RZ/A2M グループ

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。

r_usbh0_basic_config.h および r_usbh1_basic_config.h 内で「#define USBH0_CFG_HMSC_USE」および「#define USBH1_CFG_HMSC_USE」と定義してご使用ください。

動作確認デバイス

RZ/A2M 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。

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。

API などの名称は、ポート 0 とポート 1 で異なります。

本書では、API 名称などはポート 0 のものを例として記載します。

表 1.1 使用する周辺機能と用途

周辺機能	用途
ホスト PC	サンプルコードのメッセージ出力

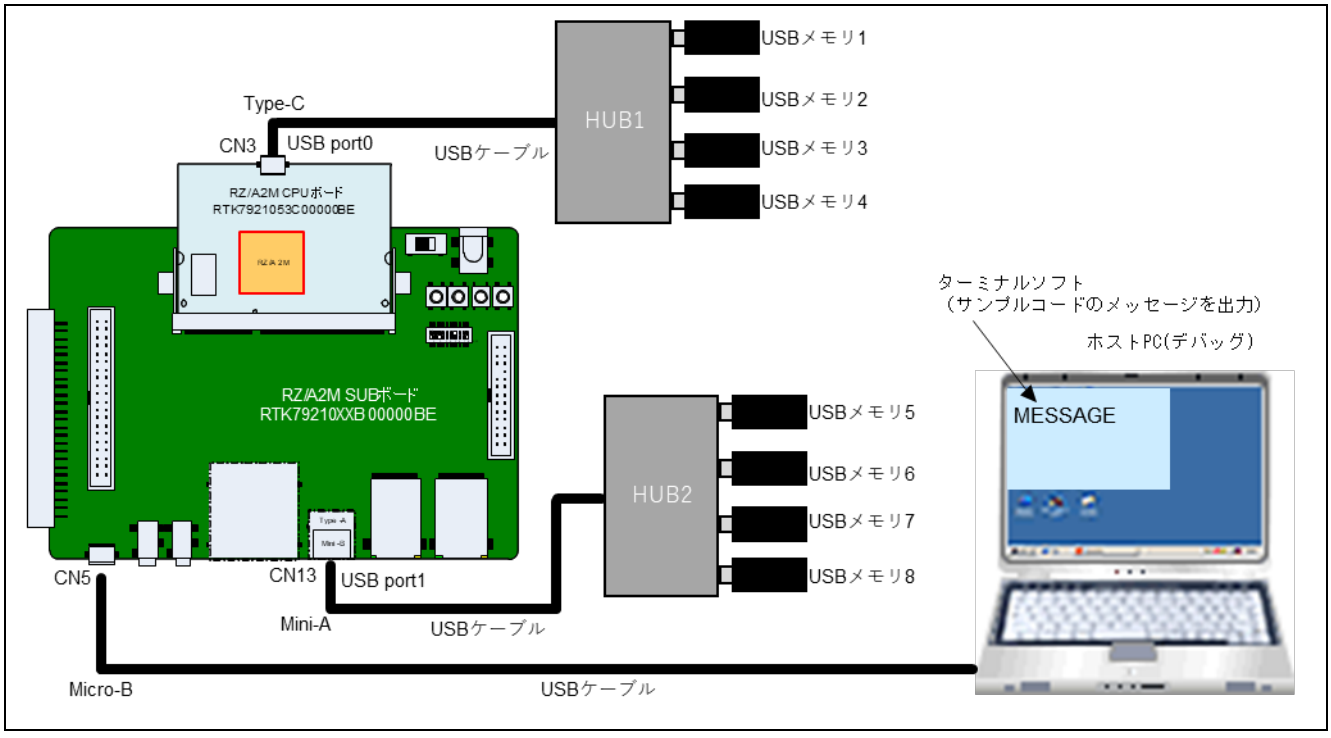

図 1.1 動作環境

1.1 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。）
- 論理ユニット番号 0 (LUN0) のみサポートします。
- セクタサイズが 512 バイトの USB ストレージ機器のみサポートします。
- 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 バイトとして処理します。

1.2 注意事項

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。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もとより、多種多様なデバイスに対する接続確認を実施してください。

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
ATAPI	AT Attachment Packet Interface
BOT	Mass storage class Bulk Only Transport
CBW	Command Block Wrapper
CSW	Command Status Wrapper
FSI	File System Interface
HCD	Host Control Driver of USB-BASIC-F/W
HMSC	Host Mass Storage Class driver
HMSCD	Host Mass Storage Class Driver unit
HMSDD	Host Mass Storage Device Driver
HUBCD	Hub Class Driver
LUN	Logical Unit Number
LBA	Logical Block Address
MGR	Peripheral device state manager of HCD
SCSI	Small Computer System Interface
Scheduler	タスク動作を簡易的にスケジューリングするモジュール
Task	処理の単位
USB-BASIC-F/W	USB basic firmware for RZ/A2Mグループ
USB	Universal Serial Bus

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 V7.6.0
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. 関連アプリケーションノート

