Resource Saving ファイルシステム マニュアル

2014. 9.14
Resource Saving System
Version.1.0.3

目次

図目次

表目次

1 ファイルシステム概要

1.1 はじめに

「Resource Saving ファイルシステム」は ROM や RAM の少ないシステム向けに実装するためのファイルシステムです。FAT12/FAT16 をサポートしています。ファイルシステムはデバイスに依存しませんがサンプルとして SD カードのドライバを用意しています。

SD カードの規格では SD が FAT12/16、SDHC が FAT32、SDXC が exFAT に対応しています。

1.2 特徴

少ない RAM サイズで動作可能

書き込みをおこなわない場合にはバッファサイズを最低2バイトの指定とすることができます。ただし、ドライバの仕組みに依存します。

書き込みをおこなう場合にはセクタサイズ(通常 512 バイト)のバッファ指定が最低となります。

アプリケーションインタフェースが使用するスタック領域は約 500 バイトです。

調整可能なバッファサイズ

上の説明にもあるように、使用するバッファサイズを調整できます。

RAM 容量の逼迫しているシステムではメモリ使用量をおさえて実装することができます。また、メモリに余裕のあるシステムではバッファを多めに確保しアクセスを高速におこなうように調整が可能です。

実装が容易

ファイルシステムのアプリケーションインタフェースは関数呼び出しでおこないます。ファイルシステムのソースをアプリケーションと一緒にコンパイル、リンクすることにより実装できます。

デバイスに直接アクセスするドライバのサンプルを用意しています。サンプルを参考にしながら実装することができます。

1.3 制限事項

ロングファイル名使用不可

ロングファイル名は255文字までのファイル名を使用できる機能です。このファイルシステムではロングファイル名は使用できません。8 文字+拡張子 3 文字の形式のファイル名の指定のみが可能です。

漢字ファイル名使用不可

8 文字+拡張子 3 文字の形式のファイル名であっても漢字のファイル名は使用できません。漢字コード中に含まれる 0x5c(“¥”)、0xe5(削除されたエントリ)についての処理をおこなっていないため漢字コードのファイル名には対応できません。

FAT12/FAT16 のみの対応

FAT32、exFAT には対応していません。FAT12/16 のみの対応です。

使用できるデバイスは1つ

複数のデバイスを同時に扱うことはできません。したがってドライブレターは使いません。

1.4 その他の特性

空き領域検索

空き領域の検索時は前に書き込んだクラスタ位置の次からおこないます。毎回 FAT を先頭から検索することはしないので新規にファイルを作る時や書き込みで次のクラスタを検索する際にはすぐに空きクラスタを見つけることができます。

1.5 テストについて

Splint

静的解析ツールの Splint を実行し指摘された項目のあいまいな記述や誤りのあった記述について修正をおこなっています。修正する必要が無いと判断した指摘事項についてはその理由を記述して残しています。

動作確認テスト

各動作環境においてドライバとファイルシステムの動作確認をおこなえるソフトウェアを実装しています。

1.6 ファイルシステム修正履歴

表 1-1 ファイルシステム修正履歴

バージョン	日付	概要
1.01	2012.10.28	初期リリース
1.02	2014. 3. 9	・ファイル名に’@’が使用できなかった不具合対応 ・ディレクトリ情報読み取り[RfReadDir()]で小文字ファイル名が大文字になっていた不具合対応 ・RfOpenDir()でルートディレクトリがオープンできなかった不具合対応 ・ドライバ識別情報の型を変更した ・H8/3048 HEW、GNU、LPC1768 の評価パッケージを追加した

1.7 ソフトウェア構造

ソフトウェアは図で示すような階層構造になっています。
ドライバ部分はサンプルの H8/3048 用 SD カードドライバを例として説明しています。

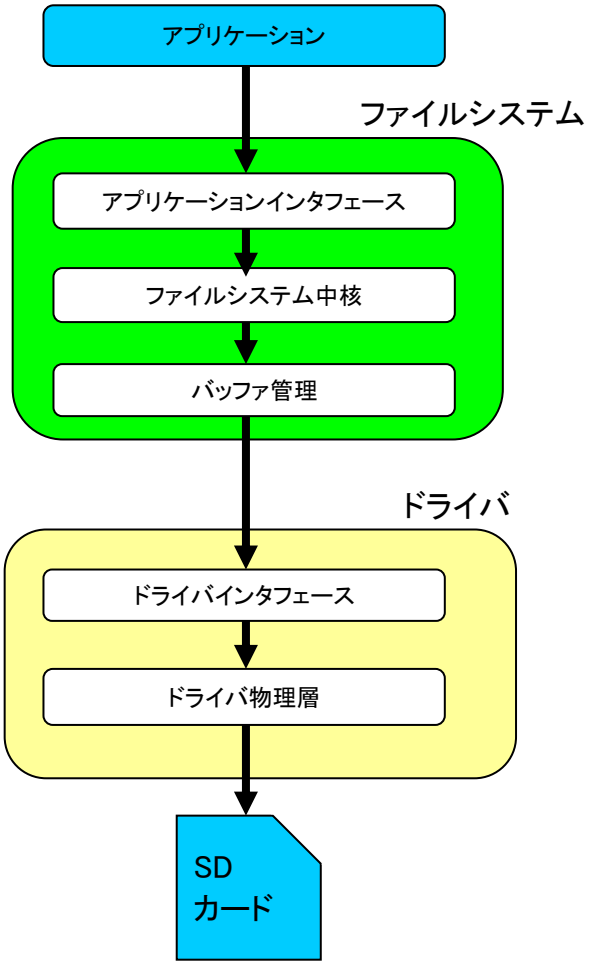

図 1-1 ソフトウェア構造図

各階層の概要を説明します。
ドライバ部分はハードウェアに合わせて実装する必要があります。

表 1-2 各階層の説明

階層名称	説 明
アプリケーションインタフェース	アプリケーションインタフェースです。アプリケーションはこの部分の関数を呼び出すことでファイルへのアクセスがおこなえます。
ファイルシステム中核	ファイルシステムの中核部分です。
バッファ管理	メディアをフラットなメモリとして扱うためのソフトウェア群です。
ドライバインタフェース	ファイルシステムのドライバインタフェースを備えたドライバです。対応するメディアとハードウェアに合わせてドライバインタフェース以下の部分を用意してください。 サンプルは SD カードの SPI モードドライバです。
ドライバ物理層	サンプルの SD カード用ドライバに特化した物理層部分です。 H8/3048 の内蔵デバイスに直接アクセスする部分をまとめています。シリアル通信、タイマ操作、ポート操作です。別のハードウェアに移植する場合は変更が必要です。

1.8 調整方法

rfSettings.c の以下の値を調整してください。

表 1-3 調整値一覧

定数名	初期値	数値の意味
GEN_BUFF_SIZE バッファサイズ	1024	読み書きに使用するバッファのサイズです。最小でも 2 バイトを指定しなければなりません。書き込みをおこなう場合はセクタサイズの 512 以上が必要です。バッファサイズはブロックサイズの整数倍の値を指定してください。バッファ管理についての詳細は「1.9 バッファサイズ」を 参照してください。
SAME_TIME_FILE_NUM 同時に開けるファイル数	1	1つのファイルのオープン中に別のファイルへのアクセスをおこなう場合など、同時にアクセス可能なファイル数を指定します。ファイルの情報を格納する領域をここで指定した数ぶん確保します。
DEL_DIR_NEST dfDelDir のネスト数	4	ディレクトリの削除では指定したディレクトリの下にあるサブディレクトリも削除します。最も下層にあるディレクトリを先に削除しなければならないためネストを検索していきます。最下層のディレクトリにたどり着くまでの間のディレクトリをすべて覚えておく必要があるのですが、上限を設けておかないとスタック領域を消費しつくしてしまいます。

また、rfSettings.c にはドライバ関数、システム関数を登録します。システム関数にはファイルの作成時刻を設定するための時刻取得機能があります。

1.9 バッファサイズの考え方

1.9.1 セクタサイズ未満の場合

書き込みをおこなう場合のバッファサイズはデバイスのセクタサイズ以上が必要です。通常は 512 バイトです。読み取りしかおこなわない場合は最低2バイトの指定をすることができます。ただし、セクタサイズ未満のバッファサイズを指定すると処理速度が遅くなります。物理的なデバイスの読み書きはセクタサイズ単位でしかおこなえないので、例えばセクタサイズが 512 バイトの際にバッファが 256 バイトだった場合は同じ場所を2回読むことになります。

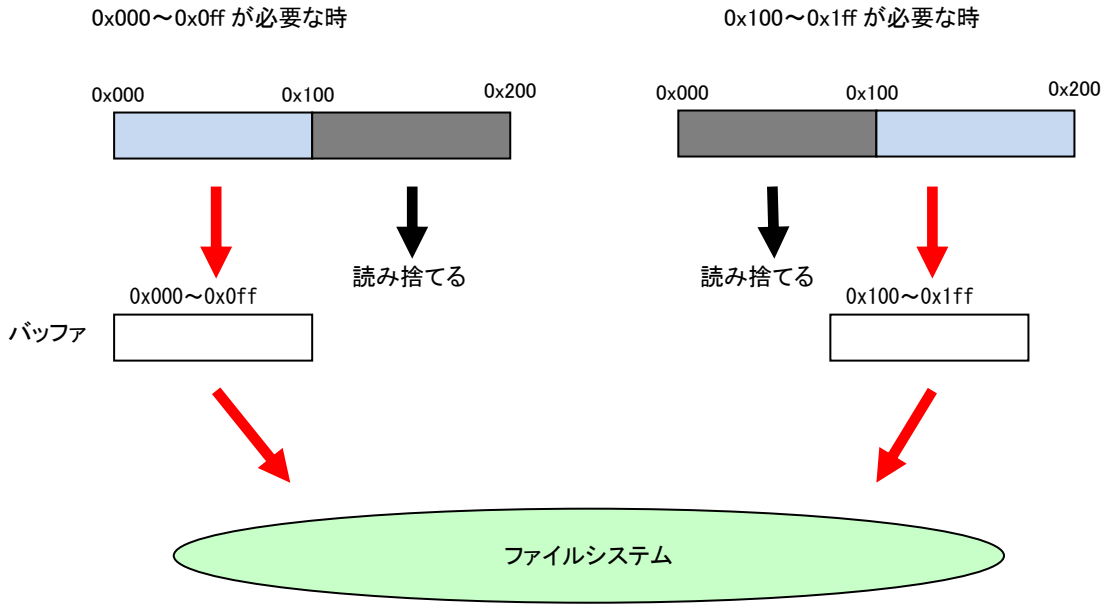

図 1-2 セクタサイズ未満のバッファ

1.9.2 複数ファイル同時使用時

同時に複数ファイルをオープンする際などにバッファサイズが1セクタ分しかないとその領域を供用するため効率が落ちます。

以下の例は「ファイル A」→「ファイル B」→「ファイル A」の順番に読み取りをする例です。異なるファイルを読むたびにバッファの内容が破棄され読み直されます。

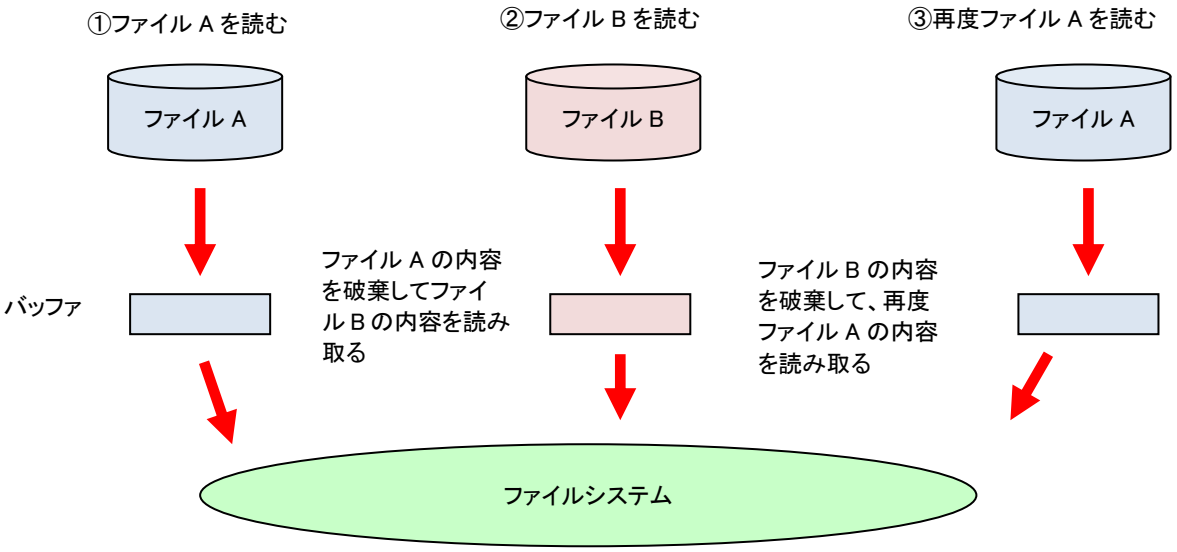

図 1-3 1つのバッファで複数ファイルを読む

同時に複数ファイルをオープンする場合はファイル数に応じてバッファサイズを確保すると効率が上がります。例えば同時に4つのファイルをオープンするのであればセクタサイズの 512×4 の 2048 を確保します。こうすると各ファイルに 512 バイトが割り当てられるので他のファイルの影響を受けづらくなります。

1.9.3 FAT のアクセス

通常の SD カードのフォーマットでは 32 セクタで1つのクラスタが構成されています。16,384 バイトを読み取ると次のクラスタ位置を求めるため FAT を参照します。FAT16 の場合は 2 バイトで1つのクラスタ情報を持つため、FAT 領域1セクタには 256 クラスタの情報が格納されています。

FAT16 でクラスタサイズが 32 セクタ(=16,384 バイト)の例で FAT へのアクセス頻度を計算してみます。

FAT 領域1セクタに 256 個のクラスタ情報が格納されています。そのため FAT 用のバッファが別にある場合は 256 クラスタのデータを読むと1回の FAT 領域の読み取りが発生します。つまり、256×16,384 = 4,194,304 バイトのデータを読むと 512 バイト(=1 セクタ)の FAT 領域の読み取りが付随して発生していることになります。データ 4,194,304 バイトの読み取りで 8193 回(= 256×32+1)の物理的なセクタリードが発生しています。

一方、FAT 用のバッファが確保されていない場合はFAT 領域を必要とする都度物理的な読み取りが発生するので 8448 回(=256×32+256)のアクセスとなります。

このパフォーマンスの差は (8448-8193)÷8193×100 ≒ 3.1% となります。そのためバッファ 512 バイトを追加確保すれば 3.1%のパフォーマンス向上が図れるということを意味します。また、DOS の標準的なフォーマットでは 1 クラスタは 8 セクタで構成されます。この場合に FAT 用のバッファを確保しなかった際のパフォーマンス悪化は約 13%となります。

システムの状況によりメモリの使用量削減とパフォーマンス向上の重要度は異なると思われます。必要に応じて調整してください。

1.9.4 ディレクトリのアクセス

ディレクトリのアクセスはファイルのオープン時とクローズ時、それとフラッシュ実行時のみとなります。これらのどのケースもデータ部分へのアクセス中ではないので読み直しが発生することはありません。

1.10 実装方法

実装方法、実装の際の注意点について説明します。

1.10.1 実装するファイル

既存のシステムにファイルシステムを追加実装する場合、以下のファイルを新たにプロジェクトに追加してください。

・FileSystem フォルダ以下のファイル
・ファイルシステム設定ファイル、rfSettings.c
・そのシステム用に作成したドライバファイル

フォーマットをおこなわないシステムでは FileSystem フォルダの RfFormat.c をリンクする必要はありません。
ファイルシステムとドライバの関連付けは rfSettings.c ファイルに関数を登録することでおこないます。
「6.ファイルシステム評価パッケージ」で実装例を説明していますので参考にしてください。

1.10.2 使用の際の注意点

ファイルシステムはリエントラントな作りにはなっていません。そのためファイルシステムへのアクセスは一か所でおこなうかアプリケーション側で排他処理をおこなってください。

1.10.3 OS 無しでの使用

read/write ではバッファサイズごとに物理的な I/O が発生します。そのため実行時間は物理的な I/O が発生する際とそうでない場合で大きな違いがあります。そのため割り込みでファイルシステムを使用することは望ましくありません。物理的な I/O で時間がかかってしまうときに他の割り込みに影響を与えてしまうからです。

できるだけファイルへのアクセスは優先度の低いメインループに集約して、メインループには物理的な I/O にかかる時間より早いタイミングでおこなわなければならない処理は入れないようにしてください。パフォーマンスについては各サンプルプロジェクトで説明しています。

1.10.4 リアルタイム OS 下での使用

ドライバで物理的な I/O が発生した際に待ちが発生することがあります。そのためタスク優先度や同じタスクでおこなう他の処理との兼ね合いに注意が必要です。

ファイルシステムをリアルタイム OS 下で効率よく使用するための実装はドライバ部分の作りに大きく依存します。改めてサンプルドライバの章でドライバの作りとともに説明します。「8 リアルタイム OS 下での使用」を参照してください。

1.10.5 ハードウェアに合わせた実装

ファイルシステムを新しいハードウェアに実装する際にはハードウェアや開発環境に合わせて以下の機能を作成する必要があります。

(1)ドライバ

メディアへのアクセスドライバ。SD カードでは、SD コマンドを管理する部分、SPI 通信部分になります。

(2)システム機能

現在時刻取得の機能です。ファイルの作成日時や更新日時設定時に呼び出されます。

(3)テスト機能

テストプログラムを実行させるためのユーザーインタフェース部です。

この機能の実装は必須ではありませんが、ドライバやシステム機能とファイルシステムの結合がうまくおこなえたかを検証するために同梱のテストプログラムを動作させて確認することをお勧めいたします。テストプログラムを動作させる際に必要となります。

1.11 ファイルシステム評価パッケージ

1.11.1 パッケージ一覧

ファイルシステムの動作をすぐに確認できるよう、以下の一覧に示すパッケージを用意しています。

No.	パッケージ	対象 CPU	対象ボード	概要
1	RsfAkiH8SCI	H8/3048F	AKI-H8/3048F (秋月電子通商)	AkiH8 のビルド環境。SPI モードドライバは SCI のみを使用し割り込みは使わない。
2	RsfAkiH8DMA	H8/3048F	AKI-H8/3048F (秋月電子通商)	AkiH8 のビルド環境。DMA+SCI で SPI モード通信をおこなう。
3	RsfGcc_H8	H8/3048F	AKI-H8/3048F (秋月電子通商)	GNU でビルド環境を実現。ドライバは AkiH8DMA と同様。
4	RsfHEW_DMA	H8/3048F	AKI-H8/3048F (秋月電子通商)	ルネサスエレクトロニクスの統合開発環境、HEW を使用した環境。
5	RsfLPC1768Xpresso	LPC1768	LPC1768Xplorer (NGX TECHNOROGIES)	LPCXpressoIDE の開発環境。DMA+SSP で SPI モード通信をおこなう。
6	RsfVcpp		WindowsPC	Windows 上でシミュレーションできるドライバを用意し、ファイルシステムの動作をおこなえるようにした環境。

1.11.2 各パッケージの説明

1.11.2.1 RsfAkiH8SCI

SD カードとの通信で SCI のみを使用した AKI-H8 の動作環境です。SD カードとの通信では割り込みを使用せず、ポーリングで送受信をおこないます。そのため処理の流れを追いやすく理解しやすい形式となっています。
「AkiH8SCI」フォルダ直下にビルドのためのバッチファイル、リンカの設定ファイル、ベクタテーブル、デバッグ出力ファイル、メインソースコードなどが格納されています。

1.11.2.2 RsfAkiH8DMA

SD カードとの通信で DMA と SCI を使用する AKI-H8 の動作環境です。SD カードとの通信は割り込みを使用して非同期に動作し「AkiH8SCI」よりもパフォーマンスを重視しています。このドライバはシーケンシャル read/write の最適化オプションに対応しています。
「AkiH8DMA」フォルダ直下にビルドのためのバッチファイル、リンカの設定ファイル、ベクタテーブル、デバッグ出力ファイル、メインソースコードなどが格納されています。

1.11.2.3 RsfGcc_H8

GNU の環境です。ドライバの動作は AkiH8DMA と同様です。Cygwin や Linux 上で makefile を使用してビルドします。DA コンバータを使用して wev ファイルを再生するサンプルプロジェクトが同梱されています。
ドライバは「AkiH8DMA」と同様です。

1.11.2.4 RsfHEW_DMA

ルネサスエレクトロニクス®の HEW 試用版の動作環境です。ドライバの動作は AkiH8DMA と同様です。
wav ファイル再生のサンプルプロジェクトが含まれています。

1.11.2.5 RsfLPC1768Xpresso

NXP セミコンダクターズ社の LPC1768Xplorer ボードを使用したサンプルです。wev ファイルを再生するサンプルでは LPCXpressoIDE が提供する FreeRTOS を使用し PWM を使用して音を再生しています。

1.11.2.6 RsfVcpp

windows 上で動作するドライバを使用した Microsoft®の Visual Studio 2010 での動作環境です。ハードウェアが無い状況でもファイルシステムの動作確認がおこなえます。
「Vcpp」フォルダ直下にプロジェクトファイルが格納されています。「RfTestMain」フォルダにデバッグ出力ファイル、メインのソースコードなどが格納されています。

1.11.3 各パッケージ共通の内容

各パッケージには全く同様の「FileSystem」、「Test」フォルダが含まれます。それぞれ「2.ファイルシステム」、「5.テストプログラム」で説明します。

2 ファイルシステム
2.1 ファイル一覧

ファイルシステムのソースファイル一覧です。

表 2-1 ファイルシステムのソースファイル一覧

フォルダ	ソースファイル	ファイル内容
FileSystem¥	RfBuffer.def	RfSettings.c の内容を反映させる定義
	RfInterface.c(.h)	ファイルシステムアプリケーションインタフェース
	RfFormat.c(.h)	フォーマットをおこなう
	RfMiddle.c(.h)	ファイルシステム用部品関数
	RfErrCode.h	エラーコード定義
	RfDiskLayout.h	ディスクレイアウト定義
	RfDriverInterface.h	ドライバインタフェース定義
	RfControlBuff.c(.h)	バッファ管理
	RfGeneral.c(.h)	ファイルシステム共通に使用する汎用機能

2.2 関数、構造体一覧

ファイルシステムのアプリケーションインタフェース関数、システム固有に用意する関数、使用する構造体の一覧を示します。

表 2-2 アプリケーションインタフェース関数一覧

分 類	関 数	概 要
初期化、終了処理	RfInitFileSystem	ファイルシステムの初期化、ファイルシステム使用前に1回だけおこなう
	RfConclude	ファイルシステムの終了、SD カードなどであればカードを取り出す前におこなう
ファイル操作	RfOpen	ファイルのオープン
	RfRead	ファイルの読み取り
	RfWrite	ファイルへの書き込み
	RfClose	Read/Write の終了
	RfSeek	読み書き位置の変更
	RfFlush	バッファデータ強制書込み
	RfGetSize	ファイルサイズを取得する
ディレクトリ操作	RfOpenDir	ディレクトリオープン
	RfReadDir	ディレクトリ読み取り
	RfCloseDir	ディレクトリ読み取りの終了
付加機能	RfMkDir	ディレクトリ作成
	RfDelFile	ファイル削除
	RfDelDir	ディレクトリ削除
	RfMove	ファイル、ディレクトリの移動
	RfGetDriveInfo	ドライブ情報を取得する
	RfFormat	フォーマット実行

表 2-3 構造体一覧

構造体名称	概 要
T_RF_FCB	ファイル管理ブロック
T_RF_DRIVE_INFO	ドライバ情報
T_RF_DIR_INFO	ディレクトリを読み取った内容
T_RF_TIME	時刻
T_RF_FSYS_VER	ファイルシステムバージョン
T_RF_DRV_VER	ドライババージョン

2.3 アプリケーションインターフェース

2.3.1 RfInitFileSystem

書式

```
unsigned short  RfInitFileSystem( void );
```

概要

ファイルシステムの初期化。

パラメータ

なし

戻り値

dfErrorCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解説

すべてのファイルシステム関数を実行する前に1回おこないます。

2.3.2 RfConclude

書式

```
unsigned short RfConclude( void );
```

概要

ファイルシステムの終了

パラメータ

なし

戻り値

正常終了時は RF_OK が返される。異常時はそれ以外。

解説

すべてのバッファ内容を物理デバイスに書き込みます。
その後 FAT1 を FAT2 へ複写します。
書き込みをおこなった場合は、カードを取り出す前に実行する必要があります。

2.3.3 RfOpen

書式

```
T_RF_FCB * RfOpen(          char * fileName,
                            unsigned short mode,
                            unsigned short * pErr
                            );
```

概　要

ファイルをオープンする

パラメータ

fileName

ファイル名

mode

オープンモード

表 2-4 オープンモード

RF_OPEN_READ	読み取り用 ファイルが存在しない時は RF_ERR_NOT_FOUND がエラーコードに設定される
RF_OPEN_WRITE_NEW	書き込み用で新規に作成する ファイルが存在する時は RF_ERR_ALREADY_EXIST が エラーコードに設定される。 ファイルが存在しない時は新規に作成する
RF_OPEN_WRITE_CREATE	書き込み用で常に生成する ファイルが存在する時は始めにサイズを 0 とする ファイルが存在しない時は新規に作成する
RF_OPEN_WRITE_OLD	すでに存在するファイルを書き込み用にオープンする ファイルが存在してもサイズは 0 にしない ファイルが存在しない時は RF_ERR_NOT_FOUND がエ ラーコードに設定される
RF_OPEN_OPT_SEQ	シーケンシャルアクセスに特化した最適化をおこなう。

上記モード を必要に応じて OR して設定します。

モード組み合わせ時の動作

RF_OPEN_READ | RF_OPEN_WRITE_NEW
存在するファイルを指定した際にはエラーとなります。
存在しないファイルを指定した際にはファイルを新規に作成します。
Write した後その部分を読むことができます。

