint のデータ転送停止要求
ARM_USBD_GetFrameNumber	Frame 番号取得
ARM_USBD_GetCapabilities	構造体 ARM_USBD_CAPABILITIES の設定情報取得
ARM_USBD_GetVersion	USB ドライババージョン番号取得
R_USB_ControlDriver	USB ドライバ処理を実行

[Note]

1. USB モジュールが SetAddress リクエストに対する自動応答機能をサポートしているため、ARM_USBD_DeviceSetAddress 関数はサポートされていません。
2. 構造体 ARM_USBD_STATE/ARM_USBD_CAPABILITIES および R_USB_ControlDriver 関数以外の API の詳細については、以下の URL を参照してください。

   https://arm-software.github.io/CMSIS_5/Driver/html/group__usbd__interface__gr.html

3.2 ARM_USBD_Initialize 関数

Table 3-2 ARM_USBD_Initialize 関数仕様

書式	int32_t ARM_USBD_Initialize(ARM_USBD_SignalDeviceEvent_t cb_device_event, ARM_USBD_SignalEndpointEvent_ cb_endpoint_event)
仕様説明	USB ドライバの初期化 (RAM 初期化、レジスタ設定、コールバック関数の登録等)
引数	ARM_USBD_SignalDeviceEvent_t : コールバック関数 ARM_USBD_SignalEndpointEvent_t : コールバック関数
戻り値	ARM_DRIVER_OK : 正常終了

備考	[Note]
	割り込み関数内で本 API をコールしないでください。
	
	[インスタンスからの関数呼び出し例]
	static void pmsc_signal_device_event(uint32_t event);
	static void pmsc_signal_endpoint_event (uint8_t ep_addr, uint32_t ep_event);
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->Initialize (&pmsc_signal_device_event,
	&pmsc_signal_endpoint_event);
	}

3.3 ARM_USBD_Uninitialize 関数

Table 3-3 ARM_USBD_Uninitialize 関数仕様

書式	int32_t ARM_USBD_UnInitialize(void)
仕様説明	USB ドライバの解放
引数	なし
戻り値	ARM_DRIVER_OK : 正常終了

備考	[Note]
	割り込み関数内で本 API をコールしないでください。
	
	[インスタンスからの関数呼び出し例]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->Uninitialize ();
	}

3.4 ARM_USBD_PowerControl 関数

Table 3-4 ARM_USBD_PowerControl 関数仕様

書式	int32_t ARM_USBD_PowerControl(ARM_POWER_STATE state)
仕様説明	USB モジュールストップビットのクリアまたはセット処理
引数	ARM_POWER_FULL : USB モジュールストップビットクリア ARM_POWER_OFF : USB モジュールストップビットセット
戻り値	ARM_DRIVER_OK : 正常終了 ARM_DRIVER_ERROR_UNSUPPORTED : 非サポートエラー

備考	[Note]
	割り込み関数内で本 API をコールしないでください。
	
	[インスタンスからの関数呼び出し例]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->PowerControl(ARM_POWER_FULL);
	}

3.5 ARM_USBD_DeviceConnect 関数

Table 3-5 ARM_USBD_DeviceConnect 関数仕様

書式	int32_t ARM_USBD_DeviceConnect(void)
仕様説明	D+ラインプルアップ許可設定
引数	なし
戻り値	ARM_DRIVER_OK : 正常終了

備考	[Note]
	割り込み関数内で本 API をコールしないでください。
	
	[インスタンスからの関数呼び出し例]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->DeviceConnect ();
	}

3.6 ARM_USBD_DeviceDisconnect 関数

Table 3-6 ARM_USBD_DeviceDisconnect 関数仕様

書式	int32_t ARM_USBD_DeviceDisconnect(void)
仕様説明	D+ラインプルアップ禁止設定
引数	なし
戻り値	ARM_DRIVER_OK : 正常終了

備考	[Note]
	割り込み関数内で本 API をコールしないでください。
	
	[インスタンスからの関数呼び出し例]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->DeviceDisconnect ();
	}

3.7 ARM_USBD_DeviceGetState 関数

Table 3-7 ARM_USBD_DeviceGetState 関数仕様

書式	ARM_USBD_STATE ARM_USBD_DeviceGetState(void)
仕様説明	USB デバイス状態取得
引数	なし
戻り値	USB デバイス状態

備考	[インスタンスからの関数呼び出し例]
	
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	ARM_USBD_STATE usb_state;
	usb_state = gp_cmsis_usb_driver->DeviceGetState ();
	}

3.8 ARM_USBD_DeviceRemoteWakeup 関数

Table 3-8 ARM_USBD_DeviceRemoteWakeup 関数仕様

書式	int32_t ARM_USBD_DeviceRemoteWakeup(void)
仕様説明	Remote Wakeup 設定
引数	なし
戻り値	ARM_DRIVER_OK : 正常終了 ARM_DRIVER_ERROR_BUSY : Remote Wakeup 処理中 ARM_DRIVER_ERROR : その他のエラー

