TOPPERS基礎実装セミナー

（ Raspberry Pi Pico(W)/Pico2(W)版：基本1.5）

基礎編：1日目

TOPPERSプロジェクト

教育ワーキング・グループ

TOPPERS

本ドキュメントに関して

1. 著作権に関しての表記

<TOPPERS基礎実装セミナー(Raspberry Pi Pico(W)/Pico2(W :基本)1日目>

Copyright (C) 2007-2025 by 竹内良輔 TOPPERS教育WG

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。

TOPPERS

スケジュール

- １日目

1. はじめに	0.2時間
2. 開発環境の説明	1.5時間
3. LCDJOYシールドを動かす	2.0時間
4. スティック・マウスを作る	1.5時間
5. ベアメタル組込み実装の勘所	0.5時間
6. まとめ	0.2時間

はじめに

規模からみた組込みシステムの分類

- 組込み機器の規模から3つの組込みシステムに分類する

  （1）小規模組込みシステム

  ワンチップCPUで制御を行う小規模な組込みシステム、ROMやRAMを拡張することもある。RTOSを用いないシステムが多い。ベアメタル開発と呼ぶ。

  （2）中規模組込みシステム

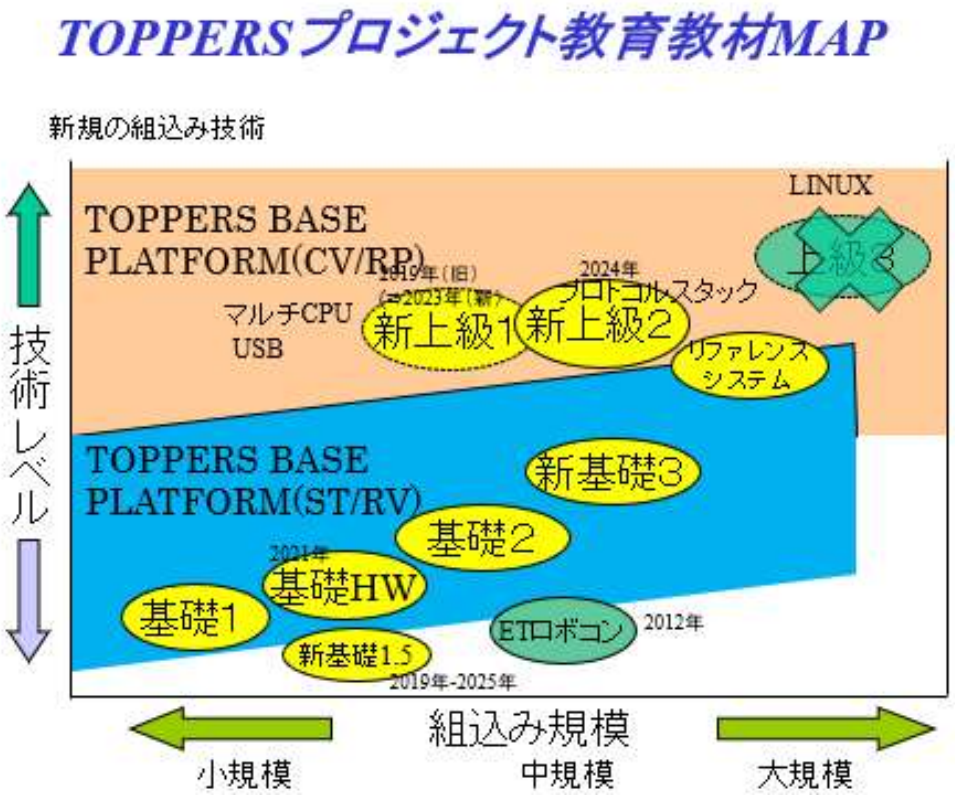
  32ビットクラスのCPUを使うシステム、ソフトウェア規模も大きくRTOSを用いてシステム的な制御を行う

  （3）大規模組込みシステム

  パソコンに近い規模の組込みシステム、実際にパソコンやワークステーションを制御用に用いる場合もある。

基礎1.5コースの概要

- TOPPERSプロジェクト教育コンテンツでは、基礎1,1.5、基礎HWがベアメタル開発セミナーとなる
- 本講座はTOPPERS BASE PLOTFORMのドライバを使用するため、基礎1，３の受講後に受講してください



基礎1.5コースの概要

- TOPPERS実装セミナー基礎編（基礎１）では、ベアメタル開発で基本的なIP（GPIO,TIMER,UART）等の開発実習を行った
- 本セミナーでは基礎３で実習したLCDJOYシールドやUSBデバイスをベーシック(pdic)やジェネラル(gdic)なドライバを用いて、ベアメタル環境での実習体験を行うコンテンツとなっている
- 基本の開発はPico、Pico2を使用する、Cortex-M0+、Cortex-M33、RISC-Vの３つのCPUで実習が可能です
- 受講者が実際に開発を行う場合の、ノウハウはTOPPERSのWebで公開している基礎１セミナーまたは基礎２セミナーのコンテンツを参考して頂きたい

基礎1.5コースの概要

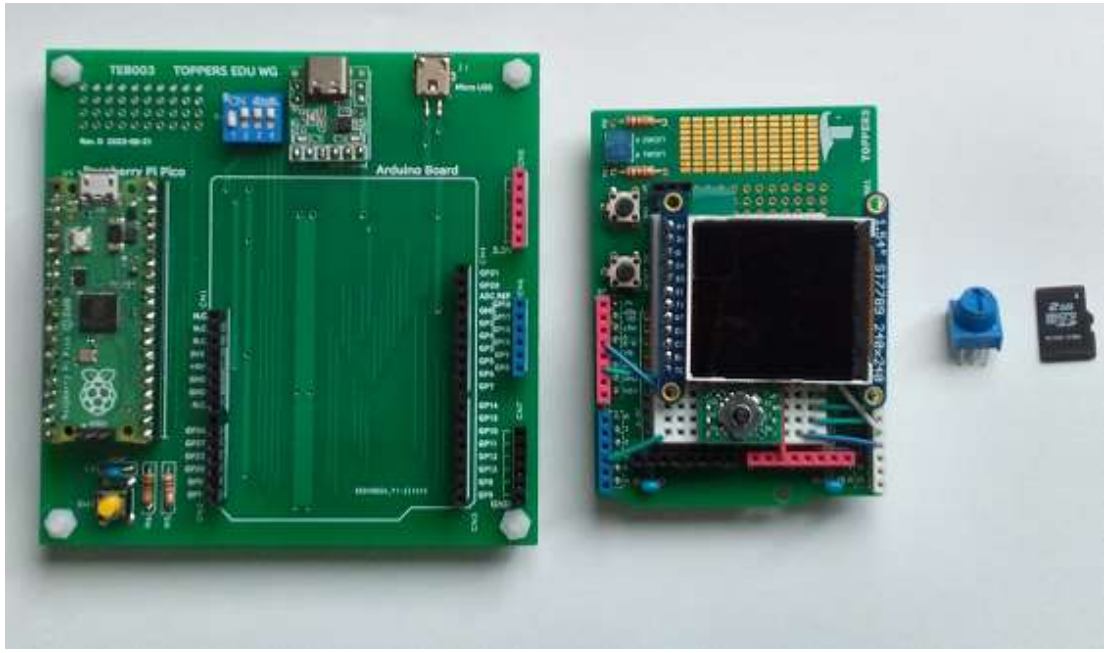
- 開発環境の説明では、使用するハードウェア、クロス開発の説明と、ベアメタル用の開発環境の説明を行います
- ベアメタル用とRTOSを使用したデバイスドライバの違いとベアメタル環境で有用な管理システムを説明します
- 「LCDJOYシールドを動かす」では、基礎3で実習したLCDJOYシールドをベアメタル環境で動作させデバイス・ドライバを用いた開発実習を行います
- 「USBデバイスHIDを動かす」では、USBデバイスやUSBのミドルウェアを使いスティック・マウスを作成します
- 「ベアメタル組込み実装の勘所」では、RTOSを用いた実装とベアメタル実装のプログラムデータサイズの比較、実際にベアメタル実装を行う上での注意点等について解説を行います

開発環境の説明

1. ハードウェア環境
2. ソフトウェア環境
3. ROMモニタを使った実行
4. デバイスドライバ解説
5. デバイスドライバ対応

セミナーで使用するボード

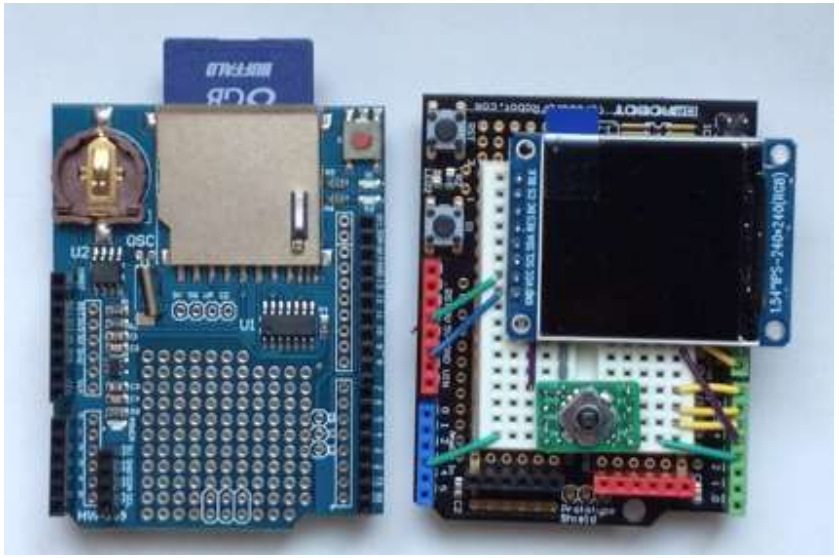
- パソコン上に開発環境がセットアップされていない場合は、開発環境をセットアップする必要があります
- 本講座を受講するには、以下のボード等が必要となります
  - 基礎3の構成と同じです



TEB003ボード
LCDJOYシールド
可変抵抗(使用しない)
SPI通信可能なSDカード

ロギング・シールドを使ったLCDJOYシールド

- ロギング・シールドと秋月電子の1.54”LCDを使ったLCDJOYシールドで代用が可能です
- ハード的な違いは、LCDとSDカードのCSの信号が逆になります



２つのシールドを二段重ねにする

TOPPERS

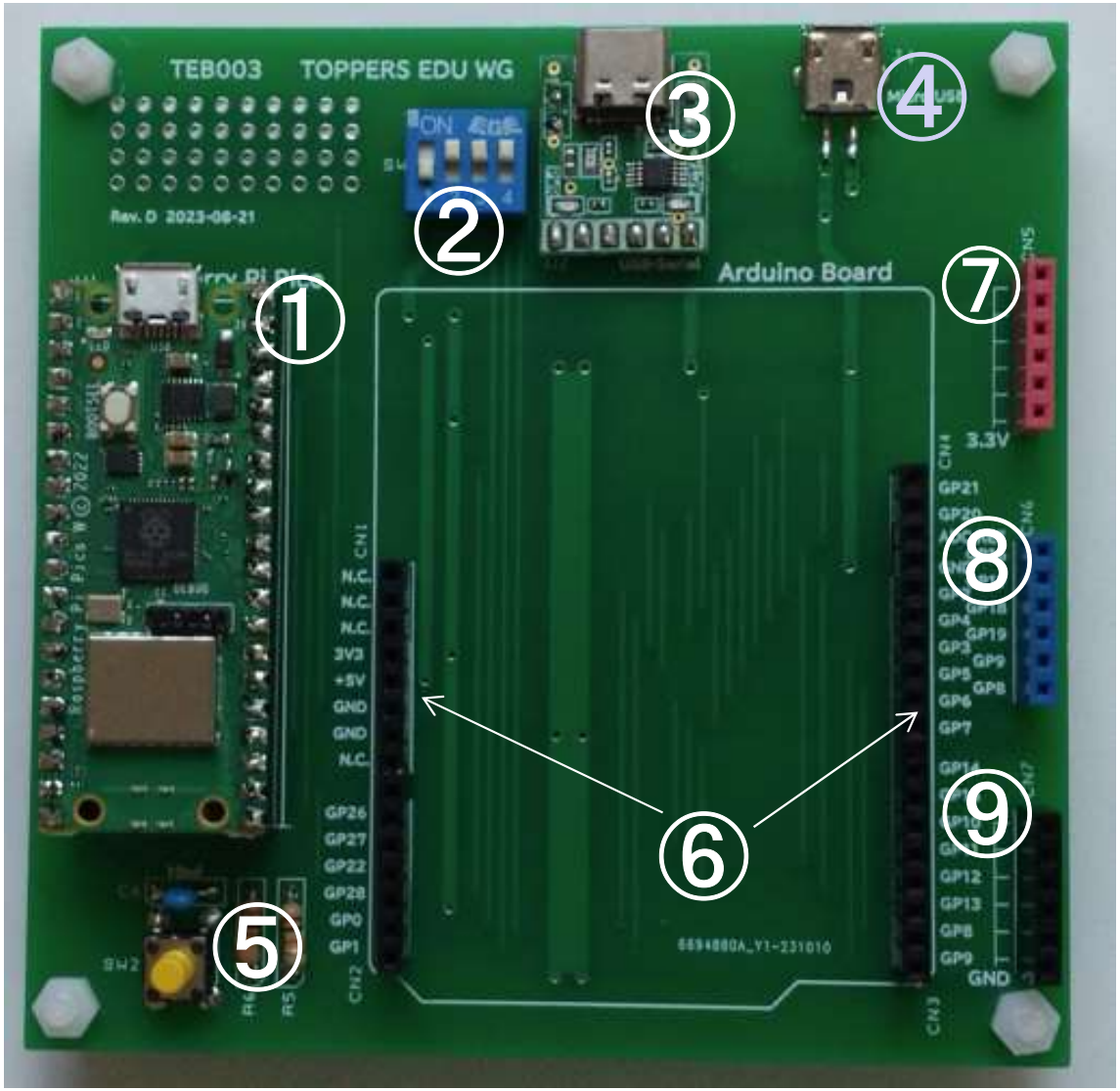
マイコンボードの概要

マイコンボードと搭載プロセッサついて解説します

- マイコンボード : Raspberry Pi Pico(W)/Pico2(W)
  - Raspberry Pi Ltd製
- 検証ボード : TEB003
- プロセッサ
  - Pico:Cortex-M0+ 125MHz
  - Pico2:Cortex-M33またはRISC-V 150MHz
- 実装機能
  - TEB003ボードで、Arduino Unoコネクタが使用可能
  - USBシリアル、USER SW、外部給電

マイコンボード : Pico/Pico2 + TEB003ボード

1. Picoコネクタ
2. DIP SW
3. USB シリアル
4. 外部給電コネクタ
5. USERスイッチ
6. Arduinoコネクタ
7. 拡張GPIO
8. 3.3Vコネクタ
9. GNDコネクタ



他のPicoボードの使用

- 本講座ではPicoコネクタに、以下のPicoボードを装着して学習ができます
  - Raspberry Pi Pico
  - Raspberry Pi Pico W
  - Raspberry Pi Pico2
  - Raspberry Pi Pico2 W
- Pico2ではCortex-M33、RISC-Vの両方の学習が可能、以下のmakeコマンドの以下のビルド引数で対象を変更可能です
  - BOARDNO 0:Pico,1:Pico2:Cortex-M33,2:Pico2:RISC-V
  - BOARDTYPE PICOとPICO_W

開発環境の説明

1. ハードウェア環境
2. ソフトウェア環境
3. ROMモニタを使った実行
4. デバイスドライバ解説
5. デバイスドライバ対応

クロス開発環境：ターゲットシステムとホストシステム

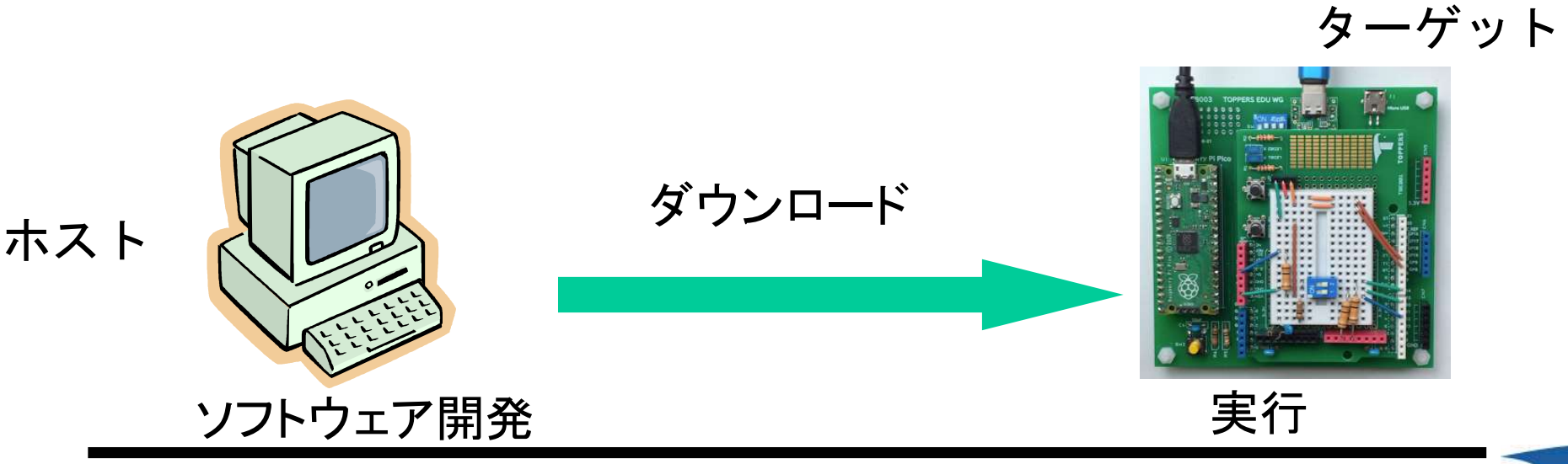
組込みシステムはクロス開発により開発します

クロス開発

- 開発環境（ホスト）と実行環境（ターゲット）が異なる開発形式
- 機械語も異なる

⟷セルフ開発

クロス開発環境の説明は、開発環境編「ソフトウェア環境の構築」の章を参照



TOPPERS

クロスコンパイラ

C言語等の高級言語記述からホストコンピュータと異なる
プロセッサのアセンブリ言語記述を生成するプログラム

- 一般に，コンパイルの前にプリプロセッサを呼び出し，コンパイル後にアセンブラ・リンカを呼び出し，実行コードを生成します
  - コンパイラドライバ
- 豊富なコンパイルオプション
  - エンディアン，フローティングの扱い，etc…
  - ターゲット非依存とターゲット依存
  - ターゲット依存のコンパイルオプションの例
    - -EB : Use big-endian byte order
    - -EL : Use little-endian byte order
    - -mthumb : USE THEMB CODE
    - -mcpu=xxxx : CPU type
    - -mfpu=yyyy : FPU type

ソフトウェア環境の構築（オープンソースを使用）

- コンパイラはGCC ARM（Windows上使用可能なARMコンパイラ）
- LINUX互換ターミナルとして、cygwinまたはmsys2を使用します
- その他のソフトウェア
  - エディタ（TeraPad, サクラエディタ，Xyzzy等）
  - ターミナルソフトウェア（TeraTermPro等）
  - PDFリーダー（Acrobat Reader等）

ターミナルソフトを使ったプログラム開発

- 本セミナーでは統合環境を使わず、LINUX互換ターミナルを使用して、コマンドで直接コンパイル等の指示を行い、プログラムを開発していきます
- 直接、コマンド設定することでツールへの設定が明確になります
- 実際は統合環境でも、下位の層でコマンドを発行して、ツールに対して指示を行っています
  - 統合環境は使い勝手がよいが、下位の層でツールに対してどのようにツールを扱うかはわからない
  - ツールに対して直接特別な指示を行う場合、どのような設定で、その指示が発行されるか不明確になるケースがあります

最低限覚えておきたい、シェルコマンド

- ターミナルで使用するシェルコマンドで、常時使用するものは、以下の通です

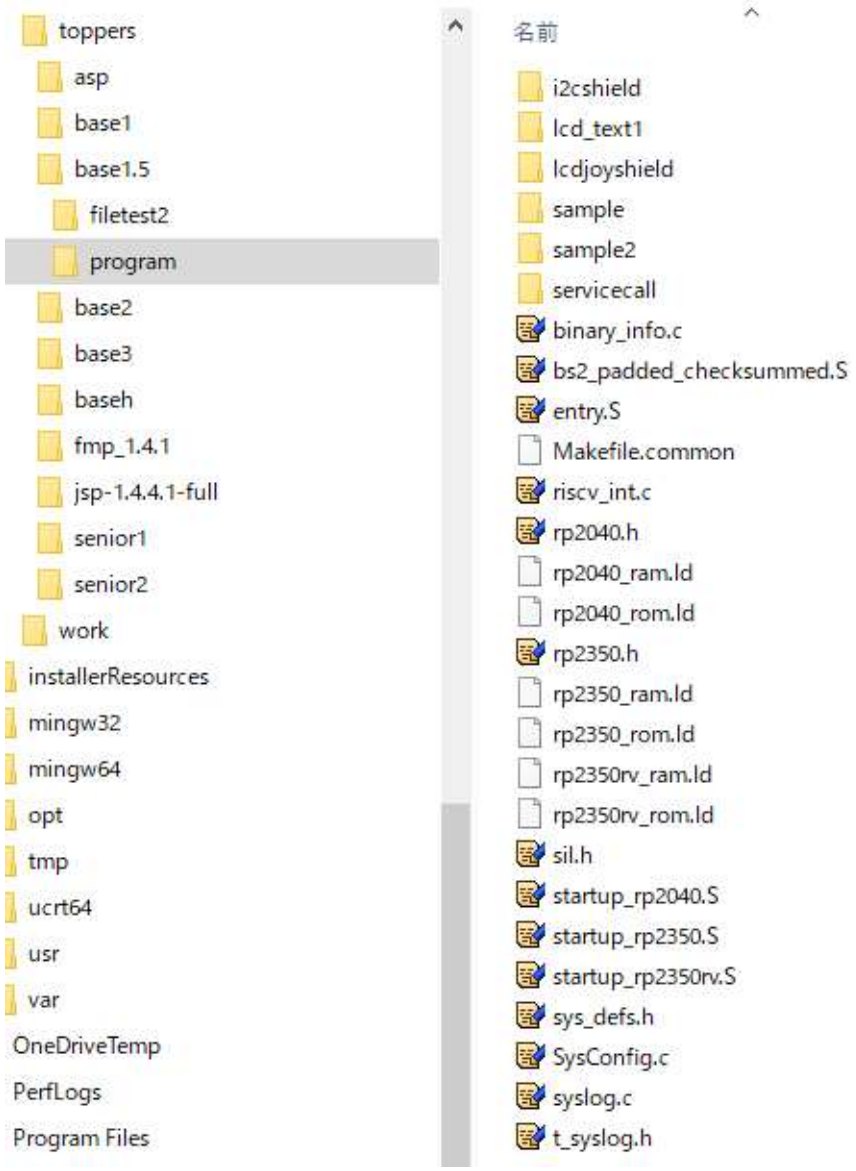
コマンド	機能
ls	ディレクトリ中のファイルを表示する
cd	ディレクトリを変更する。引数はパス名、元のディレクトリに戻るには、パス名”..”を指定 引数なしで、ホームディレクトリに戻る
diff	ファイルまたはディレクトリ内のファイルをテキスト形式で比較し、差異を表示する
history	今まで使用したコマンドの一覧を表示する
↑ ↓	↑ひとつ前に発行したコマンドを表示 ↓次のコマンドを表示
help	使用可能なデフォルトコマンドを表示する
mkdir	ディレクトリを作成する
make	Makefileを実行する。ビルドを行う