本アプリケーションノートに関連するアプリケーションノートを以下に示します。併せて参照してください。

・USB Basic Host Driver アプリケーションノート(Document No. R01AN4715JJ0120)

4. 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。

図 4.1 に HMSC のモジュール構成、表 4.1 にモジュール機能概要を示します。

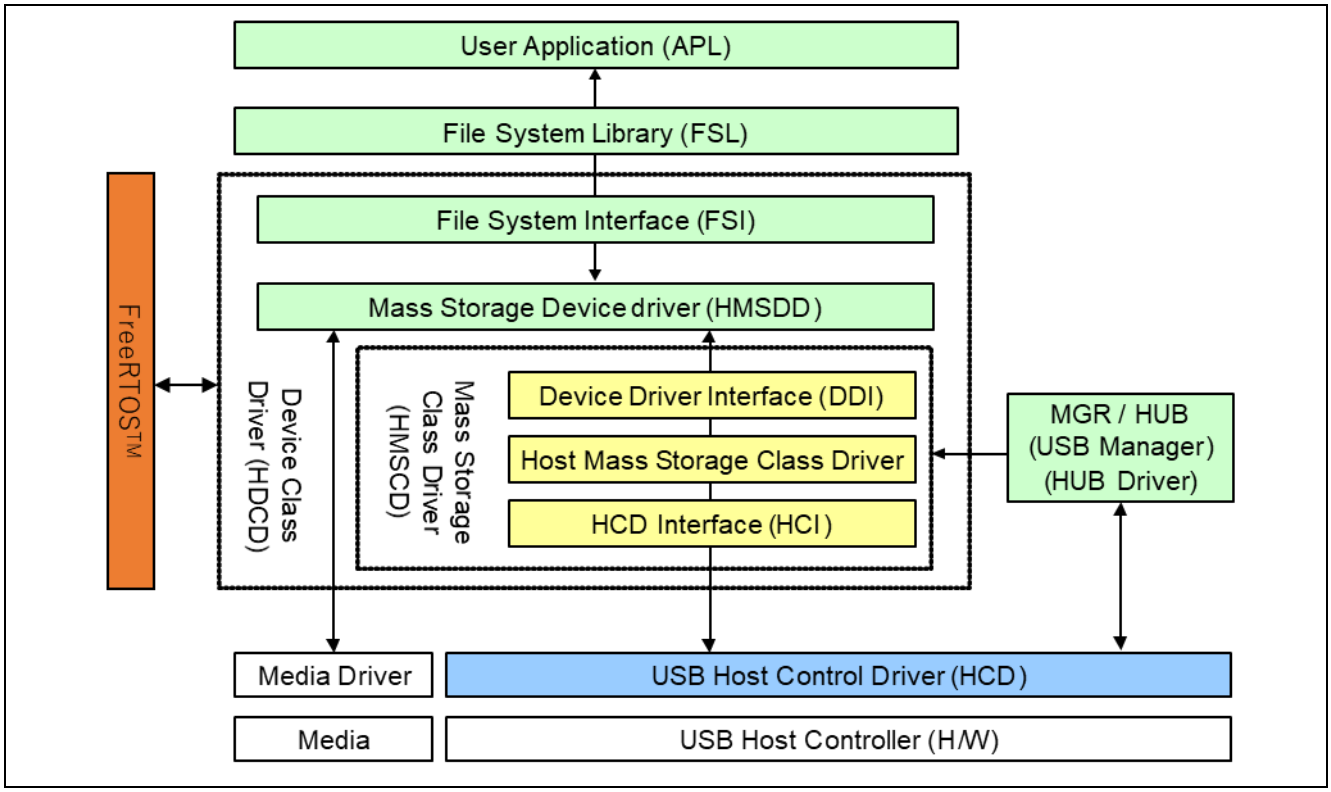

図 4.1 ソフトウェア構成図

表 4.1 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
FreeRTOS™	FreeRTOS™

5. システム資源

表 5.1～表 5.3 に、HMSC が使用しているシステム資源を示します。

表 5.1 タスク情報

関数名	タスク ID	優先度	概要
R_USBH0_HmscTask	USB_HMSC_TSK	USB_PRI_3	マスストレージクラスタスク
R_USBH0_HmscStrgDriveTask	USB_HSTRG_TSK	USB_PRI_3	HMSDD タスク

表 5.2 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

表 5.3 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

（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。

6. 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 Basic Host Driver アプリケーションノート(Document No. R01AN4715JJ0120)の「5.4 ターゲットペリフェラルリスト（TPL）」を参照してください。

