

μ T-Kernel3.0

時計ドライバ実装仕様書、
RAMディスクドライバ実装仕様書、
SDカードドライバ実装仕様書、
USBメモリドライバ実装仕様書、
FatFs実装仕様書

Version. 01. 00. 03

2023. 9. 7

目次

1. 概要	5
1.1 目的	5
1.2 実装の基本方針	5
1.3 バージョン情報	5
1.4 関連ドキュメント	5
1.5 ソースコード構成	6
1.6 プロジェクト・ファイル	6
2. 時計(クロック)ドライバ	7
2.1 時計(クロック)ドライバの概要	7
2.2 時計(クロック)ドライバの仕様	7
2.2.1 対象デバイス	7
2.2.2 デバイス名	7
2.2.3 固有機能	7
2.2.4 属性データ	7
2.2.5 固有データ	8
2.2.6 事象通知	8
2.2.7 エラーコード	8
2.3 ヘッダファイルとサービス関数	8
2.4 サンプルプログラム	8
2.5 スタックサイズ	9
3. RAM ディスクドライバ	10
3.1 RAM ディスクドライバの概要	10
3.2 RAM ディスクドライバの仕様	10
3.2.1 対象デバイス	10
3.2.2 デバイス名	10
3.2.3 固有機能	10
3.2.4 属性データ	10
3.2.5 固有データ	11
3.2.6 事象通知	12
3.2.7 エラーコード	12
3.3 ヘッダファイルとサービス関数	12
3.4 サンプルプログラム	12
3.5 スタックサイズ	12
4. SD カードドライバ	13
4.1 SD カードドライバの概要	13
4.2 SD カードドライバの仕様	13

4.2.1	対象デバイス	13
4.2.2	デバイス名	13
4.2.3	固有機能	13
4.2.4	属性データ	13
4.2.5	固有データ	14
4.2.6	事象通知	14
4.2.7	エラーコード	14
4.3	SD カードドライバが使用する資源	14
4.3.1	使用する資源の概要	14
4.3.2	イベントフラグ	15
4.3.3	タスク (shc_tsk)	15
4.3.4	割込みハンドラ (SD_Detect_hdr、SD_Int_hdr)	15
4.3.5	グループ割込み (GroupBL1Handler)	16
4.4	SD カードドライバのコンフィグレーション	16
4.4.1	SD カードドライバのコンフィグレーション・ファイル	16
4.4.2	タスクの優先度と割込みハンドラの割込み優先度	17
4.4.3	DMAC のチャンネル番号	17
4.5	ヘッダファイルとサービス関数	17
4.6	サンプルプログラム	18
4.7	スタックサイズ	18
4.7.1	スタック見積もりツール	18
4.7.2	sdc_tsk のスタックサイズ	18
4.7.3	GroupBL1Handler のスタックサイズ	19
4.7.4	SD カードドライバの処理関数	19
5.	USB メモリドライバ	20
5.1	USB メモリドライバの概要	20
5.2	USB メモリドライバの仕様	20
5.2.1	対象デバイス	20
5.2.2	デバイス名	20
5.2.3	固有機能	20
5.2.4	属性データ	20
5.2.5	固有データ	21
5.2.6	事象通知	21
5.2.7	エラーコード	21
5.3	USB メモリドライバが使用する資源	21
5.3.1	使用する資源の概要	21
5.3.2	イベントフラグ	22
5.3.3	タスク (usb_tsk)	22

5.3.4	割込みハンドラ (USB_Int_hdr、DMA_End_hdr)	22
5.3.5	選択型割込みB (PERIB)	23
5.3.6	DMAC の割込み	23
5.4	USB メモリドライバのコンフィグレーション	24
5.4.1	USB メモリドライバのコンフィグレーション・ファイル	24
5.4.2	タスクの優先度と割込みハンドラの割込み優先度	24
5.4.3	選択型割込みBの割込み番号と DMAC のチャンネル番号	25
5.4.4	VSUB 信号のアクティブレベル	25
5.5	ヘッダファイルとサービス関数	26
5.6	サンプルプログラム	26
5.7	スタックサイズ	26
5.7.1	スタック見積もりツール	26
5.7.2	usb_tsk のスタックサイズ	27
5.7.3	割込みハンドラのスタックサイズ	27
5.7.4	USB メモリドライバの処理関数	27
6.	FatFs	28
6.1	FatFs とは	28
6.2	FatFs の μ T-Kernel3.0 への移植	28
6.2.1	システム関連機能を μ T-Kernel 対応に変更	29
6.2.2	デバイス入出力を μ T-Kernel 対応に変更	29
6.3	FatFs の構成オプション	30
6.3.1	機能構成	30
6.3.2	ロケールと名前空間の構成	31
6.3.3	ドライブ/ボリューム構成	32
6.3.4	システム構成	33
6.4	サンプルプログラム	35
7.	問い合わせ先	38

1. 概要

1.1 目的

本構築手順書はルネサスエレクトロニクス社のRX用に改変した μ T-Kernel3.0 (V3.00.00)のソースコードにFatFsのFATファイルシステムを実装するための手順を記載した仕様書です。

1.2 実装の基本方針

本実装の基本方針を以下に示します。

- FatFsは μ T-Kernel3.0仕様の μ T-Kernel/SMのデバイス管理機能に準拠した仕様とする。
- FatFsが使用するget_fattime関数のタイムスタンプには時計(クロック)ドライバを利用する。時計(クロック)ドライバは、T-Engine標準デバイスドライバの仕様に準拠した形式とする。
- システムディスクとしてはRAMディスクとSDカードを提供する。RAMディスクドライバとSDカードドライバも、T-Engine標準デバイスドライバのシステムディスクドライバの仕様に準拠した形式とする。

1.3 バージョン情報

本実装に利用したFatFs (http://elm-chan.org/fsw/ff/00index_e.html) のバージョン情報は以下の通りです。

Ver.: R0.14b

リリース日: 2021/4/17

1.4 関連ドキュメント

以下に本実装仕様書の関連するドキュメントを示します。

分類	名称	発行
OS仕様	μ T-Kernel3.0仕様書 (Ver. 3.00.00)	TRONフォーラム TEF020-S004-3.00.00
ドライバ仕様	T-Engine標準デバイスドライバ 仕様	TRONフォーラム TEF040-S216-01.00.01
共通実装仕様	uTK3.0共通実装仕様書	
ハードウェア 依存実装仕様	uTK3.0_AP-RX63N-0A実装仕様書	
	uTK3.0_AP-RX65N-0A実装仕様書	
	uTK3.0_AP-RX72N-0A実装仕様書	
構築手順	uTK3.0_AP-RX63N-0A構築手順書	
	uTK3.0_AP-RX65N-0A構築手順書	
	uTK3.0_AP-RX72N-0A構築手順書	

1.5 ソースコード構成

本実装のソースコードのディレクトリ構成を以下に示します。

└ device	デバイスドライバ・ミドルウェア
└ rtc	時計(クロック)ドライバ
└ └ sysdepend	実装依存定義
└ rd	RAMディスクドライバ
└ └ sysdepend	実装依存定義
└ sd	SDカードドライバ
└ └ sysdepend	実装依存定義
└ usb	USBメモリドライバ
└ └ sysdepend	実装依存定義
└ include	アプリケーション用のインクルードファイル
└ lib	ライブラリ
└ └ libfat	FatFsファイルシステム

機種依存定義 sysdepend ディレクトリは以下のように構成されます。

└ sysdepend	実装依存定義
└ └ <ターゲット1>	ターゲット1 依存部
└ └ :	
└ └ <ターゲットn>	ターゲットn 依存部

1.6 プロジェクト・ファイル

本実装を利用するためのプロジェクト・ファイルは「mtkernel_3/ide/cs」のフォルダにあります。

アイコンの名称が ****_FAT.mtpj となっているものがファイルシステムを利用可能な μ T-Kernel3.0 のプロジェクト・ファイルです。ダブルクリックすればCS+が起動されます。

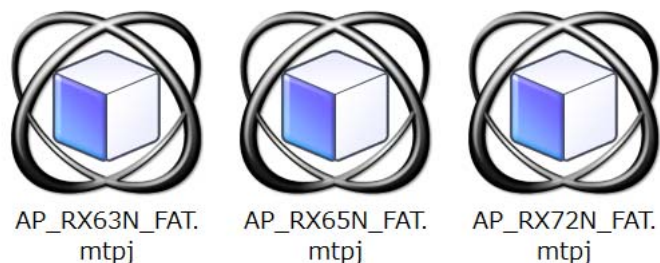


図1.1 FatFs対応のプロジェクト・ファイル

2. 時計(クロック)ドライバ

2.1 時計(クロック)ドライバの概要

時計(クロック)ドライバではハードウェア・タイマとしてRX内蔵のRTC(リアルタイムクロック)を利用します。インタフェース仕様はT-Engine標準デバイスドライバ仕様に準拠しています。また、時計(クロック)ドライバを μ T-Kernelに登録するためのrtcDrvEntry関数をサービス関数として提供します。ユーザシステムではrtcDrvEntry関数を呼び出すことにより、以降の処理で時計(クロック)ドライバを μ T-Kernel/SMのデバイス管理機能を使って利用可能となります。