TOPPERS基礎1.5セミナー開発環境

- ソフトウェア開発環境上に、ベアメタル開発に適したソフトウェア環境を使用します
- この環境のベースとしてTOPPERS基礎1セミナー環境を使用します
  - 開発ボードに合わせたサンプル実行プログラムあり
  - スタートアップルーチンが構築済
  - シリアル通信が設定済みで、これを使ったログ表示を簡単に行える
  - RAM実行形式なので、デバッグのたびにROMの書き換えが不要
  - TOPPERS BASE PLATFORMのデバイス・ドライバをベアメタル環境で使用

ベアメタルワークスペース

- ディレクトリ表示の通り、base1.5/programの中に、本セミナー用のプログラムがあります
- programには、本セミナーで使用するデバイスドライバのソースはありません
- デバイスドライバはasp-1.9.3内のTOPPERS BASE PLATFORM(RP)のドライバを使用します



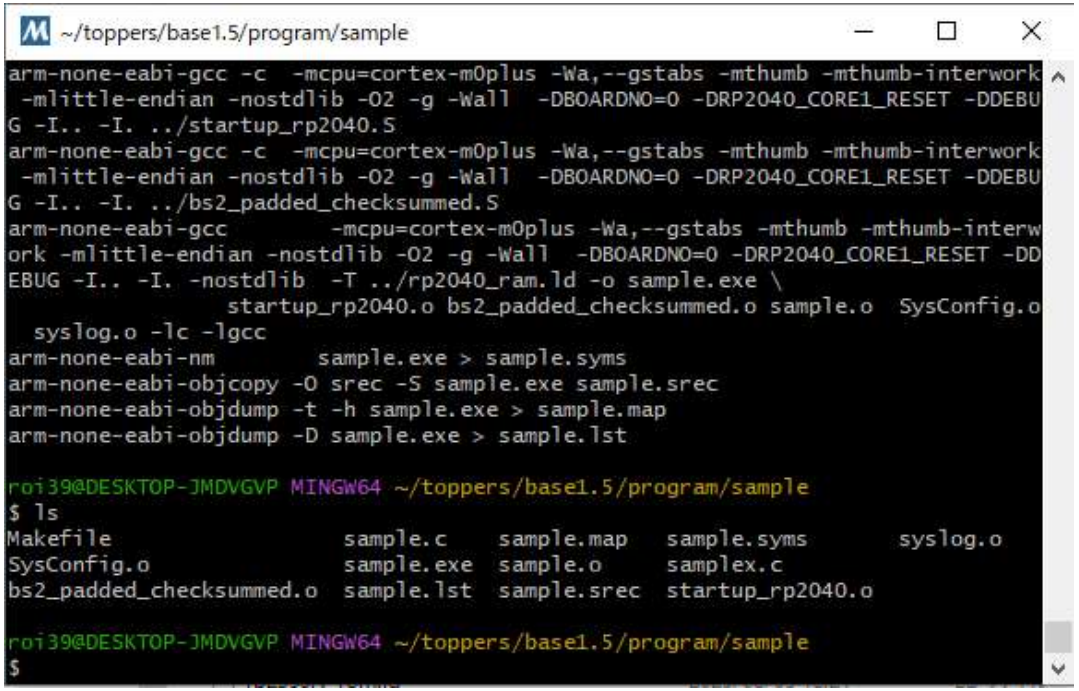
ベアメタル開発:プログラムファイル

- プログラムファイルの置き場所
  - 教材ディレクトリ/program

開発環境解説用	
sample	: スタートアップ、ログ等の説明プログラム
smaple2	: ベアメタル開発環境の説明プログラム
ベアメタル実習	
lcdjoyshield	: ASPカーネル用のLCDJOYシールド評価プログラムを ベアメタル用の書き換えたもの
lcd_text3	: 基礎3のLCDJOYシールドの簡単なプログラム
lcd_text6	: LCDの実習、最終回答
サービスコール	
servicecall	: ベアメタル開発環境のシステム部

SAMPLEプログラムのコピー

- msys2上のディレクトリにbase1.5/programをコピーし作業用のmy_programを作成します
- cdコマンドでmsys2からprogram/sampleに移動します
- makeコマンドでプログラムビルド
- lsコマンドで作成されたファイルを表示します



TOPPERS

補助プログラムの説明

- programディレクトリに、ビルドを補助するプログラムが置かれている
- Cortex-M0+/Cortex-M33/RISC-Vの対応のためプログラム数が多い
- 以下の4ファイルは、Pico/Pico2共通ファイル
  - sys_defs.hとSysConfig.cは、BOARDNOの設定でコアを判定

make BOARDNO=0	BOARDNOは省略可能、Picoをビルド
make BOARDNO=1	Pico2:Cortex-M33をビルド
make BOARDNO=2	Pico2:RISC-Vをビルド

make BOARDTYPE=PICO	BOARDTYPEは省略可能、PicoまたはPico2をビルド
make BOARDTYPE=PICO_W	Pico WまたはPico2 Wをビルド

ビルド:ファイルの確認

- デフォルトでビルド後、12ファイルが生成される
- ダウンロード実行にはsrecファイルを使用する

ファイル	ファイルの内容
sample.c	sampleプログラムC言語ソース
Makfile	GCC用MAKEファイル
sample.o	sample.cから生成された中間コードファイル
SysConfig.o	CPUの初期化、UART通信用のスタートアップルーチン中間コードファイル
startup_rp2040.o	アセンブラ・スタートアッププロクラムの中間コードファイル
syslog.o	sys_printf文、syslog文を実行するプログラム
bs2_padded_checksummed.o	rp2040用パディングデータの中間コードファイル
sample.exe	sampleプログラムの実行形式、ELF形式
sample.srec	sampleプログラムの実行形式、SRECフォーマットダウンロード用
sample.syms	sampleプログラムシンボルファイル
sample.map	sampleプログラムMAPファイル
sample.lst	sampleプログラム逆アセンブラファイル

TOPPERS

ROM実行設定に切り替え

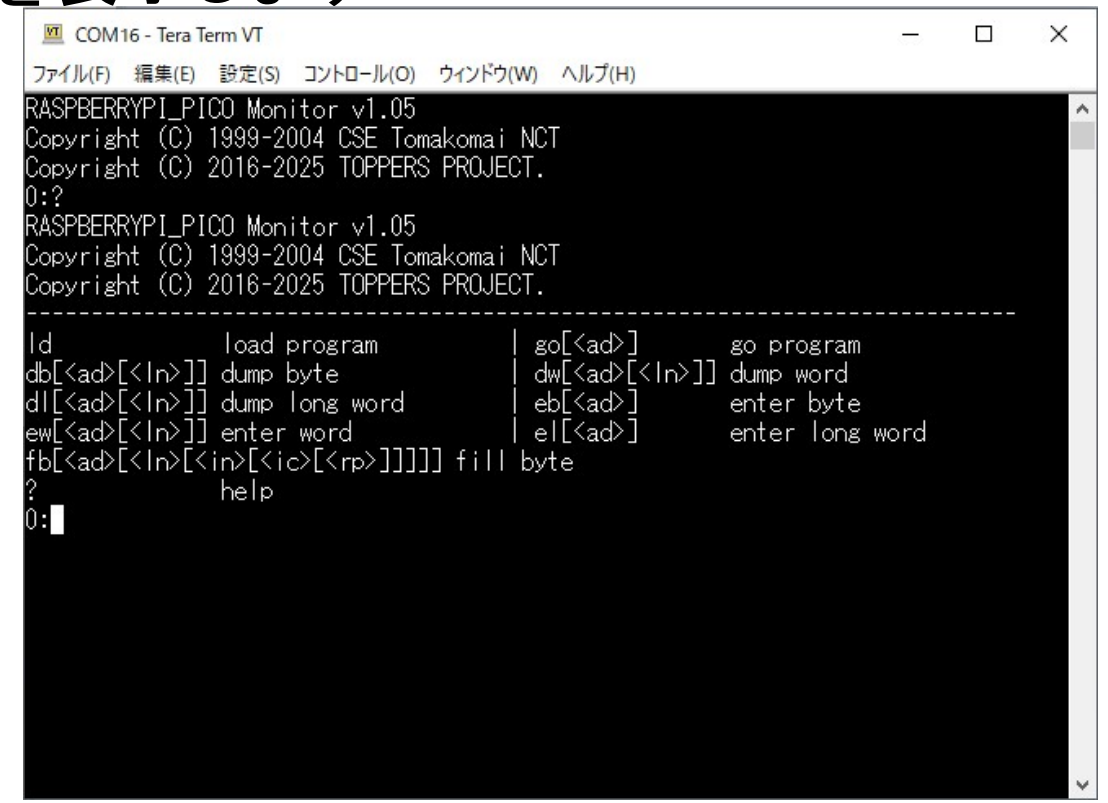
- ダウンロード実行は、RAM領域でプログラムを実行するため、電源を切れば毎回ダウンロードする必要がある
- BOOTスイッチを押し、USBケーブルをPCに接続すればPicoのフォルダが表示される、このフォルダにuf2ファイルをドロップすればFLASH ROMに書き込みができる
- ビルドのmake設定の切り替えでRAM化、ROM化が可能
  - make（return） RAM実行のビルド
  - make DBGENV=ROM（return） ROM化可能
    - uf2ファイルを作成する

開発環境の説明

1. ハードウェア環境
2. ソフトウェア環境
3. ROMモニタを使った実行
4. デバイスドライバ解説
5. デバイスドライバ対応

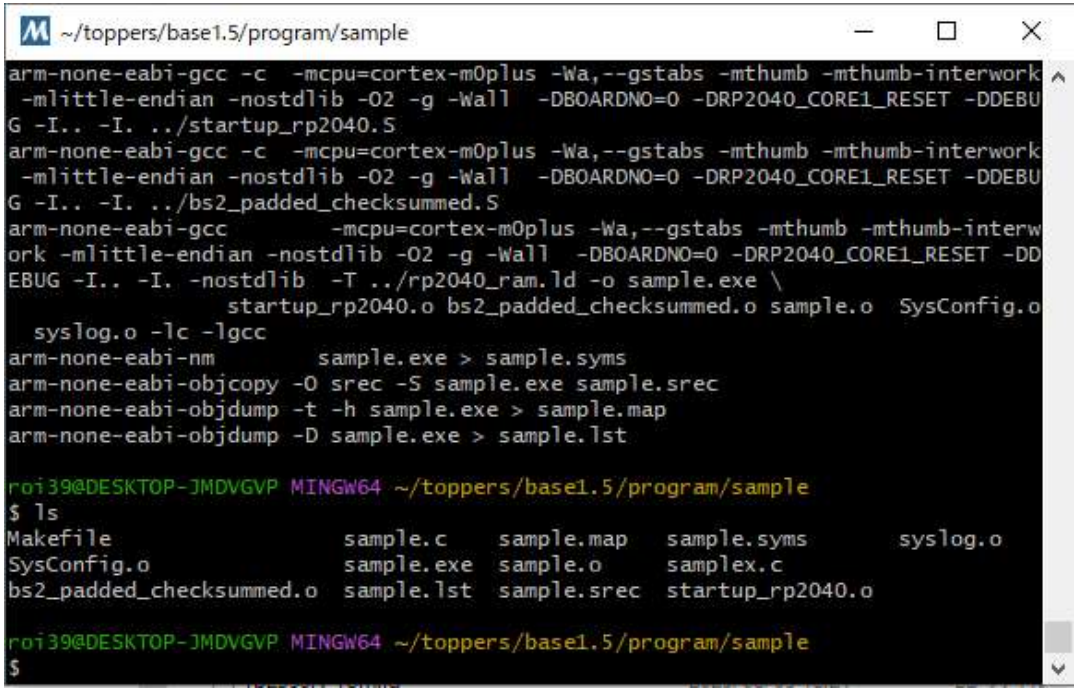
ROMモニタを使ったデバッグ環境

- Pico/Pico2にROMモニタを書き込む必要があります
- 書き込み手順は基礎１講座を参照してください
- 電源オンで0:プロンプトがでる
- ?で使用可能コマンドを表示します
- Pico/Pico2はRAMが大きいのでプログラムをダウンロードして実行が可能です



SAMPLEのビルド

- make(return) にてダウンロード実行用のプログラムをビルドします
- ビルドの手順はMakefileに記載、実行内容はsample.cに記載している

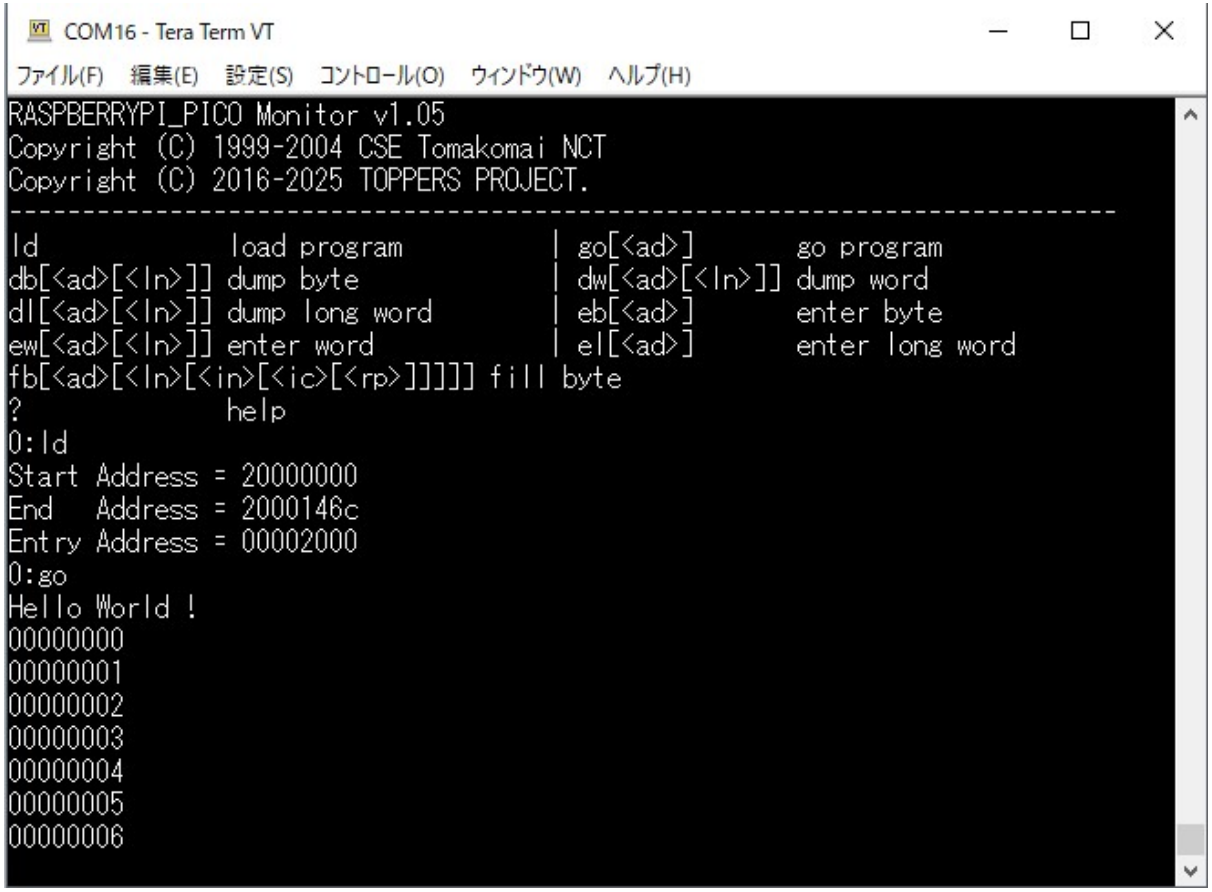


TOPPERS

実行:ダウンロード開始、実行開始

- ROMモニタでldコマンドを起動し、TeraTermにsample.srecファイルをドロップし、ダイアログのファイル転送をクリックするとダウンロードを開始する
- ダウンロード終了後、モニタのgo(return) でプログラムを実行する

Hello World！
と0.5秒ごとにカウントを表示する



デバッグ機能の確認

- sample.cの内容を確認します
- main関数には以下の2つの関数でUART表示を行う
  - sys_printf　　引数の文字列と引数を表示する
  - syslog_n　　ログ表示を行う（nは引数の数）

```
void
main(void)
{
        int count = 0;

        sys_printf("Hello World !¥n");
        for (;;) {
            syslog_1(LOG_NOTICE, “%08x“, count);
            count++;
            busy_wait();
        }
}
```


TOPPERS

デバッグ機能の確認

- Makefileを確認する
- 52行目、56行目のCOBJSにsyslog.oを追加することにより、sys_printfとsyslog文を使用できるようになる
- 後のプログラム実習で、この機能を使えばデータやIOマップの表示確認を行える

```
50 ifeq ($(DBGENV),ROM
51 CDEFS := $(CDEFS) –DROM_EXEC=1  -DBOARDTYPE=$(BOARDTYPE)
52 COBJS := $(COBJS) $(CORE).o syslog.o
53 else
54 ifeq ($(DEBUG),1)
55 CDEFS  := $(CDEFS) -DTOPPERS_RAM_EXEC -DBOARDTYPE=$(BOARDTYPE)
56 COBJS := $(COBJS) $(CORE).o syslog.o
57 endif
58 endif
```

オプション指定によるUART表示

- Makefileの54行目の設定のように、DEBUGスイッチが1の場合、syslog.oを取り込むように設定してあります
- make DEBUG=1 (return)またはROM実行でsyslog.oがリンクされDEBUGコンパイルスイッチ内の表示が有効となります
- make (return) ではsyslog.oはリンクされず、表示もおこないません
- ログ表示を行わない場合は、メモリの節約になります

```
60 ifeq ($(DEBUG),1)
61 CDEFS *= $(CDEFS) -DDEBUG
62 endif
```

オプション指定によるUART表示

- 共通のインクルードファイルsys_defs.hに、uart出力のプロトタイプ宣言あります
- DEBUGが0の場合、uart関数は関数ごと未定義となる

```
#if DEBUG == 0
#define sys_puthex(val)
#define sys_printf(format, …)
#else
#define sys_puthex(val) sys_printf(“%08x”, (val))
extern void sys_printf(const char *format, …);
#endif
```

- 以後の演習で以下のように、変数の表示が可能となる

```
led_out_bdigit(++led_count);
sys_printf(“%d¥n” led_count);
```


TOPPERS

開発環境の説明

1. ハードウェア環境
2. ソフトウェア環境
3. ROMモニタを使った実行
4. デバイスドライバ解説
5. デバイスドライバ対応

デバイス・ドライバ

- TOPPERS BASE PLATFORMではハードウェアに依存したドライバをPDIC、依存しないドライバをGDICで分類しています：次のスライドで説明
- Pico/Pico2のハードウェア依存デバイスはpico-sdkで提供されています、ハードウェア仕様書ではデバイス・ドライバを作成するには不明の部分が多いため、ハードウェア・メーカーがソフトウェアとして供給する必要があります
- PDICのインターフェイス仕様を共通化すれば、GDICはハードウェアに関係なく共通化ができます（pico-sdkのインターフェース仕様とは異なる）
- デバイス・ドライバには同期通信が必要なものと非同期通信で良いものがあり、ベアメタル用とRTOS用で仕様の差異があります

デバイスドライバ

- ハードウェアとのインターフェイスを行う関数群
- シリアル通信（UART/I2C/SPI等）に対応したハードウェアモジュールの普及により、ハードウェアを直接アクセスする下位層のデバイスドライバと、下位層(PDIC)のドライバを使ってハードウェア制御を行う上位層(GDIC)のデバイスドライバに分離しています
- デバイスドライバの構成として「μITRON4.0仕様研究会デバイスドライバ設計ガイドラインWG」が策定したDICアーキテクチャをベースに設計しています

図 2. DIC の階層構造

DICとはDevice Interface Component の略

TOPPERS

同期通信ドライバ

- 同期通信ドライバは、ハードウェアと通信しながら制御を行うドライバで、ハードウェアのタイミングに合わせて制御を行う必要があります
  - 同期通信に用いられるPDICドライバ：SPI,I2C,USB,MAC
  - 非同期通信で良いPDICドライバ：TIMER、ターミナル用UART
- 非同期通信では、バッファ等にデータを保持し割込み機能を用いればメイン関数のタイミングに関係なく制御が可能
- 同期通信の場合、メイン関数で待ちを入れながらの通信となるためベアメタルの場合、ポーリング方式となります（割込みを用いても待ちが発生する）
- RTOSのドライバでは、割込み待ち時間を他のタスク実行に割り当てられるので、CPUを有用に使用できます
  - セマフォ等の待ち機能を用いれば、簡単に構築できる

