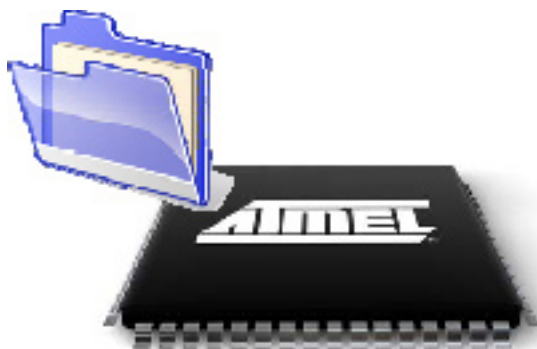


AVR114 : AT32UC3x,AT90USBx,ATmega32U4用 ATMELファイル システム管理の使い方



8ビット **AVR**[®]
マイクロコントローラ

応用記述

1. 序説

ATMELはAT32UC3x,AT90USBx,ATmega32U4デバイス用のファイルシステム管理を提供します。ファイルシステム部署はAVRコア用に最適化され、故に大きさは小さく(コード<15Kバイト、RAM<800バイト)で速度が速くなります。

ファイルシステム管理は同時に複数ファイルを開く、または再生一覧管理のような多数の機能を提供します。この応用記述はファイルシステム管理を使用する読者を手助けします。

2. 特徴と制限

2.1. 主な特徴

- FAT12,FAT16,FAT32搭載
- ASCIIとUNICODE
- 同時に多数のファイルを開く(同時アクセス)
- FAT12/FAT16/FAT32フォーマット
- 文字列パスによる誘導支援
- 直接的な作成、削除、改名
- ファイル入出力アクセス(DMA適合)
- ファイル作成、削除、改名、複写

2.2. フラグイン機能

- テキストファイル読み取り部(ASCII,UTF16,UFT16BE,UTF8)
- 再生一覧ファイル読み取り部(*.m3u,*.m3u8,*.pls,*.smp)
- 拡張濾過での誘導
- 平坦形態での誘導
- 繰り返しと乱での自動誘導機能
- POSIXインターフェース

2.3. 制限

FAT12/FAT16/FAT32仕様からの制限:

- 2Tバイトまでのディスク容量を支援
- 4Gバイトまでのファイル容量を支援
- FAT12/FAT16のルートディレクトリで512までのファイル入口(注)を支援
- ディレクトリで65535までのファイル入口(注)を支援
- フォルダの深さ制限なし

注: ファイルはそれの名前の長さに依存して1つよりも多くのファイル入口を使用し得ます。

2.4. 実装制限

- 始動でのFAT完全性検査なし
- ファイル一覧順はファイル作成順(ファイル名による並び替え支援なし)
- FAT領域数は2だけを支援(2以外の値は使用されるべきでないことがMicrosoftによって未だ強く推奨されています。)

本書は一般の方々の便宜のため有志により作成されたもので、ATMEL社とは無関係であることを御承知ください。しおりのはじめにでの内容にご注意ください。

Rev. 7824A-09/08, 7824AJ1-12/13

3. 概要

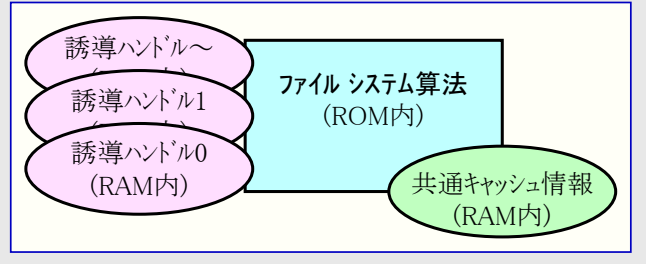
3.1. 序説

ATMELファイルシステムは主な概念の誘導ハンドルを使用します。

誘導ハンドルは1つのディレクトリの調査や1つのファイルを開くことを許します。システムは同時に複数の調査の認定とファイルを複数開くための多数のハンドルを支援します。