2.2 時計(クロック)ドライバの仕様

2.2.1 対象デバイス

リアルタイムクロックにより時刻を管理するデバイスです。

2.2.2 デバイス名

デバイス名は“CLOCK”です。

<dev_rtc.h>をインクルードすることでRTC_DEVNMマクロ名で参照できます。

2.2.3 固有機能

リアルタイムクロックの時刻設定/取得が可能です。

ただし、ハードウェア固有の機能はサポートしていません。

2.2.4 属性データ

以下の属性データのみをサポートします。

DN_CKDATETIME : 現在時刻の設定/取得 (RW : 読み込み/書き込み可)

```
data : DATE_TIM
typedef struct {
    W d_year;    /* 1900年からのオフセット(85～) */
    W d_month;   /* 月 (1 ~ 12, 0) */
    W d_day;     /* 日 (1 ~ 31) */
    W d_hour;    /* 時 (0 ~ 23) */
    W d_min;     /* 分 (0 ~ 59) */
    W d_sec;     /* 秒 (0 ~ 59) */
    W d_week;    /* 週 (1 ~ 54) ※使用しない */
    W d_wday;    /* 曜日 (0 ~ 6, 0 が日曜) */
    W d_days;    /* 日 (1 ~ 366) ※使用しない */
} DATE_TIM;
```

現在時刻(ローカル時間)をリアルタイムクロックへ設定または取得します。

d_wdayは、設定する時に誤った曜日を設定してもチェックされません。したがって、取得した曜日も必ずしも正しいことは保証されません。d_week、d_daysは使用しません。これらの値は不定となります。

2.2.5 固有データ

なし

2.2.6 事象通知

未サポート

2.2.7 エラーコード

μT-Kernel3.0仕様書のデバイス管理機能の項を参照ください。時計(クロック)ドライバ固有の特殊なエラーコードは存在しません。

2.3 ヘッダファイルとサービス関数

ヘッダファイルはdev_rtc.hです。

時計(クロック)ドライバを登録するためのサービス関数の仕様は以下の通りです。

```
ER ercd = rtcDrvEntry(void);
```

2.4 サンプルプログラム

時計(クロック)ドライバをrtcDrvEntry関数で登録、tk_opn_devシステムコールでオープン後、現在時刻をターミナルより入力し、tk_swri_devシステムコールで設定を行います。その後は1msの間隔で現在時刻をtk_srea_devシステムコールで取得し、ターミナルに出力します。

```
#include <stdio.h>
#include <tk/tkernel.h>
#include <tm/tmonitor.h>
#include <dev_rtc.h>

LOCAL DATE_TIM dt;
LOCAL VB buf[32];

EXPORT INT usermain( void )
{
    ID dd;
    SZ asize;

    if( rtcDrvEntry() < E_OK )                // RTCドライバを登録
        goto ERROR;
    if( ( dd = tk_opn_dev( RTC_DEVNM, TD_UPDATE ) ) < E_OK )    // RTCドライバをオープン
        goto ERROR;

    tm_putstring( "Input now date and time.¥n"
                  "Year:Month:Day:Week:Hour:Minute:Second¥n"
                  "Week is 0 -> Sunday ... 6 -> Saturday¥n"
                  "Ex. 2000:1:1:6:12:34:56¥n¥n");                // 現在時刻を入力
    tm_getline( buf );
    sscanf( buf, "%ld:%ld:%ld:%ld:%ld:%ld", &dt.d_year, &dt.d_month, &dt.d_day, &dt.d_wday, &dt.d_hour,
                                                    &dt.d_min, &dt.d_sec );

    tm_putstring( "¥n");
    dt.d_year -= 1900;                                           // 現在時刻をRTCドライバに書き込み
    if( tk_swri_dev( dd, DN_CKDATETIME, &dt, sizeof(dt), &asize ) < E_OK || asize != sizeof(dt) )
        goto ERROR;
    while( 1 ) {
        tk_dly_tsk( 1 );
        if( tk_srea_dev( dd, DN_CKDATETIME, &dt, sizeof(dt), &asize ) < E_OK || asize != sizeof(dt) )
            goto ERROR;                                         // 現在時刻をRTCドライバから読み込み
        dt.d_year += 1900;
        tm_printf( "¥r¥ld:¥.2ld:¥.2ld:¥ld:¥.2ld:¥.2ld:¥.2ld", dt.d_year, dt.d_month, dt.d_day, dt.d_wday,
                                                            dt.d_hour, dt.d_min, dt.d_sec );
    }
}
```



```
}  
ERROR:  
    return 0;  
}
```

時計(クロック)ドライバのサンプルプログラム (usermain_rtc.c)

サンプルプログラムの実行結果は以下の通りです。

```
Input now date and time.  
Year:Month:Day:Week:Hour:Minute:Second  
Week is 0 -> Sunday ... 6 -> Saturday  
Ex. 2000:1:1:6:12:34:56  
  
2022:11:7:1:14:13:30  
2022:11:07:1:14:13:40    ← この現在時刻が1秒毎に変化します
```

2.5 スタックサイズ

時計(クロック)ドライバの処理関数の中で各構築仕様書の5.3.7項に記載された加算条件を超えるものは存在しません。結果、rtcDrvEntry関数を含め、Call Walkerに表示された値はそのまま各スタックサイズの計算式に利用可能です

3. RAM ディスクドライバ

3.1 RAM ディスクドライバの概要

RAMディスクドライバはターゲットボード搭載の外部接続RAM（SRAMまたはSDRAM）をディスクとして操作するためのドライバです。インタフェース仕様はT-Engine標準デバイスドライバ仕様に準拠しています。また、RAMディスクドライバを μ T-Kernelに登録するためのrdDrvEntry関数をサービス関数として提供します。ユーザシステムではrdDrvEntry関数を呼び出すことにより、以降の処理でRAMディスクドライバを μ T-Kernel/SMのデバイス管理機能を使って利用可能となります。

3.2 RAM ディスクドライバの仕様

3.2.1 対象デバイス

RAMディスクを管理するデバイスです。

3.2.2 デバイス名

デバイス名は“rda”です。

<dev_disk.h>をインクルードすることでRAM_DISK_DEVNMマクロ名で参照できます。

3.2.3 固有機能

区画のサポート

物理フォーマットのサポート

※論理フォーマットはアプリケーション（formatコマンド等）で行います。

3.2.4 属性データ

以下の属性データをサポートします。

R：読み込みのみ可

W：書き込みのみ可

```
/* ディスク属性データ番号 */
typedef enum {
    DN_DISKINFO      = TDN_DISKINFO, /* ディスク情報 */
    DN_DISKFORMAT    = -100,          /* ディスクフォーマット */
    DN_DISKMEMADR     = -103,          /* メモリディスク領域先頭アドレス */
    DN_DISKCHSINFO   = -105,          /* ディスクCHS情報 */
} DiskDataNo;
```

DN_DISKINFO：ディスク情報 (R)
data：DiskInfo

```
typedef struct {
    DiskFormat format; /* フォーマット形式 */
    BOOL protect:1;    /* プロテクト状態 */
    BOOL removable:1;  /* 取り外し可否 */
}
```

```

        UW          rsv:30;          /* 予約(0)          */
        W           blocksize;       /* ブロックバイト数 */
        W           blockcont;       /* 総ブロック数     */
    } DiskInfo;

```

format : フォーマット形式
 protect : ハード的に書き込みが禁止されている状態
 removable : 取り外し可否
 blocksize : 物理ブロックサイズ(バイト数)
 blockcont : 総ブロック数

ディスク情報を取り出します。

DN_DISKFORMAT : ディスクフォーマット (W)
 data : DiskFormat

```

typedef enum
    DiskFmt_MEM = -1,          /* メモリディスク */
} DiskFormat;

```

正しいフォーマット種別を書き込むことにより物理フォーマットを開始します。
RAMディスクなので全ブロックを一定の値(0xFF)で埋め、ディスク上の情報を完全に消します。

DN_DISKMEMADR : ディスク領域先頭アドレス (R)
 data : void *

ディスクとして使用するメモリの先頭アドレス（論理アドレス）を取り出します。
 このアドレスから、DiskInfoで得られるディスク容量（blocksize * blockcontバイト）分の連続した物理メモリ空間がディスクとして使用されるメモリです。
 任意にアクセスできるのはtk_opn_devシステムコールによってデバイスをオープンしているときのみです。tk_cls_devシステムコールによってデバイスをクローズした後はアクセスしてはいけません。また、一旦クローズし、再度オープンした場合には、ディスク領域の先頭アドレスも再度取り出し、そのアドレスを使用してアクセスしなければなりません。

DN_DISKCHSINFO : ディスクCHS情報 (R)
 data : DiskCHSInfo

```

typedef struct {
    W cylinder;          /* 総シリンダ数          */
    W head;              /* シリンダ当たりのヘッド数 */
    W sector;            /* ヘッド当たりのセクタ数 */
} DiskCHSInfo;

```

ディスクのシリンダ(C)、ヘッド(H)、セクタ(S) 情報を取り出します。
 RAMディスクなので、C = 1, H = 1, S = 総ブロック数となります。

3.2.5 固有データ

以下の固有データをサポートします。

データ番号 (0 ~) : ディスクのブロック番号

データ数 : 読み込み/書き込みのブロック数

3.2.6 事象通知

未サポート

3.2.7 エラーコード

μ T-Kernel3.0仕様書のデバイス管理機能の項を参照ください。RAMディスクドライバ固有の特殊なエラーコードは存在しません。

3.3 ヘッダファイルとサービス関数

ヘッダファイルは dev_disk.h です。

RAM ディスクドライバを登録するためのサービス関数の仕様は以下の通りです。

```
ER ercd = rdDrvEntry(void);
```

3.4 サンプルプログラム

RAMディスクドライバはFatFsから利用されるデバイスドライバであるため、ユーザシステムより直接呼出しを行うのはRAMディスクドライバをシステムに登録するためのrdDrvEntry関数のみです。

