RZ/A1H グループ

R01AN3292JJ0110
Rev.1.10
Sep 30, 2016

USB Host Mass Storage Class Driver (HMSC)

要旨

本アプリケーションノートでは、USB Host マスストレージクラスドライバ(HMSC)について説明します。本ドライバは USB Basic Host Driver(USB-BASIC-F/W)と組み合わせることで動作します。以降、本ドライバを HMSC と称します。

対象デバイス

RZ/A1H グループ

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

関連ドキュメント

1. Universal Serial Bus Revision 2.0 specification
2. USB Mass Storage Class Specification Overview Revision 1.1
3. USB Mass Storage Class Bulk-Only Transport Revision 1.0
   【http://www.usb.org/developers/docs/】
4. RZ/A1H グループ、RZ/A1M グループ　ユーザーズマニュアル ハードウェア編 (ドキュメント No.R01UH0403JJ)
5. RZ/A1H グループ USB Host and Peripheral Interface Driver（ドキュメント No.R01AN329JJ）
6. RZ/A1Hグループ ARM® Development Studio 5（DS-5™）のセミホスティング機能を使用したNOR型フラッシュメモリへのダウンロード例（ドキュメントNo.R01AN1957JJ）
7. RZ/A1H グループレジスタ定義ヘッダ・ファイル iodefine.h（R01AN1860JJ）
8. RZ/A1H グループ初期設定例（R01AN1864JJ）

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

目次

1. 概要

HMSC は、USB-BASIC-F/W と組み合わせることで、USB Host マスストレージクラスドライバ（以降 HMSC と記述）として動作します。

HMSC 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。ファイルシステム、ストレージデバイスドライバと組み合わせることで BOT 対応の USB ストレージ機器と通信を行うことが可能です。ファイルシステムは FatFs を使用することを前提に作成しています。

以下に、本モジュールがサポートしている機能を示します。

- 接続された USB ストレージ機器のデバイス照合（動作可否判定）を行う。
- BOT プロトコルによるストレージコマンド通信を行う。
- USB マスストレージサブクラスの SFF-8070i(ATAPI)に対応。
- 最大 4 つのストレージ機器を接続することができます。

1.1 必ずお読みください

お客様が、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、ドキュメントドキュメント(Document No:R11AN3291JJ)に記載された API の使用をお勧めします。当該ドキュメントは、パッケージ内の"reference_documents"フォルダにあります。

[Note]

1. ドキュメント(Document No:R11AN3291JJ)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は当該ドキュメントに記載されています。
2. ドキュメント(Document No:R11AN0022JJ)に記載された API を使用した場合、本書の「6.2 FSI API 関数」、「7.4 HMSDD API 関数」、「8.6 HMSC API 関数」に記載された API を使用する必要はありません。

1.2 動作確認済環境

HMSC の動作確認済環境をTable 1.1に示します。

Table 1.1 動作確認条件

項目	内容
使用マイコン	RZ/A1H
動作周波数（注）	CPU クロック（Iφ）: 400MHz
	画像処理クロック（Gφ）: 266.37MHz
	内部バスクロック（Bφ）: 133.33MHz
	周辺クロック 1（P1φ）: 66.67MHz
	周辺クロック 0（P0φ）: 33.33MHz
動作電圧	電源電圧（I/O）: 3.3V
	電源電圧（内部）: 1.8V
統合開発環境	ARM®t統合開発環境
	・ARM Development Studio（DS-5™） Version 5.16
	IARt 統合開発環境
	・IAR Embedded Workbench for ARM Version 7.40
コンパイラ	ARM C/C++ Compiler/Linker/Assembler Ver.5.03 [Build 102]
	KPIT GNUARM-RZ v14.01
	IAR C/C++ Compiler for ARM 7.40
動作モード	ブートモード 0
	（CS0 空間 16 ビットブート）
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	GENMAI ボード
	・R7S72100 CPU ボード RTK772100BC00000BR
使用デバイス （ボード上で使用する機能）	・シリアルインターフェース（Dsub-9 コネクタ）
	・USB1 コネクタ、USB2 コネクタ

1.3 制限事項

本モジュールには以下の制限事項があります。

- 型の異なるメンバで構造体を構成しています。
  （コンパイラによっては構造体のメンバにアドレスアライメントずれを発生することがあります。）
- HMSDD にも制限事項があります。詳細は7章を参照ください。

用語一覧

APL	:	Application program
BOT	:	Mass storage class Bulk Only Transport
cstd	:	Prefix of function and file for Peripheral & Host Common Basic (USB low level) F/W
FSL	:	FAT File System Library
HCD	:	Host Control Driver of USB-BASIC-F/W
HDCD	:	Host Device Class Driver (device driver and USB class driver)
HSTD	:	Function & File for Host Basic (USB low level) F/W
HUBCD	:	Hub Class Smple Driver
MGR	:	Peripheral device state manager of HCD
non-OS	:	USB basic firmware for non-OS based system
PP	:	プリプロセス定義
Scheduler	:	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
Scheduler Macro	:	non-OSでスケジューラ機能を呼び出すために使用されるもの
Task	:	処理の単位
USB-BASIC-F/W	:	USB Basic Host Driver for RZ/A1H グループ (non-OS)
USB	:	Universal Serial Bus

2. ソフトウェア構成

HDCD(ホストデバイスクラスドライバ)は HMSDD（ホストマスストレージデバイスドライバ）と HMSCD（USB ホストマスストレージクラスドライバ）の総称です。