ドライバの排他制御

- RTOS用のドライバは、複数のタスクから制御される場合に備えて排他制御を行う必要があります
- ベアメタル用のドライバはメイン関数でしか呼び出しを行わないため、タスク用の排他制御を行う必要はありません
- PDICドライバは初期化構造体のsemidで通信用のセマフォ、semlockで排他制御を設定します

```
spi_initd.Baudrate      = 10*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;
spi_initd.DMAIntNo      = DMA_INTNO;
spi_initd.DMARxChannel  = 1;
spi_initd.DMATxChannel  = 0;
spi_initd.Mode          = SPI_MODE_MASTER;
spi_initd.semid         = SPI1DMA_SEM;
spi_initd.semlock       = 0;
```

通信用のセマフォ設定

排他制御用のセマフォ設定
0またはLOCK_SEMID設定
で排他制御をスキップする

TOPPERS

ベアメタル環境でのシステム機能

- ベアメタル環境でも、最低限のシステム機能が必要
  - システム・タイマ機能：経過時間管理をソフト的に行う
  - セマフォ機能：割込み待ち機能を共通化する
- これらの機能をμITRON仕様のサービスコール準拠で実装すれば、μITRON仕様で作成されたデバイス・ドライバをベアメタル環境で使用できます
- 以下は基本的に必要なサービスコール

μITRONサービスコール	ファイルの内容
get_tim	起動時からの1msの時間を取り出す
sig_sem	バイナリセマフォ資源の返却
twai_sem	時間機能付き、バイナリセマフォ資源確保
unl_cpu	割込み禁止
loc_cpu	割込み許可

ベアメタル環境でのシステム機能

- バイナリセマフォのひとつをシステム用に使用すると、サービスコールの拡張が可能
  - SYSTEM_SEMIDをアプリに開放せず使用
- 以下に、拡張用のマクロ定義を記載
  - servicecall/kernel.h :インクルードファイル
  - servicecall/kernel.c :ソースファイル

```
#define isig_sem    sig_sem
#define wai_sem(a)  twai_sem((a), TMO_FEVR)
#define dly_tsk(a)  twai_sem(SYSTEM_SEMID, (a))
#define slp_tsk()   twai_sem(SYSTEM_SEMID, TMO_FEVR)
#define iwup_tsk()  sig_sem(SYSTEM_SEMID)
#define wup_tsk()   sig_sem(SYSTEM_SEMID)
```

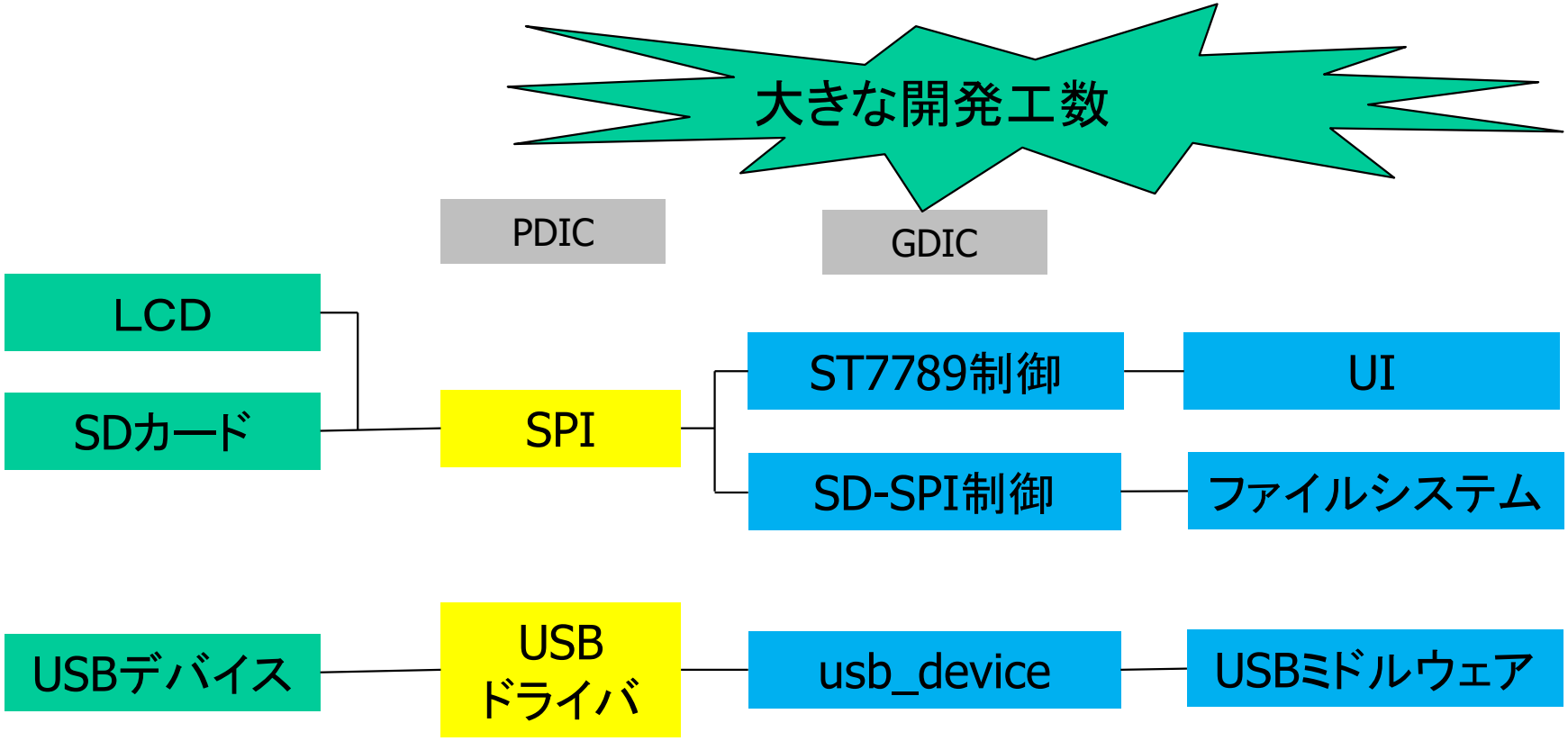
ベアメタル環境サービスコール提供ファイル

- PDICドライバはRTOSのリソースとして、セマフォや時間待ち機能を使用します
- システム部は、servicecallディレクトリに置いています
  - 多くはドライバやアプリがインクルードするファイルであり、エラーが発生しないような定義を行っている

ファイル名	機能
itron.h	μITRONの型定義
kernel.c	ベアメタル・サービスコールの本体のプログラム
kernel.h	ベアメタル・サービスコールのインクルードファイル
t_stddef.h	変数の型宣言
t_stdlib.h	コンパイル・エラー防止の中身のないインクルード・ファイル
target_stddef.h	ターゲットの型定義
target_syssvc.h	ハードウェア設定、クロック設定等の定義

デバイスドライバの取り込み

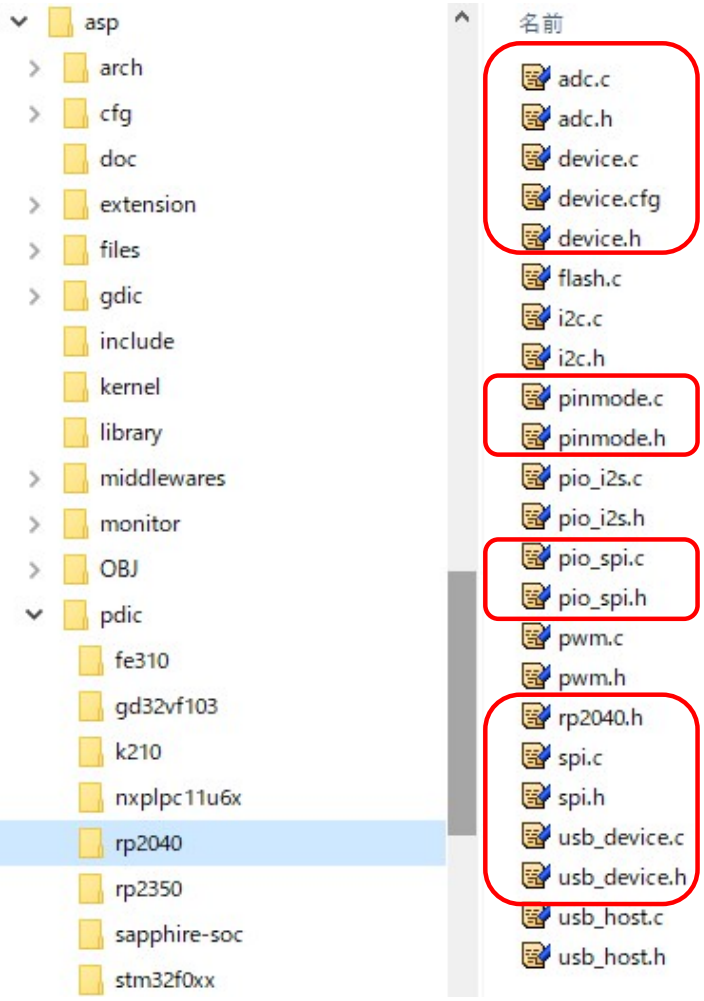
- 本実習では、LCDやSDカード等の制御にデバイスを制御するデバイスドライバが必要になります
- これらをドライバを効率よく取り込むために、TOPPERS BASE PLATFORM(RP)のドライバを使用します



TOPPERS BASEPLATFORMのドライバ

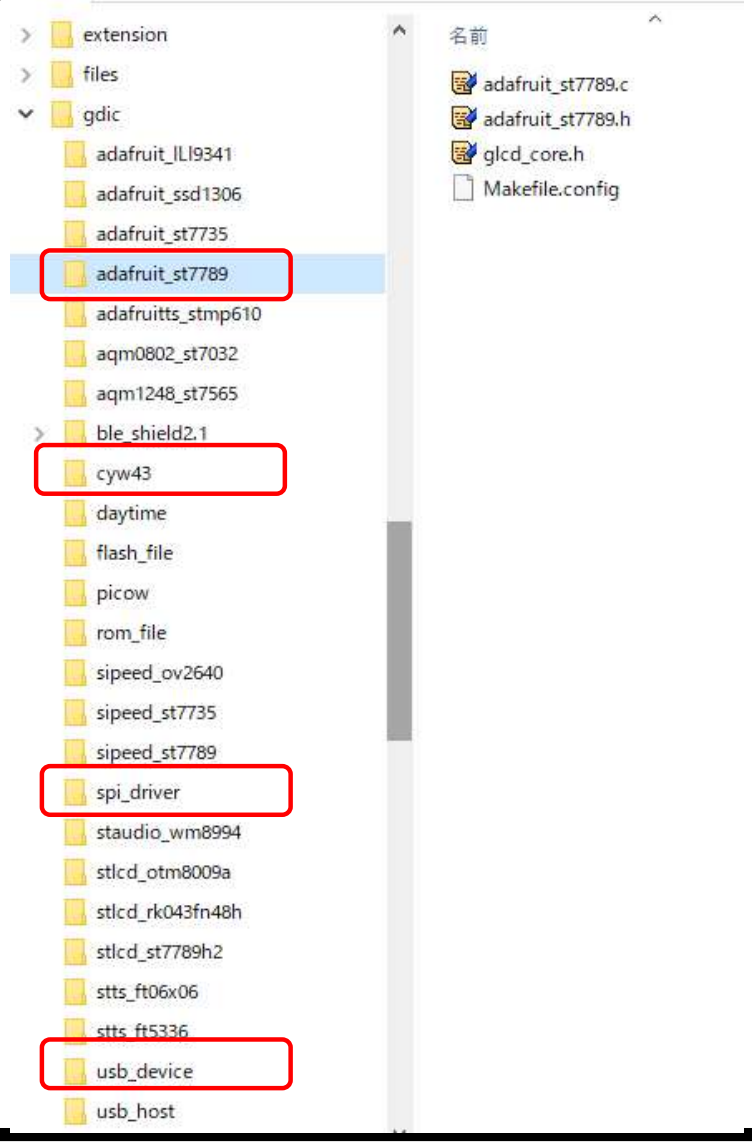
- 本実習で使用するPDICドライバは以下の通り
  - pio_spi(PICO Wで使用)

ハードウェア	PDICドライバ
LCD	device adc spi pinmode pio_spi (pdic/rp2xxx)
SDカード	
USBデバイス	USB DEVICE (pdic/rp2xxx)



TOPPERS BASEPLATFORMのドライバ

- 本実習で使用するGDICドライバは以下の通り
  - cyw43(PICO Wで使用)



ハードウェア	GDICドライバ
LCD	ST7789 (gdic/adafruit_st7789)
SDカード	SPI_DRIVER (gdic/spi_driver)
USBデバイス	USB_DEVICE (gdic/usb_device)
cyw43	CYW43 (gdic/cyw43)

開発環境の説明

1. ハードウェア環境
2. ソフトウェア環境
3. ROMモニタを使った実行
4. デバイスドライバ解説
5. デバイスドライバ対応

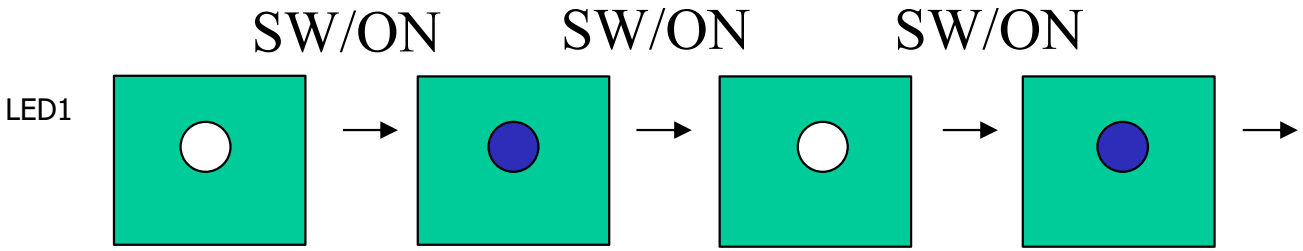
sample2解説：目的

TOPPERS BASE PLATFORMのPDICとシステム部を使って
GPIOドライバ（LED,スイッチ）の制御について学ぶ

- 初期設定，データの入出力，事象の待ち方
  - GPIOによるLED処理、スイッチのON/OFFを割込みで取り込みLEDを点灯、消灯する
- プログラミング対象の周辺デバイス
  - LED（プロマブル入出力ポート出力）
  - スイッチ（プロマブル入出力ポート入力：割込み）
- プログラムファイルの置き場所
  - 教材ディレクトリ/base1.5/program/sample2
- sampleとの比較で、拡張方法を検証

sample2 概要

- ユーザースイッチを押すと割込みが発生し、押すごとにLEDの点灯、消灯を繰り返す



- 学習内容
  - メイン関数からデバイスドライバの呼び出し
  - 割込み関数の理解
  - PDIC/LED,SWのソースの理解
  - サービスコールdly_tskの理解
- 構成ファイル
  - sample.c/kernel_cfg.h

sampleとsample2をdiffコマンドで比較

- Makefileの追加部分が明らかになる
  - aspディレクトリの取り込み　　<がsampleのMakefile
    >がsample2のMakefile

```
diff sample/Makefile sample2/Makefile
2a3,4
> RTOS_PATH = $(ROOT_PATH)/../../asp
> SRCDIR    = $(RTOS_PATH)
21a24,30
> #  GDIC Makefile のインクルード
> #
> ifeq ($(BOARDTYPE),PICO_W)
> include $(SRCDIR)/gdic/cyw43/Makefile.config
> endif
>
> #
46c55
< vpath %.c $(UTASK_DIR)
---
> vpath %.c $(UTASK_DIR):$(RTOS_PATH)/pdic/$(CORE):$(SYSSVC_DIR):$(ROOT_PATH)/servicecall
```

aspカーネルディレクトリのパスを定義

Pico W用にcyw43を取り込む（LEDの制御に必要）

aspカーネル上のVPATHを追加
$(RTOS_PATH)/pdic/$(CORE) はpdic
$(SYSSVC_DIR)はGDICとミドルウェア

TOPPERS

sampleとsample2をdiffコマンドで比較

- Makefileの追加部分が明らかになる
  - サービスコールとPDICドライバの追加
  - PDIC/GDICのインクルードパスの追加

```
49c58
< COBJS = $(OBJNAME).o $(LOCAL_COBJS)
---
> COBJS = $(OBJNAME).o kernel.o device.o pio_spi.o $(SYSSVC_COBJS)
                        $(LOCAL_COBJS)
66c75
< INCLUDES := -I$(UTASK_DIR) $(INCLUDES)
---
> INCLUDES := -I$(UTASK_DIR) -I$(RTOS_PATH)/pdic/$(CORE)
            -I$(ROOT_PATH)/servicecall $(INCLUDES)
```

サービスコールとPDICドライバのソースを追加

PDIC/GDIC等のインクルードパスを追加

TOPPERS

smaple2 : メイン関数

- 後述するスタートアップ・ルーチンから呼び出される

システム部の初期化

PDICデバイスドライバの初期化

スイッチ割込み設定

key_reqがゼロより大きくなるたびにLED表示を反転させる

カウンタを表示させ、500msの待ち

```
void
main(void)
{
    uint16_t led_data = 0;
    uint32_t count = 0;
    SYSTIM tm;
    /*疑似カーネルの初期化とATT_INIの実行*/
    init_kernel();
    pico_system_init(0);
    led_init(0);
    switch_push_init(0);
    setup_sw_func((intptr_t)sw_handle);
#ifdef __riscv
    hazard3_set_vector(INT_VECTOR_IO_BANK0, IO_BANK0_IRQHandler);
#endif
    init_int(INT_VECTOR_IO_BANK0, 0x3, true);
    led_out(led_data);
    syslog_0(LOG_NOTICE, “main start !”);
    for (;;) {
        if(key_req != 0){
            led_data ^= LED01;
            led_out(led_data);
            syslog_1(LOG_NOTICE, “led_data[%04x] !”, led_data);
            key_req = 0;
        }
        get_tim(&tim);
        syslog_1(LOG_NOTICE, “%08d %08x”, count, tm);
        count++;
        dly_tsk(500);
    }
}
```


TOPPERS

sample2：割込みコールバック関数

- main関数のswitch_push_init関数でユーザースイッチの初期化を行い、setup_sw_func関数を用いて、割込みハンドラからのコールバック関数を登録する

```
uint8_t key_req;

/*
 *  SWITCH割込み
 */
static void
sw_handle(void)
{
    syslog_0(LOG_NOTICE, “## sw_int ##”);
    key_req++;
}
```

sw_handleはユーザースイッチ押下で発生する割込みからコールバックされる
key_reqをインクリメントする

ユーザースイッチの割込み関数

```
void
IO_BANK0_IRQHandler(void)
{
    sio_bank0_handler();
}
```

ユーザースイッチを含むGPIO割込み、PDIC:device.c内の割込みハンドラを呼び出す

sample2：PDICのユーザースイッチ関数

- PDICのコモンドライバはasp/pdic/rp2xxx/device.cにある

```
/*
 *  GPIO割込みハンドラ
 */
void
sio_bank0_handler(void)
{
    uint32_t pcbase = TADR_IO_BANK0_BASE + TOFF_IO_BANK0_PROC0_IRQ_CTRL
                                + get_CoreNo() * IRQ_CTRL_SIZE;
    uint32_t offset, event, emask, ins, ine, gpio, i;
    for(gpio = 0 ; gpio < NUM_BANK0_GPIO ; gpio += 8){
        offset = (gpio >> 3) * 4;
        ine = sil_rew_mem((uint32_t *)(pcbase+TOFF_BANK0_IRQ_CTRL_INTE+offset));
        ins = sil_rew_mem((uint32_t *)(pcbase+TOFF_BANK0_IRQ_CTRL_INTS+offset));
        event = ine & ins;
        if(event == 0)
            continue;
        for(i = gpio, emask = 0xF ; i < (gpio+8) && i < NUM_BANK0_GPIO ; i++){
            if((event & emask) != 0){
                sil_wrw_mem((uint32_t *)(TADR_IO_BANK0_BASE+TOFF_IO_BANK0_INTR+offset),
                                        (event & emask));
                if(gpio_func[i] != NULL)
                    gpio_func[i]();
            }
        emask <<= 4;
        }
    }
}
```

ここでsw_handleがコールバック

TOPPERS

sample2：PDICのLED関数

- main関数からLED設定用に呼び出すled_init, led_out関数も、device.cに記述

```
/*
 *  LED接続ポート読み出し
 */
__attribute__((weak)) uint_t
led_in(void)
{
    const gio_conftype *pled = &led_confdata[0];
    uint32_t data, i;

    for(i = 0, data = 0 ; i < NUM_LED ; pled++, i++){
#if BOARDTYPE == PICO_W
        if(cyw43_gpio_get(pled->pinpos))
#else
       if((sil_rew_mem((uint32_t *)(TADR_SIO_BASE+TOFF_SIO_GPIO_OUT)) & (1<<pled->pinpos)) != 0)
#endif
            data |= (1<<i);
    }
    return data;
}
```

Pico Wはcyw43上のGPIO読み込み

PicoはPico上のGPIO読み込み

TOPPERS

sample2：PDICのLED関数

- Pico WのLEDはCyw43上のGPIOに繋がれているため、PIO_SPIドライバを使って通信を行う

```
/*
 *  LED接続ポート書き込み
 */
__attribute__((weak)) void
led_out(unsigned int led_data)
{
    const gio_conftype *pled = &led_confdata[0];
   uint32_t i;

#if BOARDTYPE == PICO_W
    /* 設定値はLED接続ビット以外は変更しない */
    for(i = 0 ; i < NUM_LED ; pled++, i++){
        cyw43_gpio_put(i, ((led_data >> i) & 1));
    }
#else
    /* 設定値はLED接続ビット以外は変更しない */
    for(i = 0 ; i < NUM_LED ; pled++, i++){
        if((led_data & (1<<i)) != 0)
            sil_wrw_mem((uint32_t *)(TADR_SIO_BASE+TOFF_SIO_GPIO_OUT_SET), (1<<pled->pinpos));
        else
            sil_wrw_mem((uint32_t *)(TADR_SIO_BASE+TOFF_SIO_GPIO_OUT_CLR), (1<<pled->pinpos));
    }
#endif
}
```