また、 μ T-Kernel3.0仕様において、デバイスドライバは複数のタスクからの利用を考慮して再入可能であることが要求されます。しかしながら、**RAMディスクドライバの処理関数では再入可能とするための排他制御は行いません。理由は、FatFsにも排他制御を行うための機能があるため、RAMディスクドライバを複数のタスクで使用する場合はFatFsが持つ排他制御機能を利用してください。**

3.5 スタックサイズ

RAMディスクドライバの処理関数の中で各構築仕様書の5.3.7項に記載された加算条件を超えるものは存在しません。結果、rdDrvEntry関数を含め、Call Walkerに表示された値はそのまま各スタックサイズの計算式に利用可能です

4. SD カードドライバ

4.1 SD カードドライバの概要

SDカードドライバはターゲットボード搭載のMicroSDカード（内蔵SDHI接続）をディスクとして操作するためのドライバです。インタフェース仕様はT-Engine標準デバイスドライバ仕様に準拠しています。また、SDカードドライバを μ T-Kernelに登録するためのsdDrvEntry関数をサービス関数として提供します。ユーザシステムではsdDrvEntry関数を呼び出すことにより、以降の処理でSDカードドライバを μ T-Kernel/SMのデバイス管理機能を使って利用可能となります。

4.2 SD カードドライバの仕様

4.2.1 対象デバイス

SDカードを管理するデバイスです。

4.2.2 デバイス名

デバイス名は“sda”です。

<dev_disk.h>をインクルードすることでSD_CARD_DEVMマクロ名で参照できます。

4.2.3 固有機能

区画のサポート

4.2.4 属性データ

以下の属性データをサポートします。

R：読み込みのみ可

W：書き込みのみ可

```
/* ディスク属性データ番号 */
typedef enum {
    DN_DISKINFO      = TDN_DISKINFO, /* ディスク情報 */
    DN_DISKFORMAT    = -100,          /* ディスクフォーマット */
    DN_DISKMEMADR     = -103,          /* メモリディスク領域先頭アドレス */
    DN_DISKCHSINFO   = -105,          /* ディスクCHS情報 */
} DiskDataNo;
```

DN_DISKINFO：ディスク情報 (R)
data : DiskInfo

```
typedef struct {
    DiskFormat format; /* フォーマット形式 */
    BOOL protect;      /* プロテクト状態 */
    BOOL removable;    /* 取り外し可否 */
    UW rsv;             /* 予約 (0) */
    W blocksize;        /* ブロックバイト数 */
    W blockcont;        /* 総ブロック数 */
}
```

```

    } DiskInfo;

format :    フォーマット形式
protect :   ハード的に書き込みが禁止されている状態
removable : 取り外し可否
blocksize : 物理ブロックサイズ(バイト数)
blockcont : 総ブロック数

```

ディスク情報を取り出します。

DN_DISKCHSINFO : ディスクCHS情報(R)
data : DiskCHSInfo

```

typedef struct {
    W cylinder;    /* 総シリンダ数          */
    W head;        /* シリンダ当たりのヘッド数 */
    W sector;      /* ヘッド当たりのセクタ数   */
} DiskCHSInfo;

```

ディスクのシリンダ(C)、ヘッド(H)、セクタ(S) 情報を取り出します。
SDカードなので、C = 1, H = 1, S = 総ブロック数となります。

4.2.5 固有データ

以下の固有データをサポートします。

データ番号 (0 ~) : ディスクのブロック番号
データ数 : 読み込み/書き込みのブロック数

4.2.6 事象通知

未サポート

4.2.7 エラーコード

μT-Kernel3.0仕様書のデバイス管理機能の項を参照ください。SDカードドライバ固有の特殊なエラーコードは存在しません。

4.3 SD カードドライバが使用する資源

4.3.1 使用する資源の概要

SDカードドライバは処理を実施するに当たり、1つのイベントフラグ、1つのタスク、2つの割込みハンドラを利用します。これらの資源は4.5節で紹介するsdDrvEntry関数を呼び出すと生成または定義されます。

使用する資源	名称	優先度
タスク	sdc_tsk	タスクの優先度は SD_CFG_TASK_PRIORITY
割込みハンドラ	SD_Detect_hdr	割込みハンドラの割込み優先度は SD_CFG_INT_PRIORITY
	SD_Int_hdr	

注：SD_CFG_TASK_PRIORITY と SD_CFG_INT_PRIORITY は次節を参照

4.3.2 イベントフラグ

SDカードドライバ内で利用するイベントフラグです。ユーザアプリケーションからのAPIの受け付けキュー、割込みハンドラとタスクの同期処理に利用します。

項目	値
拡張情報	不定
イベントフラグ属性	TA_TPRI TA_WMUL TA_DSNAME
DSオブジェクト名称	"shc_f"

備考：

TA_DSNAME、TA_USERBUFのタスク属性とDSオブジェクト名称はカーネルのコンフィグレーションによって未サポートとなる場合があります。

4.3.3 タスク (shc_tsk)

SDカードの制御を行うタスクです。ユーザアプリケーションからのAPIの受け付け、SDカードの制御を行います。SDカードの挿入・抜去にも対応します。ただし、**SD規格の「Physical Spec Version 2.00」に対応したSDカードのみを対象としています**。SD規格の「Physical Spec Version 1.x」には対応していません。

項目	値
拡張情報	不定
タスク属性	TA_HLNG TA_DSNAME TA_USERBUF
タスク起動アドレス	shc_tsk
タスク起動時優先度	SD_CFG_TASK_PRIORITY
スタックサイズ	320バイト
DSオブジェクト名称	"shc_t"

備考：

TA_DSNAME、TA_USERBUFのタスク属性とDSオブジェクト名称はカーネルのコンフィグレーションによって未サポートとなる場合があります。

4.3.4 割込みハンドラ (SD_Detect_hdr、SD_Int_hdr)

SDホストインタフェース (SDHI) の割込みハンドラです。割込みの発生によりイベントフラグ経由でshc_tskを動作させます。

項目	ターゲット	値
----	-------	---

割り込み番号	AP-RX65N-0A	113 (GROUPBL1)
	AP-RX72N-0A	113 (GROUPBL1)
割り込み優先度	—	SD_CFG_INT_PRIORITY
割り込みハンドラ起動アドレス	—	SD_Detect_hdr : CDETI (カード検出割り込み)
		SD_Int_hdr : CACI (カードアクセス割り込み)

4.3.5 グループ割り込み (GroupBL1Handler)

現在サポートしているターゲットでは、SDホストインタフェース (SDHI) の割り込みは割り込み番号 (ベクタ番号) 113のグループ割り込み (GROUPBL1) です。ただし、システムで割り込み番号113のグループ割り込みに分類された割り込みを利用する場合は、割り込みハンドラ (GroupBL1Handler) を修正し、他の割り込み要因と共有する必要があります。割り込みハンドラのソースは、mtkernel_3¥device¥sd¥sysdepend¥ターゲット¥sd_driver.c にあるGroupBL1Handler関数です。

現状の割り込みハンドラ (sd_driver.c)

```
EXPORT void GroupBL1Handler (UINT dintno)
{
    if ( IS ( SDHI, CDETI ) )           // Occur SDHI CDETI Interrupt ?
        SD_Detect_hdr ( );             // Call SDHI CDETI Interrupt
    if ( IS ( SDHI, CACI ) )           // Occur SDHI CACI Interrupt ?
        SD_Int_hdr ( );                // Call SDHI CACI Interrupt
}
```

GroupBL1Handler割り込みハンドラの中でGRPBL1レジスタの各ステータスフラグをチェックし、各割り込みハンドラを呼び出してください。

4.4 SD カードドライバのコンフィグレーション

4.4.1 SD カードドライバのコンフィグレーション・ファイル

SDカードドライバのコンフィグレーション・ファイルは、mtkernel_3¥device¥sd¥sysdepend¥ターゲット¥sd_config.h です。

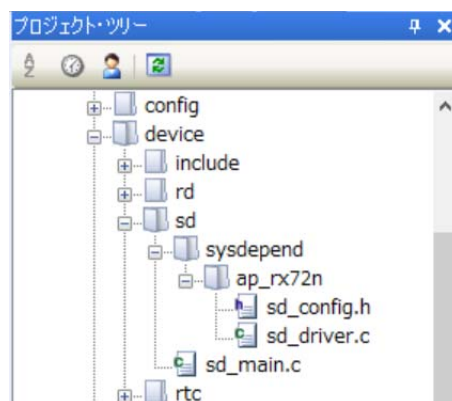


図4.1 SDカードドライバのコンフィグレーション・ファイル (AP-RX72N-0Aの例)

4.4.2 タスクの優先度と割り込みハンドラの割り込み優先度

● SD_CFG_TASK_PRIORITY

SDカードの制御を行うタスク（sd_c_tsk）のタスク優先度です。

システムの都合に応じて、1 ～ CFN_MAX_TSKPRI（タスク優先度の最大値）の範囲で変更できます。最低でもSDカードドライバを利用するタスクの優先度よりも高く設定することを推奨します。デフォルトは 3 に設定されています。

● SD_CFG_INT_PRIORITY

内蔵周辺機能のSDホストインタフェース（SDHI）の割り込み優先度です。

システムの都合に応じて、1 ～ MAX_INT_PRI（最高外部割り込み優先度）の範囲で変更できます。TIM_INT_PRI（システムタイマの割り込み優先度）よりも低く設定することを推奨します。デフォルトでは 8 に設定されています。

```
/* SD control task priority. */  
#define SD_CFG_TASK_PRIORITY          (3)  
  
/* SD interrupt priority level. */  
#define SD_CFG_INT_PRIORITY          (8)
```

