RE01 1500KBグループ

R01AN4945EJ0100
Rev.1.00
Sep 30, 2019

USB Peripheral Mass Storage Class Driver (PMSC)

要旨

本書では、RE01 1500KB グループ向け CMSIS software package の USB Peripheral Mass Storage Class Driver (以下、PMSC ドライバ)の仕様を説明します。

対象デバイス

RE01 1500KB グループ

目次

1. 概要

本ドライバは、USB Basic Peripheral ドライバと組み合わせることで、Peripheral Mass Storage Class Driver(PMSC)として動作します。PMSC は、USB マスストレージクラスの Bulk-Only Transport(BOT)プロトコルで構築されています。USB ペリフェラルコントロールドライバ、メディアドライバと組み合わせることで、BOT 対応のストレージ機器として USB ホストと通信を行うことができます。

以下に、本ドライバがサポートしている機能を示します。

- BOTプロトコルによるストレージコマンド制御
- USBホストからのマスストレージデバイスクラスリクエストに対する応答

1.1 必ずお読みください

このドライバを使ってアプリケーションプログラムを作成する場合は、USB Basic Peripheral Driver 仕様書(ドキュメント No.R01AN4944)を参照いただきますようお願いします。

1.2 制限事項

1. USB Host から送信されるマスストレージクラスコマンド(GetMaxLun)に対し、本ドライバは値 0 を返します。
2. 本ドライバがサポートするセクタサイズは 512 のみです。

1.3 注意事項

1. 本ドライバは、USB 通信動作を保証するものではありません。
2. ストレージ領域として使用するメディアを制御するメディアドライバ関数はお客様において実装いただく必要があります。

1.4 用語一覧

APL	:	Application program
BOT	:	Bulk Only Transport
Non-OS	:	USB Driver for OS-less
DDI	:	Device Driver Interface
PCD	:	Peripheral Control Driver of USB-BASIC-FW
PCI	:	PCD Interface
PMSCD	:	Peripheral Mass Storage USB Class Driver (PMSCF + PCI + DDI)
PMSCF	:	Peripheral Mass Storage Class Function
PMSDD	:	Peripheral Mass Storage Device Driver (ATAPI driver)

2. ソフトウェア構成

本ドライバは、PMSCD と PMSDD で構成されています。
PMSCD は、BOT プロトコル制御及びデータ送受信を行う PMSCF、PMSDD に対するインタフェース関数群（DDI）、PCD に対するインタフェース関数群（PCI）で構成されています。
PMSDD は、PCD を介してホストとの BOT プロトコル通信を行います。PMSDD では、PMSCD から受け取ったストレージコマンドを解析し実行します。
Figure 2-1にモジュール構成、Table 2-1に機能概要を示します。