Figure 2-1に HMSC のモジュール構成、Table 2.1にモジュール機能概要を示します。

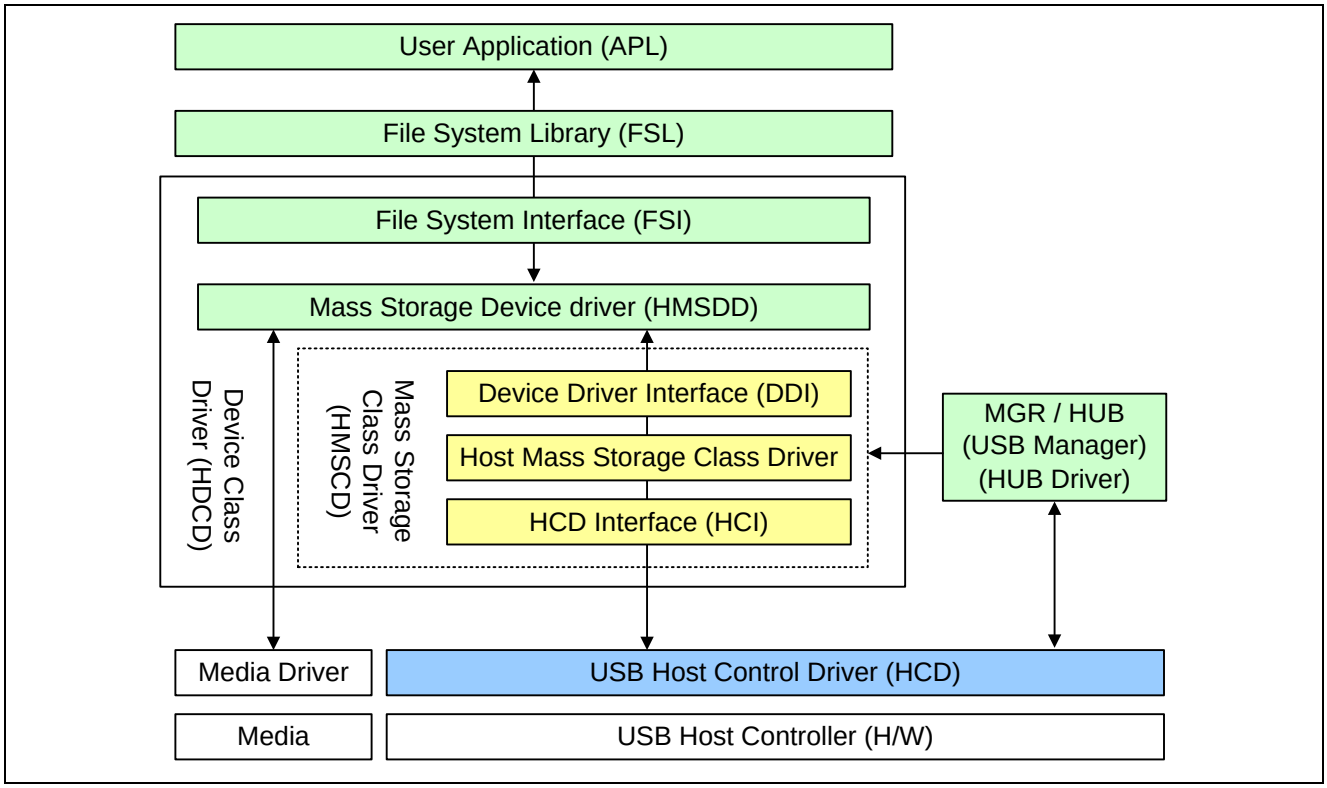

Figure 2-1 ソフトウェア構成図

Table 2.1 モジュール機能概要

モジュール名	機能概要
FSI	FSL－HMSDD 間のインタフェース関数 FSL にあわせて改変してください。
HMSDD	マスストレージデバイスドライバ
DDI	HMSDD－HMSCD 間のインタフェース関数 HMSDD に合わせて改変してください。
HMSCD	マスストレージクラスドライバ ストレージコマンドに BOT プロトコルを付加して HCD へ要求します。また、BOT のシーケンスを管理します。 システム仕様にあわせてストレージコマンドを追加（改変）してください。
HCI	HMSCD－HCD 間のインタフェース関数です。
MGR / HUB	接続されたデバイスとエニュメレーションをして HMSCD を起動します。またデバイスの状態管理も行います。
HCD	USB Host H/W 制御ドライバ

3. システム資源

Table 3.1～Table 3.3に、HMSC が使用しているシステム資源を示します。

Table 3.1 タスク情報

関数名	タスク ID	優先度	概要
usb_hmsc_Task	USB_HMSC_TSK	USB_PRI_3	マスストレージクラスタスク
usb_hmsc_StrgDriveTask	USB_HSTRG_TSK	USB_PRI_3	HMSDD タスク
usb_hmsc_SampleAplTask	USB_HMSCSMP_TSK	USB_PRI_4	APL タスク

Table 3.2 メールボックス情報

メールボックス名	使用タスク ID	待ちタスクキュー	概要
USB_HMSC_MBX	USB_HMSC_TSK	FIFO 順	HMSCD 用メールボックス
USB_HSTRG_MBX	USB_HSTRG_TSK	FIFO 順	HMSDD 用メールボックス
USB_HMSCSMP_MBX	USB_HMSCSMP_TSK	FIFO 順	APL 用メールボックス

Table 3.3 メモリプール情報

メモリプール名	待ちタスクキュー	メモリブロック（注）	概要
USB_HMSC_MPL	FIFO 順	40byte	HMSC 用固定長メモリプール
USB_HSTRG_MBX	FIFO 順	40byte	HDCD 用固定長メモリプール
USB_HMSCSMP_MPL	FIFO 順	40byte	APL 用固定長メモリプール

（注）全システムのメモリブロックの最大数は、USB_BLKMAX で定義されます。初期値は 20 です。

4. ターゲットペリフェラルリスト（TPL）

USB ホストドライバ（USB-BASIC-F/W）とデバイスクラスドライバを組み合わせて使用する際、デバイスドライバごとにターゲットペリフェラルリスト（TPL）を作成する必要があります。

TPL の詳細は USB Host and Peripheral Interfece Driver アプリケーションノート(Document No.R01AN3291JJ)の「ターゲットペリフェラルリストの設定方法」を参照してください。

5. USB ストレージ機器へのアクセス

FSI はセクタ番号を論理ブロックアドレスにセクタ数を転送データサイズに変換します。HMSDD はドライブ番号から USB ストレージ機器へのアクセスであるかを判断します。HMSCD はドライブ番号をユニット番号に変換し、BOT プロトコルに従い、HCD を経由して USB ストレージ機器にアクセスします。

