RZ/T1 グループ

R01AN2634JJ0130
Rev.1.30
Dec 07, 2017

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について説明します。本モジュールは USB Basic Firmware(USB-BASIC-F/W)と組み合わせることで動作します。以降、本モジュールを HMSC と称します。
本アプリケーションノートのサンプルプログラムは「RZ/T1 グループ初期設定 Rev.1.30」をベースに作成しています。
動作環境については「RZ/T1 グループ初期設定アプリケーションノート(R01AN2554JJ0130)」を参照してください。

対象デバイス

RZ/T1 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/T1 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. R01UH0483JJ0130)
5. RZ/T1 グループ初期設定 (ドキュメント No. R01AN2554JJ0130)
6. USB Host Basic Firmware (ドキュメント No. R01AN2633JJ0130)

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/】
FatFs ホームページ
【http://elm-chan.org/fsw/ff/00index_e.html】

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として動作します。

HMSC は、USB マスストレージクラスの BOT プロトコルで構築されています。ファイルシステム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）
- BOT プロトコルによるストレージコマンド通信
- USB マスストレージサブクラスの SFF-8070i(ATAPI)と SCSI に対応
- 複数の USB ストレージ機器を接続可能

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。

用語一覧

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/T1 グループ
USB	:	Universal Serial Bus

2. ソフトウェア構成

Figure 2-1 に HMSC のソフトウェア構成、Table 2-1 にモジュール機能概要を示します。

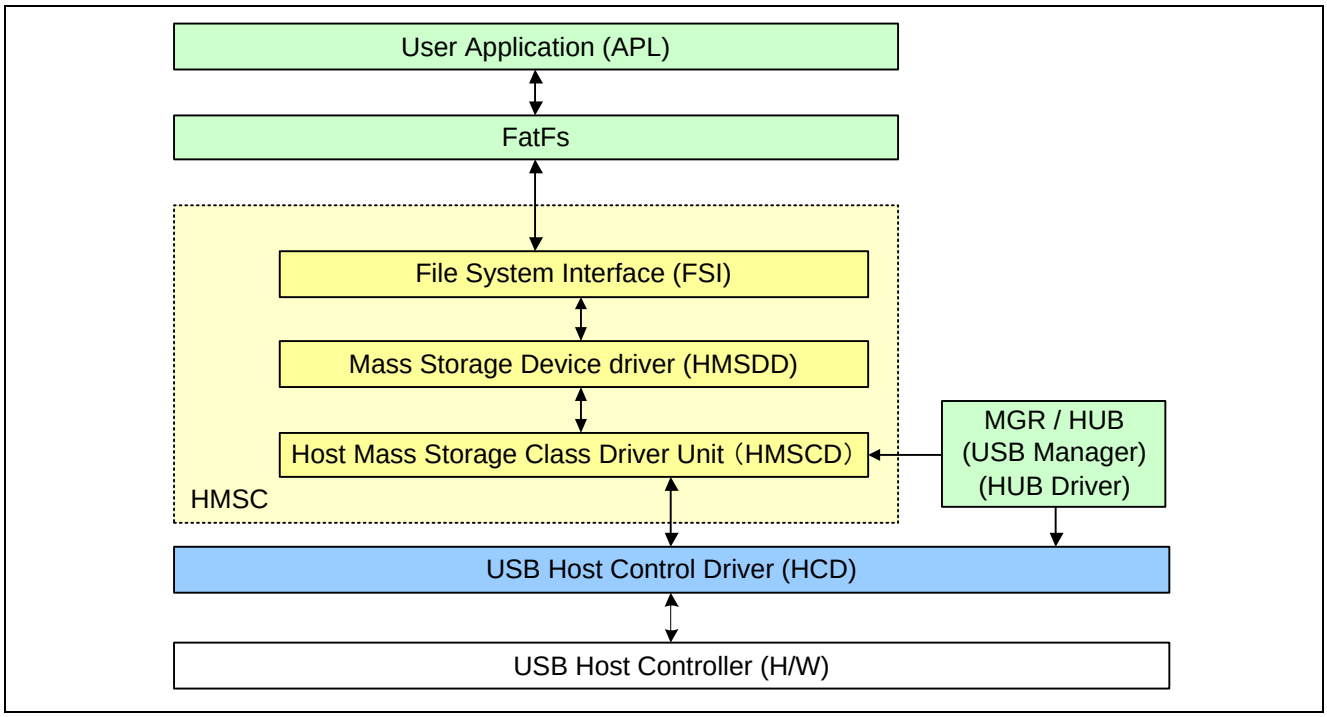


Figure 2-1 ソフトウェア構成図

Table 2-1 モジュール機能概要

モジュール名	説明
APL	ユーザアプリケーションプログラム（システムに合わせてご用意ください）
FatFs	汎用 FAT ファイルシステム（5. FatFs 組み込み手順を参照してください）
HMSC	ホストマスストレージクラスドライバ FSI と HMSDD と HMSCD で構成されています
FSI	ファイルシステムインタフェース（FatFs に対応したサンプルを提供しています）
HMSDD	ホストマスストレージデバイスドライバ ・ USB ストレージ機器情報取得 ・ USB ストレージ機器アクセス
HMSCD	ホストマスストレージクラスドライバユニット ・ MSC デバイスの照合 ・ BOT プロトコルで HCD 通信要求 ・ BOT シーケンスの管理
MGR / HUB	USB マネージャ / HUB クラスドライバ（USB-BASIC-F/W） ・ 接続されたデバイスとエニュメレーションをして HMSC を起動 ・ デバイスの状態管理
HCD	USB Host H/W 制御ドライバ（USB-BASIC-F/W）

3. ホストマスストレージクラスドライバ (HMSC)

HMSC は、FSI と HMSDD と HMSCD で構成されており、USB-BASIC-F/W と結合しています。
本章では、HMSC の機能について示します。

3.1 クラスリクエスト

Table 3-1 に HMSC がサポートするクラスリクエストを示します。

Table 3-1 クラスリクエスト

リクエスト	コード	説明	対応
Mass Storage Reset	0xFF	プロトコルエラーを解除	○
GetMaxLUN	0xFE	デバイスがサポートする最大ユニット数を取得	○

○：実装　×：未実装

3.2 ストレージコマンド

Table 3-2 に HMSC がサポートするストレージコマンドを示します。

Table 3-2 ストレージコマンド

コマンド名	コード	説明	対応
TEST_UNIT_READY	0x00	ペリフェラル機器の状態確認	○
REQUEST_SENSE	0x03	ペリフェラル機器の状態取得	○
FORMAT_UNIT	0x04	論理ユニットのフォーマット	×
INQUIRY	0x12	論理ユニットのパラメータ情報取得	○
MODE_SELECT6	0x15	パラメータ指定	○
MODE_SENSE6	0x1A	論理ユニットのパラメータ取得	×
START_STOP_UNIT	0x1B	論理ユニットのアクセス許可／禁止	×
PREVENT_ALLOW	0x1E	メディアの取り出し許可／禁止	○
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	○
SEEK	0x2B	論理ブロックアドレスに移動	×
WRITE_AND_VERIFY	0x2E	確認付きデータ書き込み	×
VERIFY10	0x2F	データ確認	×
MODE_SELECT10	0x55	パラメータ指定	×
MODE_SENSE10	0x5A	論理ユニットのパラメータ取得	○

○：実装　×：未実装

3.3 USB ストレージ機器の照合

USB-BASIC-F/W は、エニュメレーション時に GET_DESCRIPTOR リクエストで取得した情報を HMSC にコールバック関数で通知します。HMSCD は、このコールバック関数として登録するための API 関数 R_usb_hmsc_ClassCheck()を用意しています。R_usb_hmsc_ClassCheck()は、ディスクリプタ情報を解析して、照合結果を USB-BASIC-F/W の API 関数 R_usb_hstd_ReturnEnuMGR()で通知します。結果に問題がなければ、USB-BASIC-F/W はエニュメレーションを完了します。

3.4 USB ストレージ機器の情報取得

エニュメレーション完了後は、HMSDD の API 関数 R_usb_hmsc_StrgDriveSearch()をコールすることで USB ストレージ機器の情報取得処理をすることができます。この処理の完了は、R_usb_hmsc_StrgDriveSearch()コール時に登録するコールバック関数で通知されます。