誘導ハンドルは非常に小さく、40バイトのRAMだけが必要です。

図3-1. 部署構成



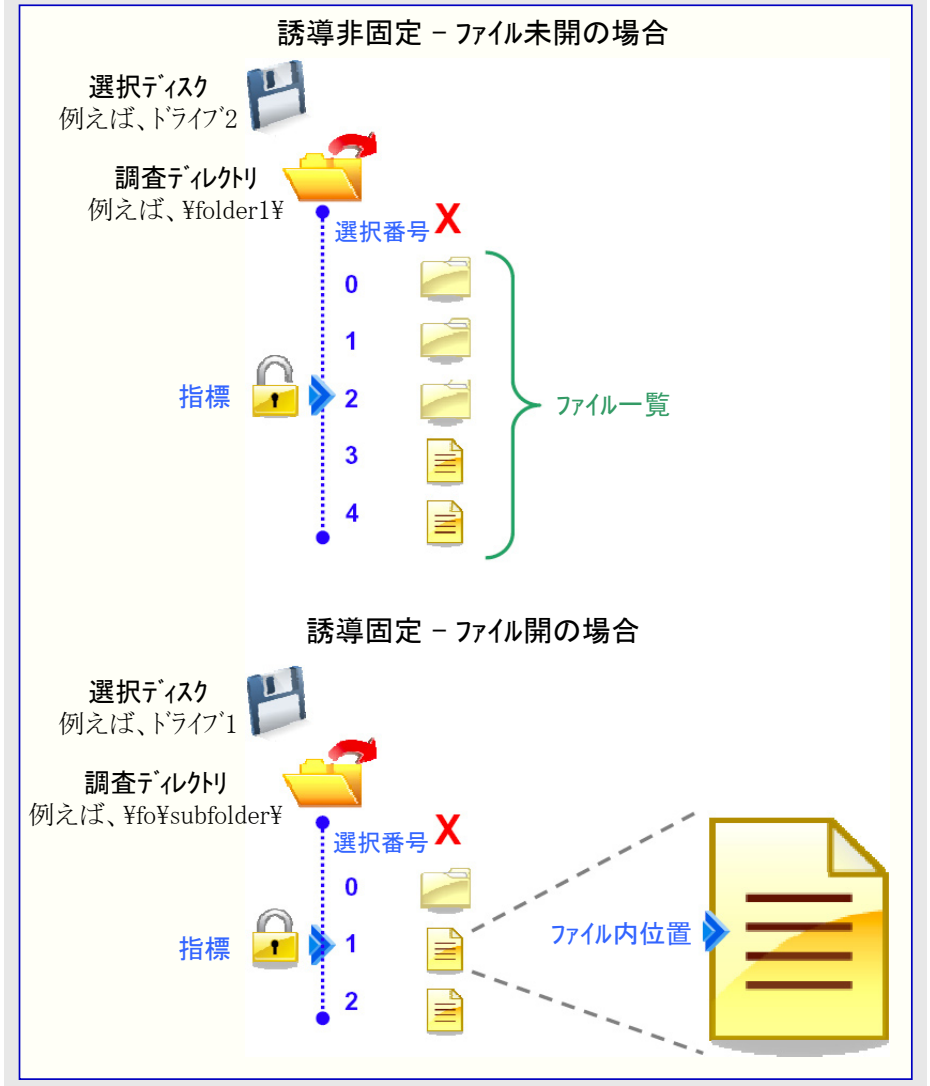
3.2. 誘導

文字列パスによるファイルアクセスは可能ですが、誘導ハンドルがファイルの一覧のアクセスを許します。誘導は一覧内で指標を移動するのに次(next)/前(previous)のような簡単な命令を用いて、**ファイル一覧**と呼ばれるディレクトリ内容を調査します。ファイル一覧はディレクトリとファイルを含むことができます。指標によって選択されたファイルが開かれている時に誘導は固定化されます。図3-2をご覧ください。

誘導ハンドル(または誘導ID)は以下の情報を含みます。

- ディスク番号
- 調査されるディレクトリの参照基準
- ファイル一覧内の指標位置
- 固定/非固定誘導フラグ (ファイル開/未開)
- ファイルが開かれている時のファイル内の位置

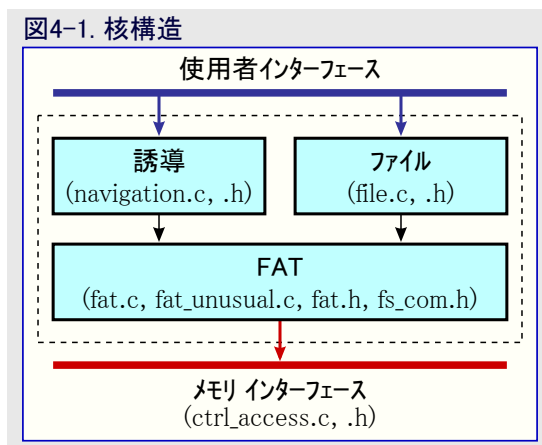
図3-2. 誘導構成



ディスクが据え付けられる時に調査される既定ディレクトリはルートディレクトリです。その後に副フォルダに指標を移動して調査されつつある新しいディレクトリとしてそれを選択("cd フォルダ"命令)することができます。また、調査されつつある新しいディレクトリとして親ディレクトリを選択("cd.."命令)することもできます。

4. 基本設計

4.1. 核



4.1.1. FAT

FAT単位部は低位単位部で、FAT構造を復号します。単位部の全てのルーチンは私的(プライベート)で使用者によって呼び出すことはできません。

4.1.2. 誘導

この単位部は次のようなルーチンを提供します。

- ・ 誘導ハンドルを選択
- ・ ファイル一覧での誘導
- ・ 選択ディレクトリを変更
- ・ ディレクトリ ツリーを変更
- ・ ディスク/ディレクトリ/ファイルについての情報取得

全てのルーチンはnavigation.hファイルで記述されます。

4.1.3. ファイル

この単位部は次のようなファイル入出力制御ルーチンを提供します。

- ・ 1バイト転送 : file_getc() - file_putc() 遅い
- ・ SRAM緩衝転送 : file_read_buf() - file_write_buf() ↓
- ・ 直接転送 (注) : file_read() - file_write() 速い