RF_OPEN_READ | RF_OPEN_WRITE_CREATE
存在するファイルを指定した際にはサイズを 0 にしてから開始します。
存在しないファイルを指定した際には新規に作成します。
Write した後その部分を読むことができます。

RF_OPEN_READ | RF_OPEN_WRITE_OLD
存在するファイルを指定した際は始めに存在するサイズのまま開始します。
存在しないファイルを指定した際にはエラーとなります。

pErr

エラーコードを格納する領域へのポインタ
dfErrorCode.h に設定されたエラーコードが格納される。正常終了時は RF_OK。

戻り値

T_RF_FCB 構造体へのポインタ。オープンが正常におこなえなかった場合は NULL が返ります。

解　説

RfOpen()ではファイルアクセスの準備をおこないます。read のときはファイルフォルダまで読みます。新規ファイルの write のときはディレクトリエントリを作成します。

2.3.4 RfRead

書式

```
unsigned short RfRead(          T_RF_FCB * pFcb,
                                char * pBuff,
                                unsigned long * pLen
                                );
```

概要

ファイルの読みとり。

パラメータ

pFcb

RfOpen()で取得した FCB ポインタ。

pBuff

読み取ったデータを格納する領域へのポインタ。

pLen

読み取りたいデータ長が格納された領域へのポインタ。RfRead 終了後は実際に読み取った長さが格納される。

戻り値

RfErrorCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解説

ファイルの内容を読み取る時に実行します。ファイルの終端まで読み込んだ後に実行すると RF_ERR_EOF が戻り値として設定されます。
あらかじめ RfOpen() のモードに RF_OPEN_READ を指定して 実行しておく必要があります。
RfRead()を実行すると次の読み取り位置は読み取った長さぶん移動します。RfRead()を連続して実行することによりファイルの内容を順番に読み取ることができます。

2.3.5 RfWrite

書式

```
unsigned short RfWrite(         T_RF_FCB * pFcb,
                                char * pBuff,
                                unsigned long * pLen
                                );
```

概要

パラメータ

pFcb

オープンで取 得した FCB ポインタ

pBuff

書き込むデータ

pLen

書き込むデータの長さが格納された領域へのポインタ。書き込み終了時は実際に書き込まれた長さが格納される。

戻り値

RfErrorCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解説

RfWrite() を実行した後ファイル内の書き込み位置は、書き込んだデータの次の位置となります。RfWrite()を連続的におこなうことにより順次に内容を追加、更新できます。

2.3.6 RfClose

書式

```
unsigned short  RfClose(T_RF_FCB * pFcb);
```

概要

ファイル操作の終了。

パラメータ

pFcb

RfOpen()で取得した T_RF_FCB ポインタ。

戻り値

RfErrorCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解 説

ファイル操作開始前に RfOpen() を実行します。その後読み書きなどの操作を実行し、そのファイルに関する操作を終了するときに RfClose() を実行します。RfWrite() をおこなった ファイルに関してはバッファ内のデータがすべてファイルに書き込まれます。

2.3.7 RfSeek

書式

```
unsigned short RfSeek(          T_RF_FCB * pFcb,
                                long offset,
                                unsigned short origin
                                );
```

概要

ファイル内の現在位置を変更する

パラメータ

pFcb

オープンで取得した FCB ポインタ

offset

origin からの距離

origin

移動の開始点。以下の3つから選択する。

RF_SEEK_SET :ファイルの先頭
RF_SEEK_END :ファイルの終端
RF_SEEK_CUR :現在位置

戻り値

RfErrorCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解説

書込みまたは読み取り位置を移動します。
読み取りモードの場合はファイルサイズを越えた位置には移動できません。
書き込みモードでオープンしていてファイルサイズより後ろに設定した場合、ファイルサイズを拡大します。ただし、この場合内容は初期化されません。

2.3.8 RfFlush

書式

```
unsigned short dfFlush( T_RF_FCB * pFcb );
```

概要

読み込みバッファ内のデータを物理デバイスに書き込む

パラメータ

pFcb

オープンで取得した FCB ポインタ

戻り値

RfErrorCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解説

書き込み時はバッファ内のデータを物理デバイスに書き込みます。
読み取り時はバッファを空にします。次に読み込むと再度物理デバイスからデータを取得します。

2.3.9 RfGetSize

書式

```
unsigned short RfGetSize(       T_RF_FCB * pFcb,
                                unsigned long * pSize
                                );
```

概要

ファイルのサイズを取得する

パラメータ

pFcb

オープンで取得した FCB ポインタ

pSize

取得したサイズを格納する領域

戻り値

RfErrCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解説

ファイルのサイズを取得します。

2.3.10 RfOpenDir

書式

```
T_RF_FCB * RfOpenDir( char * fileName, unsigned short * pErr );
```

概要

読み取りのためにディレクトリをオープンする

パラメータ

fileName

ディレクトリ名称

pErr

エラーコードを格納する領域へのポインタ
dfErrorCode.h に設定されたエラーコードが格納される。正常終了時は RF_OK。
NULL_POINTER の時はエラー

戻り値

T_RF_FCB 構造体へのポインタ。オープンが正常におこなえなかった場合は NULL が返ります。

解説

ディレクトリの名称はフルパスで指定します。同時にオープンできるファイル数を超える場合は実行できません。
RfOpenDir()実行後、RfReadDir()でディレクトリの内容を1エントリずつ読み取ります。

2.3.11 RfReadDir

書式

```
unsigned short RfReadDir( T_RF_FCB * pFcb, T_RF_DIR_INFO * pDirInfo );
```

概要

ディレクトリの内容を1レコードずつ読み取ります

パラメータ

pFcb

RfOpenDir()で取得した T_RF_FCB ポインタ。

pDirInfo

読み取ったディレクトリエントリの内容を格納する構造体へのポインタ。

戻り値

RfErrCode.h に定義されたエラーコード
正常終了時は RF_OK が返ります。ディレクトリの最後まで読み取った後に RfReadDir()を実行すると RF_ERR_EOF が返ります。

解説

ディレクトリの名称はフルパスで指定します。同時にオープンできるファイル数を超える場合は実行できません。
RfOpenDir()実行後、RfReadDir()でディレクトリの内容を1エントリずつ読み取ります。

2.3.12 RfCloseDir

書式

```
unsigned short RfCloseDir( T_RF_FCB * pFcb );
```

概要

ディレクトリ操作の終了

パラメータ

pFcb

RfOpenDir()で取得した T_RF_FCB ポインタ。

戻り値

RfErrCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解説

ディレクトリの操作を完了し T_RF_FCB を開放します。

2.3.13 RfMkDir

書式

```
unsigned short RfMkdir( char * fileName );
```

概要

新規にディレクトリを作成する

パラメータ

fileName

ディレクトリ名称

戻り値

RfErrCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解説

ディレクトリの名称はフルパスで指定します。FCB を内部で1つ確保するので他のファイルをオープンしている際に RfMkDir を使用すると、同時にオープンしているファイル数として加算されます。同時にオープンできるファイル数を超える場合は実行できません。
ディレクトリを作る際には「.」と「..」ディレクトリも同時に作成します。

2.3.14 RfDelFile

書式

```
unsigned short RfDelFile( char * filename );
```

概要

ファイルを削除する

パラメータ

fileName

ファイル名称

戻り値

RfErrorCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解説

ファイル名称はフルパスで指定します。FCB を内 部で1つ確保するので他のファイルをオープンしている際に RfDelFile を使用すると、同時にオープンしているファイル数として加算されます。同時にオープンできるファイル数を超える場合は実行できません。

ファイル名にディレクトリを指定した場合は RF_ERR_NOT_DIRECTORY が返ります。ディレクトリを削除する際は RfDelDir を実行してください。

2.3.15 RfDelDir

書式

```
unsigned short RfDelDir( char * filename );
```

概要

ディレクトリを削除する

パラメータ

fileName

ディレクトリ名称

戻り値

RfErrCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解説

ディレクトリ名称はフルパスで指定します。

ディレクトリ内のサブディレクトリ、ファイルをすべて削除します。サブディレクトリの削除時は再帰的に呼び出しをおこなっています。そのためネストの深さにスタック使用量が比例して増えてしまうのでネスト数の制限を rsSettings.c の DEL_DIR_NEST に設定しています。このネスト数を超える場合は RF_ERR_DEL_DIR_NEST が返り、削除は中断されます。

FCB を内部で1つ確保するので他のファイルをオープンしている際に RfDelDir を使用すると、同時にオープンしているファイル数として加算されます。同時にオープンできるファイル数を超える場合は実行できません。

2.3.16 RfMove

書式

```
unsigned short dfMove(          char * dirName,
                                char * fileName,
                                );
```

概要

ファイルを別のディレクトリに移動する。

パラメータ

dirName

移動先のディレクトリ名称

fileName

元のファイル名称

戻り値

RfErrCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解説

ディレクトリエントリやファイルを別のディレク トリに移動します。
ただし、移動先が元のディレクトリのサブディレクトリである場合は RF_ERR_MOVE_POSTERITY となります。
ファイル名称、ディレクトリ名称はフルパスで指定します。FCB を内部で1つ確保するので他のファイルを オープンしている際に RfMoveFile を実行すると、同時にオープンしているファイル数として加算されます。同時にオープンできるファイル数を超える場合は実行できません。

2.3.17 RfSetReadOnly

書式

```
unsigned short dfMove(          char * fileName,
                                unsigned short add,
                                );
```

概要

ファイルの読み取り専用属性を操作する。

パラメータ

fileName

ファイル名

add

読み取り専用属性を付加するか否か

戻り値

RfErrCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解説

指定されたファイルの読み取り専用属性を変更します。add を TRUE にするとファイルを読み取り専用とします。add を FALSE にすると読み取り専用属性を外します。

2.3.18 RfGetDriveInfo

書式

```
unsigned short RfGetDriveInfo( T_RF_DRIVE_INFO * pDriveInfo );
```

概要

ドライブ情報を取得する

パラメータ

pDriveInfo

T_RF_DRIVE_INFO 構造体の領域へのポインタ。

戻り値

正常終了時は RF_OK が返される。異常時はそれ以外。

解説

ドライブの情報を取得します。

2.3.19 RfFormat

書式

```
unsigned short RfFormat( unsigned short option );
```

概要

フォーマットをおこなう

パラメータ

option

フォーマットのオプション。以下の3つのモードが指定できる。
RF_FORMAT_MODE_SD　:SD カードの標準規格でフォーマットをおこなう
RF_FORMAT_MODE_DOS　:DOS 形式でフォーマットをおこなう
RF_FORMAT_MODE_MAINT:現状のフォーマットに合わせてフォーマットをおこなう

さらにオプションとして以下の設定が可能。
RF_FORMAT_OPT_CLEAR　:データ領域をクリアする
RF_FORMAT_OPT_SAME_PARTITION:RF_FORMAT_MODE_DOS 時に元と同じパーティションでフォーマットをする際に指定する

RF_FORMAT_MODE_DOS でフォーマットをおこなう際には下位1バイトに BSR の位置指定が可能です。BSR 位置はセクタ位置を指定してください。RF_FORMAT_OPT_SAME_PARTITION オプションも BSR 位置も指定しない場合は MBR の無いフォーマットをおこないます。その際デバイスの先頭セクタは MBR ではなく BSR となります。

戻り値

RfErrCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解説

他のファイルが open されている場合は RF_ERR_FMT_THERE_IS_OPENED が返されフォーマットは実行されません。
RF_FORMAT_MODE_DOS の場合は1つのクラスタサイズが小さくなるように計算します。RF_FORMAT_MODE_DOS で BSR 位置指定のない場合は先頭セクタに BSR を配置します。
RF_FORMAT_MODE_SD(SD カードの規格)ではクラスタサイズは 1G までは、16384 バイト、2G では 32768 バイトとなります。また、BSR 位置を計算し MBR、BSR を毎回作成しなおします。

2.3.20 RfGetVerSion

書式

```
unsigned short RfGetVerSion( T_RF_FSYS_VER * fsysVer );
```

概要

ファイルシステムのバージョンを取得する

パラメータ

fsysVer

バージョン情報を格納する領域へのポインタ

戻り値

RfErrCode.h に定義されたエラーコード
正常終了時は RF_OK が返される。

解説

ファイルシステムのバージョンを取得します。

2.4 データ形式

2.4.1 T_RF_FCB

概要

RfOpen()、RfOpenDir()が成功した際に返すファイル管理ブロックへのポインタです。ファイルの識別子として RfRead()、RfWrite()などの関数に引き渡します。T_RF_FCB の内容はアプリケーションには公開しません。互換性を考慮せず変更することがあるのでアプリケーションは T_RF_FCB の内容を参照して処理をおこなってはいけません。また、更新しないでください。

2.4.2 T_RF_DRIVE_INFO

形式

```
typedef struct t_rf_drive_info
{
    unsigned char    fSystem;          /* ファイルシステム(FAT12/FAT16) */
    unsigned char    sectorPerCluster; /* クラスタのセクタ数            */
    unsigned short   bytesPerSector;   /* セクタバイト数                */
    unsigned long    bytesPerCluster;  /* クラスタのバイト数            */
    unsigned long    cardSize;         /* デバイス全体のバイト数        */
    unsigned long    bpbSector;        /* BPB の開始セクタ              */
    unsigned long    fatPoint;         /* FAT の開始アドレス            */
    unsigned long    fatSize;          /* 1FAT のバイト数               */
    unsigned short   fatNum;           /* FAT 数                        */
    unsigned short   clusterNum;       /* 全クラスタ数                  */
    unsigned long    rootPoint;        /* ROOT ディレクトリの開始アドレス*/
    unsigned short   rootEntryNum;     /* ROOT ディレクトリのエントリ数 */
    unsigned long    bodyPoint;        /* body の開始アドレス           */
    unsigned char    drvInit:1;        /* 物理的な初期化が完了した区分  */
    unsigned char    mbrInit:1;        /* MBR 取得できた区分            */
    unsigned char    bpbInit:1;        /* BPB 取得できた区分            */
} T_RF_DRIVE_INFO;
```

概 要

ディスクの物理的な情報です。RfGetDriveInfo で取得します。

メンバの説明

fSystem

ファイルシステム形式(FAT12/FAT16)。以下のいずれかの設定が格納される。
RF_F_SYSTEM_UNKNOWN
RF_F_SYSTEM_FAT12
RF_F_SYSTEM_FAT16

sectorPerCluster

クラスタのセクタ数。1クラスタのセクタ数が格納される。

bytesPerSector

セクタバイト数。1セクタのバイト数が格納される。

bytesPerCluster

クラスタのバイト数。bytesPerSector×sectorPerCluster の結果。

cardSize

デバイス全体のバイト数。

bpbSector

BPB の開始セクタ。

fatPoint

FAT の開始アドレス。カード先頭からのバイト単位の位置を示す。

fatSize

1FAT のバイト数。1つの FAT のバイト数。

fatNum

FAT 数。FAT の数。FAT12/FAT16 では通常2つの FAT が存在する。

clusterNum

全クラスタ数。データを格納する領域のクラスタ数。

rootPoint

ROOT ディレクトリのデバイス先頭からのバイト単位の位置。

rootEntryNum

ROOT ディレクトリのエントリ数。ROOT ディレクトリに格納されるファイルエントリの数。

bodyPoint

body の開始アドレス。先頭クラスタのデバイス先頭からのバイト単位の位置を示す。

drvInit

物理的な初期化が完了している区分。

mbrInit

MBR の初期化が完了している区分。

bpbInit

BPB の初期化が完了している区分。

2.4.3 T_RF_TIME

形式

```
typedef struct t_rf_time {
    unsigned short year;                    /* 年 */
    unsigned short month;                   /* 月 */
    unsigned short day;                     /* 日 */
    unsigned short hour;                    /* 時 */
    unsigned short min;                     /* 分 */
    unsigned short sec;                     /* 秒 */
    unsigned short mSec;                    /* ミリ秒 */
} T_RF_TIME;
```

概要

現在時刻取得関数が設定する時刻の型。

メンバの説明

year

西暦年。4桁で指定する。ただし、有効な値は 1981～2107 年。

month

月。1～12 が有効。

day

日。1～31 が有効。

hour

時。0～23 が有効。

min

分。0～59 が有効。

sec

秒。0～59 が有効。

mSec

ミリ秒。ミリ秒単位の時刻。 0～999 が有効。

2.4.4 T_RF_DIR_INFO

形式

```
typedef struct t_fr_dir_info {
    T_RF_TIME     timeCreate;           /* 作成日付、時刻     */
    T_RF_TIME     timeAccess;           /* 最終アクセス日付   */
    T_RF_TIME     timeUpdate;           /* 更新日付、時刻     */
    unsigned long size;                 /* ファイルサイズ     */
    unsigned char attribute;            /* 属性               */
    char          name[13];             /* ファイル名         */
} T_RF_DIR_INFO;
```

概要

ディレクトリ情報を格納する領域。RfReadDir()により取得した内容が格納される。

メンバの説明

timeCreate

作成日付時刻。ミリ秒単位の時刻まで使用されます。

timeAccess

最終アクセス日付。年月日までの日付のみが使用されます。

timeUpdate

更新日付、時刻。秒までの時刻が使用されます。ミリ秒単位の時刻は使用されません。
また、更新日付の秒単位の値は2秒単位で格納されているので偶数単位となります。

size

ファイルのサイズ。ディレクトリエントリの場合はサイズは格納されません。

attribute

属性。以下の値が or で格納されます。

DIR_ATTR_READ_ONLY	:読み取り専用
DIR_ATTR_HIDDEN	:隠しファイル
DIR_ATTR_SYSTEM	:システムファイル
DIR_ATTR_VOLUME_ID	:ボリューム ID
DIR_ATTR_DIRECTORY	:ディレクトリ
DIR_ATTR_ARCHIVE	:圧縮ファイル

name

ファイル名。

2.4.5 T_RF_FSYS_VER

形式

```
typedef struct t_rf_fsys_ver {
    unsigned short fsys;        /* ファイルシステムのバージョン */
    T_RF_DRV_VER drv;           /* ドライバのバージョン         */
} T_RF_FSYS_VER;
```

概要

バージョン情報を格納するデータ型。

メンバの説明

fsys

ファイルシステムのバージョン。

drv

ドライバのバージョン。

2.4.6 T_RF_DRV_VER

形式

```
typedef struct t_rf_drv_ver {
    unsigned short ver;                 /* ドライババージョン */
    const char * type;                  /* ドライバタイプ */
} T_RF_DRV_VER;
```

概要

ドライバのバージョン情報を格納するデータ型。

メンバの説明

ver

ドライバのバージョン。

type

ドライバの形式。
ここには文字列のポインタが格納されます。
ドライバを新規に作成する際はそのドライバの形式を表す文字列を格納するようにします。

2.5 設定ファイル

2.5.1 設定項目

以下は rfSettings.c の一部です。ファイルシステムの調整のために必要な設定値です。太字、斜体の部分をシステムに合わせて修正してください。

```
/* バッファサイズの指定 */
#define    GEN_BUFF_SIZE        ( 1024 )          /* バッファのサイズ */

/* 同時にオープンできるファイル数の指定    */
#define SAME_TIME_FILE_NUM      ( 2 )             /* 同時にオープンできるファイル数          */

/* RfDelDir のネスト数          */
#define DEL_DIR_NEST            ( 4 )             /* SdDelDir のネスト数          */
```

ファイルシステムが使用するバッファサイズ、同時にオープンできるファイル数、RfDelDir()のネスト数。
RAM の容量、重視する機能により調整をしてください。調整の考え方については「1.9.バッファサイズの考え方」を参照してください。

3 システム依存機能

ファイルシステムが使用する機能をシステムに合わせて実装してください。
現状必要なシステム依存関数はファイルの作成時刻などを取得するための機能のみです。
システム関数は rfSettings.c ファイルの rfSystemFunction に登録します。

3.1 システム関数一覧

表 3-1 システム関数一覧

分 類	関 数	概 要
時刻取得	getDate	現在時刻を取得する

3.2 システム関数リファレンス

3.2.1 時刻取得関数

書式

```
unsigned short getDate( T_RF_TIME * time );
```

概要

現在時刻取得関数。

パラメータ

time

T_RF_TIME 型の構造体へのポインタ。日付、時刻を格納する。

戻り値

RF_OK: 正常終了
RF_ERR_GET_TIME:時刻取得失敗

解説

ファイルの作成、更新時に日付、時刻を取得するためファイルシステムから実行される関数です。関数内では現在時刻を取得し time が指す領域に格納してください。関数名を getDate としていますが 関数の名称は規定していません。関数ポインタを rfSettings.c の rfSystemFunction 構造体に設定します。
正常終了時には RF_OK を返すようにしてください。エラー発生時は RF_ERR_GET_TIME を戻り値として返してください。

3.3 rfSettings.c 設定例

rfSettings.c にシステム関数を設定する例を示します。
この例は、Env¥Vcpp¥depend¥setting¥rfSettings.c です。

```
/*******************************************************/
/* システム関数を登録してください                      */
/*******************************************************/
#include "dateTime.h"

const T_RF_SYSTEM_FUNCTION rfSystemFunction =
{
        getDate,        /* 時刻取得     */
        NULL,           /* 予約         */
        NULL            /* 予約         */
};
```

4 ドライバ機能

ハードウェアに合わせたドライバ関数を実装してください。
このドライバ関数はハードウェアごとに実装する必要があります。
ドライバ関数は rfSettings.c の rfDriverInterface 構造体に関数ポインタを指定してください。ここで使用している関数名は説明のために仮に決めています。使いやすい関数名を指定してください。
各評価用パッケージではサンプルのドライバを実装しています。

4.1 ドライバ関数一覧

表 4-1 サポート関数一覧

説明の関数名	T_RF_DRIVER_INTERFACE のメンバ	概 要	必須機能
drvInitialize	initialize	ドライバの初期化をおこなう	○
drvWrite	write	1ブロックの書き込みをおこなう	×
drvWritePoll	writePoll	書き込みの進捗を確認する	×
drvFill	fill	1ブロックを同一の値で埋める	×
drvRead	read	1ブロックの読み取り。	○
drvReadpoll	readPoll	読み取りの進捗確認	×
drvGetBlockSize	getBlockSize	ブロックサイズを取得する。	○
drvGetCapacity	getCapacity	メディアの容量を取得する。	○
drvConclude	conclude	ドライバの終了処理	×
drvGetVer	getVer	バージョンを取得する。	×

4.2 ドライバ関数リファレンス

4.2.1 ドライバ初期化

書式

```
unsigned short drvInitialize( void );
```

概要

ドライバ初期化。

パラメータ

なし。

戻り値

RF_OK:正常終了
その他:ユーザーが定義したエラーコード

解説

ファイルシステムの初期化時にファイルシステムから実行される関数です。ドライバの初期化では drvWrite()や drvRead()を実行できる状態にしてください。

4.2.2 物理理的な書き込み

書式

```
unsigned short drvWrite(        unsigned long addr,
                                char * buff,
                                unsigned short size
                                );
```

概要

物理的な書き込みをおこなう。

パラメータ

addr

書き込み先のアドレス。セクタの区切りに合っていなければならない。

buff

書き込むデータが格納されている領域へのポインタ。

len

書き込むデータ長。ブロック(セクタ)サイズのみが有効です。

戻り値

RF_OK:正常終了
その他:ユーザーが定義したエラーコード

解説

指定された領域のデータを指定されたアドレスに書き込みます。「物理的な書き込みの進捗確認」機能を実装しない場合は書き込みが完了するまで関数は終了してはいけません。書き込みを別のタスクや割り込みを使用して非同期で動作させる場合には進捗を確認する「物理的な書き込みの進捗確認」機能を実装してください。
「物理的な書き込み」はブロック(セクタ)サイズ単位でしかおこなえません。

4.2.3 物理理的な書き込みの進捗確認

書式

```
unsigned short drvWritePoll( unsigned short * len );
```

概要

物理的な書き込みの進捗を確認する。

パラメータ

len

書き込みがおこなえたサイズを格納する領域へのポインタ。

戻り値

RF_OK:正常終了
その他:ユーザーが定義したエラーコード

解説

非同期でおこなう「物理的な書き込み」と「セクタクリア」の進捗を確認します。「物理的な書き込み」と「セクタクリア」は同じ仕組みで実装しなければなりません。「物理的な書き込み」を非同期で実装する場合は「セクタクリア」も同様の仕組みで実装してください。ファイルシステムはバッファ内の書き込みが済んでいる部分を次の write のために使用します。
「物理的な書き込み」、「セクタクリア」が書き込みを終了するまで関数を抜けない仕組みで実装されている場合は、この機能を実装する必要はありません。ただし、この場合は「RF_OPEN_OPT_SEQ」オプションは有効に機能しません。

4.2.4 セクタクリア

書式

```
unsigned short drvFill(         unsigned long address,
                                char writeData
                                );
```

概要

1つのセクタを単一の値でクリアする。

パラメータ

address

クリアするセクタの先頭アドレス。

writeData

書き込む値。

戻り値

RF_OK:正常終了
その他:ユーザーが定義したエラーコード

解説

指定されたセクタを writeData で埋めます。「物理的な書き込み」と同様な仕組みで実装する必要があります。非同期の書き込み方式で実装する際には「物理的な書き込みの進捗確認」機能で書き込みの進捗を取得できるようにしてください。
「物理的な書き込みの進捗確認」機能を実装しない場合は当関数は書き込みが終了するまで関数を抜けてはいけません。

4.2.5 物理的な読み取り

書式

```
unsigned short drvRead(         unsigned long addr,
                                char * buff,
                                unsigned short len
                                );
```

概要

物理的な読み取りをおこなう。

パラメータ

addr

読み取るアドレス。

buff

読み取ったデータを格納する領域へのポインタ。

len

読み取るデータ長。

戻り値

RF_OK:正常終了
その他:ユーザーが定義したエラーコード

解説

指定されたアドレスのデータを読み取って、buff に格納してください。セクタサイズより少ない読み取り機能をサポートすることにより少ないバッファでのファイルシステムの実装が可能となります。ファイルシステムはバッファサイズがブロックサイズより少ない場合に、そのバッファサイズのデータを要求します。