Pico Wはcyw43上のGPIO書き込み

PicoはPico上のGPIO書き込み

TOPPERS

smaple2: SIL関数

デバイスへのアクセスを専用の関数により実現

- プログラムのハードウェア依存性が弱まる
- シミュレーション環境への対応が容易

例）デバイスドライバ設計ガイドラインのアクセスインタフェース

– データの読み込みと書き込み

```
Inline uint32_t
sil_rew_mem(const uint32_t *mem){
    uint32_t data;
    data = *((const volatile uint32_t *) mem);
    return (data);
}
```

```
Inline void
sil_wrh_mem(uint16_t *mem, uint16_t data){
    *((volatile uint16_t *)mem) = data;
}
```


TOPPERS

sample2：SYSTICKの起動

- kernel.cシステムタイマのソースを記載
  - systic_initでシステムタイマを初期化

```
/*
 *  システムタイマ初期化
 */
static void
systic_init(void)
{
#ifndef __riscv
    /* 停止 */
    sil_andw_mem((uint32_t *)(TADR_SYSTIC_CONTROL), SYSTIC_ENABLE);

    sil_wrw_mem((uint32_t *)(TADR_SYSTIC_RELOAD), TIMER_CLOCK - 1);
    sil_wrw_mem((uint32_t *)(TADR_SYSTIC_CURRENT), TIMER_CLOCK - 1);

    /* プロセッサクロックの使用で起動 */
    sil_orw_mem((uint32_t *)(TADR_SYSTIC_CONTROL), (SYSTIC_ENABLE|SYSTIC_CLKSOURCE|SYSTIC_TICINT));
#else
    uint64_t mtimecmp;

    /*
     *  タイマ周期を設定し，タイマの動作を開始する．
     */
    mtimecmp = target_current_timer_value() + TIMER_CLOCK;
    target_set_target_timer_value(mtimecmp);
    hazard3_set_vector(INT_VECTOR_MTIMECMP, SysTick_Handler);
    init_int(INT_VECTOR_MTIMECMP, 0x01, true);
#endif
}
```

Cortex-MはSysTickを使用

RISC-VはMTIMEを使用

TOPPERS

sample1 : システム時間の設定

- システムタイマの初期化後、1ms毎に割込みが発生しSysTick_Handler関数が呼び出される
- この関数からsystimeをインクリメントすることで時間が更新される

```
volatile uint32_t systime;

void
SysTick_Handler(void)
{
#ifndef __riscv
    /* probe_int のため，COUNTFLAGをクリア */
    sil_rew_mem((uint32_t *)(TADR_SYSTIC_CONTROL));
#else
    uint64_t mtimecmp;

    mtimecmp = target_get_target_timer_value();
    mtimecmp += TIMER_CLOCK;
    target_set_target_timer_value(mtimecmp);
#endif
    systime++;
    timer_1ms(systime);
}
```

システム時間のインクリメント
ユーザー関数の呼び出し

TOPPERS

sample2：dly_tsk

- 時間待ちサービスコールdly_tskはtwai_semを用いて実装している
- wup_tskサービスコール以外ではSYSTEM_SEMIDはセマフォの返却されない

kernel.hの定義

```
#ifndef NUM_SEMID
#define NUM_SEMID        4
#endif
#define SYSTEM_SEMID      (NUM_SEMID)
#define LOCK_SEMID        1024

#define iact_tsk(a)
#define isig_sem    sig_sem
#define wai_sem(a)  twai_sem((a), TMO_FEVR)
#define dly_tsk(a)  twai_sem(SYSTEM_SEMID, (a))
#define slp_tsk()   twai_sem(SYSTEM_SEMID, TMO_FEVR)
#define iwup_tsk()  sig_sem(SYSTEM_SEMID)
#define wup_tsk()   sig_sem(SYSTEM_SEMID)
```

```
#define INDEX_SEM(semid)  ((uint_t)((semid) - 1))
volatile uint8_t bsem_id[NUM_SEMID];
/*  ベアメタルμITRON sig_sem(isig_sem) */
ER sig_sem(ID semid)
{
   uint8_t no = INDEX_SEM(semid);
   if(no < NUM_SEMID){
      if(bsem_id[no] < bsem_mx[no])
         bsem_id[no]++;
         return E_OK;
      }
      else if(semid >= LOCK_SEMID)
         return E_OK;
      else
         return E_PAR;
}
/*  ベアメタルμITRON twai_sem(wai_sem) */
ER twai_sem(ID semid, TMO tmout)
{
   uint8_t no = INDEX_SEM(semid);
   uint32_t current_tim = systime;
   if(no < NUM_SEMID){
      loc_cpu();
      if(bsem_id[no] >= 0)
         bsem_id[no]--;
      unl_cpu();
      while(bsem_id[no] < 0){
         if(tmout != TMO_FEVR && (systime-current_tim) > tmout)
            return E_TMOUT;
         asm volatile ("wfi");
      }
       return E_OK;
   }
   else if(semid >= LOCK_SEMID)
      return E_OK;
   else
      return E_PAR;
}
```

セマフォの取得、または時間経過で関数を終了

TOPPERS

sample2 : startup_rp2040.S

アセンブリ言語で記述されたファイル

- 各プログラムで共通に使用できる
- セクションの配置
  - ROM実行の場合、ROM上に配置
  - RAM実行の場合、RAM上に配置
- ベクタテーブル
  - ROM実行の場合、ROM上に配置
  - RAM実行の場合、RAM上に配置
    - ベクタテーブルの設定アドレスを切り替え
- スタートアップルーチン
  - セクションの初期化
  - メイン関数の呼び出し

smaple2 : startup_rp2040.S 固定ベクタ

- PicoのCortex-M0+はリセット後，0x10000104番地のベクタから実行を始める
- ROM実行の場合、0x10000104にリセット関数(Reset_Handler)へのポインタを記載する
- 0x1000100番地にリセット時のmspのアドレスを置く
  - Pico2はリセット手順、アドレスは異なります

```
g_pfnVectors:
  .word  _estack
  .word  Reset_Handler
  .word  NMI_Handler
  .word  HardFault_Handler
  .word  MemManage_Handler
  .word  BusFault_Handler
  .word  UsageFault_Handler
  .word  0
  .word  0
  .word  0
  .word  0
  .word  SVC_Handler
  .word  DebugMon_Handler
  .word  0
  .word  PendSV_Handler
  .word  SysTick_Handler
```

RESET時のスタックポインタ

リセット時の関数ポインタ

SYSTICKの割込みベクタ

TOPPERS

sample2：startup_rp2040.S GPIO割込み

- ユーザースイッチの割込みを含む、GPIOの割込みベクタも記載しています
- この割込みも.weak宣言でDefault_Handlerに設定されています

```
g_pfnVectors:
  .word  _estack
  .word  Reset_Handler
  .word  NMI_Handler
  .word  HardFault_Handler
  .word  0
     :
  .word SysTick_Handler
  .word  Timer0_IRQHandler
     :
  .word  DMA0_IRQHandler
  .word  DMA0_IRQHandler
  .word  IO_BANK0_IRQHandler
  .word  IO_QSPI_IRQHandler
```

GPIOの割込みベクタ

```
  .weak IO_BANK0_IRQHandler
  .thumb_set IO_BANK0_IRQHandler, Default_Handler
```


TOPPERS

LCDJOYシールドを動かす

1. LCDJOYシールドサンプル
2. SDカードファイルを動かす
3. 漢字を表示する
4. ファイルライブラリィを動かす

lcd_text3概要：テキスト表示とジョイスティック対応

- ベアメタル開発でデバイスドライバを組込む
- 学習内容
  - アプリ機能の解説
  - ベアメタル対応の理解と確認
  - PDIC/GDIC/UIの対応の確認
  - 起動方法の確認
  - タスク用プログラムの確認
- 構成ファイル
  - diskutils.c　　：ダミーソースファイル（後述）
  - kernel_cfg.h　　：カーネル・リソースインクルード
  - lcd_text.c　　：ソースファイル
  - lcd_text.h　　：インクルードファイル
  - Makefile　　：メイクファイル

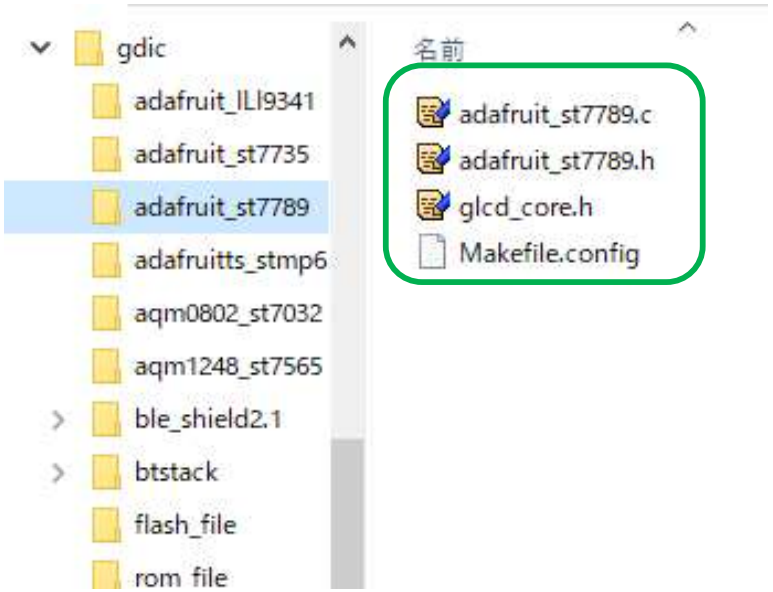
lcd_text3の動作説明

ビルドを行い動作確認を行う

- 実行にはLCDJOYシールドが必要
- lcd_text3は以下の処理を行う
  - 緑のドットを中央に1秒表示
  - LCDに”Hello World !”のテキスト表示
  - ジョイスティックを検知し、LCDに表示
  - LED点灯やPUSH SWITCHの割込み表示
- RTOS環境との違い
  - タスクモニタの機能はない
  - 標準入出力機能はない
- プログラムファイルの置き場所
  - 教材ディレクトリ/base1.5/program/lcd_text3

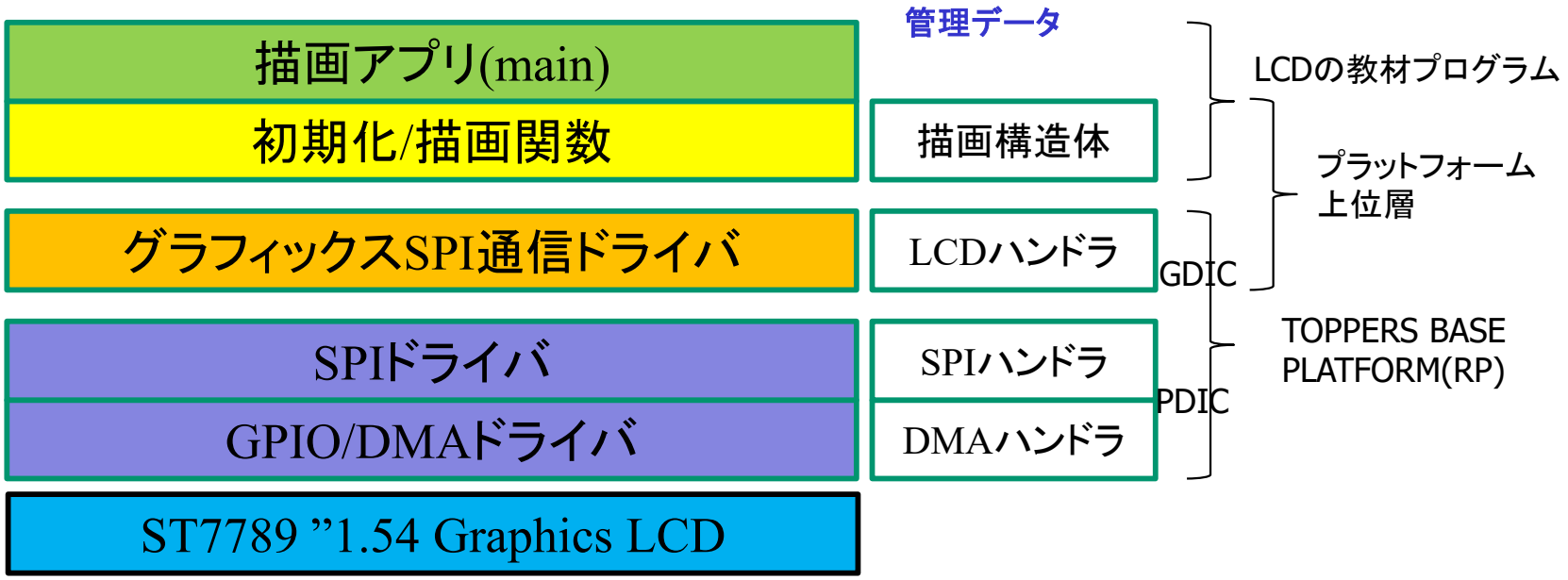
LCDドライバの設定手順

- GDICドライバ:adafruit_st7789を使用
  - LCDJOYシールドはST7789を使用のため流用可能
  - GPIOの初期化
  - SPIドライバが必要
  - GDICドライバでLCD設定
    - LCDのリセット、初期化
    - 初期データ設定
    - 表示データ設定



グラフィックLCD通信ドライバ構成

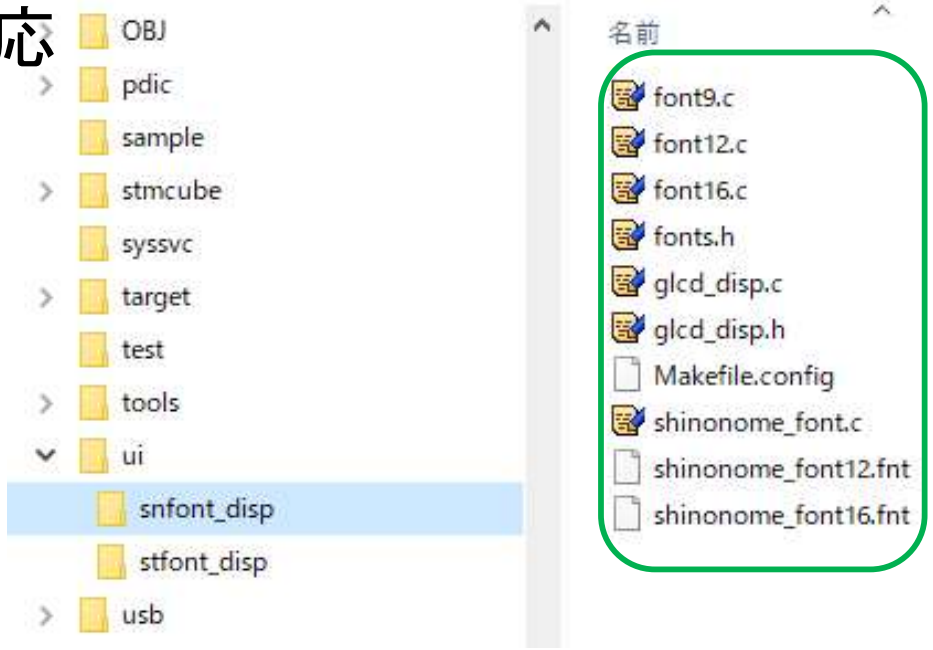
- グラフィックLCDを制御するソフトウェアの構造図を示す
  - TOPPERS BASE PLATFORMのgdicディレクトリに含まれる
  - GDICドライバはCPUの差異には依存しない
  - GDICドライバはPDICドライバのAPIに依存する
- アプリからは、SPIドライバは見えず、初期化/描画関数(adafruit_st7789ドライバ)を用いてLCD表示を行う



文字の表示手順

- UIミドルウェア：snfont_dispが必要
  - UIミドルウェアはGDICのグラフィックドライバが必要
  - 文字や文字列の表示関数を提供する
  - 以下のフォントに対応
    - ASCIIフォント：9,12,16
    - 東雲フォント12x12
    - 東雲フォント16x16

漢字フォントは容量が大きいため通常のファイルかROMファイルで対応



Makefileのドライバ取り込み

- sample2のMakefileと比較してソースの取り込みを解説

```
#
#  ボードタイプ
#
ifeq ($(BOARDTYPE),)
BOARDTYPE = PICO
endif

#
#  ミドルウェアの Makefile のインクルード
#
include $(SRCDIR)/ui/snfont_disp/Makefile.config

#
#  GDICドライバの取り込み
#
include $(SRCDIR)/gdic/adafruit_st7789/Makefile.config
ifeq ($(BOARDTYPE),PICO_W)
include $(SRCDIR)/gdic/cyw43/Makefile.config
endif

#
#  オブジェクトファイル名の定義
#
OBJNAME = lcd_text
```

Makefile.configを使ってUIミドルウェアを取り込む

Makefile.configを使ってGDICのST7789ドライバを取り込む

TOPPERS

Makefileのドライバ取り込み

- アプリでsprintf文を使用しているためソースとパスを設定
  - asp/monitor/stdioディレクリ内にsprintf.cがある

```
TARGET_DIR = .
UTASK_DIR = $(ROOT_PATH)
vpath %.c $(UTASK_DIR):$(RTOS_PATH)/pdic/$(CORE):$(SYSSVC_DIR):$(ROOT_PATH)/servicecall
                              :$(SRCDIR)/monitor/stdio
vpath %.S $(UTASK_DIR)

COBJS = $(OBJNAME).o device.o pio_spi.o spi.o pinmode.o kernel.o sprintf.o
                              $(SYSSVC_COBJS) $(LOCAL_COBJS)
AOBJS = $(LOCAL_AOBJS)

ifeq ($(DBGENV),ROM)
CDEFS  := $(CDEFS) -DROM_EXEC=1 -DBOARDTYPE=$(BOARDTYPE)
                              -DLOGSHIELD=$(LOGSHIELD) -DNOT_USEFILFONT
COBJS  := $(COBJS) syslog.o
else
ifeq ($(DEBUG),1)
CDEFS  := $(CDEFS) -DTOPPERS_RAM_EXEC -DBOARDTYPE=$(BOARDTYPE)
                              -DLOGSHIELD=$(LOGSHIELD) -DNOT_USEFILFONT
COBJS  := $(COBJS) syslog.o
endif
endif
```

使用するPDICドライバを追加

sprintf関数をアプリで使用

SDカードが使えないためファイル
フォントを使用しない設定

TOPPERS

使用するカーネル・リソースを定義：kernel_cfg.h

- 追加したSPIのPDICドライバで使用する通信セマフォのIDを定義
  - デフォルト設定で1～3のセマフォIDが使用可能

```
#ifndef TOPPERS_KERNEL_CFG_H
#define TOPPERS_KERNEL_CFG_H

#define SPI1DMA_SEM     1

#endif /* TOPPERS_KERNEL_CFG_H */
```

PDICのSPIで使用する通信セマフォのIDを定義

TOPPERS

アプリ用インクルードファイル：lcd_text.h

- 基礎3のlcd_text1で使用したインクルードファイルからハードウェア設定と必要なプロトタイプ宣言を記載

```
/*
 *  ハードウェア設定定義
 */
#define SPISDCARD_PORTID    0

#define DMA_INTNO   DMA_INT0

#define JS_EVENT   0x8000
#define JS_PUSH     0x10
#define JS_UP       0x08
#define JS_DOWN   0x04
   :
   :
#ifndef TOPPERS_MACRO_ONLY
/*
 *  関数のプロトタイプ宣言
 */
extern void sw_handle(void);
extern int file_test(void);

#endif /* TOPPERS_MACRO_ONLY */
```


TOPPERS

インクルードファイルを追加：lcd_text.c

- PDIC/GDICドライバ用のインクルードファイルを追加

```
#include <kernel.h>
#include <stdio.h>
#include <string.h>
#include "kernel_cfg.h"
#include "device.h"
#include "spi.h"
#include "pinmode.h"
#include "adafruit_st7789.h"
#include "glcd_disp.h"
#include "lcd_text.h"


#if LCDTYPE == 1
#define LCD_WIDTH    135
#define LCD_HEIGHT   240
#else
#define LCD_WIDTH    240
#define LCD_HEIGHT   240
#endif
        :
/*
 *  GPIO設定関数
 */
static void
pinMode(uint8_t no, uint8_t mode)
```

SPIドライバのインクルードファイルを追加
ADCドライバのインクルードファイルを追加
PINMODEドライバのインクルードファイルを追加
ST7789-LCDドライバのインクルードファイルを追加
UI用のインクルードファイルを追加

LCD用の定義や関数を追加
（基礎3講座で学習済み）

TOPPERS

割込みハンドラを追加：lcd_text.c

- PDICドライバ用の割込みハンドラを追加

```
/*
 *  割込みハンドラ
 */
void
IO_BANK0_IRQHandler(void)
{
    sio_bank0_handler();
}

void
SPI0_IRQHandler(void)
{
    spi_isr(SPI1_PORTID);
}

void
DMA0_IRQHandler(void)
{
    dma_isr(DMA_INTNO);
}
```

SPI,DMAの割込みハンドラを追加し、
PDICドライバ中の割込みハンドラ関数を呼び出す

メイン関数に初期化構造体を追加

- PDIC/GDICドライバ用の初期化構造体やハンドラを追加

```
void
main(void)
{
    SPI_Init_t spi_initd;
    LCD_Handler_t   *hlcd;
    SPI_Handle_t  *hspi;
    uint32_t flgptn;
    char pstring[16];
    ER ercd;

    syslog_0(LOG_NOTICE, "Sample program starts.");

    /*
     *  疑似カーネルの初期化とATT_INIの実行
     */
    kernel_init();
    pico_system_init(0);
    led_init(0);
    switch_push_init(0);
    setup_sw_func((intptr_t)sw_handle);
```

SPIドライバの初期化構造体とハンドラ
ST7789-LCDドライバのハンドラ

main関数で使用する変数の定義

TOPPERS

GPIO設定、PDC、GDICドライバの初期化

- main関数に使用するGPIOの初期設定とドライバの初期化を追加
  - pinMode関数：GPIOの初期化
  - spi_init関数：SPIドライバの初期化
  - lcd_init関数：ST7789LCDの初期化
- 各初期化手順は基礎3講座受講済