GetMaxLUN リクエストの応答結果に関わらず、ユニット数 0 として動作します。よって、LUN0 のみのサポートとなります。

Figure 3-1 に USB ストレージ機器の情報取得シーケンスを示します。

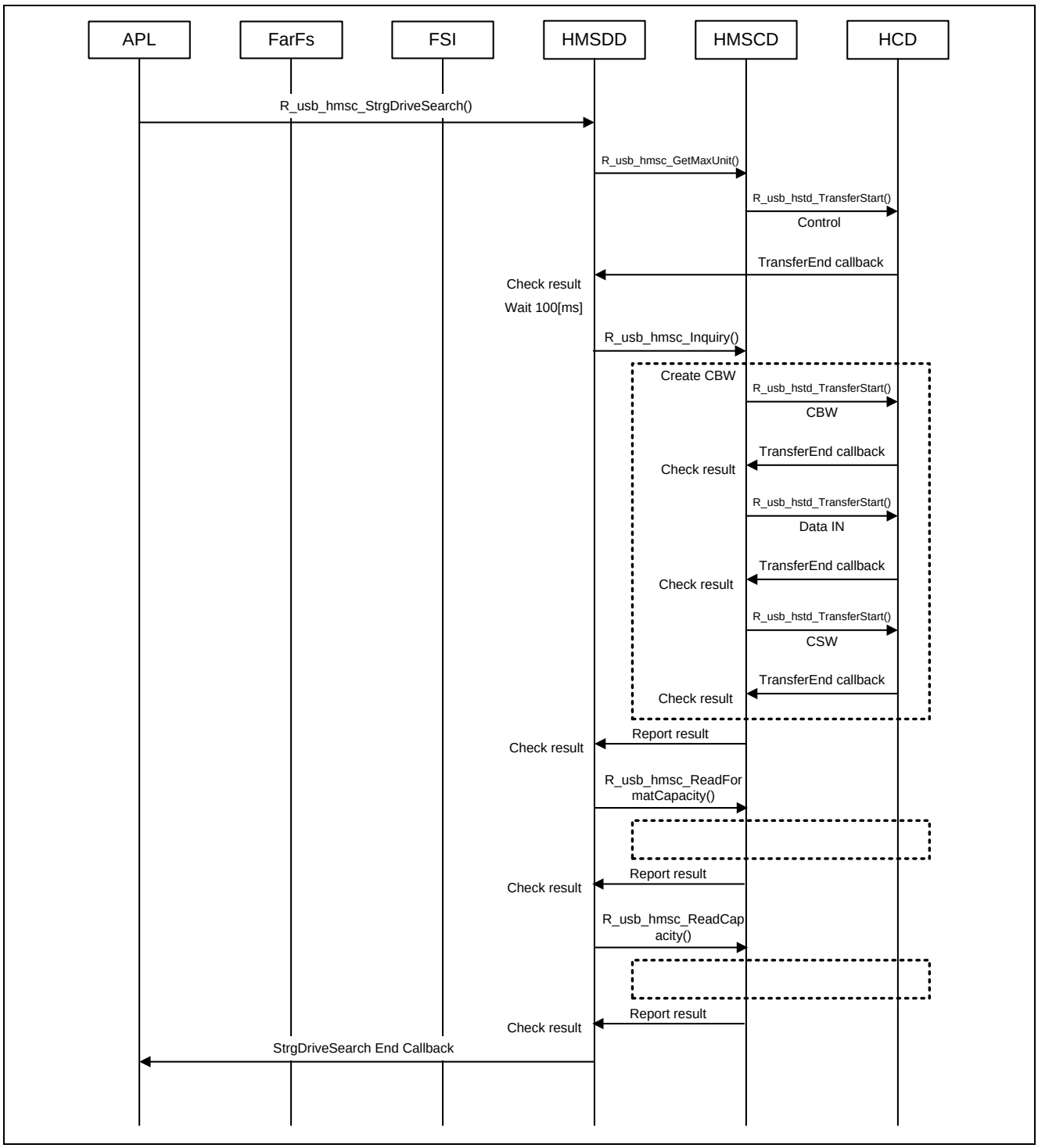


Figure 3-1 USB ストレージ機器の情報取得シーケンス

3.5 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。

Figure 3-2 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。

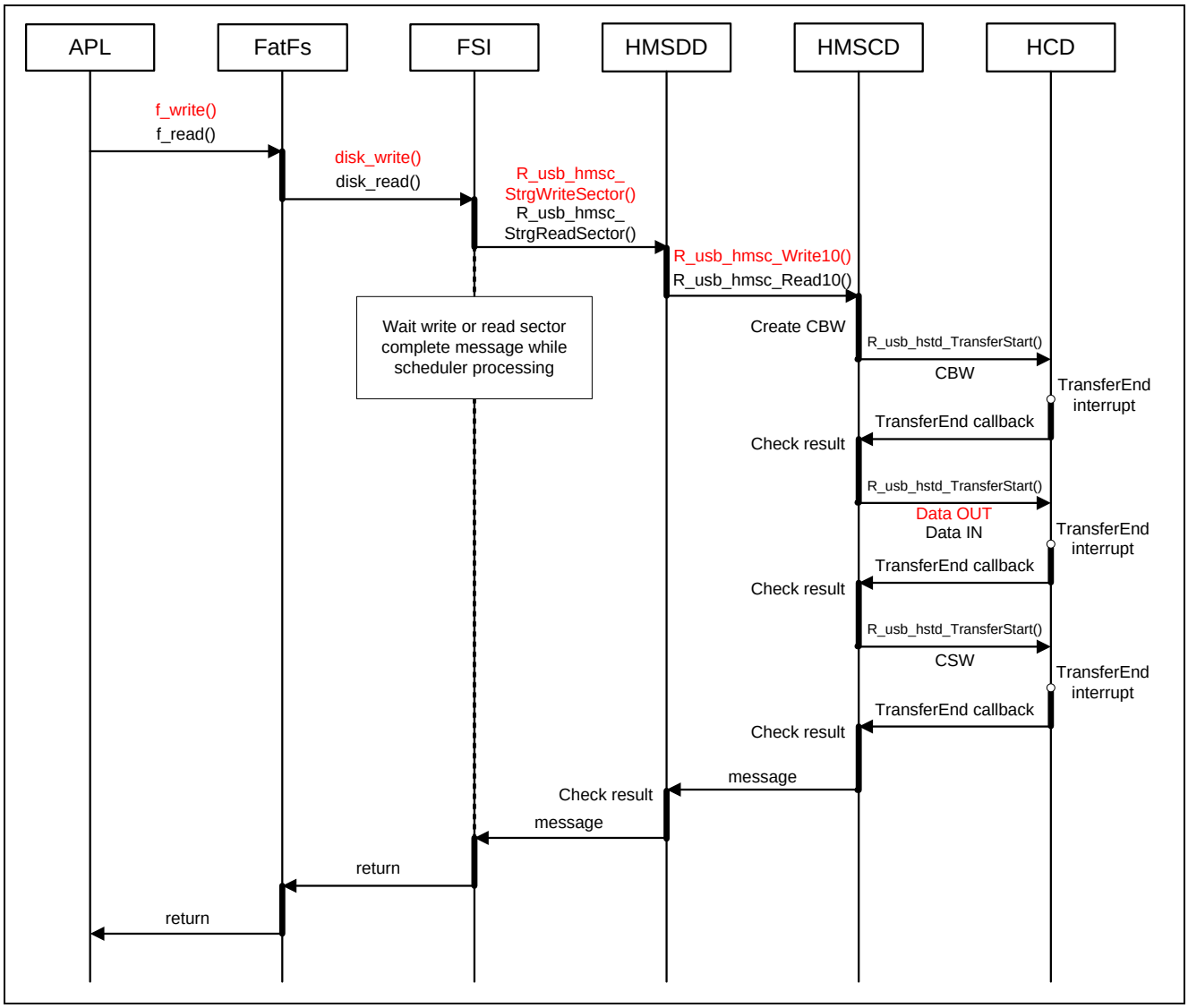


Figure 3-2 USB ストレージ機器へのアクセスシーケンス

3.6 HMSCD API 関数

HMSCD の API 関数は主に HMSDD 向けに用意しています。
通常、アプリケーションで使用する関数は、R_usb_hmsc_Task()、R_usb_hmsc_driver_start()、R_usb_hmsc_Class_Check()の 3 つです。

Table 3-3 に HMSCD の API 関数を示します。

Table 3-3 HMSCD 関数

関数名	説明
R_usb_hmsc_Task()	HMSCD タスク
R_usb_hmsc_driver_start()	HMSC ドライバ起動
R_usb_hmsc_ClassCheck()	ディスクリプタチェック処理
R_usb_hmsc_GetDevSts()	HMSCD 動作状態の応答
R_usb_hmsc_Read10()	READ10 コマンド発行
R_usb_hmsc_Write10()	WRITE10 コマンド発行
R_usb_hmsc_GetMaxUnit()	GetMaxLUN リクエスト発行
R_usb_hmsc_MassStorageReset()	MassStorageReset リクエスト発行
R_usb_hmsc_alloc_drvno()	ドライブ番号割り当て
R_usb_hmsc_free_drvno()	ドライブ番号解放
R_usb_hmsc_ref_drvno()	ドライブ番号参照

3.6.1 R_usb_hmsc_Task

HMSCD タスク

形式

void	R_usb_hmsc_Task(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

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())
  {
    R_usb_hstd_MgrTask();
    R_usb_hhub_Task();
    R_usb_hmsc_Task();
  }
}
```

3.6.2 R_usb_hmsc_driver_start

HMSC ドライバ起動

形式

void R_usb_hmsc_driver_start(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。

補足

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_usb_hmsc_driver_start();    /* Host Class Driver Task Start Setting */
}
```

3.6.3 R_usb_hmsc_ClassCheck

ディスクリプタチェック処理

形式

void	R_usb_hmsc_ClassCheck(uint16_t **table)

引数

**table	デバイス情報テーブル [0]：デバイスディスクリプタ [1]：コンフィグレーションディスクリプタ [2]：インタフェースディスクリプタ [3]：ディスクリプタチェック結果 [4]：HUB 種別 [5]：ポート番号 [6]：通信速度 [7]：デバイスアドレス

戻り値

―	―

解説

本 API はクラスドライバレジストレーション関数です。この関数は、スタートアップ時の HMSC 登録時にドライバレジストレーション構造体メンバ classcheck にコールバック関数として登録され、エニュメレーション動作のコンフィグレーションディスクリプタ受信時に呼出されます。
ペリフェラルデバイスのコンフィグレーションディスクリプタからエンドポイントディスクリプタを参照し、パイプ情報テーブルの登録をします。

補足

―

使用例

```
void  usb_hapl_registration(USB_UTR_t *ptr)
{
  USB_HCDREG_t driver;

  /* Driver check */
  driver.classcheck   = &R_usb_hmsc_ClassCheck;
}
```

3.6.4 R_usb_hmsc_GetDevSts

HMSCD 動作状態の応答

形式

uint16_t R_usb_hmsc_GetDevSts(uint16_t side)

引数

side 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

引数 side には、R_usb_hmsc_alloc_drvno()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_hmsc_GetDevSts(drvno) == USB_HMSC_DEV_DET)
  {
    /* 切断処理 */
  }
}
```

3.6.5 R_usb_hmsc_Read10

READ10 コマンド発行

形式

uint16_t	R_usb_hmsc_Read10(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

—	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

—

使用例

```
void usb_smp_task(void)
{
  uint16_t   result;

