

TOPPERS基礎実装セミナー

(Raspberry Pi Pico(W)/Pico2(W)版)

新基礎3編:2日目

TOPPERSプロジェクト
教育ワーキング・グループ

本ドキュメントに関して

1. 著作権に関する表記

＜TOPPERS基礎実装セミナー(Raspberry Pi Pico(W)/Pico2(W)/ASP版：新基本3) 2日目＞

Copyright (C) 2011-2025 by TOPPERSプロジェクト教育ワーキング・グループ

上記著作権者は、以下の(1)～(3)の条件を満たす場合に限り、本ドキュメント（本ドキュメントを改変したものを含む。以下同じ）を使用・複製・改変・再配布（以下、利用と呼ぶ）することを無償で許諾する。

- (1) 本ドキュメントを利用する場合には、上記の著作権表示、この利用条件および以下の無保証規定が、そのままの形でドキュメント中に含まれていること。
- (2) 本ドキュメントを改変する場合には、ドキュメントを改変した旨の記述を、改変後のドキュメント中に含めること。
ただし、改変後のドキュメントがTOPPERSプロジェクト指定の開発成果物である場合には、この限りではない。
- (3) 本ドキュメントの利用により直接的または間接的に生じるいかなる損害からも、上記著作権者およびTOPPERSプロジェクトを免責すること。また、本ドキュメントのユーザまたはエンドユーザからのいかなる理由に基づく請求からも、上記著作権者およびTOPPERSプロジェクトを免責すること。

本ドキュメントは、無保証で提供されているものである。上記著作権者およびTOPPERSプロジェクトは、本ドキュメントに関して、特定の使用目的に対する適合性も含めて、いかなる保障もしない。また、本ドキュメントの利用により直接的または間接的に生じたいかなる損害に関しても、その責任を負わない。

2. 本ドキュメントに関するご意見・ご提言・ご感想・ご質問等がありましたら、TOPPERSプロジェクト事務局までE-Mailにてご連絡ください。
3. 本ドキュメントの内容は、内容の改善や適正化の目的で予告無く改定することがあります。

本ドキュメントでは、Microsoft社のClip Art Galleryコンテンツを使用しています。

TRONは"The Real-time Operating system Nucleus"の略称です。ITRONは"Industrial TRON"の略称です。
μITRONは"Micro Industrial TRON"の略称です。TOPPERS/JSPはToyohashi Open Platform for Embedded Real-Time System/Just Standard Profile Kernelの略称です。」

本ドキュメント中の商品名及び商標名は、各社の商標または登録商標です。

スケジュール

■ 2日目

1. SDカードSPIインターフェイス	2.0時間
2. SPI-SDカードFATドライバ対応	1.0時間
3. SDカードファイルシステムの構築	1.5時間
4. ファイルテストプログラム	0.5時間
5. DICアーキテクチャ	0.2時間
6. LCDシールドを使ったアプリ紹介	0.5時間
7. まとめ	0.2時間

概要

- SDカード: SPI通信の方式の理解
- SPI-SDカード通信を使って、FATファイルシステムを動かす
- ハードウェアを隠蔽化するミドルウェア(ストレージデバイスマネージャ)についての学習と実習
- フリーソフトのFATファイルシステム: FatFsの実装とテストプログラムによる実行確認
- 標準的なC言語ファイル関数を提供するファイルライブラリについての学習
- DICアーキテクチャの学習

SDカードSPIインターフェース仕様

1. [SDカードSPI-I/F](#)
2. 初期化手順 READ/WRITE
3. 初期化プログラム
4. デバイスストレージマネージャー

SDカードのSPIインターフェイス

- 一般的なSDカードは、MMC通信とSPI通信をサポートしている
 - SPI通信をサポートしていないものがあるので注意
- MMC通信からSPI通信への移行は、電源ON後電源電圧が定常化した後1ms待ち、MOSI,CSをHIGHレベルに設定しSCKを74クロック以上送る

No	MMC	SPI
8	DAT1	–
7	DAT0	MOSI
6	Vss2	
5	CLK	SCLK
4	Vdd	
11	NC	
10	NC	
3	Vss1	
2	CMD	MISO
1	DAT3	SS
9	DAT2	–

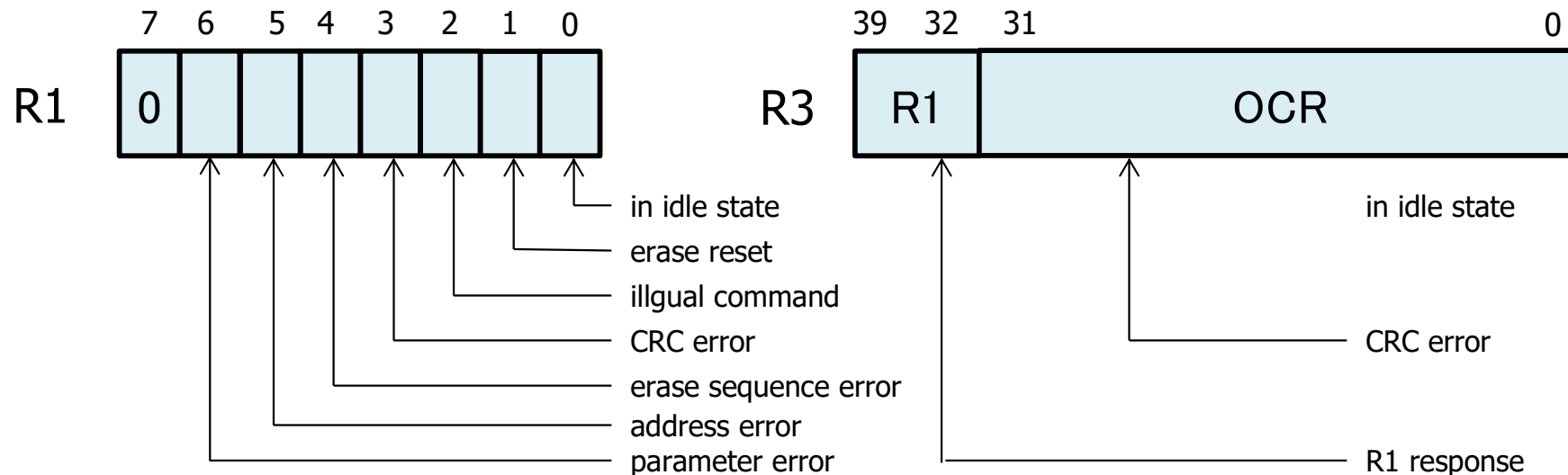
SDカード SPI通信

- SDカードのSPI通信では、SPI信号上のコマンド、レスポンスにより通信を行う(MMCのコマンドとは異なる)

コマンド:48ビット(6バイト)通信



レスポンス:8ビット(1バイト)通信 R1とR1を含むレスポンスがある



SDカード SPI通信 コマンド

*1 supply Voltage/Check Pattern

インデックス	引数		転送	機能
CMD0	なし	R1	なし	ソフトウェアリセット
CMD1	なし	R1	なし	初期化開始
ACMD41	HCS	R7	なし	SDC専用:初期化開始
CMD8	*1	R1	なし	SDC V2専用:動作電圧確認
CMD9	なし	R1	あり	CSD読み出し
CMD10	なし	R1b	あり	CID読み出し
CMD12	なし	R1	なし	リード動作停止
CMD16	ブロック長	R1	なし	ブロック長設定
CMD17	アドレス	R1	あり	シングルブロック読み出し
CMD18	アドレス	R1	あり	マルチブロック読み出し
CMD23	ブロック数	R1	なし	MMC専用:次のマルチブロック読み書きブロック数設定
ACMD23	ブロック数	R1	なし	SDC専用:次のマルチブロック書きこみまでのブロック設定
CMD24	アドレス	R1	あり	シングルブロック書き込み
CMD25	アドレス	R1	あり	マルチブロック書き込み
CMD55	なし	R1	なし	アプリケーション特化コマンド
CMD58	なし	R3	なし	OCR読み出し

SDカード SPI通信 コマンド例

- コマンドの例として、CMD0,8とCRC7の計算式を示す

47	46	45	...	40	39	...	8	7	...	1	0
0	1	コマンド番号				引数				CRC	1

CMD0 0x40 0x00 0x00 0x00 0x00 0xXX

CMD8 0x48 0x00 0x00 0x01 0xAA 0xXX

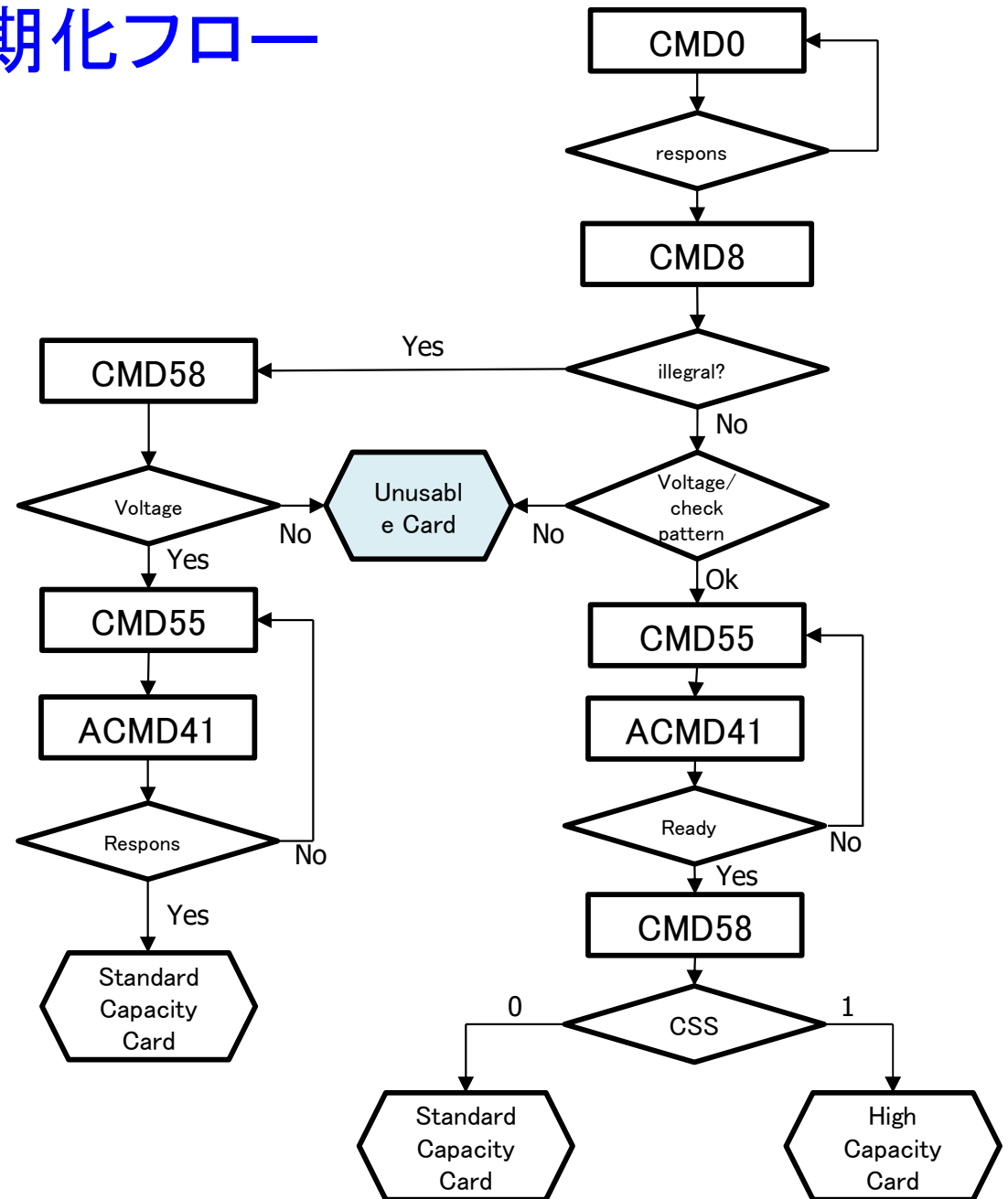
```
static uint8_t calc_CRC7(const uint8_t *data, int len)
{
    uint_t i, j; uint8_t crc = 0, dt;
    for(i = 0; i < len; i++){
        dt = *data++;
        for(j = 0; j < 8; j++){
            crc <<= 1;
            if ((crc & 0x80) ^ (dt & 0x80))
                crc ^= 0x09;
            dt <<= 1;
        }
    }
    return ((crc & 0x7f) << 1) | 1;
}
```

SDカードSPIインターフェイス仕様

1. SDカードSPI-I/F
2. 初期化手順 READ/WRITE
3. 初期化プログラム
4. デバイスストレージマネージャー

SDカード SPI通信 初期化フロー

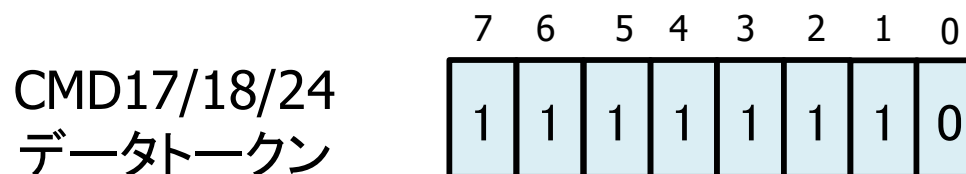
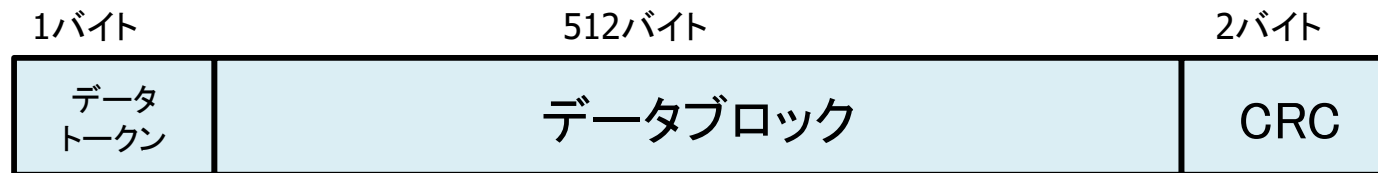
- SPIモード移行後のSPI通信を使った初期化フローは右図の通りである
- カードの種別確定後CID,CSDの取出しカード情報の取出しを行う(MMCと同等)
- その後、CMD17やCMD24を使ってブロックREAD/WRITEを行うことができる



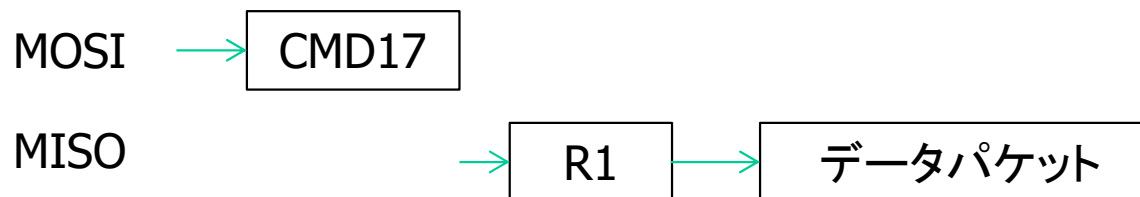
SDカード SPI通信 シングルブロックREAD

- シングルブロックREADは、CMD17発行後R1受信後データパケットを受信
- CRCの確認、エラーの場合のリトライ処理は必須

データパケット



通信手順



SDカード SPI通信 CRC-16 CCITT計算

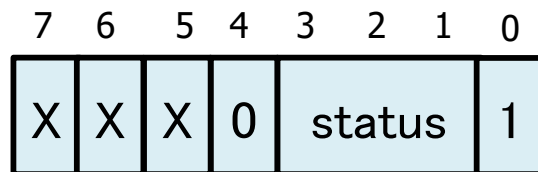
- データパケット用のCRC計算としてCRC-16 CCITTを用いる

```
static uint16_t  
calc_CRC16(const uint8_t *data, int len)  
{  
    uint16_t crc = 0;    int i, j;  
    for(i = 0; i < len; i++){  
        crc ^= (data[i]<<8);  
        for(j = 0; j < 8; j++){  
            if(crc & 0x8000)  
                crc = (crc << 1) ^ CRC16POLY;  
            else  
                crc <<= 1;  
        }  
    }  
    return crc;  
}
```

SDカード SPI通信 シングルブロックWRITE

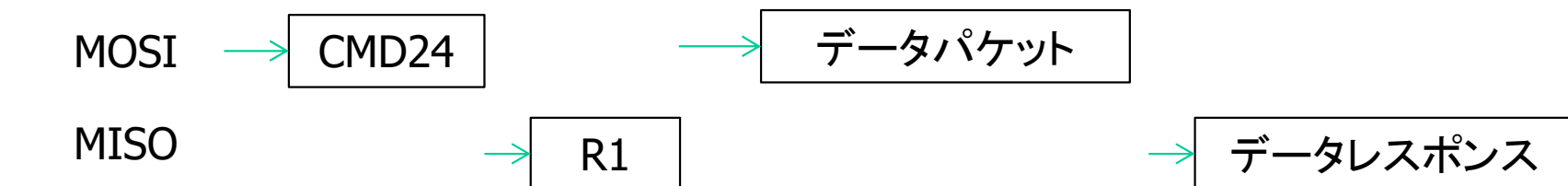
- シングルブロックWRITEはCMD24発行後R1を確認してデータパケットを送信する
- 送信後データレスポンスを受信し、エラーならばリセット、再送する

データレスポンス



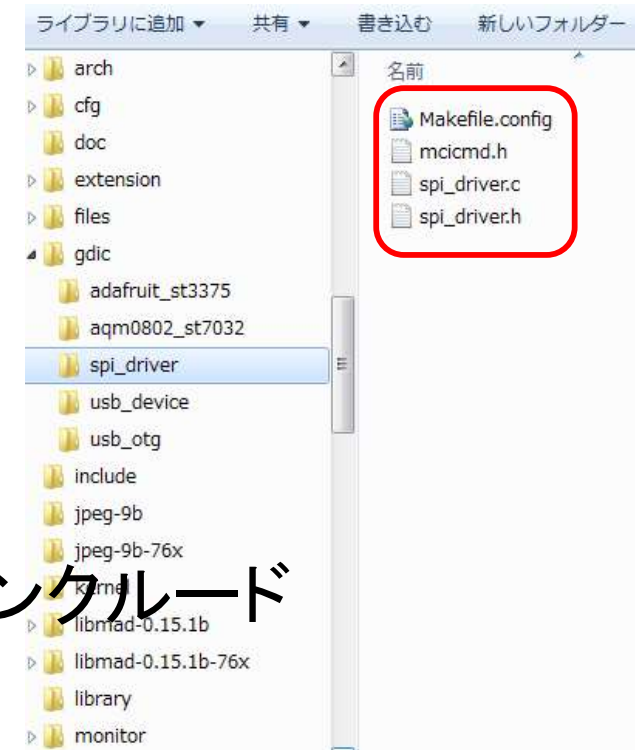
0	1	0	— Data accepted
1	0	1	— Data rejected due to a CRC error
1	1	0	— Data rejected due a write error

通信手順



SDカード SPI通信プログラム

- ここまで説明したSDカード/SPI通信プログラムは
TOPPERS BASE PLATFORM(ST):gdic/spi_driver中にドライバとして用意しているのでドライバ関数を順番に実行するだけで、アクセス可能となる
- 構成ファイル中にSDカード/SPI通信に必要なドライバ、RTOSリソースをすべて持つ
- 構成ファイル
 - Makefile.config : Makeファイルインクルード
 - spi_driver.h : ヘッダファイル
 - spi_driver.c : ソースファイル
 - mcicmd.h : ファイルシステム連結用インクルード



SPI_DRIVER(spi_driver.h)

- SPI用のSDカード情報を管理するハンドラ(SDカードハンドラ)
- 下位ドライバとしてSPIを使用するため、SPIハンドラを持つ

```
typedef struct {  
    SPI_Handle_t    *hsapi;        /* spiハンドラ */  
    uint32_t        RetryCount; /* リトライ回数 */  
    uint32_t        cardtype;     /* カードタイプ */  
    uint32_t        RCA;          /* RCA値 */  
    uint8_t         CSD[16];      /* CSD値 */  
    uint8_t         CID[16];      /* CID値 */  
    ID              spi_lock;  
    volatile uint32_t status;      /* 転送状態 */  
    uint32_t        read_count;  
    uint32_t        write_count;  
    uint32_t        read_retry_count;  
    uint32_t        write_retry_count;  
    uint32_t        read_error;  
    uint32_t        write_error;  
}SDCARD_Handler_t;
```

SPI_DRIVER(spi_driver.h)

- SDカードの種別を設定する構造体
- 初期化後、sdcard_getcardinfo関数で取り出せる

```
#define SD_CARD_V11    0
#define SD_CARD_V20    1
#define SD_CARD_HC     2
#define MMC_CARD       3
#define SD_IO_CARD     4

typedef struct {
    uint64_t    capacity; /* 容量(バイト) */
    uint32_t    blocksize; /* ブロックサイズ */
    uint32_t    maxsector; /* 最大セクタ数 */
    uint16_t    RCA; /* SD RCA */
    uint8_t     cardtype; /* カード種類 */
    uint8_t     status; /* ステータス */
}SDCARD_CardInfo_t;
```

カード種別定義

SPI_DRIVER(spi_driver.c)