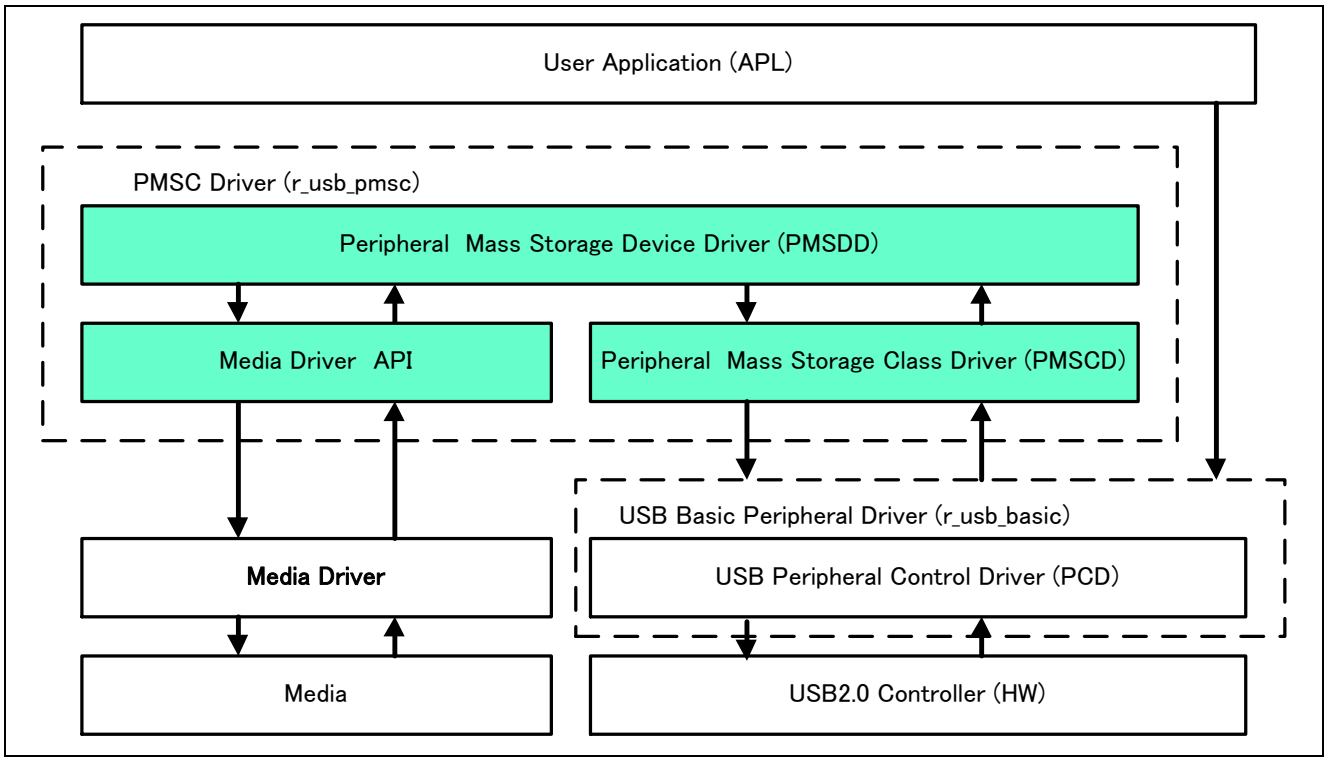

Figure 2-1 ソフトウェア構成図

Table 2-1　各モジュール機能概要

モジュール名	機能概要
PMSDD	マスストレージデバイスドライバ ・ PMSCD からのストレージコマンドの処理を行う ・ Media driver を介して Media へのアクセスを行う
DDI	PMSDD-PMSCD 間のインタフェース関数
PMSCF	マスストレージクラスドライバ ・ BOT プロトコルデータの制御、クラスリクエストの対応を行う ・ CBW の解析、データ送受信を行う ・ PMSDD/PCD との連携し CSW を生成する
PCI	PMSCD - PCD 間のインタフェース関数
PCD	USB Peripheral H/W 制御ドライバ

3. クラスドライバ概要

3.1 クラスリクエスト

本ドライバがサポートするクラスリクエストをTable 3-1に示します。

Table 3-1 クラスリクエスト

リクエスト	コード	説明
Bulk-Only Mass Storage Reset	0xFF	マスストレージデバイスと接続インタフェースのリセット
Get Max Lun	0xFE	デバイスがサポートする論理番号を通知

3.2 ストレージコマンド

本ドライバがサポートするストレージコマンドをTable 3-2に示します。下記以外のコマンドに対してはSTALL 応答あるいは CSW による FAIL エラーを返します。

Table 3-2 ストレージコマンド

コマンド名	コード	説明
TEST_UNIT_READY	0x00	ペリフェラル機器の状態確認
REQUEST_SENSE	0x03	直前のストレージコマンド実行結果のエラー情報取得
INQUIRY	0x12	論理ユニットのパラメータ情報取得
READ_FORMAT_CAPACITY	0x23	フォーマット可能な容量取得
READ_CAPACITY	0x25	論理ユニットの容量情報取得
READ10	0x28	データ読み出し
WRITE10	0x2A	データ書き込み
MODE_SENSE10	0x5A	論理ユニットのパラメータ取得

4. デバイスクラスドライバ（PDCD）

4.1 基本機能

PDCD の機能を以下に示します。

1. SFF-8070i（ATAPI）に対応
2. USB ホストからのマスストレージデバイスクラスリクエストに対する応答

4.2 BOT プロトコル概要

BOT（Bulk-Only Transport）とは、バルクイン/バルクアウトの 2 つの Endpoint のみを使用し、コマンド、データ、ステータス（コマンド処理の結果）を管理する転送プロトコルです。