割込み設定

- PDICドライバで使用する割込みを設定します
  - SPI割込み：INT_VECTOR_SPI0
  - DMA割込み：INT_VECTOR_DMA0+DMA_INTNO

```
    if((hspi = spi_init(SPI1_PORTID, &spi_initd)) == NULL){
        syslog_0(LOG_ERROR, "spi_start initialize error !");
        slp_tsk();
    }
#ifdef __riscv
    hazard3_set_vector(INT_VECTOR_IO_BANK0, IO_BANK0_IRQHandler);
    hazard3_set_vector(INT_VECTOR_SPI0, SPI0_IRQHandler);
    hazard3_set_vector(INT_VECTOR_DMA0+DMA_INTNO, DMA0_IRQHandler);
#endif
    init_int(INT_VECTOR_IO_BANK0, 2, true);
    init_int(INT_VECTOR_SPI0, 3, true);
    init_int(INT_VECTOR_DMA0+DMA_INTNO, 3, true);

    hlcd = &LcdHandle;
    hlcd->hspi = hspi;
    hlcd->rotation = 3;
    hlcd->spi_lock = 0  /*SPI1LOCK_SEM*/;
    hlcd->cs_pin   = CS_PORT;
```

RISC-Vで実行する場合は、ハンドラの関数を設定する必要がある

TOPPERS

ジョイスティック･スキャン関数を追加

- 基礎3講座ではジョイスティックのスキャンはタスクで行った、ベアメタル環境ではタスクは使用できないので、関数化してmain関数から定期的に呼び出すように修正

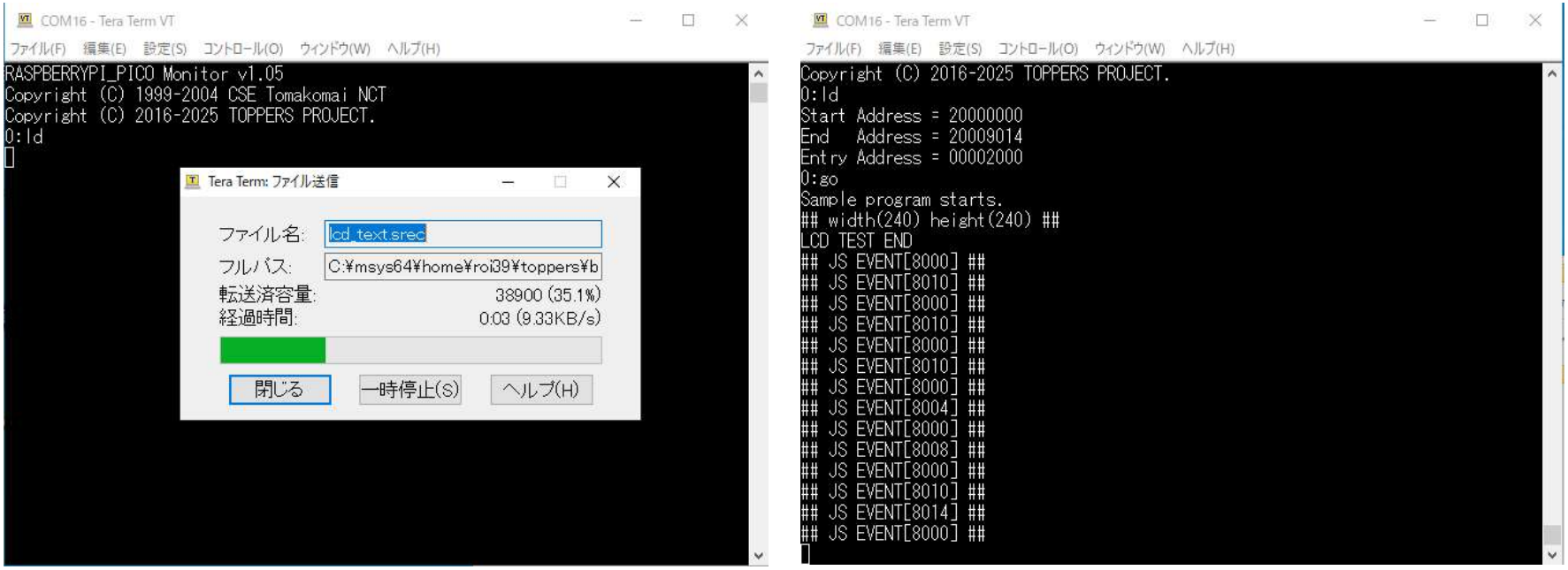
```
/*
 *  JOYSTICKセンス関数
 */
uint32_t
stick_sense(void)
{
    static uint32_t sw_save = 256;
    uint32_t sw;
    Arduino_PortControlBlock *ppin = getGpioTable(ANALOG_PIN, (PUSH_PORT-16));

    sw  = digitalRead(RIGHT_PORT);
    sw |= digitalRead(LEFT_PORT) << 1;
    sw |= digitalRead(DOWN_PORT) << 2;
    sw |= digitalRead(UP_PORT) << 3;
    sw |= ((gpioc_in_get() >> *ppin) & 1) << 4;
    sw ^= 0x1f;
    if(sw != sw_save){
        syslog_2(LOG_DEBUG, "## sw[%02x] sw_save[%02x] ##", sw, sw_save);
        sw_save = sw;
        sw |= JS_EVENT;
    }
    return sw;
}
```

関数呼び出しでも、前のデータが消えないようにstaticデータとする

TOPPERS

ダウンロード実行して、動作確認

- program/lcd_text3ディレクトリに移動し、makeコマンドでビルドする
- ビルドしたプログラムをダウンロード、実行
- LCDの表示の確認
- ジョイスティックの表示の確認



LCDJOYシールドを動かす

1. LCDJOYシールドサンプル
2. SDカードファイルを動かす
3. 漢字を表示する
4. ファイルライブラリィを動かす

ファイルシステムを追加: 目的

ミドルウェア機能を追加するプログラムの書き方を学ぶ

- メイクファイル設定，初期化，確認
  - 確認は漢字フォントを対応して表示させます
- プログラミング対象のモジュール
  - GDIC：SDカードドライバ（spi_deviver）
  - ファイルシステム：FATFS,ファイルライブラリィ（files）
  - UI：漢字ファイルフォントの対応（ui/snfont_disp）

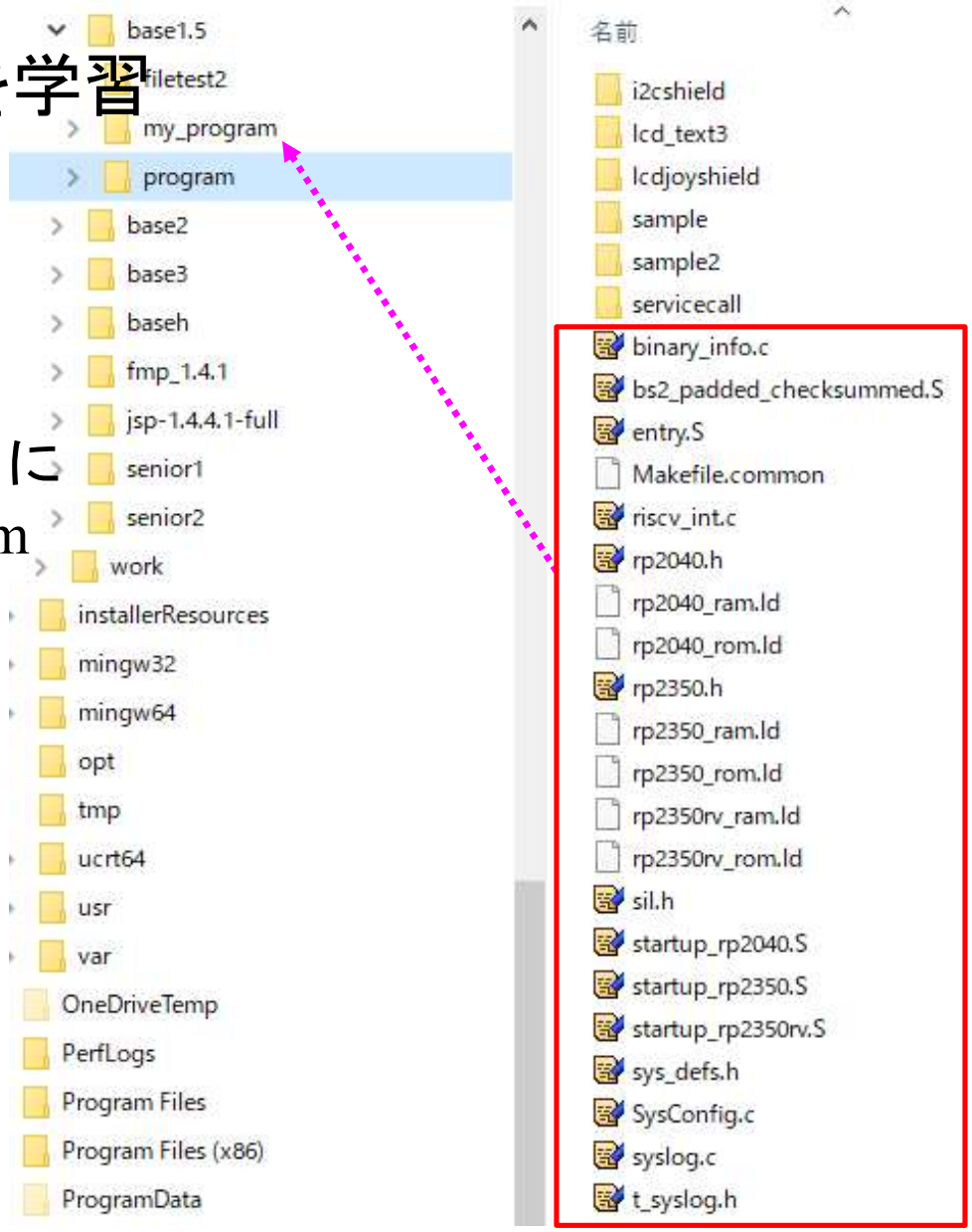
ファイルライブラリィはPCと同じようにC言語のファイル関数を使用する機能

プロジェクトの作成方法

新規プロジェクトの作成方法を学習

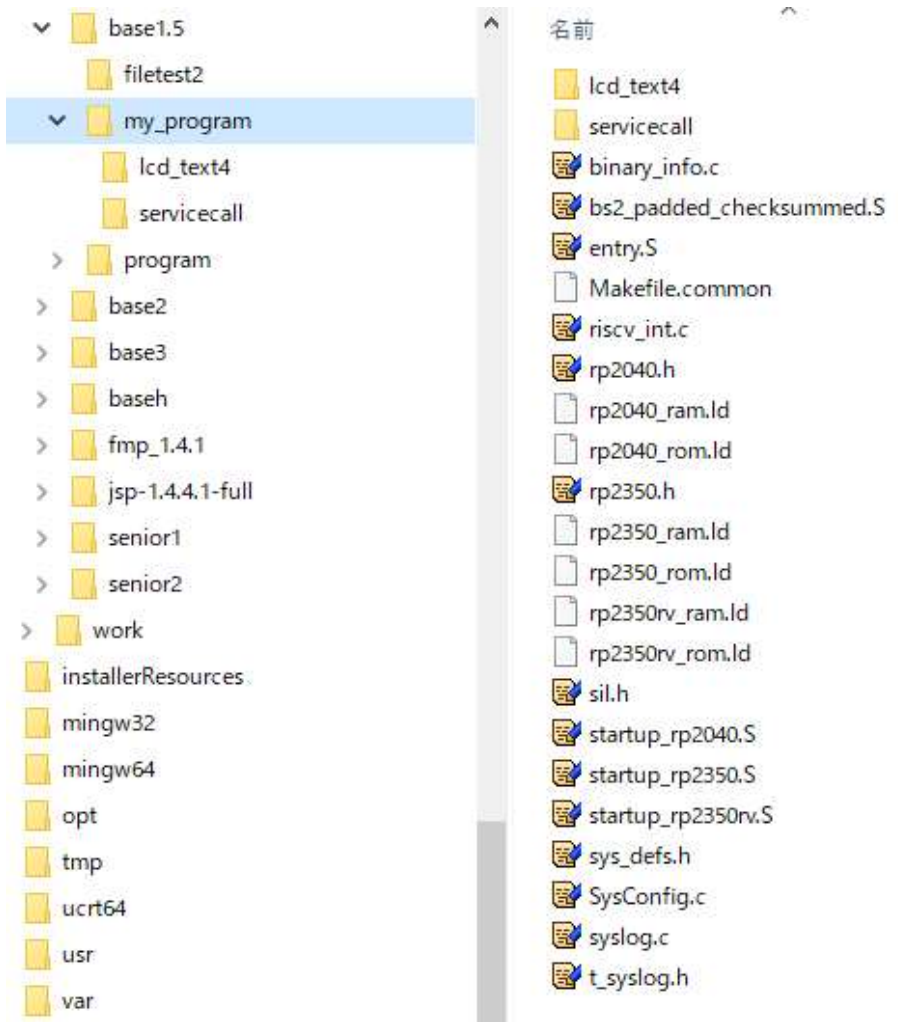
- プロジェクトのためのディレクトリ（my_program/lcd_text4）を作成
  - 作成手順は次ページ
- 教材ディレクトリ /programの直下 にある以下のファイルをmy_programフォルダにコピーする

  binary_info.c
  bs2_padding_checksummed.S
  entry.S
  Makefile.commn
  riscv_int.c
  　:
  t_syslog.h



プロジェクトの作成方法：C言語プログラム

- コンパイルするC言語プログラムを用意する
- program/lcd_text3とservicecallをmy_programにコピーする
- コピーしたlcd_text3をlcd_text4にリネームする



lcd_text4 : 作成

プログラム lcd_text4 を作成

- 手順
  - Makefileにspi_driverとファイルシステムを取り込む
  - kernel_cfg.hにセマフォIDを追加
  - メイン関数
    - インクルードファイルを追加
    - SDカードの初期化を追加
    - ファイルシステムの初期化を追加
  - 漢字ファイルフォントを有効化する:次項
  - fread関数を追加する:次項

Makefileの変更

- ファイルシステムのMakefile.configとSDカードドライバのMakefile.configをincludeで追加

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

#
#  GDIC ドライバの取り込み
#
include $(SRCDIR)/gdic/spi_driver/Makefile.config
include $(SRCDIR)/gdic/adafruit_st7789/Makefile.config
ifeq ($(BOARDTYPE),PICO_W)
include $(SRCDIR)/gdic/cyw43/Makefile.config
endif
```

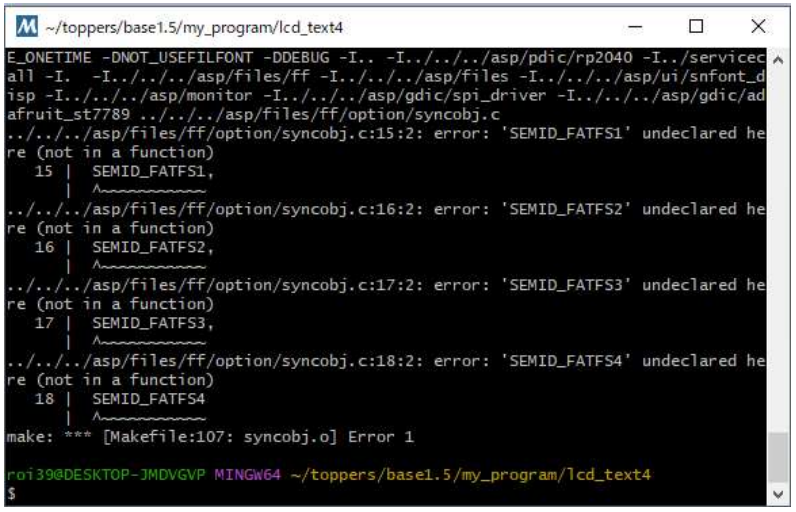
Makefileの変更

- ASPカーネルではSDカードの初期化はSDMSence_taskタスクが行ったが、関数として呼び出せるコンパイル・スイッチ：SDEV_SENSE_ONETIMEを定義します

```
ifeq ($(DBGENV),ROM)
CDEFS  := $(CDEFS) -DROM_EXEC=1 -DBOARDTYPE=$(BOARDTYPE)
                -DLOGSHIELD=$(LOGSHIELD) -DSDEV_SENSE_ONETIME -DNOT_USEFILFONT
COBJS  := $(COBJS) syslog.o
else
ifeq ($(DEBUG),1)
CDEFS  := $(CDEFS) -DTOPPERS_RAM_EXEC -DBOARDTYPE=$(BOARDTYPE)
                -DSDEV_SENSE_ONETIME
                -DLOGSHIELD=$(LOGSHIELD) -DSDEV_SENSE_ONETIME -DNOT_USEFILFONT
COBJS  := $(COBJS) syslog.o
endif
endif
```

Makefile修正後、ビルドしてみる

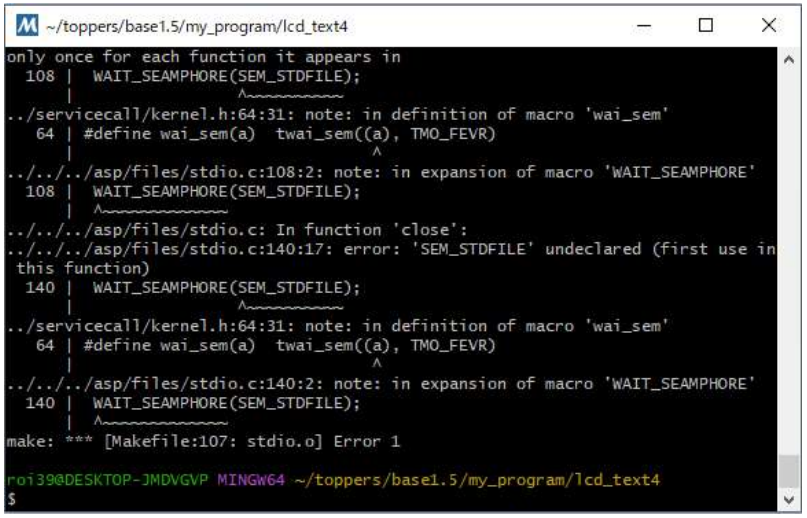
- Makefile修正後、my_program/lcd_text4ディレクトリに移動し、makeコマンドでビルドすると以下の4の排他制御用セマフォが未定義エラーとなります
- 排他制御する必要がないので4つのセマフォをLOCK_SEMIDでkernel_cfg.hに定義を追加




```
#ifndef TOPPERS_KERNEL_CFG_H
#define TOPPERS_KERNEL_CFG_H

#define SPI1DMA_SEM   1

#define SEMID_FATFS1      LOCK_SEMID
#define SEMID_FATFS2      LOCK_SEMID
#define SEMID_FATFS3      LOCK_SEMID
#define SEMID_FATFS4      LOCK_SEMID

#endif /* TOPPERS_KERNEL_CFG_H */
```

kernel_cfg.hへ排他制御セマフォIDを追加

- ４つのセマフォ追加後、再度ビルドするとSEM_STDFILEセマフォが未定義エラーとなります
- SEM_STDFILEもファイルシステム用の排他制御セマフォなので、LOCK_SEMIDでkernel_cfg.hに定義を追加
- 再ビルドするとエラーは発生しません（ソースファイルのみリンクできた）




```
#ifndef TOPPERS_KERNEL_CFG_H
#define TOPPERS_KERNEL_CFG_H

#define SPI1DMA_SEM   1

#define SEMID_FATFS1      LOCK_SEMID
#define SEMID_FATFS2      LOCK_SEMID
#define SEMID_FATFS3      LOCK_SEMID
#define SEMID_FATFS4      LOCK_SEMID
#define SEM_STDFILE       LOCK_SEMID

#endif /* TOPPERS_KERNEL_CFG_H */
```

メイン・ソースの変更：インクルードファイル追加

- ファイルシステムの制御に必要なインクルードファイルを追加

```
/*
 *  LCD-JOYシールド制御プログラムの本体(RP2040用)
 *  adafluit 1.54"TFT breakoutを使用する場合、LOGSHIELD=0
 *  log shield+秋月1.54"を使用する場合、LOGSHIELD=1
 */
#include <kernel.h>
#include <stdio.h>
#include <string.h>
#include "kernel_cfg.h"
#include "device.h"
#include "spi.h"
#include "pinmode.h"
#include "spi_driver.h"
#include "adafruit_st7789.h"
#include "glcd_disp.h"
#include "storagedevice.h"
#include "sddiskio.h"
#include "lcd_text.h"
```

SDカードドライバ用インクルードファイル

ファイルシステム用インクルードファイル

TOPPERS

メイン・ソースの変更：SDカード初期化構造体

- データ領域の追加
  - StorageDevice_t *psdev：ストレージデバイスへのポインタ
  - SDCARD_Handler *hsd：SDカードデバイスハンドラのポインタ

```
void
main(void)
{
   StorageDevice_t *psdev;
   SPI_Init_t spi_initd;
   SDCARD_Handler_t *hsd = NULL;
   LCD_Handler_t   *hlcd;
   SPI_Handle_t  *hspi;
   uint32_t flgptn;
   char pstring[16];
```


TOPPERS

メイン・ソースの変更（3）：SDカード初期設定追加

- SDカード用の初期設定関数を呼び出す
  - sdev_init(0);　　ストレージデバイス・マネージャの初期化
  - stdfile_init(0);　　POSIXファイル関数の初期化
  - sd_init(0);　　SDカードドライバの初期化

```
/*
 *  疑似カーネルの初期化とATT_INIの実行
 */
kernel_init();
pico_system_init(0);
led_init(0);
switch_push_init(0);
sdev_init(0);
stdfile_init(0);
sd_init(0);
setup_sw_func((intptr_t)sw_handle);
```


TOPPERS

メイン・ソースの変更（4）：最初のSPI転送速度変更

- SDカードにSPI通信を指定するSPIのクロックを1MHzに変更

```
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;
spi_initd.DMAIntNo      = DMA_INTNO;
spi_initd.DMARxChannel  = 1;
spi_initd.DMATxChannel  = 0;
spi_initd.Mode          = SPI_MODE_MASTER;
spi_initd.semid         = SPI1DMA_SEM;
spi_initd.semlock       = 0;

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


TOPPERS

メイン・ソースの変更（5）

- SDカードドライバの初期化を行い、SPIの通信クロックを10MHzに再設定する

```
init_int(INT_VECTOR_IO_BANK0, 2, true);
init_int(INT_VECTOR_ADC_FIFO, 1, true);
init_int(INT_VECTOR_SPI0, 3, true);
init_int(INT_VECTOR_DMA0+DMA_INTNO, 3, true);

cs_set(PORT_HIGH);
if(sdcard_start(SPISDCARD_PORTID, hspi, CS2_PORT) != E_OK){
    syslog_0(LOG_ERROR, "sdcard_start initialize error !");
}
spi_deinit(hspi);
dly_tsk(100);
spi_initd.Baudrate      = 10*1000*1000;
hspi = spi_init(SPI1_PORTID, &spi_initd);
sdcard_setspi(SPISDCARD_PORTID, hspi);
```