- SPIDRIVERは以下の機能(関数)を持つ

SPI_Handler_t *	spi_start	GPIOとSPIの初期化
void	spi_end	SPI処理終了
ER	sdcard_lock	SPIロック:SDカード用SSをLOW
ER	sdard_unlock	SPIアンロック:SDカード用SSをHIGH
void	sdcard_setspi	SDカードハンドラにSPIハンドラをセット
bool_t	sdcard_init	SDカード SPI初期化を行う
ER	sdcard_open	SDカードハンドラを初期化して取り出す
ER	sdcard_close	SDカードハンドラを終了する
ER	sdcard_getcardinfo	SDカード種別構造体を取り出す
ER	sdcard_checkCID	CIDの取得
ER	sdcard_sendCSD	CSDを取得する
ER	sdcard_configuration	コンフィギュレーションを行う
ER	sdcard_blockread	シングルブロックREADする(リトライ機能付き)
ER	sdcard_blockwrite	シングルブロックWRITEする(リトライ機能付き)
ER	sdcard_wait_transfar	ブロックデータ転送待ち

SDカードSPIインターフェース仕様

1. SDカードSPI-I/F
2. 初期化手順 READ/WRITE
3. 初期化プログラム
4. デバイスストレージマネージャー

基礎3実習：プログラムファイル

- プログラムファイルの置き場所(二日目)
 - 教材ディレクトリ¥base3¥program

SDカード SPI通信

sd_ini1 : SDカードブロック初期化プログラム

ファイルシステム

sd_file1 : FatFs対応

アプリケーション

CupRamenTimer : カップラーメンタイマアプリ

sd_init解説

- sd_initはSDカードに対する初期化プログラム
 - SDカードをSPIモードで初期化する
 - SDカードからタイプや容量などの情報を取得する
 - SDカードの0セクタをREADしDUMPする
- sd_init仕様
 - arduinoコネクタSPI(SCLK,MOSI,MISO)を用いて制御
 - SSはCS2(D4)を使用
 - 他の信号は取得できないため、挿抜制御なし
 - SPIの上位のドライバはSPIDRIVERを用いる

sd_ini1ワーク領域の構成

- sd_ini1のプログラムはprogram/sd_ini1内にある
- ドライバはTOPPERS BASE PLATFORMを使用

ファイル名	ファイルの内容	設定場所
sd_access.h	sd_ini1プログラムのインクルードファイル	/base3/program/sd_ini1
sd_access.cfg	sd_ini1プログラムコンフィギュレーションファイル	/base3/program/sd_ini1
sd_access.c	sd_ini1プログラムのソースファイル	/base3/program/sd_ini1
Makefile	sd_ini1プログラムのメイクファイル	/base3/program/sd_ini1
command.c	DEVICEコマンド拡張ソースファイル	/base3/program/sd_ini1
Makefile.config	Makefileからインクルードされる環境設定ファイル	/base3/program
spi.h	SPIドライバのインクルードファイル	/asp/pdic/stm32f4xx
spi.c	SPIドライバのソースファイル	/asp/pdic/stm32f4xx
pinmode.h	ArduinoコネクタのGPIO設定インクルードファイル	/asp/pdic/stm32f4xx
pinmode.c	ArduinoコネクタのGPIO設定ソースファイル	/asp/pdic/stm32f4xx

sd_ini1 インクルードファイル

- SPIドライバの割込み設定
 - SPIデバイスドライバ割込み設定
 - SPIで使用するDMAの割込み設定

```
#define INHNO_SPI    IRQ_VECTOR_SPI0 /* 割込みハンドラ番号 */
#define INTNO_SPI    IRQ_VECTOR_SPI0 /* 割込み番号 */
#define INTPRI_SPI   -3              /* 割込み優先度 */
#define INTATR_SPI   TA_EDGE         /* 割込み属性 */

#define DMA_INTNO    DMA_INT0
#define INHNO_DMASPI (IRQ_VECTOR_DMA0+DMA_INTNO) /* 割込みハンドラ番号 */
#define INTNO_DMASPI (IRQ_VECTOR_DMA0+DMA_INTNO) /* 割込み番号 */
#define INTPRI_DMASPI -3             /* 割込み優先度 */
#define INTATR_DMASPI TA_EDGE        /* 割込み属性 */
```

sd_init インクルードファイル

- SPI_DEVICEに渡すポートIDとGPIOの定義
- ストリームデバイスについては後述

```
#define SDCARD_DEVNO 0  
#define SDCARD_PORTID 0
```

ストリームデバイスの番号
SPIDEVICEのポートID定義

```
#if LOGSHIELD == 0  
#define CS_PORT 10  
#define CS2_PORT 4  
#else  
#define CS_PORT 4  
#define CS2_PORT 10  
#endif  
#define RST_PORT 9  
#define RS_PORT 8  
#define UP_PORT 7  
#define RIGHT_PORT 6  
#define LEFT_PORT 5  
#define DOWN_PORT 3  
#define cs_set(sw) digitalWrite(CS_PORT, sw)  
#define rs_set(sw) digitalWrite(RS_PORT, sw)  
#define rst_set(sw) digitalWrite(RST_PORT, sw)  
#define cs2_set(sw) digitalWrite(CS2_PORT, sw)
```

ArduinoコネクタのGPIO設定

sd_initコンフィギュレーションファイル

```
INCLUDE("target_timer.cfg");
INCLUDE("syssvc/syslog.cfg");
INCLUDE("syssvc/banner.cfg");
INCLUDE("syssvc/serial.cfg");
INCLUDE("syssvc/logtask.cfg");
#if BOARDNO == 0
INCLUDE("pdic/rp2040/device.cfg");
#else
INCLUDE("pdic/rp2350/device.cfg");
INCLUDE("gdic/daytime/dtime.cfg");
#endif
INCLUDE("monitor/monitor.cfg");
```

```
#include "device.h"
#include "spi.h"
#include "sd_access.h"
```

```
ATT_INI({ TA_NULL, sw_int, setup_sw_func });
ATT_INI({ TA_NULL, 0, device_info_init });
CRE_TSK(MAIN_TASK, { TA_ACT, 0, main_task, MAIN_PRIORITY, STACK_SIZE, NULL });
CRE_SEM(SPI1DMA_SEM, { TA_TPRI, 0, 1 });
CRE_SEM(SPI1LOCK_SEM, { TA_TPRI, 1, 1 });
```

```
ATT_ISR({TA_NULL, SPI1_PORTID, INTNO_SPI, spi_isr, 1 });
CFG_INT(INTNO_SPI, { TA_ENAINT | INTATR_SPI, INTPRI_SPI });
DEF_INH(INHNO_DMASPI, {TA_NULL, dma_int });
CFG_INT(INTNO_DMASPI, { TA_ENAINT | INTATR_DMASPI, INTPRI_DMASPI });
```

Picoの場合、BOARDNO=0,
Pico2の場合、BOARDNOは1ま
たは2
両方対応できるように、コンパ
イル・スイッチで対応している

sd_initのメイクファイル

- システムサービスのソースファイルに必要なドライバを追加
 - spi_driver(GDIC)を追加
 - spi.o :SPI下位(PDIC)ドライバ
 - pinmode.o :arduino-gpio設定ドライバ

```
#
# GDIC Makefile のインクルード
#
include $(SRCDIR)/gdic/spi_driver/Makefile.config
:
SYSSVC_DIR := $(SYSSVC_DIR) $(SRCDIR)/syssvc $(SRCDIR)/library ¥
              $(SRCDIR)/pdic/$(PDIC_DIR)
SYSSVC_ASMOBS := $(SYSSVC_ASMOBS)
SYSSVC_COBJS := $(SYSSVC_COBJS) banner.o syslog.o serial.o logtask.o ¥
              device.o pio_spi.o spi.o pinmode.o $(CXXRTS)
SYSSVC_CFLAGS := $(SYSSVC_CFLAGS)
SYSSVC_LIBS := $(SYSSVC_LIBS)
INCLUDES := $(INCLUDES) -I$(SRCDIR)/pdic/$(PDIC_DIR)
```

sd_initソースファイル

- メインタスク起動後、SPIの初期化とArduinoコネクタのGPIO設定を行う

```
/*  
 * CS PORT(D10-5)  
 */  
pinMode(CS_PORT, GPIO_MODE_OUTPUT);  
/*  
 * RS PORT(D8-7)  
 */  
pinMode(RS_PORT, GPIO_MODE_OUTPUT);  
/*  
 * RST PORT(D9-6)  
 */  
pinMode(RST_PORT, GPIO_MODE_OUTPUT);  
/*  
 * CS2 PORT(D4-11)  
 */  
pinMode(CS2_PORT, GPIO_MODE_OUTPUT);  
  
spi_initd.Baudrate      = 1000*1000;  
spi_initd.Direction     = SPI_DIRECTION_2LINES;  
spi_initd.CLKPhase      = SPI_PHASE_1EDGE;  
spi_initd.CLKPolarity   = SPI_POLARITY_LOW;  
spi_initd.DataSize      = SPI_DATASIZE_8BIT;  
spi_initd.FrameFormat   = SPI_MOTOROLA_MODE;
```

SDカードのCS設定

SPIの初期化
クロックは低速で

sd_initソースファイル

- MOSIに0xFFを10バイト送って、SPIカードにSPI通信要求

```
spi_initd.DMARxChannel = 1;  
spi_initd.DMATxChannel = 0;  
spi_initd.Mode          = SPI_MODE_MASTER;  
spi_initd.semId          = SPI1DMA_SEM;  
spi_initd.semLock        = 0;
```

SPIのクロックを低速にして
初期化

```
if((hspi = spi_init(SPI1_PORTID, &spi_initd)) == NULL){  
    syslog_0(LOG_ERROR, "spi_start initialize error !");  
    slp_tsk();  
}
```

```
for(i = 0 ; i < 10 ; i++){  
    aTxBuffer[i] = 0xFF;  
    cs_set(PORT_HIGH);  
    cs2_set(PORT_HIGH);
```

LCDとSPIのCSをHIGHに

```
if(spi_transmit(hspi, (uint8_t*)aTxBuffer, 10) != E_OK){  
    /* Transfer error in transmission process */  
    syslog_0(LOG_NOTICE, "## call Error_Handler(1) ##");  
}
```

SPIに0xFFを10バイト送っ
てSDカードにSPIモード要
求

```
#if SPI_WAIT_TIME == 0  
    spi_wait(hspi, 50);  
#endif  
dly_tsk(100);
```

sd_initソースファイル

- sdcard_init関数でSDカード:SPI通信の初期化を行う
- 初期化が成功すれば、sdcard_open関数でSDカードハンドラを取得
- CID/CSDとカード情報の取得、SDカードのコンフィギュレーションを行う

```
spi_deinit(hspi);
spi_initd.Baudrate = 4*1000*1000;
hspi = spi_init(SPI1_PORTID, &spi_initd);
sdcard_setspi2(SDCARD_PORTID, hspi, CS2_PORT);
if(sdcard_init(SDCARD_PORTID)){
    hsd = sdcard_open(SDCARD_PORTID);
    SVC_ERROR(sdcard_checkCID(hsd));
    SVC_ERROR(sdcard_sendCSD(hsd));
    SVC_ERROR(sdcard_getcardinfo(hsd, &SDCardInfo));
    SVC_ERROR(sdcard_configuration(hsd));
}
else{
    syslog_0(LOG_ERROR, "SD-CARD INITIAL ERROR !");
    slp_tsk();
}
```

SDカード初期化フローに従った初期化

カード情報の取得、コンフィギュレーションを行う

sd_initソースファイル

- SPIクロックを高速化
- 初期化が成功した場合、0セクタをREAD後DUMP

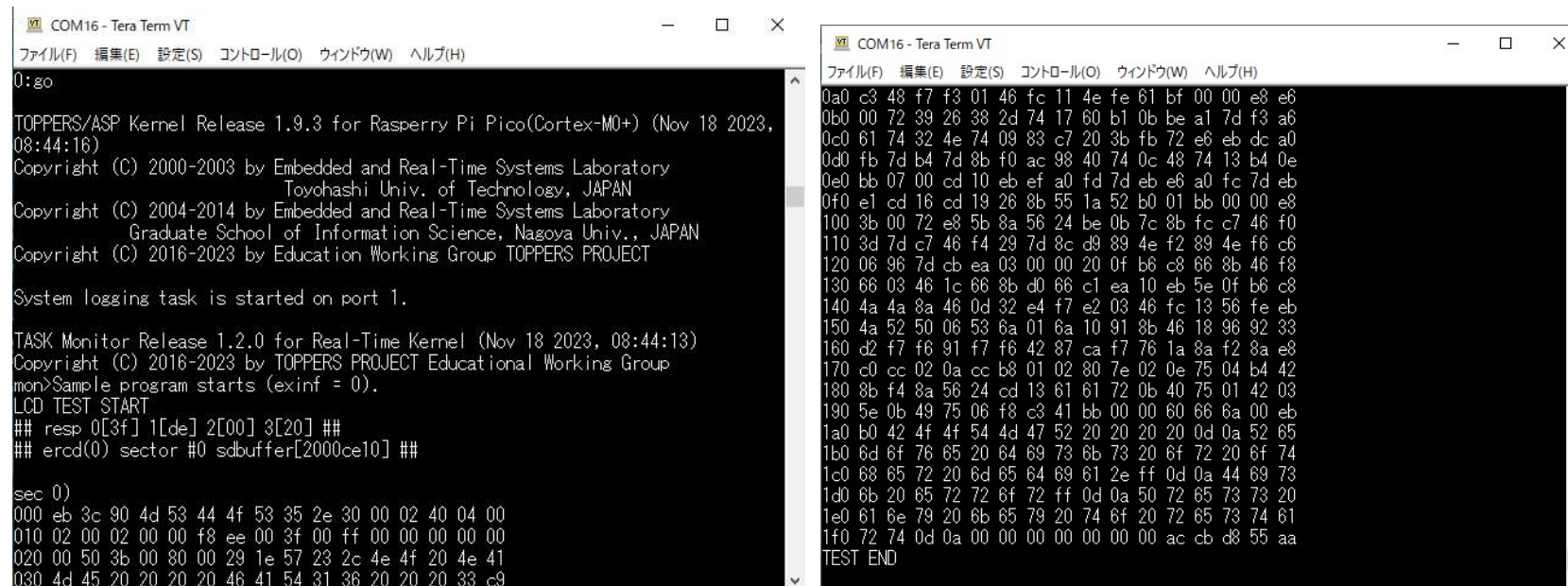
```
if(hsd != NULL)
    hsd->spi_lock = SPI1LOCK_SEM;
if(hsd != NULL && hsd->hspi != NULL){
    unsigned char *p;
    int j;
    ercd = sdcard_blockread(hsd, sdbuffer, 0*512, 512, 1);
    syslog_2(LOG_NOTICE, "## ercd(%d) sector #0 sdbuffer[%08x] ##", ercd, sdbuffer);
    dly_tsk(300);
    p = (unsigned char *)sdbuffer;
    printf("¥nsec 0 ");
    for(j = 0 ; j < 512 ; j++){
        if((j % 16) == 0)
            printf("¥n%03x ", j);
        printf("%02x ", p[j]);
    }
    printf("¥n");
}
syslog_0(LOG_NOTICE, "TEST END");
slp_tsk();
spi_deinit(hspi);
```

演習:sd_ini1のビルド実行

- SDカード SPI通信の初期化確認
- 学習内容
 - SDカード SPI通信の初期化が正しく行われるかの確認
- 手順
 - my_programにprogram/sd_ini1をコピーする
 - program/sd_ini1上でビルド、ダウンロード、実行
 - SDカードの初期化エラーなく終了することの確認
 - 0セクタのDUMPが正しく行われるかの確認

演習:sd_initをビルド、実行

- ビルド、ダウンロード、実行するとSDカードの0セクタをREADしDUMPする



```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
0:go

TOPPERS/ASP Kernel Release 1.9.3 for Raspberry Pi Pico(Cortex-M0+) (Nov 18 2023,
08:44:16)
Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory
Toyohashi Univ. of Technology, JAPAN
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory
Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT

System logging task is started on port 1.

TASK Monitor Release 1.2.0 for Real-Time Kernel (Nov 18 2023, 08:44:13)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>Sample program starts (exinf = 0).
LCD TEST START
## resp 0[3f] 1[de] 2[00] 3[20] ##
## ercd(0) sector #0 sdbuffer[2000ce10] ##

sec 0)
000 eb 3c 90 4d 53 44 4f 53 35 2e 30 00 02 40 04 00
010 02 00 02 00 00 f8 ee 00 3f 00 ff 00 00 00 00 00
020 00 50 3b 00 80 00 29 1e 57 23 2c 4e 4f 20 4e 41
030 4d 45 20 20 20 20 46 41 54 31 36 20 20 20 33 c9

COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
0a0 c3 48 f7 f3 01 46 fc 11 4e fe 61 bf 00 00 e8 e6
0b0 00 72 39 26 38 2d 74 17 60 b1 0b be a1 7d f3 a6
0c0 61 74 32 4e 74 09 83 c7 20 3b fb 72 e6 eb dc a0
0d0 fb 7d b4 7d 8b f0 ac 98 40 74 0c 48 74 13 b4 0e
0e0 bb 07 00 cd 10 eb ef a0 fd 7d eb e6 a0 fc 7d eb
0f0 e1 cd 16 cd 19 26 8b 55 1a 52 b0 01 bb 00 00 e8
100 3b 00 72 e8 5b 8a 56 24 be 0b 7c 8b fc c7 46 f0
110 3d 7d c7 46 f4 29 7d 8c d9 89 4e f2 89 4e f6 c6
120 06 96 7d cb ea 03 00 00 20 0f b6 c8 66 8b 46 f8
130 66 03 46 1c 66 8b d0 66 c1 ea 10 eb 5e 0f b6 c8
140 4a 4a 8a 46 0d 32 e4 f7 e2 03 46 fc 13 56 fe eb
150 4a 52 50 06 53 6a 01 6a 10 91 8b 46 18 96 92 33
160 d2 f7 f6 91 f7 f6 42 87 ca f7 76 1a 8a f2 8a e8
170 c0 cc 02 0a cc b8 01 02 80 7e 02 0e 75 04 b4 42
180 8b f4 8a 56 24 cd 13 61 61 72 0b 40 75 01 42 03
190 5e 0b 49 75 06 f8 c3 41 bb 00 00 60 66 6a 00 eb
1a0 b0 42 4f 4f 54 4d 47 52 20 20 20 0d 0a 52 65
1b0 6d 6f 76 65 20 64 69 73 6b 73 20 6f 72 20 6f 74
1c0 68 65 72 20 6d 65 64 69 61 2e ff 0d 0a 44 69 73
1d0 6b 20 65 72 72 6f 72 ff 0d 0a 50 72 65 73 73 20
1e0 61 6e 79 20 6b 65 79 20 74 6f 20 72 65 73 74 61
1f0 72 74 0d 0a 00 00 00 00 00 00 00 00 ac cb d8 55 aa
TEST END
```

演習:sd_ini1を改造(sd_ini2)

- SDカードのタイプ、容量を表示するプログラムを作成
- 学習内容
 - 初期化後のSDカード情報の確認
- 手順
 - sd_ini1をコピーしてsd_ini2を作り、改造する
 - SPIDRIVERのsdcard_getcardinfo関数を使用
 - 取り出したSDCARD_CardInfo_t構造体の内容をsyslogを用いて表示する

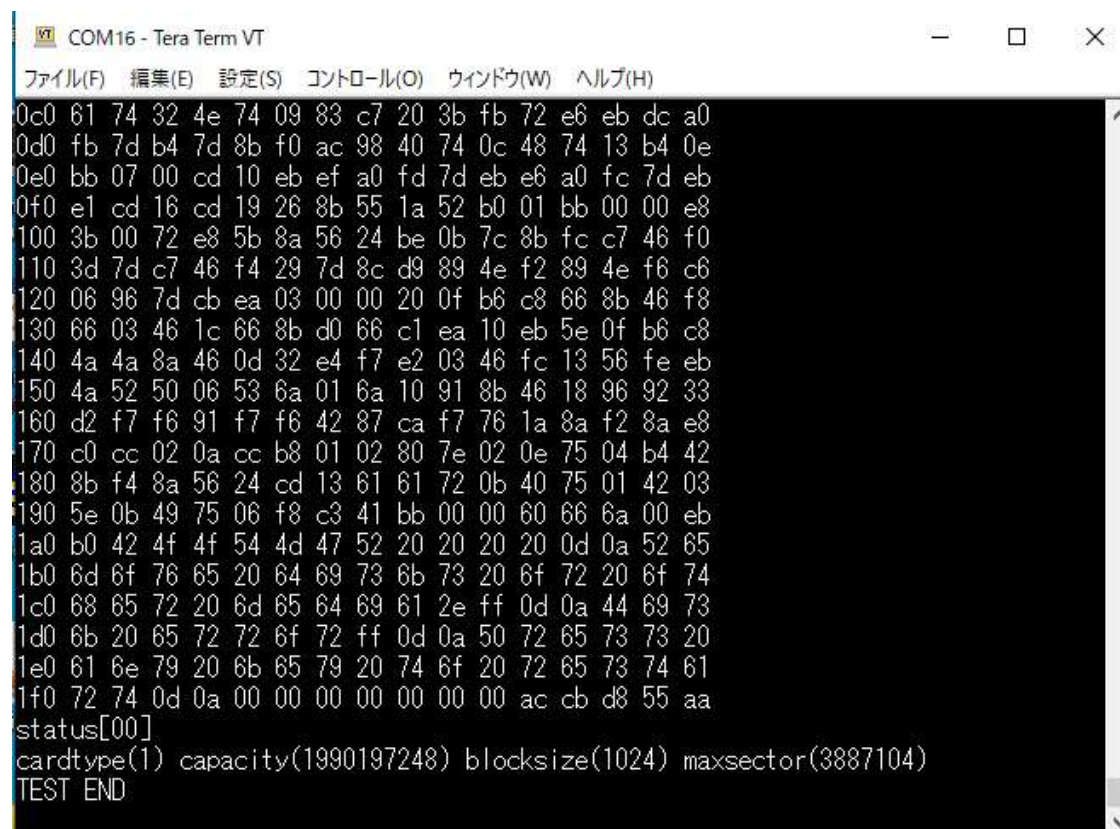
sd_ini2:sd_access.cを改造する

- 0セクタREAD後、sdcard_getcardinfo関数を用いてSDCardInfoにカード情報を取り込み、syslog関数で内容を表示
- capacityはuint64_t型なのでsyslog文でサポートしていない、int型に変換して表示する

```
if(hsd != NULL && hsd->hspi != NULL){  
    unsigned char *p;  
    int j, wk1, wk2;  
    ercd = sdcard_blockread(hsd, sdbuffer, 0*512, 512, 1);  
    :  
    printf("¥n");  
    SVC_ERROR(sdcard_getcardinfo(hsd, &SDCardInfo));  
    wk1 = SDCardInfo.capacity % 1000000000;  
    wk2 = SDCardInfo.capacity / 1000000000;  
    syslog_1(LOG_NOTICE, "status[%02x]", SDCardInfo.status);  
    syslog_5(LOG_NOTICE, "cardtype(%d) capacity(%d%09d) blocksize(%d) maxsector(%u)",  
        SDCardInfo.cardtype, wk2, wk1, SDCardInfo.blocksize, SDCardInfo.maxsector);  
}
```

SDカード情報を表示

- 2GBのSDカードの場合、type=1 (SD_CARD_V20:SDカードVersion2.0)となる
- maxsectorはブロックサイズ512バイトとした場合のセクタ数であり、表示のブロックサイズは物理的なブロックのサイズである



The screenshot shows a Tera Term VT window titled 'COM16 - Tera Term VT'. The menu bar includes 'ファイル(F)', '編集(E)', '設定(S)', 'コントロール(O)', 'ウィンドウ(W)', and 'ヘルプ(H)'. The main display area shows a hex dump of data, with each line representing a row of 16 bytes. Below the hex dump, the following text is displayed: 'status[00]', 'cardtype(1) capacity(1990197248) blocksize(1024) maxsector(3887104)', and 'TEST END'.