Figure5-1に UBS ストレージ機器へのアクセスシーケンスを示します。

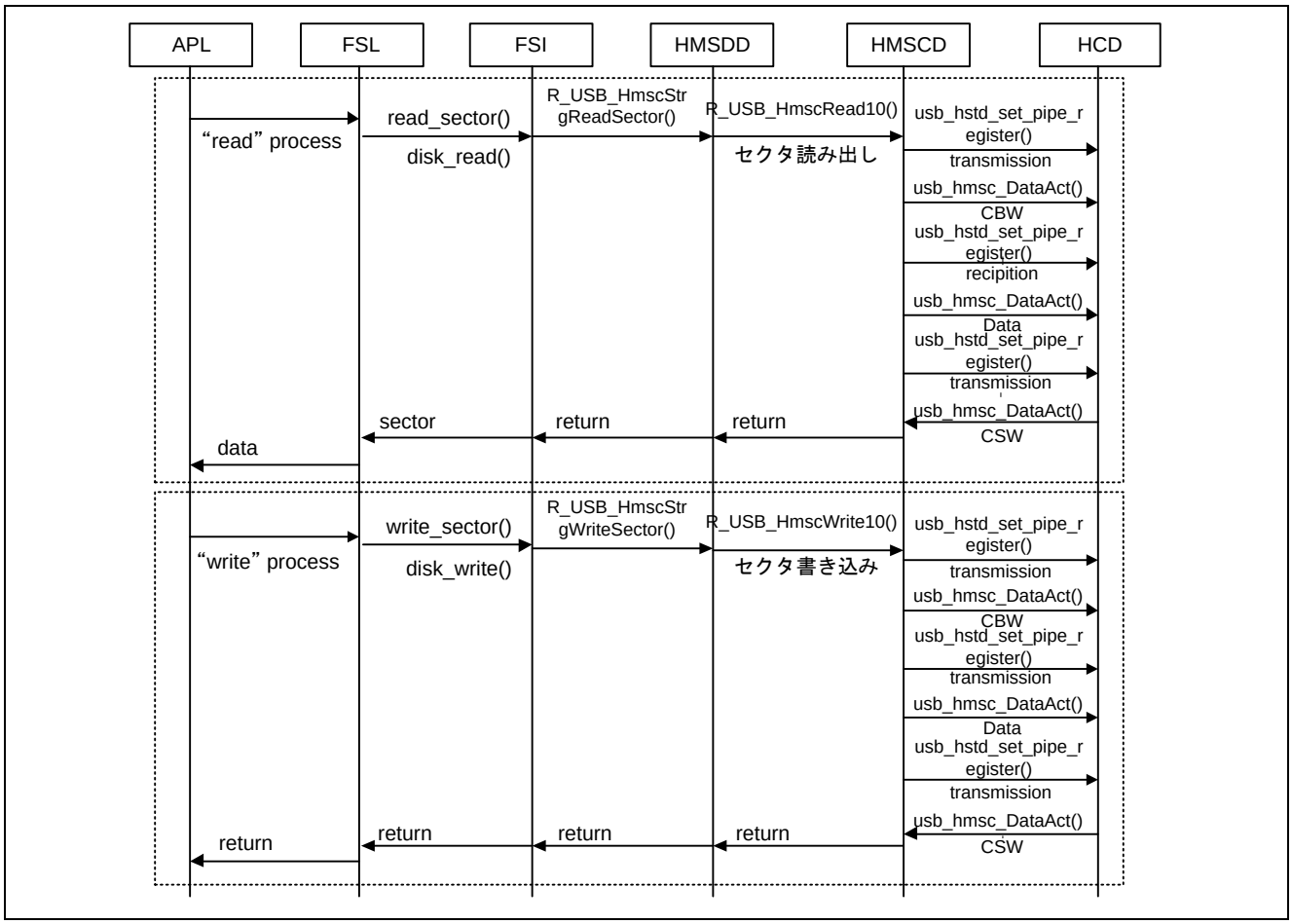

Figure5-1 USB ストレージ機器へのアクセスシーケンス

6. ファイルシステムインタフェース（FSI）

6.1 機能と特長

FSL のインタフェース仕様にあわせて作成してください。

6.2 FSI API 関数

Table 6.1にサンプルの FSI API 関数を示します。

[Note]

ドキュメント(Document No: R01AN329JJ)に記載された API を使用する場合、アプリケーションプログラムでは、以下の API を使用する必要はありません。

Table 6.1 FSI 関数

関数名	説明
disk_read	データを読み出す
disk_write	データを書き込む
R_usb_hmsc_WaitLoop	データのリード/ライド完了待ちをする (For non-OS)
dev_open	ドライブをマウントする(TFAT では未使用)
dev_close	ドライブを開放する (TFAT では未使用)

7. ホストマスストレージデバイスドライバ（HMSDD）

HMSDD は HMSCD を使用する場合に FAT によって起動されます。HMSDD はドライブ番号からストレージ機器を選択します。

7.1 制限事項

HMSDD には以下の制限があります。

- 最大 4 つの USB ストレージ機器を接続することができます。
- セクタサイズが 512 バイトの USB ストレージ機器と接続可能です。
- READ_CAPACITY コマンドに応答しないデバイスはセクタサイズを 512 バイトとして動作します。
- ストレージ機器として認識できないデバイスがあります。

7.2 論理ユニット番号 (LUN)

デバイスが GetMaxUnit リクエストに応答せず、ユニット数が未定の場合はユニット数=0 として動作します。HMSCD は、BOT 仕様で USB パケットを作成します。データパケットの CBWCB フィールド（ストレージコマンド）は SCSI 仕様に従い作成します。このとき、CBW パケットの bCBWLUN フィールドとストレージコマンド内の LUN フィールドの設定をTable 7.1に示します。

Table 7.1 LUN フィールド設定

	bCBWLUN フィールド	コマンド内の LUN フィールド
usb_ghmsc_MaxLUN=0 の場合	0	0
usb_ghmsc_MaxLUN=0 でない場合	ユニット番号	0

7.3 論理ブロックアドレス変換

BOT 仕様では論理ブロックアドレスにしたがって情報の読み出し、書き込みを行います。また情報の読み出し、書き込みはバイト数でデータサイズを指定します。

論理ブロックアドレスはセクタ番号とオフセットセクタ番号から計算します。また、転送サイズはセクタ数とセクタサイズから計算します。

論理ブロックアドレス = 論理セクタ番号 + オフセットセクタ番号
転送サイズ = セクタ数 * セクタサイズ

7.4 HMSDD API 関数

Table 7.2に HMSDD API の関数を示します。

[Note]

ドキュメント(Document No: R01AN329JJ)に記載された API を使用する場合、アプリケーションプログラムでは、以下の API を使用する必要はありません。

Table 7.2 HMSDD 関数

関数名	説明
R_USB_HmscStrgDriveTask()	ストレージ機器情報を取得する。
R_USB_HmscStrgDriveSearch()	ドライブを検索する
R_USB_HmscStrgReadSector()	データを読み出す
R_USB_HmscStrgWriteSector()	データを書き込む
R_USB_HmscStrgUserCommand()	ストレージコマンドを発行する
R_USB_HmscAllocDrvno()	ドライブ番号を割り当てる
R_USB_HmscFreeDrvno()	ドライブ番号を解放する
R_USB_HmscRefDrvno()	ドライブ番号を参照する

7.4.1 R_USB_HmscStrgDriveTask

ストレージドライブタスク

形式

void	R_USB_HmscStrgDriveTask(void)

引数

―	―

戻り値

―	―

解説

SFF-8070i(ATAPI)コマンドをストレージ機器に送信し、接続されたストレージ機器の情報を取得する処理を行います。

補足

1. 本 API はアプリケーションプログラムから呼び出してください。

使用例

```
void usb_hapl_mainloop(void)
{
   while(1)
   {
      R_usb_cstd_Scheduler();

      if(USB_FLGSET == R_usb_cstd_CheckSchedule())
      {
            R_usb_hstd_HcdTask((USB_VP_INT)0);
            R_usb_hstd_MgrTask((USB_VP_INT)0);
            R_usb_hhub_Task((USB_VP_INT)0);
            R_usb_hmsc_Task((USB_VP_INT)0)
            R_usb_hmsc_StrgDriveTask();
      }
                  :
   }
}
```

7.4.2 R_USB_HsmcStrgDriveSearch

アクセス可能なドライブを検索する

形式

USB_ER_t R_USB_HmscStrgDriveSearch(USB_UTR_t *ptr, uint16_t addr, USB_CB_t complete)

引数

*ptr	USB 通信構造体
addr	デバイスアドレス
complete	コールバック関数

戻り値

USB_DONE	正常終了
USB_ERROR	エラー終了

解説

引数 addr で指定された USB デバイスに対し、クラスリクエスト(GetMaxLun)を送信し、ストレージ機器のユニット数を確認します。また、SFF-8070i(ATAPI)コマンドを送信し、ストレージ機器がアクセス可能かを確認します。ドライブ検索が完了すると引数 complete に設定したコールバック関数がコールされます。

補足

1. 本 API は FAT ライブラリ I/F 関数から呼び出してください。
2. USB 通信用構造体 USB_UTR_t の以下のメンバ設定が必要です。

USB_REGADR_t	ipp	: USB IP のアドレス
uint16_t	ip	: USB IP 番号

使用例

```
/* Callback function */
void usb_hmsc_StrgCommandResult( USB_UTR_t *mess, uint16_t data1, uint16_t data2 )
{
        :
}

void usb_hmsc_SampleAplTask(void)
{
        :
    R_usb_hmsc_StrgDriveSearch(mess, addr,(USB_CB_t)&usb_hmsc_StrgCommandResult);
        :
}
```