物理デバイスがセクタサイズ未満の読み取りをサポートできなかったり、ドライバ関数内でその仕組みを実現できない場合はエラーの戻り値を返すようにしてください。ファイルシステムはリード時にそのエラーコードをアプリケーションに返します。

4.2.6 物理的な読み取りの進捗確認

書式

```
unsigned short drvReadPoll( unsigned long * len );
```

概要

物理的な読み取りをおこなう。

パラメータ

len

読み取りが完了した長さを格納する領域。

戻り値

RF_OK:正常終了
その他:ユーザーが定義したエラーコード

解説

「物理的な読み取り」を別なタスクや割り込みを使用して非同期に実装する場合、その進捗状況を取得します。ファイルシステムはバッファサイズ分のデータ読み取りを待つことなく読み取りが完了した部分のデータを使用します。

「RF_OPEN_OPT_SEQ」オプションを有効にするためには当関数の実装が必要です。また、その際には「物理的な読み取り」関数は読み取りを起動したら終了を待つことなく関数を抜けるようにしてください。

「物理的な読み取り」が要求されたデータのすべてを読み終わるまで関数を抜けない仕組みで実装されている場合は、当関数を実装する必要はありません。ただし、この場合は「RF_OPEN_OPT_SEQ」オプションは有効に機能しません。

4.2.7 セクタサイズを取得する

書式

```
unsigned short drvGetBlockSize( void );
```

概要

セクタサイズを取得する

パラメータ

なし

戻り値

1セクタのバイト数

解説

セクタのバイト数を取得してください。

4.2.8 物理デバイスのサイズを取得する

書式

```
unsigned long drvGetCapacity( void );
```

概要

物理的なデバイスのサイズを取得する

パラメータ

なし

戻り値

デバイスサイズ

解説

デバイスのバイト数を取得してください。

4.2.9 ドライバ終了

書式

```
unsigned short DrvConclude( void );
```

概要

ドライバの終了

パラメータ

なし

戻り値

RF_OK:正常終了
その他:ユーザーが指定したエラーコード

解説

ドライバの終了処理を実装してください。当関数実行後はデバイスを取り外せるようにしてください。特に必要がなければ当関数は実装する必要はありません。その場合は、rfDriverInterface 構造体に NULL を指定してください。

4.2.10 バージョン情報取得

書式

```
unsigned short drvGetVersion( T_RF_DRV_VER * ver );
```

概要

ドライバのバージョン情報を取得する

パラメータ

ver

バージョン情報を格納する領域へのポインタ。

戻り値

RF_OK:正常終了
その他:ユーザーが指定したエラーコード

解説

ドライバのバージョン、デバイス区分、ハードウェアバージョンを ver で指定された領域に設定してください。

4.3 rfSettings.c 設定例

rfSettings.c ドライバ関数を設定する例を示します。
この例は、Env¥AkiH8DMA¥setting¥rfSettings.c です。

```
/*****************************************************************/
/*ドライバ関数を登録してください                                 */
/*****************************************************************/
#include "DrvSdCard.h"

const T_RF_DRIVER_INTERFACE rfDriverInterface =
{
        InitSdCard,         /* ドライバ初期化                  */
        WriteSdPhysical,    /* 物理的な書き込み                */
        WriteSdPoll,        /* 書き込みの進捗問い合わせ        */
        FillSdOneBlock,     /* セクタクリア                    */
        ReadSdPhysical,     /* 読み取りの開始                  */
        ReadSdPoll,         /* 読み取りの進捗問い合わせ        */
        GetSdBlockSize,     /* セクタサイズを取得する          */
        GetSdCardSize,      /* 物理デバイスのサイズを取得する*/
        NULL,               /* ドライバ終了(省略可)            */
        GetSdDrvVer         /* バージョン情報取得              */
};
```

図 4-1 ドライバ関数登録

5 テストプログラム

動作環境に依存しない、ファイルシステムを対象としたテストプログラムを動かせるようにしています。テストプログラム自体とテストプログラムを動作させるために必要な機能のインタフェースについて説明します。

また、各動作環境ごとに固有のテストプログラムもありますがそれらについては各動作環境の説明でおこないます。

5.1 テスト機能ソースファイル

5.1.1 テスト機能一覧

ファイルシステムをテストするためのテスト機能です。ファイルの一覧とテスト機能の概要を示します。

表 5-1 テスト機能一覧

ソースファイル名	関数	概要
TstDefine.h		テスト機能を定義したファイル
TstDelDir.c	testDelDir	サブディレクトリのあるディレクトリ削除 ルートディレクトリ削除ができない
TstDelete.c	testDelete	サブディレクトのファイル、ルートディレクトリのファイルを削除する
TstFlush.c	TestFlush	Write を実行し flush 前後のバッファ内容を比較する。
TstFormat.c	TestFormat	以下のフォーマットを実行する。 ・DOS モード、BSP 位置指定 ・DOS モード、BSP 位置指定なし ・SD カードモード ・現状フォーマットのままフォーマット
TstInitial.c	TestInitial	初期化機能の有効性確認
TstMkDir.c	TestMkDir	2階層のディレクトリを作り、そこにファイルを書き込む
TstMove.c	testMove	ファイルの移動テスト ・ルート → 5階層下 ・1階層下 → 5階層下 ・ファイルが見つからないエラー ・2階層下 → 5階層下 ・5階層下 → ルート ・5階層下 → 1階層下 ・パス名に ¥が付加されるパターン
TstMulti.c	TestMulti	複数ファイルのオープン。同時にオープンできるファイル数以上のファイルをオープンしてエラーとなることのテスト。
TstReadDir.c	TestReadDir	ディレクトリの読み取りテスト
TstReadOnly.c	TestReadOnly	読み取り専用ファイルを書き込みモードでオープンする
TstSeek.c	TestSeek	Write モードでオープンして seek によりファイルサイズが拡大することの確認、クラスタ境界を越えて拡大し問題のないことを確認する。
TstSelect.c	TestSelect	テストメニューを表示し実行するテストプログラムを選択、実行する
TstWriteExist.c	TestOverWrite	存在するファイルの上書きテスト
TstWriteRead.c	TestWriteRead	読み取りのテスト

5.1.2 テストのための汎用機能

テストのための汎用機能です。効率よくテストコードを書くための機能です。テスト機能から使用されます。

表 5-2 テストのための汎用機能

ソースファイル名	関数	概要
TraceOut.h		デバッグ用入出力関数のインタフェース定義。デバッグ用判定、表示整形マクロ定義。
TstGeneral.c(.h)	TestMenu hexToChar numToChar GetErrCodeChar TestReady TestConclude dispCardInfo TstVersion	テスト用汎用機能 ・メニュー表示 ・数値の hex 表示 ・数値の 10 進数表示 ・エラーコード文字列取得 ・テスト準備 ・テスト終了 ・デバイス情報表示 ・バージョン情報表示
TstWriteFunc.c(.h)	TestWriteMany TestWriteLarge TestWriteSmallFile	書き込みの汎用機能をまとめたファイル ・沢山のファイルを作成する ・サイズの大きなファイルを作成する ・小さなファイルを作成する
TstWriteVal.c(.h)		書き込み用データ定義

5.1.3 テスト用比較マクロ

TraceOut.h に定義されているテスト用の結果比較とその結果を整形して表示するためのマクロです。

表 5-3 テスト用比較マクロ

実行形式	概要
CHK_ERR_CODE(code, cmp, str);	code と cmp を比較し等しい場合は"OK"と str、異なる場合には code をファイルシステムのエラーコードとして str とともに文字列で表示する。
CHK_CMP_HEX(code, cmp, str);	code と cmp を比較し同じ場合は"OK"と str、異なる場合はそれぞれの値と str で指定した文字列を hex 形式で表示する。
CHK_CMP_DEC(code, cmp, str);	CHK_CMP_HEX()と同様ですが符号付き 10 進数整数として表示する。
CHK_CMP_STR(dist, src, str)	文字列同士を比較し、異なる場合は文字列を表示する。
CHK_CMP_MEM(dist, src, len, str)	len の長さのメモリを比較する。等しい場合は"OK"と str、異なる場合は"NG"と str を表示する

5.2 テスト用入出力機能

5.2.1 テスト用入出力機能一覧

TEST フォルダに TraceOut.h があり、このインタフェースに合わせた機能を動作環境ごとに実装します。

表 5-4 テスト用入出力機能

関数名	概要
traceOut	デバッグ結果の出力
operationOut	操作指示の出力
operationIn	操作指示の入力

5.2.2 traceOut

書式

```
void traceOut( const char * pFormat, ... );
```

概要

フォーマット付きデバッグ出力

パラメータ

pFormat

フォーマット書式

…

フォーマットに応じたデータ

戻り値

なし

解説

デバッグ環境に応じて実装してください。
テスト結果を出力する目的で使用します。
AkiH8 環境ではシリアル出力をおこないます。Windowss シミュレーションでは TRACE 出力をおこないます。

5.2.3 operationOut

書式

```
void operationOut( const char * pFormat, ... );
```

概要

フォーマット付き操作指示出力

パラメータ

pFormat

フォーマット書式

…

フォーマットに応じたデータ

戻り値

なし

解説

traceOut と異なる出力先を設定できるようにしています。
テスト指示のための表示に使用します。
AkiH8 環境ではシリアル出力をおこないます。Windows シミュレーションでは標準出力に表示します。

5.2.4 operationIn

書式

```
unsigned char operationIn( void );
```

概要

operationOut に対応した入力

パラメータ

なし

戻り値

入力した文字

解説

操作指示のための入力目的で使用します。
AkiH8 環境ではシリアルからの入力。Windows シミュレーションは標準入力からの入力をおこないます。

6 ファイルシステム評価パッケージ

すぐに動作確認可能なパッケージを複数用意しています。それらのパッケージについて説明します。

6.1 RsfAkiH8SCI

6.1.1 概要

AKI-H8 ボードで動作させるための環境です。デバッグ用の入出力をシリアルポートにおこなっているためパソコンとシリアルケーブルで接続し動作確認をおこなうことができます。パソコン側はターミナルソフトを使用します。

6.1.2 ビルド環境の構築

AKI-H8 の開発環境がインストールされていなければなりません。

あらかじめ環境変数の PATH に AKI-H8 開発環境のコンパイラやリンカのあるフォルダ(AKI-H8¥bin)を設定しておく必要があります。

AKI-H8 が提供する C38HAB.LIB を SRC¥Env¥AkiH8SCI フォルダにコピーします。

make.bat をテキストエディタで開き「set DirTool=%root%..¥h8c¥」の「%root%..¥h8c¥」部分を AKI-H8 開発環境のインクルードファイルのパス(AKI-H8¥h8c)に変更します。

make.bat を実行するとコンパイル、リンクが実行されます。

ビルドが成功すると Obj フォルダに RfTestMain.mot ファイルが生成されます。

日本語のディレクトリ名が入っていると正常に動作しないので注意してください。

6.1.3 フォルダ構成

パッケージのフォルダ構成は以下のようになっています。

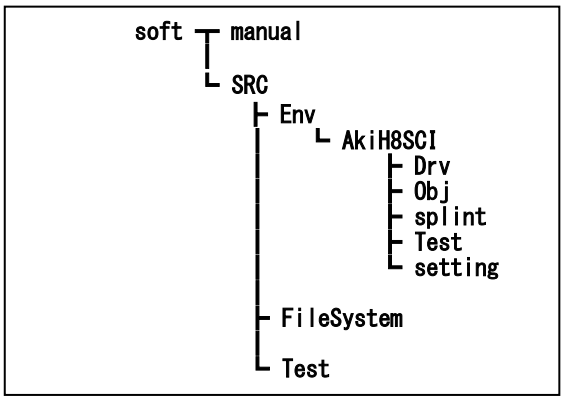

図 6-1 RsfAkiH8SCI フォルダ構成

6.1.4 ファイル一覧

RsfAkiH8SCI パッケージ固有のファイル一覧を示します。

表 6-1 RsfAkiH8SCI ファイル一覧

フォルダ	ファイル名	概要
manual	rsfManual.pdf	ファイルシステムのマニュアル。
SRC		ソースコード
Env¥AkiH8SCI	link.sub	リンカの設定ファイル
	make.bat	プロジェクトビルドのバッチファイル
	RfTestMain.c	メイン関数
	TraceOut.c	テスト用入出力
	VectorTbl.asm	割り込みベクタテーブル
Drv	DrvCpuXtal.h	クロック定義
	DrvErrCode.h	ドライバのエラーコード定義ファイル
	DrvPort.c(.h)	ポートドライバ
	DrvSdCard.c(.h)	SD カードドライバ
	DrvSdRegister.h	SD カードレジスタの定義ファイル
	DrvSdSciItu.c(.h)	SCI だけで動作する SPI ドライバ
Obj	RfTestMain.MOT	モトローラ S フォーマット形式の実行ファイル
	RfTestMain.map	マップファイル
Setting	rfSettings.c	ファイルシステムの設定ファイル
Splint	RfTestMainSplint.pdf	Splint の出力結果を表形式にまとめ、対応しない項目についてその理由を記述したもの
Test	_DebugFunc.asm	アセンブラデバッグ機能
	DebugFunc.c(.h)	C 言語デバッグ機能
	TstOrgDelDir.c	DelDir のスタック使用量調査機能
	TstOrgInitialConfirm.c	ファイル入出力のスタック使用量調査
	TstOrgSelect.c	テスト機能選択メニュー
	TstOrgErrCode.c(.h)	ドライバエラーコード表示機能
	TstOrgPerformance.c	パフォーマンス調査機能
FileSystem	…	ファイルシステム。詳細は「2.ファイルシステム」を参照してください。
TEST	…	ファイルシステムのテストプログラム。詳細は「5.テストプログラム」を参照してください。

Obj フォルダには mot 形式の実行ファイルが格納されています。CPU に書き込めばすぐに動作確認がおこなえます。

6.1.5 ドライバ(Drv フォルダ)

SPI モードで SD カードにアクセスするためのドライバです。
SPI 通信では割り込みを使用せず SCI のみを使用しています。そのため処理の流れを理解するのに適したドライバです。
RsfAkiH8SCI パッケージのドライバについては「7 サンプル SD カードドライバ」に詳細を記述しています。

6.1.6 実行ファイル(Obj フォルダ)

ビルドの中間ファイルと実行ファイルを置くフォルダです。RfTestMain.MOT は ROM に書き込む実行ファイルで、RfTestMain.map はリンカの出力リストであり、実行ファイルのメモリ配置を表します。

6.1.7 設定(setting フォルダ)

ファイルシステムの設定を記述したファイルが格納されます。詳細は以下の各章を参照してください。

- 「2.5.設定ファイル」
- 「3.3.rfSettings.c 設定例」
- 「4.3.rfSettings.c 設定例」

6.1.8 静的解析結果(splint フォルダ)

splint でファイルシステム、ドライバ関数の静的解析をおこない、対応すべき指摘事項について対応しています。ここには対応しないと判断した指摘事項についての理由を明示した表を格納しています。

6.1.9 テストプログラム(Test フォルダ)

RsfAkiH8SCI パッケージのテストプログラムとデバッグ用低水準入出力です。機能の概要を説明します。

表 6-2 AkiH8SCI テスト機能

ソースファイル名	関数	概要
_DebugFunc.asm	_GetSp	現在のスタックポインタ位置取得。スタック使用量調査で使用する。
DebugFunc.c(.h)		固有のデバッグ機能
	InitLED	・LED ポート初期化
	winkLoop	・LED 点滅
	InitDebugOut	・デバッグ入出力の初期化
	debugTxt	・文字列出力
	debugHexOut	・数値の hex 出力
	debugNum	・数値の 10 進数出力
	debugAChar	・1文字出力
	debugGetChar	・1文字入力
	debugTimeOut	・時間出力
	dumpOut	・メモリダンプ出力
	InitTimeCount	・時間計測の初期化
	StartTimeCount	・時間計測開始
	InterruptTimeCount	・時間計測の割り込み機能
	GetTimeCount	・計測時間取得
	StopTimeCount	・時間計測停止
	checkFreeSp	・未使用スタック位置取得
	fillSp	・checkFreeSp の初期化
TstOrgDelDir.c	SpChkDelDir	ディレクトリ削除を実行しスタック使用量を調査する
TstOrgDriver.c	TstOrgDriver	ドライバの単体での動作確認をおこなう
TstOrgErrCode.c(.h)	GetOrgErrCodeChar	固有のエラーコードを文字列変換する機能
TstOrgInitialConfirm.c	TstOrgInitialConfirm	基本的なファイルアクセスをおこないスタック使用量を調査する。
TstOrgPerformance.c	TstOrgPerformance	パフォーマンス調査
TstOrgSelect.c	TestOrgSelect	AkiH8SCI 固有のテストメニューの表示、実行。

6.1.10 テストメニュー

```
┬ 1. file system common test.
│   ├ 1. Test format.
│   ├ 2. Test seek.
│   ├ 3. Test read only.
│   ├ 4. Test move file.
│   ├ 5. Test make directory.
│   ├ 6. Test del directory.
│   ├ 7. Test delete.
│   ├ 8. Test write & read.
│   ├ 9. Test over write.
│   ├ a. Test multi file.
│   ├ b. Test flush.
│   ├ c. Test read directory.
│   ├ x. all Test
│   └ q. quit
│
├ 2. Inspection depending on environment.
│   ├ 1. Stack consumtion DelDir.
│   ├ 2. Stack consumtion Standard Function.
│   ├ 3. Driver Test.
│   │   ├ 1. Read  Function.
│   │   ├ 2. Read  Detail.
│   │   ├ 3. Write Function.
│   │   ├ 4. Fill  Function.
│   │   └ q. quit
│   ├ 4. Performance Test.
│   │   ├ 1. ready.
│   │   ├ 2. Make TestData.
│   │   ├ 3. Read synchronous.
│   │   ├ 4. Write New.
│   │   ├ 5. Write Old.
│   │   └ q. quit
│   └ q. quit
│
└ q. quit
```

図 6-2 AkiH8SCI テストメニュー

6.1.11 スタック使用量について

スタック領域の調査方法について説明します。スタック領域の先頭から現在の SP レジスタの指す位置までを 0xff で埋めます。目的の機能の実行後、スタック領域の先頭から 0xff でなくなる位置を探します。そこから実行前の SP レジスタの位置を引いて使用量を計算します。

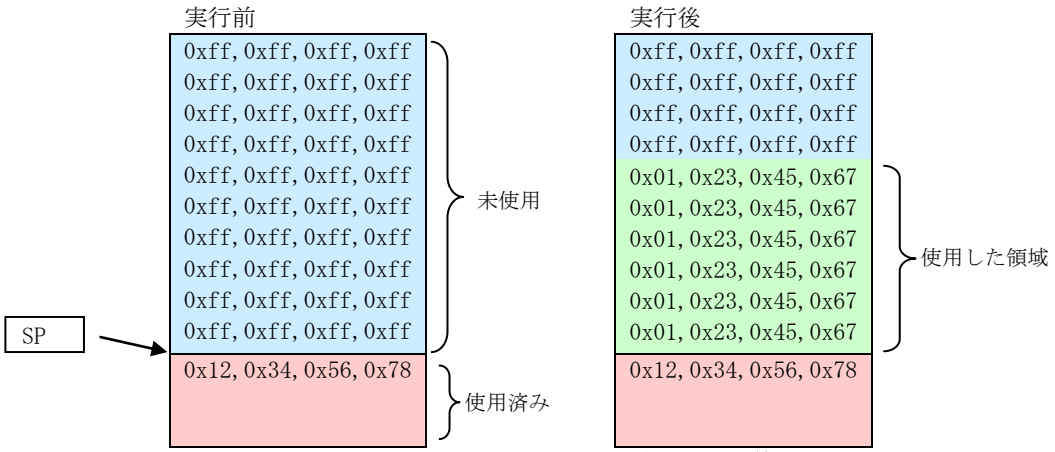

図 6-3 スタック使用量の計算

AkiH8SCI 動作環境で計測したスタック使用量調査の結果です。

表 6-3 AkiH8SCI スタック使用量

機能	解説	スタック使用量
MkDir()	MkDir() 関数でディレクトリを作る際のスタック領域の使用量です	468バイト
DelDir() 4階層	DelDir() 関数で4階層のサブディレクトリを持つディレクトリ削除時のスタック領域の使用量です。	646バイト
DelDir() 1階層	DelDir() 関数でサブディレクトリを持たないディレクトリを削除した際のスタック領域使用量です。	502バイト
RfOpen()	新規作成モードでファイルをオープンした際のスタック領域使用量です。	496バイト
RfWrite()	10 バイトのデータを書き込んだ際のスタック領域使用量です。	400バイト
RfClose()	上記 RfWrite() 実行後の RfClose() 実行後のスタック領域使用量です。	440バイト

6.1.12 パフォーマンスについて

パフォーマンスは状況により変動します。計測したパフォーマンスは性能を保証するものではありません。標準的な一例としてご理解ください。

パフォーマンス計測をおこなう際には、メディアのフォーマット、テストデータの作成をあらかじめおこなう必要があります。

[2. Inspection depending on environment.] – [4. Performance Test.] - [1. ready.] でメディアのフォーマット、[2. Make TestData.]でテストデータを作成します。

RfRead

16MB と 2GB の SD カードで計測した結果です。64K バイトのファイルに対して 2 バイトずつ read をおこないます。その際の RfRead()関数の実行時間を測定し処理時間の分布を表にしています。各枠の数字は回数を示します。

表 6-4 AkiH8SCI RfRead()のパフォーマンス

	～250μ s	250μ s～4ms	4～8ms	8～16ms	MIN	MAX
16M (Panasonic RP-SD016B)	32640	0	127	1	197us (65534byte)	14.691 us (16384byte)
2G CLASS6 (Panasonic RP-SDV02G)	32640	0	127	1	197us (65534byte)	14.395 us (32768byte)

最大の時間を消費した 8ms～16ms の1回の時間は 14,690μs で 16384 バイト目を書き込んだ RfRead()で発生しています。2G の場合は 32768byte の後の RfRead()です。これはどちらも 1 クラスタ分のデータを読み終えて次のクラスタを探すために FAT を読む必要がここで発生したためです。

2G の SD カードは Class6(転送速度 6Mbps を保証する)ですがハードウェアの制約により 2Mbps でしか通信できていないためメディアの性能を生かし切れていません。このハードウェアではこれが限界です。

4ms～8ms のところに 127 回の RfRead()が入っています。ここは物理的な read が発生したか所です。65536 バイトのデータなので 512 バイトのセクタサイズで割ると 128 回の物理的な read が発生することが分かります。差の 1 回は 8ms-16ms のところに入っています。最大時間の回には FAT とデータ部分の 2 回の物理的な read が発生しています。

RfWrite

新規ファイル作成での RfWrite()のパフォーマンスを測定しています。バッファサイズを 1024 バイトで実施した結果です。

表 6-5 AkiH8SCI 新規 RfWrite のパフォーマンス

	～250μ s	250μ s～8ms	8～16ms	16ms～	MIN	MAX
16M (Panasonic RP-SD016B)	32640	0	0	128	237us (2byte)	27,962us (0byte)
2G CLASS6 (Panasonic RP-SDV02G)	32640	0	127	1	237us (2byte)	18,837us (0byte)

RfWrite では初回に空きクラスタを探すため FAT を参照します。RfRead()の際と同様で 128 回の物理的なアクセスが発生します。セクタサイズの 512 バイト毎に物理的なアクセスをおこなっています。

物理的な書き込みでは SD カードにデータを転送した後にカード内部で書き込みの時間が確保されます。これを busy 時間といいますが、その間次の書き込みがおこなえなくなります。AkiH8SCI のドライバはデータ転送後の busy 時間が終了するまで関数を抜けません。Busy 時間はデバイスによって異なります。

上書き RfWrite

表 6-6 AkiH8SCI 上書き RfWrite のパフォーマンス

	～250μ s	250μ s～8ms	8～16ms	16ms～	MIN	MAX
16M (Panasonic RP-SD016B)	32640	1	0	127	246us (65534byte)	34,221us (16384byte)
2G CLASS6 (Panasonic RP-SDV02G)	32640	1	0	127	246us (65534byte)	25,360us (32768byte)

250μs に収まっている 32640 回の write ですが最小の実行時間が 246μs であることから 250μs にきわめて近い時間で実行されていると考えられます。

上書きの write では初回からクラスタの位置は決まっていますがデータ部分を読み取ってから書き込みをおこないます。

6.1.13 実行

パソコンと AKI-H8 ボードをシリアルケーブルで接続します。
フラッシュライタで RfTestMain.mot を AKI-H8 ボードのフラッシュ ROM に書き込みます。書き込み方法は AKI-H8 の説明書を参照してください。
パソコン側はターミナルソフトを立ち上げ通信設定を以下のようにおこなってください。

表 6-7 シリアル設定

項目	内容
ポート	シリアルケーブルを接続したポート
ボーレート	38400bps
データ長	8 ビット
パリティ	なし
ストップビット	1ビット
フロー制御	なし

ターミナルソフトを通信可能な状態にして AKI-H8 の電源を入れてください。
バージョンとテストの選択肢が表示されます。
実行したいテストケースの番号を入力します。

```
************************************************************
AkiH8SCI:System version. Build(Mar  6 2014 22:26:28)
************************************************************
FileSystem version = 0102
Driver version     = 0002
Driver type        = AKI-H8 SCI.

************************************************************
AkiH8SCI:Use only SCI driver. Not use interrupt.
************************************************************
1. file system common test.
2. Inspection depending on environment.
q. quit
==>
```

図 6-4 ターミナルソフト起動時の表示

「1. file system common test.」を選択すると初期化機能のテストと基本的なアクセステストをおこない、共通のテストメニューが表示されます。

```
==> 1

************************************************************
Test Initialize.
************************************************************
 OK open[NG case]
 OK RfInitFileSystem()
 OK open
 OK conclude
 OK open[NG case]

************************************************************
AkiH8SCI:common test.
************************************************************
1. Test format.
2. Test seek.
3. Test Read only.
4. Test move file.
5. Test make directory.
6. Test del directory.
7. Test delete.
8. Test write & read.
9. Test over write.
a. Test multi file.
b. Test flush.
c. Test read directory.
x. All test.
q. quit
==>
```