備考	[Note]
	割り込み関数内で本 API をコールしないでください。
	
	[インスタンスからの関数呼び出し例]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->DeviceRemoteWakeup ();
	}

3.9 ARM_USBD_ReadSetupPacket 関数

Table 3-9 ARM_USBD_ReadSetupPacket 関数仕様

書式	int32_t ARM_USBD_ReadSetupPacket(uint8_t *setup)
仕様説明	Setup パケット読み出し
引数	Setup パケット格納領域へのポインタ
戻り値	ARM_DRIVER_OK : 正常終了

備考	[Note]
	割り込み関数内で本 API をコールしないでください。
	
	[インスタンスからの関数呼び出し例]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->ReadSetupPacket (&usb_setup);
	}

3.10 ARM_USBD_EndpointConfigure 関数

Table 3-10 ARM_USBD_EndpointConfigure 関数仕様

書式	int32_t ARM_USBD_EndpointConfigure(uint8_t ep_addr, uint8_t ep_type, uint16_t ep_maxpacket_size)
仕様説明	指定 Endpoint のコンフィグレーション
引数	Endpoint アドレス b7: 転送方向(1: IN, 0: OUT), b6-b0: Endpoint 番号
	転送タイプ ARM_USB_ENDPOINT_BULK/ARM_USB_ENDPOINT_INTERRUPT
	MaxPacketSize
戻り値	ARM_DRIVER_OK : 正常終了 ARM_DRIVER_ERROR_PARAMETER : パラメータエラー

備考	[Note]
	割り込み関数内で本 API をコールしないでください。
	
	[インスタンスからの関数呼び出し例]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->EndpointConfigure (
	USB_EP_IN|USB_EP2,
	ARM_USB_ENDPOINT_BULK,
	64
	);
	}

3.11 ARM_USBD_EndpointUnconfigure 関数

Table 3-11 ARM_USBD_EndpointUnconfigure 関数仕様

書式	int32_t ARM_USBD_EndpointUnconfigure(uint8_t ep_addr)
仕様説明	指定 Endpoint のコンフィグレーション解除
引数	Endpoint アドレス b7: 転送方向(1: IN, 0: OUT), b6-b0: Endpoint 番号
戻り値	ARM_DRIVER_OK : 正常終了 ARM_DRIVER_ERROR_PARAMETER : パラメータエラー

備考	[Note]
	割り込み関数内で本 API をコールしないでください。
	
	[インスタンスからの関数呼び出し例]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->EndpointUnonfigure (USB_EP_IN|USB_EP2);
	}

3.12 ARM_USBD_EndpointStall 関数

Table 3-12 ARM_USBD_EndpointStall 関数仕様

書式	int32_t ARM_USBD_EndpointStall(uint8_t ep_addr, bool stall)
仕様説明	指定 Endpoint の Stall 許可/禁止
引数	Endpoint アドレス b7: 転送方向(1: IN, 0: OUT), b6-b0: Endpoint 番号 false (Clear) / true (Set)
戻り値	ARM_DRIVER_OK : 正常終了 ARM_DRIVER_ERROR_PARAMETER : パラメータエラー

備考	[Note]
	割り込み関数内で本 API をコールしないでください。
	
	[インスタンスからの関数呼び出し例]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->EndpointStall (USB_EP_IN|USB_EP2, true);
	}

3.13 ARM_USBD_EndpointTransfer 関数

Table 3-13 ARM_USBD_EndpointTransfer 関数仕様

書式	int32_t ARM_USBD_EndpointTransfer(uint8_t ep_addr, uint8_t *data, uint32_t num)
仕様説明	指定 Endpoint のデータ転送要求
引数	Endpoint アドレス b7: 転送方向(1: IN, 0: OUT), b6-b0: Endpoint 番号 転送データ格納領域へのポインタ 転送データサイズ
戻り値	ARM_DRIVER_OK : 正常終了 ARM_DRIVER_ERROR_BUSY : データ転送要求中 ARM_DRIVER_ERROR_PARAMETER : パラメータエラー ARM_DRIVER_ERROR_BUSY : その他のエラー

備考	[Note]
	割り込み関数内で本 API をコールしないでください。
	
	[インスタンスからの関数呼び出し例]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	uint8_t g_buf[64];
	
	void main(void)
	{
	gp_cmsis_usb_driver->EndpointTransfer (USB_EP_IN|USB_EP2, g_buf, 64);
	}

3.14 ARM_USBD_EndpointTransferGetResult 関数

Table 3-14 ARM_USBD_EndpointTransferGetResult 関数仕様

書式	uint32_t ARM_USBD_EndpointTransferGetResult(uint8_t ep_addr)
仕様説明	指定 Endpoint のデータ転送サイズ取得
引数	Endpoint アドレス b7: 転送方向(1: IN, 0: OUT), b6-b0: Endpoint 番号
戻り値	ARM_DRIVER_OK : 正常終了 ARM_DRIVER_ERROR_PARAMETER : パラメータエラー