USB 上で転送されるデータのうち、コマンドとステータスについては Command Block Wrapper（CBW）、Command Status Wrapper（CSW）の形式で転送を行います。BOT プロトコル概要をFigure 4-1に示します。

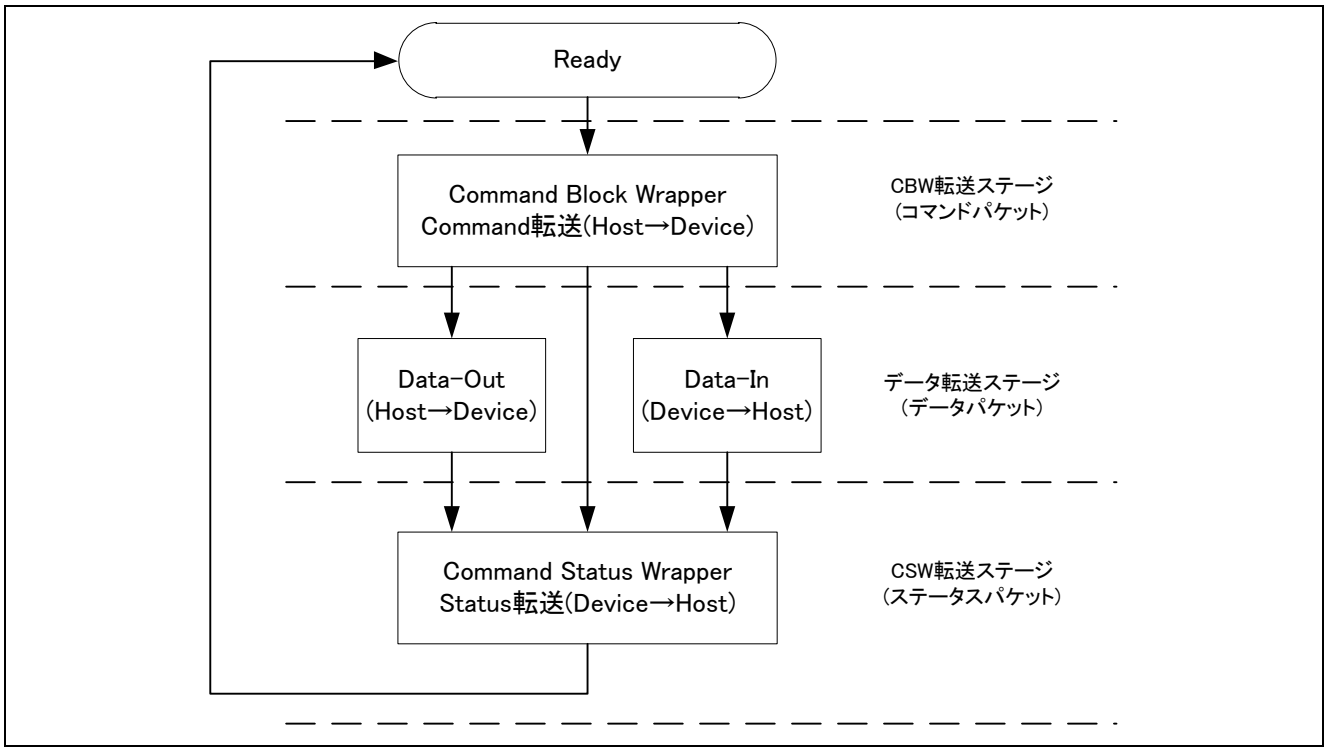

Figure 4-1 BOT プロトコル概要

5. API

アプリケーションプログラム内で使用する API については、USB Basic Peripheral Driver Specification (ドキュメント No.R01AN4944)内の「API」の章を参照してください。

6. コンフィグレーション (r_usb_pmsc_mini_cfg.h)

お客様のシステムにあわせて以下の設定をお願いします。

[Note]

必ず r_usb_basic_mini_cfg.h ファイルに対する設定もお願いします。r_usb_basic_mini_cfg.h については、USB Basic Peripheral Driver Specification (ドキュメント No.R01AN4944)内の「コンフィグレーション」の章を参照してください。