メイン・ソースの変更（6）

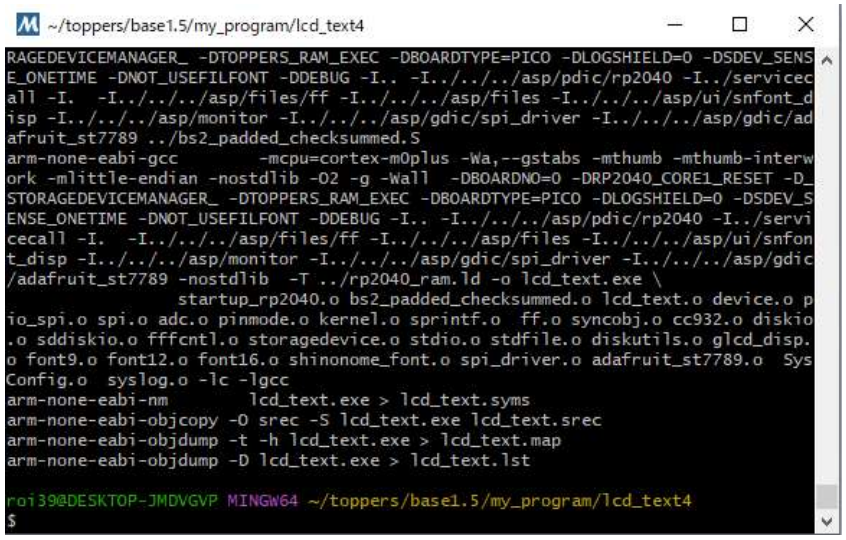
- SDMSence_task関数を読み出し、SDカードの初期化通信を行います
- psdev->sdev_local[1]にSDカードドライバのハンドラがセットされていれば、成功

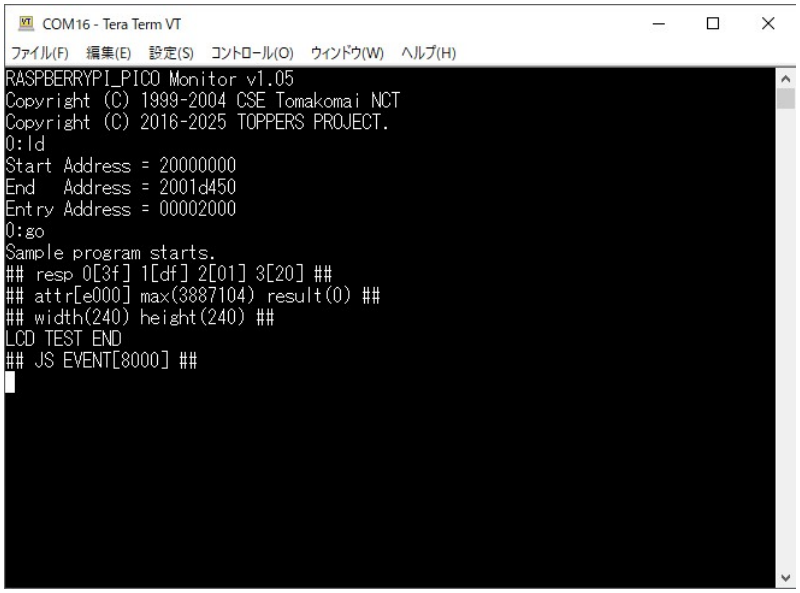
```
/*
 *  SD-CARDの設定
 */
SDMSence_task(0);
psdev = SDMGetStorageDevice(SDCARD_DEVNO);
hsd = (SDCARD_Handler_t *)psdev->_sdev_local[1];
if(hsd == NULL){
    syslog_0(LOG_ERROR, "sdcard initial error !");
}
(void)(hsd);

hlcd = &LcdHandle;
hlcd->hspi = hspi;
hlcd->rotation = 3;
hlcd->spi_lock = 0  /*SPI1LOCK_SEM*/;
```

この状態で、ビルド実行

- 正しく変更できれば、正常にビルドできるはず
- lcd_text.srecをダウンロード実行してみる
- 実行ログに「## attr[e000] max(3887104) result(0) ##」が表示されれば、正しくSDカードの初期化が行われた
- タスクモニタがないため、簡単な初期化確認はできない

max(3887104)はSDカードのセクタ数なので、カードにより値は変わる





TOPPERS

LCDJOYシールドを動かす

1. LCDJOYシールドサンプル
2. SDカードファイルを動かす
3. 漢字を表示する
4. ファイルライブラリィを動かす

lcd_text4 : LCDに漢字を表示する

プログラム lcd_text4 を改造

- 手順
  - 漢字ファイルフォントを有効化する
  - fread関数を実装する
    - 問題点の説明
    - 対応プログラムを組込みプラットフォームから探す
    - 最適なfread関数を実装

Makefileの変更：漢字フォントファイルを有効にする

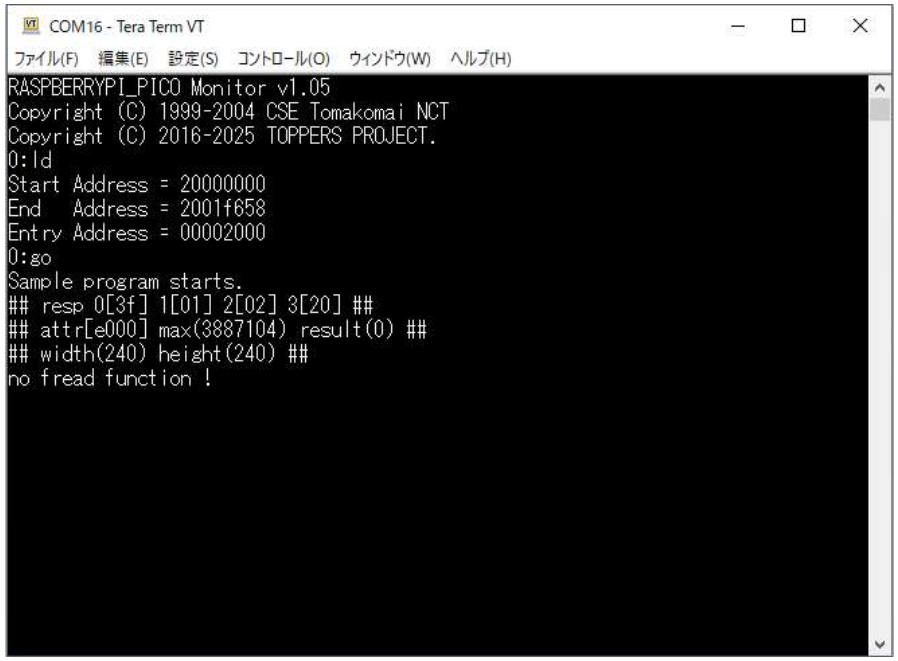
- Makefile中の-DNOT_USEFILFONT定義を
  -DMAX_FONT_IMAGE=24に
  書き換えればォントファイルは有効となる
- 漢字用のUTF-8Nコードはlcd_text.cのstring1/2のコンパイルスイッチで切り替わります

```
ifeq ($(DBGENV),ROM)
CDEFS  := $(CDEFS) -DROM_EXEC=1 -DBOARDTYPE=$(BOARDTYPE)
                -DLOGSHIELD=$(LOGSHIELD) -DSDEV_SENSE_ONETIME
                -DMAX_FONT_IMAGE=24
COBJS  := $(COBJS) syslog.o
else
ifeq ($(DEBUG),1)
CDEFS  := $(CDEFS) -DTOPPERS_RAM_EXEC -DBOARDTYPE=$(BOARDTYPE)
                -DLOGSHIELD=$(LOGSHIELD) -DSDEV_SENSE_ONETIME
                -DMAX_FONT_IMAGE=24
COBJS  := $(COBJS) syslog.o
endif
endif
```

この状態で、ビルド実行

- ダウンロード、実行すると「no fread function !」を表示して停止
- ファイル上のフォントはfread関数を用いて読み出す
- TOPPERS BASE PLATFORM上のfread関数を取り込んでいない
- 仮のfread関数がdiskutils.cにあり、エラーを表示して停止している

```
/*
 *  標準データ読み出し関数
 */
__attribute__((weak)) size_t
fread(void *buf, size_t size, size_t count, FILE *st)
{
    sys_printf("no fread function !¥n");
    slp_tsk();
    return 0;
}
```



fread関数を追加

- TOPPERS BASE PLATFORMでは標準入出力機能でfreadを提供しています（monitor/stdio/stddev.c）
- fread関数は標準入出力機能に対応しているが、ベアメタル環境では標準入出力機能を取り込めないので同等の関数を用意する必要があります

stddev.c中のfreadのソース

```
/*
 *  標準データ読み出し関数
 */
size_t fread(void *buf, size_t size, size_t count, FILE *st)
{
    int result;
    if(st == NULL)
        st = stdin;
    result = st->_func_ins(st, size*count, buf);
    return result / size;
}
```

標準入出力機能
stがNULLの場合、標準入力を設定する

TOPPERS

fread関数を追加

- diskutils.cのにfread関数を標準入出力機能に対応しない形で修正します

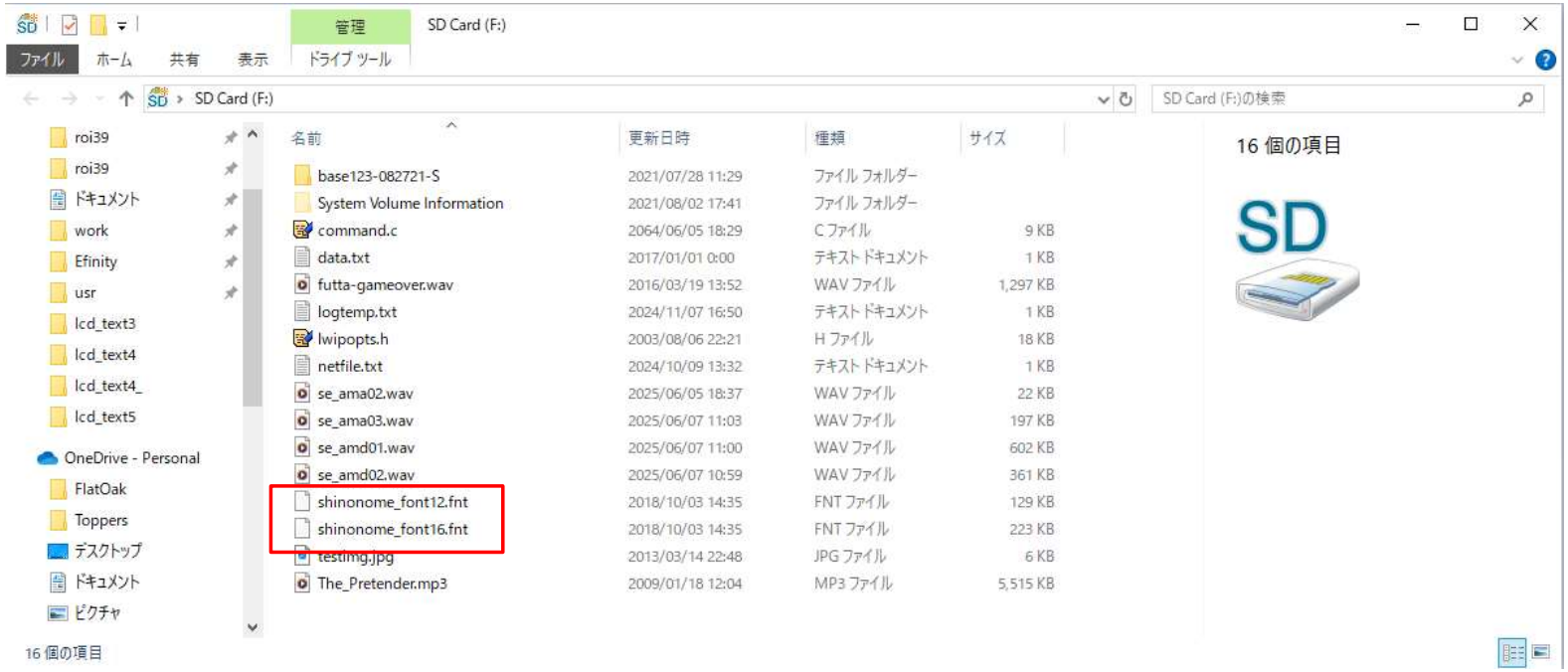
標準入出力機能を取り去ったfread関数

```
/*
 *  標準データ読み出し関数
 */
__attribute__((weak)) size_t
size_t fread(void *buf, size_t size, size_t count, FILE *st)
{
    int result = st->_func_ins(st, size*count, buf);
    return result / size;
}
```


TOPPERS

SDカードのフォントイメージを確認

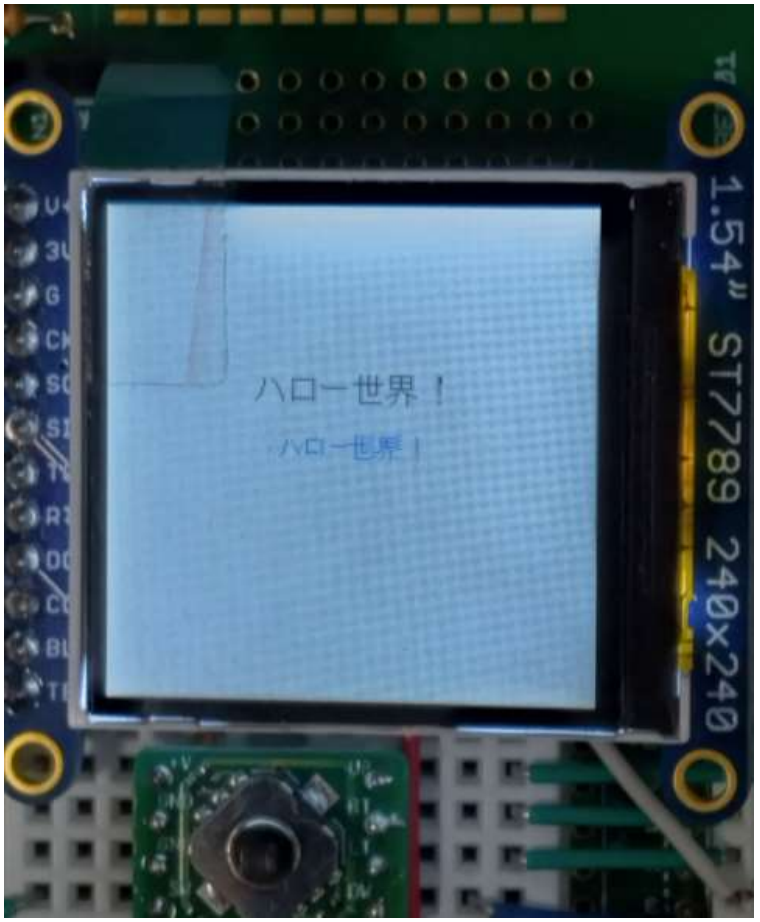
- SDカード中に東雲漢字フォントのイメージファイルを確認
  - shinonome_font12.fnt　東雲12x12フォントイメージファイル
  - shinonome_font16.fnt　東雲16x16フォントイメージファイル
- ない場合はasp/ui/snfont_dispディレクトリにありますのでSDカードにコピーしてください



TOPPERS

この状態で、ビルド実行

- 再度、ビルド、ダウンロード実行すると「Hello Wolrd !」が「ハロー世界！」の表示に変わります



LCDJOYシールドを動かす

1. LCDJOYシールドサンプル
2. SDカードファイルを動かす
3. 漢字を表示する
4. ファイルライブラリィを動かす

lcd_text5 : 演習

プログラム lcd_text5 の説明

- 機能
  - 基礎3のPC互換のファイル操作file_test.cを対応してみる
  - メイン関数
    - push switchを押すと、file_test関数を実行するように修正
  - Makefileにfile_test.cを追加
    - file_test.cは基礎3で使用したC言語のファイル関数を使用してファイルをテストするプログラム、実行関数はfile_test();
  - 発生した問題点の解説と改造を行います

lcd_text5 : 作成

プログラム lcd_text5 を作成

- 手順
  - lcd_text4をコピーしてlcd_text5を作成
  - filetestディレクトリ中のfile_test.cをlcd_text5にコピー
  - Makefileを修正しfile_test.cを取り込む
  - メイン関数
    - push switchを押すとfile_test()関数を実行するように修正
  - ビルド、ダウンロード実行して問題点を解析
  - diskutils.cを改造します
    - fwrite/fgetc

Makefileの変更：file_test.cをビルドファイルに追加

- Makefile中のCOBJSにfile_test.oを追加する

```
TARGET_DIR = .
UTASK_DIR = $(ROOT_PATH)
vpath %.c $(UTASK_DIR):$(RTOS_PATH)/pdic/$(CORE):$(SYSSVC_DIR)
            :$(ROOT_PATH)/servicecall:$(SRCDIR)/monitor/stdio
vpath %.S $(UTASK_DIR)

COBJS = $(OBJNAME).o file_test.o device.o pio_spi.o spi.o pinmode.o
            kernel.o sprintf.o $(SYSSVC_COBJS) $(LOCAL_COBJS)
AOBJS = $(LOCAL_AOBJS)
```

メイン関数でのfile_testの対応

- ジョイスティックのセンス前に「READY test start [push switch] !」を表示
- PUSHスイッチを押すと、file_test関数を実行するように修正

```
syslog_0(LOG_NOTICE, "READY test start [push switch] !");
 while(1){
    flgptn = stick_sense();
    if(flgptn != 0){
       syslog_1(LOG_NOTICE, "## JS EVENT[%04x] ##", flgptn);
       flgptn &= ~JS_EVENT;
       sprintf(pstring, "sw=%02x", (uint8_t)flgptn);
       DrawProp.pFont = &Font16;
       if(flgptn == 0)
          lcd_DisplayStringAt(&DrawProp, 0, 140, (uint8_t *)"      ", CENTER_MODE);
       else
          lcd_DisplayStringAt(&DrawProp, 0, 140, (uint8_t *)pstring, CENTER_MODE);
       if((flgptn & JS_PUSH) != 0){
          int result = file_test();
          if(result != 0)
             syslog_1(LOG_ERROR, "sd_test error(%d) !", result);
          dly_tsk(2000);
       }
    }
```


TOPPERS

fwrite関数、fgetcを追加

- file_test.cでは、fwriteとfgetcのファイルライブラリィ関数を使用しています
- monitor/stdio/stddev.cを確認して、標準入出力機能を削除したプログラムに修正します

diskutils.c中の対応関数を修正

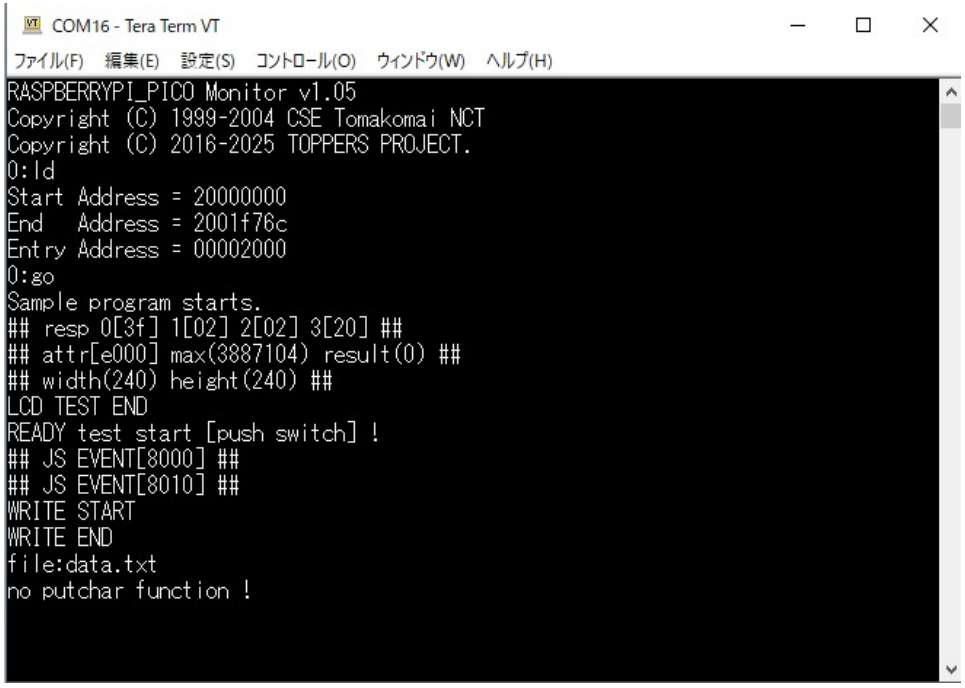
```
/*
 * 標準データ書き込み関数
 */
__attribute__((weak)) size_t
fwrite(const void *buf, size_t size, size_t count, FILE *st)
{
    int result = st->_func_outs(st, size*count, (char *)buf);
    return result / size;
}
/*
 * 標準入力関数
 */
__attribute__((weak)) int
fgetc(FILE *st)
{
    return st->_func_in(st);
}
```