備考	[Note]
	1. ARM_USB_EndpointTransferAbort 関数によりデータ転送を停止した場合、本 API は、データ転送サイズを取得できません。
	2. 割り込み関数内で本 API をコールしないでください。
	
	[インスタンスからの関数呼び出し例]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	uint32_t usb_size;
	usb_size = gp_cmsis_usb_driver->EndpointTransferGetResult (USB_EP_OUT|USB_EP1);
	}

3.15 ARM_USBD_EndpointTransferAbort 関数

Table 3-15 ARM_USBD_EndpointTransferAbort 関数仕様

書式	int32_t ARM_USBD_EndpointTransferAbort(uint8_t ep_addr)
仕様説明	指定 Endpoint のデータ転送停止要求
引数	Endpoint アドレス b7: 転送方向(1: IN, 0: OUT), b6-b0: Endpoint 番号
戻り値	ARM_DRIVER_OK : 正常終了 ARM_DRIVER_ERROR_PARAMETER : パラメータエラー

備考	[Note]
	割り込み関数内で本 API をコールしないでください。
	
	[インスタンスからの関数呼び出し例]
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	gp_cmsis_usb_driver->EndpointTransferAbort(USB_EP_IN|USB_EP2);
	}

3.16 ARM_USBD_GetFrameNumber 関数

Table 3-16 ARM_USBD_GetFrameNumber 関数仕様

書式	uint16_t ARM_USBD_GetFrameNumber(void)
仕様説明	Frame 番号取得
引数	なし
戻り値	Frame 番号

備考	[インスタンスからの関数呼び出し例]
	
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	uint16_t frame_number;
	frame_number = gp_cmsis_usb_driver->FrameNumber();
	}

3.17 ARM_USBD_GetCapabilities 関数

Table 3-17 ARM_USBD_GetCapabilities 関数仕様

書式	ARM_USBD_CAPABILITIES ARM_USBD_GetCapabilities(void)
仕様説明	USB ドライバ機能取得
引数	なし
戻り値	ドライバ機能 (ARM_USBD_CAPABILITIES 構造体)

備考	[インスタンスからの関数呼び出し例]
	
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	ARM_USBD_CAPABILITIES capabilities;
	capabilities = gp_cmsis_usb_driver->GetCapabilities();
	}

3.18 ARM_USBD_GetVersion 関数

Table 3-18 ARM_USBD_GetVersion 関数仕様

書式	ARM_DRIVER_VERSION ARM_USBD_GetVersion(void)
仕様説明	ドライババージョン番号取得
引数	なし
戻り値	ドライババージョン番号 (ARM_DRIVER_VERSION 構造体)

備考	[インスタンスからの関数呼び出し例]
	
	ARM_DRIVER_USBD *gp_cmsis_usb_driver= &Driver_USBD0;
	
	void main(void)
	{
	ARM_DRIVER_VERSION version;
	version = gp_cmsis_usb_driver->GetVersion();
	}

3.19 R_USB_ControlDriver 関数

Table 3-19 R_USB_ControlDriver 関数仕様

書式	void R_USB_ControlDriver(void)
仕様説明	USB ドライバ実行
引数	なし
戻り値	なし

備考	[Note]
	1. 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。
	2. 割り込み関数内で本 API をコールしないでください。
	
	[関数呼び出し例]
	void main(void)
	{
	while(1)
	{
	R_USB_ControlDriver();
	:
	}
	}

4. コールバック関数

USB イベントが完了した時、USB ドライバは、以下のコールバック関数をコールします。USB イベントについては、「**4.3 USBイベント**」を参照してください。

4.1 ARM_USBD_SignalDeviceEvent 関数

書式	void ARM_USBD_SignalDeviceEvent(uint32_t event)
仕様説明	USB イベントが完了した時にコールされる関数
引数	event USB 完了イベント ARM_USBD_EVENT_VBUS_ON ARM_USBD_EVENT_VBUS_OFF ARM_USBD_EVENT_RESET ARM_USBD_EVENT_SUSPEND ARM_USBD_EVENT_RESUME
戻り値	なし

4.2 ARM_USBD_SignalEndpointEvent 関数

書式	void ARM_USBD_SignalDeviceEvent(uint8_t ep_addr, uint32_t event)
仕様説明	USB イベントが完了した時にコールされる関数
引数	ep_addr USB イベントが完了した Endpoint アドレス b7: 転送方向(1: IN, 0: OUT), b6-b0: Endpoint 番号 event USB 完了イベント ARM_USBD_EVENT_SETUP ARM_USBD_EVENT_OUT ARM_USBD_EVENT_IN
戻り値	なし