図 6-5 共通テストメニュー

全体を実行する「x」を選択したテスト結果の一部を示します。

```
************************************************************
Test format.
************************************************************
 OK RfInitFileSystem()

1.=====================================
   ==== RfFormat( RF_FORMAT_MODE_DOS | 0x39 )
 OK RfFormat()
 OK RfInitFileSystem()
==== card info ====
capacity(14909440)
bpbPoint(0x00007200)
bpbPoint(0x00000039)sect
fat size(5632)
rootPoint(0x0000a200)
bodyPoint(0x0000e200)
clusterNum(3625)
cluster size(4096)
sector/cluster(8)
FAT12

：省略


************************************************************
Test seek.
************************************************************
 OK RfInitFileSystem()
 OK RfFormat()
 OK mkdir
 OK open
 OK seek
 OK seek
 OK fileSize
 OK write
 OK write
 OK write
 OK write
 OK fileSize
 OK seek
 OK seek
 OK fileSize
 OK close
 OK open
 OK open
 OK write
 OK seek
 OK close
 OK open
 OK seek
 OK seek error case.
 OK close
 OK open RF_OPEN_OPT_SEQ
 OK write1
 OK write2
 OK seek to top.
 OK read
 OK read value.+0(16)
 OK seek to border.
 OK write3. +510(2)
 OK read value.+512(16)
 OK seek to border.
 OK read value. +510(2)
 OK close

************************************************************
Test Read Only.
************************************************************
 OK RfInitFileSystem()
 OK RfFormat()
 OK RfOpen()
 OK write()
 OK diff before write and after.
 OK RfClose()
 OK RfSetReadOnly()
 OK RfOpen(RF_OPEN_WRITE_OLD) [NG] RF_ERR_READ_ONLY
 OK RfOpen(RF_OPEN_READ)
 OK RfRead()
 OK read value.
```

図 6-6 テスト結果一部

成功すると行の先頭に「OK」、失敗すると「!NG!!」が表示されコメントが続きます。すべてのテストケースが「OK」となっていれば正常に動作しています。

6.2 RsfAkiH8DMA

6.2.1 概要

ビルド環境、実行方法などは AkiH8SCI と同様です。

DMA を使用して割り込みの発生回数を減らすとともに、順次処理に特化した最適化オプション(RF_OPEN_OPT_SEQ)を指定することにより次に必要となるデータをバッファ内に先に取り込みパフォーマンスを向上することができます。

6.2.2 ビルド環境の構築

AkiH8SCI のビルド環境構築を参照してください。

6.2.3 フォルダ構成

パッケージのフォルダ構成は以下のようになっています。

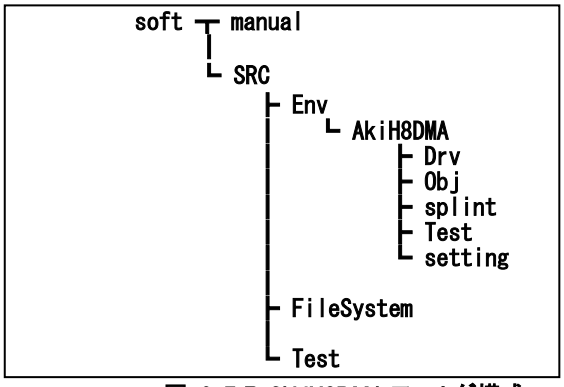

図 6-7 RsfAkiH8DMA フォルダ構成

6.2.4 ファイル一覧

AkiH8DMA の動作環境固有のファイル一覧を示します。ほとんど AkiH8SCI と同様です。

表 6-8 AkiH8DMA ファイル一覧

フォルダ	ファイル名	概要
manual	rsfManual.pdf	ファイルシステムのマニュアル。
SRC		ソースコード
Env¥AkiH8DMA	link.sub	リンカの設定ファイル
	make.bat	プロジェクトビルドのバッチファイル
	RfTestMain.c	メイン関数
	TraceOut.c	テスト用入出力
	VectorTbl.asm	割り込みベクタテーブル
Drv	DrvSdRegister.h	SD カードレジスタの定義ファイル
	DrvErrCode.h	ドライバのエラーコード定義ファイル
	DrvSdCard.c(.h)	SD カードドライバ
	DrvPort.c(.h)	ポートドライバ
	DrvSdDmaSciItu.c(.h)	SCI と DMA を使用する SPI ドライバ
	DrvCpuXtal.h	クロック定義
Obj	RfTestMain.map	マップファイル
	RfTestMain.MOT	モトローラ S フォーマット形式の実行ファイル
Setting	rfSettings.c	ファイルシステムの設定ファイル
Splint	RfTestMainSplint.pdf	Splint の出力結果を表形式にまとめ、対応しない項目についてその理由を記述したもの
Test	TstOrgSelect.c	テスト機能選択メニュー
	TstOrgDelDir.c	DelDir のスタック使用量調査機能
	TstOrgInitialConfirm.c	ファイル入出力のスタック使用量調査
	TstOrgDriver.c	ドライバ機能テスト
	TstOrgPerformance.c	パフォーマンス調査機能
	TstOrgErrCode.c(.h)	ドライバエラーコード表示機能
	DebugFunc.c(.h)	C 言語デバッグ機能
	_DebugFunc.asm	アセンブラデバッグ用機能
FileSystem	…	ファイルシステム。詳細は「2.ファイルシステム」を参照してください。
TEST	…	ファイルシステムのテストプログラム。詳細は「5.テストプログラム」を参照してください。

Obj フォルダには mot 形式の実行ファイルが格納されています。CPU に書き込めばすぐに動作確認がおこなえます。

6.2.5 ドライバ

SPI モードで SD カードにアクセスするためのドライバです。

SPI 通信では DMA を使用して非同期に動作できる仕組みを備えています。AkiH8SCI 環境よりパフォーマンス向上を考慮した設計となっています。

6.2.6 テストプログラム

ドライバテストとパフォーマンス調査に非同期の機能が加わっていますが、その他は AkiH8SCI と同様です。

6.2.7 テストプログラムメニュー構成

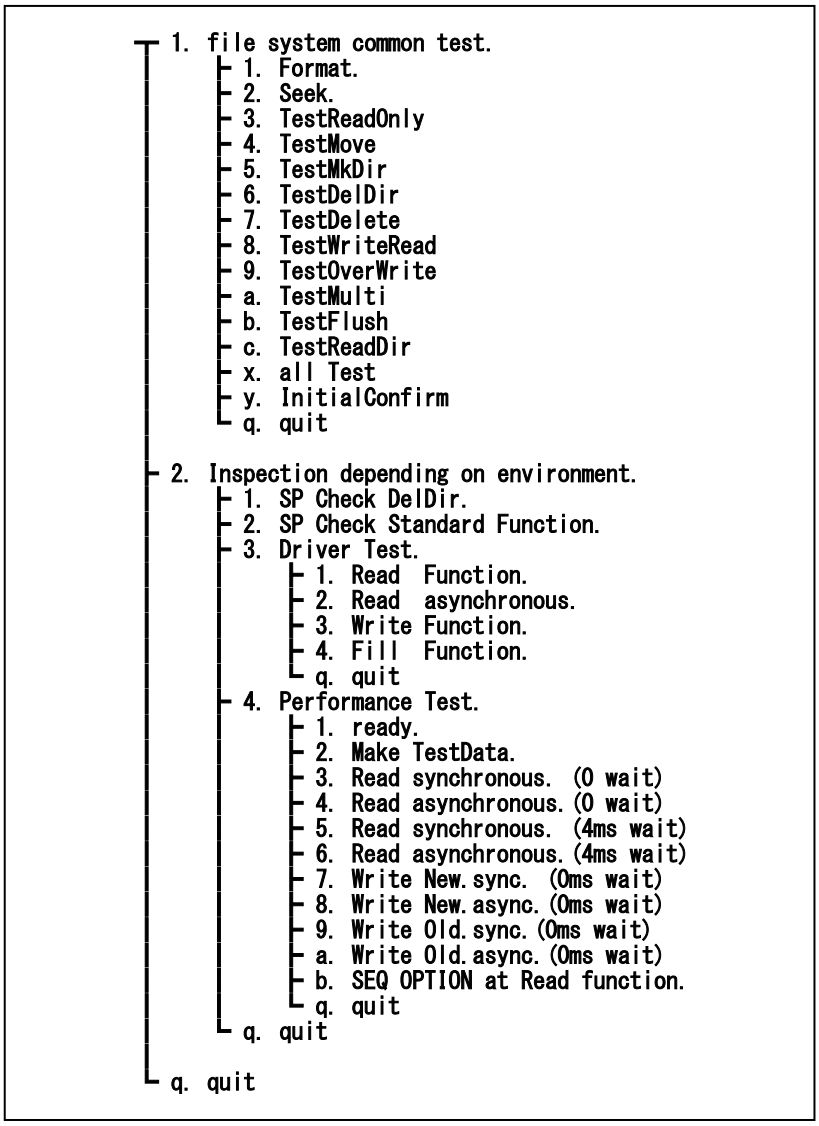

図 6-8 AkiH8DMA テストメニュー

基本的には AkiH8SCI と同様ですが、こちらの動作環境(AkiH8DMA)では非同期のテスト項目が追加されています。
ドライバの Read/Write 機能はきっかけを与えるだけで基本的には DMA を使用して通信をおこない、終了を割り込みで検出します。非同期で read/write をおこなった際の効果を見るために待ち時間を設け、同期して動作する場合との比較をおこなっています。

6.2.8 スタック使用量について

AkiH8DMA 動作環境でのスタック領域の使用量を調べた結果です。スタック領域計算の方法については AkiH8SCI の「6.1.11 スタック使用量について」を参照ください。

表 6-9 AkiH8DMA スタック使用量

機能	解説	スタック使用量
MkDir()	MkDir() 関数でディレクトリを作る際のスタック領域の使用量です。	482バイト
DelDir() 4階層	DelDir() 関数で4階層のサブディレクトリを持つディレクトリ削除時のスタック領域の使用量です。	660バイト
DelDir() 1階層	DelDir() 関数でサブディレクトリを持たないディレクトリを削除した際のスタック領域使用量です。	516バイト
RfOpen()	新規作成モードでファイルをオープンした際のスタック領域使用量です。	510バイト
RfWrite()	10 バイトのデータを書き込んだ際のスタック領域使用量です。	406バイト
RfClose()	上記 RfWrite() 実行後の RfClose() のスタック領域使用量です。	434バイト

6.2.9 パフォーマンスについて

パフォーマンスは状況により変動します。計測したパフォーマンスは性能を保証するものではありません。標準的な一例としてご理解ください。

RfRead

DMA 使用の AkiH8DMA 環境では RF_OPEN_OPT_SEQ オプションを付加すると順番にデータ処理する場合のための最適化をおこなうことができます。RF_OPEN_OPT_SEQ オプションを付加した場合のパフォーマンスを「非同期」として以下の表にパフォーマンス計測結果を載せています。65,536 バイトのファイルを RfRead()で2バイトずつ読み取りその時間を計測しています。

表 6-10 AkiH8DMA RfRead() のパフォーマンス

	～250uS	～500us	～1ms	～2ms	～4ms	～8ms	MIN	MAX
16M 同期 (Panasonic RP-SD016B)	32005	635	0	127	0	1	197us (65534byte)	4,840us (16384byte)
16M 非同期 (Panasonic RP-SD016B)	31879	635	253	0	1	0	197us (65534byte)	3,967us (16382byte)
2G CLASS6 同期 (Panasonic RP-SDV02G)	32259	381	0	0	127	1	197us (65534byte)	6,349us (32768byte)
2G CLASS6 非同期 (Panasonic RP-SDV02G)	32133	381	126	0	127	1	197us (65534byte)	4,280us (32766byte)

2 バイトの RfRead()をおこなうごとに 4ms の待ちを入れた場合の結果です。RF_OPEN_OPT_SEQ オプションを付加した場合、非同期で動作するので次の RfRead() までの時間を空けるとその間に物理的な読み取りが進み効率よく実行することができます。

表 6-11 AkiH8DMA RfRead() のパフォーマンス 4ms の待ち

	～250uS	～500us	～1ms	～2ms	～4ms	～8ms	MIN	MAX
16M 同期 (Panasonic RP-SD016B)	32640	0	0	127	0	1	197us (65534byte)	4,840us (16384byte)
16M 非同期 (Panasonic RP-SD016B)	32641	0	126	0	1	0	197us (65534byte)	3,966us (16382byte)
2G CLASS6 同期 (Panasonic RP-SDV02G)	32640	0	0	0	127	1	198us (65534byte)	6,349us (32768byte)
2G CLASS6 非同期 (Panasonic RP-SDV02G)	32641	0	126	0	0	1	197us (65534byte)	4,280us (32766byte)

RfWrite

新規ファイルの RfWrite() で RF_OPEN_OPT_SEQ の無い場合と付加した場合を比較した結果です。RfWrite()で2バイトずつ書き込み 65,536 バイトのファイルを作っています。計測している時間は RfWrite()の時間です。

表 6-12 AkiH8DMA 新規ファイル RfWrite()のパフォーマンス

	～250uS	～500us	～1ms	～2ms	～4ms	～8ms	MIN	MAX
16M 同期 (Panasonic RP-SD016B)	30862	1525	124	256	0	1	237us (40byte)	4,063us (0byte)
16M 非同期 (Panasonic RP-SD016B)	28065	4445	124	133	0	1	245us (40byte)	4,120us (0byte)
2G CLASS6 同期 (Panasonic RP-SDV02G)	31733	658	126	250	0	1	237us (40byte)	4,412us (0byte)
2G CLASS6 非同期 (Panasonic RP-SDV02G)	31264	1123	126	254	0	1	245us (40byte)	4,423us (0byte)

上書き RfWrite

既存ファイルの RfWrite() で RF_OPEN_OPT_SEQ の無い場合と付加した場合を比較した結果です。

表 6-13 AkiH8DMA 既存ファイル RfWrite()のパフォーマンス

	～250uS	～500us	～1ms	～2ms	～4ms	～8ms	～16ms	16ms～	MIN	MAX
16M 同期 (Panasonic RP-SD016B)	29570	3070	0	1	0	0	0	127	246us (65534byte)	20,051us (16384byte)
16M 非同期 (Panasonic RP-SD016B)	1	32513	127	0	0	0	124	3	246us (65534byte)	19,189us (16384byte)
2G CLASS6 同期 (Panasonic RP-SDV02G)	15128	17512	0	0	1	0	127	0	246us (65534byte)	15,980us (11264byte)
2G CLASS6 非同期 (Panasonic RP-SDV02G)	1	32513	0	0	127	120	7	0	246us (65534byte)	10,281us (49663byte)

6.2.10 実行

テストプログラムの実行方法は AkiH8SCI と同様です。「6.1.13.実行」を参照してください。

6.3 RsfHEW_DMA

6.3.1 概要

ルネサスエレクトロニクス®の IDE、HEW の試用版での動作環境です。HEW のバージョンは、4.06.00.047、ツールチェインバージョンは、7.0.0.0 で確認しています。

HEW で用意されたレジスタ定義ファイルや標準的に用意されるファイルなどの違いを反映していますが、基本的にドライバは AkiH8DMA と同様です。

6.3.2 ビルド方法

RsfHEW_DMA….zip パッケージを解凍します。

SRC¥Env¥Hew_DMA¥RfTestMain フォルダにある RfTestMain.hws を HEW で開きます。

フォルダのパスが異なるため以下の警告が表示されるので、「はい」と答えて先に進みます。

図 6-9 HEW プロジェクトフォルダ移動警告

ビルドしたいプロジェクトをアクティブプロジェクトに設定してビルドを実行します。

6.3.3 フォルダ構成

6.3.3.1 フォルダ構成図

フォルダ構成を以下に示します。

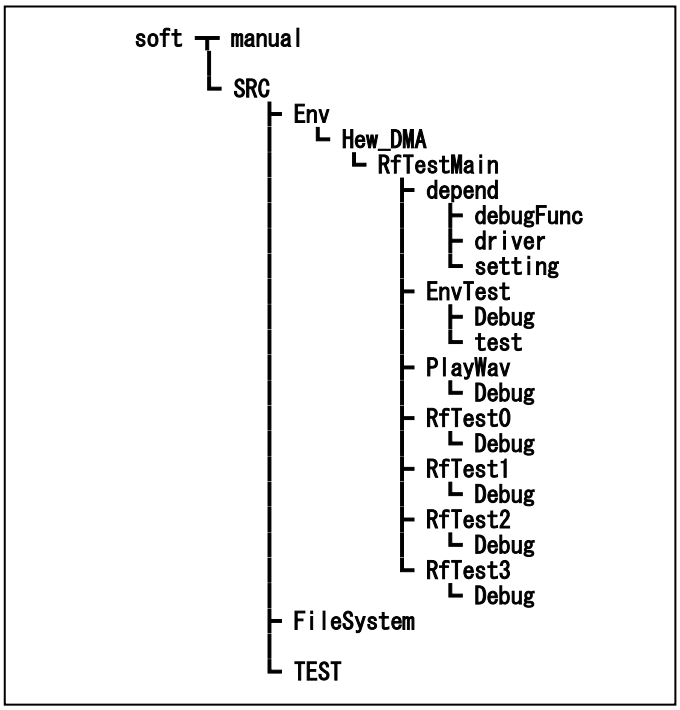

図 6-10 HEW_DMA フォルダ構成

6.3.3.2 フォルダ説明

HEW_DMA パッケージに含まれるフォルダと各プロジェクトの概要を以下の表で説明します。

表 6-14 フォルダとプロジェクト一覧

フォルダ	概要
manual	ファイルシステムのマニュアル
SRC	ソースコード
Env¥HEW_DMA¥RfTestMain	HEW_DMA 環境
depend	HEW_DMA 環境に共通するファイル
EnvTest	環境依存のテストを実行するプロジェクト
PlayWav	wav ファイル再生のサンプルプロジェクト
RfTest0	ファイルシステムのテストプログラムを実行するプロジェクト。「1.Test Format.」、「2.Test seek.」、「3. Test read only.」を実行する。
RfTest1	同様に「4. Test move file.」、「5. Test make directory.」、「6. Test del directory.」、「7.Test delete.」を実行する。
RfTest2	同様に「8. Test write & read」を実行する。
RfTest3	同様に「9. Test over write.」、「a. Test multi file.」、「b. Test flush..」、「c. Test read directory.」を実行する。
FileSystem	ファイルシステムです。詳細は「2.ファイルシステム」を参照してください。
TEST	ファイルシステムのテストプログラム。詳細は「5.テストプログラム」を参照してください。

6.3.4 ファイル一覧

HEW_DMA の動作環境固有のファイルについて説明します。

表 6-15 HEW_DMA ファイル一覧(1)

フォルダ			ファイル名	概要
manual			rsfManual.pdf	ファイルシステムのマニュアル
SRC¥Env¥HEW_DMA¥RfTestMain			RfTestMain.hws	HEW のプロジェクトファイル
	depend	debugFunc	DebugFunc.c(.h)	C 言語デバッグ機能
			_DebugFunc.asm	アセンブラデバッグ用機能
			TstOrgErrCode.c(.h)	ドライバエラーコード表示機能
			TraceOut.c	テスト用入出力
		Driver	DrvSdRegister.h	SD カードレジスタの定義ファイル
			DrvErrCode.h	ドライバのエラーコード定義ファイル
			DrvSdCard.c(.h)	SD カードドライバ
			DrvPort.c(.h)	ポートドライバ
			DrvSdDmaSciItu.c(.h)	SCI と DMA を使用する SPI ドライバ
			DrvCpuXtal.h	クロック定義
		Setting	rfSettings.c	ファイルシステムの設定ファイル
	EnvTest		dbsct.c	セクション情報ファイル
			intprg.c	割り込みエントリポイント
			iodefine.h	ペリフェラルレジスタ定義ファイル
			resetprg.c	ブート部ソース
			EnvTest.hwp DefaultSession.hsf	HEW のプロジェクトファイル
			stacksct.h	スタック定義ファイル
			typedefine.h	データ型定義ファイル
			EnvTest.c	メイン関数
		Debug	EnvTest.map	マップファイル
			EnvTest.mot	モトローラ S フォーマット形式の実行ファイル
		test	TstOrgSelect.c	テスト機能選択メニュー
			TstOrgDelDir.c	DelDir のスタック使用量調査機能
			TstOrgInitialConfirm.c	ファイル入出力のスタック使用量調査
			TstOrgDriver.c	ドライバ機能テスト
			TstOrgPerformance.c	パフォーマンス調査機能
	PlayWav		dbsct.c	HEW が生成したセクション情報ファイル
			intprg.c	HEW が生成した割り込みエントリポイント
			iodefine.h	HEW が生成したペリフェラルレジスタ定義ファイル
			resetprg.c	HEW が生成したブート部ソース
			PlayWav.hwp DefaultSession.hsf	HEW が生成したプロジェクトファイル
			stacksct.h	HEW が生成したスタック定義ファイル
			typedefine.h	HEW が生成したデータ型定義ファイル
			PlayWav.c	メイン関数
			soundOut.c(.h)	音声出力ドライバ部
		Debug	PlayWav.map	マップファイル
			PlayWav.mot	モトローラ S フォーマット形式の実行ファイル

表 6-16 HEW_DMA ファイル一覧(2)

フォルダ		ファイル名	概要
SRC¥Env¥HEW_DMA¥RfTestMain			
RfTest0		dbsct.c	HEW が生成したセクション情報ファイル
		intprg.c	HEW が生成した割り込みエントリポイント
		iodefine.h	HEW が生成したペリフェラルレジスタ定義ファイル
		resetprg.c	HEW が生成したブート部ソース
		RfTest0.hwp DefaultSession.hsf	HEW が生成したプロジェクトファイル
		stacksct.h	HEW が生成したスタック定義ファイル
		typedefine.h	HEW が生成したデータ型定義ファイル
		RfTest.c	メイン関数
		TstSelect.c	ファイルシステムテストプログラムのメニュー管理
	Debug	RfTest0.map	マップファイル
		RfTest0.map	モトローラ S フォーマット形式の実行ファイル
RfTest1		RfTest0 と同様でプロジェクト名を RfTest1 に変更	
RfTest2		RfTest0 と同様でプロジェクト名を RfTest2 に変更	
RfTest3		RfTest0 と同様でプロジェクト名を RfTest3 に変更	

6.3.5 サンプルプロジェクト

6.3.5.1 プロジェクト概要

表 6-17 HEW_DMA プロジェクト概要

プロジェクト	概要
EnvTest	環境依存のテストを実行するプロジェクトです。スタック使用量、パフォーマンスを調べます。
PlayWav	wav ファイルの再生をおこなうサンプルプロジェクトです。SD カードのルートディレクトリにある wav ファイルを検索し再生します。また、内部データとして保持している 50Hz、440Hz の sin 波を再生します。
RfTest0～3	ファイルシステムのテストプロジェクトです。ファイルシステムの機能テストをおこないます。 試用版の HEW で動作可能なサイズに分割しています。

各プロジェクトの Obj フォルダには mot 形式の実行ファイルが格納されています。CPU に書き込めばすぐに動作確認がおこなえます。

6.3.5.2 実行方法

テストプログラムの実行方法は AkiH8SCI と同様です。「6.1.13.実行」を参照してください。

6.3.6 環境依存のテスト (EnvTest)

6.3.6.1 メニュー構成

```
┬ 1. SP Check DelDir.
├ 2. SP Check Standard Function.
├ 4. Performance Test.
│    ├ 1. Ready.
│    ├ 2. Make TestData.
│    ├ 3. Read synchronous.  (0 wait)
│    ├ 4. Read asynchronous. (0 wait)
│    ├ 5. Read synchronous.  (4ms wait)
│    ├ 6. Read asynchronous. (4ms wait)
│    ├ 7. Write New. sync.  (0ms wait)
│    ├ 8. Write New. async. (0ms wait)
│    ├ 9. Write Old. sync. (0ms wait)
│    ├ a. Write Old. async. (0ms wait)
│    ├ b. SEQ OPTION at Read function.
│    └ q. quit
└ q. quit
```

図 6-11 HEW_DMA EnvTest メニュー

6.3.6.2 スタック使用量について

HEW_DMA 動作環境でのスタック領域の使用量を調べた結果です。サイズ優先の最適化をかけています。
スタック領域計算方法については「6.1.11 スタック使用量について」を参照ください。

表 6-18 HEW_DMA スタック使用量

機能	解説	スタック使用量
MkDir()	MkDir() 関数でディレクトリを作る際のスタック領域の使用量です。	378バイト
DelDir() 4階層	DelDir() 関数で4階層のサブディレクトリを持つディレクトリ削除時のスタック領域の使用量です。	484バイト
DelDir() 1階層	DelDir() 関数でサブディレクトリを持たないディレクトリを削除した際のスタック領域使用量です。	388バイト
RfOpen()	新規作成モードでファイルをオープンした際のスタック領域使用量です。	378バイト
RfWrite()	10 バイトのデータを書き込んだ際のスタック領域使用量です。	334バイト
RfClose()	上記 RfWrite() 実行後の RfClose() のスタック領域使用量です。	342バイト

6.3.6.3 パフォーマンスについて

パフォーマンスは状況により変動します。計測したパフォーマンスは性能を保証するものではありません。標準的な一例としてご理解ください。

RfRead

DMA 使用の HEW_DMA 環境では AkiH8DMA と同様に RF_OPEN_OPT_SEQ オプションを付加すると順番にデータ処理する場合のための最適化をおこなうことができます。RF_OPEN_OPT_SEQ オプションを付加した場合を「非同期」として以下の表にパフォーマンスを記述しています。65,536 バイトのファイルを RfRead()で2バイトずつ読み取りその時間を計測しています。