7.4.3 R_USB_HmscStrgReadSector

引数で指定されたドライブのセクタ情報を読み出す

形式

USB_ER_t R_USB_HmscStrgReadSector(USB_UTR_t *ptr, uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

引数

*ptr	USB 通信構造体
side	ドライブ番号
*buff	読み出しデータ格納領域
secno	セクタ番号
seccnt	セクタ数
trans_byte	転送データ長

戻り値

USB_DONE	正常終了
USB_ERROR	エラー終了

解説

引数で指定されたドライブのセクタ情報を読み出します
本 API は、ストレージ機器からドライブ情報が正しく読み込めなかった場合にエラーを返します。

補足

1. 本 API は FAT ライブラリ I/F 関数から呼び出してください。
2. USB 通信用構造体 USB_UTR_t の以下のメンバ設定が必要です。

USB_REGADR_t	ipp	: USB IP のアドレス
uint16_t	ip	: USB IP 番号

使用例

```
int read_sector(USB_UTR_t *ptr, int side, unsigned char *buff, unsigned long secno,
              long seccnt)
{
              :
    error = R_usb_hmsc_StrgReadSector(ptr, (uint16_t)side, buff, secno, (uint16_t)seccnt,
              trans_byte);

    if( error == USB_ERROR )
    {
              :
        return (-1);
    }
    return (0);
}
```

7.4.4 R_USB_HmscStrgWriteSector

引数で指定されたドライブへセクタ情報を書き込む

形式

USB_ER_t R_USB_HmscStrgWriteSector(USB_UTR_t *ptr, uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

引数

*ptr	USB 通信構造体
side	ドライブ番号
*buff	書き込みデータ格納領域
secno	セクタ番号
seccnt	セクタ数
trans_byte	転送データ長

戻り値

USB_DONE	正常終了
USB_ERROR	エラー終了

解説

引数で指定されたドライブのセクタ情報を書き込みます
本 API は以下の場合にエラーを返します。
1. ストレージ機器からドライブ情報が正しく読み込めなかった場合

補足

1. 本 API は FAT ライブラリ I/F 関数からから呼び出してください。
2. USB 通信用構造体 USB_UTR_t の以下のメンバ設定が必要です。

USB_REGADR_t	ipp	: USB IP のアドレス
uint16_t	ip	: USB IP 番号

使用例

```
int write_sector(USB_UTR_t *ptr, int side, unsigned char *buff, unsigned long secno,
              long seccnt)
{
              :
    error = R_USB_HmscStrgWriteSector(ptr, (uint16_t)side, buff, secno,
              (uint16_t)seccnt, trans_byte);

    if( error == USB_ERROR )
    {
              :
       return (-1);
    }
    return (0);
}
```

7.4.5 R_USB_HmscStrgUserCommand

ストレージコマンドを発行する

形式

uint16_t R_USB_HmscStrgUserCommand(USB_UTR_t *ptr, uint16_t side, uint16_t command , uint8_t *buff, USB_CB_t complete)

引数

*ptr	USB 通信構造体
side	ドライブ番号
command	発行するコマンド
*buff	データポインタ
complete	コールバック関数

戻り値

USB_DONE	正常終了
USB_ERROR	エラー終了

解説

引数で指定されたドライブへストレージコマンドを発行します。コマンドが完了すると引数 complete に指定したコールバック関数がコールされます。

以下に、R_USB_HmscStrgUserCommand が対応するストレージコマンドを示します。

ストレージコマンド	概要
USB_ATAPI_TEST_UNIT_READY	ペリフェラル機器の状態確認
USB_ATAPI_REQUEST_SENCE	ペリフェラル機器の状態取得
USB_ATAPI_INQUIRY	論理ユニットのパラメータ情報取得
USB_ATAPI_MODE_SELECT6	パラメータ指定
USB_ATAPI_PREVENT_ALLOW	メディアの取り出し許可/禁止
USB_ATAPI_READ_FORMAT_CAPACITY	フォーマット可能な容量取得
USB_ATAPI_READ_CAPACITY	論理ユニットの容量情報取得
USB_ATAPI_MODE_SENSE10	論理ユニットのパラメータ取得

補足

1. 本 API は、ユーザアプリケーションプログラムまたはクラスドライバで呼び出してください。
2. USB 通信用構造体 USB_UTR_t の以下のメンバ設定が必要です。

USB_REGADR_t	ipp	: USB IP のアドレス
uint16_t	ip	: USB IP 番号

使用例

```
/* Callback function */
void  strgcommand_complete(USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
          :
}

void usb_smp_task(void)
{
          :
 /* TEST_UNIT_READY 発行 */
     err = R_USB_HmscStrgUserCommand(ptr, side, USB_ATAPI_TEST_UNIT_READY, buf,
           (USB_CB_t)strgcommand_complete);
           :
}
```

7.4.6 R_USB_HmscAllocDrvno

ドライブ番号割り当て

形式

uint16_t R_USB_HmscAllocDrvno(uin16_t ipno, uint16_t devadr)

引数

ipno	USB モジュール番号
devadr	MSC デバイスのデバイスアドレス

戻り値

－	ドライブ番号

解説

接続された MSC デバイスに対しドライブ番号の割り当てを行います。

補足

1. 本 API によって割り当てられたドライブ番号は、FAT API の引数に指定することができます。
2. 引数 ipno には、MSC デバイスを接続した USB モジュールの番号(下記参照)を指定してください。

USB モジュール	USB モジュール番号
USB0	USB_IP0
USB1	USB_IP1

使用例

```
void usb_smp_task(void)
{
        :
    /* ドライブ番号割り当て */
    drvno = R_USB_HmscAllocDrvno(USB_IP1, devadr);
        :
}
```

7.4.7 R_USB_HmscFreeDrvno

ドライブ番号解放

形式

void R_USB_HmscFreeDrvno(uint16_t drvno)

引数

drvno ドライブ番号

戻り値

なし

解説

引数で指定されたドライブ番号を解放します。

補足

なし。

使用例

```
void usb_smp_task(void)
{
        :
   /* ドライブ番号解放 */
   R_USB_HmscFreeDrvno( drvno );
        :
}
```

7.4.8 R_USB_HmscRefDrvno

ドライブ番号参照

形式

uint16_t R_USB_HmscRefDrvno(uint16_t ipno, uint16_t devadr)

引数

ipno	USB モジュール番号
devadr	MSC デバイスのデバイスアドレス

戻り値

－	ドライブ番号

解説

引数で指定された USB モジュール番号とデバイスアドレスをもとにドライブ番号を参照します。

補足

1. 引数 ipno には、MSC デバイスを接続した USB モジュールの番号(下記参照)を指定してください。

USB モジュール	USB モジュール番号
USB0	USB_IP0
USB1	USB_IP1

使用例

```
void usb_smp_task(void)
{
        :
   /* ドライブ番号解放 */
   drvno = R_USB_HmscRefDrvno(USB_IP0, devadr);
        :
}
```