4.3 USB イベント

USB イベントが完了すると USB ドライバは、ARM_USBD_SignalDeviceEvent 関数(コールバック関数)または ARM_USBD_SignalEndpointEvent 関数(コールバック関数)をコールします。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 event に設定されます。

4.3.1 ARM_USBD_SignalDeviceEvent 関数

以下の USB イベントを検出すると、USB ドライバは ARM_USBD_SignalDeviceEvent 関数をコールします。

引数 event	内容
ARM_USBD_EVENT_VBUS_ON	VBUS(Hi)検出
ARM_USBD_EVENT_VBUS_OFF	VBUS(Low)検出
ARM_USBD_EVENT_RESET	USB Reset 検出
ARM_USBD_EVENT_SUSPEND	SUSPEND 検出
ARM_USBD_EVENT_RESUME	RESUME 検出

4.3.2 ARM_USBD_SignalEndpointEvent 関数

以下の USB イベントが完了すると、USB ドライバは ARM_USBD_SignalEndpointEvent 関数をコールします。

引数 event	内容
ARM_USBD_EVENT_SETUP	Setup パケット受信完了
ARM_USBD_EVENT_OUT	OUT データ受信完了
ARM_USBD_EVENT_IN	IN データ送信完了

[Note]

ARM_USBD_EndpointTransferAbort 関数の引数に Endpoint0 を指定した場合、本コールバック関数はコールされません。

4.4 コールバック関数登録

アプリケーションプログラムでは、以下の 2 つのコールバック関数(ARM_USBD_SignalDeviceEvent 関数と ARM_USBD_SignalEndpointEvent 関数)を定義し、ARM_USBD_Initialize 関数を使って、これらのコールバック関数を USB ドライバに登録してください。

[登録例]

```
ARM_DRIVER_USBD    *gp_cmsis_usb_driver= &Driver_USBD0;

/*  コールバック関数(ARM_USBD_SignalDeviceEvent 関数) */
void usb_apl_signal_device_event(uint32_t event)
{
      :
}
/*  コールバック関数(ARM_USBD_SignalEndpointEvent 関数) */
void usb_apl_signal_endpoint_event(uint8_t ep_addr, uint32_t event)
{
      :
}

void main(void)
{
                        :
      /*  コールバック関数の登録  */
       gp_cmsis_usb_driver->Initialize ( usb_apl_signal_device_event, usb_apl_signal_endpoint_event);
                        :
}
```

5. 構造体

5.1 ARM_DRIVER_USB 構造体

型	メンバ
ARM_DRIVER_VERSION	(*GetVersion) (void)
ARM_USBD_CAPABILITIES	(*GetCapabilities) (void)
int32_t	(*Initialize) (ARM_USBD_SignalDeviceEvent_t cb_device_event ,ARM_USBD_SignalEndpointEvent_t cb_endpoint_event)
int32_t	(*Uninitialize) (void);
int32_t	(*PowerControl) (ARM_POWER_STATE state)
int32_t	(*DeviceConnect) (void)
int32_t	(*DeviceDisconnect) (void)
ARM_USBD_STATE	(*DeviceGetState) (void)
int32_t	(*DeviceRemoteWakeup) (void)
int32_t	(*DeviceSetAddress) (void)
int32_t	(*ReadSetupPacket) (uint8_t *setup)
int32_t	(*EndpointConfigure) (uint8_t ep_addr, uint8_t ep_type, uint16_t ep_max_packet_size)
int32_t	(*EndpointUnconfigure) (uint8_t ep_addr)
int32_t	(*EndpointStall) (uint8_t ep_addr)
int32_t	(*EndpointTransfer) (uint8_t ep_addr, uint8_t *data, uint32_t num)
uint32_t	(*EndpointTransferGetResult) (uint8_t ep_addr)
int32_t	(*EndpointTranferAbort) (uint8_t ep_addr)
uint16_t	(*GetFrameNumber) (void)

5.2 ARM_USBD_STATE 構造体

型	メンバ		内容
uint32_t	vbus:	1	VBUS 状態 (ON: 1, OFF:0)
uint32_t	speed:	2	USB デバイススピード (ARM_USBD_SPEED_FULL のみ)
uint32_t	active:	1	USB デバイス状態 (Configured 状態: 1, その他: 0)
uint32_t	reserved:	28	--

5.3 ARM_USBD_CAPABILITIES 構造体

型	メンバ		内容
uint32_t	vbus_detection:	1	VBUS 検出機能 (サポート:1, 非サポート: 0)
uint32_t	event_vbus_on:	1	アタッチイベント通知機能 (サポート:1, 非サポート: 0)
uint32_t	event_vbus_off:	1	デタッチイベント通知機能 (サポート:1, 非サポート: 0)
uint32_t	reserved:	29	--