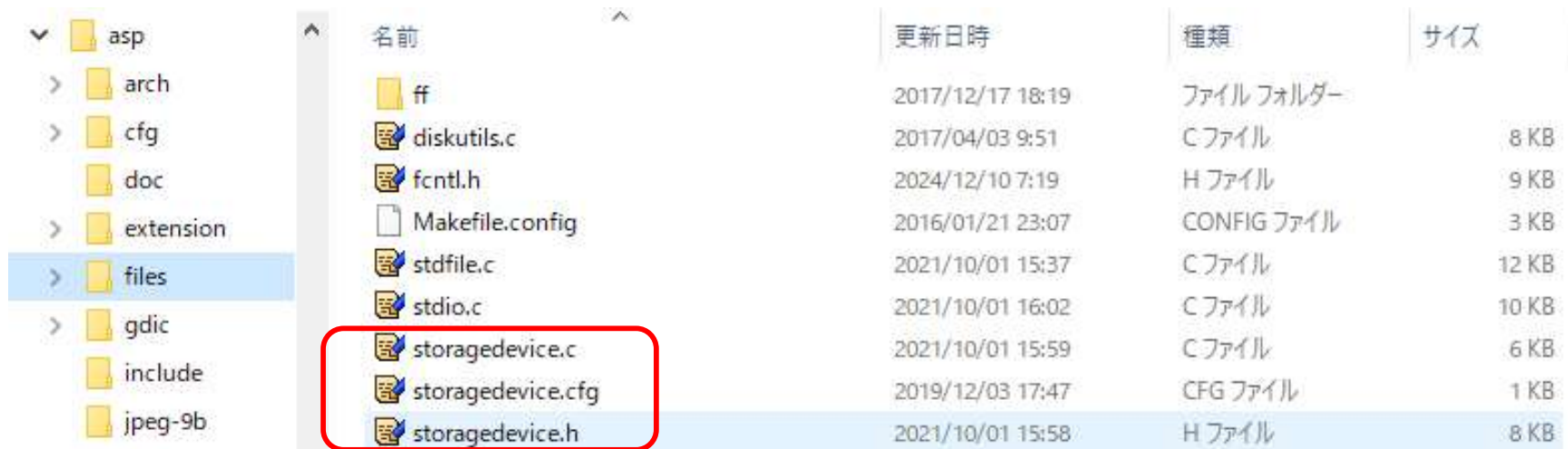
```
0c0 61 74 32 4e 74 09 83 c7 20 3b fb 72 e6 eb dc a0
0d0 fb 7d b4 7d 8b f0 ac 98 40 74 0c 48 74 13 b4 0e
0e0 bb 07 00 cd 10 eb ef a0 fd 7d eb e6 a0 fc 7d eb
0f0 e1 cd 16 cd 19 26 8b 55 1a 52 b0 01 bb 00 00 e8
100 3b 00 72 e8 5b 8a 56 24 be 0b 7c 8b fc c7 46 f0
110 3d 7d c7 46 f4 29 7d 8c d9 89 4e f2 89 4e f6 c6
120 06 96 7d cb ea 03 00 00 20 0f b6 c8 66 8b 46 f8
130 66 03 46 1c 66 8b d0 66 c1 ea 10 eb 5e 0f b6 c8
140 4a 4a 8a 46 0d 32 e4 f7 e2 03 46 fc 13 56 fe eb
150 4a 52 50 06 53 6a 01 6a 10 91 8b 46 18 96 92 33
160 d2 f7 f6 91 f7 f6 42 87 ca f7 76 1a 8a f2 8a e8
170 c0 cc 02 0a cc b8 01 02 80 7e 02 0e 75 04 b4 42
180 8b f4 8a 56 24 cd 13 61 61 72 0b 40 75 01 42 03
190 5e 0b 49 75 06 f8 c3 41 bb 00 00 60 66 6a 00 eb
1a0 b0 42 4f 4f 54 4d 47 52 20 20 20 20 0d 0a 52 65
1b0 6d 6f 76 65 20 64 69 73 6b 73 20 6f 72 20 6f 74
1c0 68 65 72 20 6d 65 64 69 61 2e ff 0d 0a 44 69 73
1d0 6b 20 65 72 72 6f 72 ff 0d 0a 50 72 65 73 73 20
1e0 61 6e 79 20 6b 65 79 20 74 6f 20 72 65 73 74 61
1f0 72 74 0d 0a 00 00 00 00 00 00 00 00 ac cb d8 55 aa
status[00]
cardtype(1) capacity(1990197248) blocksize(1024) maxsector(3887104)
TEST END
```

SDカードSPIインターフェイス仕様

1. SDカードSPI-I/F
2. 初期化手順 READ/WRITE
3. 初期化プログラム
4. デバイスストレージマネージャー

ストレージデバイスマネージャ

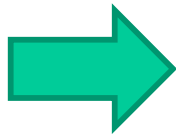
- TOPPERS BASE PLATFORMは複数のストレージデバイスに対応している。これらを管理するためにストレージデバイスマネージャという機能を持つ
 - SDカード:MMC
 - SDカード:SPI
 - USBメモリ:MSC
- ストレージデバイスマネージャはasp/filesにファイルシステム(FatFs/ファイルライブラリ)とともに実装されている



名前	更新日時	種類	サイズ
ff	2017/12/17 18:19	ファイル フォルダ	
diskutils.c	2017/04/03 9:51	C ファイル	8 KB
fcntl.h	2024/12/10 7:19	H ファイル	9 KB
Makefile.config	2016/01/21 23:07	CONFIG ファイル	3 KB
stdfile.c	2021/10/01 15:37	C ファイル	12 KB
stdio.c	2021/10/01 16:02	C ファイル	10 KB
storagedevice.c	2021/10/01 15:59	C ファイル	6 KB
storagedevice.cfg	2019/12/03 17:47	CFG ファイル	1 KB
storagedevice.h	2021/10/01 15:58	H ファイル	8 KB

ストリームデバイスの機能

- デバイスの管理機能
 - メディアの挿抜管理(注1)
 - 初期化、終了処理
- 複数のデバイスをファイルシステム(FATFS)に対応させる
 - デバイス毎にアクセス処理(関数)が異なる
- ファイルシステム(FATFS)の関数をPOSIXやC言語標準ファイル関数(ファイルライブラリ)に紐付ける機能をもつ
 - fopen/fclose/fread/fwrite等
 - open/close/read/write等

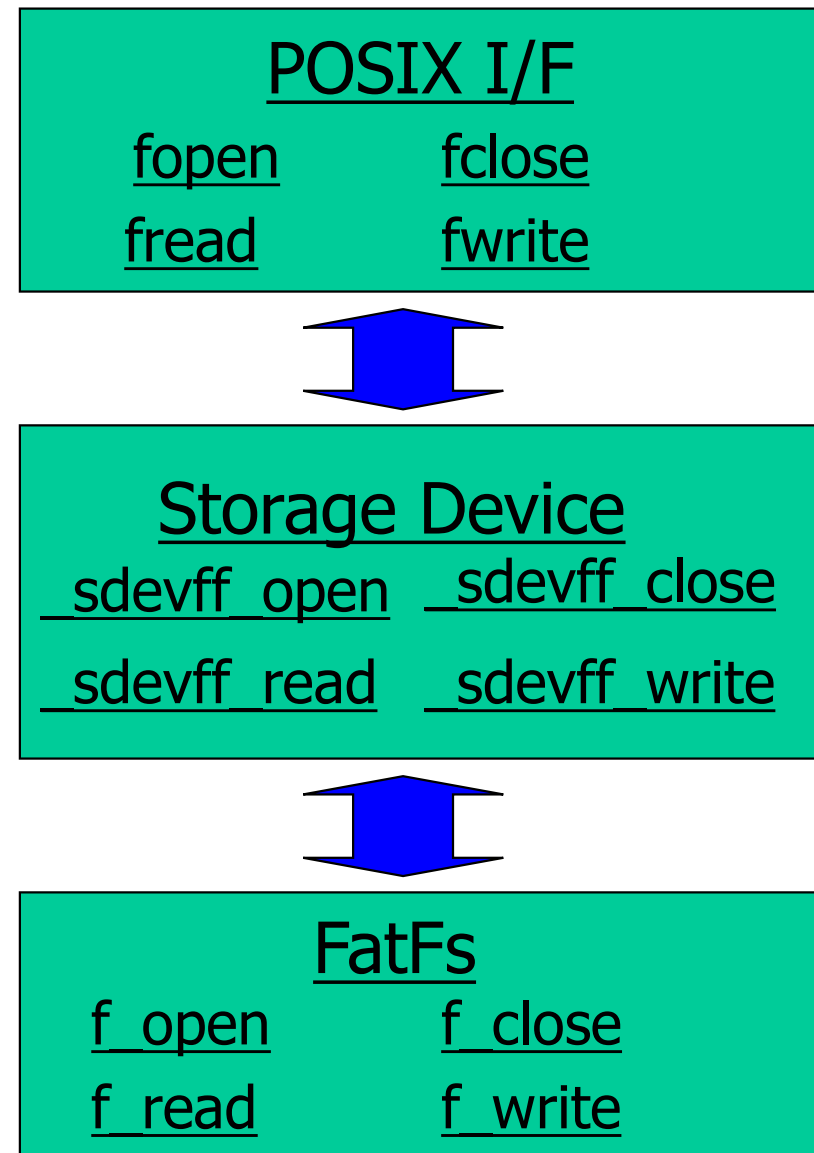


LINUX等で開発したファイルプログラムがデバイスに関係なく、組込み機器で使えるようになる

注1) SPI通信には挿抜検知はない

複数のデバイスをファイルシステムに対応させる

- ストレージデバイスマネージャは教材として作成したデバイスドライバ管理プログラムです
- アプリケーションに対してPOSIXファイルインターフェイスを供給する
- POSIXファイルインターフェイスからのアクセスをデバイス毎のコールバック関数を通して各ファイルアクセスインターフェイスに変換する



ストレージデバイスマネージャ・インクルードファイル

- storagedevice.hはデバイスハンドラ(StorageDevice_t)を定義
- ハンドラは個々のデバイスに対して定義し、デバイスの状態と情報の管理、通知のコールバック関数、デバイスドライバ関数構造体(StorageDeviceFunc_t)、ファイルライブラリ関数構造体(StorageDeviceFileFunc_t)を持つ

```
/*  
 * ストレージデバイス型定義  
 */  
typedef struct StorageDevice {  
    uint16_t    _sdev_attribute;  
    uint8_t     _sdev_devno;  
    uint8_t     _sdev_port;  
    uint32_t    _sdev_maxsec;  
    uint32_t    _sdev_secsz;  
    uint16_t    _sdev_instimer;  
    uint16_t    _sdev_inswait;  
    void        (*_sdev_notice)(void *psdev, bool_t sw);  
    void        *_sdev_local[4];  
    StorageDeviceFunc_t *pdevf;  
    StorageDeviceFileFunc_t *pdevff;  
} StorageDevice_t;
```

デバイスの属性(次ページ)

通知用コールバック関数

デバイスの初期化、アクセス関数を持つ構造体

ファイルライブラリ用関数構造体

ストレージデバイスマネージャ・インクルードファイル

- デバイス番号 (0～9)
- `_sdev_attribute` のビット定義
 - `SDEV_ACTIVE` が ON ならデバイスアクティブ
 - `SDEV_EMPLOY` が ON ならメディア使用可能

```
#ifndef NUM_STORAGEDEVICE
#define NUM_STORAGEDEVICE 4          /* デフォルトのストレージデバイス数 */
#endif
#define MINIMUM_DEVNO 0
#define MAXIMUM_DEVNO 9
#ifndef DEFAULT_DEVNO
#define DEAFULT_DEVNO MINIMUM_DEVNO
#endif

#define SDEV_ACTIVE (1<<15)         /* アクティブなデバイス */
#define SDEV_INSERTCHK (1<<14)      /* 挿入・排出の設定あり */
#define SDEV_CHKREMOVE (1<<13)      /* 排出検査する */
#define SDEV_ONEXIT (1<<12)         /* 一度排出あり */
#define SDEV_EMPLOY (1<<8)           /* 使用可能 */
#define SDEV_DEVEERROR (1<<7)       /* DEVICE ERROR */
#define SDEV_DEVNOTUSE (1<<0)       /* DEVICE 使用不可 */
#define SDEV_NOTUSE 255
```

ストレージデバイスマネージャ・インクルードファイル

- StorageDeviceFunc_tはデバイス毎のデバイス関数を定義する関数構造体である
 - _sdevf_senseが、挿抜チェックとデバイスの初期化、終了処理を行う関数
 - 他はファイルシステム用アクセスドライバ
 - TOPPERS BASE PLATFORMでは各デバイスに対応した関数が実装済みである

```
/*
 * デバイスドライバー用関数テーブル定義
 */
typedef struct StorageDeviceFunc {
    int (*_sdevf_sense)(void *psdev, bool_t on);
    int (*_sdevf_diskinit)();
    int (*_sdevf_diskstatus)();
    int (*_sdevf_diskread)();
    int (*_sdevf_diskwrite)();
    int (*_sdevf_diskioctl)();
} StorageDeviceFunc_t;
```

_sdevf_senseを実装すれば、
メイン関数に初期化処理を記
載する必要はない
TOPPERS BASE PLATFORMの
files/ff/sddiskio.c(sdcard_sen
s)に実装済み

ストレージデバイスマネージャ・コンフィギュファイル

- 初期化関数の設定

- sdev_init : ストレージデバイスマネージャの初期化
- stdfile_init : ファイルライブラリの初期化
- volume_info_init : ストレージ用デバッグコマンド拡張

- SDM_TASKはメディアの挿抜チェック用タスク、挿抜機能がない場合(コンパイルスイッチ: SDEV_SENSE_ONETIMEを有効化)はタスク化しない

```
#include "storagedevice.h"
ATT_INI({TA_NULL, 0, sdev_init});
ATT_INI({TA_NULL, 0, stdfile_init});
ATT_INI({TA_NULL, 0, volume_info_init});
CRE_SEM(SEM_STDFILE, { TA_TPRI, 1, 1 });

#ifdef SDEV_SENSE_ONETIME
CRE_TSK(SDM_TASK, { TA_ACT, 1, SDMSence_task, 15, 1024, NULL });
#endif
```

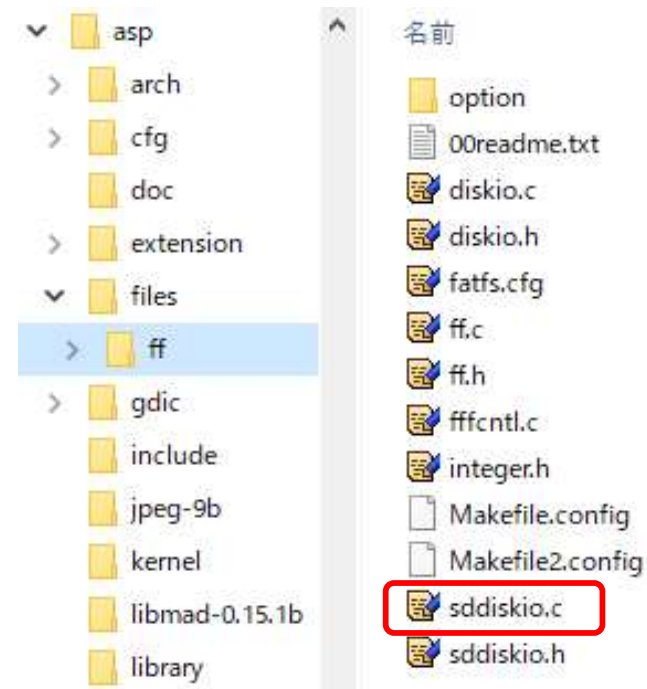
ストレージデバイスマネージャ機能

- ストレージデバイスマネージャは7つの関数で制御する
- デバイスが存在し有効化するためにSDMSetupDeviceで登録を行う: 登録後は関数構造体で動作する
- SDEV_SENSE_ONETIMEが設定されている場合、SDMSence_taskはデバイスの初期化関数として動作

関数名	型	機能
sdev_init	void	ストレージデバイスマネージャの初期化
sdev_terminate	void	ストレージデバイスマネージャの終了
SDMSetupDevice	ER	デバイスをセットする
SDMGetDeviceNo	ER_ID	パス名からデバイスIDを取り出す
SDMGetStorageDevice	StorageDevice_t*	デバイスIDからStorageDevice_t(デバイスハンドラ)を取り出す
SDMEmploy	ER	デバイスの属性を更新する
SDMSence_task	void	挿抜チェック、デバイス初期化等を行う

演習: sd_ini3ストレージデバイスマネージャ対応

- SDカード: ファイルシステムに対応するために、SPI通信の初期化処理をストレージデバイスマネージャで実行させるように修正する(次の章でファイルシステムを動かす、ここでは初期化のみ代行させる)
 - TOPPERS BASE PLATFORMの機能を使ってSDカード: SPI通信の初期化を行う
- 学習内容
 - ストレージデバイスマネージャによって、ハードウェアと初期化プログラムが分離されることを確認
- 手順
 - sd_ini2をコピーしてsd_ini3を作り、改造する
 - SDカードのドライバ
files/ff/sddiskio.cを取り込み改造する
 - 正式な初期化手順のファイルライブラリ関数構造体設定と
f_mountは、ここでは使用しない



演習 : Makefileの修正

- asp/files/ff/sddiskio.cをsd_ini3にコピーする
- storeagedevice.cが、システムサービスとして追加されるように修正する
 - Makefileの212行目にSYSSVC_COBJSにコピーしたsddiskio.oが追加されるように修正する
 - storagedeviceは取り込み済

```
207 #
208 # ストレージファイルマネージャー
209 #
210 INCLUDES := $(INCLUDES) -I$(SRCDIR)/files
211 SYSSVC_DIR := $(SYSSVC_DIR) $(SRCDIR)/files
212 SYSSVC_COBJS := $(SYSSVC_COBJS) storagedevice.o sddiskio.o
```

コンフィギュレーションファイルの修正

- sd_access.cfgを修正
- storagedevice.hインクルード参照を追加
- ATT_INIを使ってストレージデバイスマネージャの初期化を追加
- SDカードをセンスするタスクを追加

```
21 #include "device.h"
22 #include "spi.h"
23 #include "storagedevice.h"
24 #include "sd_access.h"
25
26 ATT_INI({ TA_NULL, sw_int, setup_sw_func });
27 ATT_INI({ TA_NULL, 0, device_info_init });
28 ATT_INI({TA_NULL, 0, sdev_init});
29
30 CRE_TSK(MAIN_TASK, { TA_ACT, 0, main_task, MAIN_PRIORITY, STACK_SIZE, NULL });
31 CRE_TSK(SDM_TASK, { TA_ACT, 1, SDMSence_task, 15, 1024, NULL });
```

ソースファイルの修正

- ストレージデバイスマネージャがメインタスクで使用できるようにstoragedevice.hをインクルードとして追加
- メインタスクではシステムサービスが初期設定したデバイスハンドラ(StorageDevice_t)を使用して、ストレージデバイスマネージャを操作することができる
- デバイスハンドラはデバイスIDをキーに取り出せる
 - SDカードでデバイスIDは「SDCARD_DEVNO」

```
57 #include "device.h"
58 #if BOARDNO != 0
59 #include "dtime.h"
60 #endif
61 #include "spi.h"
62 #include "pinmode.h"
63 #include "spi_driver.h"
64 #include "storagedevice.h"
65 #include "sd_access.h"
```

ソースファイルの修正

- メインタスクをストレージデバイスマネージャを呼び出せるようにStorageDevice_t(デバイスハンドラ)へのポインタ *pdevを変数を追加

```
120 void
121 main_task(intptr_t exinf)
122 {
123     StorageDevice_t *psdev;
124     SPI_Init_t    spi_initd;
125     SDCARD_Handler_t *hsd = NULL;
126     SPI_Handle_t  *hspi;
127     ER_UINT       ercd;
128     int i;
```