4.4.3 DMAC のチャンネル番号

● SD_CFG_DMA_CHANNEL

SDカードとの間のデータ転送に利用するDMACのチャンネル番号です。

システムの都合に応じて、0 ～ 7 の範囲で変更できます（範囲以外の値の場合は 7 の指定となります）。デフォルトでは 4 に設定されています。

```
/* SD DMA channel. */  
#define SD_CFG_DMA_CHANNEL          (4)
```

4.5 ヘッダファイルとサービス関数

ヘッダファイルは dev_disk.h です。

SD カードドライバを登録するためのサービス関数の仕様は以下の通りです。

```
ER ercd = sdDrvEntry(void);
```

SD カードの挿入を待つためのサービス関数の仕様は以下の通りです。本サービス関数は SD カードが挿入されていない場合、WAITING 状態となります。

```
ER ercd = sdWaitInsertEvent(TMO tmout);
```

SD カードの抜去を待つためのサービス関数の仕様は以下の通りです。本サービス関数は SD カードが挿入されている場合、WAITING 状態となります。

```
ER ercd = sdWaitRejectEvent(TMO tmout);
```

4.6 サンプルプログラム

SDカードドライバはFatFsから利用されるデバイスドライバであるため、ユーザシステムより直接呼出しを行うのは4.5節で紹介したサービス関数のみです。

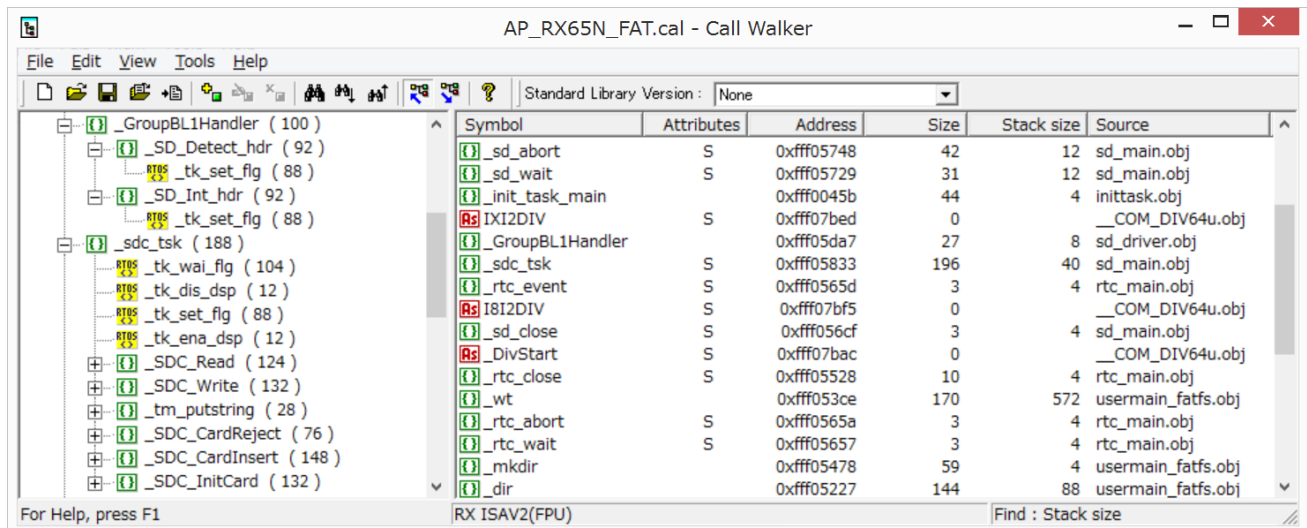
また、 μ T-Kernel3.0仕様において、デバイスドライバは複数のタスクからの利用を考慮して再入可能であることが要求されます。しかしながら、SDカードドライバの処理関数では再入可能とするための排他制御は行いません。理由は、FatFsにも排他制御を行うための機能があるため、SDカードドライバを複数のタスクで使用する場合はFatFsが持つ排他制御機能を利用してください。

4.7 スタックサイズ

4.7.1 スタック見積もりツール

本実装においてもスタック見積もりツールを使用することが可能です。スタック見積もりツールの起動方法やリアルタイムOSオプションの設定等はターゲット毎の構築仕様書の第5章を参照ください。

以降はスタック見積もりツールを使用することを前提に各スタックサイズを紹介します。以下にスタック見積もりツールの表示例を示します。



Symbol	Attributes	Address	Size	Stack size	Source
_sd_abort	S	0xffff05748	42	12	sd_main.obj
_sd_wait	S	0xffff05729	31	12	sd_main.obj
_init_task_main		0xffff0045b	44	4	inittask.obj
__COM_DIV64u.obj		0xffff07bed	0		
_sd_driver.obj		0xffff05da7	27	8	sd_driver.obj
_sd_tsk	S	0xffff05833	196	40	sd_main.obj
_rtc_event	S	0xffff0565d	3	4	rtc_main.obj
__COM_DIV64u.obj		0xffff07bf5	0		
_sd_close	S	0xffff056cf	3	4	sd_main.obj
__COM_DIV64u.obj		0xffff07bac	0		
_rtc_close	S	0xffff05528	10	4	rtc_main.obj
_wt		0xffff053ce	170	572	usermain_fatfs.obj
_rtc_abort	S	0xffff0565a	3	4	rtc_main.obj
_rtc_wait	S	0xffff05657	3	4	rtc_main.obj
_mkdir		0xffff05478	59	4	usermain_fatfs.obj
_dir		0xffff05227	144	88	usermain_fatfs.obj

図4.2 SDカードドライバのスタック見積もり結果

4.7.2 sdc_tsk のスタックサイズ

sdc_tskタスクが独自に使用するスタックサイズは188バイトです。コンパイラのバージョン等が変化しても、それほど大きな違いはないはずです。このサイズにタスクコンテキストのサイズを加えても、現状のサンプルで確保されている320バイトは超えないと考えて構いません。

ただし、カーネル管理外割込みを複数レベル使用すると320バイトを超える可能性があります。も

し、カーネル管理外割込みを複数レベル使用する場合はターゲット毎の構築仕様書の第5章の内容に従ってスタックサイズを算出し、mtkernel_3¥device¥sd¥sd_maini.cで生成・確保されているスタックサイズを変更してください。

```
u.t_ctsk.stksz = 320;           // Set Task StackSize
u.t_ctsk.itskpri = SDC_GetTaskPri(); // Set Task Priority
```

4.7.3 GroupBL1Handler のスタックサイズ

GroupBL1Handler割込みハンドラが独自に使用するスタックサイズは100バイトです。このサイズがSDカードドライバのコンフィグレーションで指定した SD_CFG_INT_PRIORITY 割込み優先度で実行される割込みハンドラのスタックサイズとなります。この数値を使って例外スタックのサイズを算出してください。ただし、4.3.5項で説明した通り、SDカードドライバの割込みはグループ割込みです。同じグループの割込みをサポートした時は、解析ツールの表示結果に従って、数値を見直してください。詳しくはターゲット毎の構築仕様書の第5章を参照してください。

4.7.4 SD カードドライバの処理関数

SDカードドライバの処理関数の中で各構築仕様書の5.3.7項に記載された加算条件を超えるものは存在しません。結果、sdDrvEntry関数を含め、Call Walkerに表示された値はそのまま各スタックサイズの計算式に利用可能です。

5. USB メモリドライバ

5.1 USB メモリドライバの概要

USBメモリドライバはターゲットボード搭載のUSBコネクタ（内蔵USBホスト接続）に接続されたUSBメモリをディスクとして操作するためのドライバです。インタフェース仕様はT-Engine標準デバイスドライバ仕様に準拠しています。また、USBメモリドライバを μ T-Kernelに登録するためのusbDrvEntry関数をサービス関数として提供します。ユーザシステムではusbDrvEntry関数を呼び出すことにより、以降の処理でUSBメモリドライバを μ T-Kernel/SMのデバイス管理機能を使って利用可能となります。

5.2 USB メモリドライバの仕様

5.2.1 対象デバイス

USBメモリを管理するデバイスです。ただし、HUBはサポートしていません。

5.2.2 デバイス名

デバイス名は“uda”です。

<dev_disk.h>をインクルードすることでUSB_MSC_DEVMマクロ名で参照できます。

5.2.3 固有機能

区画のサポート

5.2.4 属性データ

以下の属性データをサポートします。

R：読み込みのみ可

W：書き込みのみ可

```
/* ディスク属性データ番号 */
typedef enum {
    DN_DISKINFO      = TDN_DISKINFO, /* ディスク情報 */
    DN_DISKFORMAT    = -100,          /* ディスクフォーマット */
    DN_DISKMEMADR     = -103,          /* メモリディスク領域先頭アドレス */
    DN_DISKCHSINFO   = -105,          /* ディスクCHS情報 */
} DiskDataNo;
```

DN_DISKINFO：ディスク情報 (R)
data : DiskInfo

```
typedef struct {
    DiskFormat format; /* フォーマット形式 */
    BOOL protect;      /* プロテクト状態 */
    BOOL removable;    /* 取り外し可否 */
    UW rsv;             /* 予約 (0) */
    W blocksize;        /* ブロックバイト数 */
    W blockcont;        /* 総ブロック数 */
}
```

```
} DiskInfo;
```

```
format :   フォーマット形式
protect :   ハード的に書き込みが禁止されている状態
removable :   取り外し可否
blocksize :   物理ブロックサイズ(バイト数)
blockcont :   総ブロック数
```