  /* READ10 発行 */
  result = R_usb_hmsc_Read10(side, buff, secno, seccnt, trans_byte);
  if(result != USB_HMSC_OK)
  {
    /* エラー処理 */
  }
}
```

3.6.6 R_usb_hmsc_Write10

WRITE10 コマンド発行

形式

uint16_t	R_usb_hmsc_Write10(uint16_t side, const 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

－

使用例

```
void  usb_smp_task(void)
{
  uint16_t  result;

  /* WRITE10 発行 */
  result = R_usb_hmsc_Write10(side, buff, secno, seccnt, trans_byte);
  if(result != USB_HMSC_OK)
  {
    /* エラー処理 */
  }
}
```

3.6.7 R_usb_hmsc_GetMaxUnit

GetMaxLUN リクエスト発行

形式

USB_ER_t R_usb_hmsc_GetMaxUnit(uint16_t addr, USB_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

USB_OK	GET_MAX_LUN 発行
USB_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 comlete に指定したコールバック関数がコールされます。

補足

―

使用例

```
void usb_smp_task(void)
{
  USB_ER_t   err;

  /* 最大ユニット数取得*/
  err = R_usb_hmsc_GetMaxUnit(devadr, usb_hmsc_StrgCheckResult);
  if(err == USB_ERROR)
  {
    /* エラー処理 */
  }
}
```

3.6.8 R_usb_hmsc_MassStorageReset

MassStorageReset リクエスト発行

形式

USB_ER_t R_usb_hmsc_MassStorageReset(uint16_t addr, USB_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

USB_OK	MASS_STORAGE_RESET 発行
USB_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 comlete に指定したコールバック関数がコールされます。

補足

―

使用例

```
void  usb_smp_task(void)
{
  USB_ER_t  err;

  /* プロトコルエラー解除*/
  err = R_usb_hmsc_MassStorageReset(devadr, usb_hmsc_CheckResult);
  if(err == USB_ERROR)
  {
    /* エラー処理 */
  }
}
```

3.6.9 R_usb_hmsc_alloc_drvno

ドライブ番号割り当て

形式

uint16_t R_usb_hmsc_alloc_drvno(uint16_t *side, uint16_t devadr)

引数

*side	ドライブ番号ポインタ
devadr	MSC デバイスのデバイスアドレス

戻り値

USB_OK	正常終了
USB_ERROR	エラー終了

解説

接続された MSC デバイスに対しドライブ番号を割り当て、引数*side に格納します。
ドライブ番号は 0 から順に割り当てられます。

補足

―

使用例

```
void usb_smp_task(void)
{
   /* ドライブ番号割り当て */
   R_usb_hmsc_alloc_drvno(&drvno, devadr);

}
```

3.6.10 R_usb_hmsc_free_drvno

ドライブ番号解放

形式

uint16_t R_usb_hmsc_free_drvno(uint16_t side)

引数

side	ドライブ番号

戻り値

USB_OK	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番号を解放します。

補足

―

使用例

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

3.6.11 R_usb_hmsc_ref_drvno

ドライブ番号参照

形式

uint16_t R_usb_hmsc_ref_drvno(uint16_t *side, uint16_t devadr)

引数

*side	ドライブ番号ポインタ
devadr	MSC デバイスのデバイスアドレス

戻り値

USB_OK	正常終了
USB_ERROR	エラー終了

解説

引数で指定されたデバイスアドレスのドライブ番号を参照して、引数*side に格納します。

補足

―

使用例

```
void usb_smp_task(void)
{
    /* ドライブ番号参照 */
    R_usb_hmsc_ref_drvno(&drvno, devadr);
}
```

3.7 HMSDD API 関数

Table 3-4 に HMSDD API の関数を示します。

Table 3-4 HMSDD 関数

関数名	説明
R_usb_hmsc_StrgDriveTask()	マスストレージドライバタスク
R_usb_hmsc_StrgDriveSearch()	ドライブ情報取得
R_usb_hmsc_StrgDriveOpen()	ドライブオープン
R_usb_hmsc_StrgDriveClose()	ドライブクローズ
R_usb_hmsc_StrgReadSector()	セクタ読み出し
R_usb_hmsc_StrgWriteSector()	セクタ書き込み
R_usb_hmsc_StrgCheckEnd()	読み出し／書き込み完了確認
R_usb_hmsc_StrgUserCommand()	ストレージコマンド発行

3.7.1 R_usb_hmsc_StrgDriveTask

マスストレージドライブタスク

形式

void	R_usb_hmsc_StrgDriveTask(void)

引数

―	―

戻り値

―	―

解説

ストレージコマンドを送信し、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

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_hapl_mainloop(void)
{
  while(1)
  {
    R_usb_cstd_Scheduler();

    if(USB_FLGSET == R_usb_cstd_CheckSchedule())
    {
       R_usb_hstd_MgrTask();
       R_usb_hhub_Task();
       R_usb_hmsc_Task();
       R_usb_hmsc_StrgDriveTask();
    }
  }
}
```

3.7.2 R_usb_hmsc_StrgDriveSearch

ドライブ情報取得

形式

uint16_t R_usb_hmsc_StrgDriveSearch(uint16_t addr, USB_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

USB_OK	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、情報取得処理をします。
処理が完了すると引数 complete に設定したコールバック関数がコールされます。
詳細は 3.4 項を参照してください。

補足

本 API をコールしてからコールバック関数がコールされるまでは、スケジューラを動作させてください。また、他の USB 関連処理をしないでください。

使用例

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

/* Callback function */
void usb_hmsc_StrgCommandResult( USB_UTR_t *mess )
{
        :
}
```

3.7.3 R_usb_hmsc_StrgDriveOpen

ドライブオープン

形式

uint16_t R_usb_hmsc_StrgDriveOpen(uint16_t *side, uint16_t addr)

引数

*side	ドライブ番号ポインタ
addr	デバイスアドレス

戻り値

USB_OK	正常終了
USB_ERROR	エラー終了

解説

引数で指定されたアドレスをオープンします。
エニュメレーション完了後にコールしてください。

補足

1. 関数内で R_usb_hmsc_alloc_drvno() を使用してドライブ番号を取得します。
2. 関数内で R_usb_hstd_GetPipeID() を使用してパイプ番号を設定します。

使用例

```
void msc_configured(uint16_t devadr)
{
    R_usb_hmsc_StrgDriveOpen(&drvno, devadr);
}
```

3.7.4 R_usb_hmsc_StrgDriveClose

ドライブクローズ

形式

uint16_t R_usb_hmsc_StrgDriveClose(uint16_t side)

引数

side	ドライブ番号

戻り値

USB_OK	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をクローズします。

補足

1. 関数内で R_usb_hmsc_free_drvno() を使用してドライブ番号を開放します。
2. 関数内で R_usb_hstd_ClearPipe() を使用してパイプ情報をクリアします。

使用例

```
void msc_detach(uint16_t devadr)
{
    R_usb_hmsc_StrgDriveClose(drv_no);
}
```

3.7.5 R_usb_hmsc_StrgReadSector

セクタ読み出し

形式

uint16_t	R_usb_hmsc_StrgReadSector(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

USB_OK	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 FSI で呼び出してください。
2. 関数内で R_usb_hmsc_Read10() を使用してセクタ情報を読み出します。
3. 関数内で R_usb_hmsc_GetDevSts() を使用して接続状況を確認しており、切断状態であれば、読み出し前にエラー終了します。

使用例

```
DRESULT disk_read(BYTE pdrv, BYTE* buff, DWORD sector, UINT count)
{
    uint32_t       err;
    uint32_t       tran_byte;

    /* set transfer length */
    tran_byte = count * _MIN_SS;

    /* read function */
    err = R_usb_hmsc_StrgReadSector(pdrv, buff, sector, (uint16_t)count,
tran_byte);
    if (err != USB_OK)
    {
        return RES_ERROR;
    }
}
```

3.7.6 R_usb_hmsc_StrgWriteSector

セクタ書き込み

形式

uint16_t	R_usb_hmsc_StrgWriteSector(uint16_t side, const 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

USB_OK	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、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 FSI で呼び出してください。
2. 関数内で R_usb_hmsc_Write10() を使用してセクタ情報を書き込みます。
3. 関数内で R_usb_hmsc_GetDevSts() を使用して接続状況を確認しており、切断状態であれば、書き込み前にエラー終了します。

使用例

```
DRESULT disk_write(BYTE pdrv, const BYTE* buff, DWORD sector, UINT count)
{
    uint32_t       err;
    uint32_t       tran_byte;

    /* set transfer length */
    tran_byte = count * _MIN_SS;

    /* write function */
    err = R_usb_hmsc_StrgWriteSector(pdrv, buff, sector, (uint16_t)count,
tran_byte);
    if (err != USB_OK)
    {
        return RES_ERROR;
    }
}
```

3.7.7 R_usb_hmsc_StrgCheckEnd

読み出し／書き込み完了確認

形式

uint16_t R_usb_hmsc_StrgCheckEnd(void)

引数

―

戻り値

USB_FALSE	読み出し／書き込み中
USB_TRUE	読み出し／書き込み完了
USB_ERROR	読み出し／書き込みエラー

解説

読み出し／書き込みシーケンスの状態を返却します。

補足

本 API は、R_usb_hmsc_StrgReadSector()または R_usb_hmsc_StrgWriteSector()をコールした後に定期的に呼び出してください。

使用例

```
DRESULT disk_write(BYTE pdrv, const BYTE* buff, DWORD sector, UINT count)
{
    uint32_t       err;

    /* write function */
    R_usb_hmsc_StrgWriteSector(pdrv, buff, sector, (uint16_t)count, tran_byte);