ソースファイルの修正

- sdcard_init以降の初期化関数を以下に置き換える
 - sd_init : ストレージデバイスマネージャーにSDカード設定
 - センスタスクがSDカードを認識して、hsdをセットするのを待つ
 - SDGetStorageDeviceでストレージデバイスハンドラを取出しローカル領域からSPIDRIVERハンドラを取得

```
195  spi_wait(hspi, 50);
196 #endif
197  dly_tsk(100);
198  spi_deinit(hspi);
199  spi_initd.Bardrate = 4*1000*1000;
200  hsp = spi_init(SPI1_PORTID, &spi_initd);
201  sd_init(0);
202  sdcard_setspi2(SDCARD_PORTID, hspi, CS2_PORT);
203  psdev = SDMGetStorageDevice(SDCARD_DEVNO);
204  for(i = 0 ; i < 50 && hsd == NULL ; i++){
205      hsd = (SDCARD_Handler_t *)psdev->_sdev_local[1];
206      dly_tsk(100);
207  }
```

sdcard_sense関
数と同等の処理

修正前の処理

```
sdcard_setup2(SDCARD_PORTID, hspi, CS2_PORT);
if(sdcard_init(SDCARD_PORTID)){
    hsd = sdcard_open(SDCARD_PORTID);
    SVC_ERROR(sdcard_checkCID(hsd));
    SVC_ERROR(sdcard_sendCSD(hsd));
    SVC_ERROR(sdcard_getcardinfo(hsd, &SDCardInfo));
    SVC_ERROR(sdcard_configuration(hsd));
}
else{
    syslog_0(LOG_ERROR, "SD-CARD INITIAL ERROR !");
    slp_tsk();
}
if(hsd != NULL)
    hsd->spi_lock = SPI1LOCK_SEM;
```

ソースファイルの修正

- センスタスクによる初期化終了を待ち、エラー判定を行う
- ビルドを行うと、sddiskioからリンクエラーが発生する

```
208  if(hsd != NULL){
209      hsd->spi_lock = SPI1LOCK_SEM;
210      while((psdev->_sdev_attribute & SDEV_EMPLOY) == 0){
211          dly_tsk(100);
212      }
213  }
214  else{
215      syslog_0(LOG_ERROR, "SD-CARD INITIAL ERROR !");
216      slp_tsk();
217  }
218  if(hsd != NULL && hsd->hspi != NULL){
```

sdcard_sense関
数と同等の処理

修正前の処理

```
sdcard_setup2(SDCARD_PORTID, hspi, CS2_PORT);
if(sdcard_init(SDCARD_PORTID)){
    hsd = sdcard_open(SDCARD_PORTID);
    SVC_ERROR(sdcard_checkCID(hsd));
    SVC_ERROR(sdcard_sendCSD(hsd));
    SVC_ERROR(sdcard_getcardinfo(hsd, &SDCardInfo));
    SVC_ERROR(sdcard_configuration(hsd));
}
else{
    syslog_0(LOG_ERROR, "SD-CARD INITIAL ERROR !");
    slp_tsk();
}
if(hsd != NULL)
    hsd->spi_lock = SPI1LOCK_SEM;
```

リンクエラーの原因

- fatfsSDeviceFileFuncとf_mountがリンクエラーとなる
 - fatfsSDeviceFileFuncはファイルライブラリ関数配列
 - f_mountはFatFsのマウント関数
- 今回は初期化をプラットフォームで実行するだけで、上記はファイルシステムがない場合不要な関数なので、コピーしたsddiskio.cをエラーが出ないように修正する

```

M ~/toppers/base3/my_program/sd_ini3
_pico_gcc/rp2040_ram.ld -o asp.exe \
    sd_access.o command.o log_output.o vasylog.o t_perror.o strerr
or.o bs2_default_padded_checksummed.o target_inithook.o chip_serial.o task_ex
pansion.o monitor.o stddev.o printf.o sprintf.o fprintf.o scanf.o sscanf.o armv7
m.o spi_driver.o storagedevice.o sddiskio.o banner.o syslog.o serial.o logtask.o
device.o pio_spi.o spi.o pinmode.o kernel_cfg.o -lkernel -lc -lgcc
c:/program files (x86)/gnu arm embedded toolchain/10 2020-q4-major/bin/./lib/gc
c/arm-none-eabi/10.2.1/./././././arm-none-eabi/bin/ld.exe: sddiskio.o: in func
tion 'sdcard_sense':
C:\msys64\home\roi39\toppers\base3\my_program\sdk_ini3\sddiskio.c:166: undefined
reference to 'f_mount'
c:/program files (x86)/gnu arm embedded toolchain/10 2020-q4-major/bin/./lib/gc
c/arm-none-eabi/10.2.1/./././././arm-none-eabi/bin/ld.exe: C:\msys64\home\roi3
9\toppers\base3\my_program\sdk_ini3\sddiskio.c:129: undefined reference to 'f_mou
nt'
c:/program files (x86)/gnu arm embedded toolchain/10 2020-q4-major/bin/./lib/gc
c/arm-none-eabi/10.2.1/./././././arm-none-eabi/bin/ld.exe: sddiskio.o: in func
tion 'sd_init':
C:\msys64\home\roi39\toppers\base3\my_program\sdk_ini3\sddiskio.c:115: undefined
reference to 'fatfsSDeviceFileFunc'
collect2.exe: error: ld returned 1 exit status
make: *** [Makefile:367: asp.exe] Error 1

roi39@DESKTOP-JMDVGP MINGW64 ~/toppers/base3/my_program/sd_ini3
$

```

sddiskio.cの修正

- リンクエラーとなるsd_init関数中のpsdev->pdevffの設定はファイルライブラリを使用しない場合は、不要なのでコメントで削除する

```
94 void sd_init(intptr_t exinf)
95 {
96     StorageDevice_t *psdev;
97
98     SDMSSetupDevice(SDCARD_DEVNO, &psdev);
99     psdev->pdevf      = (StorageDeviceFunc_t *)&fatfsSDeviceFunc;
100     // psdev->pdevff   = (StorageDeviceFileFunc_t *)&fatfsSDeviceFileFunc;
101     psdev->_sdev_sectsize = 512;
102     psdev->_sdev_port     = SDCARD_PORTID;
103     #ifdef SDEV_INSWAIT_TIME
104     psdev->_sdev_inswait  = SDEV_INSWAIT_TIME;
105     #else
106     psdev->_sdev_inswait  = 0;
107     #endif
108     psdev->_sdev_attribute |= SDEV_INSERTCHK|SDEV_CHKREMOVE;
109     psdev->_sdev_local[0]  = &ActiveFatFsObj;
110 }
```

sddiskio.cの修正

- sdcard_sense関数はSPI/MMCの初期化を行う関数である
- FatFsを取り込んでいない、f_mountはFatFsのマウント関数なのでコメント文削除する

```
115 static int sdcard_sense(void *pif, BOOL on)
116 {
117     StorageDevice_t *psdev = pif;
118     bool_t exist = mci_ses_por(((StorageDevice_t *)psdev)->_sdev_port);
119     MCIPCB *pmci;
120     int result = FR_DISK_ERR;
121
122     pmci = psdev->_sdev_local[1];
123     if(on && !exist){
124         // f_mount(psdev->_sdev_devno, 0);
125         psdev->_sdev_attribute &= ~SDEV_DEVEERROR;
126         mci_cls_por(pmci);
127         return TRUE;
128     }
129     else if(!on && exist){
130         psdev->_sdev_instimer += SENSE_TIME;
131         if((psdev->_sdev_attribute & SDEV_ONEEXIT) != 0 && ....
132         return FALSE;
```

sddiskio.cの修正

- 165-169のf_mountの呼び出し部分もコメント文で削除
- ビルドでできたら、ダウンロード、実行して正しく初期化できることを確認する

```
156     if(MciConfiguration(pmci) != E_OK)
157         psdev->_sdev_attribute |= SDEV_DEVEERROR;
158     if(MciSetWideBus(pmci) != E_OK)
159         psdev->_sdev_attribute |= SDEV_DEVEERROR;
160     //  if((psdev->_sdev_attribute & SDEV_DEVEERROR) == 0)
161     //      result = f_mount(psdev->_sdev_devno, &ActiveFatFsObj);
162     //  if(result != FR_OK)
163     //      psdev->_sdev_attribute |= SDEV_DEVEERROR;
164     //  else
165         psdev->_sdev_local[0] = &ActiveFatFsObj;
166     #if 1
```

sddiskio.cの検証

- sdcard_sense関数では、SPIの初期化用にSPI_DRIVERのSPI初期化関数をコールしていない
- なのに、なぜ、SPIの初期化が行われるのか？

```
else if(!on && exist){
    psdev->_sdev_instimer += SENSE_TIME;
    if((psdev->_sdev_attribute & SDEV_ONEEXIT) != 0 && psdev->_sdev_instimer < ...
        return FALSE;
    pmci = mci_opn_por(((StorageDevice_t *)psdev)->_sdev_port);
    if(pmci == NULL)
        return FALSE;
    psdev->_sdev_local[1] = pmci;
    if(MciCheckCID(pmci) != E_OK){
        psdev->_sdev_attribute |= SDEV_DEVEERROR;
        return TRUE;
    }
    if(MciSetAddress(pmci) != E_OK){
        psdev->_sdev_attribute |= SDEV_DEVEERROR;
        return TRUE;
    }
}
```

sddiskio.cの検証

- sddiskio.cにリンクするmcicmd.hがsddiskioでの初期化関数をSPI_DRIVERの初期化関数にリネームして置き換える
- MMC用の初期化関数はgdc/mcicmd.hにあるsddiskはこれを参照すれば、まったくプログラムを書き換えず、SDカードMMCの初期化を行うことができる

mcicmd.hの内容

```
#define mci_ses_por    sdcard_init
#define mci_opn_por    sdcard_open
#define mci_cls_por    sdcard_close
#define MciCheckCID    sdcard_checkCID
#define MciSetAddress   sdcard_dummy
#define MciSendCID     sdcard_sendCSD
                        :
```

SPI-SDカードFATドライバ対応

1. READコマンド作成
2. セクタデータの書き換え
3. SPIの依存性をなくする

演習：セクタREADデバッグコマンド(sd_cmd1)

- デバッグコマンドを使って指定したセクタをREADするプログラムを作る
- 学習内容
 - 開発補助となるデバッグコマンドの作成
- 手順
 - sd_ini3をコピーしてsd_cmd1を作成して改造
 - command.cを改造
- 仕様
 - DEVICE READ セクタ番号(return)
 - 指定したセクタ番号のデータをDUMPする

演習 : command.cの改造

- program/sd_ini3中のcommand.cは1から10セクタまでのセクタWRITEできるプログラムが実装されている
 - 通常のSDカードでは1から10セクタはreservedセクタであり、データが書き換えられても問題ない
- command.cを改造する
- セクタREADを行うread_funcを作成し、デバッグコマンドに登録すればよい

```
/*  
 * デバイスコマンド番号  
 */  
static int_t clk_func(int argc, char **argv);  
:  
static int_t cup_func(int argc, char **argv);  
static int_t read_func(int argc, char **argv);  
static int_t write_func(int argc, char **argv);
```

演習: command.cの改造

- 全ページを含めて青文字の3行を追加すればコマンドが登録できる

```
static const COMMAND_INFO device_command_info[] = {
    {"CLK",          clk_func},
    :
    {"CUP",          cup_func},
    {"READ",         read_func},
    {"WRITE", write_func}
};

#define NUM_DEVICE_CMD (sizeof(device_command_info)/sizeof(COMMAND_INFO))

static const char device_name[] = "DEVICE";
static const char device_help[] =
" Device CLK (index)          return clock¥n"
"      VER                   display version¥n"
"      :
"      DIP (no) (on-1,off-0) dipsw control¥n"
"      READ command (no)     sdcard read  ¥n"
"      WRITE command (no)    sdcard write ¥n";
```

演習: read_funcを作成

- write_func関数とメインタスク中の0セクタDUMPプログラムを参考にして、read_func関数を完成されてください
- READのバッファはsdbufferを使用する

```
/*  
 * SDカードセクタREAD  
 */  
static int_t read_func(int argc, char **argv)  
{  
    :  
    return 0;  
}
```

演習：ソースファイルの修正(ヒント)

- メインタスクの最初の0セクタ表示(赤文字の行)は read_funcとして利用可能

```
hspi = spi_start(&SPI_Setup);
if(hsd != NULL && hsd->hspi != NULL){
    unsigned char *p;
    int j, wk1, wk2;

    ercd = sdcard_blockread(hsd, sdbuffer, 0*512, 512, 1);
    syslog_2(LOG_NOTICE, "## ercd(%d) sector #0 sdbuffer[%08x] ##", ercd, sdbuffer);
    dly_tsk(300);
    p = (unsigned char *)sdbuffer;
    printf("¥nsec 0) ");
    for(j = 0 ; j < 512 ; j++){
        if((j % 16) == 0)
            printf("¥n%03x ", j);
        printf("%02x ", p[j]);
    }
    printf("¥n");

    SVC_ERROR(sdcard_getcardinfo(hsd, &SDCardInfo));
```

セクタREAD正解

- デバイスIDをキーにSPDRIVERのハンドラを取り出せれば、
sdcard_blockread関数を用いてセクタREADができる

```
/*  
 * SDカードセクタREAD  
 */  
static int_t read_func(int argc, char **argv)  
{  
    StorageDevice_t *psdev = SDMGetStorageDevice(SDCARD_DEVNO);  
    SDCARD_Handler_t *hsd;  
    uint32_t sect;  
    unsigned char *p;  
    ER ercd;  
    int j;  
    if(argc < 2)  
        return -1;  
    sect = atiox(argv[1]);  
    if(psdev == NULL)  
        return -1;  
    hsd = (SDCARD_Handler_t *)psdev->_sdev_local[1];  
    if(hsd == NULL)  
        return -1;
```

セクタREAD正解

- sdcard_blockreadの引数は
 - SPIDRIVERハンドラ、READメモリアドレス
 - セクタ位置、セクタサイズ、セクタ数
 - セクタ位置は64ビットデータ(sect*セクタサイズ)

```
ercd = sdcard_blockread(hsd, sdbuffer, sect*512, 512, 1);
if(ercd != E_OK){
    printf("sdcard read error(%d)\n", ercd);
    return -1;
}
dly_tsk(100);
p = (unsigned char *)sdbuffer;
printf("\nsect(%d) [%08x] ", sect, (unsigned int)sdbuffer);
for(j = 0 ; j < 512 ; j++){
    if((j % 16) == 0)
        printf("\n%03x ", j);
    printf("%02x ", p[j]);
}
printf("\n");
return 0;
}
```

SDカードセクタダンプ

- ビルドし、ダウンロード実行する
- returnキーでプロンプトを表示するので
- device read 2(return)で2セクタ目をダンプ

```
COM3 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
1e0 61 6e 79 20 6b 65 79 20 74 6f 20 72 65 73 74 61
1f0 72 74 0d 0a 00 00 00 00 00 00 00 00 ac cb d8 55 aa
status[00]
cardtype(1) capacity(1990197248) blocksize(1024) maxsector(3887104)
## STOP ##

mon>device read 2

sec 2) [2000be80]
000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

SPI-SDカードFATドライバ対応

1. READコマンド作成
2. セクタデータの書き換え
3. SPIの依存性をなくする

演習: セクタ1のデータ書き換え

- 作成したセクタREAD/WRITEコマンドを使ってセクタ1を書き換える

```
mon> dev read 1
sect(1) [2000ed84]
000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
:
mon>s b 2000ed84
2000ed84 00 =aa aa
2000ed85 00 =bb bb
2000ed86 00 =.
mon>dev write 1
sect(1) OK
mon>dev read 1
sect(1) [2000ed84]
000 aa bb 00 00 00 00 00 00 00 00 00 00 00 00 00 00
010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
:
```

セクタ1をsdbufferに読み込み

set byteコマンドで1バイト目と2バイト目を書き替え

sdbufferからセクタ1に書き込み

セクタ1をsdbufferに再び読み込み

SPI-SDカードFATドライバ対応

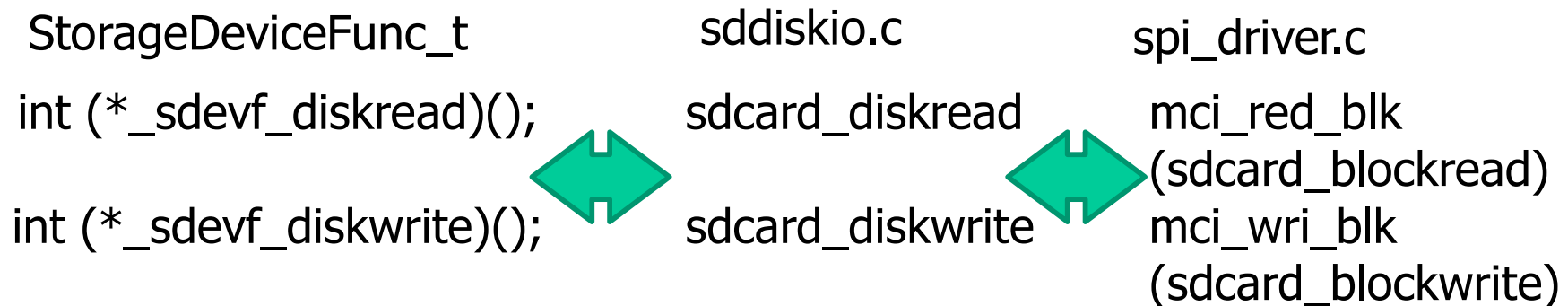
1. READコマンド作成
2. セクタデータの書き換え
3. SPIの依存性をなくする

演習: sd_cmd2

- 作成したセクタREAD/WRITEコマンドを改造しSPI以外のデバイスにも使用可能にする
- 学習内容
 - READ/WRITE等のアクセス機能の隠蔽化
- 手順
 - sd_cmd1をコピーしてsd_cmd2を作り、修正する
 - command.cを改造する
 - command.cからSPIDRIVERへの直接コールが無くても、セクタREAD/WRITEできることを確認

READ/WRITE関数の隠蔽化

- FatFsからのREAD/WRITE処理
 - FatFsからのREAD/WRITEはストレージデバイスマネージャのStorageDeviceFunc_t構造体を介して行われる
- FatFsからのREADは、_sdevf_diskread関数ポインタを呼び出して、mci_red_blkがリネームするsdcard_blockread関数を呼び出す
- これにより、関数ポインタを入れ替えれば種々のデバイスのREAD/WRITEに対応が可能となる



演習 : commandの改造内容

- 現状、デバッグコマンドはSPIDRIVERを通してセクタREAD/WRITEする。通信方式に関係なくストレージデバイスマネージャを介してセクタREAD/WRITEできるように修正する
- 仕様
 - ストレージデバイスドライバのデバイス関数(`StorageDeviceFunc_t`)を用いてセクタREAD/WRITEする
 - FatFsはデバイス関数を用いてREAD/WRITEを行うため、通信方式が変わっても、FatFsのデバイスドライバには変更は発生しない

演習 : command.cの修正

- インクルードファイルをspidriver.hからdiskio.hに入れ替える
- diskio.hはFatFsが提供するデバイスドライバ用インクルードファイル
 - ドライバの返り値等の定義がある

```
#include "syssvc/serial.h"
#include "syssvc/syslog.h"
#include "kernel_cfg.h"
#include "device.h"
#if BOARDNO != 0
#include "dtime.h"
#endif
#if BOARDTYPE==PICO_W
#include "cyw43.h"
#endif
#include "diskio.h"
#include "storagedevice.h"
#include "sd_access.h"
```

spi_driver.hを書き換える
上位のモジュールは
下位のドライバ(spi_driver.h)
を参照しなくてよくなる

演習: read_funcの修正

- SPIDRIVERハンドラを取り出さず、直接READ用の関数(_sdevf_diskread)でセクタREADを行う
- 戻り値の判定はdiskio.hの戻り値で判定

```
/*
 * SDカードセクタREAD
 */
static int_t read_func(int argc, char **argv)
{
    StorageDevice_t *psdev = SDMGetStorageDevice(SDCARD_DEVNO);
    uint32_t sect;
    unsigned char *p;
    int result, j;
    if(argc < 2)
        return -1;
    sect = atoi(argv[1]);
    if(psdev == NULL)
        return -1;
    result = psdev->pdevf->_sdevf_diskread(psdev, sdbuffer, sect, 1);
    if(result != RES_OK){
        printf("sdcard read error(%d)\n", result);
        return -1;
    }
    dly_tsk(100);
}
```

演習: write_funcの修正

- READと同様に書き換える