注: 直接転送はRAM空間内でのデータ転送なしで2つのメモリ領域間で直接的にデータを転送(例えば2つのメモリ空間間でのDMA)するために開発されています。

全てのルーチンはfile.hファイルで記述されます。

4.1.4. メモリ

メモリとのインターフェースは次の通りです。

- ・ mem_test_unit_ready() - メモリ状態検査
- ・ mem_wr_protect() - メモリ書き込み保護検査
- ・ memory_2_ram() - メモリ読み込み
- ・ ram_2_memory() - メモリ書き込み

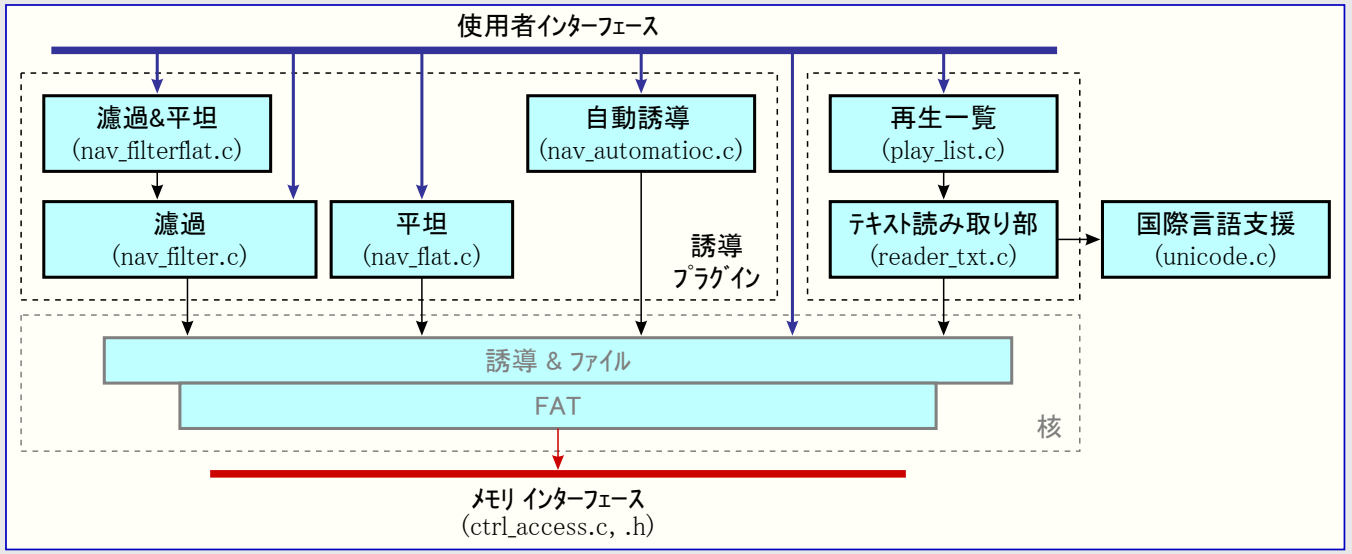
全てのルーチンはctrl_access.hファイルで記述されます。

4.1.5. 異常管理

多数のルーチンは状態TRUEまたは、失敗状態の場合にFALSEを返し、異常についてより多くの情報を取得するためにfs_g_status全体変数が異常識別子を含みます。異常一覧はfs_com.hファイルで利用可能です。

4.2. プラグイン

図4-2. プラグイン構造



4.2.1. テキスト読み取り部

このプラグインは読み取り専用形態で基本的なテキストファイルのオープンを許します。支援されるテキスト形式はASCII、UTF16、UTF16BE、UTF8です。複数オープンテキストファイルが支援されます。

4.2.2. 再生一覧

このプラグインは読み取り専用形態でファイル再生一覧を管理します。支援される拡張子は*.m3u、*.m3U8、*.pls、*.smpです。再生一覧の大きさ制限は65535ファイルです。複数オープンは支援されません。

4.2.3. 自動誘導

自動誘導は再生部/表示部のために開発されました。プラグインは以下の使用者仕様で“ファイル一覧”を構築します。

- 拡張子濾過
- 制限走査 (フォルダ、フォルダと副フォルダ、1つのディスク、全ディスク)
- 乱機能

4.2.4. 濾過動作形態

このプラグインは誘導部署からの拡張子によって“ファイル一覧”を濾過します。

例:

```
// ディスク構造
folder1
|  folder3
|  |  file4. mp3
|  file5. txt
folder2
|  file6. txt
file1. mp3
file2. txt
file3. mp3

// 誘導部によって提供された“ファイル一覧”
// “ルート”は“選択されたディレクトリ”
folder1
folder2
file1. mp3
file2. txt
file3. mp3

// 誘導部によって提供された“ファイル一覧”
// “folder1”は“選択されたディレクトリ”
folder3
file5. txt

// “*.mp3”濾過で初期化された“濾過誘導”プラグインで提供された“ファイル一覧”
// “ルート”は“選択されたディレクトリ”
folder1
folder2
file1. mp3
file3. mp3

// “*.mp3”濾過で初期化された“濾過誘導”プラグインで提供された“ファイル一覧”
// “folder1”は“選択されたディレクトリ”
folder3
```