表 6-19 HEW_DMA RfRead()のパフォーマンス

	～250uS	～500us	～1ms	～2ms	～4ms	～8ms	MIN	MAX
16M 同期 (Panasonic RP-SD016B)	32513	127	0	127	0	1	140us (65534byte)	4,560 us (16384byte)
16M 非同期 (Panasonic RP-SD016B)	32387	127	253	0	1	0	141us (65536byte)	3,706 us (16382byte)
2G CLASS6 同期 (Panasonic RP-SDV02G)	32513	127	0	0	127	1	141us (266byte)	6,042 us (32768byte)
2G CLASS6 非同期 (Panasonic RP-SDV02G)	32514	0	126	0	128	0	141us (65534byte)	3,997 us (32766byte)

2 バイトの RfRead()をおこなうごとに 4ms の待ちを入れた場合の結果です。RF_OPEN_OPT_SEQ オプションを付加した場合、非同期で動作するので次の RfRead() までの時間を空けるとその間に物理的な読み取りが進み効率よく実行することができます。

表 6-20 HEW_DMA RfRead パフォーマンス 4ms の待ち

	～250uS	～500us	～1ms	～2ms	～4ms	～8ms	MIN	MAX
16M 同期 (Panasonic RP-SD016B)	32640	0	0	127	0	1	140 us (65534byte)	4,560 us (16384byte)
16M 非同期 (Panasonic RP-SD016B)	32641	0	126	0	1	0	140 us (65534byte)	3,705 us (16384byte)
2G CLASS6 同期 (Panasonic RP-SDV02G)	32640	0	0	0	127	1	141 us (65534byte)	6,042 us (32768byte)
2G CLASS6 非同期 (Panasonic RP-SDV02G)	32641	0	126	0	1	0	141 us (65534byte)	3,997 us (32766byte)

RfWrite

新規ファイルの RfWrite() で RF_OPEN_OPT_SEQ の無い場合と付加した場合を比較した結果です。RfWrite()で2バイトずつ書き込み 65,536 バイトのファイルを作っています。計測している時間は RfWrite()の時間です。

表 6-21 HEW_DMA 新規ファイル RfWrite()のパフォーマンス

	～250uS	～500us	～1ms	～2ms	～4ms	～8ms	MIN	MAX
16M 同期 (Panasonic RP-SD016B)	31876	653	253	3	1	0	189us (58byte)	3,831 us (0byte)
16M 非同期 (Panasonic RP-SD016B)	31748	764	251	4	1	0	197us (56byte)	3,842 us (0byte)
2G CLASS6 同期 (Panasonic RP-SDV02G)	32380	12	374	1	0	1	189us (26byte)	4,142 us (0byte)
2G CLASS6 非同期 (Panasonic RP-SDV02G)	32248	264	253	2	0	1	197us (26yte)	4,153 us (0byte)

上書き RfWrite

既存ファイルの RfWrite() で RF_OPEN_OPT_SEQ の無い場合と付加した場合を比較した結果です。

表 6-22 HEW_DAM 既存ファイル RfWrite()のパフォーマンス

	～250uS	～500us	～1ms	～2ms	～4ms	～8ms	～16ms	16ms～	MIN	MAX
16M 同期 (Panasonic RP-SD016B)	32513	127	0	1	0	0	126	1	197 us (65534byte)	18,953 us (16384byte)
16M 非同期 (Panasonic RP-SD016B)	31879	635	127	0	0	0	126	1	197 us (65534byte)	18,128 us (16382byte)
2G CLASS6 同期 (Panasonic RP-SDV02G)	32513	127	0	0	1	120	6	1	197 us (65534byte)	16,333 us (32768byte)
2G CLASS6 非同期 (Panasonic RP-SDV02G)	32133	381	0	0	127	120	7	0	197 us (65534byte)	9,441 us (33278byte)

6.3.7 wav ファイル再生プロジェクト(PlayWav)

wav ファイル再生サンプルプロジェクトのメニュー構成です。

```
┬ 1. play wav file.
│     ├ 1. ¥AQUARIUM.WAV
│     ├ 2. ¥AUTUMN.WAV
│     ├ 3. ¥NIGHTM~1.WAV
│     ├ 4. ¥ONE'SF~1.WAV
│     ├ 5. ¥QUIETC~1.WAV
│     ├ 6. ¥QUIETM~1.WAV
│     ├ 7. ¥QUIETS~1.WAV
│     ├ 8. ¥ROUTE6~1.WAV
│     ├ 9. ¥THEROU~1.WAV
│     ├ a. ¥WHITER~1.WAV
│     ├ b. ¥WINTER~1.WAV
│     └ q. quit
├ 2. out sin wave.
│     ├ 1. 50Hz
│     ├ 2. 440Hz
│     └ q. quit
└ q. quit
```

図 6-12 HEW_DMA PlayWav プロジェクトメニュー構成

6.3.7.1 play wav file メニュー

SD カードのルートディレクトリにある、拡張子が wav のファイルを表示します。
ファイル名のメニュー項目番号を入力するとそのファイルを再生します。
ここに表示されているファイルはテストを実行した際の物です。これらのサンプルファイルもテストデータとして提供しており、以下 URL からダウンロードできます。ファイル内容については「10.サンプル wav ファイル」を参照してください。

http://homepage3.nifty.com/resove/fileSystem

6.3.7.2 out sin wave メニュー

テストソースコード内部で持っている 50Hz と 440Hz の sin 波形を出力します。

6.3.7.3 wev ファイル仕様

wev ファイルは 11,025Hz サンプリング、ステレオ、8 ビット分解能のものが再生できます。

6.3.7.4 ソフトウェアの仕組み

DMA を使用して wav ファイルから読み取ったデータを D/A 出力します。D/A データレジスタが 1 バイトで、ch0 と ch1 が連続したアドレスにあるため DMA を word 転送とし左右チャネルのデータを同時に転送しています。

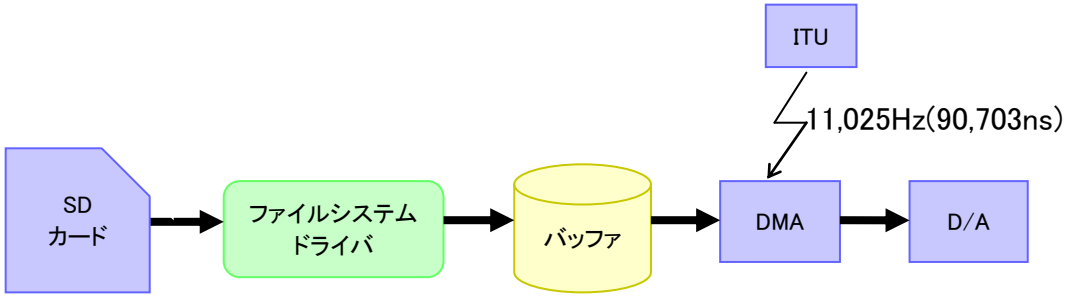

図 6-13 HEW_DMA PlayWav データの流れ

ファイルの読み取りは 512 バイトのセクタ毎に物理的なアクセスが発生し時間がかかってしまいます。読み取り速度をなるべく均一にするため wav ファイルの読み取りも 512 バイト単位でおこなっています。512 バイトのバッファを2つ用意し片方のバッファの DMA 転送中にもう片方のバッファに wav ファイルから読み込んでいます。
11,025Hz のサンプリングスピードで 512 バイト(2ch ぶん、256sample)の D/A 出力では、256sample÷11,025Hz ≒ 23ms の時間がかかります。「6.3.6 環境依存のテスト (EnvTest)」の調査結果でセクタの読み取りが発生する際には 2ms 以内、FAT の読み取りが発生する際でも 8ms 以内に読み取りが終了するので十分間に合うと考えられます。

6.3.8 ファイルシステムテストプログラム (RfTest0～3)

ファイルシステムテストプログラムは HEW 試用版の実行ファイルサイズ制限のため4つのプロジェクトに分割しています。内容については「5.1 テスト機能ソース」を参照してください。

6.3.8.1 メニュー構成

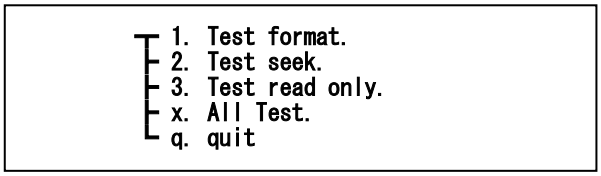

図 6-14 HEW_DMA RfTest0 メニュー構成

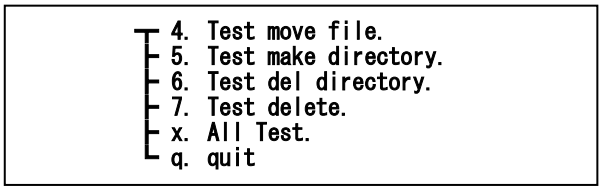

図 6-15 HEW_DMA RfTest1 メニュー構成

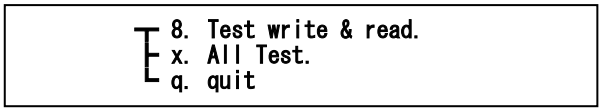

図 6-16 HEW_DMA RfTest2 メニュー構成

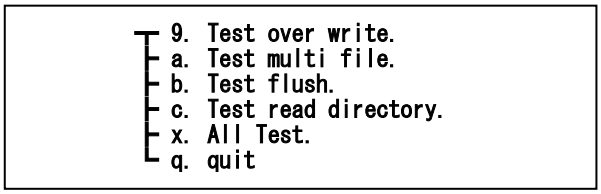

図 6-17 HEW_DMA RfTest3 メニュー構成

6.4 RsfGcc_H8

6.4.1 概要

GNU、GCC での動作環境です。AkiH8DMA のレジスタ定義ファイルを使用しています。GCC で使用する標準的なラベルや割り込み関数の定義などは反映していますが、基本的にドライバは AkiH8DMA と同様です。
GCC バージョンは以下で確認しています。

binutils-2.17
newlib-1.20.0
gcc-4.7.2

また、ビルド環境は、cygwin(mintty1.1.2)、Linux(2.6.32-5-686(Debian2.6.32-46))で動作確認しています。

6.4.2 フォルダ構成

6.4.2.1 フォルダ構成図

フォルダ構成を以下に示します。

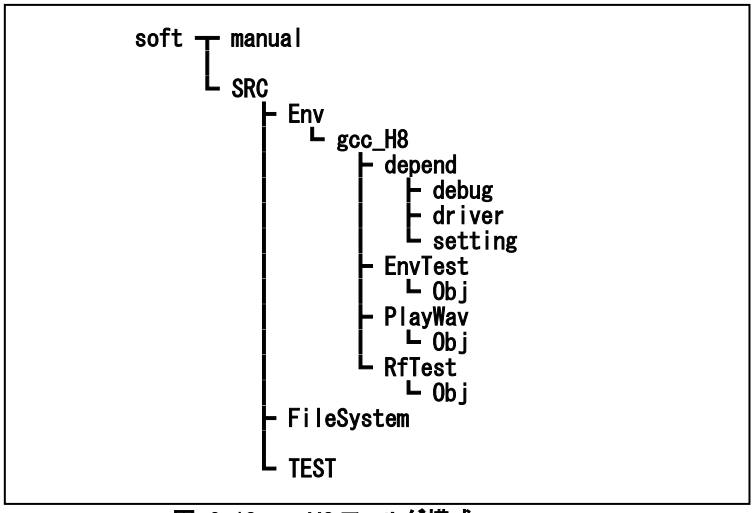

図 6-18 gcc_H8 フォルダ構成

6.4.2.2 フォルダ説明

gcc_H8 パッケージに含まれるフォルダと各プロジェクトの概要を以下の表で説明します。

表 6-23 フォルダとプロジェクト一覧

フォルダ	概要
manual	ファイルシステムのマニュアル
SRC	ソースコード
Env¥gcc_H8	Gcc 環境
Depend	gcc_H8 環境に共通するファイルを格納している
EnvTest	環境依存のテストを実行するプロジェクト
PlayWav	wav ファイル再生のサンプルプロジェクト
RfTest	ファイルシステムのテストプログラムを実行するプロジェクト。
FileSystem	ファイルシステム。詳細は「2.ファイルシステム」を参照してください。
TEST	ファイルシステムのテストプログラム。詳細は「5.テストプログラム」を参照してください。

6.4.3 ファイル一覧

gcc_H8 の動作環境固有のファイル一覧を示します。

表 6-24 gcc_H8 ファイル一覧

フォルダ	ファイル名	概要
manual	rsfManual.pdf	ファイルシステムのマニュアル
SRC¥Env¥gcc_H8	Makefile	プロジェクト生成のメイクファイル
Depend	LinkerScfript.x	リンク設定を記述するリンカスクリプトファイル
debug	DebugFunc.c(.h)	C 言語デバッグ機能
	_DebugFunc.asm	アセンブラデバッグ用機能
	RfVsprintf.c	標準の vsprintf()に代わる機能
	TraceOut.c	テスト用入出力
	TstOrgErrCode.c(.h)	ドライバエラーコード表示機能
driver	3048F.H	H8/3048 のペリフェラルレジスタ定義ファイル
	DrvCpuXtal.h	クロック定義
	DrvDma.c(.h)	DMA 機能ドライバ
	DrvErrCode.h	ドライバのエラーコード定義ファイル
	DrvPort.c(.h)	ポートドライバ
	DrvSdCard.c(.h)	SD カードドライバ
	DrvSdDmaSciItu.c(.h)	SCI と DMA を使用する SPI ドライバ
	DrvSdRegister.h	SD カードレジスタの定義ファイル
setting	rfSettings.c	ファイルシステムの設定ファイル
EnvTest	EnvTest.c	環境依存テストのメイン
	Makefile	メイクファイル
	TstOrgDelDir.c	DelDir のスタック使用量調査機能
	TstOrgDriver.c	ドライバ機能テスト
	TstOrgInitialConfirm.c	ファイル入出力のスタック使用量調査
	TstLowDriver.c	低レベルドライバ機能テスト
	TstOrgPerformance.c	パフォーマンス調査機能
	TstOrgSelect.c	テスト機能選択メニュー
	VectorTbl.s	割り込みベクタテーブル
Obj	EnvTest.abs	abs 形式の実行ファイル
	EnvTest.map	マップファイル
	EnvTest.mot	環境依存部テストの実行ファイル
PlayWav	Makefile	PlayWav プロジェクトのメイクファイル
	PlayWav.c	Wav ファイル再生ユーザーインタフェース部
	soundOut.c(.h)	音再生ドライバ部分
	VectorTbl.s	割り込みベクタテーブル
Obj	EnvTest.abs	Abs 形式の実行ファイル
	EnvTest.mot	モトローラ S フォーマットの実行ファイル
	EnvTest.map	マップファイル
RfTest	Makefile	RfTest プロジェクトのメイクファイル
	RfTest.c	ファイルシステムテストのメイン
	VectorTbl.s	割り込みベクタテーブル
Obj	RfTest.abs	Abs 形式の実行ファイル
	RfTest.mot	マップファイル
	RfTest.map	モトローラ S フォーマット形式の実行ファイル

6.4.4 サンプルプロジェクト

表 6-25 gcc_H8 プロジェクト概要

プロジェクト	概要
EnvTest	環境依存のテストを実行するプロジェクトです。スタック使用量、パフォーマンスを調べます。
PlayWav	wav ファイルの再生をおこなうサンプルプロジェクトです。SD カードのルートディレクトリにある wav ファイルを検索し再生します。また、内部データとして保持している 50Hz、440Hz の sin 波を再生します。
RfTest	ファイルシステムテストプロジェクトです。ファイルシステムの機能テストをおこないます。

各プロジェクトの Obj フォルダには mot 形式の実行ファイルが格納されています。CPU に書き込めばすぐに動作確認がおこなえます。

6.4.4.1 ビルド方法

各プロジェクトフォルダにはそれぞれ makefile がありそのフォルダ下のプロジェクトのビルドをおこないます。gcc_H8/Env フォルダで「make」を実行すると3つのプロジェクトのビルドをおこないます。

6.4.4.2 実行方法

テストプログラムの実行方法は AkiH8SCI と同様です。「6.1.13.実行」を参照してください。

6.4.5 環境依存のテスト(EnvTest)

環境依存のテストはスタック使用量、パフォーマンス計測、ドライバテストをおこなうプログラムです。

6.4.5.1 メニュー構成

```
┬ 1. SP consumtion DelDir.
├ 2. SP consumption Standard Function.
├ 3. Low Driver Test.
├ 4. Driver Test.
├ 5. Performance Test.
│    ├ 1. Ready.
│    ├ 2. Make TestData.
│    ├ 3. Read synchronous.  (0 wait)
│    ├ 4. Read asynchronous. (0 wait)
│    ├ 5. Read synchronous.  (4ms wait)
│    ├ 6. Read asynchronous. (4ms wait)
│    ├ 7. Write New.sync.  (0ms wait)
│    ├ 8. Write New.async. (0ms wait)
│    ├ 9. Write Old.sync. (0ms wait)
│    ├ a. Write Old.async. (0ms wait)
│    ├ b. SEQ OPTION at Read function.
│    └ q. quit
│
└ q. quit
```

図 6-19 gcc_H8 環境依存部テストメニュー

6.4.5.2 スタック使用量について

gcc_H8 動作環境でのスタック領域の使用量を調べた結果です。
スタック領域計算の方法については「6.1.11 スタック使用量について」を参照ください。

表 6-26 gcc_H8 スタック使用量

機能	解説	スタック使用量
MkDir()	MkDir() 関数でディレクトリを作る際のスタック領域の使用量です。	428バイト
DelDir() 4階層	DelDir() 関数で4階層のサブディレクトリを持つディレクトリ削除時のスタック領域の使用量です。	596バイト
DelDir() 1階層	DelDir() 関数でサブディレクトリを持たないディレクトリを削除した際のスタック領域使用量です。	472バイト
RfOpen()	新規作成モードでファイルをオープンした際のスタック領域使用量です。	456バイト
RfWrite()	10 バイトのデータを書き込んだ際のスタック領域使用量です。	378バイト
RfClose()	上記 RfWrite() 実行後の RfClose() のスタック領域使用量です。	384バイト

6.4.5.3 パフォーマンスについて

パフォーマンスは状況により変動します。計測したパフォーマンスは性能を保証するものではありません。標準的な一例としてご理解ください。

RfRead

DMA 使用の gcc_H8 環境では AkiH8DMA と同様に RF_OPEN_OPT_SEQ オプションを付加すると順番にデータ処理する場合のための最適化をおこなうことができます。RF_OPEN_OPT_SEQ オプションを付加した場合を「非同期」として以下の表にパフォーマンスを記述しています。65,536 バイトのファイルを RfRead()で2バイトずつ読み取りその時間を計測しています。

表 6-27 gcc_H8 RfRead()のパフォーマンス

	～250uS	～500us	～1ms	～2ms	～4ms	～8ms	8ms～	MIN	MAX
16M 同期 (Panasonic RP-SD016B)	32640	635	0	0	127	1	0	196us (65534byte)	6,030 us (16384byte)
16M 非同期 (Panasonic RP-SD016B)	31879	635	126	127	0	1	0	196us (65534byte)	4,587 us (16382byte)
2G CLASS6 同期 (Panasonic RP-SDV02G)	32259	381	0	0	0	127	1	196 us (65534byte)	9,525us (32768byte)
2G CLASS6 非同期 (Panasonic RP-SDV02G)	32133	381	126	0	127	1	0	196 us (65534byte)	5,894 us (32766byte)

2 バイトの RfRead()をおこなうごとに 4ms の待ちを入れた場合の結果です。RF_OPEN_OPT_SEQ オプションを付加した場合、非同期で動作するので次の RfRead() までの時間を空けるとその間に物理的な読み取りが進み効率よく実行することができます。

表 6-28 gcc_H8 RfRead パフォーマンス 4ms の待ち

	～250uS	～500us	～1ms	～2ms	～4ms	～8ms	8ms～	MIN	MAX
16M 同期 (Panasonic RP-SD016B)	32640	0	0	0	127	1	0	196 us (65534byte)	6,030 us (16384byte)
16M 非同期 (Panasonic RP-SD016B)	32641	0	126	0	0	1	0	196 us (65534byte)	4,587 us (16382byte)
2G CLASS6 同期 (Panasonic RP-SDV02G)	32640	0	0	0	0	127	1	196 us (65534byte)	9,525 us (32768byte)
2G CLASS6 非同期 (Panasonic RP-SDV02G)	32641	0	126	0	0	1	0	196 us (65534byte)	5,894 us (32766byte)

RfWrite

新規ファイルの RfWrite() で RF_OPEN_OPT_SEQ の無い場合と付加した場合を比較した結果です。RfWrite()で2バイトずつ書き込み65,536 バイトのファイルを作っています。計測している時間は RfWrite()の時間です。

表 6-29 gcc_H8 新規ファイル RfWrite()のパフォーマンス

	～250uS	～500us	～1ms	～2ms	～4ms	～8ms	MIN	MAX
16M 同期 (Panasonic RP-SD016B)	30989	1399	124	3	252	1	224us (38byte)	4,713 us (0byte)
16M 非同期 (Panasonic RP-SD016B)	30986	1526	124	4	127	1	234 us (36byte)	4,761 us (0byte)
2G CLASS6 同期 (Panasonic RP-SDV02G)	31741	769	126	1	130	1	224 us (28byte)	6,062us (0byte)
2G CLASS6 非同期 (Panasonic RP-SDV02G)	31723	914	126	2	2	1	234 us (26yte)	6,071 us (0byte)

上書き RfWrite

既存ファイルの RfWrite() で RF_OPEN_OPT_SEQ の無い場合と付加した場合を比較した結果です。

表 6-30 gcc_H8 既存ファイル RfWrite()のパフォーマンス

	～250uS	～500us	～1ms	～2ms	～4ms	～8ms	～16ms	16ms～	MIN	MAX
16M 同期 (Panasonic RP-SD016B)	29825	2815	0	0	1	0	0	127	242 us (65534byte)	22,405 us (16384byte)
16M 非同期 (Panasonic RP-SD016B)	1	32513	0	127	0	0	0	127	242 us (65534byte)	20,975 us (16382byte)
2G CLASS6 同期 (Panasonic RP-SDV02G)	15887	16753	0	0	0	1	126	1	242 us (65534byte)	17,330 us (32768byte)
2G CLASS6 非同期 (Panasonic RP-SDV02G)	1	32513	0	0	127	0	127	0	241 us (65534byte)	13,716 us (32766byte)

6.4.6 wav ファイル再生サンプル(PlayWav)

wav ファイル再生サンプルプロジェクトのメニュー構成です。

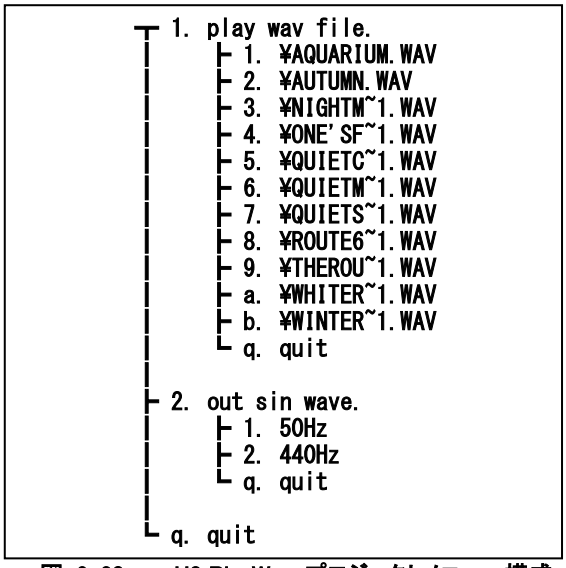

図 6-20 gcc_H8 PlayWav プロジェクトメニュー構成

内容は、HEW_DMA と同様です。「6.3.7wav ファイル再生プロジェクト(PlayWav)」を参照してください。

6.4.7 ファイルシステムテストプログラム(RfTest)

ファイルシステムのテストプロジェクトです。内容については「5.1 テスト機能ソース」を参照してください。

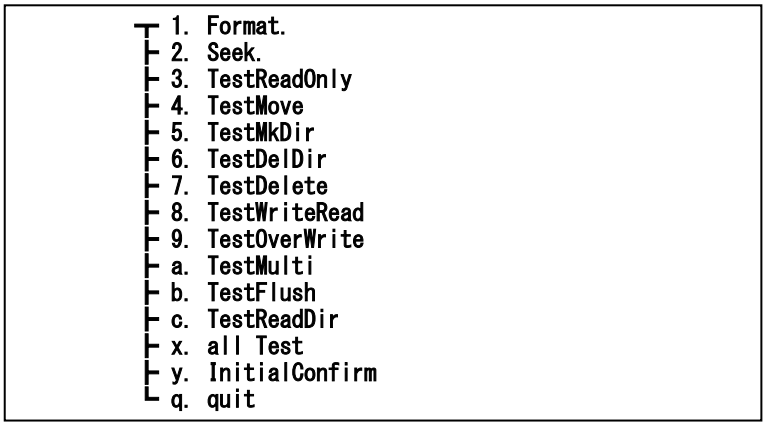

図 6-21 gcc_H8 ファイルシステムテストメニュー

6.5 RsfLPC1768XPresso

6.5.1 概要

NGX TECHNOLOGIES 社の「LPC1768 Xplorer」ボードでサンプルプログラムを動作させる環境です。
LPCXpressoIDE を使用してビルド、実行します。LPCXpresso のバージョンは 6.1.0 で確認しています。
LPC1768Xpresso 環境のドライバは RF_OPEN_OPT_SEQ には対応していません。SD カードとの通信は 20MHz でおこなっています。

6.5.2 ファイル一覧

RsfLPC1768Xpresso パッケージに含まれるフォルダとその概要を以下に示します。

表 6-31 LPC1768Xpresso のプロジェクト

フォルダ	概要
manual	ファイルシステムのマニュアル
CMSISv1p30_LPC17xx	CMSIS ライブラリ
depend	環境依存ライブラリ
FileSystem	ファイルシステムライブラリ
LPC1768-xplorer_usbVcom_lib	仮想 COM ポートライブラリ
mainEnvTest	環境依存テストプロジェクト
mainPlayWav	Wav ファイル再生サンプルプロジェクト
mainRfTest	ファイルシステムテストプロジェクト
TEST	ファイルシステムテストライブラリ