7. USB ストレージ機器へのアクセス

情報取得完了後は、FatFs の API 関数で USB ストレージ機器にアクセスすることができます。

FatFs の API 関数をコールすると、ファイルシステム処理途中で FSI がコールされ HMSDD が動作します。

HMSDD は、処理に応じた HMSCD の API 関数をコールします。

それによって動作する HMSDC は、クラスリクエストの発行や BOT プロトコルに従った USB パケットの作成をします。

BOT プロトコルでは LBA に従ってデータ転送を行い、バイト数で転送サイズを指定します。

図 7.1 に USB ストレージ機器へのアクセスシーケンスを示します。

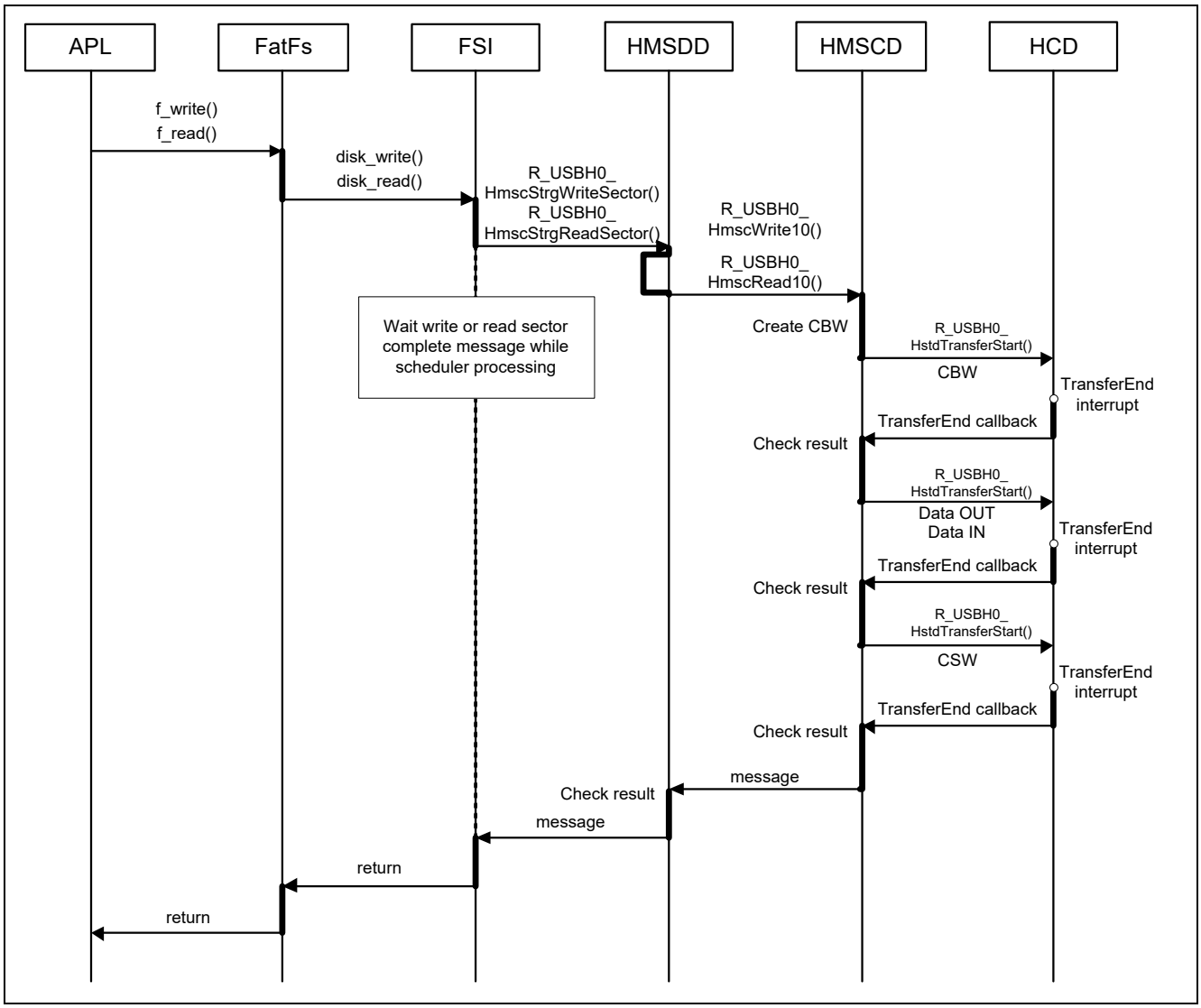

図 7.1 USB ストレージ機器へのアクセスシーケンス

8. ファイルシステムインタフェース（FSI）

8.1 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。

8.2 FSI API 関数

表 8.1 にサンプルの FSI API 関数を示します。

表 8.1 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

9. 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。

9.1 制限事項

HMSDD には以下の制限があります。

・ 1 ポートあたり最大 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。

9.2 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表 9.1 に示します。

表 9.1 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

9.3 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

9.4 HMSDD API 関数

表 9.2 に HMSDD API の関数を示します。

表 9.2 HMSDD 関数

関数名	ポート	説明
R_USBH0_HmscStrgDriveTask() R_USBH1_HmscStrgDriveTask()	0 1	ストレージ機器情報を取得する。
R_USBH0_HmscStrgDriveSearch() R_USBH1_HmscStrgDriveSearch()	0 1	ドライブを検索する
R_USBH0_HmscStrgReadSector() R_USBH1_HmscStrgReadSector()	0 1	データを読み出す
R_USBH0_HmscStrgWriteSector() R_USBH1_HmscStrgWriteSector()	0 1	データを書き込む
R_USBH0_HmscStrgUserCommand() R_USBH1_HmscStrgUserCommand()	0 1	ストレージコマンドを発行する