    /* Wait USB write sequence(WRITE10) */
    do
    {
        R_usb_cstd_Scheduler();
        if (R_usb_cstd_CheckSchedule() == USB_FLGSET)
        {
            R_usb_hstd_MgrTask();      /* MGR task */
            R_usb_hhub_Task();         /* HUB task */
            R_usb_hmsc_task();         /* HMSC Task */
            R_usb_hmsc_StrgDriveTask(); /* HSTRG Task */
        }
        err = R_usb_hmsc_StrgCheckEnd();
    }
    while (err == USB_FALSE);

    /* Set transfer result */
    if (err != USB_TRUE)
    {
        return RES_ERROR;
    }
    else
    {
        return RES_OK;
    }
}
```

3.7.8 R_usb_hmsc_StrgUserCommand

ストレージコマンド発行

形式

uint16_t R_usb_hmsc_StrgUserCommand(uint16_t side, uint16_t command , uint8_t *buff, USB_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

USB_OK	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、本 API 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. 関数内で HMSCD の API 関数を使用してストレージコマンドを発行します。
2. 関数内で R_usb_hmsc_GetDevSts() を使用して接続状況を確認しており、切断状態であれば、ストレージコマンド発行前にエラー終了します。

使用例

```
/* Callback function */
void  strgcommand_complete(USB_UTR_t *mess)
{
          :
}

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

3.8 FSI 関数

FatFs は、単なるファイルシステムレイヤなので、ストレージデバイス制御レイヤは含まれません。

FatFs は、下位レイヤに対しインタフェースを要求しており、使用するプラットフォームやストレージデバイスに対応した制御関数を提供する必要があります。

HMSC は、この制御関数のサンプル（FSI 関数）を用意しています。FatFs の仕様を確認の上、必要があればシステムに合わせて変更してください。

Table 3-5 に FSI 関数一覧を示します。

Table 3-5 FSI 関数一覧

関数名	説明
disk_status	デバイスの状態取得
disk_initialize	デバイスの初期化
disk_read	データの読み出し
disk_write	データの書き込み
disk_ioctl	その他のデバイス制御
get_fattime	日付と時刻の取得

3.8.1 disk_status

デバイスの状態取得

形式

DSTATUS disk_status(BYTE pdrv)

引数

pdrv [IN] 物理ドライブ番号(0-9)

戻り値

現在のストレージデバイスの状態を次のフラグの組み合わせ値で返します。
STA_NOINIT デバイスが初期化されていないことを示すフラグ
STA_NODISK メディアが存在しないことを示すフラグ
STA_PROTECT メディアがライトプロテクトされていることを示すフラグ (サンプルでは未使用)

解説

サンプルアプリケーションの変数 R_usb_disk_status[pdrv] の値を返却します。
pdrv に 10 以上が設定された場合は、STA_NODISK | STA_NOINIT を返却します。

補足

なし

3.8.2 disk_initialize

デバイスの初期化

形式

DSTATUS disk_initialize(BYTE pdrv)

引数

pdrv [IN] 物理ドライブ番号(0-9)

戻り値

現在のストレージデバイスの状態を次のフラグの組み合わせ値で返します。
STA_NOINIT デバイスが初期化されていないことを示すフラグ
STA_NODISK メディアが存在しないことを示すフラグ
STA_PROTECT メディアがライトプロテクトされていることを示すフラグ (サンプルでは未使用)

解説

R_usb_disk_status[pdrv] の値を返却します。

補足

この関数は FatFs の管理下にあり、自動マウント動作により必要に応じて呼び出されます。
アプリケーションからの呼び出しは禁止です。
再初期化が必要なときは、FatFs の API 関数（f_mount()）を使用してください。

3.8.3 disk_read

データの読み出し

形式

DRESULT disk_read(BYTE pdrv, BYTE* buff, DWORD sector, UINT count)

引数

pdrv	[IN] 物理ドライブ番号(0-9)
*buff	[OUT] 読み出しバッファへのポインタ
sector	[IN] 読み出し開始セクタ番号
count	[IN] 読み出すセクタ数(1-128)

戻り値

RES_OK	正常終了
RES_ERROR	読み出し中にエラーが発生
RES_PARERR	コマンドが不正 (サンプルでは未使用)
RES_NOTRDY	ストレージデバイスが動作可能状態ではない (サンプルでは未使用)

解説

HMSDD の API 関数 R_usb_hmsc_StrgReadSector() をコールします。
R_usb_hmsc_StrgReadSector()の引数設定は以下の通りです。

引数	設定値
uint16_t side	pdrv
uint8_t *buff	buff
uint32_t secno	sector
uint16_t seccnt	(uint16_t)count
uint32_t trans_byte	count * _MIN_SS

USB 読み出しシーケンスが完了するまで、本関数内でループしてスケジューラを動作します。
途中で USB 切断を検出した場合は、RES_ERROR を返却します。

補足

なし

3.8.4 disk_write

データの書き込み

形式

DRESULT disk_write(BYTE pdrv, const BYTE* buff, DWORD sector, UINT count)

引数

pdrv	[IN] 物理ドライブ番号(0-9)
*buff	[IN] 書き込むデータへのポインタ
sector	[IN] 書き込み開始セクタ番号
count	[IN] 書き込むセクタ数(1-128)

戻り値

RES_OK	正常終了
RES_ERROR	書き込み中にエラーが発生
RES_PARERR	コマンドが不正 (サンプルでは未使用)
RES_NOTRDY	ストレージデバイスが動作可能状態ではない (サンプルでは未使用)

解説

HMSDD の API 関数 R_usb_hmsc_StrgWriteSector()をコールします。
R_usb_hmsc_StrgWriteSector()の引数設定は以下の通りです。

引数	設定値
uint16_t side	pdrv
const uint8_t *buff	buff
uint32_t secno	sector
uint16_t seccnt	(uint16_t)count
uint32_t trans_byte	count * _MIN_SS

USB 書き込みシーケンスが完了するまで、本関数内でループしてスケジューラを動作します。
途中で USB 切断を検出した場合は、RES_ERROR を返却します。

補足

なし

3.8.5 disk_ioctl

その他のデバイス制御

形式

DRESULT disk_ioctl(BYTE pdrv, BYTE cmd, void* buff)

引数

pdrv	[IN] 物理ドライブ番号(0-9)
sector	[IN] 制御コマンド
*buff	[I/O] データ受け渡しバッファ

戻り値

RES_OK	正常終了
RES_ERROR	エラーが発生 (サンプルでは未使用)
RES_PARERR	コマンドが不正 (サンプルでは未使用)
RES_NOTRDY	ストレージデバイスが動作可能状態ではない (サンプルでは未使用)

解説

全てのコマンドに対して処理をせず、RES_OK を返却します。

補足

なし

3.8.6 get_fattime

日付と時刻の取得

形式

DWORD get_fattime(void)

引数

― ―

戻り値

現在のローカルタイムを DWORD 値にパックして返します。ビットフィールドは下記の通りです。
bit 31:25 1980 年を起点とした年を 0..127 でセット
bit 24:21 月を 1..12 の値でセット
bit 20:16 日を 1..31 の値でセット
bit 15:11 時を 0..23 の値でセット
bit 10:5 分を 0..59 の値でセット
bit 4:0 秒/2 を 0..29 の値でセット

解説

日付と時刻情報は設定せず、0x00000000 を返却します。

補足

なし

3.9 スケジューラ設定

Table 3-6 に、HMSC のスケジューラ設定を示します。

Table 3-6 スケジューラ設定

関数名	タスク ID	優先度	メールボックス名	メモリプール名	概要
R_usb_hmsc_StrgDriveTask	USB_HSTRG_TSK	USB_PRI_3	USB_HSTRG_MBX	USB_HSTRG_MPL	HSTRG タスク
R_usb_hmsc_task	USB_HMSC_TSK	USB_PRI_3	USB_HMSC_MBX	USB_HMSC_MPL	HMSCD タスク
R_usb_hub_task	USB_HUB_TSK	USB_PRI_3	USB_HUB_MBX	USB_HUB_MPL	HUB タスク
R_usb_hstd_MgrTask	USB_MGR_TSK	USB_PRI_2	USB_MGR_MBX	USB_MGR_MPL	MGR タスク
r_usb_hstd_HciTask	USB_HCI_TSK	USB_PRI_1	USB_HCI_MBX	USB_HCI_MPL	HCD タスク