1. Inquiry コマンド応答データ設定

本ドライバは以下の各定義に指定したデータを Inquiry コマンドの応答データとして USB ホストに送信します。

(1). Vendor Information 設定

Inquiry コマンドの応答データである Vendor Information を指定してください。必ず 8 バイトのデータをダブルクォーテーションで括って指定してください。

```
#define    USB_CFG_PMSC_VENDOR    Vendor Information
```

例)

```
#define    USB_CFG_PMSC_VENDOR    "Renesas "
```

(2). Product Information 設定

Inquiry コマンドの応答データである Product Information を指定してください。必ず 16 バイトのデータをダブルクォーテーションで括って指定してください。

```
#define    USB_CFG_PMSC_PRODUCT    Product Information
```

例)

```
#define    USB_CFG_PMSC_PRODUCT    "Mass Storage    "
```

(3). Product Revision Level 設定

Inquiry コマンドの応答データである Product Revision Level を指定してください。必ず 4 バイトのデータをダブルクォーテーションで括って指定してください。

```
#define    USB_CFG_PMSC_REVISION    Product Revision Level
```

例)

```
#define    USB_CFG_PMSC_REVISION    "1.00"
```

2. 転送セクタ数設定

PCD(USB Peripheral Control Driver)に要求する 1 回のデータ転送の最大セクタサイズを指定してください。本ドライバは、PCD に対し"1 セクタ(512)×USB_CFG_PMSC_TRANS_COUNT "バイトの値を転送サイズとして指定します。

この値を大きくすることにより PCD に対するデータ転送要求回数が減るため転送速度性能が向上する可能性がありますが、"1 セクタ(512)×USB_CFG_PMSC_TRANS_COUNT "バイト分の RAM が消費されますのでご注意ください。

```
#define    USB_CFG_PMSC_TRANS_COUNT    転送セクタ数 (1 から 255)
```

例)

```
#define    USB_CFG_PMSC_TRANS_COUNT    8
```

7. メディアドライバインタフェース

PMSC では、仕様の異なるメディアドライバへのアクセスを容易にするために共通のメディアドライバ API 関数を使用しています。

7.1 メディアドライバ API 関数

メディアドライバ API は、PMSC から呼び出され、ユーザによって実装されたメディアドライバ関数をコールします。本章では、メディアドライバ API 関数のプロトタイプと各関数の実装に必要な処理について説明します。

メディアドライバ API 関数一覧を示します。

Table 7-1 メディアドライバ API

メディアドライバ API	処理概要
R_USB_media_initialize	メディアドライバの初期化
R_USB_media_open	メディアドライバオープン
R_USB_media_close	メディアドライバクローズ
R_USB_media_read	メディアリード
R_USB_media_write	メディアライト
R_USB_media_ioctl	メディアデバイス特有のコントロール命令を処理

7.1.1 R_USB_media_initialize

メディアドライバ関数をメディアドライバに登録します。

形式

bool R_USB_media_initialize(media_driver_t * p_media_driver);

引数

p_meida_driver	メディアドライバ構造体領域へのポインタ

戻り値

TRUE	正常終了
FALSE	エラー発生

解説

ユーザによって実装されたメディアドライバ関数をメディアドライバに登録します。なお、ユーザアプリケーションプログラム内の初期化処理等において必ず本APIをコールしてください。

補足

1. 上記「引数」、「戻り値」および「解説」等に記載した内容をサポートするメディアドライバ関数をユーザにおいて実装する必要があります。
2. ユーザによって実装されたメディアドライバ関数の登録方法については、「7.3 ストレージメディアドライバの登録」を参照ください。
3. 本 API はデバイスのレジスタ初期化処理やデバイス上の動作を開始するものではありません。それらの処理は R_USB_media_open()関数で行います。
4. この PMSC は複数種類のメディアドライバ関数を登録するための機能をサポートしていません。

使用例

```
if (!R_USB_media_initialize(&g_ram_mediadriver))
{
        /* Handle the error */
}
result = R_USB_media_open();
if (USB_MEDIA_RET_OK != result)
{
        /* Process the error */
}
```