```
/*
 * SDカードセクタWRITE
 */
static int_t write_func(int argc, char **argv)
{
    StorageDevice_t *psdev = SDMGetStorageDevice(SDCARD_DEVNO);
    uint32_t sect;
    int result;
    if(argc < 2)
        return -1;
    sect = atoi(argv[1]);
    if(sect == 0 || sect > 10){
        printf("sdcard write sector error(%d)¥n", sect);
        return -1;
    }
    if(psdev == NULL)
        return -1;
    result = psdev->pdevf->_sdevf_diskwrite(psdev, sdbuffer, sect, 1);
    if(result != RES_OK){
        printf("sdcard write error(%d)¥n", result);
        return -1;
    }
}
```

SDカードファイルシステムの構築

1. [FatFs](#)
2. ファイルシステムの対応
3. ファイルライブラリの対応
4. ファイルユーティリティの対応

FatFs: FATファイルシステム

- FatFsは赤松武志さんの作成によるFATファイルシステムモジュールです
- フリーソフトウェアとして公開
- FATファイルシステムでは、ストレージデバイスの読み出し、書き込みをデバイスインターフェイスにて行う
- FatFsでは、デバイスインターフェイス関数をdiskio.cというソースに記載している
- この教材では、diskio.cのデバイスインターフェイス関数を作成する実習を行う
- diskio.cをSDメモリーカードアクセス用のデバイスインターフェイス関数に書き換えることによって、SDカードをFATファイルとしてアクセスすることができる

FatFsの概要

- FatFsはフリーソフトウェアとして公開されている
- FatFsは以下の特徴を持つ
 - Windows互換FATファイル・システム
 - プラットフォームに依存しない
 - コンパクトなコード・サイズとRAM使用量
 - 複数のボリュームに対応（物理ドライブ・区画）
 - 複数のANSI/OEMコードページに対応
 - ロングファイルネームにも対応
（Unicode APIも選択可能）
 - RTOSに対応
 - 記述はC89に準拠

FatFsのAPI関数

- FatFsの上位レイヤへのインターフェイスは以下のとおり

API関数	機能
f_mount	ワーク・エリアの登録削除
f_open	ファイルのオープン・作成
f_close	ファイルのクローズ
f_read	ファイルの読み出し
f_write	ファイルの書き込み
f_sync	キャッシュデータのフラッシュ
f_lseek	リード/ライト・ポインタの移動
f_opendir	ディレクトリのオープン
f_readdir	ディレクトリの読み出し
f_stat	ファイル情報の取得
f_getfree	ボリュームの空き領域の取得
f_truncate	ファイルの切り詰め
f_unlink	ファイルやディレクトリの削除

API関数	機能
f_mkdir	ディレクトリの作成
f_chmod	アトリビュートの変更
f_utime	タイムスタンプの変更
f_rename	名前の変更
f_chdir	カレント・ディレクトリの変更
f_chdrive	カレント・ドライブの変更
f_mkfs	ボリュームのフォーマット
f_forward	ファイル・データをストリームに転送
f_putc	文字の書き込み
f_puts	文字列の書き込み
f_printf	書式付き文字列の書き込み
f_gets	文字列の読み込み

本教材では未使用

FatFsのデバイスインターフェイス

- FatFsがボリュームにアクセスするために、デバイスインターフェイスを用意する必要がある
- 要求されるデバイスインターフェイスはオプション指定に従う

デバイスインターフェイス関数	必要となる条件	機能
disk_initialize	常時	ドライブの初期化
disk_status		ディスクステータス取得
disk_read		セクタの読み込み
disk_write	_FS_READONLY == 0	セクタの書き込み
get_fattime		時刻の取得
disk_ioctl(CTRL_SYNC)		書き込み時の同期
disk_ioctl(GET_SECTOR_COUNT)	_USE_MKFS == 1	セクタサイズ取得
disk_ioctl(GET_SECTOR_SIZE)	_MAX_SS >= 1024	総セクタ数取得
disk_ioctl(GET_BLOCK_SIZE)	_USE_MKFS == 1	消去ブロックサイズ取得

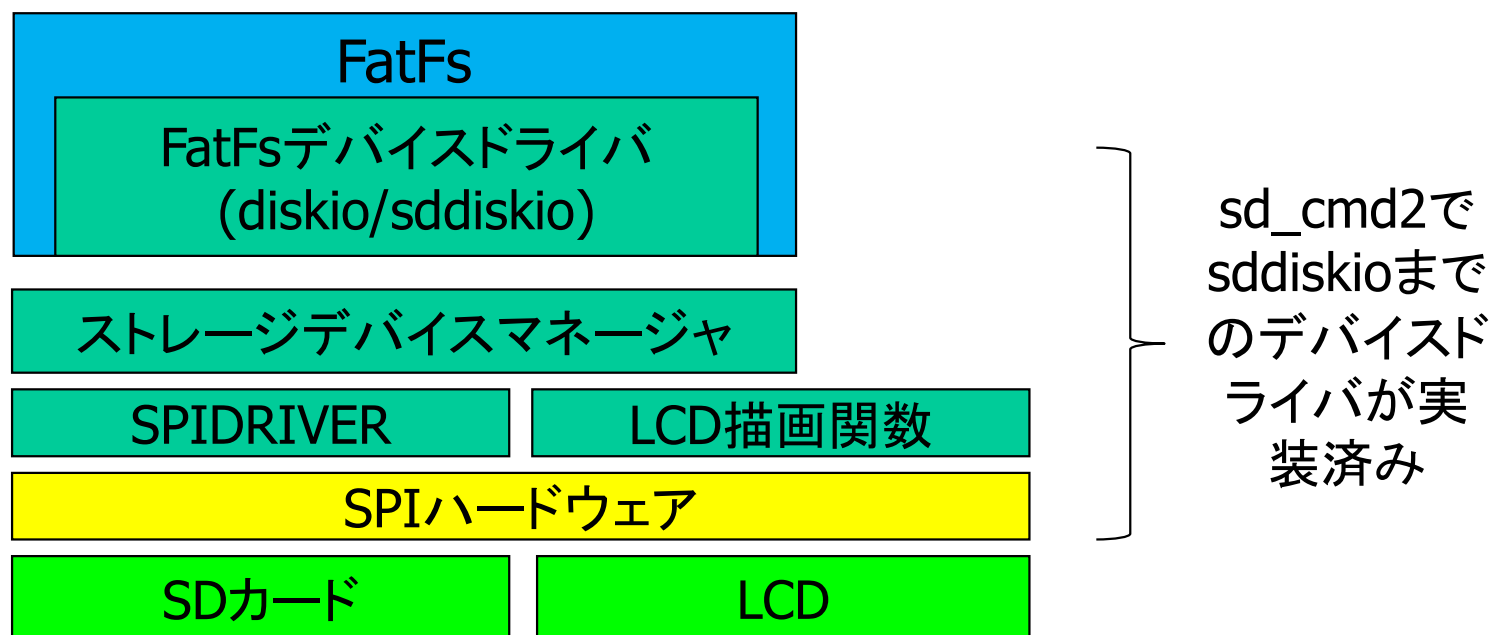
 は本教材では使用しない

SDカードファイルシステムの構築

1. FatFs
2. [ファイルシステムの対応](#)
3. ファイルライブラリの対応
4. ファイルユーティリティの対応

現状のプログラムの構造(sd_cmd2)

- これまでの作業で、sddiskioまでのデバイスドライバが実装済み
- FatFsとFatFs用デバイスドライバ(diskio.c)を取り込めばFatFsが動作するようになる



演習: sd_file1ファイルWRITEプログラム

- FatFsの取り込み、FatFsのファイル関数を使ってSDカードにdata.txtファイルを作成する
- 学習内容
 - FatFsのファイル関数を使ってファイル処理を確認
- 手順
 - my_programにsd_file1をコピーし、diffコマンドでsd_cmd2と比較
 - sd_file1をビルドする
 - ダウンロード実行し、file_test.cのファイル処理を行う
- 仕様
 - ファイル処理はfile_test.cに用意
 - 使用するファイル関数は
 - f_open/f_write/f_close

演習: sd_file1とsd_cmd2との違い

- FatFsを取り込むにはfiles/ff以下のコンフィギュレーションファイルとソースファイルを取り込めばよい
 - ff.c : FatFs本体
 - diskio.c : FatFs用デバイスドライバ
 - sddiskio.c : SDカード対応デバイスドライバ
 - option/syncobj.c : SYNC－OBJECT
 - option/cc932.c : 日本語UCODE
 - fatfs.cfg : FatFsで使用するRTOSリソース
- ソースファイル、インクルードパスはfiles/ff/Makefile.configに記載しているのでMakefileで、このMakefile.configをインクルードするだけでFatFsを取り込むことができる

演習 : files/ff/Makefile.config

- files/ff/Makefile.configの記載は以下のとおり
- コンフィギュレータ以外の設定がすべてされている
- Makefile.configをインクルードすれば、細かいファイルのMakefileへの記述はしなくて済む

```
#  
# コンパイルオプション  
#  
INCLUDES := $(INCLUDES) -I$(SRCDIR)/files/ff  
  
#  
# ファイルシステムに関する定義  
#  
SYSSVC_DIR := $(SYSSVC_DIR) $(SRCDIR)/files/ff $(SRCDIR)/files/ff/option  
SYSSVC_COBJS := $(SYSSVC_COBJS) ff.o syncobj.o cc932.o diskio.o sddiskio.o ¥  
                fffcntl.o
```

演習 : Makefileの差異

- files/ff/Makefile.configの取り込み
- file_test.oを追加、sddiskio.oを削除

```
176 APPL_CXXOBS = $(APPLNAME).o
177 APPL_COBS = command.o file_test.o
178 else
179 APPL_COBS = $(APPLNAME).o command.o file_test.o
180 endif
```

ファイルテストプログラムを別ファイルで指定

```
188 #
189 # ミドルウェア Makefile のインクルード
190 #
191 include $(SRCDIR)/monitor/Makefile.config
192 include $(SRCDIR)/files/ff/Makefile.config
193
```

files/ff/Makefile.configの取り込み

```
208 #
209 # ストレージファイルマネージャー
210 #
```

sddiskio.oの取り込みをやめる
files/ff/Makefile.configに記載があるため

```
211 INCLUDES := $(INCLUDES) -I$(SRCDIR)/files -I$(SRCDIR)/files/ff
212 SYSSVC_DIR := $(SYSSVC_DIR) $(SRCDIR)/files $(SRCDIR)/files/ff
213 SYSSVC_COBS := $(SYSSVC_COBS) storagedevice.o sddiskio.o
```

演習 : sd_access.cfgの差異

- FatFs用RTOSリソース定義files/ff/fatfs.cfgを追加する
- FatFsの初期化関数を含むため、ストレージデバイスマネージャ:sdev_init()の後に、記載する必要がある

```
#include "device.h"  
#include "spi.h"  
#include "storagedevice.h"  
#include "sd_access.h"
```

```
ATT_INI({TA_NULL, 0, sdev_init});  
INCLUDE("files/ff/fatfs.cfg");
```

ストレージデバイスマネージャの初期化後にFatFsの初期化を行わなくてはならない

```
ATT_INI({ TA_NULL, sw_int, setup_sw_func });  
ATT_INI({ TA_NULL, 0, device_info_init });
```

```
CRE_TSK(MAIN_TASK, { TA_ACT, 0, main_task, MAIN_PRIORITY,  
STACK_SIZE, NULL });
```

演習: sd_access.cの差異

```
hspi = spi_init(SPI1_PORTID, &spi_initd);  
sd_init();
```

コンフィギュレータで初期化するため削除

```
SDeviceHead._get_datetime = get_fat_time;  
sdcard_setspi2(SDCARD_PORTID, hspi, CS2_PORT);  
psdev = SDMGetStorageDevice(SDCARD_DEVNO);  
psdev->_sdev_notice = storagedev_notice;  
for(i = 0 ; i < 50 && hsd == NULL ; i++){  
    hsd = (SDCARD_Handler_t *)psdev->_sdev_local[1];  
    dly_tsk(100);  
}  
if(hsd != NULL){  
    hsd->spi_lock = SPI1LOCK_SEM;  
    while((psdev->_sdev_attribute & SDEV_EMPLOY) == 0){  
        dly_tsk(100);  
    }  
}  
else{  
    syslog_0(LOG_ERROR, "SD-CARD INITIAL ERROR !");  
    slp_tsk();  
}
```

演習: sd_file.cの差異

- セクターのDUMPをやめ、キー入力後、file_testを実行するように修正

```
if(hsd != NULL && hsd->hspi != NULL){
    int wk1, wk2;

    SVC_PERROR(sdcard_getcardinfo(hsd, &SDCardInfo));
    wk1 = SDCardInfo.capacity % 1000000000;
    wk2 = SDCardInfo.capacity / 1000000000;
    syslog_1(LOG_NOTICE, "status[%02x]", SDCardInfo.status);
    syslog_5(LOG_NOTICE, "cardtype(%d) capacity(%d%09d) blocksize(%d) maxsector(%u)",
        SDCardInfo.cardtype, wk2, wk1, SDCardInfo.blocksize, SDCardInfo.maxsector);

    syslog_0(LOG_NOTICE, "## READY test start [dev cup 1(return)] ##");
    while(get_cup_command() == 0){
        dly_tsk(100);
    }
    result = file_test();
    if(result != 0)
        syslog_1(LOG_ERROR, "sd_test error(%d) !", result);
}
```

キー入力を待つ
file_test()を実行

演習 : file_test.c

- “data.txt”を作成し、“01234567890¥r¥n”を書き込むプログラム

```
static const char test_data[] = "1234567890¥r¥n";
int file_test(void)
{
    FILE file;
    UINT cnt;
    int result;
    printf("WRITE START¥n");
    result = f_open(&file, "0:data.txt", FA_WRITE | FA_CREATE_ALWAYS);
    if(result != FR_OK){
        printf("f_open error(%d) !¥n", result);
        return -1;
    }
    result = f_write(&file, (const void *)test_data, sizeof(test_data)-1, &cnt);
    if(result != FR_OK){
        printf("f_write error(%d) !¥n", result);
        f_close(&file);
        return -1;
    }
    result = f_close(&file);
    printf("write count(%d) result(%d)¥n", cnt, result);
    printf("WRITE END¥n");
    return 0;
}
```

“data.txt”を書き込みモードで
オープン

データの書き込み

ファイルのクローズ

演習: sd_file1をビルド実行

- ビルド、ダウンロード、実行する
- 初期化終了後、mon> dev cup 1(return)でテストプログラムを実行させる

```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

Toyohashi Univ. of Technology, JAPAN
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory
Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT

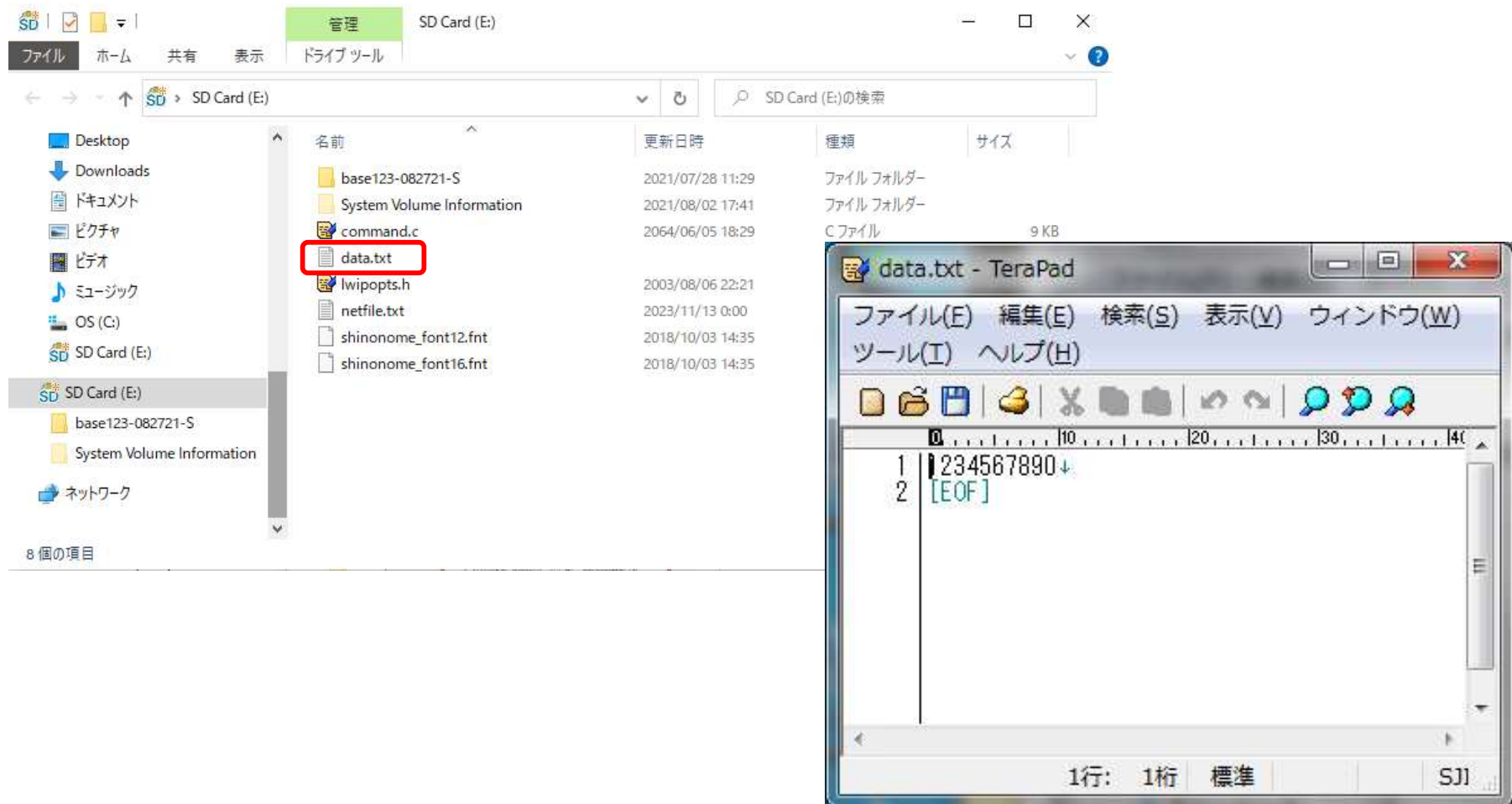
System logging task is started on port 1.

TASK Monitor Release 1.2.0 for Real-Time Kernel (Nov 18 2023, 22:03:27)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>Sample program starts (exinf = 0).
LCD TEST START
## resp 0[3f] 1[00] 2[00] 3[00] ##
## attr[e000] max(3887104) result(0) ##
MEDIA[20022494]no(0) on/off(1) !
status[00]
cardtype(1) capacity(1990197248) blocksize(1024) maxsector(3887104)
## READY test start [dev cup 1(return)] ##