8. ホストマスストレージクラスドライバ（HMSCD）

8.1 機能と特徴

HMSCD は、接続された USB ストレージ機器の動作可否判定（デバイス照合）及び BOT プロトコルによるストレージコマンド通信を行います。

HMSCD は、HMSDD に対するインタフェース（DDI 関数）、HCD に対するインタフェース（HCI 関数）、HMSCD 本体の 3 層構造です。HMSCD は、USB ストレージ機器のアクセスに必須なストレージコマンドとサンプルのストレージコマンドに対応しています。

HMSCD の特徴を以下に示します。

- USB マスストレージクラスの BOT に対応しています。
- USB マスストレージサブクラスの SFF-8070i（ATAPI）に対応しています。
- データ転送に使用するパイプを、IN/OUT 転送で共有しています。

8.2 HMSCD に対する要求発行

USB ストレージ機器へアクセスする場合は、後述のインタフェース関数を用います。

HMSCD は、上位層からの要求に対してコールバック関数の戻り値で結果を通知します。

8.3 HMSCD 構造体

Table 8.1、Table 8.2に HMSCD が使用する構造体構成を示します。

Table 8.1 USB_MSC_CBW_t 構造体

型	メンバ名	説明	備考
uint32_t	dCBWSignature	CBW Signature	0x55534243: USBC
uint32_t	dCBWTag	CBW Tag	CSW に対応する Tag
uint8_t	dCBWDTL_Lo	CBW DataTransfer Length	送受信するデータのデータ長
uint8_t	dCBWDTL_ML		
uint8_t	dCBWDTL_MH		
uint8_t	dCBWDTL_Hi		
uint8_t	bmCBWFlags	CBW Direction	データ送受信方向
uint8_t	bCBWLUN	Logical Unit Number	ユニット番号
uint8_t	bCBWCBLength	CBWCB Length	コマンド長
uint8_t	CBWCB[16]	CBWCB	コマンドブロック

Table 8.2 USB_MSC_CSW_t 構造体

型	メンバ名	説明	備考
uint32_t	dCSWSignature	CSW Signature	0x55534253: USBS
uint32_t	dCSWTag	CSW Tag	CBW に対応する Tag
uint8_t	dCSWDataResidue_Lo	CSW DataResidue	使用されたデータ長
uint8_t	dCSWDataResidue_ML		
uint8_t	dCSWDataResidue_MH		
uint8_t	dCSWDataResidue_Hi		
uint8_t	bCSWStatus	CSW Status	コマンドステータス
uint8_t	dummy	dummy	偶数調整

8.4 USB ホストコントロールドライバインタフェース（HCI）

HCI は、HMSCD、HCD 間のインタフェース関数です。

RTOS の場合、HCI はドライブがマウントされるまで HCD のクラス領域を使用してメッセージを送受信します。ドライブのマウントが終了している場合は HMSCD 専用領域を使用します。

non-OS の場合、HMSCD の領域を使用してメッセージを送受信します。

なお、複数のデバイスに対し、同時にエニュメレーション、アクセスを行うことはできません。

8.5 デバイスドライバインタフェース（DDI）

DDI は、HMSDD、HMSCD 間のインタフェース関数です。

DDI 関数は HMSCD 起動関数、終了関数、接続デバイスのチェック関数及び、サンプルのストレージコマンド関数で構成されます。

8.6 HMSC API 関数

Table 8.3～Table 8.5に HMSCD の API 関数を示します。

[Note]

ドキュメント(Document No: R01AN329JJ)に記載された API を使用する場合、アプリケーションプログラムでは、以下の API を使用する必要はありません。

Table 8.3 HMSCD 関数

関数名	説明
R_USB_HmscGetDevSts()	HMSCD 動作状態の応答を行う
R_USB_HmscDriverStart()	HMSC ドライバ起動

Table 8.4 HCI 関数

関数名	説明
R_USB_HmscGetMaxUnit()	Get_MaxLUN リクエストを発行する
R_USB_HmscMassStorageReset()	MassStorageReset リクエストを発行する

Table 8.5 DDI 関数

関数名	説明
R_USB_HmscClassCheck()	接続デバイスの確認を行う
R_USB_HmscRead10()	READ10 コマンドを発行する
R_USB_HmscWrite10()	WRITE10 コマンドを発行する

8.6.1 R_USB_HmscGetDevSts

HMSCD 動作状態の応答を行う

形式

uint16_t R_USB_HmscGetDevSts(uint16_t drvno)

引数

drvno	ドライブ番号

戻り値

usb_ghmsc_AttSts	USB_HMSC_DEV_ATT（接続） USB_HMSC_DEV_DET（切断）

解説

HMSCD の動作状態を取得します。

補足

1. ユーザアプリケーションプログラムまたはクラスドライバで本 API を呼び出してください。
2. 引数 drvno には、R_usb_hmsc_alloc_drvno 関数によって割り当てられたドライブ番号を指定してください。

使用例

```
void  usb_smp_task(USB_UTR_t *ptr)
{
    :
  /* デバイス状態確認*/
  if(R_USB_HmscGetDevSts(drvno) == USB_HMSC_DEV_DET)
  {
    /* 切断処理 */
  }
    :
}
```

8.6.2 R_USB_HmscTask

ホストマスストレージクラスタスク

形式

void	R_USB_HmscTask(void)

引数

―	―

戻り値

―	―

解説

HMSCD のタスクです。
non-OS 時、BOT の制御を行います。

補足

1. スケジューラ処理を行うループ内で本 API を呼び出してください。

使用例

```
void  usb_smp_mainloop(void)
{
  while(1)
  {
  /* スケジューラ起動 */
  R_usb_cstd_Scheduler();
  /* フラグチェック */
  if(USB_FLGSET == R_usb_cstd_CheckSchedule())
  {
       :
    usb_hmsc_Task();
       :
  }
    :
}
```

8.6.3 R_USB_HmscDriverStart

HMSC ドライバ起動

形式

void R_USB_HmscDriverStart(USB_UTR_t *ptr)

引数

*ptr	USB 通信構造体

戻り値

―	―

解説

HMSC ドライバタスクの優先度を設定します。

優先度が設定されることで、メッセージの送受信が可能になります。

補足

1. 初期設定時にユーザアプリケーションで本 API を呼び出してください。
2. USB 通信用構造体 USB_UTR_t の以下のメンバ設定が必要です。

USB_REGADR_t	ipp	: USB IP のアドレス
uint16_t	ip	: USB IP 番号

使用例

```
void usb_hstd_task_start( void )
{
  USB_UTR_t  *ptr;
    :
  ptr->ip = USB_HOST_USBIP_NUM;   /* USB IP 番号設定 */
  ptr->ipp = R_usb_cstd_GetUsbIpAdr( ptr->ip );  /* USB IP アドレス設定 */
    :
  R_USB_HmscDriverStart( ptr );      /* Host Class Driver Task Start Setting */
    :
}
```

8.6.4 R_USB_HmscGetMaxUnit

Get_MaxLUN リクエストを発行する

形式

USB_ER_t R_USB_HmscGetMaxUnit(USB_UTR_t *ptr, uint16_t addr, USB_CB_t complete)

引数

*ptr	USB 通信構造体
addr	デバイスアドレス
complete	コールバック関数

戻り値

USB_E_OK	GET_MAX_LUN 発行
USB_E_ERROR	GET_MAX_LUN 未発行
USB_E_QOVR	GET_MAX_LUN 未発行

解説

GET_MAX_LUN リクエストを発行して、最大ユニット数を取得します。リクエストが完了すると引数 comlete に指定したコールバック関数がコールされます。