4.2.5. 平坦動作形態

平坦動作形態はフォルダレベルを無視して選択したフォルダの副フォルダ内に存在するファイル/フォルダで“ファイル一覧”を提供します。

例:

```
// ディスク構造
folder1
|
| folder3
| | file4
| | file5
|
| folder2 | file6
file1
file2
file3

// 誘導部によって提供された“ファイル一覧”
// “ルート”は“選択されたディレクトリ”
folder1
folder2
file1
file2
file3

// 誘導部によって提供された“ファイル一覧”
// “folder1”は“選択されたディレクトリ”
folder3
file5

// “平坦誘導部”プラグインによって提供された“ファイル一覧”
// “ルート”は“選択されたディレクトリ”
folder1
folder3
file4
file5
folder2
file6
file1
file2
file3

// “平坦誘導部”プラグインによって提供された“ファイル一覧”
// “folder1”は“選択されたディレクトリ”
folder3
file4
file5
```

4.2.6. 濾過&平坦動作形態

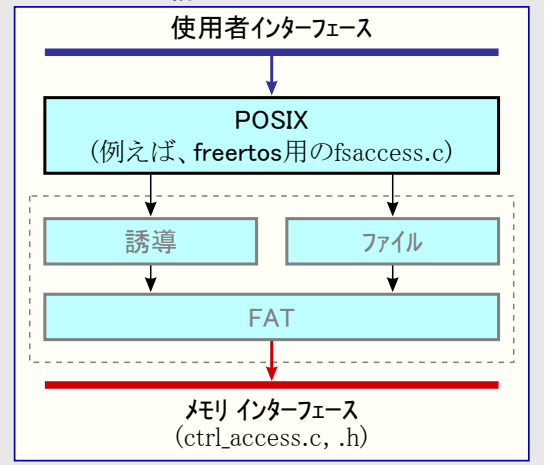
このプラグインは“濾過動作形態”と“平坦動作形態”の特徴を含みます。

4.3. POSIXインターフェース

POSIX : Unixオペレーティングシステムの変形に適合する可搬型オペレーティングシステム
インターフェース(Portable Operating System Interface)

このインターフェースはAVR32UC3製品でだけ利用可能です。

図4-3. POSIX構造



5. 形態設定

5.1. 核

応用に依存して、以下の形態設定で核の大きさを最適化することができます。

- 支援されるFATの許可/禁止 (FAT12/FAT16/FAT32)
- UNICODEまたはASCII支援の許可/禁止
- 機能レベルの選択 (READ/WRITE/WRITE_COMPLET(注))
- キャッシュ情報量の選択
- 最大誘導ハンドル数の選択

注: WRITE_COMPLETはWRITE機能と少しの追加機能を含みます。

形態設定は`conf_explorer.h`ファイルで定義されます。

```
// メモリ ルーチン(例えばmemset(), memcpy_ram2ram()など)を提供するインクルード ヘッダ ファイル
#define LIB_MEM <string.h>
// ディスク アクセス ルーチン(例えばram_2_memory(), mem_wr_protect()など)を提供するインクルード ヘッダ ファイル
#define LIB_CTRLACCESS "ctrl_access.h"

// 支援するFAT (ENABLEDまたはDISABLED)
#define FS_FAT_12 ENABLED
#define FS_FAT_16 ENABLED
#define FS_FAT_32 ENABLED

// 調査部はASCIIまたはUNICODE文字列形式のどちらか、または両方を支援するかもしれません。
#define FS_ASCII DISABLED
#define FS_UNICODE ENABLED

// 誘導部は最初のパーティションだけ(DISABLED)、または複数パーティション(ENABLED)を支援するかもしれません。
#define FS_MULTI_PARTITION DISABLED

// ファイルシステム核での機能水準
// 以下の内から選択してください。:
// - FSFEATURE_READ: 全ての読み込み機能
// - FSFEATURE_WRITE: nav_file_copy(), nav_file_paste(), nav_file_del(), file_create(), file_open(MODE_WRITE),
// file_write(), file_putc()
// - FSFEATURE_WRITE_COMPLET: FSFEATURE_WRITE関数とnav_drive_format(), nav_dir_make(), nav_file_rename(),
// nav_file_dateset(), nav_file_attributset()
// - FSFEATURE_ALL: 全ての関数
#define FS_LEVEL_FEATURES (FSFEATURE_READ | FSFEATURE_WRITE_COMPLET)

// ファイルのクラスタ一覧を格納するのに使用されるキャッシュ数 (0以上でなければなりません。)
// 多数のオープン ファイルの場合に重要
#define FS_NB_CACHE_CLUSLIST 3

// 誘導部ID数は一度に開くファイルの最大数と一度での調査最大数を定義します(0<n<256)。
// 各誘導部は50バイト未満のRAMを使用します。
#define FS_NB_NAVIGATOR 3

// 誘導部は複写ファイルを開くのに以下の誘導部IDを、貼り付けファイルを作成するのに現在の誘導部IDを使用します。
#define FS_NAV_ID_COPYFILE 2 // 貼り付けルーチンが呼ばれる時に現在使用されているIDと違わなければなりません。
```