7.1.2 R_USB_media_open

メディアデバイスおよびメディアドライバの初期化処理を行います。

形式

usb_media_ret_t R_USB_media_open(void);

引数

--

戻り値

USB_MEDIA_RET_OK	正常終了
USB_MEDIA_RET_PARAERR	パラメータエラー
USB_MEDIA_RET_DEV_OPEN	デバイスがすでにオープン済
USB_MEDIA_RET_NOTRDY	デバイスが応答しないまたは存在しない
USB_MEDIA_RET_OP_FAIL	その他のエラー

解説

メディアデバイスおよびメディアドライバおよびの初期化を行い、メディアデバイスおよびメディアドライバをレディ状態にします。なお、ユーザアプリケーションプログラム内の初期化処理等において必ず本APIをコールしてください。

補足

1. 本関数を呼び出す前に R_USB_media_initialize()関数を呼び出す必要があります。
2. R_USB_media_close()関数をコールしない限り、R_USB_media_open()の呼び出し回数は 1 回のみです。なお、R_USB_media_close()関数を呼び出した後であれば、デバイス設定を初期状態に戻すために本関数を再び呼び出すことができます。
3. 上記「引数」、「戻り値」および「解説」等に記載した内容をサポートするメディアドライバ関数をユーザにおいて実装する必要があります。

使用例

```
if (!R_USB_media_initialize(&g_ram_mediadriver))
{
    /* Handle the error */
}

result = R_USB_media_open();
if (USB_MEDIA_RET_OK != result)
{
    /* Process the error */
}
```

7.1.3 R_USB_media_close

メディアドライバ用のリソースを解放し、ハードウェアを非アクティブな状態に戻します。

形式

usb_media_ret_t R_USB_media_close(void);

引数

--

戻り値

USB_MEDIA_RET_OK	正常終了
USB_MEDIA_RET_PARAERR	パラメータエラー
USB_MEDIA_RET_OP_FAIL	その他のエラー

解説

メディアドライバ用のリソースを解放し、ハードウェアを非アクティブな状態に戻します。

補足

1. 本関数を呼び出す前に R_USB_media_initialize()関数を呼び出す必要があります。
2. 上記「引数」、「戻り値」および「解説」等に記載した内容をサポートするメディアドライバ関数をユーザにおいて実装する必要があります。

使用例

```
result = R_USB_media_close();
if (USB_MEDIA_RET_OK != result)
{
    /* Process the error */
}
```

7.1.4 R_USB_media_read

メディアデバイスからデータブロックを読み出します。

形式

usb_media_ret_t R_USB_media_read(uint8_t *p_buf, uint32_t lba, uint8_t count);

引数

p_buf	メディアデバイスから読みだされたデータを格納する領域へのポインタ
lba	読み出し開始論理ブロックアドレス
count	読み出しブロック数 (セクタ数)

戻り値

USB_MEDIA_RET_OK	正常終了
USB_MEDIA_RET_PARAERR	パラメータエラー
USB_MEDIA_RET_NOTRDY	デバイスがレディ状態ではありません
USB_MEDIA_RET_OP_FAIL	その他のエラー

解説

メディアデバイスからデータブロックのリード処理を行います。(第 2 引数で指定された LBA(Logical BLock Address)から第 3 引数(count)に指定されたブロック数分のデータブロックをリードします。)

リードデータは第 1 引数(p_buf)で指定した領域に格納されます。

補足

1. 本関数を呼び出す前に R_USB_media_initialize()関数を呼び出す必要があります。
2. 上記「引数」、「戻り値」および「解説」等に記載した内容をサポートするメディアドライバ関数をユーザにおいて実装する必要があります。

使用例

```
result = R_USB_media_read(&buffer, lba, 1);
if (USB_MEDIA_RET_OK != result)
{
/* Process the error */
}
```

7.1.5 R_USB_media_write

メディアデバイスにデータブロックを書き込みます。

形式

usb_media_ret_t R_USB_media_write(uint8_t *p_buf, uint32_t lba, uint8_t count);

引数

p_buf	メディアデバイスに書き込むデータを格納する領域へのポインタ
lba	書き込み開始論理ブロックアドレス
count	書き込みブロック数 (セクタ数)