mon>dev cup 1
mon>WRITE START
write count(12) result(0)
WRITE END
TEST END
```

演習：Windowsを使ってファイル確認

- SDカードを、Windowsにマウントし、data.txtが作成され
- 内容が正しいことを確認する



SDカードファイルシステムの構築

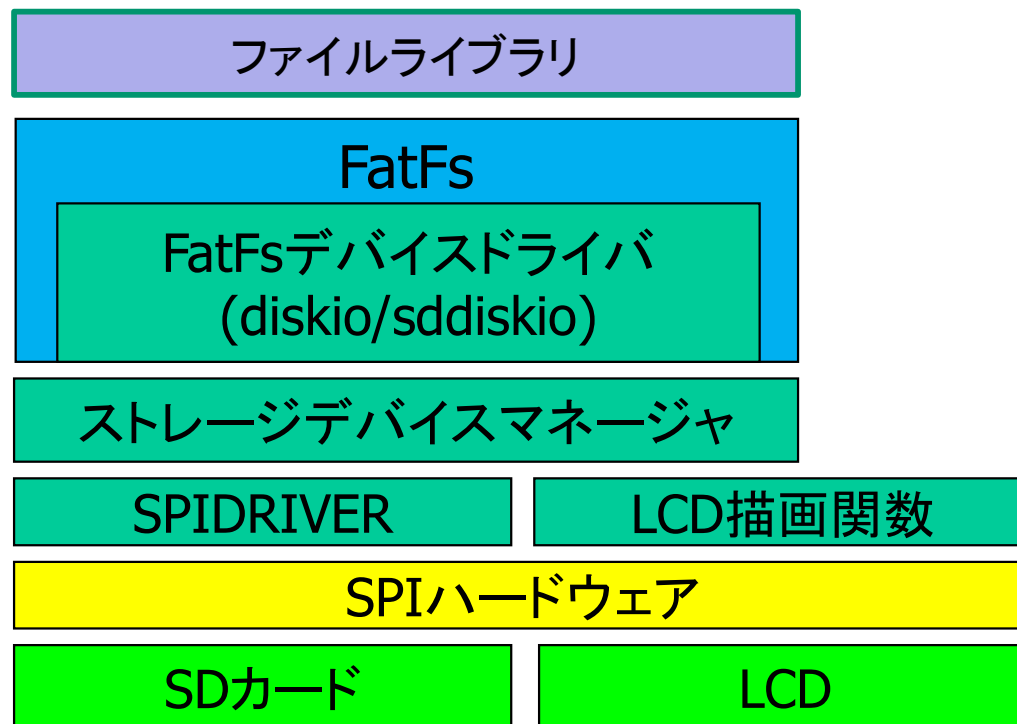
1. FatFs
2. ファイルシステムの対応
3. [ファイルライブラリの対応](#)
4. ファイルユーティリティの対応

ファイルライブラリの対応

- FatFsのファイル関数は独自の仕様である
- LINUXやWindows上でファイル処理プログラムを作成する場合、C言語等の標準的なファイル関数を用いる
- file_test.cが巨大なプログラムの場合、すべて組込み環境でビルドを行い、ST-LINKでFLASH-ROMに書き込みデバッグを行うのはストレスのかかる作業である
- 組込み側でファイルライブラリに対応し、C言語の標準的なファイル関数をサポートすれば、LINUX上で開発、デバッグ検証し、最後に組込み機器上のプログラムに置き換えればよく、大きな開発工数の削減が可能である

ファイルライブラリ

- FatFs上に標準的なC言語ファイル関数をサポートしたファイルライブラリをポーティングする
- TOPPERS BASE PLATFORMのファイルライブラリは files/stdio.c、files/stdfile.cが、これにあたる



C言語ファイル関数: ファイルライブラリ

- プラットフォームのファイルAPIをPOSIX準拠とする
- ファイルライブラリで、よく使用される関数を選択する
 - C言語標準のもの(C89、C99)
 - POSIX.1-2001でよく使われるもの
 - LINUX固有であるが、必要なもの
- C89,C99はISOで定めたC言語標準
 - C89は1989年、C99は1999年の発行でC99は2000年5月にANSIとして受理されている
- アプリケーションからファイルを操作する場合、これらの関数を使用する

C言語標準の関数1

書式	機能	返り値
<code>FILE *fopen(const char *path, const char *mode);</code>	pathで指定された名前のファイルを開きストリームと結びつける	成功するとFILE型のポインタを返す。失敗するとNULL
<code>int fclose(FILE *fp);</code>	fpで指定されるストリームをフラッシュ後、クローズする	正常終了すると0、エラー時EOFを返す
<code>size_t fread(void *ptr, size_t size, size_t nmemb, FILE *fp);</code>	fpで指定されるストリームからnmemb個のデータを読みptrに格納する。個々のデータはsizeバイトの長さ	成功した場合要素の個数、エラーの場合少ない数
<code>size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *fp);</code>	ptrからnmemb個のデータをfpで指定されたストリームに書き込む。個々のデータはsizeバイトの長さ	成功した場合要素の個数、エラーの場合少ない数

C言語の関数2

書式	機能	返り値
<code>int fputc(int c, FILE *fp);</code>	キャラクタcをfpストリームに書き込む	成功した場合c、エラー時EOF
<code>int fputs(const char *s, FILE *fp);</code>	文字列sをfpストリームに書き込む	成功した場合負でない値、エラー時EOF
<code>int putc(int c, FILE *fp);</code>	マクロで実装されている可能性ありという点を除きfputcと同じ	成功した場合c、エラー時EOF
<code>int fgetc(FILE *fp);</code>	fpから次の文字を読み込む、ファイルの終わリエラー時EOF	機能を参照
<code>char *fgets(char *s, int size, FILE *fp);</code>	fpから最大size-1個の文字をsバッファに読み込む	エラー時NULLを返す
<code>int fflush(FILE *fp);</code>	fpストリームのユーザ領域のバッファをフラッシュする	成功時0、エラー時EOFを返す

C言語の関数3

- fpで指定するストリームはファイルを指すとは限らない、標準入出力の場合以下のストリームを使用する
 - stdin 標準入力用のストリーム
 - stdout 標準出力用のストリーム
 - stderr エラー出力用のストリーム
- ストリームを使用しないC言語関数を以下に示す

書式	機能	返回值
int rename(const char *oldpath, const char *newpath);	ファイルの名前を変更し、必要ならばディレクトリ間の移動を行う	成功した場合0、エラーの場合-1を返す
int remove(const char *pathname);	ファイルシステムからファイル名を削除する	成功した場合0、エラーの場合-1を返す

POSIX.1-2001の関数1

書式	機能	返回值
<code>int open(const char *pathname, int flags);</code>	pathnameに対応のファイルディスクリプタを返す	エラーの場合、-1
<code>int close(int fd);</code>	ファイルディスクリプタをクローズする	正常時0、エラー時-1
<code>ssize_t read(int fd, void *buf, int count);</code>	ファイルディスクリプタから最大countバイトをbufに読み込む	読み込みサイズ、0で終端、-1でエラー
<code>ssize_t write(int fd, void *buf, int count);</code>	bufからファイルディスクリプタに最大countバイトを書き込む	書き込みサイズ、-1でエラー
<code>int stat(const char *path, struct stat *buf);</code> <code>int fstat(int fd, struct stat *buf);</code> <code>int lstat(const char *path, struct stat *buf);</code>	ファイルについての情報を返す	成功時0、エラー時-1

POSIX.1-2001の関数2

書式	機能	返り値
<code>int access(const char *pathname, int mode);</code>	pathnameにアクセスできるかをチェック	成功時0、その他は-1
<code>int mkdir(const char *path, mode_t mode);</code>	ディレクトリを作成する	成功時0、その他は-1
<code>int rmdir(const char *pathname);</code>	空のディレクトリを削除する	成功時0、その他は-1
<code>int chmod(const char *path, mode_t mode);</code>	ファイルのアクセス許可を変更する	成功時0、失敗時は-1
<code>int opendir(const char *name);</code>	nameに対応したディレクトリストリームをオープンし、ストリームを返す	成功時ストリームのポインタ、エラー時NULL
<code>int closedir(dir *dirp);</code>	ディレクトリストリームをクローズする	成功時0、失敗時は-1

LINUX固有の関数

- LINUX固有であるが、ディレクトリ表示等に必要なため追加した関数は以下のとおり

書式	機能	返り値
<code>int readdir(unsigned int fd, struct dirent *dirp, unsigned int count);</code>	ファイルディスクリプタfdが参照しているディレクトリからdirent構造体を読み、dirpで指されたバッファに格納する	成功すれば1、エラーならば-1を返す
<code>int statfs(const char *path, struct statfs *buf);</code>	マウントされたファイルシステムについての情報を返す	成功すれば0、エラーならば-1

演習：パソコン上でファイルテスト

- base3/filetest1に標準C言語ファイル関数を使ったfile_test.cを用意している
- filetest1をmy_programにコピーし、my_program/file_test1ディレクトリに移動
- このプログラムはmain関数をもつfilecmd.cから呼び出されパソコン上で実行可能である
- filecmdをビルド、実行しdata.txtができることを確認する

```
$ ../filetest1  
$ make  
$ ./filecmd
```

演習: filecmdの実行確認

- 実行すると、WRITE START/WRITE ENDと書きこんだdata.txtが表示される
- data.txtがワーク領域に作成される

```
~/toppers/base3/my_program/filetest1
rm -f *.o core filecmd.exe

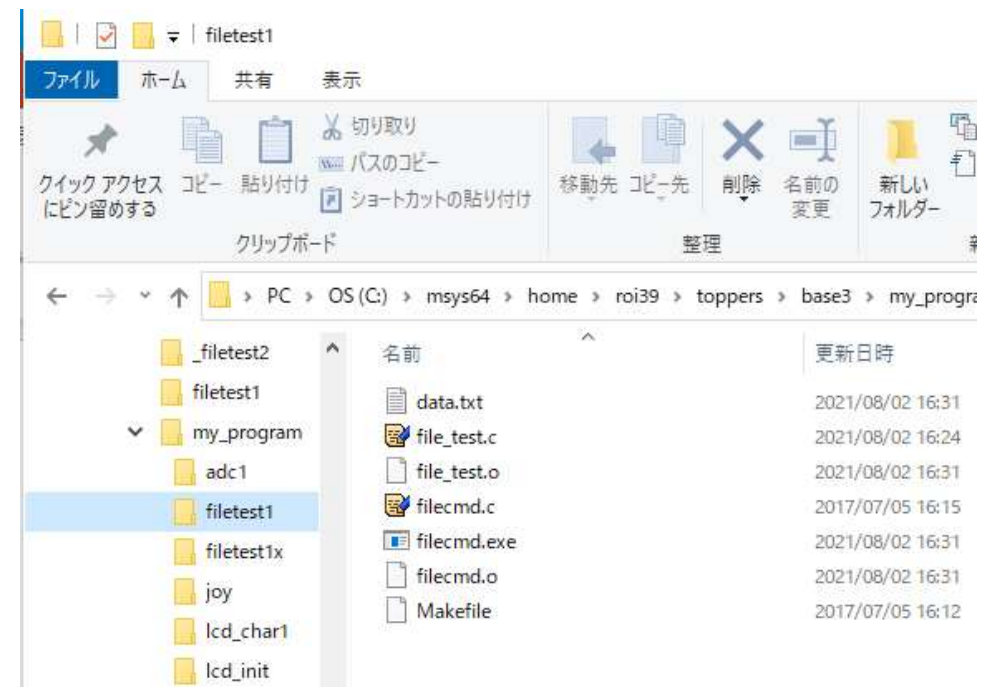
roi39@DESKTOP-JMDVGP MINGW64 ~/toppers/base3/filetest1
$ ls
Makefile data.txt file_test.c filecmd.c

roi39@DESKTOP-JMDVGP MINGW64 ~/toppers/base3/filetest1
$ cd ../my_program/filetest1

roi39@DESKTOP-JMDVGP MINGW64 ~/toppers/base3/my_program/filetest1
$ make
gcc -O3 -c -o filecmd.o filecmd.c
gcc -O3 -c -I. -o file_test.o file_test.c
gcc -O3 filecmd.c -I. -o filecmd file_test.o

roi39@DESKTOP-JMDVGP MINGW64 ~/toppers/base3/my_program/filetest1
$ ./filecmd
WRITE START
WRITE END
file:data.txt
1234567890

roi39@DESKTOP-JMDVGP MINGW64 ~/toppers/base3/my_program/filetest1
$
```



演習: sd_file2 ファイルライブラリ対応

- ファイルライブラリを取り込み、パソコンで検証したfile_test.cを実行させる
- 学習内容
 - ファイルライブラリを使えば、UNIX環境で実行確認したプログラムが組み込み機器でも使用可能なことを確認
- 手順
 - sd_file1をコピーしてsd_file2を作る
 - sd_file1中のFatFs用のfile_test.cをパソコンで実行確認したfile_test.cに置き換える
 - ファイルライブラリの取り込みを行う
- 仕様
 - パソコンで検証したfile_test.cが正しく動作する
 - 使用するファイル関数は
 - fopen、fwrite、fclose

演習 : Makefileの修正

- システムサービスにファイルライブラリの部品を取り込む
 - `stdio.o`
 - `stdfile.o`

```
# ミドルウェア Makefile のインクルード
#
include $(SRCDIR)/files/ff/Makefile.config
include $(SRCDIR)/monitor/Makefile.config

#
# ストレージファイルマネージャー
#
INCLUDES := $(INCLUDES) -I$(SRCDIR)/files -I$(SRCDIR)/files/ff
SYSSVC_DIR := $(SYSSVC_DIR) $(SRCDIR)/files $(SRCDIR)/files/ff
SYSSVC_COBJS := $(SYSSVC_COBJS) storagedevice.o stdio.o stdfile.o
```

演習: コンフィギュレーションファイルの修正

- sd_file.cfgに、ストレージデバイスマネージャの初期化の後にファイルライブラリの初期化関数実行を行う
- ファイルライブラリで使用するセマフォを追加

```
INCLUDE("monitor/monitor.cfg");

#include "device.h"
#include "spi.h"
#include "storagedevice.h"
#include "sd_access.h"

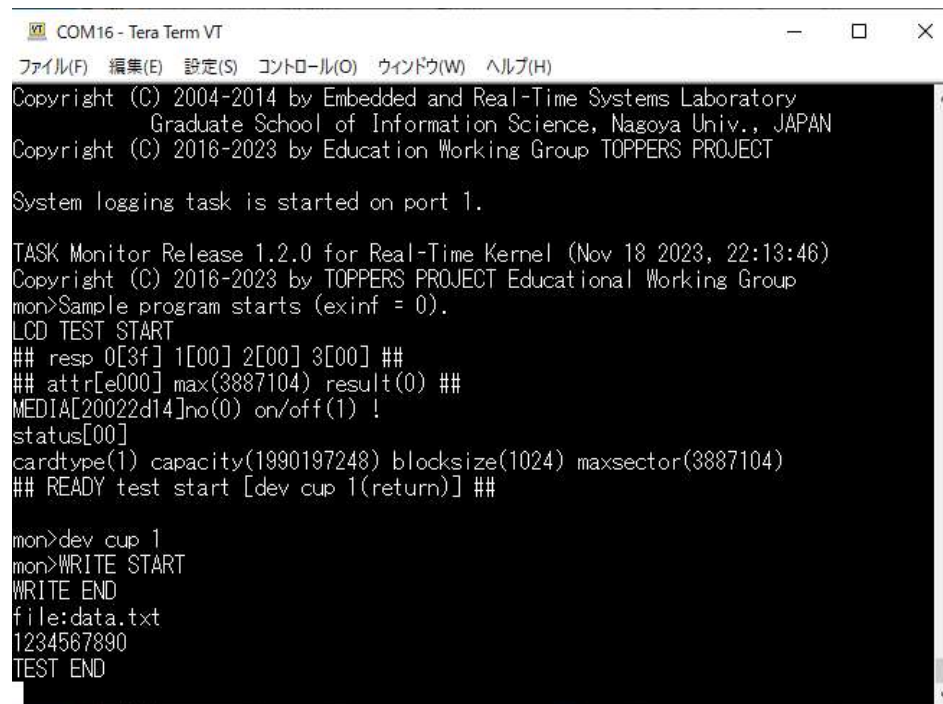
ATT_INI({TA_NULL, 0, sdev_init});
ATT_INI({TA_NULL, 0, stdfile_init});
INCLUDE("files/ff/fatfs.cfg");

INCLUDE("files/ff/fatfs.cfg");
ATT_INI({ TA_NULL, 0, device_info_init });
ATT_INI({ TA_NULL, sw_handle, setup_sw_func });

CRE_SEM(SEM_STDFILE, { TA_NULL, 1, 1 });
```

演習: Windowsを使ってファイル確認

- FatFsの場合と同様に、SDカードをWindowsにマウントし、data.txtが作成され
- 内容が正しいことを確認する

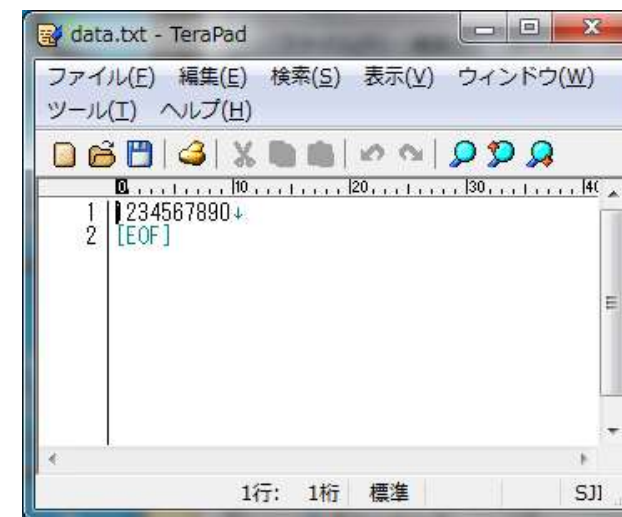
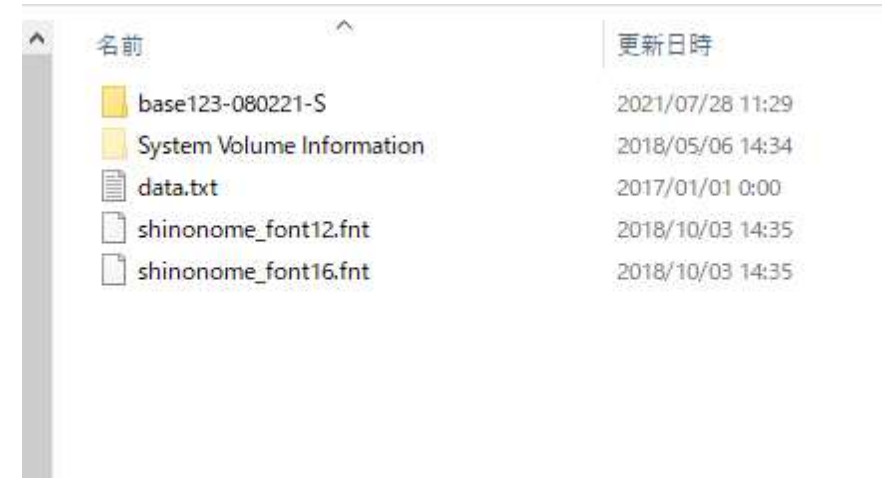


```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
Copyright (C) 2004-2014 by Embedded and Real-Time Systems Laboratory
Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2016-2023 by Education Working Group TOPPERS PROJECT

System logging task is started on port 1.

TASK Monitor Release 1.2.0 for Real-Time Kernel (Nov 18 2023, 22:13:46)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>Sample program starts (exinf = 0).
LCD TEST START
## resp 0[3f] 1[00] 2[00] 3[00] ##
## attr[e000] max(3887104) result(0) ##
MEDIA[20022d14]no(0) on/off(1) !
status[00]
cardtype(1) capacity(1990197248) blocksize(1024) maxsector(3887104)
## READY test start [dev cup 1(return)] ##

mon>dev cup 1
mon>WRITE START
WRITE END
file:data.txt
1234567890
TEST END
```



SDカードファイルシステムの構築

1. FatFs
2. ファイルシステムの対応
3. ファイルライブラリの対応
4. ファイルユーティリティの対応

ボリュームコマンド

- ボリュームコマンドは、ファイル管理系のタスクモニタ拡張コマンドである
- カテゴリ: volumeで追加される
- プログラムはfiles/diskutils.cにある

第二コマンド	引数	機能	
DIR	デバイスパス	デバイスパスのファイルを表示する	
MKDIR	ファイルパス	ファイルパスのディレクトリを作成する	
RMDIR	ファイルパス	ファイルパスのディレクトリを削除する	
ERASE	ファイルパス	ファイルパスのファイルを削除する	注1

演習: ボリュームコマンド対応

- ボリュームコマンドコマンドを追加し、作成したdata.txtを表示する
- 学習内容
 - デバッグ・コマンドの拡張で開発効率の向上を確認する
- 手順
 - sd_file2をコピーしてsd_file3を作り、修正する
 - files/Makefile.configを使って拡張を行う

演習 : Makefileの修正

- \$(SRCDIR)/files/Makefile.configをミドルウェアのインクルードに追加する
- 不要となったストレージデバイスマネージャ用にシステムサービス設定を削除する

```
# ミドルウェア Makefile のインクルード
#
include $(SRCDIR)/files/ff/Makefile.config
include $(SRCDIR)/files/Makefile.config
include $(SRCDIR)/monitor/Makefile.config

#
# GDIC Makefile のインクルード
#
include $(SRCDIR)/gdic/spi_driver/Makefile.config
ifeq ($(BOARDTYPE),PICO_W)
include $(SRCDIR)/gdic/cyw43/Makefile.config
endif
```

「ストレージファイルマネージャ」の記述はfiles/Makefile.configに含まれる

```
#
# システムサービスに関する定義
```

演習: コンフィギュレーションファイルの修正

- files/storagedevice.cfgをfiles/ff/fatfs.cfgの前に追加する
- 不要な初期化やセマフォやタスクを削除する

```
INCLUDE("gdic/daytime/dtime.cfg");  
#endif  
INCLUDE("files/storagedevice.cfg");  
INCLUDE("monitor/monitor.cfg");
```

```
#include "device.h"  
#include "spi.h"  
#include "storagedevice.h"  
#include "sd_access.h"
```

```
ATT_INI({ TA_NULL, 0, sdev_init });  
ATT_INI({ TA_NULL, 0, stdfile_init });  
INCLUDE("files/ff/fatfs.cfg");  
ATT_INI({ TA_NULL, sw_int, setup_sw_func });  
ATT_INI({ TA_NULL, 0, device_info_init });
```

```
CRE_SEM(SEM_STDFILE, { TA_NULL, 1, 1 });
```

```
CRE_TSK(MAIN_TASK, { TA_ACT, 0, main_task, MAIN_PRIORITY, STACK_SIZE, NULL });  
CRE_TSK(SDM_TASK, { TA_ACT, 1, SDMSence_task, 15, 1024, NULL });
```

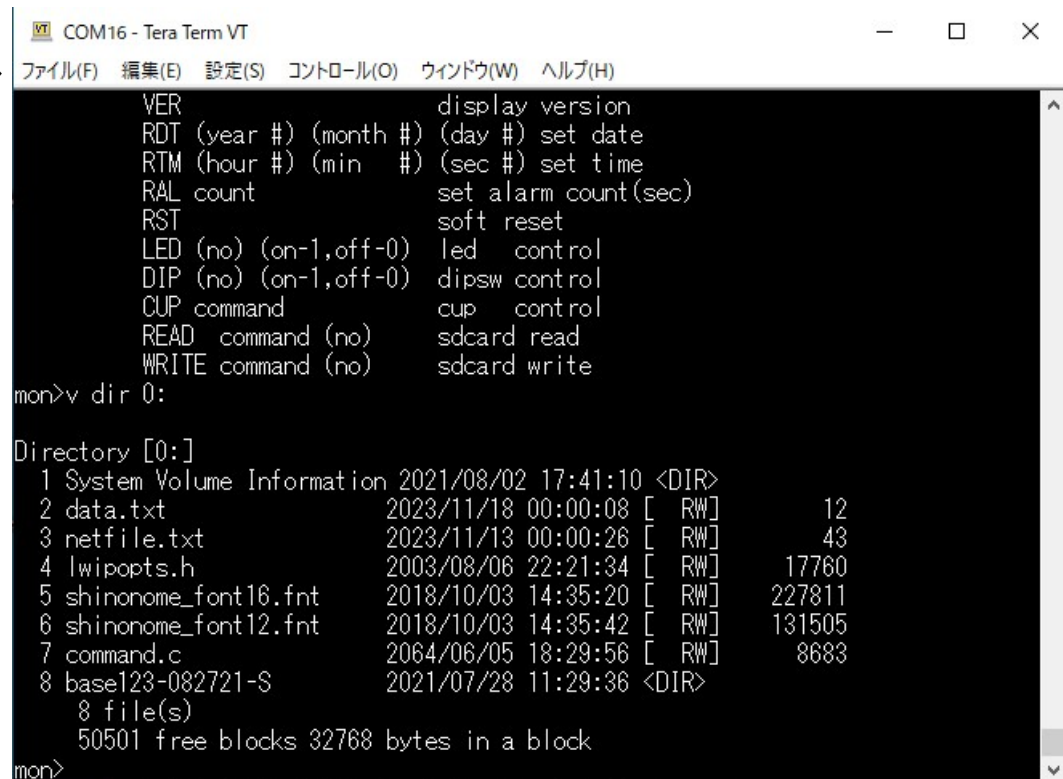
以下の記述はstoragedevice.cfgに記載されているので削除