5.2. プラグイン

プラグイン形態設定は`conf_explorer.h`ファイルで定義されます。

5.2.1. 再生一覧

```
// 再生一覧ファイルを開くのに使用される誘導部
#define FS_NAV_ID_PLAYLIST 2
// 再生一覧を含む現在のパスを格納する空間を割り当てる再生一覧インターフェース
// 割り当てライブラリでの例
#define PLAYLIST_BUF_ALLOC( size ) malloc( size )
#define PLAYLIST_BUF_FREE( buf ) free( buf )
// 割り当てライブラリなしでの例
#define PLAYLIST_BUF_ALLOC( size ) ((sizeof(g_buffer_512)>512)? NULL : g_buffer_512)
#define PLAYLIST_BUF_FREE( buf )
```

5.2.2. 自動誘導

```
// "利用可能な全ファイル計数"機能は始動時の時間節約のために禁止することができます。
#define FS_NAV_AUTOMATIC_NBFILE ENABLE
// "自動誘導"プラグイン(nav_automat.c)によって提供される一覧でのファイル順を指定
#define NAV_AUTO_FILE_IN_FIRST // 機能を禁止するには注釈を外してください。
// 乱操作の分割量(8ファイル単位)
#define NAVAUTO_MAX_RANGE_RAND 8 // 8×8=64ファイル
// 乱数法(バイト値)
#include "rand.h"
#define NAVAUTO_GET_RAND( value ) (value=rand())
```

5.2.3. 平坦誘導

```
// 部署に平坦動作形態を許します。(nav_flat.c & navfilterflat.c)
#define NAVIGATION_MODE_FLAT // 機能を禁止するには注釈を外してください。
```

6. 例

注: 全ての例はATMELファイルシステムが「電源ON/OFF手順」例のように初期化されるとの仮定です。

6.1. 電源ON/OFF手順

ファイルシステム部署を初期化する手順は次の通りです。

```
nav_reset();
```

ファイルシステム部署を停止する前に実行する手順は次の通りです。

```
// 開かれたファイルがある場合はそれらを閉じてください。
// 結局、FATキャッシュ内に存在するデータを破棄します。
nav_exit();
```

規準: 同時に同じディスクで2つのファイルシステム管理を持つことは認められません。

ATMELファイルシステム(FS)と別のファイルシステム(FS)間でディスクを共用するシステムを持つ場合、別のFSへ飛ぶ前にATMELのFSを停止して、他のFSから抜け出した後でATMELのFSを再初期化しなければなりません。

例: ATMELのマイクロコントローラが大容量記憶装置動作形態なら、USBホストFS(例えば、Windows®のそれ)が大容量記憶装置を処理するために、ATMELのFSはOFFにされなければなりません。

6.2. ディスク状態調査

ディスクを調査するには2つのルーチンが必要です。

- nav_drive_set()はメモリドライブの存在の検出を許します。
- nav_partition_mount()はメモリの存在を調べてパーティションの搭載を試みます。

注: メモリドライブの数は動的かもしれませんが(例えば、U-DISKは複数のメモリドライブを持つことができます)。

```

Bool check_disk( U8 lun )
{
    nav_select(0); // 誘導部ID選択
    if( !nav_drive_set(lun) )
    {
        printf("Driver memory no available\n");
        return FALSE;
    }
    // ここではメモリが選択されます。

    if( !nav_partition_mount() )
    {
        switch( fs_g_status )
        {
            case FS_ERR_HW_NO_PRESENT:
                printf("Disk not present\n");
                break;
            case FS_ERR_HW:
                printf("Disk access error\n");
                break;
            case FS_ERR_NO_FORMAT:
                printf("Disk no formatted\n");
                break;
            case FS_ERR_NO_PART:
                printf("No partition available on disk\n");
                break;
            case FS_ERR_NO_SUPPORT_PART:
                printf("Partition no supported\n");
                break;
            default :
                printf("Other system error\n");
                break;
        }
        return FALSE;
    }
    // ここではメモリ上にパーティションが搭載され、誘導部はルートディレクトリ上です。

    return TRUE;
}

```

6.3. ファイルの日付変更

この例は作成日付と最終アクセス日を変更します。

```

Bool changedate( void )
{
// ファイル日付変更の例
const U8 _MEM_TYPE_SLOW_ date_create[]="2005122512301050"; // 日付= 12/25/2005 12h30mn10.5s
const U8 _MEM_TYPE_SLOW_ date_write[]="2006072319005130"; // 日付= 07/23/2006 19h51mn30s

nav_select(0); // 本手順用に誘導部ID 0を選択

if( !nav_drive_set( LUN_ID_NF_DISKMASS ) )
return FALSE;
if( !nav_partition_mount() )
return FALSE;
// 最初のファイルまたはディレクトリを選択
if( !nav_filelist_set( 0 , FS_FIND_NEXT ) )
return FALSE;
// 作成日付変更
if( !nav_file_dateset( date_create , FS_DATE_CREATION ) )
return FALSE;
// 最終アクセス日変更
if( !nav_file_dateset( date_write , FS_DATE_LAST_WRITE ) )
return FALSE;

return TRUE;
}