戻り値

USB_MEDIA_RET_OK	正常終了
USB_MEDIA_RET_PARAERR	パラメータエラー
USB_MEDIA_RET_NOTRDY	デバイスがレディ状態ではありません
USB_MEDIA_RET_OP_FAIL	その他のエラー

解説

メディアデバイスへデータブロックのライト処理を行います。(第 2 引数で指定された LBA(Logical BLock Address)へ第 3 引数(count)に指定されたブロック数分のデータブロックをライトします。)

ライトデータは第 1 引数(p_buf)が示す領域に格納してください。

補足

1. 本関数を呼び出す前に R_USB_media_initialize()関数を呼び出す必要があります。
2. 上記「引数」、「戻り値」および「解説」等に記載した内容をサポートするメディアドライバ関数をユーザにおいて実装する必要があります。

使用例

```
result = R_USB_media_write(&buffer, lba, 1);
if (MEDIA_RET_OK != result)
{
/* Process the error */
}
```

7.1.6 R_USB_media_ioctl

メディアドライバ等の情報を取得します。

形式

usb_media_ret_t R_USB_media_ioctl(ioctl_cmd_t command, void *p_data);

引数

command	コマンドコード
p_data	メディア情報を格納する領域へのポインタ

戻り値

USB_MEDIA_RET_OK	正常終了
USB_MEDIA_RET_PARAERR	パラメータエラー
USB_MEDIA_RET_NOTRDY	デバイスがレディ状態ではありません
USB_MEDIA_RET_OP_FAIL	その他のエラー

解説

本関数はメディアドライバ固有のコマンドを引数(command)に指定し、メディアドライバからの戻り情報を取得する処理を行います。

PMSCではメディアドライバに対するコマンドコードとして、以下のコマンドを使用しています。

MEDIA_IOCTL_GET_NUM_BLOCKS	メディア領域のブロック数
MEDIA_IOCTL_GET_BLOCK_SIZE	1 ブロックサイズ

補足

1. 本関数を呼び出す前に R_USB_media_initialize()関数を呼び出す必要があります。
2. ユーザは引数(command)に指定するコマンドコードを新たに定義することができます。
3. 上記「引数」、「戻り値」および「解説」等に記載した内容をサポートするメディアドライバ関数をユーザにおいて実装する必要があります。

使用例

```
uint32_t num_blocks;
uint32_t block_size;
uint64_t capacity;.

result = R_USB_media_ioctl(MEDIA_IOCTL_GET_NUM_BLOCKS, (void *)&num_blocks);
result = R_USB_media_ioctl(MEDIA_IOCTL_GET_BLOCK_SIZE, (void *)&block_size);

capacity = (uin64_t)block_size * (uint64_t)num_blocks;
```

7.2 構造体/列挙型定義

メディアドライバ API 関数で使用される構造体/列挙型を以下に示します。
これらは r_usb_media_driver_if.h ファイルで定義されています。

7.2.1 usb_media_driver_t (構造体)

usb_media_driver_t は、ユーザによって実装されたメディアドライバ関数へのポインタを保持する構造体です。
以下に、usb_media_driver_t 構造体を示します。

```
typedef struct media_driver_s
{
        usb_media_open_t    pf_media_open;    /* オープン関数のポインタ */
        usb_media_close_t   pf_media_close;   /* クローズ関数のポインタ */
        usb_media_read_t    pf_media_read;    /* リード関数のポインタ */
        usb_media_write_t   pf_media_write;   /* ライト関数のポインタ */
        usb_media_ioctl_t   pf_media_ctrl;    /* コントロール関数のポインタ */
} usb_media_driver_t
```

7.2.2 usb_media_ret_t (列挙型)

usb_media_ret_t には、メディアドライバ API が返す戻り値が定義されています。

```
typedef enum
{
        USB_MEDIA_RET_OK = 0,          /* 正常終了 */
        USB_MEDIA_RET_NOTRDY,          /* デバイスがレディ状態でない */
        USB_MEDIA_RET_PARERR,          /* パラメータエラー */
        USB_MEDIA_RET_OP_FAIL,         /* その他のエラー */
        USB_MEDIA_RET_DEV_OPEN,        /* デバイスは既にオープンしている */
} usb_media_ret_t
```