6. マクロ定義

定義	値
ARM_USBD_EVENT_VBUS_ON	(1UL << 0)
ARM_USBD_EVENT_VBUS_OFF	(1UL << 1)
ARM_USBD_EVENT_VBUS_RESET	(1UL << 2)
ARM_USBD_EVENT_HIGH_SPEED	(1UL << 3)
ARM_USBD_EVENT_SUSPEND	(1UL << 4)
ARM_USBD_EVENT_RESUME	(1UL << 5)
ARM_USBD_EVENT_SETUP	(1UL << 0)
ARM_USBD_EVENT_OUT	(1UL << 1)
ARM_USBD_EVENT_IN	(1UL << 2)
ARM_USB_SPEED_LOW	0
ARM_USB_SPEED_FULL	1
ARM_USB_SPEED_HIGH	2
ARM_USB_ENDPOINT_CONTROL	0
ARM_USB_ENDPOINT_ISOCHRONOUS	1
ARM_USB_ENDPOINT_BULK	2
ARM_USB_ENDPOINT_INTERRUPT	3

7. コンフィグレーション (r_usb_basic_mini_cfg.h)

1. USB 動作モード設定

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. デバイスクラス設定

USB_CFG_PMSC_USE を定義してください。

```
#define     USB_CFG_PMSC_USE        // Peripheral Mass Storage Class
```

3. 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4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。

4. 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3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5の定義において DMA Channel 番号を指定してください。

5. DMA Channel 設定

上記4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
```

[Note]

(1). DMA Channel 番号には、USB_CFG_CH0 から USB_CFG_CH3 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。

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. 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。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。

```
#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
```

[Note]

(1). 上記の DBLB 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。

8. 関数の RAM 配置

USB ドライバ関数を RAM 上で実行するための設定です。

```
#define     USB_CFG_FUNC     SYSTEM_SECTION_CODE          // 関数を RAM に配置しない
#define     USB_CFG_FUNC     SYSTEM_SECTION_RAM_FUNC      // 関数を RAM に配置する
```

[Note]

USB_CFG_FUNC 定義に対し SYSTEM_SECTION_RAM_FUNC を指定した場合、USB ドライバの全関数が RAM に配置されます。

8. USB 動作処理概要および関連 API

8.1 アタッチ処理

USB デバイスを USB Host にアタッチ(接続)すると、USB ドライバは ARM_USBD_SignalDeviceEvent 関数(コールバック関数)をコールします。このコールバック関数の引数には、ARM_USBD_EVENT_VBUS_ON がセットされています。アプリケーションプログラムは、このコールバック関数によってアタッチを検出することができます。

アタッチ検出後、アプリケーションプログラムでは、D+ラインをプルアップするための ARM_USBD_DeviceConnect 関数をコールしてください。D+ラインがプルアップされると USB Host は、USB デバイスが接続されたと判断し、USB デバイスに対し、USB Reset を送信した後、Enumeration 処理を開始します。

関連 API:

ARM_USBD_SignalDeviceEvent 関数(event: ARM_USBD_EVENT_VBUS_ON)
ARM_USBD_DeviceConnet 関数

8.2 Default 状態検出

D+ラインがプルアップされると、USB Host は USB Reset を送信します。USB モジュールが USB Reset を受信すると、USB デバイスステートが Default 状態に遷移します。

Default 状態検出後、アプリケーションプログラムでは、Enumeration 処理を行うため Endpoint 番号 0 のコンフィグレーションを行う必要があります。Endpoint をコンフィグレーションするときは、ARM_USBD_EndpointConfigure 関数を使用してください。

[Note]

USB デバイスステートが Default 状態に遷移すると、USB ドライバは、ARM_USBD_SignalDeviceEvent 関数(コールバック関数)をコールします。このコールバック関数の引数には、ARM_USBD_EVENT_RESET がセットされています。アプリケーションプログラムは、このコールバック関数によって Default 状態を検出することができます。

関連 API:

ARM_USBD_SignalDeviceEvent 関数(event: ARM_USBD_EVENT_RESET)
ARM_USBD_EndpointConfigure 関数

8.3 Enumeration (リクエスト)処理

アプリケーションプログラムは、以下の処理により、USB Host との Enumeration 処理を行う必要があります。

8.3.1 Setup ステージ処理 (USB リクエスト受信)

USB Host から送信される USB リクエスト(Setup)を本ドライバが受信すると、コールバック関数(ARM_USBD_SignalEndpointEvent)がコールされます。受信した USB リクエスト(Setup:8 バイト)は、ARM_USBD_ReadSetupPacket 関数により取得することができます。

関連 API:

ARM_USBD_SignalEndpointEvent 関数(event: ARM_USBD_EVENT_SETUP)
ARM_USBD_ReadSetupPacket 関数

8.3.2 データステージ処理

データステージのデータを転送する場合は、ARM_USBD_EndpointTransfer 関数を使用します。

1. ARM_USBD_EndpointTransfer 関数の第一引数にエンドポイント 0 を指定してください。

2. ARM_USBD_EndpointTransfer 関数の第二引数に転送データを格納する領域の先頭アドレスを指定してください。
3. ARM_USBD_EndpointTransfer 関数の第三引数に転送データサイズを指定してください。
4. ARM_USBD_EndpointTransfer 関数をコールしてください。

[Note]

上記のデータ転送完了時、USB ドライバはコールバック関数(ARM_USBD_SignalEndpointEvent)をコールします。

関連 API:

ARM_USBD_EndpointTransfer 関数
ARM_USBD_SignalEndpointEvent 関数(event: ARM_USBD_EVENT_IN)

8.3.3 ステータスデータ処理

以下の場合、本ドライバは受信した USB リクエストに対するステータスステージの処理を行いません。アプリケーションプログラムから USB リクエストに対するステータスステージの処理を行う必要があります。

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 USB リクエストに ACK 応答する場合

ARM_USBD_EndpointTrasnfer 関数を使用してください。

記述例)