```

6.4. パス文字列の使用

ファイルやパスをアクセスするのにテキスト パス(ASCIIまたはUNICODE)を使用することができます。

nav_setcwd()は以下の文字列を受け入れます。

- 現在のディレクトリでファイルまたはディレクトリを検索するための" name.txt"
- 現在のディレクトリで一致する名前を持つファイルまたはディレクトリを検索するための" nam*"
- "¥"と"/"は等価で支援されます。
- 現在のディレクトリでディレクトリを検索してそのディレクトリへ移行するための" name¥"
- 現在のディレクトリでファイルを検索するための" name2.txt"
- ルートディレクトリでファイルを検索するための" ¥name2.txt"
- ".¥"は支援されますが必須ではありません。
- ".¥.¥"は支援されます。
- "A:¥"は支援され、'A'はディスク0に対応します。

```

Bool search_path( void )
{
const _MEM_TYPE_SLOW_ U8 path[]="dir1/file.txt";

nav_select(0); // 空き誘導部選択

// ディスクを選んでそれを搭載
nav_drive_set(1);
nav_partition_mount();
// ここで、誘導部はルートディレクトリです。

#ifdef (FS_ASCII == ENABLED) && (FS_UNICODE == ENABLED)
nav_string_ascii(); // ASCII名形式選択
#endif

// ディスク1の現在のディレクトリで"dir1/file.txt"ファイルを検索
if( !nav_setcwd( (FS_STRING)path , TRUE, FALSE ) )
return FALSE;

return TRUE;
}

```

6.5. 複数書き込みファイルの加速

6.5.1. 概要

多数のfile_write_buf()またはfile_putc()の呼び出し(例えば、記録ファイルの場合、ディスクでの書き込みアクセス数がFAT表の構築に重要なため、実行が遅くなるかもしれません。ファイルで多くの時間にデータを書いてその書き込みアクセスを分割する場合(例えば、記録ファイル)は、もっと効率的なコード手順を用いてファイルのFAT表を構築することが重要かもしれません。

各々実行されるfile_write_buf()またはfile_putc()呼び出し(例えば、記録ファイル)ルーチンはFAT表での予約が必要です。「例A」をご覧ください。

効率を増すため、他の命令が実行される前に、単一の事前手順内でFAT内の必要な全空間の予約作成を提案します。「例B」をご覧ください。この例はもっと効率的な書き込みルーチンに帰着します。6.5.4項をご覧ください。

6.5.2. 例A

この例はファイルを満たす通常の手順です。

```
// ファイルを満たす。>1Mバイト
#define FILL_FILE_NB_WRITE 855L
#define FILL_FILE_BUF_SIZE 120L

Bool fill_file( void )
{
    const UNICODE _MEM_TYPE_SLOW_ name[50]={'l','o','g','.','b','i','n',0};
    U16 u16_nb_write;

    memset( g_trans_buffer , 0x55 , FILL_FILE_BUF_SIZE );

    if( !nav_drive_set(LUN_DISK) ) // ディスクに移行
        return FALSE;
    if( !nav_partition_mount() ) // ディスクのパーティション搭載
        return FALSE;

    if( !nav_file_create( (const FS_STRING) name ) ) // ファイル作成
        return FALSE;
    if( !file_open(FOPEN_MODE_W) ) // ファイル容量0強制的書き込み形態でファイルを開く
        return FALSE;

    for( u16_nb_write=0; u16_nb_write<FILL_FILE_NB_WRITE; u16_nb_write++ )
    {
        // ここは、各ファイル書き込みでFAT領域の割り当てが走行し、
        // 故に書き込みに多数の緩衝部を持つなら、事項が遅くなるかもしれません。
        if( !file_write_buf( g_trans_buffer , FILL_FILE_BUF_SIZE ) )
        {
            file_close();
            return FALSE;
        }
    }
    file_close();
    return TRUE;
}
```