4. サンプルアプリケーション

本章では、HMSC と USB-BASIC-F/W を組み合わせ、USB ドライバとして使用するために必要な初期設定の方法と、メインルーチン処理方法を示します。

4.1 動作環境

Figure 4-1 に HMSC の動作環境例を示します。

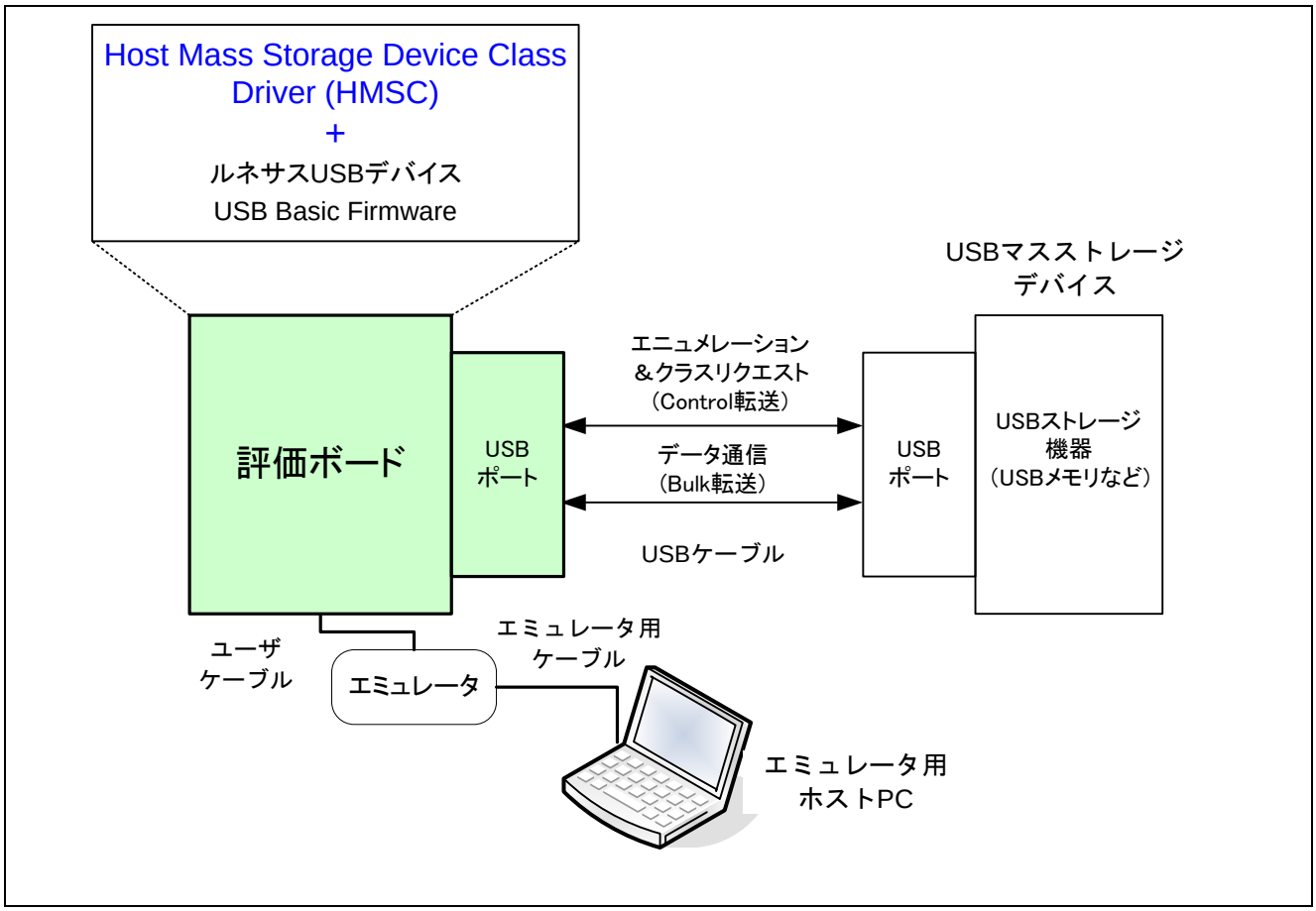


Figure 4-1 動作環境例

4.2 仕様概要

HMSC のサンプルアプリケーション主な機能を以下に示します。

1. MSC デバイスが接続されるとエニュメレーション処理をします。
2. MSC デバイスのエニュメレーションが完了するとドライブ情報取得処理をします。
3. MSC デバイスに 512 バイトのサイズのファイルを書き込みます。
4. MSC デバイスに書き込んだファイルの読み込みをします。
5. ファイル内容を比較して一致した場合に、ドライブ番号に対応した LED を点灯します。

4.3 初期設定

初期設定例を以下に示します。

```
void usb_hmsc_apl(void)
{
    /* MCU の設定「4.3.1」参照*/
    usb_mcu_setting();

    /* USB ドライバの設定「4.3.2」参照 */
    R_usb_hstd_MgrOpen();
    R_usb_cstd_SetTaskPri(USB_HUB_TSK,  USB_PRI_3);  // (注)
    R_usb_hhub_Registration(USB_NULL);               // (注)
    msc_registration();
    R_usb_hmsc_driver_start();

    /* メインルーチン「4.4」参照 */
    usb_hapl_mainloop();
}
```

(注) 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。

4.3.1 MCU 設定

USB モジュールをハードウェアマニュアルの初期設定シーケンスに従って設定し、USB 割り込みハンドラの登録と USB 割り込み許可設定をしています。

4.3.2 USB ドライバ設定

USB ドライバの設定では、スケジューラへのタスク登録及び USB-BASIC-F/W に対するクラスドライバの情報登録を行います。以下に、手順を示します。

1. USB-BASIC-F/W の API 関数（R_usb_hstd_MgrOpen()）を呼び出し、HCD タスクと MGR タスクを登録する。
2. HUB クラスドライバ API 関数（R_usb_hhub_Registration()）を呼び出し、HUB タスクを登録する。
3. クラスドライバ登録用構造体（USB_HCDREG_t）の各メンバに情報を設定後、USB-BASIC-F/W の API 関数（R_usb_hstd_DriverRegistration()を呼び出し、クラスドライバを登録する。
4. HMSC クラスドライバの API 関数（R_usb_hmsc_driver_start()）を呼び出し、HMSC タスクと HSTRG タスクを登録する。

USB_HCDREG_t で宣言された構造体に設定する情報例を以下に示します。

```
void msc_registration(void)
{
    /* クラスドライバ登録用構造体 */
    USB_HCDREG_t      driver;

    /* USB の規格で定められたクラスコードを設定 */
    driver.ifclass    = (uint16_t)USB_IFCLS_MSC;
    /* ターゲットペリフェラルリストを設定 */
    driver.tpl        = (uint16_t*)&usb_gapl_devicetpl; (注 1)
    /* エニュメレーション中に行われるクラスチェック関数を設定 */
    driver.classcheck = &R_usb_hmsc_class_check;
    /* エニュメレーション完了時に呼び出される関数を設定 */
    driver.devconfig  = &msc_configured;
    /* USB デバイス切断時に呼ばれる関数を設定 */
```

```
    driver.devdetach  = &msc_detach;
    /* デバイスをサスペンド状態に移行時に呼ばれる関数を設定 */
    driver.devsuspend = &msc_suspend;
    /* デバイスのサスペンド状態解除時に呼ばれる関数を設定 */
    driver.devresume  = &msc_resume;

    /* HCD へクラスドライバ情報を登録 */
    R_usb_hstd_DriverRegistration(&driver);
}
```

（注1） TPL(Target Peripheral List)はアプリケーション内で定義してください。TPL については USB Basic Firmware アプリケーションノート(Document No.R01AN2633JJ)を参照してください。

4.4 メインルーチン

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_usb_cstd_Scheduler()）を呼び出すことで動作します。

メインルーチン内で R_usb_cstd_Scheduler()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_usb_cstd_Scheduler()呼び出し後、R_usb_cstd_CheckSchedule()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）

```
void usb_hapl_mainloop(void)
{
  while(1)  /* メインルーチン */
  {
    /* イベントの確認と取得、フラグセット (注 1) */
    R_usb_cstd_Scheduler();

    /*イベント有無の判定、フラグクリア */
    if(USB_FLGSET == R_usb_cstd_CheckSchedule())
    {
        R_usb_hstd_MgrTask();    /* MGR タスク */
        R_usb_hhub_Task();       /* HUB タスク (注 3) */
        R_usb_hmsc_task();       /* HMSC タスク */
        R_usb_StrgDriveTask();   /* HSTRG タスク */
    }
    msc_main();                  /* APL */
  }
}
```

(注 2)

（注1） R_usb_cstd_Scheduler()でイベントを取得後、処理を行う前に再度 R_usb_cstd_Scheduler()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） これらの処理は、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。