各フォルダのファイル内容を説明します。

表 6-32 LPC1768Xpresso ファイル一覧(1)

フォルダ	ファイル名	概要
manual	rsfManual.pdf	ファイルシステムのマニュアル
CMSISv1p30_LPC17xx	…	NGX から取得した CMSIS ライブラリなので説明は省略します。インクルードファイルのパスを修正しています。
depend	deteTime.c(.h)	日付、時刻設定機能
	DebugFunc.c(.h)	デバッグ機能
	DrvDma.c(.h)	DMA ドライバ
	DrvErrCode.h	ドライバのエラーコード定義ファイル
	DevSdCard.c(.h)	SD カードドライバ
	DrvSdRegister.h	SD カードレジスタの定義ファイル
	DrvSpiForSd.c(.h)	SD カード用 SPI 通信のドライバ
	rfSettings.c	ファイルシステムの設定ファイル
	settingClock.c(.h)	RTC(時計)ドライバ
	TraceOut.c	テスト用入出力
	TstOrgErrCode.c(.h)	ドライバエラーコード表示機能
FileSystem	…	FileSystem です。ライブラリ化するため FileSystem ソースをワークスペース内に置いています。詳細は「2.ファイルシステム」を参照してください。
LPC1768-xplorer_usbVcom_lib	…	仮想 COM ポートライブラリです。NGX から提供されたサンプルライブラリなので説明を省略します。
mainEnvTest	src¥crp.c	RED ライブラリ使用時の定義
	src¥cr_startup_lpc176x.c	スタートアップ
	src¥TstOrgDelDir.c	DelDir のスタック使用量調査機能
	src¥TstOrgDriver.c	ドライバ機能テスト
	src¥TstOrgInitialConfirm.c	ファイル入出力のスタック使用量調査
	src¥TstOrgPerformance.c	パフォーマンス調査機能
	src¥TstOrgSelect.c	テスト機能選択メニュー
	debug¥mainEnvTest.map	プロジェクトのマップファイル
	debug¥mainEnvTest.axf	axf 形式の実行ファイル
	debug¥mainEnvTest.hex	hex フォーマットの実行ファイル。

表 6-33 LPC1768Xpresso ファイル一覧(2)

フォルダ	ファイル名	概要
mainPlayWav	FreeRTOS_include	FreeRTOS のフォルダなので説明を省略します
	FreeRTOS_portable	FreeRTOS のフォルダなので説明を省略します
	FreeRTOS_src	FreeRTOS のフォルダなので説明を省略します
	src¥cr_startup_lpc17.c	スタートアップ
	src¥FreeRTOSConfig.h	OS の定義ファイル
	src¥main.c	wav ファイル再生プロジェクトのメイン
	src¥playWave.c(.h)	wav ファイル再生のユーザーインタフェース
	src¥soundOut.c(.h)	音声再生のドライバ部分
	debug¥mainPlayWav.axf	axf 形式の実行ファイル
	debug¥mainPlayWav.bin	bin 形式の実行ファイル
	debug¥mainPlayWav.hex	hex フォーマットの実行ファイル
	debug¥mainPlayWav.map	プロジェクトのマップファイル
mainRfTest	cr_startup_lpc176x.c	スタートアップ
	crp.c	RED ライブラリ使用時の定義
	Debug¥mainRfTest.axf	axf 形式の実行ファイル
	Debug¥mainRfTest.map	プロジェクトのマップファイル
	main.c	ファイルシステムテストのメイン
TEST	…	ファイルシステムのテストプログラム。詳細は「5.テストプログラム」を参照してください。

6.5.3 サンプルプロジェクト

6.5.3.1 プロジェクト概要

以下のサンプルプロジェクトを用意しました。各プロジェクトの概要を以下に示します。

表 6-34 LPC1768XPresso プロジェクト

プロジェクト	概要
mainRfTest	ファイルシステムのテストプロジェクトです。ファイルシステムの機能テストをおこないます。
mainEnvTest	環境に依存するテストプロジェクトです。ドライバの基本機能をテストします。また、スタック使用量、パフォーマンスについて調べます。
mainPlayWav	wav ファイルの再生をおこなうサンプルプロジェクトです。SD カードのルートディレクトリにある wav ファイルを検索し再生します。また、内部データとして保持している 50Hz、440Hz の sin 波を再生します。

また、各プロジェクトには Debug フォルダの下に hex 形式の実行ファイルが格納されています。Flash Magic などで直接書き込むとすぐに動作確認がおこなえます。

6.5.3.2 ビルド方法

LPCXpressoIDE を使用してワークスペースを作成し、「Inport project(s)」で RsfLPC1768Xpresso….zip パッケージからすべてのプロジェクトを取りこみます。プロジェクトはパッケージの「manual」以外の各フォルダです。

- CMSISv1p30_LPC17xx
- depend
- FileSystem
- LPC1768-xplorer_usbVcom_lib
- mainEnvTest
- mainPlayWav
- mainRfTest
- TEST

取りこんだらすべてをビルドしてください。

6.5.3.3 実行方法

どのプロジェクトも仮想 COM ポートとしてパソコンと接続するようになっています。仮想 COM ポートの設定は「LPC1768 Xplorer」の「Quick Start Guide」を参照してください。

実行ファイルを実機に書き込み電源を入れる、またはデバッガで実行を開始すると LED が 3、3、7 拍子で点滅します。仮想 COM ポートの準備ができた合図なので、パソコン側ではターミナルソフトなどを立ち上げ何か1文字送信します。

LPC1768 側は最初の文字を受信するとメニュー表示を開始します。

6.5.3.4 通信設定

パソコンのターミナルソフト設定は以下のようにおこないます。

表 6-35 シリアル設定

項目	内容
ポート	仮想 COM ポートとして認識したポート
ボーレート	9600bps
データ長	8 ビット
パリティ	なし
ストップビット	1ビット
フロー制御	なし

また、端末の改行コードは送信が LF、受信は CR と設定してください。

日付、時刻の設定で BS(バックスペース)キーを使えるようにしているため、BS を有効にしてください。

6.5.4 環境依存のテスト(mainEnvTest)

6.5.4.1 メニュー構成

```
┬ 1. Inspection depending on environment.
│    ├ 1. Stack consumption DelDir.
│    ├ 2. Stack consumption Standard Function.
│    ├ 3. Driver Test.
│    │    ├ 1. Read  Function.
│    │    ├ 2. Read Detail.
│    │    ├ 3. Write Function.
│    │    ├ 4. Fill  Function.
│    │    └ q. quit
│    ├ 4. Performance Test.
│    │    ├ 1. ready.
│    │    ├ 2. Make TestData.
│    │    ├ 3. Read synchronous.
│    │    ├ 4. Write New.
│    │    ├ 5. Write Old.
│    │    └ q. quit
│    └ q. quit
│
├ 2. Clock setting.
│
└ q. quit
```

図 6-22 LPC1768XPresso mainEnvTest プロジェクトのメニュー構成

6.5.4.2 環境依存のテスト

「1. Inspection depending on environment.」は環境に依存するテスト項目です。スタック使用量、ドライバの単独テスト、パフォーマンス計測がおこなえます。

6.5.4.3 Clock setting.

LPC1768 は内蔵ペリフェラルとして RTC(Real Time Clock)を持っており、その設定をおこなうのが「2.Clock setting.」メニューです。
Clock setting は以下の手順でおこないます。

```
Now setting 2013/12/01 18:07:05
New date "yyyymmdd" =>
20131201
New time "hhmmss" =>
181600
```

①現在設定されている日付、時刻が表示される

②設定したい日付を「yyyymmdd」形式で入力し enter キーで確定する

③同様に、時刻を「hhmmss」形式で入力し enter キーで確定する

図 6-23 LPC1768XPresso Clock setting

バックアップ電源が切れているなど、初期化されていない状態で起動した場合は、2014 年 1 月 1 日、00:00:00 に自動的に設定されます。

6.5.4.4 スタック使用量について

表 6-36 LPC1768Xpresso スタック使用量

機能	解説	スタック使用量
MkDir()	MkDir() 関数でディレクトリを作る際のスタック領域の使用量です。	574バイト
DelDir() 4階層	DelDir() 関数で4階層のサブディレクトリを持つディレクトリ削除時のスタック領域の使用量です。	766バイト
DelDir() 1階層	DelDir() 関数でサブディレクトリを持たないディレクトリを削除した際のスタック領域使用量です。	614バイト
RfOpen()	新規作成モードでファイルをオープンした際のスタック領域使用量です。	606バイト
RfWrite()	10 バイトのデータを書き込んだ際のスタック領域使用量です。	494バイト
RfClose()	上記 RfWrite() 実行後の RfClose() のスタック領域使用量です。	510バイト

6.5.4.5 パフォーマンスについて

パフォーマンスは状況により変動します。計測したパフォーマンスは性能を保証するものではありません。標準的な一例としてご理解ください。

RfRead

65,536 バイトのファイルを RfRead()で2バイトずつ読み取りその時間を計測しています。

表 6-37 LPC1768Xpresso RfRead()のパフォーマンス

	～250uS	～500us	～1ms	～2ms	～4ms	～8ms	8ms～	MIN	MAX
2G CLASS6 (SanDisk)	32640	0	0	127	1	0	0	9us (2byte)	2,692us (16384byte)

RfWrite

新規ファイルの RfWrite() で2バイトずつ書き込み 65,536 バイトのファイルを作っています。計測している時間は RfWrite()の時間です。

表 6-38 LPC1768Xpresso 新規ファイル RfWrite()のパフォーマンス

	～250uS	～500us	～4ms	～8ms	～16ms	16ms～	MIN	MAX
2G CLASS6 (SanDisk)	32640	0	0	126	2	0	10us (2byte)	15,229us (28160byte)

上書き RfWrite

既存ファイルの RfWrite() の結果です。

表 6-39 LPC1768Xpresso 既存ファイル RfWrite()のパフォーマンス

	～250uS	～500us	～1ms	～2ms	～4ms	～8ms	～16ms	16ms～	MIN	MAX
2G CLASS6 (SanDisk)	32640	0	0	1	0	124	1	2	11us (4byte)	16,572us (23040byte)

6.5.5 wav ファイル再生プロジェクト(mainPlayWav)

6.5.5.1 メニュー構成

wav ファイル再生サンプルプロジェクトのメニュー構成です。

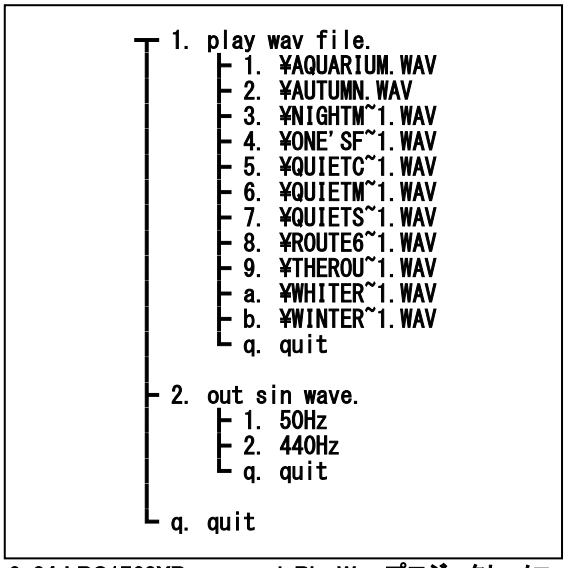

図 6-24 LPC1768XPresso mainPlayWav プロジェクト メニュー構成

6.5.5.2 play wav file メニュー

SD カードのルートディレクトリにある、拡張子が wev のファイルを表示します。

ファイル名のメニュー項目番号を入力するとそのファイルを再生します。

ここに表示されているファイルはテストを実行した際の物です。これらのサンプルファイルもテストデータとして提供しており、以下 URL からダウンロードできます。ファイル内容については「10.サンプル wav ファイル」を参照してください。

http://homepage3.nifty.com/resove/fileSystem

6.5.5.3 out sin wave メニュー

テストソースコード内部で持っている 50Hz と 440Hz の sin 波形を出力します。

6.5.5.4 wev ファイル仕様

wev ファイルは 11,025Hz サンプリング、ステレオ、8 ビット分解能のものが再生できます。

6.5.5.5 ソフトウェアの仕組み

PWM を使用して音声を再生しています。パルスのデューティー比を変更しその平均電圧を音とする仕組みです。
同じデューティーのパルスを 5 回づつ出力するためパルスの周波数を 11,025Hz の 5 倍の 55,125Hz としています。
出力データを左右チャネルの PWM レジスタに格納するとともに PWM のラッチレジスタ、LER への書き込みをおこなうため、3 つの DMA チャネルを同時に使用して実現しています。DMA の動作タイミングはすべて 11,025Hz です。

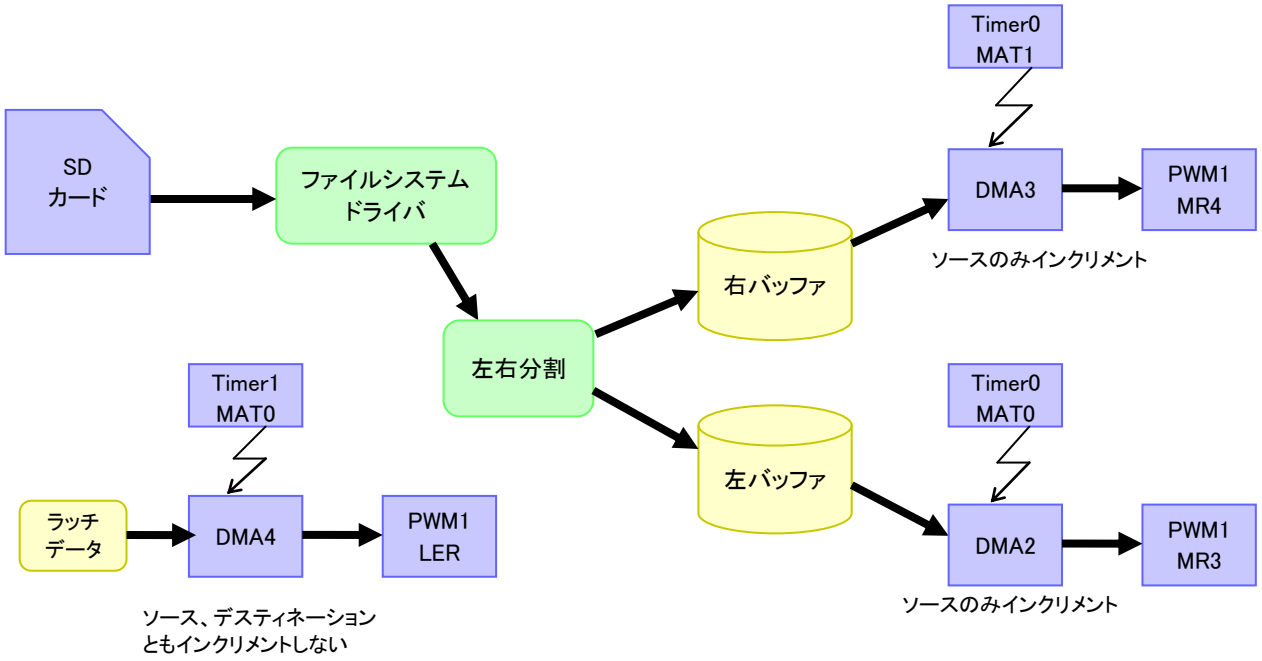

図 6-25 LPC1768XPresso PlayWav 仕組み

ファイルの読み取りは 512 バイトのセクタ毎に物理的なアクセスが発生し時間がかかってしまいます。読み取り速度をなるべく均一にするため wav ファイルの読み取りも 512 バイト単位でおこなっています。512 バイトのバッファを2つ用意し片方のバッファの DMA 転送中にもう片方のバッファに wav ファイルから読み込んでいます。
11,025Hz のサンプリングスピードで 512 バイト(2ch ぶん、256sample)の PWM 出力では、256sample÷11,025Hz ≒ 23ms の時間がかかります。「6.5.4.2 環境依存のテスト」の調査結果でセクタの読み取りが発生する際には 2ms 以内、FAT の読み取りが発生する際でも 4ms 以内に読み取りが終了するので十分間に合うと考えられます。

mainPlayWav プロジェクトでは OS を使用し複数タスクで実現しています。タスク間の同期方法を示します。

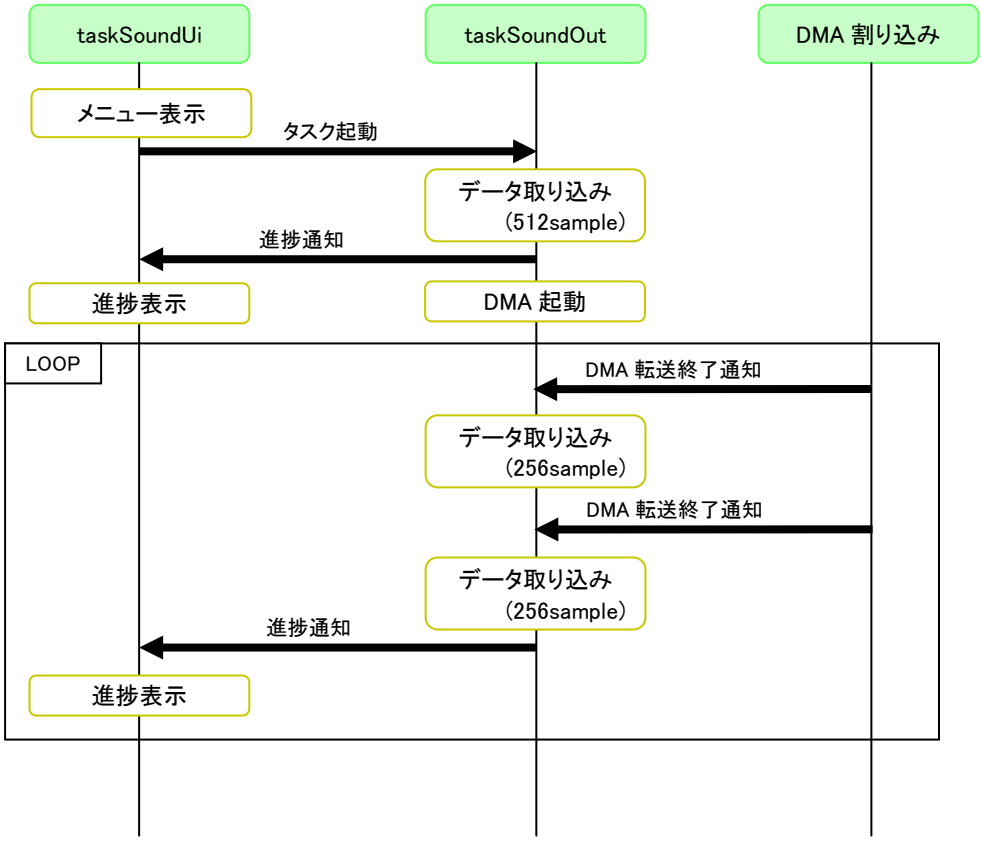

図 6-26 LPC1768Xpresso PlayWav タスク間同期

6.5.6 ファイルシステムテストプロジェクト(mainRfTest)
6.5.6.1 メニュー構成

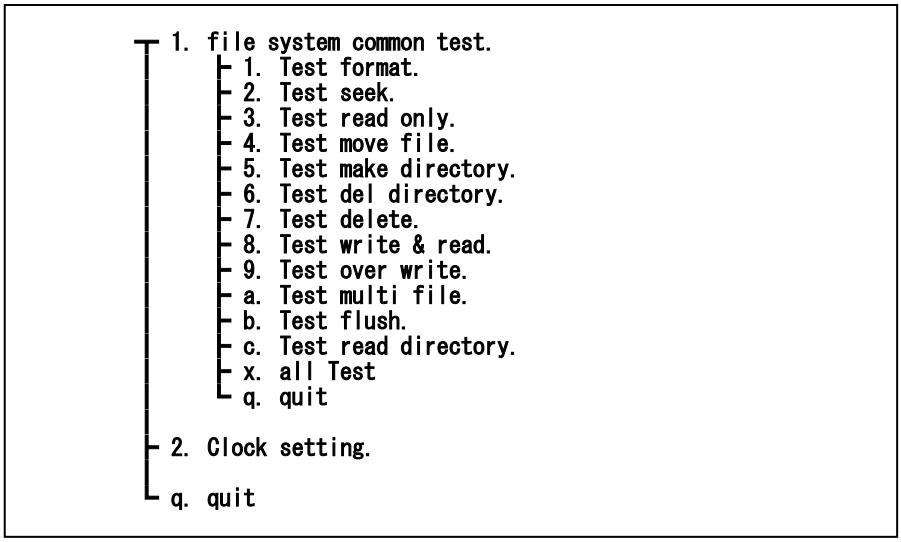

図 6-27 LPC1768XPresso mainRfTest プロジェクトメニュー

6.5.6.2 file system common test.

ファイルシステムの共通テストです。内容については「5.1 テスト機能ソース」を参照してください。

6.6 RsfVcpp

6.6.1 概要

Windows 上で動作確認をおこなうための環境です。シミュレーションするドライバを実装しファイルシステムのデバッグをおこなうことを可能としました。Visual Studio 2010 のプロジェクトファイルを同梱しています。Express Edition でビルド可能な作りにしています。

ターゲットのハードウェアがなくてもシミュレーションドライバとファイルシステムを動かす環境があるのでソフトウェアのデバッグを先に進めることが可能となります。

6.6.2 フォルダ構成

フォルダ構成を以下に示します。

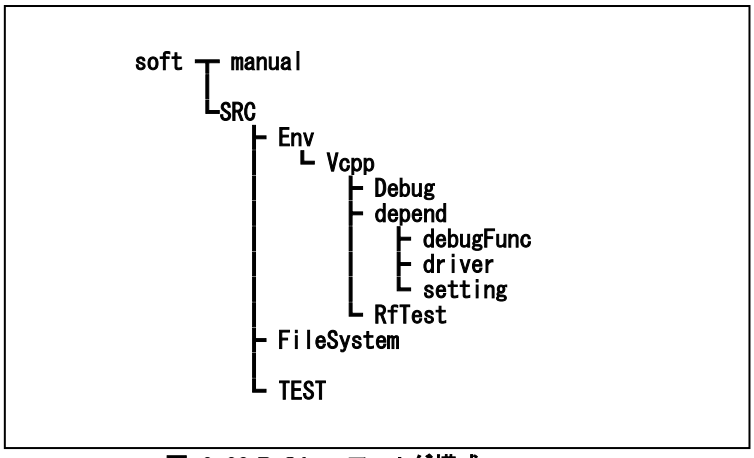

図 6-28 RsfVcpp フォルダ構成

6.6.3 ファイル一覧

表 6-40 Vcpp ファイル一覧

フォルダ	ファイル名	概要
manual	rsfManual.pdf	ファイルシステムのマニュアル
SRC		ソースコード
Env¥Vcpp	RfTest.sln	Visual studio のプロジェクトファイル
RfTest	RfTest.cpp	メイン
	RfTest.vcxproj	Visual studio が生成したプロジェクトファイル
	stdafx.cpp(.h)	Visual studio が生成したプリコンパイル済みヘッダ
	targetver.h	Visual studio が生成したプラットフォームバージョン
Debug	RfTest.exe	実行ファイル
depend		環境依存パッケージ
DebugFunc	TraceOut.c	テスト用入出力
	TstOrgErrCode.c(.h)	ドライバエラーコード表示機能
Drver	cardAccess.cpp(.h)	メディアに直接アクセスするドライバ
	dateTime.cpp(.h)	現在時刻取得機能
	deviceSearch.cpp(.h)	デバイス検索部
	DrvErrCode.h	SD カードドライバ
	DrvSdCard.cpp(.h)	ファイルシステムドライバインタフェース
	letter.cpp(.h)	文字列操作
setting	rfSettings.c	ファイルシステムの設定ファイル
FileSystem	…	ファイルシステム。詳細は「2.ファイルシステム」を参照してください。
TEST	…	ファイルシステムのテストプログラム。詳細は「5.テストプログラム」を参照してください。

6.6.4 動作環境

Windows のコンソールアプリケーションとして動作します。メニュー表示と指示はコンソールに、テスト結果は Visual Studio のトレース出力に表示されます。そのためデバッグモードで実行しないと結果を見ることができません。実行結果をコンソール画面に表示すると表示行数の制限により古い結果が見えなくなってしまうためにそのようにしています。

また、物理デバイスに直接アクセスするため管理者権限で実行しなければなりません。Windows vista や Windows7 では Visual Studio を管理者権限で起動します。

一部 API が使用できないため Windows のバージョンをチェックしています。Windows 2000 以降で実行できます。

6.6.5 ビルド

Visula Studio を管理者権限で起動します。

[開く]-[プロジェクト／ソリューション]から Vcpp フォルダの RfTest.sln を開き、ビルドをおこないます。

6.6.6 操作方法

「F5」キーを押してデバッグモードで実行を開始します。
コンソール画面が表示されます。

```
注意!!
    管理者権限で起動されていません。
    visual studio を管理者として実行してください。

何かキーを押せば終了します。
```

図 6-29 管理者権限の注意

管理者権限で起動されていないときには上記のメッセージが表示されます。キーボード入力をおこなうと終了します。

```
注意!!
    windows 上で format のテストをした際には、終了時にいったん
  メディアを取り出して再度挿入してください。
    正しく認識できなくなることがあります。

何かキーを押してください。
```

図 6-30 format 実行時の注意

管理者権限で起動されている際には、最初に format 実行時の注意が表示されます。format 実行後は Windows が認識しているメディアの情報と実際のフォーマットが異なることがあり、そのまま使用すると矛盾が生じてうまく認識できなくなることがあるので注意してください。
何かしらのキーを押すと先に進みます。