```
ARM_USBD_EndpointTransfer ((USB_EP_IN | USB_EP0), USB_NULL, USB_NULL);
```

(1). 第 1 引数には、Endpoint 番号 0 を指定してください。
(2). 第 2 引数と第 3 引数には、USB_NULL を指定してください。

2. USB リクエストに STALL 応答する場合

ARM_USBD_EndpointStall 関数を使用してください。第一引数には、Endpoint0 を指定し、第二引数には、true を指定してください。

記述例)

```
ARM_USBD_EndpointStall(USB_EP0, true);
```

[Note]

データステージをサポートする USB リクエストの場合、アプリケーションプログラムによってデータステージ処理が行われた後、USB ドライバによってステータスステージの処理が行われます。

関連 API:

ARM_USBD_EndpointStall 関数
ARM_USBD_EndpointTransfer 関数
ARM_USBD_SignalEndpointEvent 関数(event: ARM_USBD_EVENT_IN)

8.4 Enumeration 完了

Enumeration が完了すると USB デバイスステートが Configured 状態に遷移し、Bulk/Interrupt 転送による USB データ通信が行えます。アプリケーションプログラムは、Configured 状態検出後、USB データ通信を行うため各 Endpoint をコンフィグレーションする必要があります。Endpoint のコンフィグレーションには、ARM_USBD_EndpointConfigure 関数を使用します。

[Note]

USB モジュールが SetConfiguration リクエストを受信した時、USB デバイスステートが Configured 状態に遷移します。アプリケーションプログラムでは、受信リクエストをチェックすることにより

Configured 状態へ遷移したかどうかを判断できます。USB リクエストの受信方法については、「8.3.1 **Setupステージ処理 (USBリクエスト受信)**」を参照してください。

関連 API:

ARM_USBD_SignalEndpointEvent 関数(event: ARM_USBD_EVENT_SETUP)
ARM_USBD_ReadSetupPacket 関数
ARM_USBD_EndpointConfigure 関数

8.5 USB データ通信

USB デバイスステートが Configured 状態に遷移し、各 Endpoint のコンフィグレーションが完了すると、Bulk 転送や Interrupt 転送による USB データ転送が行えます。Bulk/Interrupt 転送によるデータ転送は ARM_USBD_EndpointTransfer 関数を使用してください。

[Note]

1. データ転送完了時、USB ドライバはコールバック関数(ARM_USBD_SignalEndpointEvent)をコールします。アプリケーションプログラムは、このコールバック関数によってデータ転送完了を検出することができます。
2. 受信データサイズを取得する場合は、ARM_USBD_EndpointTransferGetResult 関数を使用してください。

関連 API:

ARM_USBD_EndpointTransfer 関数
ARM_USBD_SignalEndpointEvent 関数(event: ARM_USBD_EVENT_OUT)
ARM_USBD_SignalEndpointEvent 関数(event: ARM_USBD_EVENT_IN)
ARM_USBD_EndpointTransferGetResult 関数

8.6 Suspend

USB Host からの Suspend を USB モジュールが検出すると、USB ドライバは、ARM_USBD_SignalDeviceEvent 関数(コールバック関数)をコールします。このコールバック関数の引数には、ARM_USBD_EVENT_SUSPEND がセットされています。アプリケーションプログラムは、このコールバック関数によって Suspend を検出することができます。