4.4.1 APL

APL は、ステート遷移により管理を行っています。
Table 4-1 にステート一覧を示します。

Table 4-1 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

Figure 4-2 に APL の処理フローチャートを示します。

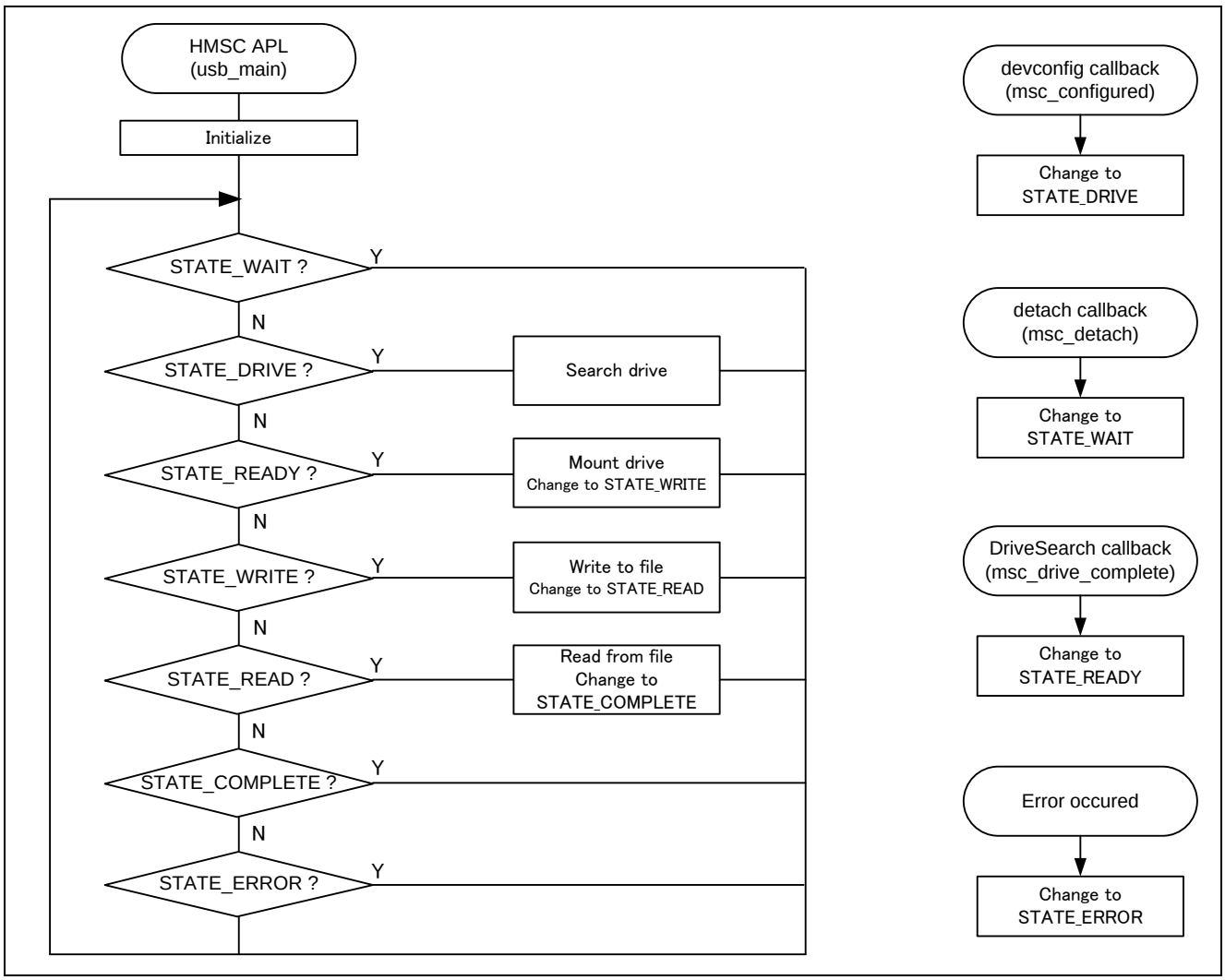


Figure 4-2 メインループ処理フローチャート

4.4.2 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、USB_HCDREG_t 構造体のメンバ devconfig に設定されたコールバック関数 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（drive_search_lock）を確認して、オフであれば処理を開始します。
2. drive_search_lock をオンにします。
3. R_usb_hmsc_StrgDriveSearch()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_usb_hmsc_StrgDriveSearch()に登録したコールバック関数 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 hmscdemX.txt を作成します。ファイル名中の X 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 hmscdem1.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 hmscdemX.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. ドライブ番号に対応した LED を点灯します。
6. 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 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 msc_detach()は、USB_HCDREG_t 構造体のメンバ devdetach に設定した関数です。

5. FatFs 組み込み手順

本サンプルプログラムをビルドするには、お客様にて FatFs を組み込む必要があります。
FatFs の組み込み手順を以下に示します。

5.1 FatFs をウェブサイトから入手

FatFs は、下記の URL でオープンソースとして公開されています。
http://elm-chan.org/fsw/ff/00index_e.html

1. ページを下へスクロールすると、ダウンロード用のリンクが「Resources」セクション内に見つかります。
2. 「FatFs R0.13」をクリックして FatFs をダウンロードし、任意のフォルダへ保存して下さい。

本サンプルプログラムは、バージョン R0.13 用に作成されています。FatFs が更新されている場合には、「Previous release」リンク内からこのバージョンを探して下さい。

5.2 FatFs を適切なフォルダへ展開

ダウンロードした FatFs の ZIP ファイル(ff13.zip)を展開し、Figure 5-1 に示す通りにサンプルプログラムのワークスペース内へ移動して下さい。
サンプルプログラムは、このフォルダ構成以外では動作しませんので、ご注意下さい。

```
RZ_T1_USBh_HMSC/
|-- inc/
`-- src/
    |-- common/
    |   |-- nor_boot/
    |   `-- serial_boot/
    |-- drv/
    |   `-- usbh/
    |       |-- basic/
    |       `-- hmsc/
    |           `-- ff13/   <--- "FatFs" here!
    |               |-- documents/
    |               |   |-- doc/
    |               |   `-- res/
    |               `-- source/
    |
    `-- sample/
```

Figure 5-1 FatFs を展開するフォルダ

5.3 ワークスペースを開き、プロジェクトをビルド

サンプルプログラムのワークスペースを開き、サンプルプロジェクトをビルドして下さい。
サンプルプログラムは、FatFs をリンクするように設定されていますが、開発環境が e² studio または DS-5 の場合、ff13 フォルダ直下の documents フォルダと source 下の diskio.c ファイルをビルド対象外としてください。

5.4 注意事項

お客様の製品へ FatFs を組み込む場合には、FatFs のライセンスをご確認いただき、お客様の責任にて実施して下さい。

Appendix A. 初期設定の変更点

USB-BASIC-F/W を動作させるために「RZ/T1 グループ初期設定 Rev.1.30」を変更しています。
サンプルプログラムは、IAR embedded workbench for ARM(以下、EWARM)と DS-5 と e² studio の環境をサポートしています。
本章では、変更点について記述します。

フォルダとファイル

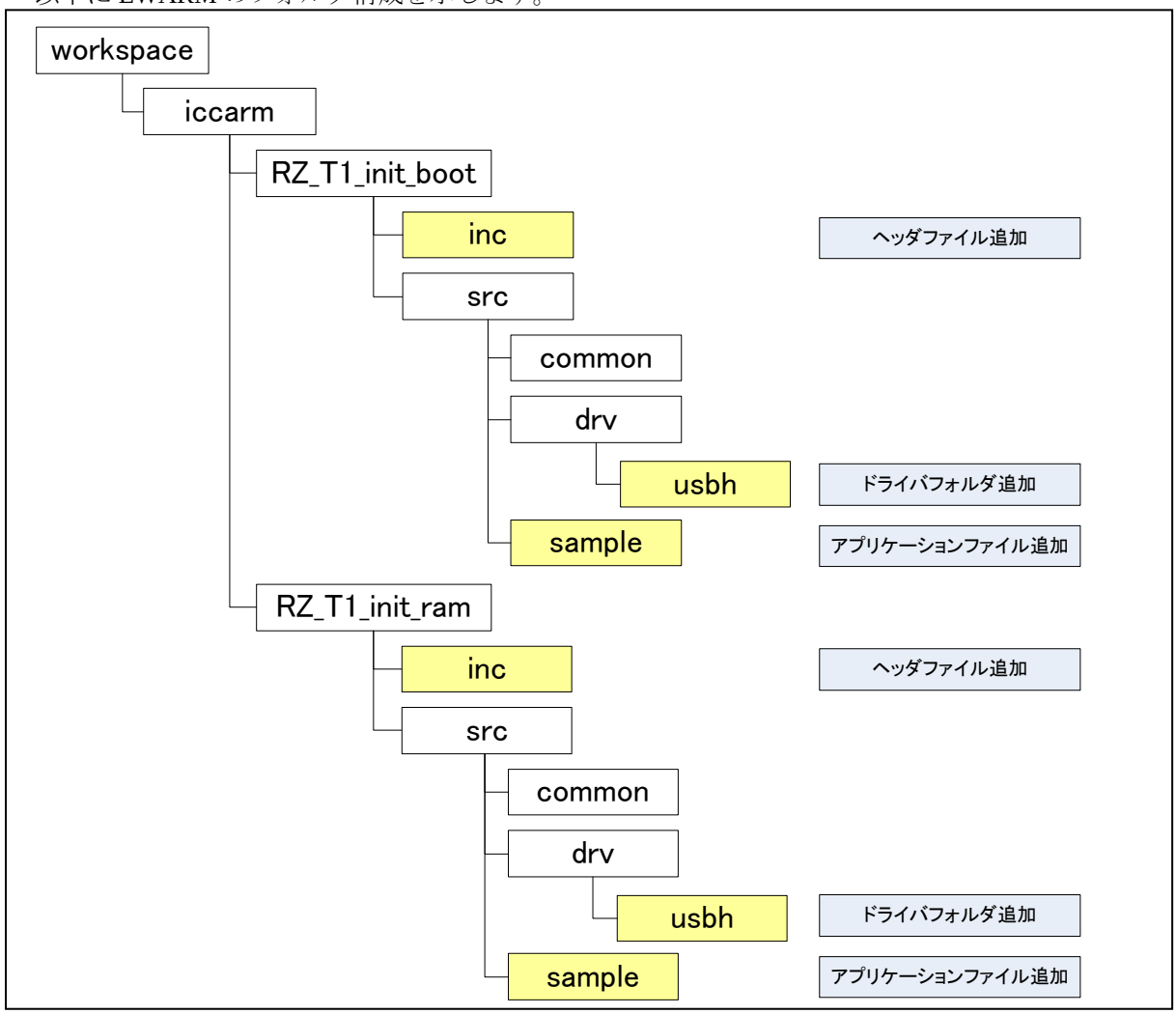
「RZ/T1 グループ初期設定 Rev.1.30」では、開発環境とブート方法によってフォルダ構成が異なります。
全ての開発環境とブート方法の各フォルダに対して、下記の変更をしています。

・”inc”フォルダに下記のファイルを追加
r_usb_basic_config.h
r_usb_basic_if.h
r_usb_hatapi_define.h
r_usb_hmsc_config.h
r_usb_hmsc_if.h