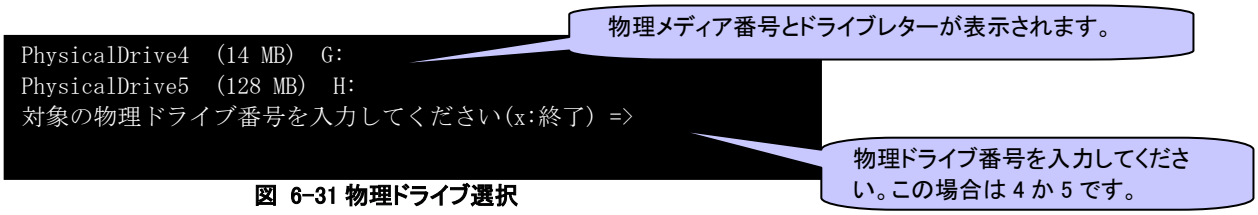

図 6-31 物理ドライブ選択

使用可能なリムーバブルメディアの一覧が表示されます。
テスト対象の物理デバイス番号を入力してください。

```
PhysicalDrive4 を書き換えます。よろしいですか？(y/n) =>
```

図 6-32 物理メディア書き換えの再確認

再確認のメッセージが表示されます。間違いがなければ「y」と入力し先に進んでください。

```
************************************************************
Vcpp : System version. Build(Sep 14 2014 09:10:10)
************************************************************
FileSystem version = 0102
Driver version     = 0003
Driver type        = Vcpp simuration.


************************************************************
Test Initialize.
************************************************************
 OK open[NG case]
 OK RfInitFileSystem()
 OK open
 OK conclude
 OK open[NG case]
```

図 6-33 デバッグ出力

バージョン番号と初期化のテストが実施され結果がデバッグ出力に表示されます。

標準出力にはテストケースの選択メニューが表示されます。

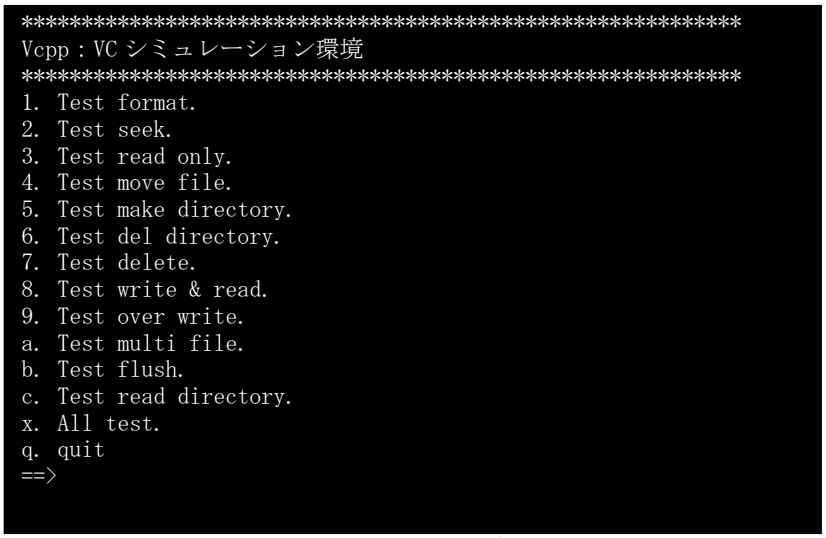

図 6-34 テストケース選択

6.6.7 注意事項

テスト対象のドライブを書き換えます。
元の内容は破壊されますので慎重に実行してください。

7 サンプル SD カードドライバ

ドライバの例として「AkiH8SCI」動作環境のドライバについて説明します。
このドライバは SD カードのドライバで CPU は H/3048F で動作し、SD カードとは SPI モードで通信をします。

7.1 ドライバ構造

ドライバはドライバインタフェース層と物理層とに分かれています。

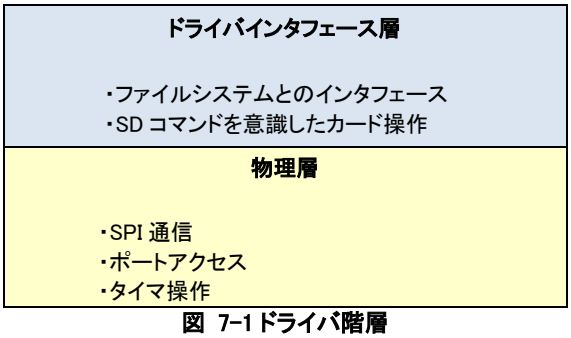

図 7-1 ドライバ階層

7.1.1 ファイル一覧

表 7-1 ドライバファイル一覧

階層	ソースファイル	ファイル内容
ドライバインタフェース層	DrvSdCard.c	ドライバインタフェース。SD カードのコマンドを意識した I/O をおこなう。
物理層	DrvCpuXtal.h	CPU クロック定義
	DrvSdSciItu.c	H8/3048 のペリフェラル、SCI、ポート、ITU(タイマ)アクセスの物理層

7.1.2 ドライバインタフェース層

ドライバインタフェース層はファイルシステムが規定している関数インタフェースを備えた関数群です。SD カードとの通信コマンドを組み立て、カードとのやり取りをおこないます。

7.1.3 物理層

物理層は直接ハードウェアへのアクセスをおこなう部分です。H8/3048 専用のドライバです。H8/3048 のペリフェラルである SCI、ITU、ポートを使用して SPI 通信、ポート状況の取得、設定、時間指定の待ちをおこないます。

7.2 ドライバインタフェース

表 7-2 サンプルドライバインタフェース関数一覧

分類	関数名	概要
初期化	InitSdCard()	カードを挿入したら一回だけおこなう。
書き込み	WriteSdPhysical()	書き込む。
	FillSdOneBlock()	同一のデータで1ブロック書き込む。
読み取り	ReadSdPhysical()	読み取る。
状況取得	GetSdBlockSize()	ブロックサイズを取得する。
	GetSdCardSize()	カード容量を取得する。
	GetSdDrvVer()	ドライババージョン情報を取得する

7.3 ドライバインタフェース関数内容説明

関数インタフェースは「4.ドライバ機能」で規定している通りです。ここでは関数の戻り値、内部動作について説明します。

7.3.1 InitSdCard

戻り値

表 7-3 InitSdCard 戻り値

戻り値定義	意味
RF_OK	正常終了
RF_ERR_INIT_RESET_TIMEOUT	リセットコマンドタイムアウト
RF_ERR_INIT_INIT_TIMEOUT	初期化プロセスタイムアウト
RF_ERR_CSD_IN_IDLE_STATE	アイドルステート中（SD カードからのレスポンス）
RF_ERR_CSD_ERASE_RESET	消去リセット中（SD カードからのレスポンス）
RF_ERR_CSD_ILLEGAL_COMMAND	コマンドエラー（SD カードからのレスポンス）
RF_ERR_CSD_COM_CRC_ERROR	CRC エラー（SD カードからのレスポンス）
RF_ERR_CSD_ERASE_SEQ_ERROR	消去シーケンスエラー（SD カードからのレスポンス）
RF_ERR_CSD_ADDRESS_ERROR	アドレスエラー（SD カードからのレスポンス）
RF_ERR_CSD_PARAMETER_ERROR	パラメータエラー（SD カードからのレスポンス）
RF_ERR_CSD_DATA_RCV_TIMEOUT	データ受信タイムアウト
RF_ERR_CSD_RESPONSE_TIMEOUT	コマンドレスポンスタイムアウト
RF_ERR_CSD_TOKEN_RCV_TIMEOUT	トークン受信タイムアウト
RF_ERR_CSD_CHECK_BIT	CSD 内チェックビット不正

処理内容

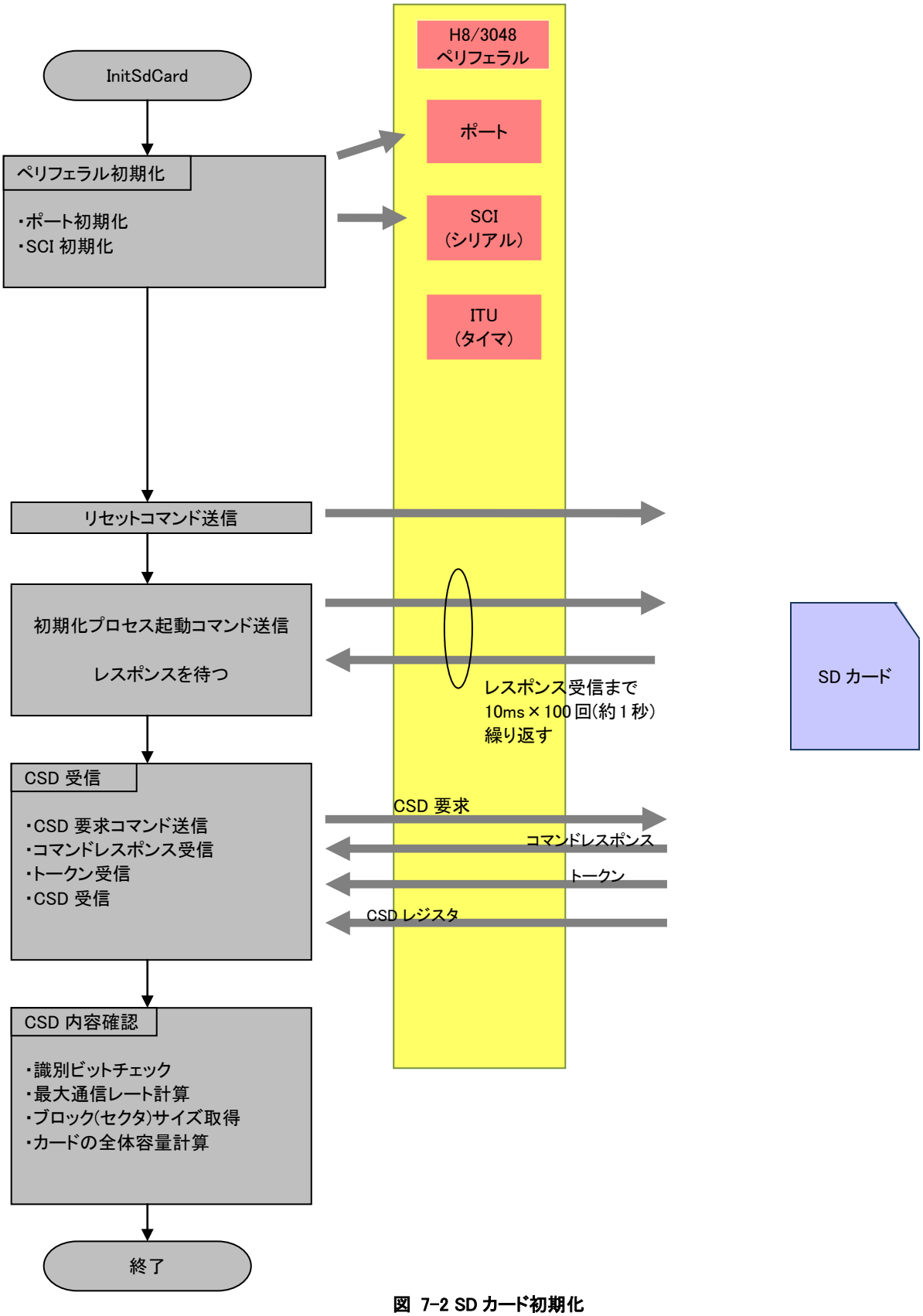

図 7-2 SD カード初期化

ハードウェアの初期化をおこないます。SD カードの初期化とカードの基本情報を取得し、後のアクセスの準備をおこないます。

7.3.2 WriteSdPhysical

戻り値

表 7-4 WriteSdPhysical 戻り値

戻り値定義	意味
RF_OK	正常終了
RF_ERR_BUFF_ADDRESS	アドレスがブロックの先頭と合っていない 要求しているデータがディスク内に収まっていない
RF_ERR_BUFF_LEN	要求サイズがブロックの区切りと合わない 要求サイズが1ブロック未満である
RF_ERR_WRITE_TIMEOUT	書き込みタイムアウト
RF_ERR_WRITE_CRC_ERR	書き込み時 CRC エラー （データに対する SD カードからのレスポンス）
RF_ERR_WRITE_PHYSICAL	書き込みの物理的エラー（データに対する SD カードからのレスポンス）
RF_ERR_WRITE_IN_IDLE_STATE	アイドルステート中である （コマンドに対する SD カードからのレスポンス）
RF_ERR_WRITE_ERASE_RESET	消去、初期化中である （コマンドに対する SD カードからのレスポンス）
RF_ERR_WRITE_ILLEGAL_COMMAND	コマンドエラー （コマンドに対する SD カードからのレスポンス）
RF_ERR_WRITE_COM_CRC_ERROR	CRC エラー （コマンドに対する SD カードからのレスポンス）
RF_ERR_WRITE_ERASE_SEQ_ERROR	消去シーケンス中である （コマンドに対する SD カードからのレスポンス）
RF_ERR_WRITE_ADDRESS_ERROR	アドレスエラー （コマンドに対する SD カードからのレスポンス）
RF_ERR_WRITE_PARAMETER_ERROR	コマンドパラメータエラー （コマンドに対する SD カードからのレスポンス）

処理内容

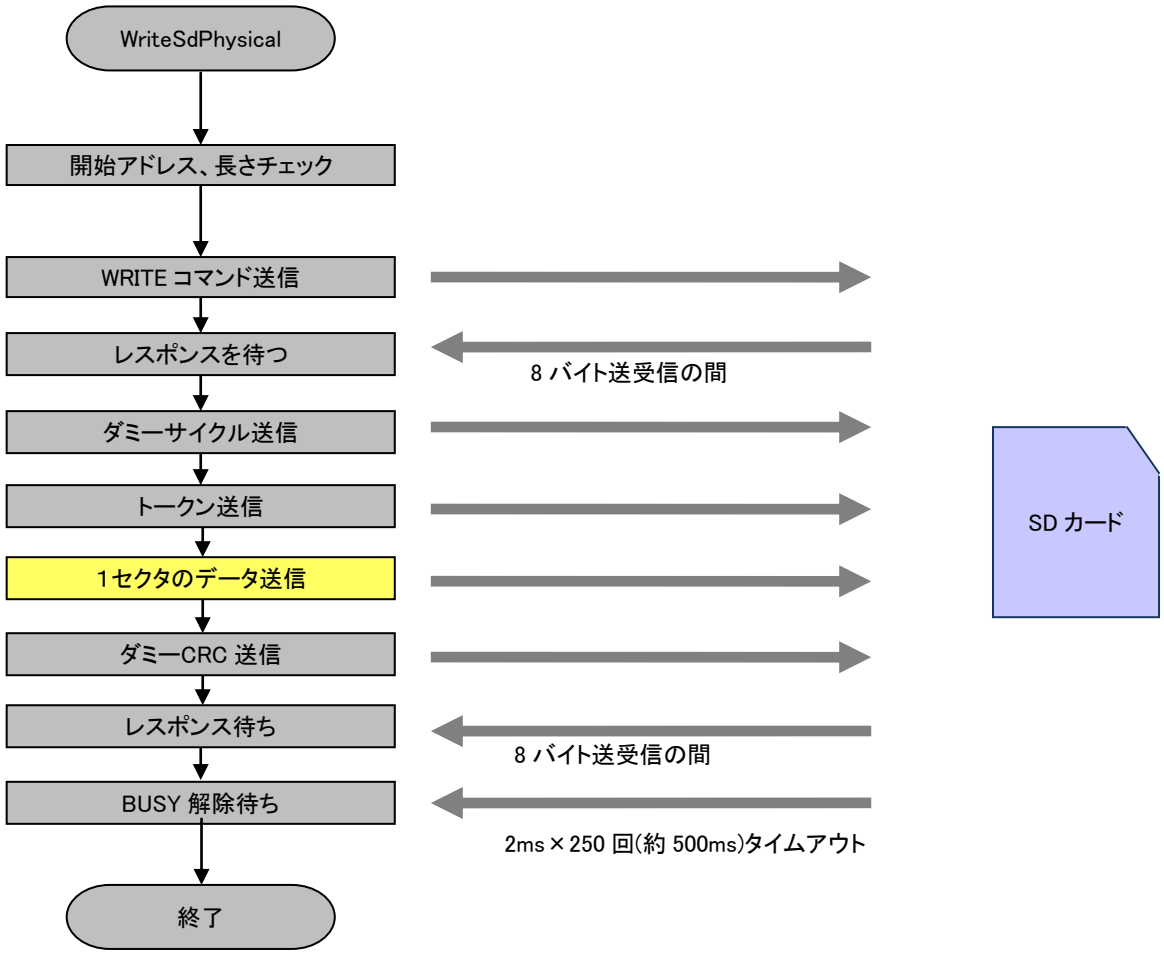

図 7-3 WriteSdPhysical 動作

Write コマンドを送信しレスポンスを待った後1セクタぶんのデータを送信します。

7.3.3 FillSdOneBlock

WriteSdPhysical と動作、戻り値とも同じです。ただし、「1セクタのデータ送信」ではバッファを使用せずに同一のデータを送ります。

7.3.4 ReadSdPhysical

戻り値

表 7-5 ReadSdPhysical 戻り値

戻り値定義	意味
RF_OK	正常終了
RF_ERR_BUFF_INITIAL	初期化が終了していない
RF_ERR_BUFF_ADDRESS	要求しているデータ位置がディスク内に収まっていない
RF_ERR_BUFF_LEN	要求サイズが 0 である
RF_ERR_RESPONSE_TIMEOUT	コマンドレスポンス受信タイムアウト
RF_ERR_DATA_RECEIVE_TIMEOUT	データ受信タイムアウト
RF_ERR_TOKEN_RECEIVE_TIMEOUT	トークン受信タイムアウト
RF_ERR_READ_IN_IDLE_STATE	読み込みの物理的エラー （コマンドに対する SD カードからのレスポンス）
RF_ERR_READ_ERASE_RESET	消去、リセット中である （コマンドに対する SD カードからのレスポンス）
RF_ERR_READ_ILLEGAL_COMMAND	コマンドエラー （コマンドに対する SD カードからのレスポンス）
RF_ERR_READ_COM_CRC_ERROR	CRC エラー （コマンドに対する SD カードからのレスポンス）
RF_ERR_READ_ERASE_SEQ_ERROR	消去中であるエラー （コマンドに対する SD カードからのレスポンス）
RF_ERR_READ_ADDRESS_ERROR	アドレスエラー （コマンドに対する SD カードからのレスポンス）
RF_ERR_READ_PARAMETER_ERROR	パラメータエラー （コマンドに対する SD カードからのレスポンス）

処理内容

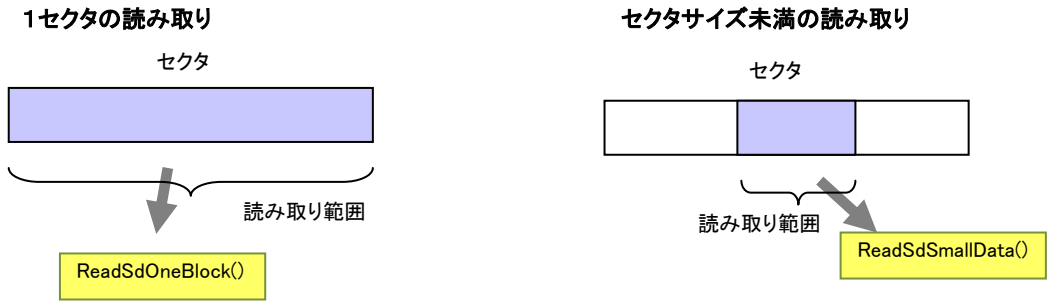

図 7-4 読み取りのパターン

ReadSdPhysical()は図のように1セクタの読み取りと1セクタ未満の一部の読み取りがおこなえます。
セクタサイズ未満の読み取りは ReadSdSmallData()関数が行い、セクタ単位の読み取りは ReadSdOneBlock()関数がおこないます。
ハード的にセクタサイズ未満の読み取りがおこなえないような場合はエラーコードを返すように実装してください。

1セクタ読み取り動作(ReadSdOneBlock)

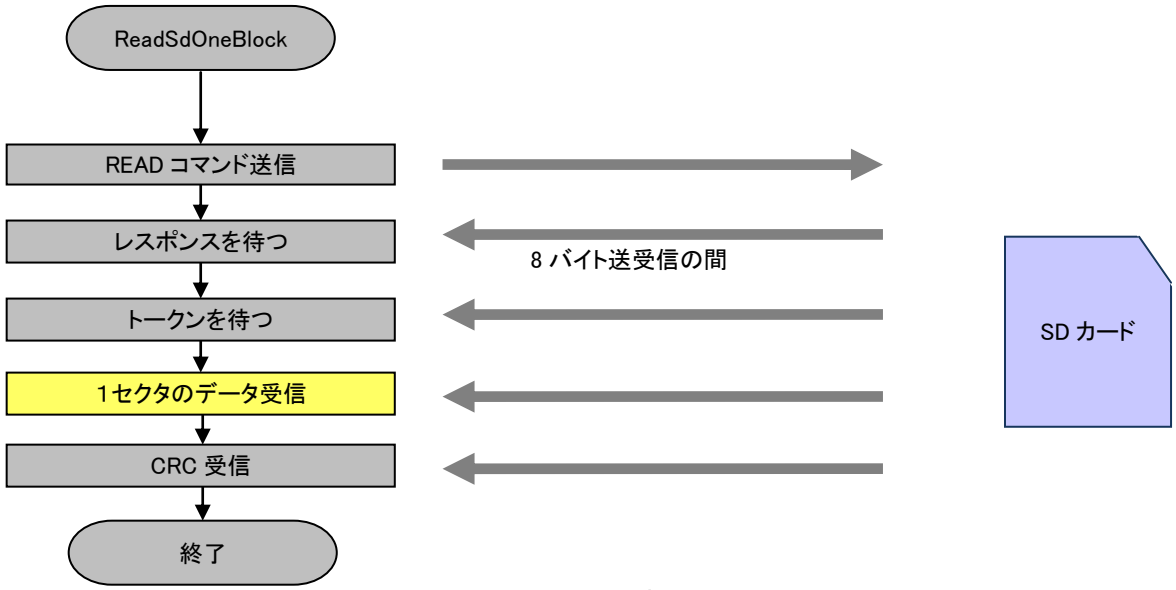

図 7-5 セクタ読み取り動作

Read コマンド送信後1セクタぶんのデータを受信します。

セクタの一部を読み取る(ReadSdSmallData)

セクタの一部を読み取る際には上の図の「1セクタのデータ受信」で不要な部分を読み捨て必要な部分のみをバッファに格納します。

7.3.5 GetSdBlockSize

戻り値

ブロックサイズ。ブロックサイズが計算されていないときは 0 を返す。

処理内容

CSD レジスタから計算したブロックサイズ(セクタサイズ)を返します。

7.3.6 GetSdCardSize

戻り値

ディスクの全体サイズを返す。デバイスサイズが計算されていないときは 0 を返す。

処理内容

CSD レジスタから計算したデバイス全体のサイズを返します。

7.3.7 GetSdDrvVer

戻り値

常に RF_OK。

処理内容

引数で指定された T_RF_DRV_VER のポインタが指す領域にドライバのバージョン情報を設定します。

7.4 物理層関数一覧

表 7-6 物理層関数一覧

分類	関数名	概要
初期化	InitSdTrans()	ペリフェラルの初期化をおこなう
SPI 通信	SendSdOne()	1バイト送受信
	RecvSdBlock()	配列の受信と固定データの送信をおこなう
	SendSdBlock()	配列の送信をおこなう(受信データは取得しない)
	SendSdRepeat()	固定データの連続送信をおこなう(受信データは取得しない)
	ChangeSdRate()	通信レートを変更する
ポートアクセス	InitSdPort()	ポートの入出力を設定する
	GetSdInsert()	SD カードの挿入状態を取得する
	GetSdProtect()	SD カードが書込み禁止状態かを取得する
	SetSdOnChipsel()	SD カードのチップセレクトの ON/OFF を設定する
ITU アクセス	InitSdWait()	タイマの初期化
	SdWaitTimer()	ms 単位の指定時間の待ち

物理層の関数インタフェースについてはファイルシステムとして規定していません。サンプルドライバの一部として物理層の関数インタフェース、内部処理などについて説明します。

SD カード用のドライバを他のハードウェアに移植する場合はドライバインタフェースをそのまま使用して、物理層の改造だけをおこなうことによりインプリメント可能となることを目指して階層分けしています。

7.5 物理層関数リファレンス

7.5.1 InitSdTrans

書式

```
void InitSdTrans( void );
```

概要

SCI をクロック同期式で動作させるための初期化をおこなう

パラメータ

なし

戻り値

なし

解説

H8/3048 の SCI の初期化をおこないます。

7.5.2 SendSdOne

書式

```
unsigned short SendSdOne( unsigned char transmitData );
```

概要

1バイト送受信

パラメータ

transmitData

送信するデータ

戻り値

SD_DRV_ERR_VAL のときは受信エラー
その他の時は受信したデータ

解説

1バイトの送信と受信を同時におこないます。
送信前に SCI のステータスをチェックし送信可能でなければ SD_DRV_ERR_VAL を返します。

7.5.3 RecvSdBlock

書式

```
unsigned short RecvSdBlock(     unsigned char * receiveBuff,
                                unsigned char transData,
                                unsigned short len
                                );
```

概要

連続データの受信

パラメータ

receiveBuff

受信したデータを格納する領域

transData

送信するデータ

len

送受信するデータ長

戻り値

実際に受信したバイト数

解説

len で指定した回数の送信と同長さの受信をおこないます。送信は transData を繰り返し送信します。
送受信が完了するまで関数を抜けません。

7.5.4 SendSdBlock

書式

```
unsigned short SendSdBlock(     unsigned char * transBuff,
                                unsigned short len
                                );
```

概要

連続データの送信

パラメータ

transBuff

送信するデータが格納されている領域

len

送信するデータ長

戻り値

実際に受信したバイト数

解説

内部的には送信と受信を同時におこなっていて実際に受信したバイト数を戻り値として返します。
transBuff の内容を len で指定した長さぶん送信します。
受信したデータは読み捨てます。