6.5.3. 例B

この例は予め割り当てられた加速書き込みアクセス ルーチンを許します。

```
// File fill >1MB
#define FILL_FILE_NB_WRITE 855L
#define FILL_FILE_BUF_SIZE 120L

Bool fill_file_fast( void )
{
    const UNICODE _MEM_TYPE_SLOW_ name[50]={'l','o','g','_','f','a','s','t','.','b','i','n',0};
    _MEM_TYPE_SLOW_ Fs_file_segment g_recorder_seg;
    UI6 u16_nb_write;

    memset( g_trans_buffer , 0x55 , FILL_FILE_BUF_SIZE );

    if( !nav_drive_set(LUN_DISK) // ディスクに移行
        return FALSE;
    if( !nav_partition_mount() // ディスクのパーティション搭載
        return FALSE;

    if( !nav_file_create( (const FS_STRING) name ) // ファイル作成
        return FALSE;
    if( !file_open(FOPEN_MODE_W)) // ファイル容量0強制的書き込み形態でファイルを開く
        return FALSE;

    // 予め割り当てるセグメントの大きさを定義(512バイト単位)
    // 注: 総量を知らない場合、より多く割り当てることができます。
    g_recorder_seg.u16_size = (FILL_FILE_NB_WRITE*FILL_FILE_BUF_SIZE + 512L)/512L;

    // ****事前割当***** 満たすセグメント
    if( !file_write( &g_recorder_seg )
        {
            file_close();
            return FALSE;
        }

    // 割り当てたセグメントの大きさ検査
    if( g_recorder_seg.u16_size < ((FILL_FILE_NB_WRITE*FILL_FILE_BUF_SIZE + 512L)/512L) )
        {
            file_close();
            return FALSE;
        }

    // 大きさをリセットするためにファイルを閉じて開く
    file_close(); // ファイルを閉じる。このルーチンは直前の割り当てを取り去りません。
    if( !file_open(FOPEN_MODE_W)) // ファイル容量0強制的書き込み形態でファイルを開く
        return FALSE;

    for( u16_nb_write=0; u16_nb_write<FILL_FILE_NB_WRITE; u16_nb_write++ )
        {
            // ここで、ファイルのクラスタ一覧が既に割り当てられ、書き込みルーチンがより速くなります。
            if( !file_write_buf( g_trans_buffer , FILL_FILE_BUF_SIZE )
                {
                    file_close();
                    return FALSE;
                }
        }
    file_close();
    return TRUE;
}
```

6.5.4. 統計

以下の例の結果は次の通りです。

256Mバイトのディスク(FAT16、クラスタ=4Kバイト)で100.2Kバイト(緩衝部120バイト×855回の書き込み)のファイル作成

- ・例Aで、FATでの同じセクタの書き込みアクセス数は最大で50回、平均で25回です。
- ・例Bで、FATでの同じセクタの書き込みアクセス数は最大で2回、平均で1回です。

256Mバイトのディスク(FAT16、クラスタ=4Kバイト)で1.1Mバイト(緩衝部120バイト×10000回の書き込み)のファイル作成

- ・例Aで、FATでの同じセクタの書き込みアクセス数は最大で256回、平均で147回です。
- ・例Bで、FATでの同じセクタの書き込みアクセス数は最大で2回、平均で1回です。

注: FAT時間作成が使用されるFAT形式に依存するのを知ることが重要です。FATでの割り当てはFAT32よりもFAT(FAT12,FAT16)でより速くなります。2Gバイト未満のディスク容量なら、`nav_drive_format(FS_FORMAT_FAT)`を呼ぶ時にFATの形式を強制することができます。FAT32は同じファイル容量と同じディスク容量に対してFAT表でFAT16よりも多く(×4)の書き込みを必要とします。

6.6. 別のディスクへのディスク複写

この例は3つの誘導部ハンドルを使用します。

- ・転送元ディスク調査用
- ・転送先ディスク調査用
- ・複写/貼り付け機能用に`conf_explorer.h`でFS_NAV_ID_COPYFILEによって定義された誘導部

```

Bool copydisk( void )
{
    const UNICODE _MEM_TYPE_SLOW_ name[50];
    U8 u8_folder_level = 0;

//trace("Mount drive¥n");    /** 3つの誘導部使用 (0:SD調査、1:NFディスク調査、2:ファイル複写ルーチンによって使用)
    nav_select( 0 );
    if( !nav_drive_set( LUN_ID_MMC_SD ) )
        return FALSE;
    if( !nav_partition_mount() )
        return FALSE;
    nav_select( 1 );
    if( !nav_drive_set( LUN_ID_NF_DISKMASS ) )
        return FALSE;
    if( !nav_partition_mount() )
        return FALSE;

    // 全てのフォルダとファイルの走査と作成の繰り返し
    while(1)
    {
        // 現在のディレクトリにディレクトリが無ければ、SDとNANDフラッシュ ディスクの親ディレクトリへ
        while(1)
        {
//trace("Search files or dir¥n");
            // Reselect SD
            nav_select( 0 );
            if( nav_filelist_set( 0 , FS_FIND_NEXT ) )
                break;    // 次のファイルとディレクトリ発見

            // 現在のディレクトリに他のディレクトリやファイルが無ければ、SDとNANDフラッシュ ディスクの親ディレクトリへ
            if( 0 == u8_folder_level )
            {
                // フォルダ更新終了
//trace("End of copy¥n");
                return TRUE;    /******* 複写終了 *****/
            }

//trace("Go to parent¥n");
            // 注意、nav_dir_gotoparent()ルーチンは親ディレクトリへ行き、一覧で子ディレクトリを選択
            u8_folder_level--;
            if( !nav_dir_gotoparent() )
                return FALSE;
            // NANDフラッシュ誘導部を選択してSDの同じディレクトリへ行く
            nav_select( 1 );
            if( !nav_dir_gotoparent() )
                return FALSE;
        } // while(1)終了
    }
}