9.4.2 R_USBH0_HmscStrgDriveTask

ストレージドライブタスク

形式

void	R_USBH0_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_USBH0_CstdScheduler();

    if(USBH0_FLGSET == R_USBH0_CstdCheckSchedule())
    {
         R_USBH0_HstdMgrTask();             /* MGR task */
         R_USBH0_HhubTask();                /* HUB task */
         R_USBH0_HmscTask();                /* HMSC Task */
         R_USBH0_HmscStrgDriveTask();       /* HSTRG Task */
    }
                :
  }
}
```

9.4.4 R_USBH0_HmscStrgDriveSearch

アクセス可能なドライブを検索する

形式

uint16_t R_USBH0_HmscStrgDriveSearch(uint16_t addr, usbh0_utr_cb_t complete)

引数

addr	デバイスアドレス
complete	コールバック関数

戻り値

USBH0_OK	正常終了
USBH0_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。

使用例

```
/* Callback function */
void usb_hmsc_StrgCommandResult( st_usbh0_utr_t *utr )
{
        :
}

void usb_hmsc_SampleAplTask(void)
{
        :
    R_USBH0_HmscStrgDriveSearch(addr, usb_hmsc_StrgCommandResult);
        :
}
```

9.4.6 R_USBH0_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

uint16_t	R_USBH0_HmscStrgReadSector(uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

引数

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

USBH0_OK	正常終了
USBH0_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。

使用例

```
int read_sector(int side, unsigned char *buff, unsigned long secno, long seccnt)
{
                :
    error = R_USBH0_HmscStrgReadSector(uint16_t)side, buff, secno, (uint16_t)seccnt,
                trans_byte);

    if( error == USBH0_ERROR )
    {
                :
        return (-1);
    }
    return (0);
}
```

9.4.7 R_USBH0_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

uint16_t	R_USBH0_HmscStrgWriteSector(uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

引数

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

USBH0_OK	正常終了
USBH0_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。

使用例

```
int write_sector(int side, unsigned char *buff, unsigned long secno, long seccnt)
{
              :
  error = R_USBH0_HmscStrgWriteSector((uint16_t)side, buff, secno, (uint16_t)seccnt,
              trans_byte);

    if( error == USBH0_ERROR )
    {
              :
        return (-1);
    }
    return (0);
}
```

9.4.9 R_USBH0_HmscStrgUserCommand

ストレージコマンドを発行する

形式

uint16_t R_USBH0_HmscStrgUserCommand(uint16_t side, uint16_t command , uint8_t *buff, usbh0_utr_cb_t complete)

引数

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

USBH0_OK	正常終了
USBH0_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_USBH0_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。

使用例

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

void   usb_smp_task(void)
{
            :
  /* TEST_UNIT_READY 発行 */
  err = R_USBH0_HmscStrgUserCommand(side, USB_ATAPI_TEST_UNIT_READY, buf,
           strgcommand_complete);
            :
}
```

10. ホストマスストレージクラスドライバ（HMSCD）

10.1 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。

10.2 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。

10.3 HMSCD 構造体

表 10.1、表 10.2 に HMSCD が使用する構造体構成を示します。

表 10.1 st_usbh0_hmsc_cbw_t 構造体

型	メンバ名	説明	備考
uint32_t	dcbw_signature	CBW Signature	0x55534243: USBC
uint32_t	dcbw_tag	CBW Tag	CSW に対応する Tag
uint8_t	dcbw_dtl_lo	CBW DataTransfer Length	送受信するデータのデータ長
uint8_t	dcbw_dtl_ml		
uint8_t	dcbw_dtl_mh		
uint8_t	dcbw_dtl_hi		
uint8_t	bmcbw_flags	CBW Direction	データ送受信方向
uint8_t	bcbw_lun	Logical Unit Number	ユニット番号
uint8_t	bcbw_cb_length	CBWCB Length	コマンド長
uint8_t	cbw_cb[16]	CBWCB	コマンドブロック

表 10.2 st_usbh0_hmsc_csw_t 構造体

型	メンバ名	説明	備考
uint32_t	dcsw_signature	CSW Signature	0x55534253: USBS
uint32_t	dcsw_tag	CSW Tag	CBW に対応する Tag
uint8_t	dcsw_data_residue_lo	CSW DataResidue	使用されたデータ長
uint8_t	dcsw_data_residue_ml		
uint8_t	dcsw_data_residue_mh		
uint8_t	dcsw_data_residue_hi		
uint8_t	bcsw_status	CSW Status	コマンドステータス