7.5.5 SendSdRepeat

書式

```
unsigned short SendSdRepeat(    unsigned char transData,
                                unsigned short len
                                );
```

概要

同一データの繰り返し送信

パラメータ

transData

送信するデータ

len

送信するデータ長

戻り値

実際に受信したバイト数

解説

transData で指定したデータを繰り返し送信します。
受信したデータは読み捨てます。

7.5.6 ChangeSdRate

書式

```
unsigned short ChangeSdRate( unsigned long rate );
```

概要

通信レートの変更

パラメータ

rate

通信レート。bps の通信レートを直接指定する。

戻り値

TRUE:成功
FALSE:失敗

解説

rate から SCI のレジスタに設定できる値を計算し設定します。設定できる範囲外の時は FALSE を返します。

7.5.7 InitSdPort

書式

```
void InitSdPort();
```

概要

ポートの初期化

パラメータ

なし

戻り値

なし

解説

使用するポートの入出力方向、初期値を設定します。

7.5.8 GetSdInsert

書式

```
short GetSdInsert();
```

概要

カード挿入状況の取得

パラメータ

なし

戻り値

TRUE ：カードが挿入されている
FALSE:カードが挿入されていない

解説

カードの認識ポートを読み取り状況を返します。

7.5.9 GetSdProtect

書式

```
short GetSdProtect( void );
```

概要

カードプロテクト状況の取得

パラメータ

なし

戻り値

TRUE :カードは書込み禁止である
FALSE:書込み禁止ではない

解説

カードが挿入されていない場合は TRUE を返します。
カードが挿入されている時はプロテクトのポートを読み取りその状況を返します。

7.5.10 SetSdOnChipsel

書式

```
void SetSdOnChipsel( short OnOff );
```

概要

SPI のチップセレクトを設定する

パラメータ

OnOff

TRUE :チップセレクトを有効とする
FALSE:チップセレクトを無効とする

戻り値

なし

解説

SPI モードの CS 信号を制御します。
H8/3048 では SCI に SPI モードの機能は無いため CS の設定はソフトウェアでおこないます。
通信をおこなう直前に CS を有効にし、通信が終了した後 CS を無効にします。

7.5.11 InitSdWait

書式

```
void InitSdWait( void );
```

概要

タイマ初期化

パラメータ

なし

戻り値

なし

解説

1ms の時間測定をおこないやすいようにタイマの設定をおこないます。SdWaitTimer()を実行する前に一回だけおこないます。

7.5.12 SdWaitTimer

書式

```
void dfpWaitMsTimer( short time );
```

概要

ms 単位の時間の待ちをおこないます

パラメータ

time

ms 単位の待ち時間

戻り値

なし

解説

time で指定した ms 単位の時間が経過するのを待ちます。

8 リアルタイム OS 下での使用

リアルタイム OS 下でファイルシステムを利用するためにはドライバ部分の作りを工夫する必要があります。

ドライバで待ちが発生した際にサンプルドライバはその場で状態が変化するまでループする作りになっています。そのため、物理的な I/O にかかる時間より早い間隔で処理する必要のあるタスクはファイルへのアクセスをおこなうタスクよりも優先度を高くする必要があります。

タスク優先度をファイルシステムを使用するか否かに大きく依存しないようにするにはドライバの待ちの仕組みを変更する必要があります。ドライバ内で何かの事象を待っている部分を OS のシステムコールを利用して待つように変更します。事象の変化を割り込みや別のタスクから通知するようにします。そのような変更をおこなうことにより、事象を待ち始めてから事象が発生するまでの時間を他のタスクに明け渡すことができます。

8.1 システムコールのオーバーヘッド

ただし、AkiH8SCI 環境のドライバを使用する際には注意が必要です。SCI の送受信はすべて1バイトごとにおこなっています。1バイトの送信にかかる時間を計算してみると 100Kbps では 80μs 、1Mbps で 8μs となります。この時間を OS のシステムコールを使用して別のタスクに明け渡してもパフォーマンス向上の効果は薄いと考えられます。OS のシステムコールを実行するオーバーヘッドがあるためです。

使用する OS や CPU の速度により異なるので正確なことは言えませんが、16MHz で動作する AKI-H8/3048F の場合、システムコールのオーバーヘッドとタスク切り替えにかかる時間は 30μs を超えます。一旦別のタスクに制御を明け渡し、再度自タスクに戻ってくるので2回のタスク切り替えが発生します。2回のタスク切り替えで 60μs の時間がかかってしまうので 1Mbps の場合は OS のシステムコールを使って同期を取るとパフォーマンスを圧迫してしまうことになります。この場合はむしろ通信が終わるまでループをして待っていた方がパフォーマンスが良い、ということになります。

AkiH8DMA 環境の場合は DMA を使って連続的に送受信をおこなっています。例えば 1 セクタのデータ送信をする場合には 512×8μs ≒4ms の時間がかかります。この時間を OS のシステムコールを使って他のタスクに明け渡すことはパフォーマンス向上に有効な手段になります。

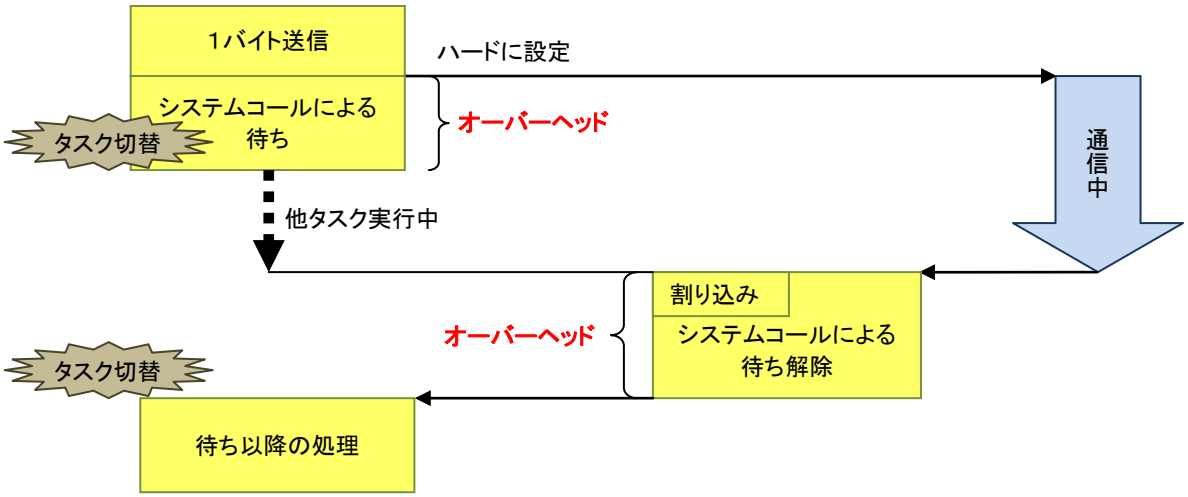

図 8-1 OS のシステムコールを使用することによるオーバーヘッド

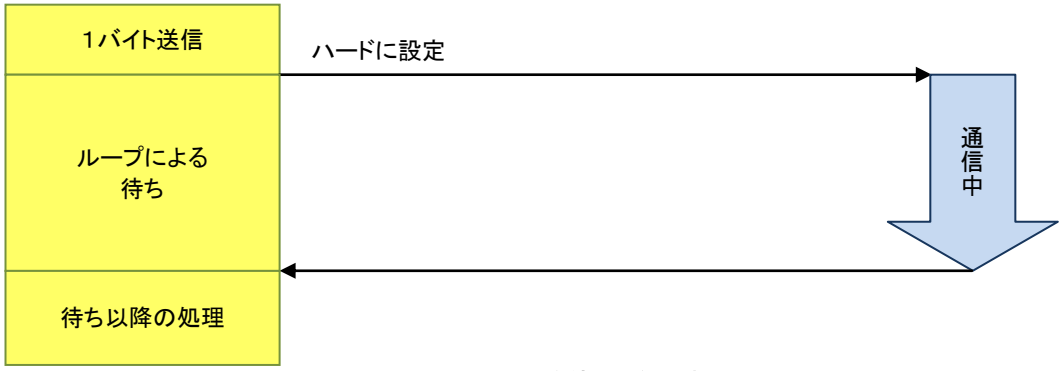

図 8-2 OS のシステムコールを使用しない待ち

8.2 処理時間のばらつき

「6.1.12.パフォーマンスについて」のパフォーマンス計測結果で分かるように、物理的な書き込みが発生する時とそうでない時では処理時間に大きな差があります。

例えば 4ms ごとにデータを取得しそれをファイルに落とすことを考えてみます。

データ取得のタイミングはタイマや OS のシステムコールを使用して正確に通知されます。図のフローのように通知を受けて、データを取得し、書き込みをおこなうという流れで処理をおこなうのですが、物理的書き込みが発生する際には恐らく 4ms を超えてしまいます。

物理的な I/O にかかる時間は CPU の性能やハードウェアの制約、また、使用されるメディアなどにより異なります。SD カードの場合は接続されるカードによっても異なるのでソフトウェア実装の際には時間がかかってもよいような作りにしておく必要があります。

物理的な I/O にかかる時間はシステムの設計にもかかわってくるので実測しておいてください。

データ取得とファイルへの書き込みはタスクを分け、データはバッファリングするべきです。

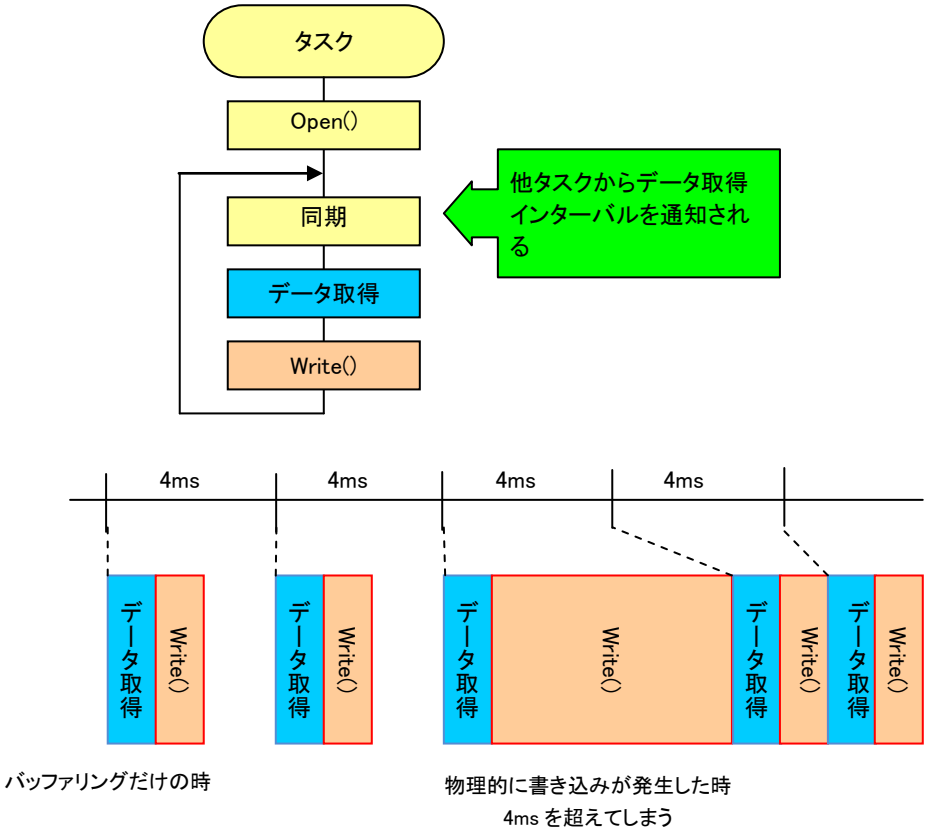

図 8-3 書き込み時間のばらつき

8.3 OS 使用の例

LPC1768Xpresso パッケージの mainPlayWave プロジェクトはリアルタイム OS を使用しています。ドライバ内部で OS の機能を使用して同期を取っている部分について説明します。

OS に依存する部分は DrvSdSynchro.c にまとめています。

表 8-1 DrvSdSynchro.c の機能

関数	解説
initSdTransfer	送受信完了の同期を取る機能の初期化。 セマフォの資源数を 0 にしている。
waitSdTransfer	送受信が完了するのを待つ。OS 無しの場合はこの関数内でポーリングして転送完了を待っていた。OS 実装の際はセマフォ解放を待つ。
notifySdTransfer	送受信の完了を通知する。DMA 転送完了の割り込みから実行して waitSdTransfer に転送終了を通知する。セマフォを解放する。
InitSdWait	時間指定の遅延機能の初期化。特に初期化は必要ないが OS を使用しない場合の機能との互換性のために初期化関数を用意している。
SdWaitTimer	時間指定の遅延。OS の遅延機能を使用して ms 単位の時間指定での遅延をおこなう。

同期を取るのに OS の機能を使用しなかった部分を以下にまとめます。いずれもごくわずかな時間なので OS のシステムコールを使用するオーバーヘッドがデメリットと考え OS の機能は使用していません。

表 8-2 OS の機能を使用しなかった部分

関数	解説
SendSdOne	1バイトの送受信をおこなっているがごくわずかな時間なので OS のシステムコールは使用せずポーリングでおこなっている。
ChangeSdRate	通信レートを変更した後に1ビット分の待ちを入れているがこれも短時間なのでループでおこなっている。

ファイルシステム全体がリエントラント(再入可能)ではないのですが排他処理はおこなっていません。このサンプルではファイルシステムを使用するタスクが1つなので問題は起きません。複数タスクからファイルシステムを使用する場合はファイルシステムの機能全体を排他処理する必要があります。

9 テスト用ハードウェア
9.1 拡張ボード

9.1.1 SD カード基板回路図

AKI-H8 ボードに SD カードを接続するための基板の回路図です。

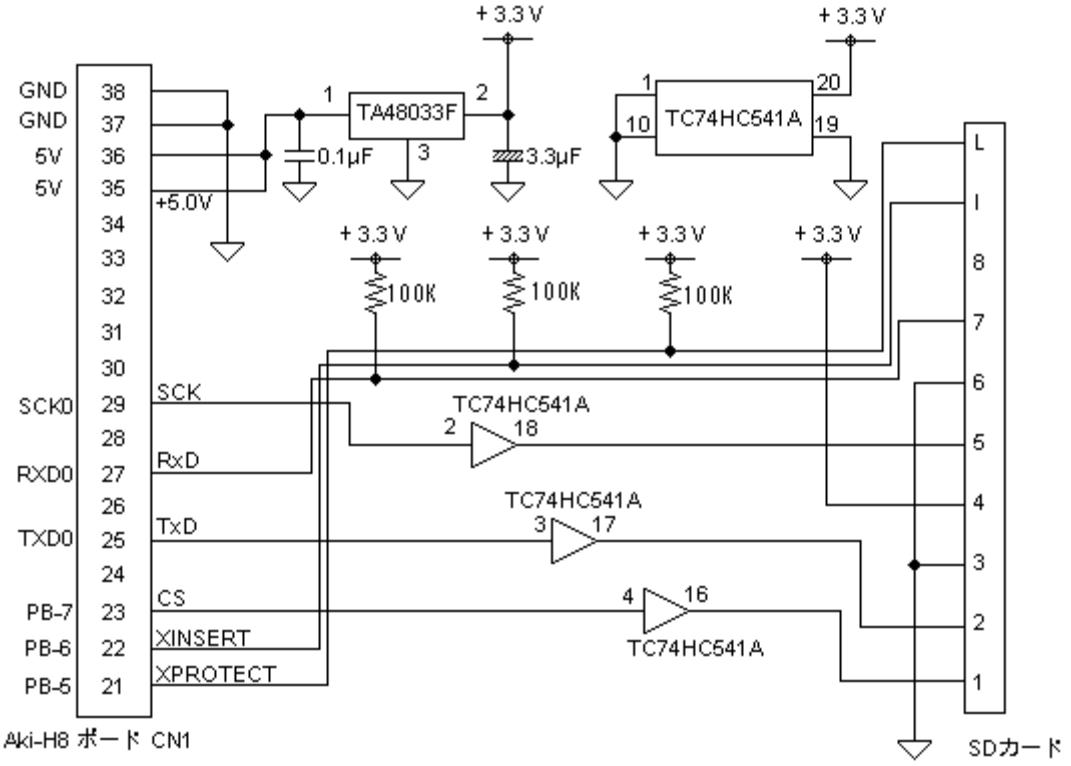

図 9-1 AKI-H8 用 SD カード基板回路図

9.1.2 音声出力部回路図

PWM や DA の出力からスピーカーを鳴らすために以下の回路を左右チャネル分取りつけています。AKI-H8 ボード、LPC1768Xplorer ボードとも同じ拡張基板を取り付けます。

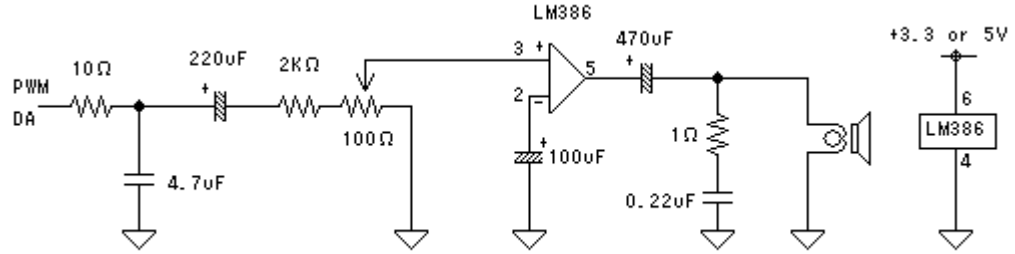

図 9-2 音声出力部回路図

9.2 ボード画像

9.2.1 AKI-H8 ボード

AKI-H8 ボードに SD カード基板、音声出力部を接続したデバッグ用ハードウェアです。
AKI-H8 ボードは秋月電子通商®のボードです。AkiH8SCI、AkiH8DMA、HEW_DMA、gcc_H8 環境はこのボードで動作します。

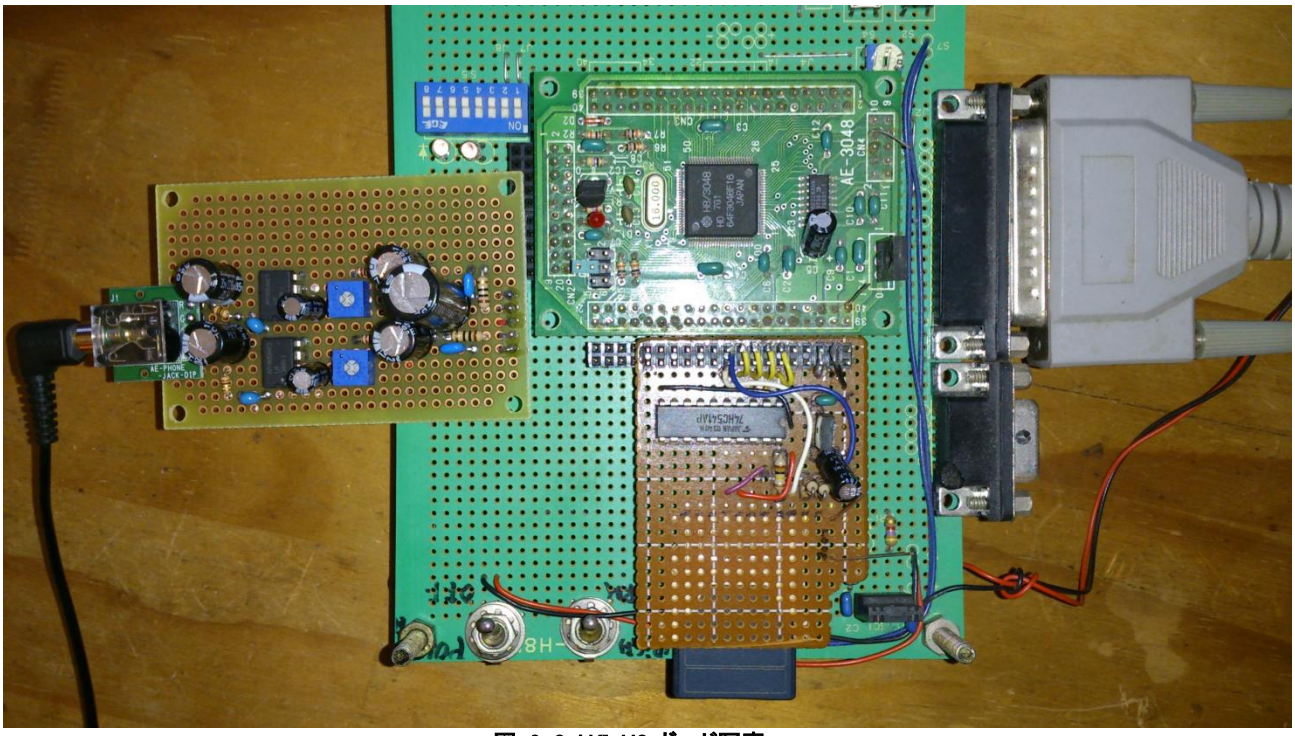

図 9-3 AKI-H8 ボード写真

9.2.2 LPC1768Xplorer ボード

NGX TECHNOLOGIES®の LPC1768Xplorer ボードです。音声出力部を拡張したデバッグ用ハードウェアです。

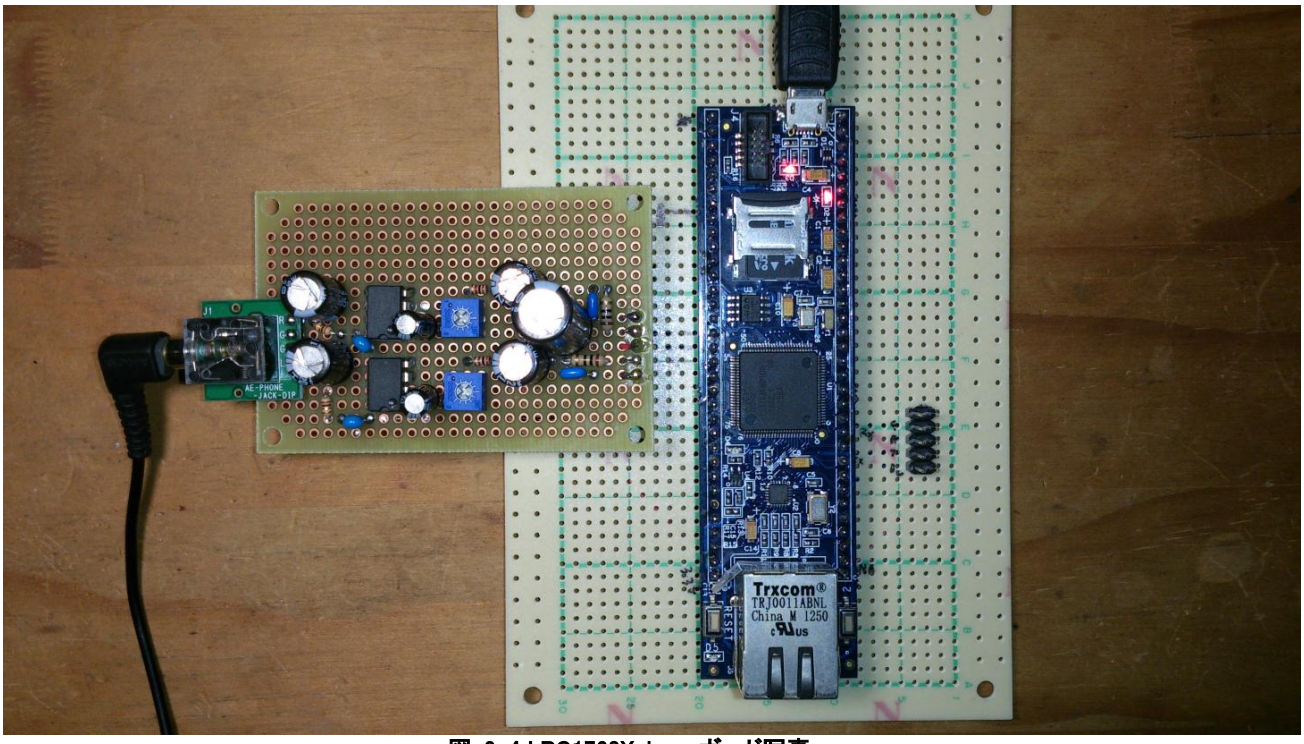

図 9-4 LPC1768Xplorer ボード写真

10 サンプル wav ファイル

10.1 wav ファイル形式

サンプルの wav ファイル再生プロジェクトで再生できるファイルの仕様は以下のようになっています。

表 10-1 wav ファイル仕様

規格	設定
サンプリング	11,025Hz
チャネル数	ステレオ(2ch)
分解能	8bit

10.2 サンプル wav ファイル

すぐに再生可能な wav ファイルを用意しています。この wav ファイルを SD カードのルートディレクトリにコピーしてお使いください。
サンプル wav ファイルは以下です。

表 10-2 サンプル wav ファイル一覧

ファイル名	デフォルトの ショートファイル名	時間[分:秒]
Aquarium.wav	AQUARIUM.WAV	3:46
Autumn.wav	AUTUMN.WAV	4:09
Night Mare.wav	NIGHTM~1.WAV	3:38
One's Fate.wav	ONE'SF~1.WAV	2:56
Quiet Cloud.wav	QUIETC~1.WAV	4:11
Quiet Memory.wav	QUIETM~1.WAV	3:34
Quiet Summer.wav	QUIETS~1.WAV	3:23
Route 6.wav	ROUTE6~1.WAV	4:03
The Roundabout Way.wav	THEROU~1.WAV	4:39
White Road.wav	WHITER~1.WAV	3:46
Winter Waltz.wav	WINTER~1.WAV	2:43

これらの音源はすべて酒井激(さかい　つよし)氏から提供いただいたオリジナルの楽曲です。著作権は酒井激氏にあります。
以下 URL からダウンロードできます。

http://homepage3.nifty.com/resove/fileSystem