補足

1. ユーザアプリケーションプログラムまたはクラスドライバで本 API を呼び出してください。
2. USB 通信用構造体 USB_UTR_t の以下のメンバ設定が必要です。

USB_REGADR_t	ipp	: USB IP のアドレス
uint16_t	ip	: USB IP 番号

使用例

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  USB_ER_t  err;
    :
  /* 最大ユニット数取得*/
  err = R_USB_HmscGetMaxUnit(mess, addr, (USB_CB_t)usb_hmsc_StrgCheckResult);
  if(err == USB_E_QOVR)
  {
    /* エラー処理 */
  }
    :
}
```

8.6.5 R_USB_HmscMassStorageReset

MassStorageReset リクエストを発行する

形式

USB_ER_t R_USB_HmscMassStorageReset(USB_UTR_t *ptr, uint16_t drvnum, USB_CB_t complete)

引数

*ptr	USB 通信構造体
drvnum	ドライブ番号
complete	コールバック関数

戻り値

USB_E_OK	MASS_STORAGE_RESET 発行
USB_E_ERROR	MASS_STORAGE_RESET 未発行
USB_E_QOVR	MASS_STORAGE_RESET 未発行

解説

MASS_STORAGE_RESET リクエストを発行して、プロトコルエラーを解除します。リクエストが完了すると引数 comlete に指定したコールバック関数がコールされます。

補足

1. ユーザアプリケーションプログラムまたはクラスドライバで本 API を呼び出してください。
2. USB 通信用構造体 USB_UTR_t の以下のメンバ設定が必要です。

USB_REGADR_t	ipp	: USB IP のアドレス
uint16_t	ip	: USB IP 番号

使用例

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  USB_ER_t  err;
    :
  /* プロトコルエラー解除*/
  err = R_USB_HmscMassStorageReset(ptr, drvnum, (USB_CB_t)usb_hmsc_CheckResult);
  if(err == USB_E_QOVR)
  {
    /* エラー処理 */
  }
    :
}
```

8.6.6 R_USB_HmscClassCheck

接続デバイスの確認を行う

形式

void R_USB_HmscClassCheck(USB_UTR_t *ptr, uint16_t **table)

引数

*ptr	USB 通信構造体
**table	未使用

戻り値

―	―

解説

デバイス数、ドライブ数のチェックと、インタフェースディスクリプタテーブルの解析を行います。
下記項目にて HMSCD とのマッチングを確認し、動作可能の場合はシリアル番号を読み出します。
Bulk エンドポイントディスクリプタテーブルでパイプ情報テーブルを更新（Endpoint アドレス、Max パケットサイズ等）します。
インタフェースディスクリプタの情報確認
bInterfaceSubClass = USBC_ATAPI / USBC_SCSI
bInterfaceProtocol = USBC_BOTP
bNumEndpoint > USBC_TOTALEP
ストリング Descriptor の情報確認
12 文字以上のシリアル番号（エラー時は警告表示）
エンドポイント Descriptor の情報確認
bmAtributes = 0x02（Bulk エンドポイントが必要）
bEndpointAdress（IN/OUT 双方のエンドポイントが必要）

補足

1. アプリケーションプログラム内で USB_HCDREG_t 構造体のメンバ classcheck にこの API を設定し、この API をコールバック関数として登録してください。
2. USB 受信処理結果はコールバック関数の引数”USB_UTR_t *ptr”で得られます。
3. USB_UTR_t 構造体については、USB Basic Fir mware アプリケーションノートの USB 通信用構造体を参照して下さい。
4. 動作可能なドライブ数は USB_MAXDRIVE で定義されています。(r_usb_hmsc_define.h 参照)

使用例

```
void  usb_hapl_registration(USB_UTR_t *ptr)
{
    :
  /* Driver check */
  driver.classcheck   = &R_USB_HmscClassCheck;
    :
}
```

8.6.7 R_USB_HmscRead10

READ10 コマンドを発行

形式

uint16_t R_USB_HmscRead10(USB_UTR_t *ptr, uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

引数

*ptr	USB 通信構造体
side	ドライブ番号
*buff	読み出しデータエリア
secno	セクタ番号
seccnt	セクタ数
trans_byte	転送データ長

戻り値

－	エラーコード。

解説

USB デバイスに READ10 コマンドを発行します。
コマンドエラーの場合は REQUEST_SENSE コマンドを発行しエラー情報を取得します。

補足

1. ユーザアプリケーションプログラムまたはクラスドライバで本 API を呼び出してください。
2. USB 通信用構造体 USB_UTR_t の以下のメンバ設定が必要です。

USB_REGADR_t	ipp	: USB IP のアドレス
uint16_t	ip	: USB IP 番号

使用例

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  uint32_t  result;
    :
  /* READ10 発行 */
  result = R_USB_HmscRead10(ptr, side, buff, secno, seccnt, trans_byte);
  if(result != USB_HMSC_OK)
  {
    /* エラー処理 */
  }
    :
}
```

8.6.8 R_USB_HmscWrite10

WRITE10 コマンドを発行する

形式

uint16_t R_USB_HmscWrite10(USB_UTR_t *ptr, uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

引数

*ptr	USB 通信構造体
side	ドライブ番号
*buff	読み出しデータエリア
secno	セクタ番号
seccnt	セクタ数
trans_byte	転送データ長

戻り値

－	エラーコード。

解説

USB デバイスに WRITE10 コマンドを発行します。
コマンドエラーの場合は REQUEST_SENSE コマンドを発行しエラー情報を取得します。

補足

1. ユーザアプリケーションプログラムで本 API を呼び出してください。
2. USB 通信用構造体 USB_UTR_t の以下のメンバ設定が必要です。

USB_REGADR_t	ipp	: USB IP のアドレス
uint16_t	ip	: USB IP 番号

使用例

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  uint32_t  result;
    :
  /* WRITE10 発行 */
  result = R_USB_HmscWrite10(ptr, side, buff, secno, seccnt, trans_byte);
  if(result != USB_HMSC_OK)
  {
    /* エラー処理 */
  }
    :
}
```

9. サンプルアプリケーション

9.1 アプリケーション仕様

HMSC のサンプルアプリケーション(以降、APL) の主な機能を以下に示します。

1. MSC デバイスに対するエニュメレーション処理およびドライブ認識処理を行います。
2. 上記1完了後、ファイル'hmscdemo.txt'を一回、MSC デバイスに対し書き込みます。
3. 上記書き込み後、ファイル'hmscdemo.txt'のリード処理を繰り返します。
4. USB Hub を使用することにより、最大 4 つの MSC デバイスに対し上記1から 3 の処理を行うことができます。

[Note]

ファイル'hmscdemo.txt'が格納された MSC デバイスを接続した場合、'hmscdemo.txt'が上書きされます。
APL はデバイス接続時、ドライブ認識処理完了時、デタッチ時にターミナルソフトにログを表示します。
ターミナルソフトの設定は、「Table 1.1　動作確認条件」を参照してください。