```
ATT_INI({TA_NULL, 0, sdev_init});  
ATT_INI({TA_NULL, 0, stdfile_init});  
CRE_SEM(SEM_STDFILE, { TA_NULL, 1, 1 });  
CRE_TSK(SDM_TASK, { TA_ACT, 1, ... NULL });
```

演習: ボリューム用のコマンドを実行

- FLASH-ROMに書き込み、実行しvolumeコマンドでディレクトリ表示すると”data.txt”が表示される
 - mon> volume dir 0: (return)

SDカードの容量が大きい場合、全ての表示に時間がかかるので注意(ルートディレクトリのサイズが大きいため)



```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
VER                display version
RDT (year #) (month #) (day #) set date
RTM (hour #) (min #) (sec #) set time
RAL count          set alarm count(sec)
RST                soft reset
LED (no) (on-1,off-0) led control
DIP (no) (on-1,off-0) dipsw control
CUP command        cup control
READ command (no)  sdcard read
WRITE command (no) sdcard write
mon>v dir 0:
Directory [0:]
1 System Volume Information 2021/08/02 17:41:10 <DIR>
2 data.txt                  2023/11/18 00:00:08 [ RW]      12
3 netfile.txt               2023/11/13 00:00:26 [ RW]      43
4 lwipopts.h                2003/08/06 22:21:34 [ RW]    17760
5 shinonome_font16.fnt      2018/10/03 14:35:20 [ RW]   227811
6 shinonome_font12.fnt      2018/10/03 14:35:42 [ RW]   131505
7 command.c                 2064/06/05 18:29:56 [ RW]    8683
8 base123-082721-S          2021/07/28 11:29:36 <DIR>
8 file(s)
50501 free blocks 32768 bytes in a block
mon>
```

ファイルテストプログラム

演習: FILE VERIFYを行う

- sd_file3のfile_testをREAD VERIFYを行うように修正
- 学習内容
 - ファイルライブラリにより、UNIX環境で組込み環境と同等のプログラム検証が行えることを確認する
- 手順
 - filetest1/file_test.cをFILE VERIFYするプログラムに書き換え
 - Bshell上でビルドしてVERIFY動作を確認
 - 修正したfile_test.cをsd_file3にコピーし、ビルドを行い
 - 組込み環境でもFILE VERIFYできることを確認

演習 : file_test.cをFILE VERIFYするよう書き換え

- file_test.cをwrite後、readとverifyするようにプログラム修正
- Bshell環境で実行確認

```
printf("VERIFY START¥n");
pfile = fopen("data.txt", "rb");
if(pfile == NULL){
    printf("fopen error !¥n");
    return -1;
}
while((c = fgetc(pfile)) != EOF){
    read_buffer[fsize++] = c;
}
fclose(pfile);
printf("READ END size(%d)¥n", fsize);
if(fsize != (sizeof(test_data)-1)){
    printf("file size error original size(%d) file size(%d) !¥n", (sizeof(test_data)-1), fsize);
    return -1;
}
for(i = 0 ; i < fsize ; i++){
    if(test_data[i] != read_buffer[i])
        printf("verify error (%d) [%02x][%02x] !¥n", i, test_data[i], read_buffer[i]);
}
return 0;
```

Verify Programの例

c はsigned charにすること

演習 : file_test.cをFILE VERIFYするよう書き換え

- file_test.cをwrite後、readとverifyするようにプログラム修正
- Bshell環境で実行確認

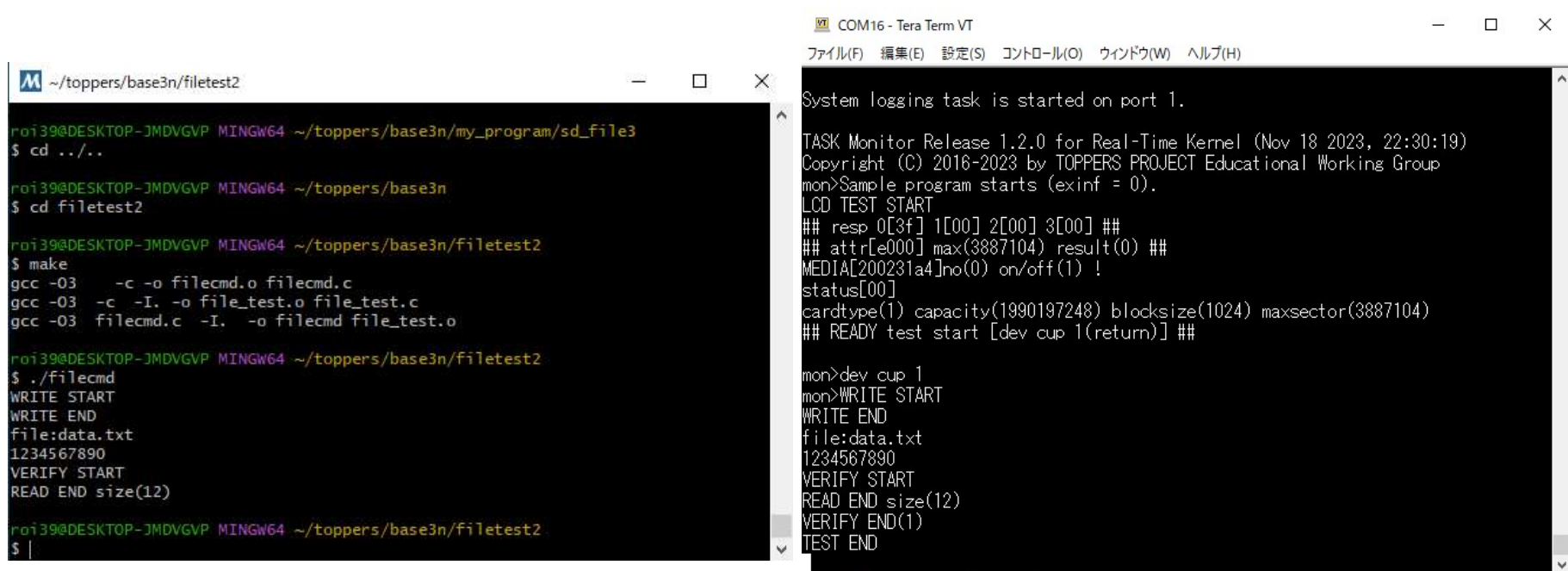
```
printf("VERIFY START\n");
pfile = fopen("data.txt", "rb");
if(pfile == NULL){
    printf("fopen error !\n");
    return -1;
}
while((c = fgetc(pfile)) != EOF){
    read_buffer[fsize++] = c;
}
fclose(pfile);
printf("READ END size(%d)\n", fsize);
if(fsize != (sizeof(test_data)-1)){
    printf("file size error original size(%d) file size(%d) !\n", (sizeof(test_data)-1), fsize);
    return -1;
}
for(i = 0 ; i < fsize ; i++){
    if(test_data[i] != read_buffer[i])
        printf("verify error (%d) [%02x][%02x] !\n", i, test_data[i], read_buffer[i]);
}
return 0;
```

Verify Programの例

c はsigned charにすること

演習: SDカードでもverifyできることを確認

- Bshell上で正しくプログラムが動作することを確認する
- 作成したfile_test.cをsd_file3にコピーし、ビルド、ダウンロード、実行する
- Bshellと表示が同じであることを確認する



```
~/toppers/base3n/filetest2
roi39@DESKTOP-JMDVGV MINGW64 ~/toppers/base3n/my_program/sd_file3
$ cd ../../

roi39@DESKTOP-JMDVGV MINGW64 ~/toppers/base3n
$ cd filetest2

roi39@DESKTOP-JMDVGV MINGW64 ~/toppers/base3n/filetest2
$ make
gcc -O3 -c -o filecmd.o filecmd.c
gcc -O3 -c -I. -o file_test.o file_test.c
gcc -O3 filecmd.c -I. -o filecmd file_test.o

roi39@DESKTOP-JMDVGV MINGW64 ~/toppers/base3n/filetest2
$ ./filecmd
WRITE START
WRITE END
file:data.txt
1234567890
VERIFY START
READ END size(12)

roi39@DESKTOP-JMDVGV MINGW64 ~/toppers/base3n/filetest2
$ |
```

```
COM16 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

System logging task is started on port 1.

TASK Monitor Release 1.2.0 for Real-Time Kernel (Nov 18 2023, 22:30:19)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>Sample program starts (exinf = 0).
LCD TEST START
## resp 0[3f] 1[00] 2[00] 3[00] ##
## attr[e000] max(3887104) result(0) ##
MEDIA[200231a4]no(0) on/off(1) !
status[00]
cardtype(1) capacity(1990197248) blocksize(1024) maxsector(3887104)
## READY test start [dev cup 1(return)] ##