TOPPERS

この状態で、ビルド実行

- ダウンロード、実行する後、push switchを押すと、「no putchar function ! 」を表示して停止
- file_test.cでは、標準出力関数putcharを使用しています
- TOPPERS BASE PLATFORM上のputchar関数を取り込んでいない
- 仮のputchar関数がdiskutils.cにあり、エラーを表示して停止しています

```
/*
 * 一文字出力プログラム
 */
__attribute__((weak)) int
putchar(int c)
{
    sys_printf("no putchar function !¥n");
    slp_tsk();
    return 1;
}
```



putchar関数を修正

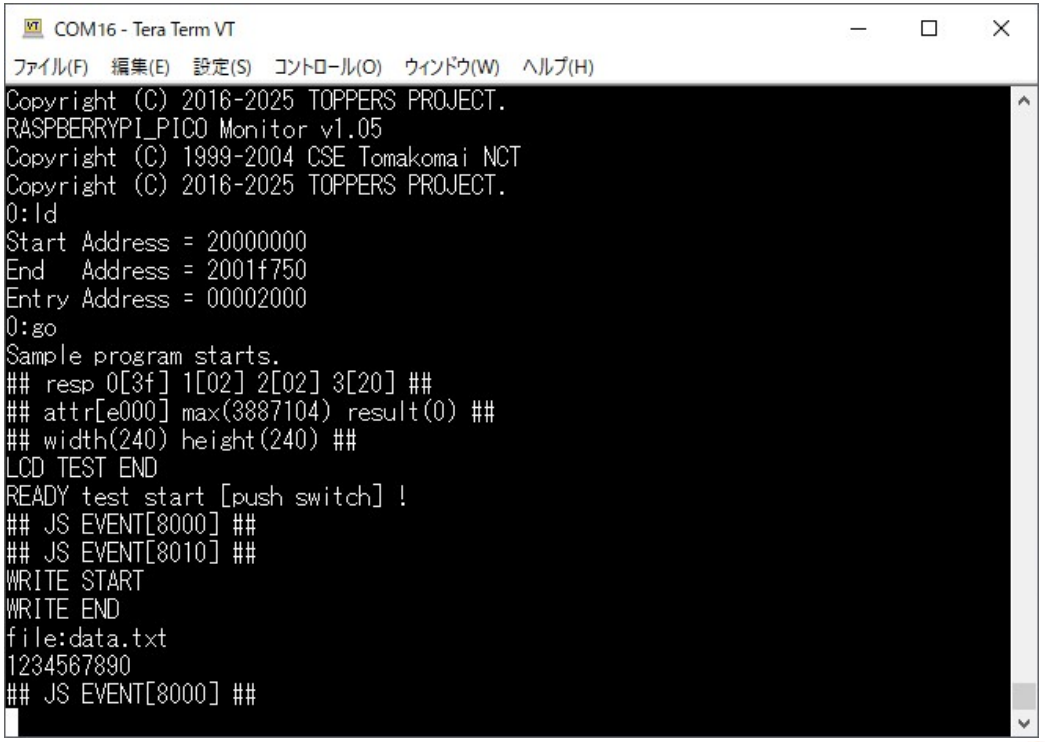
- diskutils.cのにputchar関数をベアメタル環境で使用しているuart出力関数：sys_putcを使って書き換える

```
/*
 * 一文字出力プログラム
 */
__attribute__((weak)) int
putchar(int c)
{
    sys_putc(c);
    if(c == '¥n')
        sys_putc('¥r');
        return 1;
}
```


TOPPERS

再度、ビルド実行

- ダウンロード、実行する後、push switchを押すと、正しくファイルの書き込み、読み出しが行われる
- 組込みソフトウェアプラットフォームでは、標準入出力やファイルライブラリィを検証して実装しているが、ベアメタル開発では、管理できない機能を拡張するのは得策とは言えない



TOPPERS

ファイルライブラリィを削除し、最適化する

- file_test.cをファイルライブラリィではなく、FATFsの関数で作り直したプログラムfile_test2.cをbase.1.5/filetestに置く
- プログラミングにはFATFsの仕様を良く理解する必要がある

```
#include "sys_defs.h"
#include <string.h>
#include "ff.h"

static const char filename[]  = "0:data.txt";
static const char test_data[] = "1234567890¥r¥n";

int putchar(int c)
{
    sys_putc(c);
    if(c == '¥n')
        sys_putc('¥r');
    return 1;
}
int file_test(void)
{
    static FIL FileObj;
    FRESULT result;
    signed char c;
    UINT len;

    printf("WRITE START¥n");
```

ベアメタル環境とFATFsのインクルードファイルに変更

putchar関数に変わる関数を用意する

FATFsのファイルデータと返り値変数を用意

TOPPERS

FATFsの仕様を理解すれば、簡単に書き換え可能

```
    result = f_open(&FileObj, filename, FA_WRITE | FA_OPEN_ALWAYS);
    if(result != FR_OK){
        printf("fopen error !¥n");
        return -1;
    }
    result = f_write(&FileObj, test_data, sizeof(test_data)-1, &len);
    if(result != FR_OK || len != sizeof(test_data)-1){
        printf("f_write error(%d) !¥n", result);
        f_close(&FileObj);
        return -1;
    }
    f_close(&FileObj);
    printf("WRITE END¥n");
    result = f_open(&FileObj, filename, FA_READ);
    if(result != FR_OK){
        printf("fopen error !¥n");
        return -1;
    }
    printf("file:%s¥n", filename);
    while((f_read(&FileObj, &c, 1, &len)) == FR_OK){
        if(len != 1)
            break;
        putchar(c);
    }
    f_close(&FileObj);
    return 0;
}
```

fopenをFATFsのf_openに変更

fwriteをf_writeに変更

fcloseをf_closeに変更

ファイルライブラリィを削除の結果

- file_test2.cをそのまま動作させるプログラムを base1.5/program/lcd_text6に置いています
- 漢字フォントファイル対応で必要なファイルライブラリィを対応したプログラムはdiskutils.cに記載しています
- Makefileとdiskutils.cは、各自確認をお願いします
- PicoのROM実行版でlcd_text5とlcd_text6のRAMサイズを比較すると半分以下となった、ベアメタル開発ではSDKのドライバより上位のプログラムは、その仕様に基づき使用する方が良い
  通常のワンチップCPUはRAMサイズが小さいものが多い

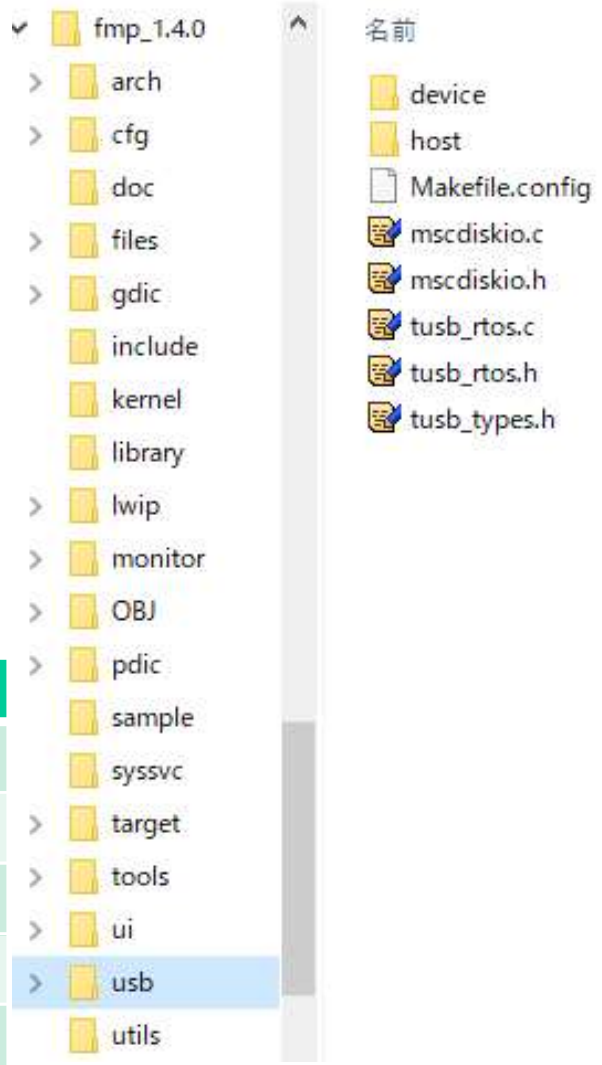
セクション	lcd_text5（バイト）	lcd_text6（バイト）	比率（%）
.text	41,076	38,440	93.58
.rodata	81,185	81,157	99.97
.data	3,084	3,084	100.0
.bss	14,994	5,280	35.21
ROM	125,345	122,681	97.87
RAM	18,078	8,368	46.27

スティック・マウスを作る

1. USBデバイスHIDについて
2. スティック・マウスの仕様
3. プログラムの作成
4. 問題点の対応

USBミドルウェアの配置

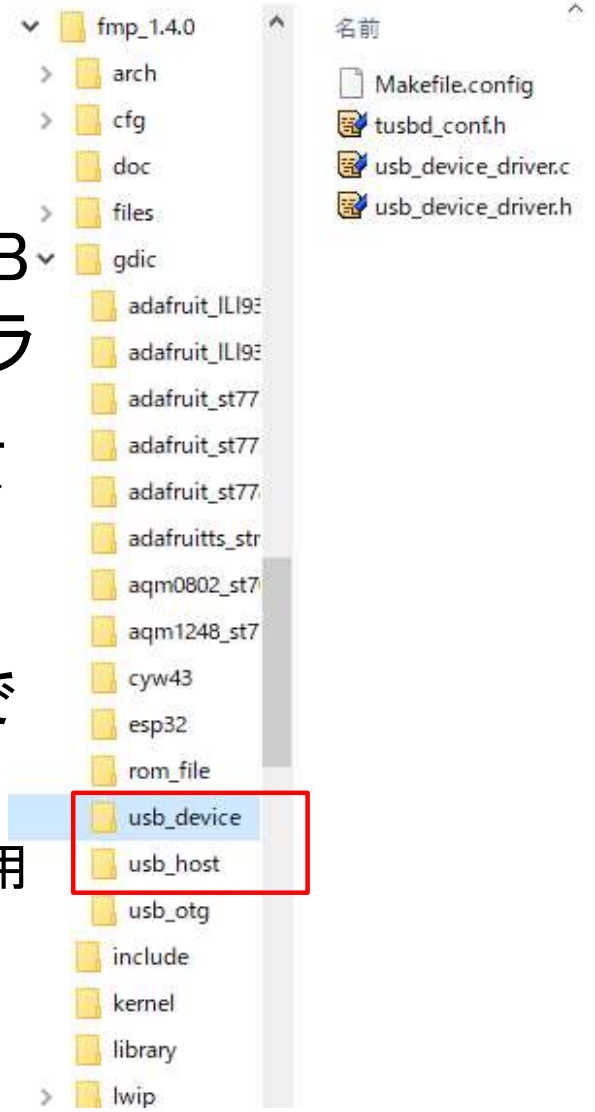
- TOPPERS BASE PLATFORMのベースディレクトリ上のusbディレクトリに配置
  - device以下にUSBデバイスクラス
  - host以下にUSBホストクラス
- 直下に共用ファイルを置く



ファイル名	内容
Makefile.config	USBミドルウェアのMakefileインクルード
mscdiskio.c	FatFS用MSC用デバイスドライバソース
mscdiskio.h	FatFS用MAC用デバイスドライバインクルード
tusb_rtos.c	ミドルウェア用RTOSインターフェイスソース
tusb_rtos.h	ミドルウェア用RTOSインターフェイスインクルード
tusb_types.h	ミドルウェアタイプ宣言インクルード

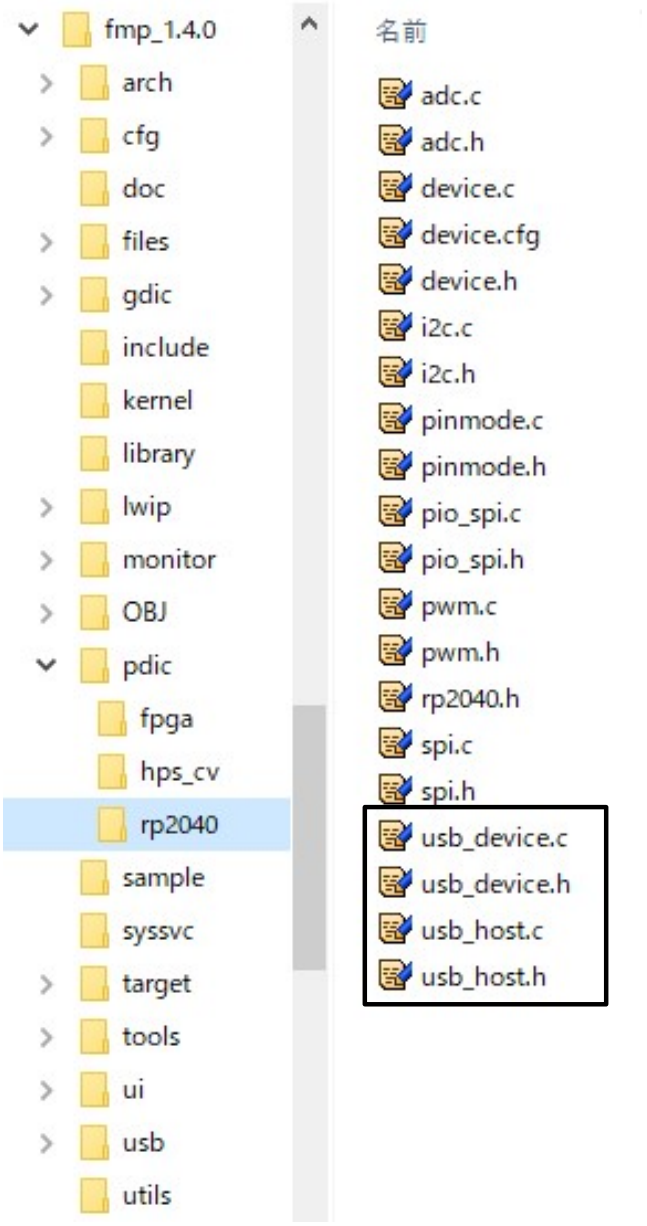
USB用GDICドライバ

- USBミドルウェアは、RTOSやCPUに依存しない
- SoC毎にUSBコアは異なります。USBコアに依存した部分はUSB-PDICドライバに置いています。USBミドルウェアとUSB-PDICドライバを連結するドライバがGDICドライバとなります
- TOPPERS BASE PLATFORM(RP)では、2つのドライバを用意している
  - usb_device:RP2040/RP2350-USBデバイス用
  - usb_host:RP2040/RP2350-USBホスト用



Raspberry Pi Pico用USB PDICドライバ

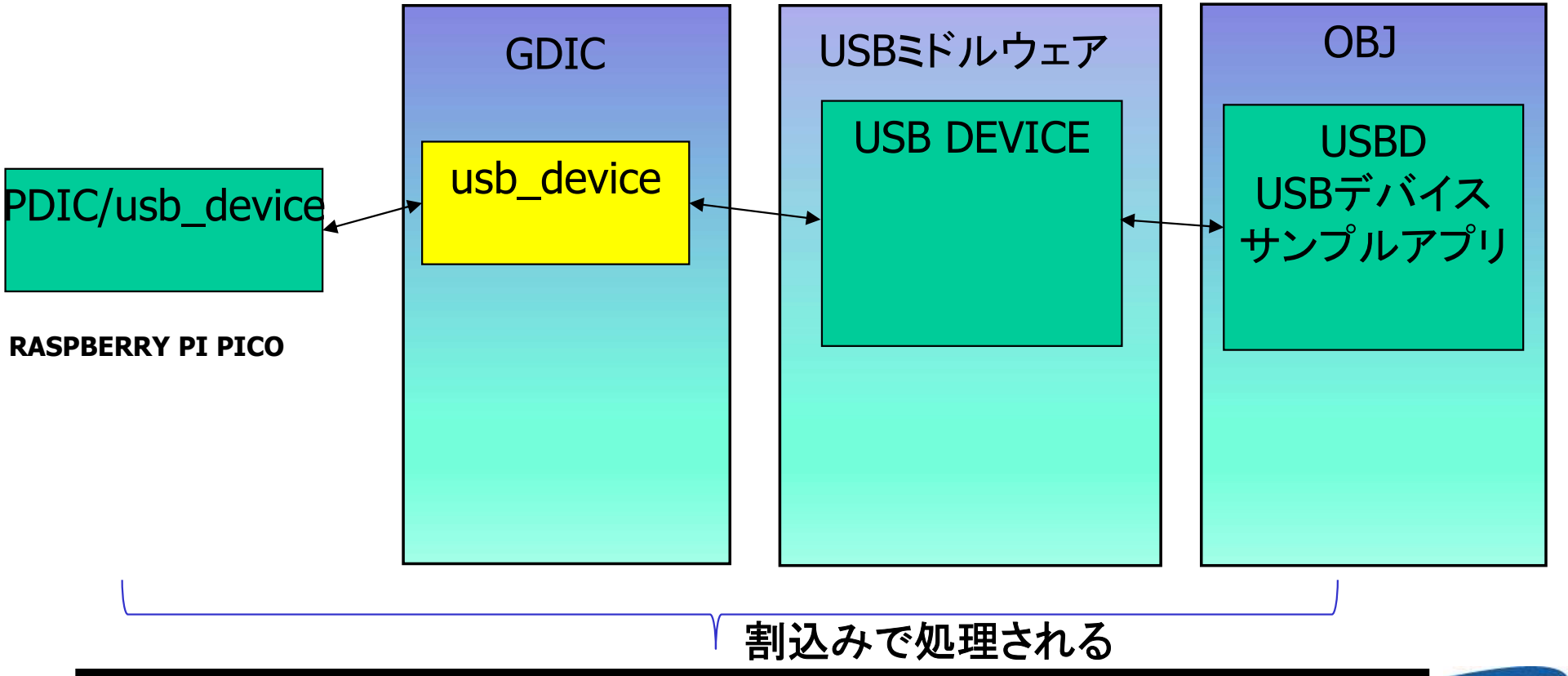
- Raspberry Pi PicoはUSBとしてUSBホスト、デバイス機能を持つ、PDICドライバはpdic/rp2040またはrp2350中の2つのドライバがPDICドライバであります
  - usb_device.c
  - usb_device.h
  - usb_host.c
  - usb_host.h
- ホスト、デバイスの機能は同時実行不可



TOPPERS BASE PLATFORM USBモジュール

- USBモジュールは3つに分かれています
  - TOPPERS USBミドルウェア
  - GDIC/usb_device
  - USBデバイスドライバ（PDIC/usb_device）

Makefileのインクルード
コンフィギュレーション
のインクルードを使って
結合



スティック・マウスを作る

1. USBデバイスHIDについて
2. スティック・マウスの仕様
3. プログラムの作成
4. 問題点の対応

スティックマウス仕様

- ジョイスティックとPUSH SWITCHを使ってマウスの操作を行う
- HIDのマウスデータ4バイト構成で以下のレイアウト
  - 0バイト:ビットでボタンの状態
  - 1バイト:X座標の移動量
  - 2バイト:Y座標の移動量
  - 3バイト:0で未使用

ボタンとスティック	定義名	動作
ジョイスティックPUSH	JS_PUSH	マウス第1ボタン
TEB003 PUSH SWITCH	PUSH_SWITCH	マウス第2ボタン
ジョイスティックUP	JS_UP	カーソルを上移動
ジョイスティックDOWN	JS_DOWN	カーソルを下移動
ジョイスティックLEFT	JS_LEFT	カーソルを左移動
ジョイスティックRIGHT	JS_RIGHT	カーソルを右移動

スティックマウス仕様

- PCとのコネクション完了後、以下の関数でHIDのマウスデータ4バイトデータを送信します
  - USBD_HID_SendReport
  - この関数は割込みハンドラから使用可能
- 送信の最終結果はUSBへのリプライで割込みにより返されます

スティック・マウスを作る

1. USBデバイスHIDについて
2. スティック・マウスの仕様
3. プログラムの作成
4. 問題点の対応

プロジェクトの作成方法：C言語プログラム

- コンパイルするC言語プログラムを用意する
- program/lcd_text3をmy_programにコピーする
- コピーしたlcd_text3をstickmouse1にリネームする
- lcd_text.cとlcd_text.hをstickmouse.cとstickmouse.hにリネームする
- Makefile中のOBJNAMEを変更
  - lcd_text → stickmouse
- stickmouse.cのインクルードファイルを変更
  - 51行目のlcd_text.h → stickmouse.h

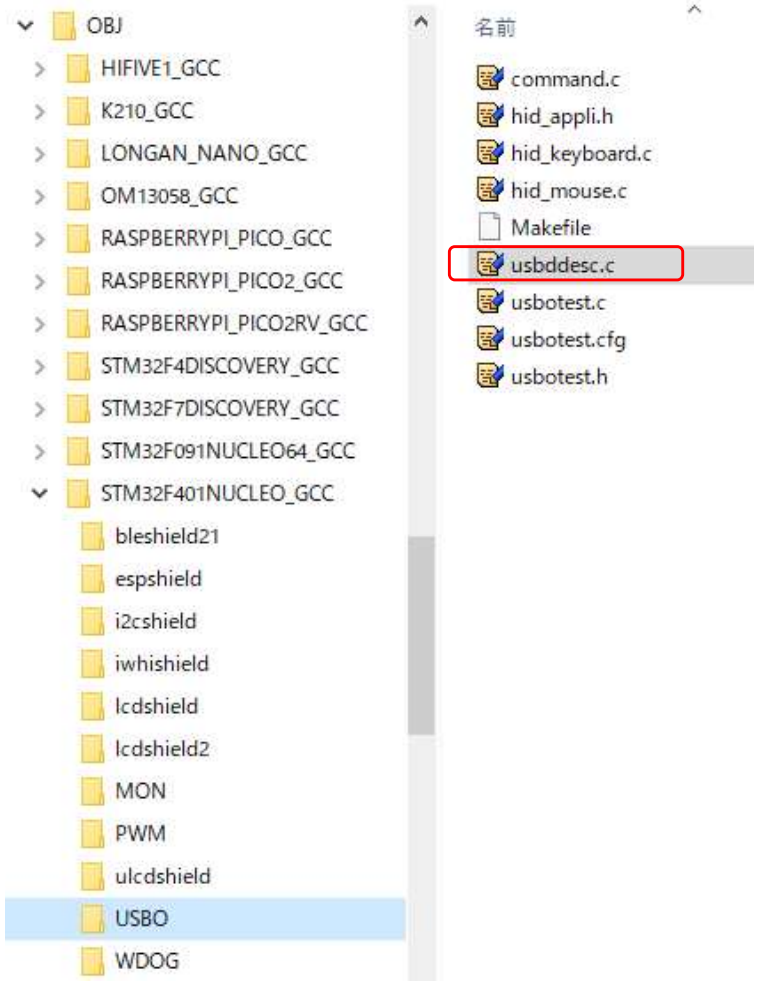
stickmouse1：作成

プログラム stickmouse1を作成

- 手順
  - デバイス・ディスクリプタ定義ファイルを取り込む
  - MakefileにUSBデバイス用ソースを取り込む
  - インクルードファイル
    - PUSH SWITCHの定義を追加
  - メイン関数
    - インクルードファイルを追加
    - USBハンドラや変数を追加
    - マウスの位置やボタン設定関数を作成
    - 第2ボタンの情報取得を追加
    - USBデバイスHIDの初期化を追加
    - 位置やボタン関数の呼び出しを追加