関連 API:

ARM_USBD_SignalDeviceEvent 関数(event: ARM_USBD_EVENT_SUSPEND)

8.7 Resume

USB Host からの Resume を USB モジュールが検出すると、USB ドライバは、ARM_USBD_SignalDeviceEvent 関数(コールバック関数)をコールします。このコールバック関数の引数には、ARM_USBD_EVENT_RESUME がセットされています。アプリケーションプログラムは、このコールバック関数によって Resume を検出することができます。

関連 API:

ARM_USBD_SignalDeviceEvent 関数(event: ARM_USBD_EVENT_RESUME)

8.8 RemoteWakeup

USB Host に RemoteWakeup を送信する場合、ARM_USBD_DeviceRemoteWakeup 関数をコールしてください。

関連 API:

ARM_USBD_DeviceRemoteWakeup 関数

8.9 デタッチ処理

USB デバイスを USB Host からデタッチ(切断)すると、USB ドライバは ARM_USBD_SignalDeviceEvent 関数(コールバック関数)をコールします。このコールバック関数の引数には、ARM_USBD_EVENT_VBUS_OFF がセットされています。アプリケーションプログラムは、このコールバック関数によってデタッチを検出することができます。

関連 API:

ARM_USBD_SignalDeviceEvent 関数(event: ARM_USBD_EVENT_VBUS_OFF)
ARM_USBD_EndpointUnconfigure 関数

8.10 API コールシーケンス

API コールシーケンスを以下に示します。

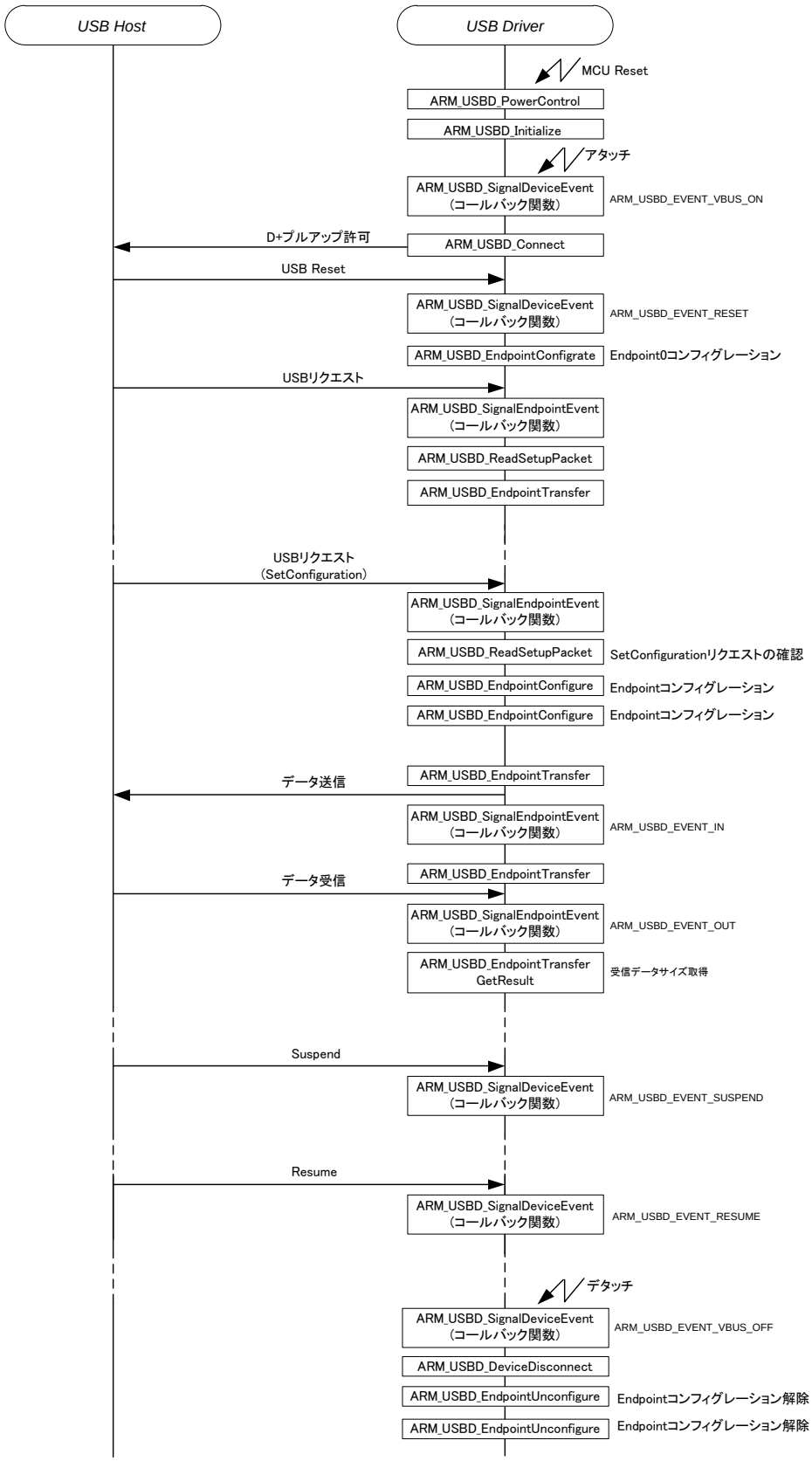

Figure 8-1　API コールフロー

9. クラスリクエスト処理

クラスリクエストの処理手順については、「8.3 Enumeration (リクエスト)処理」を参照してください。

[Note]

サポートしていないクラスリクエストを受信した場合、Stall 応答してください。Stall 応答については、「8.3.3 **ステータスデータ処理**」を参照してください。

10. Zero-Length パケット送信

USB Host に対し Zero-Length パケットを送信する場合、ARM_USBD_EndpointTransfer 関数(API)を使用してください。

記述例) Endpoint1 に Zero-Length パケットを送信する場合

ARM_USBD_EndpointTransfer ((USB_EP_IN | USB_EP1), USB_NULL, USB_NULL);

1. 第 1 引数に、Zero-Length パケットを送信する Endpoint 番号を指定してください。
2. 第 2 引数と第 3 引数に USB_NULL を指定してください。

11. DMA/DTC 転送

11.1 基本仕様

本ドライバが使用する DMA/DTC 転送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）	2 バイト転送：16 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

[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 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、MaxPacketSize×n 倍のサイズを確保してください。

11.2.2 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_DMAC_Open

[Note]

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

USB_CFG_USB0_DMA_TX, USB_CFG_USB0_DMA_RX

例)