10.4 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。

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。

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。

10.5 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。

10.6 HMSC API 関数

表 10.3～表 10.5 に HMSCD の API 関数を示します。

表 10.3 HMSCD 関数

関数名	ポート	説明
R_USBH0_HmscGetDevSts() R_USBH1_HmscGetDevSts()	0 1	HMSCD 動作状態の応答を行う
R_USBH0_HmscTask() R_USBH1_HmscTask()	0 1	HMSCD のタスク処理
R_USBH0_HmscDriverStart() R_USBH1_HmscDriverStart()	0 1	HMSC ドライバ起動
R_USBH0_HmscAllocDrvno() R_USBH1_HmscAllocDrvno()	0 1	ドライブ番号を割り当てる
R_USBH0_HmscFreeDrvno() R_USBH1_HmscFreeDrvno()	0 1	ドライブ番号を解放する
R_USBH0_HmscRefDrvno() R_USBH1_HmscRefDrvno()	0 1	ドライブ番号を参照する

表 10.4 HCI 関数

関数名	ポート	説明
R_USBH0_HmscGetMaxUnit() R_USBH1_HmscGetMaxUnit()	0 1	Get_MaxLUN リクエストを発行する
R_USBH0_HmscMassStorageReset() R_USBH1_HmscMassStorageReset()	0 1	MassStorageReset リクエストを発行する

表 10.5 DDI 関数

関数名	ポート	説明
R_USBH0_HmscClassCheck() R_USBH1_HmscClassCheck()	0 1	接続デバイスの確認を行う
R_USBH0_HmscRead10() R_USBH1_HmscRead10()	0 1	READ10 コマンドを発行する
R_USBH0_HmscWrite10() R_USBH1_HmscWrite10()	0 1	WRITE10 コマンドを発行する

10.6.2 R_USBH0_HmscGetDevSts

HMSCD 動作状態の応答を行う

形式

uint16_t R_USBH0_HmscGetDevSts(uint16_t drvno)

引数

drvno ドライブ番号

戻り値

USBH0_TRUE 接続
USBH0_FALSE 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。

使用例

```
void   usb_smp_task()
{
    :
  /* Checking the device state */
  if(R_USBH0_HmscGetDevSts(drvno) == USBH0_FALSE)
  {
    /* Detach processing */

  }
    :
}
```

10.6.4 R_USBH0_HmscTask

ホストマスストレージクラスタスク

形式

void	R_USBH0_HmscTask(void)

引数

―	―

戻り値

―	―

解説

HMSCD のタスクです。
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)
    {
    /* Scheduler processing */
    R_USBH0_CstdScheduler();
    /* Checking flag */
    if(USBH0_FLGSET == R_USBH0_CstdCheckSchedule())
    {
          :
      R_USBH0_HmscTask();
          :
    }
      :
}
```

10.6.6 R_USBH0_HmscDriverStart

HMSC ドライバ起動

形式

void	R_USBH0_HmscDriverStart(void)

引数

―	―

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。

使用例

```
void usb_hstd_task_start( void )
{
    :
  R_USBH0_HmscDriverStart();     /* Host Class Driver Task Start Setting */
    :
}
```

10.6.7 R_USBH0_HmscAllocDrvno

ドライブ番号割り当て

形式

uint16_t R_USBH0_HmscAllocDrvno(uin16_t *p_side, uint16_t devadr)

引数

*p_side	ドライブ番号(出力)
devadr	MSC デバイスのデバイスアドレス

戻り値

USBH0_OK	ドライブ番号の割り当て成功
USBH0_ERROR	ドライブ番号の割り当て失敗

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。

使用例

```
void  usb_smp_task(void)
{
   uint16_t    *side;
   uint16_t    devadr;
   uint16_t    err;
            :
   /* Allocates the drive number */
   drvno = R_USBH0_HmscAllocDrvno(side, devadr);
            :
}
```

10.6.9 R_USBH0_HmscFreeDrvno

ドライブ番号解放

形式

uint16_t R_USBH0_HmscFreeDrvno(uint16_t side)

引数

side	ドライブ番号

戻り値

USBH0_OK	ドライブ番号の解放成功
USBH0_ERROR	ドライブ番号の解放失敗

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)
{
   uint16_t    devno;
   uint16_t    err;
            :
   /* Frees the drive number */
    err = R_USBH0_HmscFreeDrvno(drvno);
    if(err == USBH0_ERROR)
    {
         /* Error processing */
    }
            :
}
```

10.6.10 R_USBH0_HmscRefDrvno

ドライブ番号参照

形式

uint16_t R_USBH0_HmscRefDrvno(uint16_t *p_side, uint16_t devadr)

引数

*p_side	ドライブ番号(出力)
devadr	MSC デバイスのデバイスアドレス

戻り値

USBH0_OK	ドライブ番号の参照成功
USBH0_ERROR	ドライブ番号の参照失敗

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

なし

使用例

```
void  usb_smp_task(void)
{
    uint16_t    *side;
    uint16_t    devadr;
    uint16_t    err;
            :
    /* Frees the drive number */
    err = R_USBH0_HmscRefDrvno(side, devadr);
    if(err == USBH0_ERROR)
    {
        /* Error processing */
    }
            :
}
```