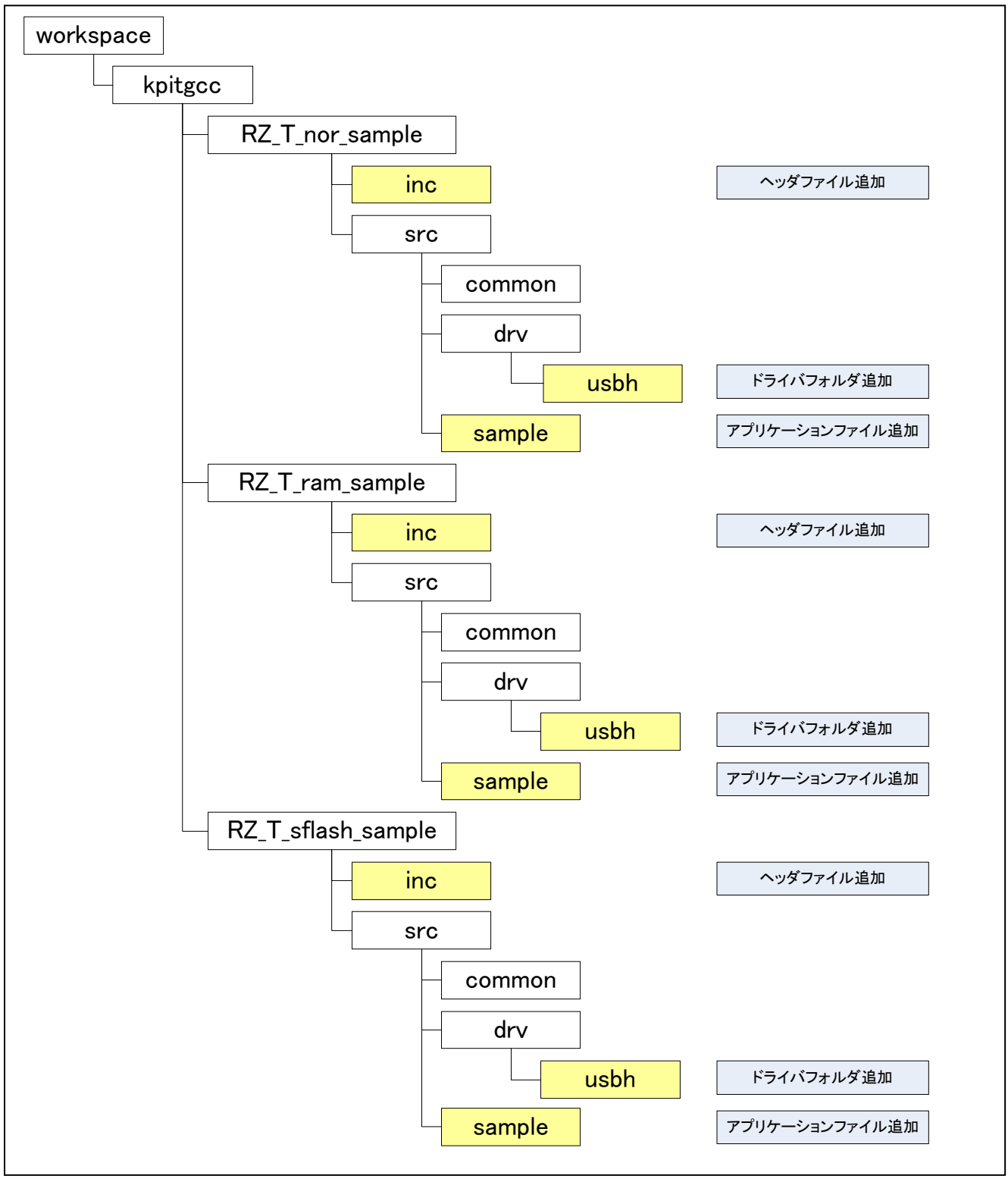
・”sample”フォルダに下記のファイルを追加
r_usb_main.c
r_usb_hmsc_apl.c
r_usb_hmsc_apl.h

・”drv”フォルダに usbh フォルダと usbh フォルダ以下のファイルを追加

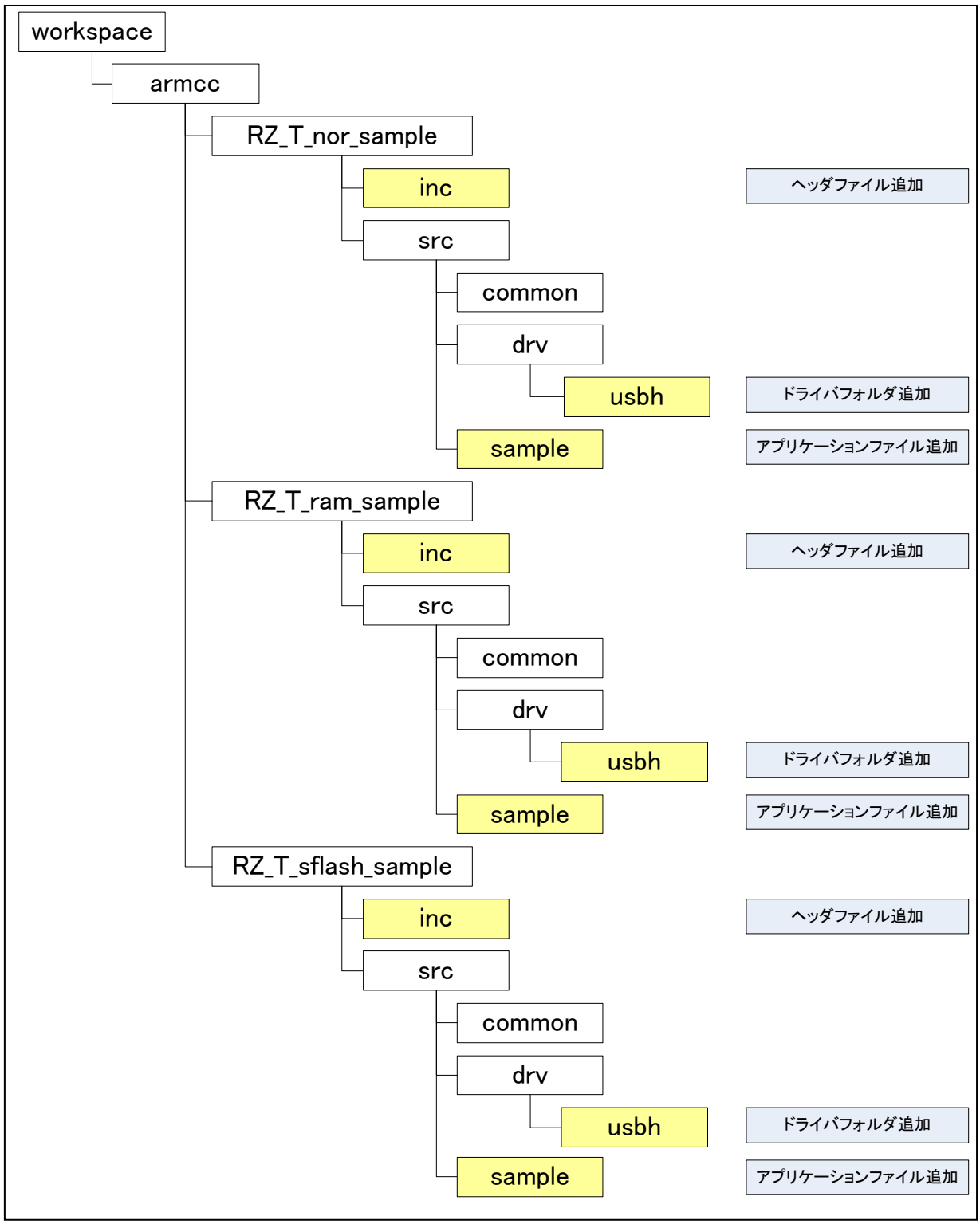
以下に EWARM のフォルダ構成を示します。



以下に e² studio のフォルダ構成を示します。



以下に DS-5 のフォルダ構成を示します。



セクション

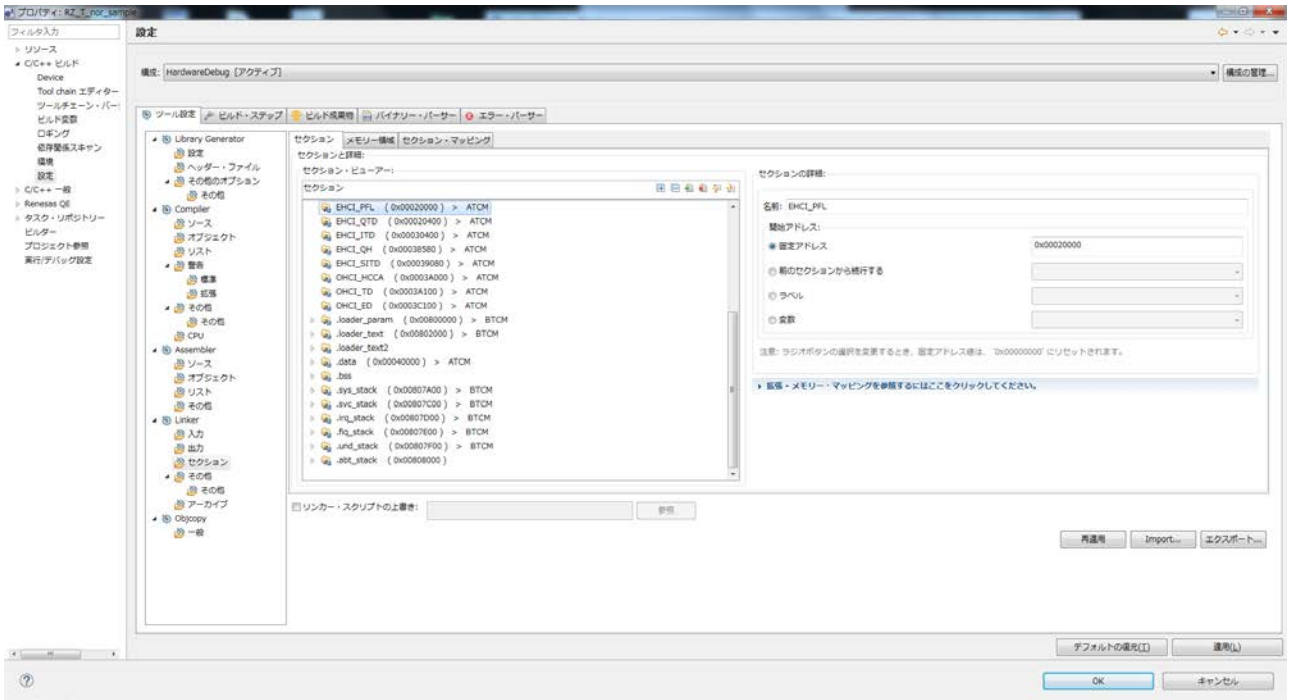
コード領域とデータ領域のセクションサイズを変更して、以下のセクションを追加しています。