9.2 アプリケーション処理概要

APL は、初期設定、メインループの 2 つの部分から構成されます。以下にそれぞれの処理概要を示します。

9.2.1 初期設定

初期設定では、USB コントローラの初期設定およびアプリケーションプログラムの初期化処理を行います。

9.2.2 メインループ

このメインループでは、R_USB_GetEvent 関数からの戻り値とステートを使ってプログラム制御を行っています。USB のイベント管理(アタッチ/デタッチ等)を、R_USB_GetEvent 関数の戻り値により行い、MSC デバイスに対するファイルライト/ファイルリードをステート変数の状態参照により行います。

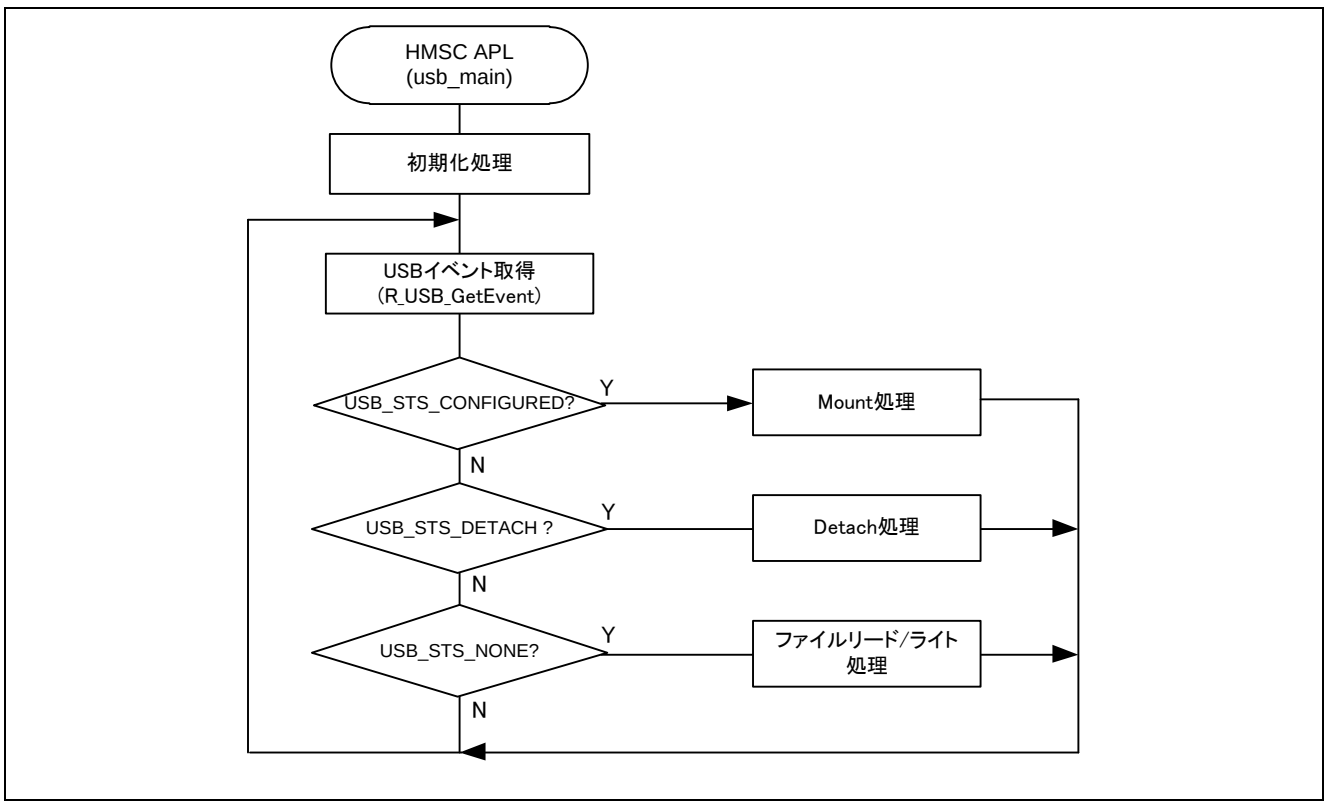

Figure 9-1 メインループ処理

1) Mount 処理（USB_STS_CONFIGURED）

GENMAI ボードに MSC デバイスが ATTACH され、Enumeration およびドライブ認識処理完了後に R_USB_GetEvent 関数をコールすると戻り値に USB_STS_CONFIGURED がセットされます。アプリケーションプログラムでは、戻り値に USB_STS_CONFIGURED がセットされていることを確認すると、Mount 処理等を行った後、ファイルライト/リード処理を行うため、ステート管理用変数に対し STATE_FILE_WRITE を設定します。

2) ファイルライト/ファイルリード処理（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に USB_STS_NONE がセットされます。アプリケーションプログラムでは、戻り値に USB_STS_NONE がセットされていることを確認すると以下の処理を行います。本アプリケーションプログラムは、USB Hub を使って最大 4 つの MSC デバイスに対しファイルライト/ファイルリードを行います。各 MSC デバイスの管理はステート管理用変数を使ってデバイスアドレスにより行います。

a. ステート管理用変数が STATE_FILE_WRITE の場合
FAT API を使って MSC デバイスに対しファイルライト処理を行います。ファイルライト完了後、ステート管理用変数に STATE_FILE_READ を設定します。

b. ステート管理用変数が STATE_FILE_READ の場合
FAT API を使って MSC デバイスに対しファイルリード処理を行います。MSC デバイスがデタッチされるまで MSC デバイスに対するファイルリード処理が継続されます。