10.6.12 R_USBH0_HmscGetMaxUnit

Get_MaxLUN リクエストを発行する

形式

usbh0_er_t R_USBH0_HmscGetMaxUnit(uint16_t addr, usbh0_utr_cb_t complete)

引数

addr	デバイスアドレス
complete	コールバック関数

戻り値

USBH0_OK	GET_MAX_LUN 発行
USBH0_ERROR	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 complete 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。

使用例

```
void   usb_smp_task()
{
  usbh0_er_t   err;
    :
  /* Getting Max unit number */
  err = R_USBH0_HmscGetMaxUnit(addr, usb_hmsc_StrgCheckResult);
  if(err == USBH0_ERROR)
  {
    /* Error processing */
  }
    :
}
```

10.6.14 R_USBH0_HmscMessStorageReset

MassStorageReset リクエストを発行する

形式

usbh0_er_t R_USBH0_HmscMassStorageReset(uint16_t drvnum, usbh0_utr_cb_t complete)

引数

drvnum	ドライブ番号
complete	コールバック関数

戻り値

USBH0_OK	MASS_STORAGE_RESET 発行
USBH0_ERROR	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 complete 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。

使用例

```
void   usb_smp_task()
{
  usbh0_er_t     err;
    :
  /* Cancel the protocol error */
  err = R_USBH0_HmscMassStorageReset(drvnum, usb_hmsc_CheckResult);
  if(err == USBH0_ERROR)
  {
    /* Error processing */
  }
    :
}
```

10.6.16 R_USBH0_HmscClassCheck

接続デバイスの確認を行う

形式

void	R_USBH0_HmscClassCheck(uint16_t **table)

引数

**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 st_usbh0_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. 動作可能なドライブ数は USBH0_MAXSTRAGE で定義されています。(r_usbh0_hmsc_config.h 参照)

使用例

```
void   usb_hapl_registration()
{
    :
  /* Driver check */
  driver.classcheck = &R_USBH0_HmscClassCheck;
    :
}
```

10.6.18 R_USBH0_HmscRead10

READ10 コマンドを発行

形式

uint16_t R_USBH0_HmscRead10(uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

引数

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

USBH0_OK

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。

使用例

```
void   usb_smp_task()
{
  uint32_tresult;
    :
  /* Issuing READ10 */
  R_USBH0_HmscRead10(side, buff, secno, seccnt, trans_byte);
    :
}
```

10.6.20 R_USBH0_HmscWrite10

WRITE10 コマンドを発行する

形式

uint16_t	R_USBH0_HmscWrite10(uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

引数

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

USBH0_OK

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。

使用例

```
void   usb_smp_task()
{
    :
  /* Issuing WRITE10 */
  R_USBH0_HmscWrite10(side, buff, secno, seccnt, trans_byte);
    :
}
```

11. サンプルアプリケーション

11.1 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. アプリケーションを起動すると、ターミナルソフト上に下記に示すメッセージが出力されます。
   SAMPLE> USB HMSC Application
   0 : READ loop mode
   1 : READ stop mode
2. MSC デバイスが接続されるとエニュメレーション処理を行います。
3. MSC デバイスのドライブ情報取得処理を行います。
4. 'a'が 512 個書き込まれたテキストファイル'SAMPLE?.txt’ を MSC デバイスに生成し、生成ファイルを一度読み出します。ファイル名の?は 0-7 のいずれかの値です。
5. ここでターミナルソフトから"0"を入力すると生成したファイルの内容を読み込み続けます。
   "1"を入力するとファイル読み込みを終了します。

[Note]

ファイル’ SAMPLE?.txt’が格納された MSC デバイスを接続した場合、’ SAMPLE?.txt’が上書きされます。

ターミナルソフトの設定は、「表 2.1　動作確認条件」を参照してください。

11.2 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。

11.2.1 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。

11.2.2 メインループ

USB ドライバは初期設定後アプリケーションのメインルーチン内でスケジューラ（R_USBH0_CstdScheduler()）を呼び出すことで動作します。

メインルーチン内で R_USBH0_CstdScheduler()を呼ぶことでイベントの有無を確認し、イベントがある場合、スケジューラにイベントが発生していることを通知するためのフラグをセットします。

R_USBH0_CstdScheduler()呼び出し後、R_USBH0_CstdCheckSchedule()を呼び出しイベントの有無を確認してください。また、イベントの取得とそのイベントに対する処理は定期的に行う必要があります。（注 1）