ディスク情報を取り出します。

DN_DISKCHSINFO : ディスクCHS情報 (R)
data : DiskCHSInfo

```
typedef struct {
    W cylinder;    /* 総シリンダ数          */
    W head;        /* シリンダ当たりのヘッド数 */
    W sector;      /* ヘッド当たりのセクタ数   */
} DiskCHSInfo;
```

ディスクのシリンダ(C)、ヘッド(H)、セクタ(S) 情報を取り出します。
SDカードなので、C = 1, H = 1, S = 総ブロック数となります。

5.2.5 固有データ

以下の固有データをサポートします。

データ番号 (0 ~) : ディスクのブロック番号

データ数 : 読み込み/書き込みのブロック数

5.2.6 事象通知

未サポート

5.2.7 エラーコード

μT-Kernel3.0仕様書のデバイス管理機能の項を参照ください。USBメモリドライバ固有の特殊なエラーコードは存在しません。

5.3 USBメモリドライバが使用する資源

5.3.1 使用する資源の概要

USBメモリドライバは処理を実施するに当たり、1つのイベントフラグ、1つのタスク、2つの割り込みハンドラを利用します。これらの資源は5.5節で紹介するusbDrvEntry関数を呼び出すと生成または定義されます。

使用する資源	名称	優先度
タスク	usb_tsk	タスクの優先度は USB_CFG_TASK_PRIORITY
割り込みハンドラ	USB_Int_hdr	割り込みハンドラの割り込み優先度は USB_CFG_INT_PRIORITY
	DMA_End_hdr	

注：USB_CFG_TASK_PRIORITY と USB_CFG_INT_PRIORITY は次節を参照

DMACのチャンネル4～7を利用すると DMA_End_hdr は r_dmaca_intdmac74i_isr となります

5.3.2 イベントフラグ

USBメモリドライバ内で利用するイベントフラグです。ユーザアプリケーションからのAPIの受け付けキュー、割込みハンドラとタスクの同期処理に利用します。

項目	値
拡張情報	不定
イベントフラグ属性	TA_TPRI TA_WMUL TA_DSNAME
DSオブジェクト名称	"usb_f"

備考：

TA_DSNAME、TA_USERBUFのタスク属性とDSオブジェクト名称はカーネルのコンフィグレーションによって未サポートとなる場合があります。

5.3.3 タスク (usb_tsk)

USBメモリの制御を行うタスクです。ユーザアプリケーションからのAPIの受け付け、USBメモリの制御を行います。USBメモリの挿入・抜去にも対応します。ただし、HUBはサポートしていません。

項目	値
拡張情報	不定
タスク属性	TA_HLNG TA_DSNAME TA_USERBUF
タスク起動アドレス	usb_tsk
タスク起動時優先度	USB_CFG_TASK_PRIORITY
スタックサイズ	400バイト
DSオブジェクト名称	"usb_t"

備考：

TA_DSNAME、TA_USERBUFのタスク属性とDSオブジェクト名称はカーネルのコンフィグレーションによって未サポートとなる場合があります。

5.3.4 割込みハンドラ (USB_Int_hdr、DMA_End_hdr)

USBホストインタフェース (USB10) の割込みハンドラとDMACの割込みハンドラです。割込みの発生によりイベントフラグ経由でusb_tskを動作させます。なお、DMACの割込みハンドラはチャンネル4～7の場合、r_dmaca_intdmac74i_isr となります。

項目	ターゲット	値
----	-------	---

割込み番号	AP-RX63N-0A	35 (USB10)
	AP-RX65N-0A	USB_CFG_VECTOR_NUMBER (128～207)
	AP-RX72N-0A	USB_CFG_VECTOR_NUMBER (128～207)
割込み優先度	—	USB_CFG_INT_PRIORITY
割込みハンドラ起動アドレス	—	USB_Int_hdr : USB10 (USB割込み)

項目	ターゲット	値
割込み番号	AP-RX63N-0A	198～201 (DMAC0I～DMAC3I)
	AP-RX65N-0A	120～124 (DMAC0I～DMAC3I、DMAC74I)
	AP-RX72N-0A	120～124 (DMAC0I～DMAC3I、DMAC74I)
割込み優先度	—	USB_CFG_INT_PRIORITY
割込みハンドラ起動アドレス	—	DMA_End_hdr (DMAC0I～DMAC3I)
		DMAC74I_Handler (DMAC74I)

5.3.5 選択型割込みB (PERIB)

RX65NやRX72NではUSBホストインタフェース (USB10) の割込みは選択型割込みB (PERIB) となり、ベクタ番号128～207が選択できます。デフォルトでは128となっていますが、コンフィグレーションにより変更が可能となっています。

5.3.6 DMAC の割込み

DMACの割込みはチャンネル番号が 0 ～ 3 であれば個別の割込み番号 (120～123) となりますが、チャンネル番号が 4 ～ 7 の場合は共通の割込み番号 (124) となります。このため、チャンネル番号が 0 ～ 3 の場合は割込みハンドラとして DMA_End_hdr が当該の割込みハンドラとして定義されます。一方、チャンネル番号が 4 ～ 7 の場合は割込みハンドラとして DMAC74I_Handler が当該の割込みハンドラとして定義され、その中で割込み要因を調べて、対応したチャンネルの割込みハンドラを呼び出すようになっています。もし、コンフィグレーションで使用するDMACのチャンネル番号で 4 ～ 7 を指定した場合は、割込みハンドラ (DMAC74I_Handler) を修正し、他のDMACチャンネルと割込みと共有する必要があります。割込みハンドラのソースは、mtkernel_3¥device¥sd¥sysdepend¥ターゲット¥usb_driver.c にある DMAC74I_Handler関数です。

現状の割込みハンドラ (usb_driver.c)

```
#if USB_CFG_DMA_CHANNEL >= 4
LOCAL void DMAC74I_Handler (UINT intno)
{
    if( DMAC.DMIST.BYTE & 0x10 ) {           // Channel 4 Interrupt ?
    #if USB_CFG_DMA_CHANNEL == 4
        DMA_End_hdr ( intno );
    #endif
    }
    if( DMAC.DMIST.BYTE & 0x20 ) {           // Channel 5 Interrupt ?
    #if USB_CFG_DMA_CHANNEL == 5
```

```

DMA_End_hdr( intno );
#endif
    if( DMAC.DMIST.BYTE & 0x40 ) {           // Channel 6 Interrupt ?
#ifdef USB_CFG_DMA_CHANNEL == 6
        DMA_End_hdr( intno );
#endif
    }
    if( DMAC.DMIST.BYTE & 0x80 ) {           // Channel 7 Interrupt ?
#ifdef USB_CFG_DMA_CHANNEL == 7
        DMA_End_hdr( intno );
#endif
    }
}
#endif

```

DMAC74I_Handler割込みハンドラの中でDMISTレジスタの各ステータスフラグをチェックし、各割込みハンドラを呼び出してください。

5.4 USB メモリドライバのコンフィグレーション

5.4.1 USB メモリドライバのコンフィグレーション・ファイル

USBメモリドライバのコンフィグレーション・ファイルは、mtkernel_3¥device¥sd¥sysdepend¥ターゲット¥usb_config.h です。

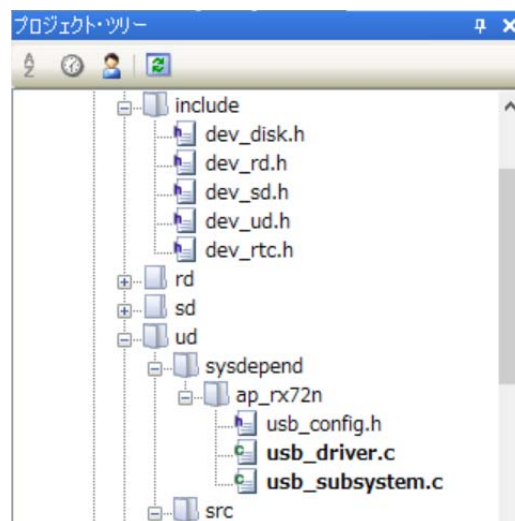


図5.1 USBメモリドライバのコンフィグレーション・ファイル（AP-RX72N-0Aの例）

5.4.2 タスクの優先度と割込みハンドラの割込み優先度

● USB_CFG_TASK_PRIORITY

USBメモリの制御を行うタスク（usb_tsk）のタスク優先度です。

システムの都合に応じて、1 ～ CFN_MAX_TSKPRI（タスク優先度の最大値）の範囲で変更できます。最低でもUSBメモリドライバを利用するタスクの優先度よりも高く設定することを推奨します。デフォルトは 3 に設定されています。

● USB_CFG_INT_PRIORITY

内蔵周辺機能のUSBホストインタフェース（USB0）とDMACの割込み優先度です。

システムの都合に応じて、1 ～ MAX_INT_PRI（最高外部割込み優先度）の範囲で変更できます。

TIM_INT_PRI（システムタイマの割込み優先度）よりも低く設定することを推奨します。デフォルトでは 8 に設定されています。

```
/* USB control task priority. */  
#define USB_CFG_TASK_PRIORITY          (3)  
  
/* USB interrupt priority level. */  
#define USB_CFG_INT_PRIORITY          (8)
```

5.4.3 選択型割込みBの割込み番号と DMAC のチャネル番号

● USB_CFG_VECTOR_NUMBER

USB0I割込みが選択型割込みBの場合の割込み番号です。RX63NIには本設定は存在しません。

システムの都合に応じて、128 ～ 207 の範囲で変更できます。デフォルトでは 128 に設定されています。

● USB_CFG_DMA_CHANNEL

USBメモリとの間のデータ転送に利用するDMACのチャネル番号です。

システムの都合に応じて、0 ～ 7 の範囲で変更できます。デフォルトでは 3 に設定されています。

固定の割込み番号となる 0 ～ 3 の範囲を推奨します。

```
/* USB vector number. */  
#define USB_CFG_VECTOR_NUMBER          (128)  
  
/* USB DMA channel number. */  
#define USB_CFG_DMA_CHANNEL            (3)
```