セクション名	アドレス	割り当てる変数	ファイル
EHCI_PFL	0x00020000	ehci_PeriodicFrameList	r_usb_hEhciMemory.c
EHCI_QTD	0x00020400	ehci_Qtd	
EHCI_ITD	0x00030400	ehci_Itd	
EHCI_QH	0x00038580	ehci_Qh	
EHCI_SITD	0x00039080	ehci_Sitd	
OHCI_HCCA	0x0003A000	ohci_hcca	r_usb_hOhciMemory.c
OHCI_TD	0x0003A100	ohci_TdMemory	
OHCI_ED	0x0003c100	ohci_EdMemory	

e² studio

e² studio の設定画面でセクションを設定しています。
変更内容は下記の通りです。
・.data セクションの固定アドレスを 0x0007F000 から 0x00040000 に変更
・EHCI と OHCI のセクション設定を追加

[プロジェクト] → [プロパティ] → [C/C++ ビルド] → [設定] → [セクション] で参照できます。



コード内の変数定義は下記の通りです。

r_usb_hEhciMemory.c

```
#ifdef __GNUC__
static uint32_t         ehci_PeriodicFrameList[ USB_EHCI_PFL_SIZE ]
                __attribute__ ((section ("EHCI_PFL")));
static USB_EHCI_QH      ehci_Qh[ USB_EHCI_NUM_QH ]
                __attribute__ ((section ("EHCI_QH")));
static USB_EHCI_QTD     ehci_Qtd[ USB_EHCI_NUM_QTD ]
                __attribute__ ((section ("EHCI_QTD")));
static USB_EHCI_ITD     ehci_Itd[ USB_EHCI_NUM_ITD ]
                __attribute__ ((section ("EHCI_ITD")));
static USB_EHCI_SITD    ehci_Sitd[ USB_EHCI_NUM_SITD ]
                __attribute__ ((section ("EHCI_SITD")));
#endif  /* __GNUC__ */
```

r_usb_hOhciMemory.c

```
#ifdef __GNUC__
static USB_OHCI_HCCA_BLOCK              ohci_hcca
                __attribute__ ((section ("OHCI_HCCA")));
static USB_OHCI_HCD_TRANSFER_DESCRIPTOR ohci_TdMemory[USB_OHCI_NUM_TD]
                __attribute__ ((section ("OHCI_TD")));
static USB_OHCI_HCD_ENDPOINT_DESCRIPTOR ohci_EdMemory[USB_OHCI_NUM_ED]
                __attribute__ ((section ("OHCI_ED")));
#endif  /* __GNUC__ */
```

EWARM

EWARM では、リンカ設定ファイル(.icf ファイル)でセクションを設定しています。

RAM 領域の開始アドレスを 0x00070000 から 0x00040000 に、USER_PRG 領域の終了アドレスを 0x0001FFFF に変更しています。

```
define symbol __ICFEDIT_region_RAM_start__   = 0x00040000;

define symbol __region_USER_PRG_end__        = 0x0001FFFF;
```

EHCI と OHCI を固定アドレスにするため、メモリリージョン定義を追加しています。

```
define region EHCI_MEM1_region = mem:[from 0x00020000 to 0x000203FF];
define region EHCI_MEM2_region = mem:[from 0x00020400 to 0x00039FFF];

define region OHCI_MEM1_region = mem:[from 0x0003A000 to 0x0003A0FF];
define region OHCI_MEM2_region = mem:[from 0x0003A100 to 0x0003FFFF];

do not initialize  { section EHCI_PFL, section EHCI_QH, section EHCI_QTD, section EHCI_ITD, section
EHCI_SITD, section OHCI_HCCA, section OHCI_TD, section OHCI_ED };

place in EHCI_MEM1_region { section EHCI_PFL };
place in EHCI_MEM2_region { section EHCI_QH, section EHCI_QTD, section EHCI_ITD, section EHCI_SITD };

place in OHCI_MEM1_region { section OHCI_HCCA };
place in OHCI_MEM2_region { section OHCI_TD, section OHCI_ED };
```

コード内の変数定義は下記の通りです。

r_usb_hEhciMemory.c

```
#ifdef __ICCARM__
#pragma location="EHCI_PFL"
static uint32_t                 ehci_PeriodicFrameList[ USB_EHCI_PFL_SIZE ];
#pragma location="EHCI_QH"
static USB_EHCI_QH              ehci_Qh[ USB_EHCI_NUM_QH ];
#pragma location="EHCI_QTD"
static USB_EHCI_QTD             ehci_Qtd[ USB_EHCI_NUM_QTD ];
#pragma location="EHCI_ITD"
static USB_EHCI_ITD             ehci_Itd[ USB_EHCI_NUM_ITD ];
#pragma location="EHCI_SITD"
static USB_EHCI_SITD            ehci_Sitd[ USB_EHCI_NUM_SITD ];
#endif  /* __ICCARM__ */
```

r_usb_hOhciMemory.c

```
#ifdef __ICCARM__
#pragma location="OHCI_HCCA"
static USB_OHCI_HCCA_BLOCK                  ohci_hcca;
#pragma location="OHCI_TD"
static USB_OHCI_HCD_TRANSFER_DESCRIPTOR     ohci_TdMemory[USB_OHCI_NUM_TD];
#pragma location="OHCI_ED"
static USB_OHCI_HCD_ENDPOINT_DESCRIPTOR     ohci_EdMemory[USB_OHCI_NUM_ED];
#endif  /* __ICCARM__ */
```

DS-5

DS-5 では、scatter ファイルでセクションを設定しています。

RAM 領域の開始アドレスを 0x00040000 に変更、0 クリアするメモリ(ZI)領域を DATA 領域に続けて確保するよう修正しています。

```
DATA            0x00040000    UNINIT
{
    * (+RW)
}
BSS             +0
{
    * (+ZI)
}
```

EHCI と OHCI を固定アドレスにするため、メモリ定義を追加しています。

```
    EHCI_PERIODIC_FRAMELIST 0x00020000    0x400
    {
            r_usb_hEhciMemory.o(EHCI_PFL)
    }
    EHCI_QTD          +0
    {
            r_usb_hEhciMemory.o(ehci_Qtd)
    }
    EHCI_ITD          +0
    {
            r_usb_hEhciMemory.o(ehci_Itd)
    }
    EHCI_QH           +0
    {
            r_usb_hEhciMemory.o(ehci_Qh)
    }
    EHCI_SITd         +0
    {
            r_usb_hEhciMemory.o(ehci_Sitd)
    }
    OHCI_HCCA         0x0003A000      0x100
    {
            r_usb_hOhciMemory.o(OHCI_HCCA)
    }
    OHCI_TDMEMORY     +0
    {
            r_usb_hOhciMemory.o(OHCI_TD)
    }
    OHCI_EDMEMORY     +0
    {
            r_usb_hOhciMemory.o(OHCI_ED)
    }
```

コード内の変数定義は下記の通りです。

r_usb_hEhciMemory.c

```
#ifdef __CC_ARM
#pragma arm section zidata = "EHCI_PFL"
static uint32_t         ehci_PeriodicFrameList[ USB_EHCI_PFL_SIZE ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_QH"
static USB_EHCI_QH      ehci_Qh[ USB_EHCI_NUM_QH ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_QTD"
static USB_EHCI_QTD     ehci_Qtd[ USB_EHCI_NUM_QTD ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_ITD"
static USB_EHCI_ITD     ehci_Itd[ USB_EHCI_NUM_ITD ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_SITD"
static USB_EHCI_SITD    ehci_Sitd[ USB_EHCI_NUM_SITD ];
#pragma arm section zidata
#endif
```

r_usb_hOhciMemory.c

```
#ifdef __CC_ARM
#pragma arm section zidata = "OHCI_HCCA"
static USB_OHCI_HCCA_BLOCK                  ohci_hcca;
#pragma arm section zidata
#pragma arm section zidata = "OHCI_TD"
static USB_OHCI_HCD_TRANSFER_DESCRIPTOR    ohci_TdMemory[USB_OHCI_NUM_TD];
#pragma arm section zidata
#pragma arm section zidata = "OHCI_ED"
static USB_OHCI_HCD_ENDPOINT_DESCRIPTOR    ohci_EdMemory[USB_OHCI_NUM_ED];
#pragma arm section zidata
#endif
```

USB-BASIC-F/W 関数の呼び出し

¥src¥sample¥int_main.c の main()に USB-BASIC-F/W の usbh_main()の呼び出しを追加しています。

```
extern void usbh_main(void);

int main (void)
{
    /* Initialize the port function */
    port_init();

    /* Initialize the ECM function */
    ecm_init();

    /* Initialize the ICU settings */
    icu_init();

    /* USBh main */
    usbh_main();

    while (1)
    {
        /* Toggle the PF7 output level(LED0) */
        PORTF.PODR.BIT.B7 ^= 1;

        soft_wait();  // Soft wait for blinking LED0

    }

}
```

ホームページとサポート窓口

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

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

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Aug 21, 2015	—	初版発行
1.10	Dec 25, 2015	44	Appendix A 追加
1.20	Feb 29, 2016	47,51,52	DS-5 関連情報追加
1.30	Dec 07, 2017	—	RZ/T1 初期設定 Ver 1.30 に対応
		43	FatFs のバージョンとフォルダ構成図を変更

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。

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、その他の不適切に使用しないでください。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、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、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。また、当社製品および技術を、(1)核兵器、化学兵器、生物兵器等の大量破壊兵器およびこれらを運搬することができるミサイル（無人航空機を含みます。）の開発、設計、製造、使用もしくは貯蔵等の目的、(2)通常兵器の開発、設計、製造または使用の目的、または(3)その他の国際的な平和および安全の維持の妨げとなる目的で、自ら使用せず、かつ、第三者に使用、販売、譲渡、輸出、賃貸もしくは使用許諾しないでください。
   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. お客様の転売、貸与等により、本書（本ご注意書きを含みます。）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、当社は一切その責任を負わず、お客様にかかる使用に基づく当社への請求につき当社を免責いただきます。
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. 本資料に記載された情報または当社製品に関し、ご不明点がある場合には、当社営業にお問い合わせください。

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。

(Rev.3.0-1　2016.11)

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：https://www.renesas.com/contact/