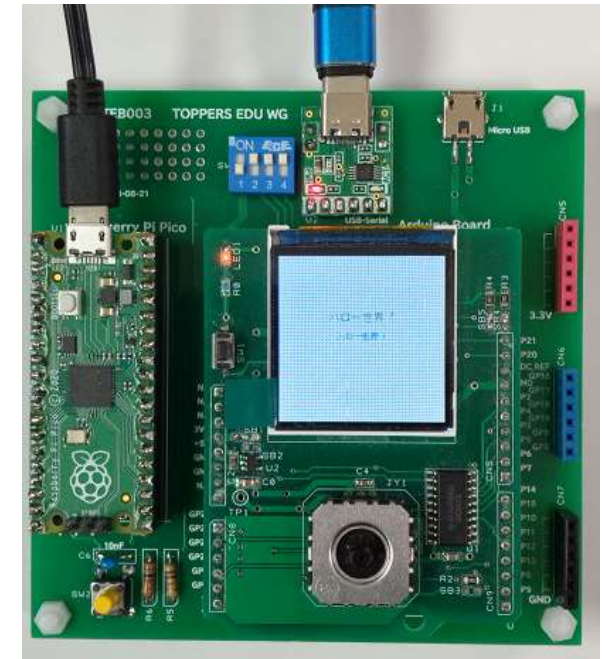
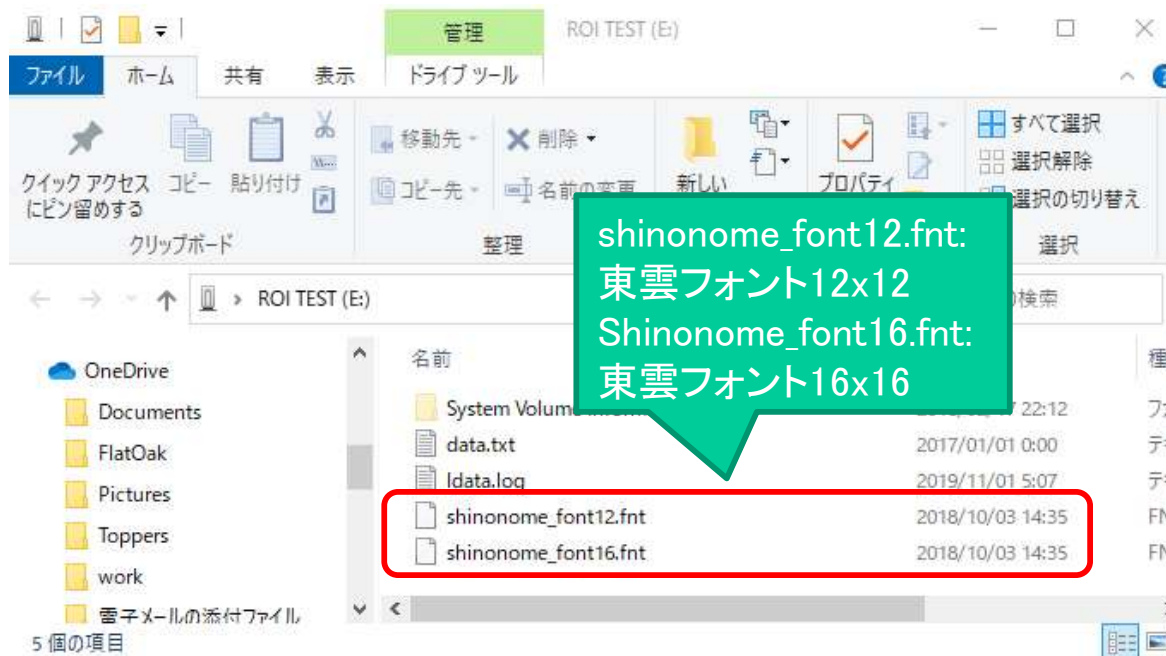
mon>dev cup 1
mon>WRITE START
WRITE END
file:data.txt
1234567890
VERIFY START
READ END size(12)
VERIFY END(1)
TEST END
```

ファイルシステムの有用性

- STM32F746/769 Discoveryを使ったMedia PlayerではLinux用のJPEGライブラリ(jpeg-9b)、SP3伸長ライブラリ(libmad-0.15)を使用している
- 両方とのファイル変換用のライブラリであるが、ファイルシステムのLinuxとの互換性が高いため、ソースコードは1行も修正せずTOPPERS BASE PLATFORM上で動作している
 - 但し、ビルド環境の差異でMakefileとコンフィギュレーション用のインクルードファイルのみ修正している

LCD表示の漢字対応

- 1日目のUIミドルウェア対応を漢字に対応する
- 漢字フォントをSDカードに置いているためファイル対応が必要となる
- lcd_text3とlcd_text4をdiffコマンドで比較すれば、ファイルシステム対応が明確となる

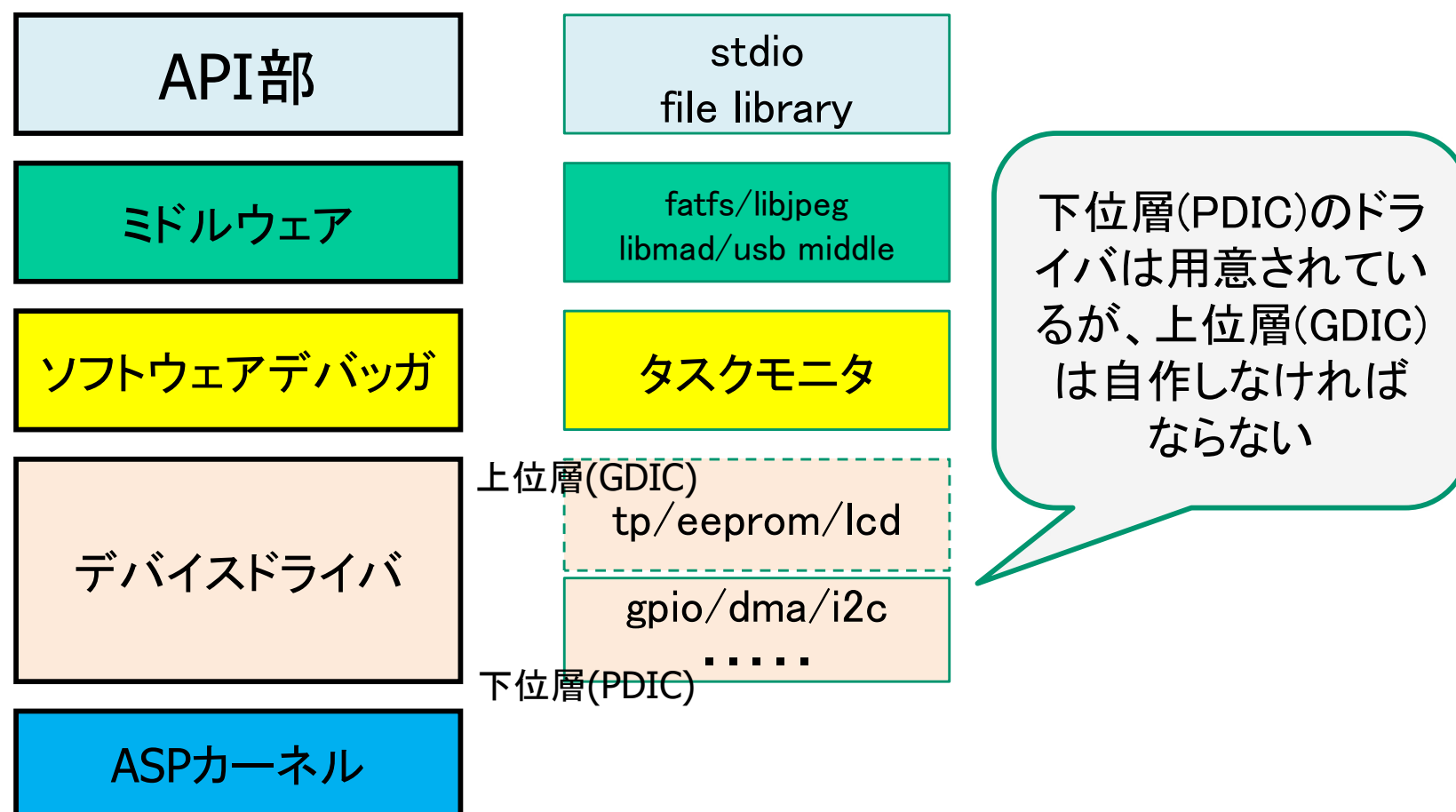


DICアーキテクチャ

1. 組込みプラットフォーム
2. DICアーキテクチャ

組み込みプラットフォームを構成する部品

- 組み込みプラットフォームは単純な構成では以下の5つの構成からなる
- BASE PLATFORMとの比較図を以下に示す



DICアーキテクチャで定義するプラットフォーム

- DICアーキテクチャでは、デバイスドライバに対する標準的なモデルを提示し、ガイドラインに従って作成することによって、デバイスドライバの流通を目指した
 - 実際は、ハードウェアIPが標準化されない限り、デバイスドライバも多様化する
- 以下のような、組込みプラットフォームを想定する
- DICアーキテクチャの構成自体は実質に即しており、BASE PLATFORMのモデルとして使用した



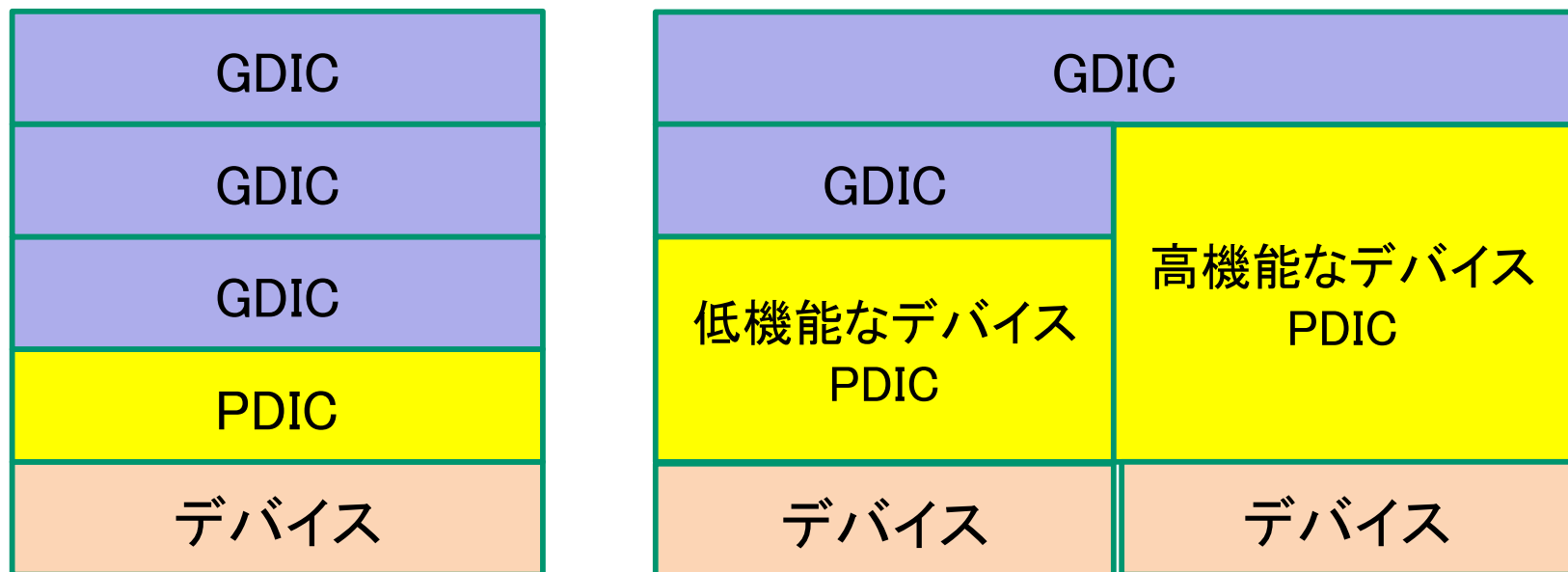
図1. デバイスドライバとDIC

DICアーキテクチャ

1. 組込みプラットフォーム
2. DICアーキテクチャ

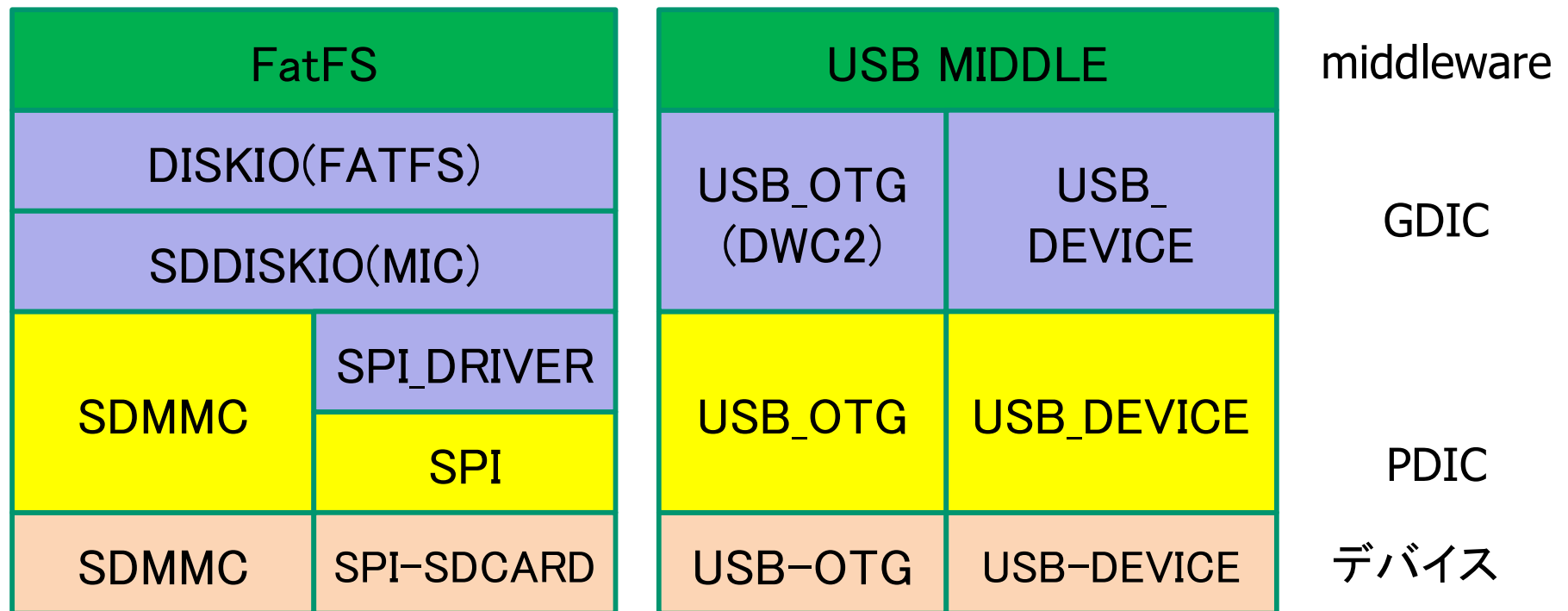
DICの階層モデル

- PDIC: Primitive DIC
 - デバイスに対して最低限のインターフェイスを提供
 - UART/I2C/SPIはデバイスに対するI/Fを提供
- GDIC: General DIC
 - PDICに対して機能を追加するモジュール



TOPPERS BASE PLATFORMのデバイス実装

- ファイルシステムとUSB MIDDLEWAREはデバイスドライバがDIC構造であるために、異なったデバイスを問題なく共有することができる

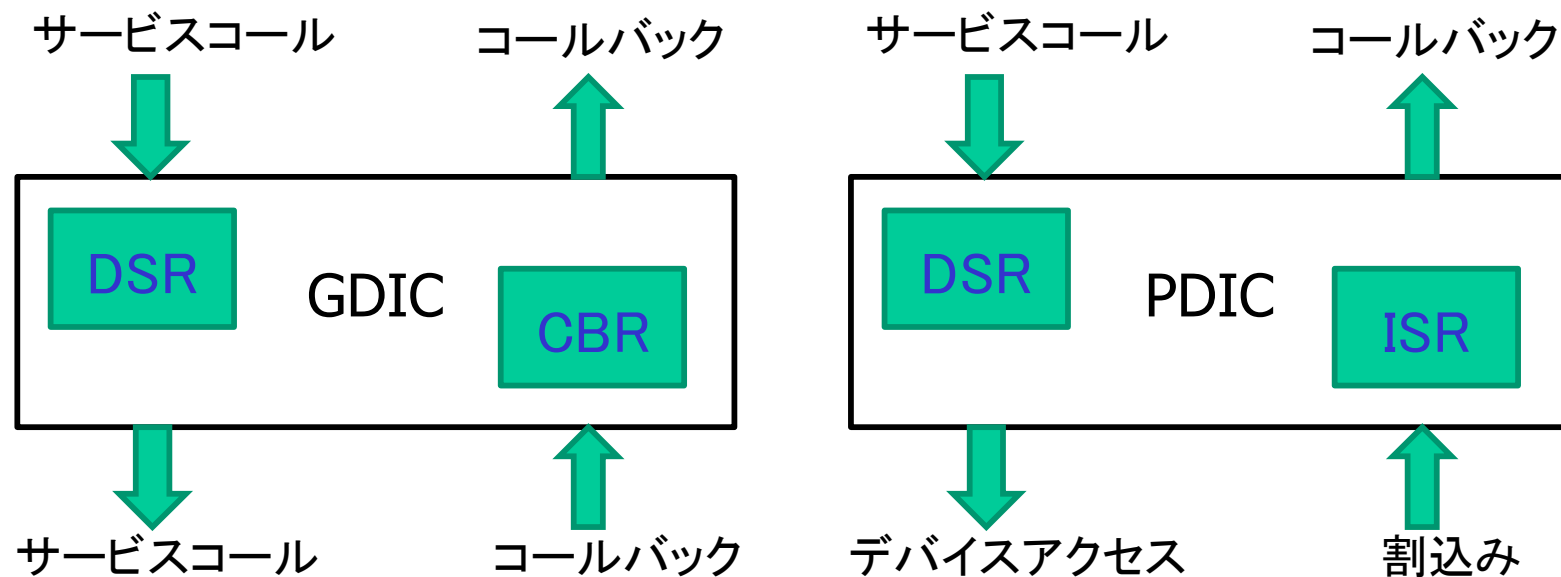


DICのAPIの枠組み

- DICのAPI(ITRON仕様に準ずる)
 - サービルコール
 - コールバック
 - 静的API
 - パラメータとリターンパラメータ
 - データ型
 - 定数
 - マクロ
 - ヘッダファイル
- PDICでは処理の流れを3つ規定してる
 - 即時完了型サービス(ポーリング)
 - 同期型サービス
 - 非同期型サービス(処理を待たずサービスコールを返す)

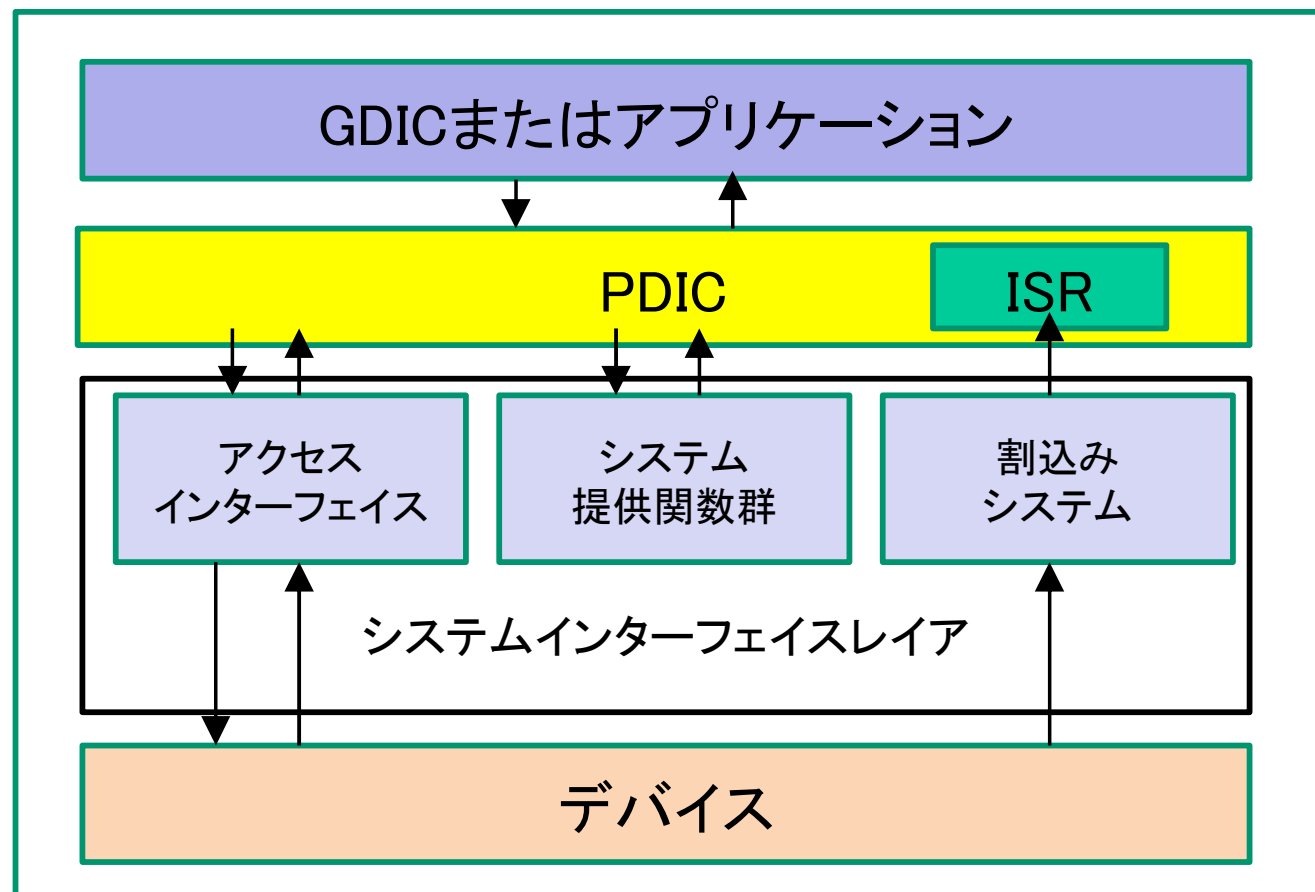
DIC内のサービスルーチン

- DIC内のルーチン
 - DSR: デバイスサービスルーチン
 - サービスコールを実現する
 - CBR: コールバックルーチン
 - コールバックを実現する
 - ISR: 割り込みサービスルーチン
 - 割り込み処理を行う



仮想アーキテクチャ

- DICで定義するデバイスドライバの仮想アーキテクチャ
 - システムインターフェイスレイア (SIL) の部分は TOPPERSカーネルで提供



TOPPERS BASE PLATFORMの実装

- デバイスドライバでは、非同期処理は行わない
 - 処理が煩雑となる
 - リアルタイムカーネルを使用するので、不要な待ちは発生しない
 - 上位層のミドルウェアで非同期処理を使用する
- 複合ドライバのI/FレベルはPDIC記述する
 - PDICにタイムアウト付のリアルタイムカーネルの機能を用いる
- GDICには、コールバック関数を持ち込まない
 - 処理が複雑になる
 - リアルタイムカーネルを用いるため不要
- SILはカーネルのものを使用

詳細なドキュメント

- TOPPERS BASE PLATFORMの詳細なドキュメントとして「Reference Manual」がWebで公開しています。
- SPI/ADCドライバやファイルシステムについて、深く理解するために「Reference Manual」を活用してください
 - <https://www.toppers.jp/edu-baseplatform.html>

LCDシールドを使った アプリ紹介

1. カップラーメンタイマ

カップラーメンタイマ (Raspberry Pi Pico(W)版) の仕様

- 概要

- カップラーメンの完成を知らせるタイマ
- Joy Stickで操作を行う
- お知らせ時間の延長(+30秒)、短縮(-5秒)
- タイマをスタートしてから、設定したお知らせ時間が
- 経過したこと(タイマ終了)を知らせる

- Joy Stick

- 左 : タイマを起動(プッシュスイッチ)
- 左または右: タイマを停止(プッシュスイッチ)
- 上 : タイマ起動中設定時間を30秒延長
- 下 : タイマ起動中設定時間を5秒短縮
- 上 : タイマ停止中のLCDの表示を経過時間表示に切替
- 下 : タイマ停止中のLCDの表示をステータス表示に切替

作成方法

基礎3、1日目の演習内容を参考にして作成する

- 文字列描画
- 時計盤の表示(円、直線の描画)
- Joy Stickの状態監視と処理
- 周期ハンドラを使用して、1秒の基本時間を作成
- タイマプロセスの状態遷移判定と各状態毎の処理
 - 起動からの経過時間(current_time)をカウントする
 - タイマ(timer)の設定(初期設定、+、-)
 - 計時中の経過時間(count_time)をカウントする

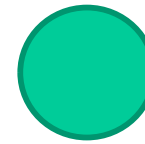
タイマ停止状態

－ 状態遷移の監視

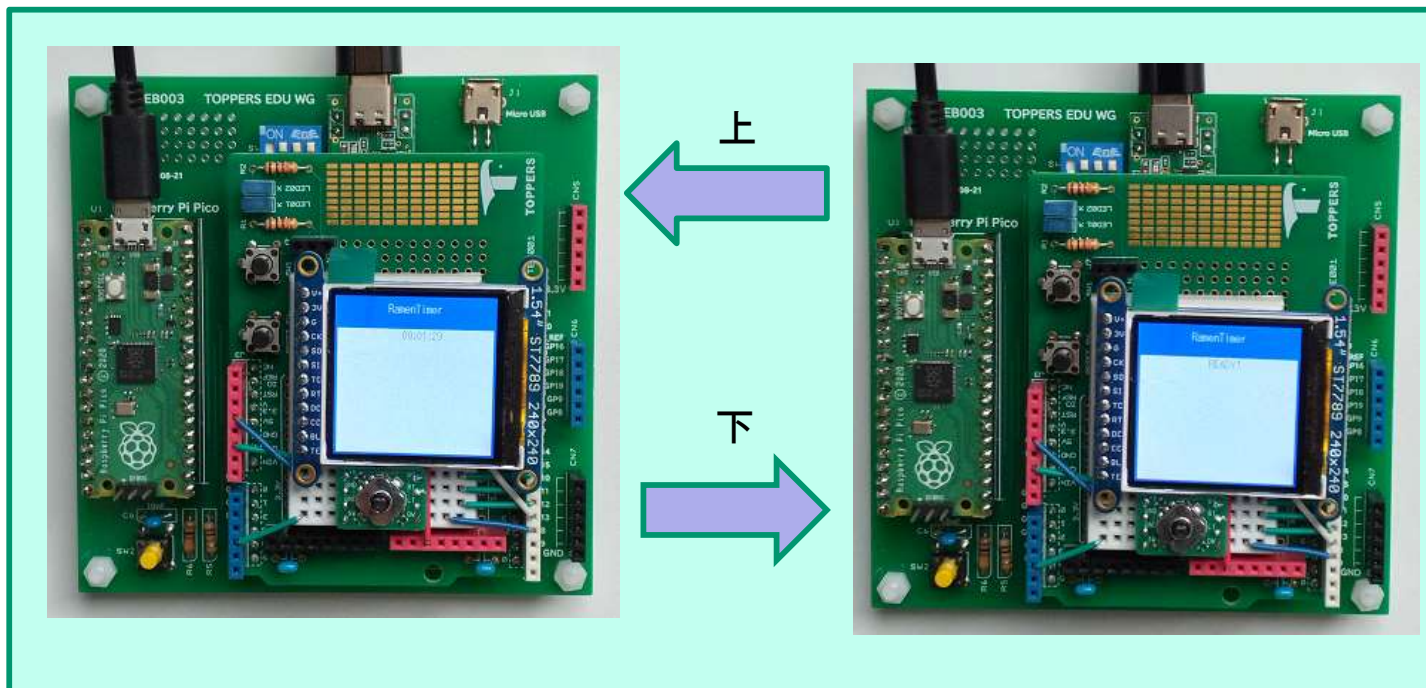
- イベントフラグの通知を監視する
 - BASE_TIME current_timeを+1
 - 右: プッシュスイッチ 計時中モードに遷移
 - 上 表示モードに遷移
 - 下 停止モードに遷移

カップラーメンタイマの仕様

- LCDの表示 (タイマ停止中)

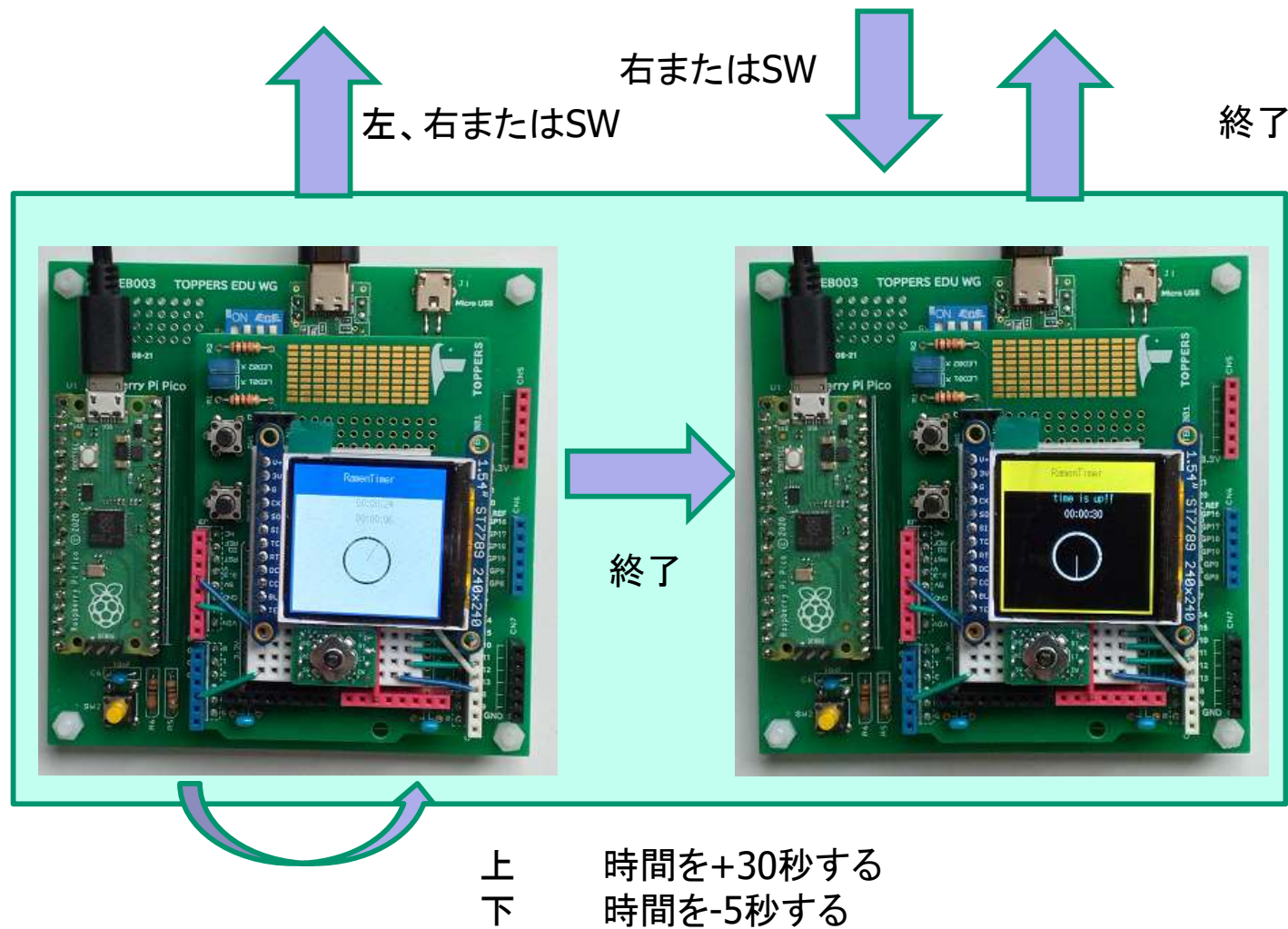


起動前



カップラーメンタイマの仕様

- LCDの表示（タイマ動作中）



カップラーメンタイマのプログラムファイル

base3/proguram/CupRamenTimer

CupRamenTimer.c
CupRamenTimer.h
CupRamenTimer.cfg
Makefile

基礎3の1日目の実習と同じく、base3/my_programにCupRamenTimerをコピーしてビルド、実行する。

make BOARDNO=1のビルドで、Pico2:Cortex-M33
make BOARDNO=2のビルドで、Pico2:RISC-V
をビルドし、検証が行える

まとめ

基礎3講座2日目のまとめ

- SD-CARD:GDICドライバについての学習
- ファイルシステム、ファイルライブラリについての学習
 - ファイルライブラリを使ったシミュレーション環境
- 組み込みプラットフォームとデバイスドライバの構成(DICアーキテクチャ)についての学習
- 組み込みプラットフォームを使ったアプリケーション開発についての学習
 - GDICドライバの追加で、機能追加が可能

2日目 : my_program

- my_programには、以下のディレクトリができ正しくプログラムが実行されているはずです

実習済プログラム

adc2	: 割込みで可変抵抗値を表示するもの
joy2	: Joy stickの位置を表示するもの
lcd_init	: ライン描画するように改造したもの
lcd_char1	: 文字表示するもの
led_text1	: テキスト表示するもの
led_text3	: UIミドルウェアに対応して”Hello world !”を表示する
sd_ini2	: SDカード情報の読み出しができる
sd_ini3	: SDカードアクセスをドライバー化
sd_cmd1	: SDカードセクターダンプコマンド
sd_cmd2	: SDカードアクセスを関数テーブル化
sd_file2	: FATFS及びファイルシステム対応
sd_file3	: ファイルライブラリ、ファイルベリファイ対応

参考文献

- TOPPERS BASE PLATFORM(RP) REFERENCE MANUAL
- Lecture 12:SPI and SD cards University at Buffalo
- μ ITRON4.0 仕様研究会 デバイスドライバ設計ガイドラインWG 中間報告書

著者リスト

- SDカードSPIインターフェイス
- SPIを使ってSDカードアクセス
- SDカードファイルシステムに構築
- ファイルテストプログラム
- DICアーキテクチャ
 - 竹内 良輔 (TOPPERSプロジェクト教育WG主査)
- Adafruitシールドを使ったアプリ紹介
 - 森田浩 (Advanced Data Controls, Corp.)