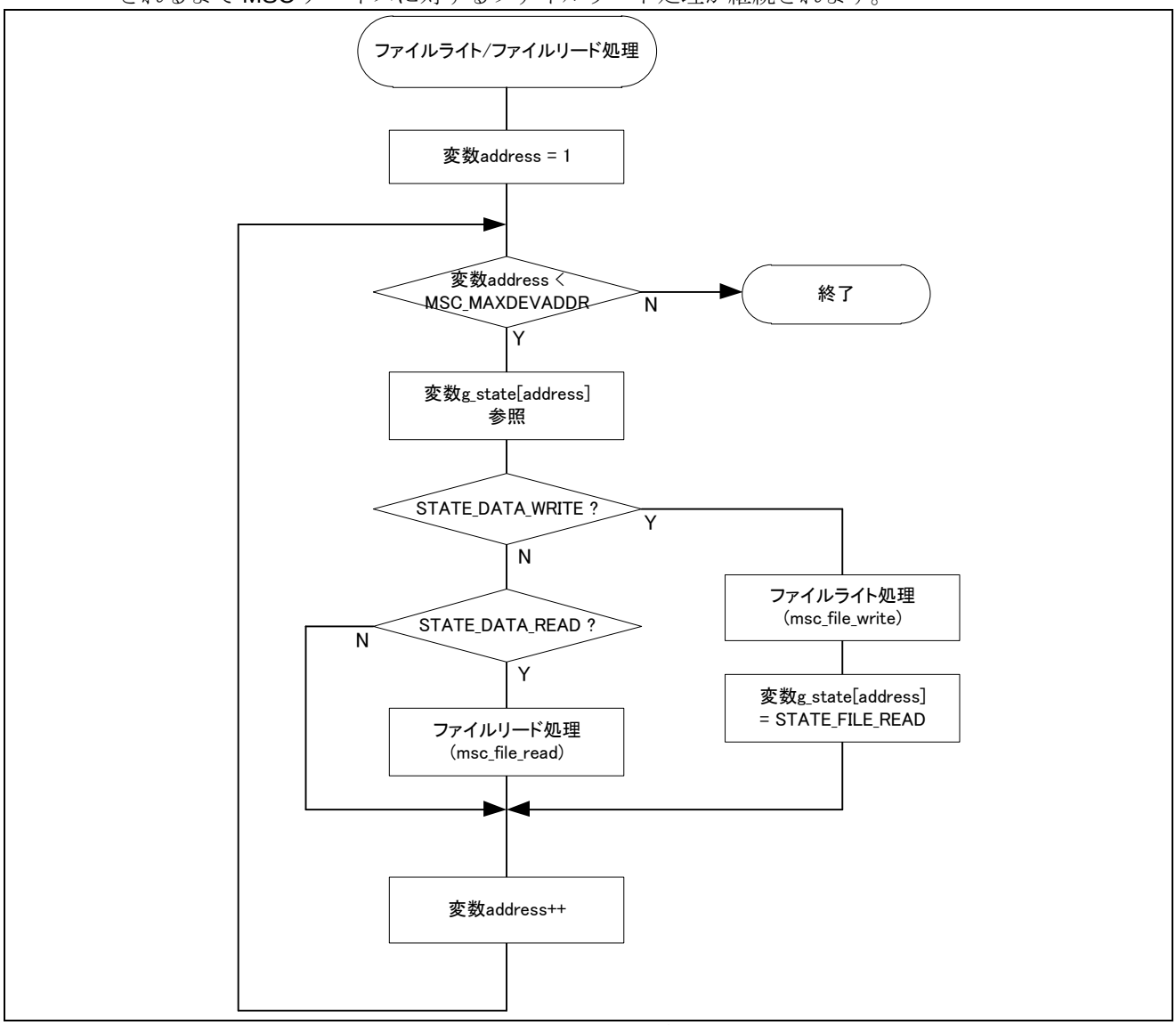

Figure 9-2 ファイルリード/ライト処理

3) Detach 処理（USB_STS_DETACH）

GENMAI ボードから MSC デバイスが DETACH された後、R_USB_GetEvent 関数をコールすると戻り値に USB_STS_DETACH がセットされます。アプリケーションプログラムでは、戻り値に USB_STS_DETACH がセットされていることを確認すると、ステート管理用変数に対し STATE_DETACH を設定します。

10. セットアップ

10.1 ハードウェア

HMSC の動作環境例をFigure 10-1に示します。評価ボードのセットアップ、エミュレータなどの使用方法については各取扱説明書を参照ください。

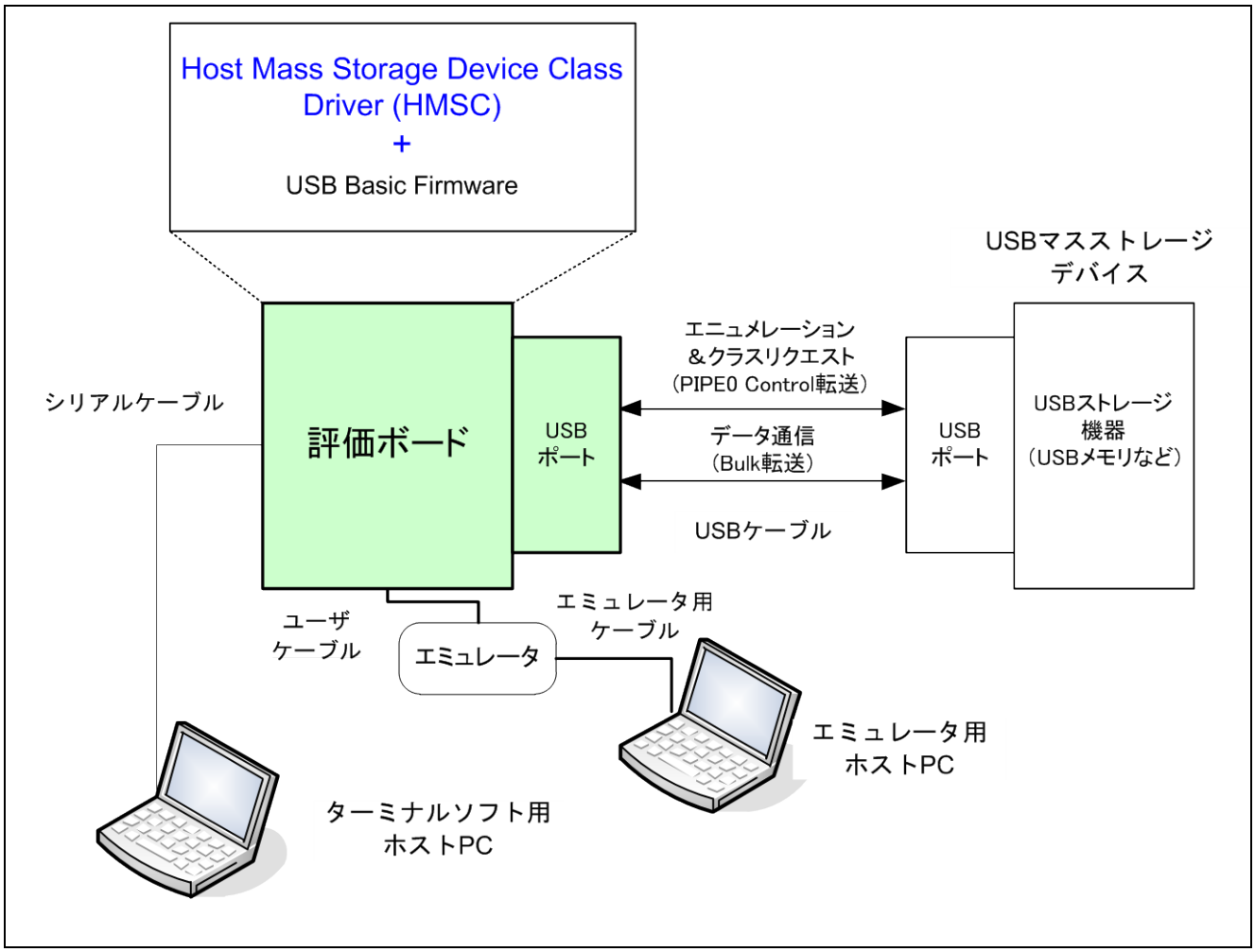

Figure 10-1 動作環境例

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	2016.06.01	—	初版発行
1.10	2016.09.30	—	USB-BASIC-F/W が改定されたため、バージョンアップ

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、本文を参照してください。なお、本マニュアルの本文と異なる記載がある場合は、本文の記載が優先するものとします。

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. リザーブアドレスのアクセス禁止

   【注意】リザーブアドレス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、事前に問題ないことをご確認下さい。

   同じグループのマイコンでも型名が違うと、内部メモリ、レイアウトパターンの相違などにより、特性が異なる場合があります。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。

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、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 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に関し、当社は、何らの責任を負うものではありません。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. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製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）、交通用信号機器、
   防災・防犯装置、各種安全装置等
   当社製品は、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（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障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、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、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。お客様が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. 本資料に記載されている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。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
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。

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その総株主の議決権の過半数を直接または間接に保有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。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

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

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

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