5.4.4 VSUB 信号のアクティブレベル

● USB_CFG_VBUS_ACTIVE

VBUS信号のアクティブレベルを指定します。

```
/* USB power source. 0:Low Active, 1:High Active */  
#define USB_CFG_VBUS_ACTIVE            (1)
```

5.5 ヘッダファイルとサービス関数

ヘッダファイルは dev_disk.h です。

USB メモリドライバを登録するためのサービス関数の仕様は以下の通りです。

```
ER ercd = usbDrvEntry(void);
```

USB メモリの挿入を待つためのサービス関数の仕様は以下の通りです。本サービス関数は USB メモリが挿入されていない場合、WAITING 状態となります。

```
ER ercd = usbWaitAttachEvent(TMO tmout);
```

USB メモリの抜去を待つためのサービス関数の仕様は以下の通りです。本サービス関数は USB メモリが挿入されている場合、WAITING 状態となります。

```
ER ercd = usbWaitDetachEvent(TMO tmout);
```

5.6 サンプルプログラム

USBメモリドライバはFatFsから利用されるデバイスドライバであるため、ユーザシステムより直接呼出しを行うのは5.5節で紹介したサービス関数のみです。

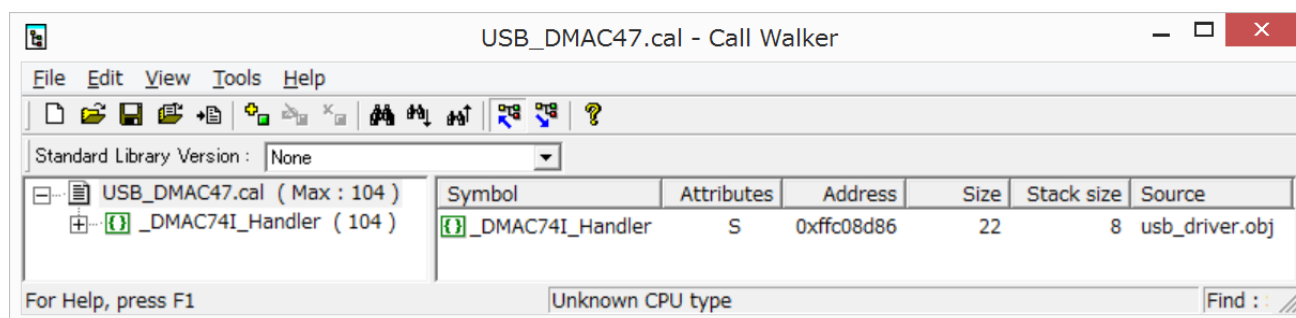
また、 μ T-Kernel3.0仕様において、デバイスドライバは複数のタスクからの利用を考慮して再入可能であることが要求されます。しかしながら、**USBメモリドライバの処理関数では再入可能とするための排他制御は行いません。理由は、FatFsにも排他制御を行うための機能があるため、USBメモリドライバを複数のタスクで使用する場合はFatFsが持つ排他制御機能を利用してください。**

5.7 スタックサイズ

5.7.1 スタック見積もりツール

本実装においてもスタック見積もりツールを使用することが可能です。スタック見積もりツールの起動方法やリアルタイムOSオプションの設定等はターゲット毎の構築仕様書の第5章を参照ください。

以降はスタック見積もりツールを使用することを前提に各スタックサイズを紹介します。以下にスタック見積もりツールの表示例を示します。



Symbol	Attributes	Address	Size	Stack size	Source
_ud_open (4)		0xffc06101	3	4	usb_main.obj
_USB_Int_hdr (112)		0xffc08c85	3	4	r_usb_rx_mcu.obj
_usb_tsk (276)	S	0xffc0625d	233	40	usb_main.obj
_DMA_End_hdr (96)		0xffc069aa	20	4	r_usb_dma.obj
_ud_event (4)	S	0xffc061a8	3	4	usb_main.obj
_ud_wait (116)	S	0xffc0615f	31	12	usb_main.obj
_ud_abort (100)	S	0xffc0617e	42	12	usb_main.obj
_ud_close (4)	S	0xffc06104	3	4	usb_main.obj
_ud_exec (100)	S	0xffc06107	88	12	usb_main.obj

図5.2 USBメモリドライバのスタック見積もり結果

5.7.2 usb_tsk のスタックサイズ

usb_tskタスクが独自に使用するスタックサイズは276バイトです。コンパイラのバージョン等が変化しても、それほど大きな違いはないはずです。このサイズにタスクコンテキストのサイズを加えても、**現状のサンプルで確保されている400バイトは超えない**と考えて構いません。

ただし、カーネル管理外割込みを複数レベル使用すると400バイトを超える可能性があります。もし、カーネル管理外割込みを複数レベル使用する場合はターゲット毎の構築仕様書の第5章の内容に従ってスタックサイズを算出し、mtkernel_3¥device¥sd¥usb_maini.cで生成・確保されているスタックサイズを変更してください。

```
u.t_ckpt.stksz = 400; // Set Task StackSize
u.t_ckpt.itkspri = USB_GetTaskPri(); // Set Task Priority
```

5.7.3 割込みハンドラのスタックサイズ

割込みハンドラが独自に使用するスタックサイズの最大値はUSB_Int_hdr割込みハンドラの112バイトです。このサイズがUSBメモリドライバのコンフィグレーションで指定した USB_CFG_INT_PRIORITY 割込み優先度で実行される割込みハンドラのスタックサイズとなります。この数値を使って例外スタックのサイズを算出してください。ただし、5.3.6項で説明した通り、DMAC74I_Handler割込みハンドラを使用する場合、他の割込みハンドラのサポートによって112バイトを超える可能性があります。その際は解析ツールの表示結果に従って、数値を見直してください。詳しくはターゲット毎の構築仕様書の第5章を参照してください。

5.7.4 USBメモリドライバの処理関数

USBメモリドライバの処理関数の中で各構築仕様書の5.3.7項に記載された加算条件を超えるものは存在しません。結果、usbDrvEntry関数を含め、Call Walkerに表示された値はそのまま各スタックサイズの計算式に利用可能です。

6. FatFs

6.1 FatFs とは

FatFsは、小規模な組み込みシステム向けの汎用FATファイルシステムモジュールです。

本実装におけるFatFsライブラリは、オープンソースとして公開されているFatFsのソースコードを μ T-Kernel3.0用に移植したものです。移植のベースとなったFatFsのバージョンR0.14bのソースコードは以下のURLからダウンロードできます。

http://elm-chan.org/fsw/ff/00index_e.html

FatFsでは、システムディスクドライバ（本実装ではRAMディスクドライバとSDカードドライバとUSBメモリドライバ）と時計（クロック）ドライバを利用します。これらはT-Engine標準デバイスドライバ仕様に準拠したデバイスドライバです。このため、FatFsも、T-Engine標準デバイスドライバ仕様のシステムディスクドライバで動作するように変更してあります。なお、ターゲットと各ドライバの対応は以下の通りです。

表6.1 ターゲットと各ドライバの対応

	RAMディスクドライバ	SDカードドライバ	USBメモリドライバ
	論理ドライブ番号：0	論理ドライブ番号：1	論理ドライブ番号：2
AP-RX63N-0A	サポート	未サポート	サポート
AP-RX65N-0A	サポート ^{注)}	サポート ^{注)}	サポート
AP-RX72N-0A	サポート	サポート	サポート

注) AP-RX65N-0Aはハードウェアの構成でRAMディスクとSDカードを同時に使用することができません。

デフォルトの設定では、SDカードドライバのみが動作するようになっています。

また、本実装における各ディスクの論理ドライブ番号は以下の通りです。

- ・ RAMディスク ➡ 0
- ・ SDカード ➡ 1
- ・ USBメモリ ➡ 2

6.2 FatFs の μ T-Kernel3.0 への移植

μ T-Kernel3.0用に移植したFatFsのソースコードは以下のディレクトリに含まれています。

mtkernel_3¥lib¥libfat

上記のディレクトリには、さらに2つのディレクトリが含まれており、各ディレクトリにはそれぞれ以下のソースコード、またはドキュメントが含まれています。

- ・ documents

FatFsの付属ドキュメントが含まれています。

- ・ source

FatFs本体のソースコードが含まれています。

FatFsを μ T-Kernel3.0用に移植するために、以下の項目について修正しています。

- ・ システム機能関連の定義を μ T-Kernelの機能を利用して実装
 - ・ デバイス入出力を μ T-Kernel/SMのデバイス管理機能に準拠した形式で実装
- その他の箇所は改変していません。

6.2.1 システム関連機能を μ T-Kernel 対応に変更

FatFsをリエントラント対応で動作させる場合は、FF_SYNC_tというオブジェクトを定義して排他制御を行う設計になっています。FF_SYNC_tはOSによって提供される排他制御機能を利用して実装します。

本実装では、 μ T-Kernelのセマフォまたはミューティクス（コンフィグレーションで選択可能）を利用しています。具体的に、以下のファイルを μ T-Kernelのセマフォまたはミューティクスを利用する方式に変更しています。

```
mtkernel_3¥lib¥libfat¥source¥ffconf.h
mtkernel_3¥lib¥libfat¥source¥ffsystem.c
```