FreeRTOS では、Non OS メインループで呼び出している 4 つの Non OS 関数を OS タスクとして使用するため、BSP_CFG_RTOS_USED = 1 として、R_USBH0_Init()を呼び出してください。
そのため、BSP_CFG_RTOS_USED = 1 のときは、メインループで 4 つの Non OS 関数の呼び出しは不要です。
スケジューラ R_USBH0_CstdScheduler()は、Non OS、OS に関わらず各タスク関数にメッセージ送信し、タスク処理を制御します。

```
  void usb_main(void)
  {
    while(1)  // Main routine
    {
      // Confirming the event and getting (Note 1)
      R_USBH0_CstdScheduler();
#if(BSP_CFG_RTOS_USED == 0)
      // Judgment whether the event is or not
      if(USBH0_FLGSET == R_USBH0_CstdCheckSchedule())
      {
          R_USBH0_HstdMgrTask();          // MGR task
          R_USBH0_HhubTask();             // HUB task (Note 3)
          R_USBH0_HmscTask();             // MSC task
          R_USBH0_HmscStrgDriveTask();    // STRG driver task
      }
#endif /* (BSP_CFG_RTOS_USED == 0) */
      usbh0_msc_main();    // User application program
    }
  }
```

(Note 2)

（注1） R_USBH0_CstdScheduler()でイベントを取得後、処理を行う前に再度 R_USBH0_CstdScheduler()で他のイベントを取得すると、最初のイベントは破棄されます。イベント取得後は必ず各タスクを呼び出し、処理を行ってください。
（注2） OSLess では、アプリケーションプログラムのメインループ内に必ず記述してください。
（注3） HUB を使用する場合のみ、本関数を呼び出す必要があります。

11.2.3 APL

APL は、ステート遷移により管理を行っています。

表 11.1 にステート一覧を示します。

表 11.1 ステート一覧

ステート	概要
STATE_WAIT	接続待ち
STATE_DRIVE	ドライブ認識中
STATE_READY	ドライブ接続中
STATE_READ	ファイルリード
STATE_WRITE	ファイルライト
STATE_COMPLETE	処理完了
STATE_ERROR	エラー発生