7.2.3 ioctl_cmd_t (列挙型)

ioctl_cmd_t には、R_USB_media_ioctl 関数に指定するコマンドコードが定義されています。

```
typedef enum
{
    USB_MEDIA_IOCTL_GET_NUM_BLOCKS,     /* 論理ブロック数取得 */
    USB_MEDIA_IOCTL_GET_BLOCK_SIZE,     /* 論理ブロックサイズ取得 */
} ioctl_cmd_t
```

[Note]

メディアドライバ実装にあたりユーザ独自のコマンドコードを追加する場合、上記 ioctl_cmd_t に追加してください。

7.3 ストレージメディアドライバの登録

PMSC のストレージメディアをフラッシュメモリなどの他のストレージメディアへ変更する場合、ユーザは、そのストレージメディアに対する読み出しまたは書き込みを行うためのメディアドライバ関数を実装し、メディアドライバ API に登録する必要があります。

以下に、シリアル SPI フラッシュのメディアドライバ関数登録手順を示します。

1. 登録するメディアドライバ関数の作成

ユーザがシリアル SPI フラッシュ用メディアドライバ関数として以下の関数を実装したものとします。

1. usb_media_ret_t spi_flash_open (void)
2. usb_media_ret_t spi_flash_close (void)
3. usb_media_ret_t spi_flash_read(uint8_t *p_buf,uint32_t lba, uint8_t count)
4. usb_media_ret_t spi_flash_write(uint8_t *p_buf,uint32_t lba, uint8_t count)
5. usb_media_ret_t spi_flash_ioctl(ioctl_cmd_t ioctl_cmd,void * ioctl_data)

2. メディアドライバ関数のメディア API への登録

(1). シリアル SPI フラッシュ用の usb_media_driver_t 構造体を定義してください。

この構造体の各メンバには、該当するメディアドライバ関数へのポインタを設定してください。

```
struct media_driver_t    g_spi_flash_mediadriver =
{
      &spi_flash_open,
      &spi_flash_close,
      &spi_flash_read,
      &spi_flash_write,
      &spi_flash_ioctl
};
```

(2). アプリケーションプログラムで、上記の usb_media_driver_t 構造体へのポインタを R_USB_media_initialize 関数(API)の引数に指定し、初期化処理を行ってください。

```
== アプリケーションプログラム ==
R_USB_media_initialize(& g_spi_flash_mediadriver );
```

上記手順をおこなうことにより、メディアドライバが呼び出すメディアドライバ関数としてシリアル SPI フラッシュ関数が登録されます。

7.4 ストレージメディアドライバの実装

ご使用になるストレージメディアに対応するメディアドライバ関数をお客様において実装いただく必要があります。実装したメディアドライバ関数は、PMSC から「7.1 **メディアドライバAPI関数**」に記載された API を経由してコールされます。

[Note]

メディアドライバ関数の実装に必要な処理については、「7.1 **メディアドライバAPI関数**」に記載された各 API 仕様を参考してください。

7.5 メディアドライバ関数のプロトタイプ宣言

メディアドライバ関数のプロトタイプ宣言を以下に示します。

```
1.  usb_media_ret_t  (*media_open_t) (uint8_t);                                  /* Open 関数型  */
2.  usb_media_ret_t  (*media_close_t)(uint8_t);                                  /* Close 関数型  */
3.  usb_media_ret_t (*media_read_t)(uint8_t, uint8_t*, uint32_t, uint8_t);       /* Read 関数型  */
4.  usb_media_ret_t (*media_write_t)(uint8_t, uint8_t*, uint32_t, uint8_t);      /* Write 関数型  */
5.  usb_media_ret_t  (*media_ioctl_t)(uint8_t, ioctl_cmd_t, void *);            /* Control 型関数  */
```

8. アプリケーションの作成方法

USB Basic Peripheral Driver Specification (ドキュメント No.R01AN4944)内の「**アプリケーションプログラムの作成方法**」の章を参照してください。

[Note]

ユーザアプリケーションプログラム内の初期化処理等において必ず R_USB_media_initialze 関数(API)および R_USB_media_open 関数(API)をコールしてください。

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/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。