6.2.2 デバイス入出力を μ T-Kernel 対応に変更

μ T-Kernelでは、デバイスドライバのインタフェースをT-Engine標準デバイスドライバ仕様として規定しています。本実装のFatFsでは、ディスク操作のための関数に上記のドライバ仕様を適用しています（ただし、ドライバ側での排他制御は実施していません）。具体的には、以下のファイルをシステムディスクドライバに準拠した方式に変更しています。

```
mtkernel_3¥lib¥libfat¥source¥diskio.c
```

また、FatFsではget_fattime関数でタイムスタンプの時刻を取得しますが、その時刻管理もT-Engine標準デバイスドライバ仕様に準拠した時計（クロック）ドライバを利用する方式に変更しています。具体的には、以下のファイルに含まれるget_fattime関数を時計（クロック）ドライバに準拠した方式に変更しています。

```
mtkernel_3¥lib¥libfat¥source¥fatlower.c
```

6.2.3 FatFsの機能制限

μ T-Kernel3.0用に移植したFatFsは、公開されているFatFs R0.14bをそのまま利用しています。オリジナルのFatFsに実装されていない機能を追加したり、オリジナルのFatFsに実装されていた機能を削除したりするといった改造は行っていません。

FatFsの機能の詳細に関しては、FatFsの付属ドキュメント、またはソースコードを確認してください。

```
mtkernel_3¥lib¥libfat¥documents¥00index_e.html
```

6.3 FatFsの構成オプション

FatFsライブラリでは、サポートする機能の有効／無効を選択することができます。

以下のファイルで定義している各マクロの設定値を変更してから、ビルドを行ってください。

mtkernel_3¥lib¥libfat¥source¥ffconf.h

6.3.1 機能構成

● FF_FS_READONLY

0：リード/ライト（デフォルト値）

1：リードオンリー

リードオンリー構成では、書き込みAPI関数、f_write、f_sync、f_unlink、f_mkdir、f_chmod、f_rename、f_truncate、f_getfreeとオプションの書き込み関数が削除されます。

● FF_FS_MINIMIZE

基本API関数を段階的に削除します。

値	説明
0	全ての基本API関数が利用可能。（デフォルト値）
1	f_stat、f_getfree、f_unlink、f_mkdir、f_chmod、f_utime、f_truncate、f_rename関数が削除されます。
2	1に加え、f_opendir、f_readdir、f_closedir関数が削除されます。
3	2に加え、f_lseek関数が削除されます。

● FF_USE_FIND

フィルタ付きディレクトリ読み出し機能の構成（0：無効（デフォルト値）または 1：有効）。有効にすると、f_findfirst、f_findnext関数が利用可能になります。FF_FS_MINIMIZEは、1以下でなければなりません。

● FF_USE_MKFS

ボリューム作成機能の構成（0：無効 または 1：有効（デフォルト値））。有効にするとf_mkfs関数が利用可能になります。

● FF_USE_FASTSEEK

高速シーク機能の構成（0：無効（デフォルト値）または 1：有効）。

● FF_USE_EXPAND

連続領域割り当て機能の構成（0：無効（デフォルト値）または 1：有効）。有効にするとf_expand関数が利用可能になります。

● FF_USE_CHMOD

メタデータ操作機能の構成（0：無効（デフォルト値）または 1：有効）。有効にすると、f_chmod、

f_utime関数が利用可能になります。FF_FS_READONLY = 0 でなければなりません。

● FF_USE_LABEL

ボリュームラベル操作機能の構成（0：無効（デフォルト値）または 1：有効）。有効にすると、f_getlabel、f_setlabel関数が利用可能になります。

● FF_USE_FORWARD

ストリーミング読み出し機能（f_forward関数）の構成（0：無効（デフォルト値）または 1：有効）。

● FF_USE_STRFUNC

文字列入出力API関数f_gets、f_putc、f_puts、f_printfの構成。

値	説明
0	文字列入出力API関数を使用しない。（デフォルト値）
1	文字列入出力API関数を使用する。データのLF-CRLF変換はしない。
2	文字列入出力API関数を使用する。データのLF-CRLF変換をする。

● FF_PRINT_LLI

f_printfでのlong long型のサポート（0：未サポート（デフォルト値）または 1：サポート）。

● FF_PRINT_FLOAT

f_printfでの浮動小数点型のサポート（0：未サポート（デフォルト値）または 1：サポート）。

● FF_STRF_ENCODE

LFNが有効で FF_LFN_UNICODE ≥ 1 の場合、文字列関数は文字エンコーディングを変換します。

FF_STRF_ENCODEは、f_gets、f_putc、f_puts、f_printfの関数を介して読み書きされるファイルのエンコーディングを指定します。

値	説明
0	ANSI/OEM
1	UTF-16LE
2	UTF-16BE
3	UTF-8

5.3.2 ロケールと名前空間の構成

● FF_CODE_PAGE

パス名等の文字列データのコード ページを指定します。不適切な設定はファイル オープン エラーの原因になる可能性があります。

値	説明
932	日本語（DBCS）（デフォルト値）
その他	マニュアルを参照ください。

● FF_USE_LFN

長いファイル名（LFN）のサポートを指定します。LFN操作の作業バッファとして（FF_MAX_LFN + 1） * 2バイト（exFAT構成時はさらに（FF_MAX_LFN + 44） / 15 * 32バイト）を使用します。このため、作業バッファにスタックを使用する場合、スタックオーバーフローに注意してください。

値	説明
0	LFN機能を使わない。8.3形式の名前のみ使用可能。（デフォルト値）
1	LFN機能を使う。作業バッファは静的に確保。常にスレッド セーフではない。
2	LFN機能を使う。作業バッファはスタックに確保。
3	LFN機能を使う。作業バッファはヒープに確保。

● FF_MAX_LFN

LFN作業バッファのサイズを文字単位で指定（12～255）します。LFN機能が無効のときは意味を持ちません。

● FF_LFN_UNICODE

LFN機能が有効な場合にAPIの文字エンコーディングを指定します。

値	説明	TCHAR型
0	ANSI/OEM	char
1	UTF-16	WCHAR
2	UTF-8	char
3	UTF-32	DWORD

また、文字列I/O関数の動作もこのオプションの影響を受けます。LFN機能が有効でない場合、このオプションは意味を持ちません。

● FF_FS_RPATH

相対パス機能を指定します。

値	説明
0	相対パス機能を使わない。パス名は常にルート ディレクトリから辿る（デフォルト値）。
1	相対パス機能を使う。f_chdir、f_chdrive関数が利用可能になる。
2	1に加え、f_getcwd関数が利用可能になる。

5.3.3 ドライブ/ボリューム構成

● FF_VOLUMES

利用するボリューム（論理ドライブ）の数を 1 から 9 の範囲で設定します（本実装では 3 です）。

論理ドライブ番号 0 : RAMディスク

論理ドライブ番号 1 : SDカード

論理ドライブ番号 2 : USBメモリ

● FF_STR_VOLUME_ID

● FF_VOLUME_STRS

FF_STR_VOLUME_IDは、任意の文字列でのボリュームIDのサポートを指定します。

FF_STR_VOLUME_IDが 1 または 2 に設定されている場合、パス名のドライブ番号として任意の文字列を使用できます。FF_VOLUME_STRSは、各論理ドライブのボリュームID文字列を定義します。アイテムの数はFF_VOLUMESより少なくしてはなりません。ボリュームIDストリングの有効な文字は、A から Z、a から z、および 0 から 9 ですが、大文字と小文字を区別せずに比較されます。

FF_STR_VOLUME_ID >= 1 で、FF_VOLUME_STRSが定義されていない場合、ユーザー定義のボリューム文字列テーブルを次のように定義する必要があります。

```
const char* VolumeStr[FF_VOLUMES] = {"ram", "flash", "sd", "usb", ...
```

● FF_MULTI_PARTITION

物理ドライブ上の複数ボリュームのサポートを指定します。

無効：0（デフォルト値）では、各論理ドライブ番号は同じ物理ドライブ番号にバインドされ、物理ドライブで見つかったFATボリュームのみがマウントされます。

有効：1 の場合、各論理ドライブ番号は、VolToPart[]にリストされている任意の物理ドライブおよびパーティションにバインドできます。f_fdisk関数も利用可能になります。

● FF_MIN_SS

● FF_MAX_SS

サポートされるセクタ サイズの範囲を構成します（512、1024、2048、または 4096）。殆どのシステム、汎用メモリカード、およびハードディスクでは、常に両方の512を設定しますが、オンボードフラッシュメモリおよび一部のタイプの光学メディアでは、より大きな値が必要になる場合があります。FF_MAX_SSがFF_MIN_SSより大きい場合、FatFsは可変セクタ サイズ モード用に構成されます。

● FF_LBA64

64ビットLBAのサポートを指定します（0：無効（デフォルト値）または 1：有効）。

64ビットLBAを有効にするには、exFATも有効にする必要があります（FF_FS_EXFAT = 1）。

● FF_MIN_GPT

f_mkfsおよびf_fdisk関数でパーティショニング形式としてGPTを切り替える最小セクタ数。最大0x100000000です。FF_LBA64 = 0の場合、このオプションは意味を持ちません。

● FF_USE_TRIM

ATA-TRIMのサポートを指定します（0：無効（デフォルト値）または 1：有効）。Trim機能を有効にする場合、disk_ioctl関数にCTRL_TRIMコマンドを実装する必要があります。