```
R_DMACA_Open(USB_CFG_USB0_DMA_TX);
R_DMACA_Open(USB_CFG_USB0_DMA_RX);
```

12. 注意事項

12.1 NVIC への USB 割り込み登録

以下の USB 割り込みは、r_system_cfg.h にて NVIC 登録してください。

1. USBI 割り込み
2. USBR 割り込み
3. D0FIFO 割り込み
4. D1FIFO 割り込み

登録例を以下に示します。

```
#define    SYSTEM_CFG_EVENT_NUMBER_USBFS_D0FIFO    SYSTEM_IRQ_EVENT_NUMBER8
#define    SYSTEM_CFG_EVENT_NUMBER_USBFS_D1FIFO    SYSTEM_IRQ_EVENT_NUMBER9
#define    SYSTEM_CFG_EVENT_NUMBER_USBFS_USBI      SYSTEM_IRQ_EVENT_NUMBER10
#define    SYSTEM_CFG_EVENT_NUMBER_USBFS_USBR      SYSTEM_IRQ_EVENT_NUMBER11
```

12.2 クロック設定について

USB を使用する場合、UCLK に 48MHz を供給する必要があります。UCLK に 48MHz を供給するためには r_core_cfg.h ファイル内の以下の定義に対する設定を行ってください。

1. SYSTEM_CFG_MOSC_ENABLE　必ず"1"を指定してください。
2. SYSTEM_CFG_LOCO_ENABLE　必ず"0"を指定してください。
3. SYSTEM_CFG_PLL_ENABLE　必ず"1"を指定してください。
4. SYSTEM_CFG_PLL_DIV
5. SYSTEM_CFG_PLL_MUL
6. SYSTEM_CFG_CLOCK_SOURCE　必ず"5"を指定してください。
7. SYSTEM_CFG_PCKB_DIV
8. SYSTEM_CFG_POWER_CONTROL_MODE　必ず"0"を指定してください。

12.3 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。

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、「7. **コンフィグレーション (r_usb_basic_mini_cfg.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. Driver_USBD.h
2. r_usb_basic_if.h (必ずインクルードしてください。)
3. 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。)
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 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). USB 端子設定用の関数 R_USB_Pinset() / R_USB_Pinclr()が用意されています。

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_DMAC_Open

[Note]

DMA/DTC 転送を行う場合は、r_usb_basic_mini_cfg.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_mini_cfg.h)」を参照してください。

初期化例)

(1). DTC

```
e_dma_err_t     ret;

ret = Driver_DTC.Open();
if (DMA_OK != ret)
{
      /* do something */
}
```

(2). DMA

```
ret = Driver_DMAC0.Open();
if (DMA_OK != ret)
{
      /* do something */
}

ret = Driver_DMAC1.Open();
if (DMA_OK != ret)
{
      /* do something */
}
```

3. USB 関連初期化

ARM_USBD_Initialize 関数をコールし、使用する USB モジュール(HW)、USB ドライバ(SW)の初期化およびコールバック関数(ARM_USBD_SignalDeviceEvent 関数, ARM_USBD_ SignalEndpointEvent 関数)の登録を行ってください。

13.3.3 Descriptor の作成

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「**2.4 Descriptor**」を参照してください。

13.3.4 メインルーチンの作成

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_ControDriver 関数をコールしてください。USB 関連完了イベントは、ARM_USBD_SignalDeviceEvent 関数または ARM_USBD_SignalEndpointEvent 関数の引数に設定されます。アプリケーションプログラムでは、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]

ARM_USBD_EndpointTransfer 関数による USB データ通信は、Configured 状態の検出を確認した後で行ってください。

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/inquiry

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Sep 30, 2019	—	初版発行

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。