図 11.1 に APL の処理フローチャートを示します。

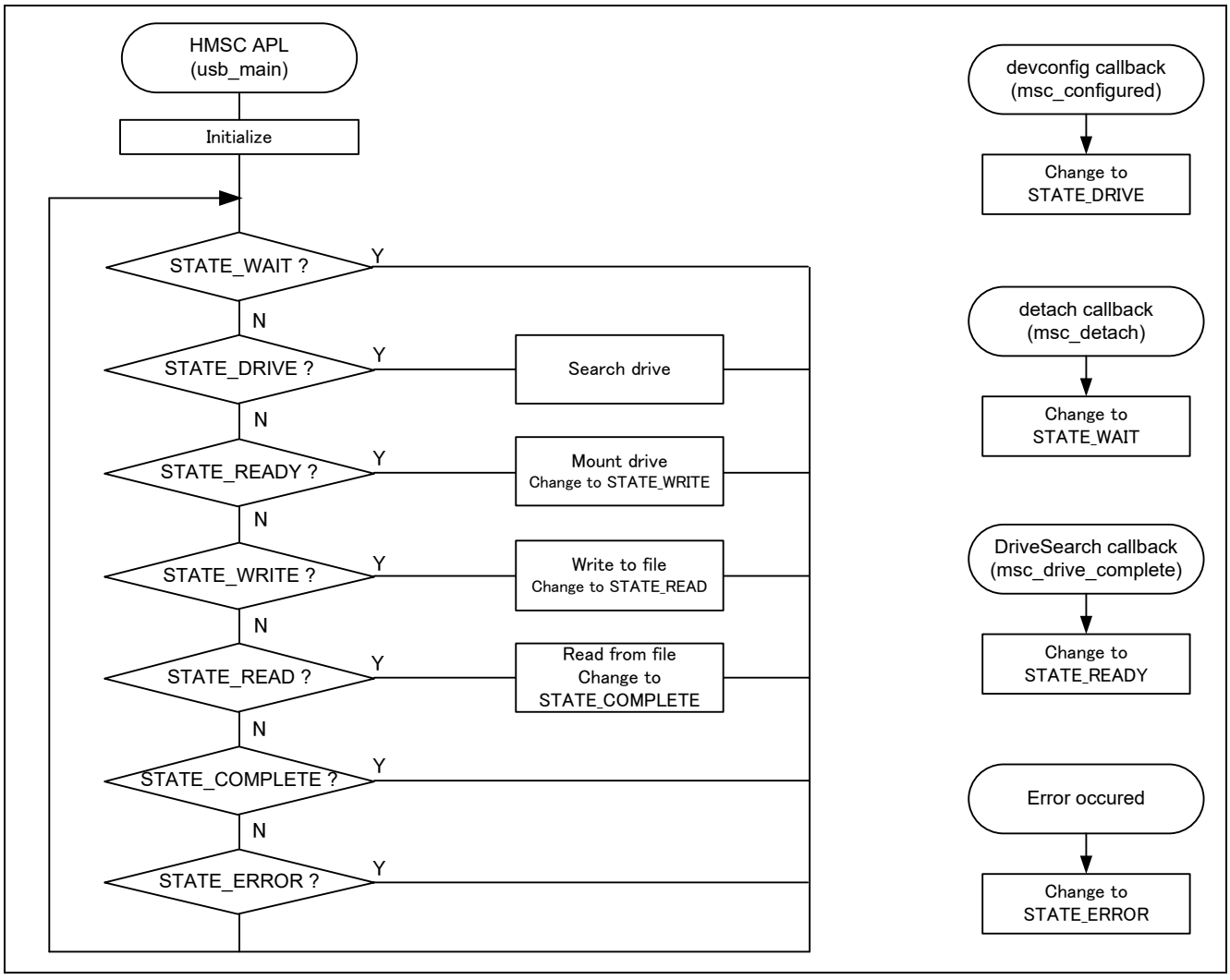

図 11.1 メインループ処理フローチャート

11.2.4 ステートの管理

以下にステートごとの処理概要を示します。

1) 接続待ち（STATE_WAIT）

== 概略 ==

このステートでは、MSC デバイスが接続されるのを待ちます。エニュメレーションが完了すると、ステートを STATE_DRIVE に変更します。

== 内容 ==

1. 初期化関数がステートを STATE_WAIT にします。
2. MSC デバイスが接続されるまで STATE_WAIT 状態を継続します。
3. MSC デバイスが接続されてエニュメレーションが完了すると、usbh0_cb_t 構造体のメンバ devconfig に設定されたコールバック関数 usbh0_msc_configured()が USB ドライバよりコールされます。
4. ステートを STATE_DRIVE に変更します。

2) ドライブ情報取得（STATE_DRIVE）

== 概要 ==

このステートでは、接続された MSC デバイスのドライブ情報取得処理をして、ステートを STATE_READY に変更します。

== 内容 ==

1. ドライブ認識中フラグ変数（usbh0_drive_search_lock）を確認して、オフであれば処理を開始します。
2. usbh0_drive_search_lock をオンにします。
3. R_USBH0_HmscStrgDriveSearch ()をコールして、MSC デバイスにクラスリクエスト GetMaxLUN とストレージコマンドを送信し、ドライブ情報取得処理を行います。
4. ドライブ情報取得処理が完了すると、R_USBH0_HmscStrgDriveSearch ()に登録したコールバック関数 usbh0_msc_drive_complete ()がコールされます。
5. ステートを STATE_READY に変更します。

3) ドライブ接続 （STATE_READY）

== 概要 ==

このステートでは、認識したドライブのマウント処理をして、ステートを STATE_WRITE に変更します。

== 内容 ==

1. 認識したドライブ番号から f_mount()をコールして接続します。
2. ステートを STATE_WRITE に変更します。

4) ファイルライト（STATE_WRITE）

== 概要 ==

このステートでは、接続したドライブにファイルライト処理をして、ステートを STATE_READ に変更します。

== 内容 ==

1. f_open()をコールして、ファイル作成＋書き込みモードでファイルオープンします。
2. f_write()をコールして、全て'a'である 512 バイトのファイル SAMPLE? .txt を作成します。ファイル名中の?はドライブ番号に対応します。例えば、ドライブ 1 の場合は SAMPLE1 .txt になります。

3. f_close()をコールして、ファイルクローズします。
4. ステートを STATE_READ に変更します。

5) ファイルリード（STATE_READ）

== 概要 ==

このステートでは、接続したドライブにファイルリード処理をして、ステートを STATE_COMPLETE に変更します。

== 内容 ==

1. f_open()をコールして、読み出しモードでファイルオープンします。
2. f_read()をコールして、ファイル SAMPLE? .txt を読み出します。
3. 全て'a'の 512 バイトのデータか確認します。
4. f_close()をコールして、ファイルクローズします。
5. ステートを STATE_COMPLETE に変更します。

6) 処理完了（STATE_COMPLETE）

== 概要 ==

サンプルアプリケーションの処理が正常終了したとき、このステートになります。

== 内容 ==

処理はありません。

7) エラー終了（STATE_ERROR）

== 概要 ==

サンプルアプリケーションの処理が異常終了したとき、このステートになります。

== 内容 ==

処理はありません。

8) デタッチ処理

接続された MSC デバイスが切断されると USB ドライバよりコールバック関数 usbh0_msc_detach ()がコールされます。このコールバック関数では、変数の初期化、ドライブ接続状態の解除およびステートを STATE_WAIT に変更する処理を行います。なお、コールバック関数 usbh0_msc_detach ()は、usbh0_cb_t 構造体のメンバ devdetach に設定した関数です。

12. 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。）

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2019 年 4 月 15 日	-	初版発行
1.10	2019 年 5 月 17 日	5	表 2.1　動作確認条件 コンパイラオプション"-mthumb-interwork"を削除
1.20	2019 年 12 月 17 日	5	表 2.1　動作確認条件 コンパイラオプション"-g3"を"-None"に変更 FreeRTOS/OSLess 双方のサポート

製品ご使用上の注意事項

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

1. 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。