6.3.4 システム構成

● FF_FS_TINY

Tinyバッファ構成を指定します（0：Normal（デフォルト値）または 1：Tiny）。Tiny構成では、ファイルオブジェクト（FIL）のサイズがFF_MAX_SSバイトに縮小されます。ファイル オブジェクトから削除されたプライベート セクタ バッファの代わりにファイル システム オブジェクト（FATFS）内の共通セクタ バッファがファイル データ転送に使用されます。

● FF_FS_EXFAT

exFATファイルシステムのサポートを指定します（0：無効（デフォルト値）または 1：有効）exFATを有効にするにはLFNも有効にする必要があります（FF_USE_LFN >= 1）。exFATを有効にするとANSI C(C89)の互換性が破棄されることに注意してください。

● FF_FS_NORTC

● FF_NORTC_MON

● FF_NORTC_MDAY

● FF_NORTC_YEAR

FF_FS_NORTCはタイムスタンプ機能を指定します。システムにRTC機能がない場合、または有効なタイムスタンプが必要ない場合は、FF_FS_NORTC = 1 を設定してタイムスタンプ機能を無効にします。この場合、FatFsによって変更されたすべてのオブジェクトには、ローカル時間でFF_NORTC_MON、FF_NORTC_MDAY、FF_NORTC_YEARによって定義された固定タイムスタンプ使用されます。

タイムスタンプ機能を有効にする（FF_FS_NORTC = 0（デフォルト値））と、get_fattime関数で取得した現在の時刻がタイムスタンプとして使用されます。この場合、FF_NORTC_MON、FF_NORTC_MDAY、FF_NORTC_YEARには意味がありません。

これらのオプションは読み取り専用構成（FF_FS_READONLY = 1）では意味がありません。

● FF_FS_NOFSINFO

FAT32ボリュームの正確な空き容量を取得する必要がある場合、設定値のビット0をセットするとボリューム マウント後の最初の時点でf_getfree関数が全FATスキャンを行って空き容量を得ます。ビット1 は最後割り当てクラスタ番号の使用を制御します。

値	説明
ビット0 = 0	FSINFOの空きクラスタ情報が有効なときはそれを利用する。
ビット0 = 1	FSINFOの空きクラスタ情報を利用しない。
ビット1 = 0	FSINFOの最終割り当てクラスタ番号が有効なときはそれを利用する。
ビット1 = 1	FSINFOの最終割り当てクラスタ番号を利用しない。

● FF_FS_LOCK

ファイル ロック機能を設定し、重複したファイル オープンおよびオブジェクト オープンに対する不正な操作を制御します。FF_FS_READONLY = 1 の場合、この設定は 0 でなければなりません。

値	説明
0	ファイルロック機能を無効にします。ボリュームの破損を避けるためには、アプリケーションプログラムは不正なオープンを避け、開いているオブジェクトを削除して名前を変更する

	必要があります（デフォルト値）。
>0	ファイルロック機能を有効にします。この設定値は、ファイル ロック制御下で同時に開くことができるファイル/サブディレクトリの数を意味します。ファイル ロック制御は再入可能性とは無関係であることに注意してください。

- **FF_FS_REENTRANT**
- **FF_FS_TIMEOUT**
- **FF_SYNC_t**

FatFsモジュール自体のリエントランシ（スレッド セーフ）の設定をします（0：無効（デフォルト値）または 1：有効）。異なるボリュームに対するファイル アクセスは、この設定に関係なく常にリエントラントです。f_mount、f_mkfs、f_fdiskなどのボリューム操作関数は、この設定に関係なく常にリエントラントではありません。同じボリュームに対するファイル アクセス（つまり、ファイル システム オブジェクトの排他使用）のみが、この設定の制御下にあります。

値	説明
0	非リエントラントです（デフォルト値）。FF_FS_TIMEOUTとFF_SYNC_tは意味を持ちません。
1	リエントラントです。セマフォの機能で排他制御を行います。
2	リエントラントです。ミューティックスの機能で排他制御を行います。

FF_FS_TIMEOUTには、μT-Kernelのタイムティック単位でタイムアウト時間を指定します。永久待ちを利用する場合は**TMO_FEVR**が利用可能です。

FF_SYNC_tは変更しないでください。

6.4 サンプルプログラム

ファイルシステムを利用するための簡易なSHELLをサンプルプログラム（usermain_fatfs.c）として提供します。サンプルプログラムが持っているファイルシステムを操作するコマンドは以下の通りです。簡易なSHELLであるため、エラー処理は行っていません。SDカードは実行の途中で2GBのものをSDカードスロットに挿入しています。

- 論理ドライブ番号：
論理ドライブの変更。
- dir
現在のディレクトリにあるファイルとサブディレクトリの一覧を表示。
- mkdir ディレクトリ名
現在のディレクトリにサブディレクトリを作成。
- rmdir ディレクトリ名
現在のディレクトリにあるサブディレクトリを削除。
- cd ディレクトリ名
サブディレクトリへの移動。cd .. なら、親ディレクトリに移動。
- wt ファイル名 サイズ

現在のディレクトリにファイルを作成。サイズはファイルに書き込む文字数（バイト数）。

● rd ファイル名

現在のディレクトリにある指定されたファイルの内容を表示。

● rm ファイル名

現在のディレクトリにある指定されたファイルを削除。

サンプルプログラムの実行結果は以下の通りです。

```
microT-Kernel Version 3.00
```

```
Input now date and time.  
Year:Month:Day:Week:Hour:Minute:Second  
Week is 0 -> Sunday ... 6 -> Saturday  
Ex. 2000:1:1:6:12:34:56
```

```
2022:11:7:1:14:13:30
```

タイムスタンプの入力

```
0:>dir  
0:>wt aaa.txt 20  
0:>dir  
AAA.TXT          20 2022:11:07 14:13:42  
0:>rd aaa.txt  
!"#$%&'()*+,-./0123  
0:>mkdir bbb  
0:>dir  
AAA.TXT          20 2022:11:07 14:13:42  
BBB      DIR      0 2022:11:07 14:14:00  
0:>cd bbb  
0:bbb>dir  
0:bbb>wt aaa.txt 30  
0:bbb>dir  
AAA.TXT          30 2022:11:07 14:14:14  
0:bbb>rd aaa.txt  
!"#$%&'()*+,-./0123456789:;<=  
0:bbb>rm aaa.txt  
0:bbb>dir  
0:bbb>cd ..  
0:>dir  
AAA.TXT          20 2022:11:07 14:13:42  
BBB      DIR      0 2022:11:07 14:14:00
```

0:> はRAMドライブ

```
0:>1:  
Please insert SD card.  
Insert SD Card.  
BlockCount = 3936256
```

SDカードにドライブを変更
SDカードの挿入

```
1:>dir  
SYSTEM~1 DIR      0 2022:04:12 07:41:34  
1:>wt aaa.txt 40  
1:>dir  
SYSTEM~1 DIR      0 2022:04:12 07:41:34  
AAA.TXT          40 2022:11:07 14:15:06  
1:>rd aaa.txt  
!"#$%&'()*+,-./0123456789:;<=?  
@ABCDEFGF  
1:>mkdir ccc  
1:>dir  
SYSTEM~1 DIR      0 2022:04:12 07:41:34  
AAA.TXT          40 2022:11:07 14:15:06
```

1:> はSDカードドライブ

```

CCC      DIR      0 2022:11:07 14:15:24
1:>cd ccc
1:ccc>dir
1:ccc>wt aaa.txt 50
1:ccc>dir
AAA.TXT      50 2022:11:07 14:15:42
1:ccc>rd aaa.txt
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMN
1:ccc>rm aaa.txt
1:ccc>dir
1:ccc>cd ..
1:>dir
SYSTEM~1  DIR      0 2022:04:12 07:41:34
AAA.TXT      40 2022:11:07 14:15:06
CCC      DIR      0 2022:11:07 14:15:24
1:>rmdir ccc
1:>dir
SYSTEM~1  DIR      0 2022:04:12 07:41:34
AAA.TXT      40 2022:11:07 14:15:06
1:>2:
Please attach USB memory.
Attach USB Memory.
BlockCount = 2001887
2:>dir
SYSTEM~1  DIR      0 2023:09:08 07:36:48
2:>wt aaa.txt 50
2:>dir
SYSTEM~1  DIR      0 2023:09:08 07:36:48
AAA.TXT      50 2000:01:01 00:00:00
2:>rd aaa.txt
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMN
2:>mkdir ddd
2:>dir
SYSTEM~1  DIR      0 2023:09:08 07:36:48
AAA.TXT      50 2000:01:01 00:00:00
DDD      DIR      0 2000:01:01 00:00:00
2:>cd ddd
2:ddd>dir
2:ddd>wt aaa.txt 60
2:ddd>dir
AAA.TXT      60 2000:01:01 00:00:00
2:ddd>rd aaa.txt
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMN
2:ddd>rm aaa.txt
2:ddd>dir
2:ddd>cd ..
2:>dir
SYSTEM~1  DIR      0 2023:09:08 07:36:48
AAA.TXT      50 2000:01:01 00:00:00
DDD      DIR      0 2000:01:01 00:00:00
2:>rmdir ddd
2:>dir
SYSTEM~1  DIR      0 2023:09:08 07:36:48
AAA.TXT      50 2000:01:01 00:00:00
2:>

```

USBメモリにドライブを変更
USBメモリの挿入

2:> はUSBメモリドライブ

7. 問い合わせ先

本実装に関する問い合わせや他のRXファミリへのポーティングに関する相談は以下のメールアドレス宛にお願い致します。

yuji_katori@yahoo.co.jp

トロンフォーラム学術・教育WGメンバ

鹿取 祐二（かとり ゆうじ）

なお、上記のメールアドレスは余儀なく変更される場合がありますが、その際はご了承ください。

以上