```

```

    if( nav_file_isdir()
    {
//trace("Dir found - create dir & CD¥n");
    /** ここで、新しいディレクトリが発見されて選択されます。
    // 現在の選択の名前取得 (=SDのディレクトリ名)
    if( !nav_file_name( (FS_STRING)name , 50 , FS_NAME_GET, FALSE ) )
        return FALSE;
    // (SDの)ディレクトリへ移行
    if( !nav_dir_cd()
        return FALSE;
    u8_folder_level++;
    // NANDフラッシュ ディスク選択
    nav_select( 1 );
    // NANDフラッシュ ディスクでフォルダ作成
    if( !nav_dir_make( (FS_STRING)name ) )
    {
        if( FS_ERR_FILE_EXIST != fs_g_status )
            return FALSE;
        // ここで、名前既存誤り
    }
    // ここで誘導部はNANDフラッシュでフォルダを選択
    if( !nav_dir_cd()
    {
        if( FS_ERR_NO_DIR == fs_g_status )
        {
            // ファイルが同一フォルダ名を持つため、FYC⇒複写不能
        }
        return FALSE;
    }
    // ここで、フォルダが作成され、誘導部がこのディレクトリに移行されます。
    }
else
    {
//trace("File found - copy file¥n");
    /** ここで、新しいディレクトリが発見されて選択されます。
    // 現在の選択の名前取得 (=SDのファイル名)
    if( !nav_file_name( (FS_STRING)name , 50 , FS_NAME_GET , FALSE ) )
        return FALSE;
    if( !nav_file_copy()
        return FALSE;

    // NANDフラッシュの現在のディレクトリでファイル貼り付け
    nav_select( 1 );
    while( !nav_file_paste_start( (FS_STRING)name ) )
    {
        // Error
        if( fs_g_status != FS_ERR_FILE_EXIST )
            return FALSE;
//trace("del file¥n");
        // ファイル既存、これを削除
        if( !nav_file_del( TRUE ) )
            return FALSE;
        // ここで、貼り付け再試行
    }
    // 複写動作
    {
    U8 status;
    do{
        status = nav_file_paste_state(FALSE);
    }while( COPY_BUSY == status );

    if( COPY_FINISH != status )
        return FALSE;
    }
    } // if ディレクトリまたはファイル
    } // 最初のwhile(1)の終了
    }

```



本社

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131
USA
TEL 1(408) 441-0311
FAX 1(408) 487-2600

国外営業拠点

Atmel Asia

Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
TEL (852) 2245-6100
FAX (852) 2722-1369

Atmel Europe

Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
TEL (33) 1-30-60-70-00
FAX (33) 1-30-60-71-11

Atmel Japan

104-0033 東京都中央区
新川1-24-8
東熱新川ビル 9F
アトメル ジャパン株式会社
TEL (81) 03-3523-3551
FAX (81) 03-3523-7581

製品窓口

ウェブサイト

www.atmel.com

技術支援

avr@atmel.com

販売窓口

www.atmel.com/contacts

文献請求

www.atmel.com/literature

お断り: 本資料内の情報はATMEL製品と関連して提供されています。本資料またはATMEL製品の販売と関連して承諾される何れの知的所有権も禁反言あるいはその逆によって明示的または暗示的に承諾されるものではありません。ATMELのウェブサイトに位置する販売の条件とATMELの定義での詳しい説明を除いて、商品性、特定目的に関する適合性、または適法性の暗黙保証に制限せず、ATMELはそれらを含むその製品に関連する暗示的、明示的または法令による如何なる保証も否認し、何ら責任がないと認識します。たとえATMELがそのような損害賠償の可能性を進言されたとしても、本資料を使用できない、または使用以外で発生する(情報の損失、事業中断、または利益の損失に関する制限なしの損害賠償を含み)直接、間接、必然、偶然、特別、または付随して起こる如何なる損害賠償に対しても決してATMELに責任がないでしょう。ATMELは本資料の内容の正確さまたは完全性に関して断言または保証を行わず、予告なしでいつでも製品内容と仕様の変更を行う権利を保留します。ATMELはここに含まれた情報を更新することに対してどんな公約も行いません。特に別の方法で提供されなければ、ATMEL製品は車載応用に対して適当ではなく、使用されるべきではありません。ATMEL製品は延命または生命維持を意図した応用での部品としての使用に対して意図、認定、または保証されません。

© Atmel Corporation 2008. 全権利予約済 ATMEL®、ロゴとそれらの組み合わせ、AVR®とその他はATMEL Corporationの登録商標または商標またはその付属物です。他の用語と製品名は一般的に他の商標です。

© HERO 2013.

本応用記述はATMELのAVR114応用記述(doc7824.pdf Rev.7824A-09/08)の翻訳日本語版です。日本語では不自然となる重複する形容表現は省略されている場合があります。日本語では難解となる表現は大幅に意識されている部分もあります。必要に応じて一部加筆されています。頁割の変更により、原本より頁数が少なくなっています。

必要と思われる部分には()内に英語表記や略称などを残す形で表記しています。

青字の部分はリンクとなっています。一般的に赤字の0,1は論理0,1を表します。その他の赤字は重要な部分を表します。