デバイス・ディスクリプタ定義ファイルをコピー

- PCにUSBデバイスの属性を通知するための属性を定義したファイルが必要となります
- STマイクロエレクトロニクス社のF401ボードにUSB-HIDとCIDの属性を定義したusbddesc.cファイルがあるのでこれを流用します
- usbddesc.cをstickmouseディレクトリにコピーします



TOPPERS

デバイス・ディスクリプタ定義usbddesc.c

- usbddesc.cには、デバイス・ディスクリプタの定義が記載されいます

```
#include "tusbd_base.h"

#define MANUFACTURE_LENGTH          64
#define PRODUCT_LENGTH              32
#define CONFIGURATION_LENGTH        32
#define INTERFACE_LENGTH            32

/* Private define STM defintion -----------------------------------------------------*/
#if USB_DEVICE_ID == 0
#define USBD_VID                    0x0000
#define USBD_PID                    0x0001
#define USBD_RELNO                  0x0110
#define USBD_MANUFACTURER_STRING    "TOPPERS"
#else
#define USBD_VID                    0x0483
#define USBD_PID                    0x5740
#define USBD_RELNO                  0x0200
#define USBD_MANUFACTURER_STRING    "STMicroelectronics"
#endif
#if USB_DEVICE_ID == 0
#define USBD_PRODUCT_STRING         "HID Mouse"
#define USBD_CONFIGURATION_HS_STRING  "HID Config"
#define USBD_INTERFACE_HS_STRING    "HID Interface"
#define USBD_CONFIGURATION_FS_STRING  "HID Config"
#define USBD_INTERFACE_FS_STRING    "HID Interface"
```

USB_DEVICE_IDが0ならばHID用のディスクリプタを作成する

TOPPERS

MakefileにUSBデバイスのソースを追加

- ミドルウェアにUSBモジュールを追加
- GDICにusb_deviceを追加する

```
#
# ミドルウェアの Makefile のインクルード
#
include $(SRCDIR)/usb/Makefile.config
include $(SRCDIR)/ui/snfont_disp/Makefile.config

#
# GDIC ドライバの取り込み
#
include $(SRCDIR)/gdic/usb_device/Makefile.config
include $(SRCDIR)/gdic/adafruit_st7789/Makefile.config
ifeq ($(BOARDTYPE),PICO_W)
include $(SRCDIR)/gdic/cyw43/Makefile.config
endif
```

MakefileにUSBデバイスHIDのソースを追加

- USBモジュール中のUSBデバイスHIDのソースを追加
- HIDのデバイス定義USB_DEVICE_ID=0も定義

```
#
#  GDICドライバの取り込み
#
include $(SRCDIR)/gdic/usb_device/Makefile.config
include $(SRCDIR)/gdic/adafruit_st7789/Makefile.config
ifeq ($(BOARDTYPE),PICO_W)
include $(SRCDIR)/gdic/cyw43/Makefile.config
endif

#
#  USB DEVICE CLASSの追加
#
SYSSVC_COBJS := $(SYSSVC_COBJS) tusbd_hid.o
INCLUDES := $(INCLUDES) -I$(SRCDIR)/usb/device/HID
CDEFS := $(CDEFS) -DUSB_DEVICE_ID=0
```

Makefileに必要なドライバを追加

- ビルド用のファイルにデバイス・ディスクリプタ定義とUSBデバイスのドライバを追加
- ここでmakeコマンドでビルドを行い、正しくビルドされることを確認します

```
vpath %.S $(UTASK_DIR)

COBJS = $(OBJNAME).o usbddesc.o device.o pio_spi.o usb_device.o spi.o
                pinmode.o kernel.o sprintf.o $(SYSSVC_COBJS) $(LOCAL_COBJS)
AOBJS = $(LOCAL_AOBJS)

ifeq ($(DBGENV),ROM)
CDEFS  := $(CDEFS) -DROM_EXEC=1 -DBOARDTYPE=$(BOARDTYPE)
                -DLOGSHIELD=$(LOGSHIELD) -DNOT_USEFILFONT
COBJS  := $(COBJS) syslog.o
```

stickmouse.h：PUSH SWITCHの定義を追加

- TEB003上のPUSH SWITCHをマウスの第2ボタンとして使用するので、GPIOの定義を追加

```
#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 PUSH_PORT   (16+3)

#define PSW_PIN     22

#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)
```

PUSH SWITCHはGPIO22に接続

TOPPERS

stickmouse.c：インクルードファイル等を追加

- USBデバイスの制御を行うインクルードファイルとジョイスティックの変化確認用の定数を定義します

```
#include "usb_device.h"
#include "adafruit_st7789.h"
#include "glcd_disp.h"
#include "stickmouse.h"
#include "tusbd_base.h"
#include "tusbd_device.h"

#if LCDTYPE == 1
#define LCD_WIDTH    135
#define LCD_HEIGHT   240
#else
#define LCD_WIDTH    240
#define LCD_HEIGHT   240
#endif

#define CURSOR_STEP     5
#define POSTION_MASK    (JS_YMASK | JS_XMASK)
#define BUTTON_MASK     (PUSH_SWITCH | JS_PUSH)
```

USBデバイスドライバの定義

USBデバイス・モジュールの定義

20ms毎のマウスの移動量を5とする

TOPPERS

stickmouse.c：USB用のハンドラや変数を追加

- usbddesc.cでHID用のディスクリプタの初期化関数MakeUsbDescriptorのプロトタイプ宣言を追加
- USBデバイスの制御に必要なハンドラや変数を追加

```
static const uint8_t string1[] = {
    "Hello World !"
};
#endif

extern void MakeUsbDescriptor(TUSBD_Handle_t *);

LCD_Handler_t  LcdHandle;
LCD_DrawProp_t DrawProp;
TUSBD_Handle_t USBD_Device;
uint8_t  HID_Buffer[4];
uint8_t  usb_mode = 0;
```


TOPPERS

stickmouse.c：マウス情報を設定する関数を作成

- positionからマウスの位置情報に変換する関数GetPointerDataを作成

```
/*
 *  HID MOUSE DEVICE CLASS REQUEST
 *  parameter1:  pbuf  データ設定バッファ
 */
static void
GetPointerData(uint8_t *pbuf, uint32_t position)
{
    pbuf[0]  = pbuf[3] = 0;
    if((position & JS_PUSH) != 0)
        pbuf[0] |= 1;
    if((position & PUSH_SWITCH) != 0)
        pbuf[0] |= 2;
    if((position & JS_RIGHT) != 0)
        pbuf[1] += CURSOR_STEP;
    else if((position & JS_LEFT) != 0)
        pbuf[1] -= CURSOR_STEP;
    if((position & JS_DOWN) != 0)
        pbuf[2] += CURSOR_STEP;
    else if((position & JS_UP) != 0)
        pbuf[2] -= CURSOR_STEP;
}
```

stickmouse.c：USB用割込みハンドラを追加

- USBデバイス用の割込みハンドラを追加
  - usb_isrはUSBデバイスドライバ中のハンドラ、割込みサービスルーチンで記載されている

```
void
SPI0_IRQHandler(void)
{
   spi_isr(SPI1_PORTID);
}

void
DMA0_IRQHandler(void)
{
   dma_isr(DMA_INTNO);
}

void
USBCTRL_IRQHandler(void)
{
   usb_isr(USB1_PORTID);
}
```

stickmouse.c : stick_sence関数を改造

- TSB003のPUSH SWITCHの状態を取り込むように修正

```
uint32_t
stick_sense(void)
{
    static uint32_t sw_save = 256;
    uint32_t sw;
    Arduino_PortControlBlock *ppin = getGpioTable(ANALOG_PIN, (PUSH_PORT-16));
    sw  = digitalRead(RIGHT_PORT);
    sw |= digitalRead(LEFT_PORT) << 1;
    sw |= digitalRead(DOWN_PORT) << 2;
    sw |= digitalRead(UP_PORT) << 3;
    sw |= ((gpioc_in_get() >> *ppin) & 1) << 4;
    sw |= ((gpioc_in_get() >> PSW_PIN) & 1) << 5;
    sw ^= 0x3f;
    if(sw != sw_save){
        syslog_2(LOG_DEBUG, "## sw[%02x] sw_save[%02x] ##", sw, sw_save);
        sw_save = sw;
        sw |= JS_EVENT;
    }
    return sw;
}
```

stickmouse.c：メイン関数を改造

- 処理に必要な変数を追加

```
void
main(void)
{
    SPI_Init_t spi_initd;
    LCD_Handler_t   *hlcd;
    SPI_Handle_t  *hspi;
    USB_DEV_Init_t USB_Data_Init;
    USB_DEV_Handle_t *husb;
    TUSBD_ERCODE result;
    uint32_t flgptn;
    char pstring[16];
    uint8_t save_dev_state = 0;
    uint32_t save_position = 0;

    syslog_0(LOG_NOTICE, "Sample program starts.");

    /*
     * 疑似カーネルの初期化とATT_INIの実行
     */
    kernel_init();
    pico_system_init(0);
    led_init(0);
    switch_push_init(0);
//  setup_sw_func((intptr_t)sw_handle);
```

USBデバイスの操作に必要な変数を追加

ループ時に前の状態を保存する変数を追加

PUSH SWITCHの割込み表示をやめる

TOPPERS

stickmouse.c：メイン関数を改造

- SPIドライバの初期化の後に、USBドライバの初期化を追加

```
    /*
     *  USB DEVICE初期化
    */
   memset(&USB_Data_Init, 0, sizeof(USB_DEV_Init_t));
    USB_Data_Init.dev_endpoints = 16;
    USB_Data_Init.speed = USB_SPEED_FULL;
    husb = usbd_init(USB1_PORTID, &USB_Data_Init);
    if(husb == NULL){
        syslog_0(LOG_ERROR, "USB DEVICE INITIAL ERROR !");
        slp_tsk();
    }

#ifdef __riscv
    hazard3_set_vector(INT_VECTOR_IO_BANK0, IO_BANK0_IRQHandler);
    hazard3_set_vector(INT_VECTOR_SPI0, SPI0_IRQHandler);
    hazard3_set_vector(INT_VECTOR_DMA0+DMA_INTNO, DMA0_IRQHandler);
    hazard3_set_vector(INT_VECTOR_USBCTRL, USBCTRL_IRQHandlerr);
#endif
    init_int(INT_VECTOR_IO_BANK0, 2, true);
    init_int(INT_VECTOR_SPI0, 3, true);
    init_int(INT_VECTOR_DMA0+DMA_INTNO, 3, tru
    init_int(INT_VECTOR_USBCTRL, 0x3, true);
```

USBドライバの割込み設定も行う

TOPPERS

stickmouse.c：メイン関数を改造

- ループの前に、USBデバイスモジュールの初期化とスタート設定を行う

```
/*
 *  USB DEVICEミドルウェア初期化
 */
USBD_Device.pSysData = husb;
result = tusbdInit(&USBD_Device, USB_DEVICE_ID);
if(result != TUSBD_E_OK){
    syslog_1(LOG_ERROR, "USB DEVICE MIDDLEWARE INITIAL ERROR(%d) !", result);
    slp_tsk();
}
usbd_setupepdata(husb, 0x81, &husb->dpram->epx_data[0 * 64]);
usbd_setupepdata(husb, 0x01, &husb->dpram->epx_data[1 * 64]);
usbd_setupepdata(husb, 0x82, &husb->dpram->epx_data[2 * 64]);
MakeUsbDescriptor(&USBD_Device);

/*
 *  USB DEVICEスタート
 */
result = tusbdStart(&USBD_Device);
usb_mode = 1;
syslog_1(LOG_NOTICE, "usbd start result(%d) !", result);

while(1){
```

USBデバイスモジュールの初期化

デバイス・ディスクリプタの初期化

TOPPERS

stickmouse.c：メイン関数を改造

- ループの前に、USBデバイスモジュールの初期化とスタート設定を行う

```
while(1){
    flgptn = stick_sense();
    if((flgptn & JS_EVENT) != 0){
        syslog_1(LOG_NOTICE, "## JS EVENT[%04x] ##", flgptn);
        flgptn &= ~JS_EVENT;
        sprintf(pstring, "sw=%02x", (uint8_t)flgptn);
        DrawProp.pFont = &Font16;
        if(flgptn == 0)
            lcd_DisplayStringAt(&DrawProp, 0, 140, (uint8_t *)"      ", CENTER_MODE);
        else
            lcd_DisplayStringAt(&DrawProp, 0, 140, (uint8_t *)pstring, CENTER_MODE);
    }
    if(usb_mode == 1 &&
       ((flgptn & BUTTON_MASK) != (save_position & BUTTON_MASK) || (flgptn & POSTION_MASK) != 0)){
       GetPointerData(HID_Buffer, flgptn);
       if((result = USBD_HID_SendReport(&USBD_Device, HID_Buffer, 4)) == TUSBD_E_OK)
           memset(&HID_Buffer, 0, sizeof(HID_Buffer));
       save_position = flgptn;
    }
    if(USBD_Device.dev_state != save_dev_state){
       syslog_2(LOG_NOTICE, "dev_state(%d) result(%d)", USBD_Device.dev_state, result);
       save_dev_state = USBD_Device.dev_state;
    }
    dly_tsk(20);
}
```

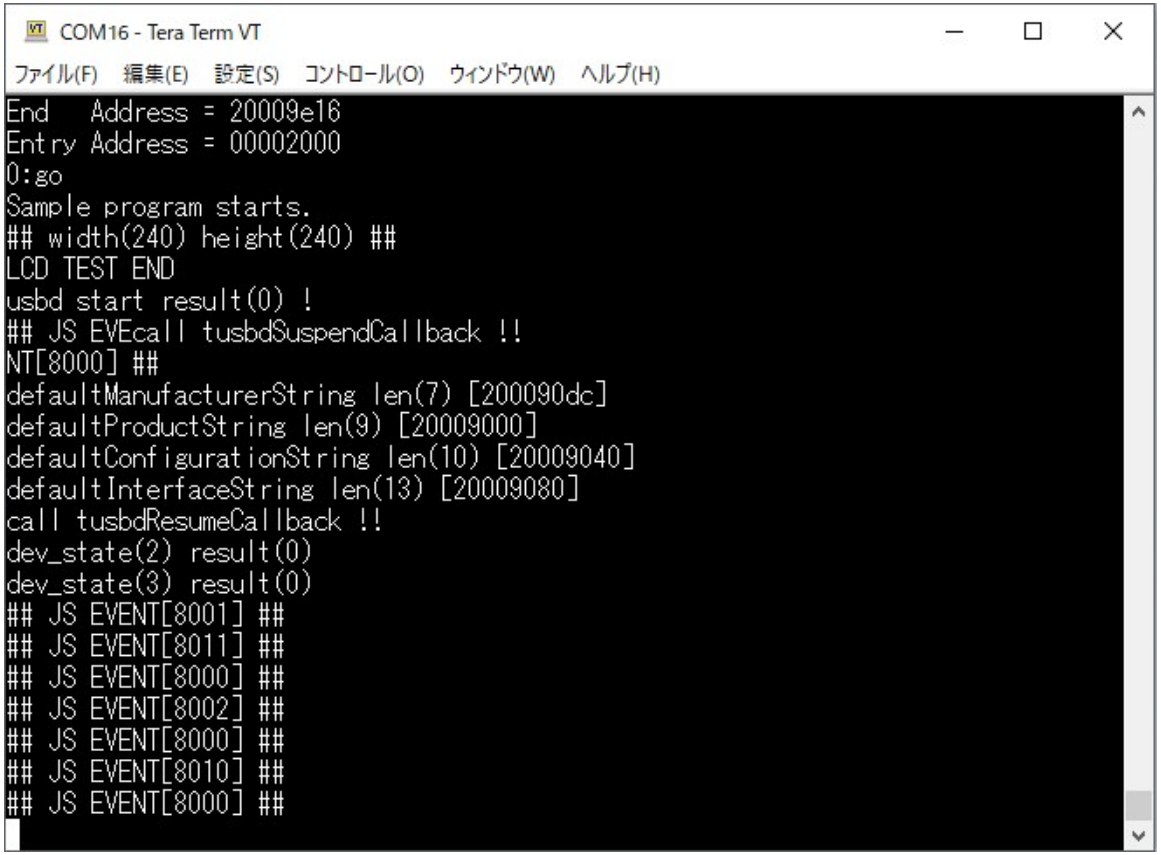
PCにマウスの変化を伝える条件を判定

USBデバイスの状態変化を表示

TOPPERS

ビルド実行

- 電源用USBはPCに接続、ビルドして実行してみます
- dev_state(3)でPCがマウスとして認識
- 一応動作するが、第1ボタンのダブルクリックでアプリが起動しません



TOPPERS

スティック・マウスを作る

1. USBデバイスHIDについて
2. スティック・マウスの仕様
3. プログラムの作成
4. 問題点の対応

STICKマウスのダブルクリック対応: 目的

SDK等のデバイス制御では対応できない機能の対策

- GPIOの入力を厳密に行わないとダブルクリックできない
  - GPIO割込み入力の問題
  - チャタリングの対応
  - 割込み中からのHID通信

- プログラムファイルの置き場所
  - stickmouse1をコピーしstickmouse2を作成して対策
- 実習で対策を行う

チャタリングの問題

- TEB003のPUSH SWITCHを押すと、一度押したにも関わらず「## sw int ##」の表示が2度発生することがある
- これはチャタリングという現象で、SWITCHオンオフ時ノイズが発生して、エッジ割込みではノイズで割込みが発生します
- TEB003のPUSH SWITCHではコンデンサを入れてノイズ対策を行っているが、それでも発生しています
- ジョイスティックのGPIO回路はノイズ除去を行っていないため、よりチャタリングが発生すると思われます
- 細かいオンオフタイミングをPCに通知するために、ソフト的なチャタリング対策が必要となります

stickmouse2：作成

プログラム stickmouse2を作成

- 手順
  - チャタリング対応したボタンセンスプログラムを作成
  - 20msはポジションのみの設定とする改造
  - 第１、第２ボタンの変換通知を割込み中から行うように改造
  - 排他制御の対応

1msの割込みサービスの提供

- システム機能のシステムタイマでは、weak属性でtimer_1ms関数の呼び出しを行っている
- アプリ中で、timer_1ms関数を定義すれば1msのタイマ・サービスを受けられる

```
/*
 *  1MSタイマハンドラ
 */
__attribute__((weak)) void
timer_1ms(uint32_t systick)
{
}
/*
 *  割込みハンドラ
 */
void
SysTick_Handler(void)
{
    :
   systime++;
   timer_1ms(systime);
}
```


TOPPERS

チャタリング対応プログラムを作成する

- スイッチの状態を管理するswitch_tを定義する
  - sense_valueにセンスしたタイミングのSWの状態を保存する
  - countに同一の状態が続いたカウント数を保存する
- sense_valueが3ms継続すると正しいSWの値（current_value）として確定する

型	アイテム名	内容
uint8_t	current_value	現在のSWの状態
uint8_t	sense_value	ハードウェアのSWの状態
uint8_t	pin	ピンのGPIO#
uint8_t	count	状態のカウント

```
#define NUM_SWITCH  2
#define PSW_PIN     22

#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)

#ifndef TOPPERS_MACRO_ONLY

typedef struct _switch_t
{
   uint8_t  current_value;
   uint8_t  sense_value;
   uint8_t  pin;
   uint8_t  count;
} switch_t;
```

スイッチの変化を管理する構造体を定義

- 第1、第2ボタンに対応する2つのPUSH SWITCHの状態を管理する構造体をpush_switchを追加する

```
extern void MakeUsbDescriptor(TUSBD_Handle_t *);

LCD_Handler_t  LcdHandle;
LCD_DrawProp_t DrawProp;
TUSBD_Handle_t USBD_Device;
switch_t push_switch[NUM_SWITCH];
uint8_t  HID_Buffer[4];
uint8_t  usb_mode = 0;
```

timer_1ms関数をアプリに記載

- チャタリング防止機能付きのtimer_1ms関数をstickmouse.cに追加する

```
/*
 *  SYSTICKからの1ms呼び出し
 */
void
timer_1ms(uint32_t systick)
{
    switch_t *p;
    uint32_t gpin = gpioc_in_get();
    uint_t  sw, i;
    for(i = 0 ; i < NUM_SWITCH ; i++){
        p = &push_switch[i];
        if(gpin & (1 << p->pin))
            sw = 0;
        else
            sw = 1;
        if(p->sense_value != sw){
            p->count = 1;
            p->sense_value = sw;
        }
        else
            p->count++;
```

GPIOの値からPUSH SWITCHの状態をswにセットする

sense_valueとswの値が異なるならば、countを1にリセットして、sense_valueにswをセット

sense_valueとswの値が同じならばカウントアップ

TOPPERS

timer_1ms関数をアプリに記載

- current_valueとsense_valueが異なる場合、カウンタが3ms経過していたならば、ボタンが変化したと判定してcurrent_value値を更新する
- 直ぐにPCにボタンデータを通知する

```
        if(p->current_value != p->sense_value && p->count > 3){
            TUSBD_ERCODE result;
            p->current_value = p->sense_value;
            sw  = push_switch[0].current_value << 4;
            sw |= push_switch[1].current_value << 5;
            GetPointerData(HID_Buffer, sw);
            if((result = USBD_HID_SendReport(&USBD_Device, HID_Buffer, 4)) == TUSBD_E_OK)
                memset(&HID_Buffer, 0, sizeof(HID_Buffer));
        }
    }
}
```

ここでボタンの状態が確定する

確定後、すぐにマウスのボタン変化としてPCに送信

TOPPERS

stick_senseプログラムを修正

- push_switch構造体の初期化をメイン関数に追加
- 設定値以外はゼロ

```
    /*
     *  疑似カーネルの初期化とATT_INIの実行
     */
    kernel_init();
    pico_system_init(0);
    led_init(0);
    switch_push_init(0);
//  setup_sw_func((intptr_t)sw_handle);

    push_switch[0].pin = *getGpioTable(ANALOG_PIN, (PUSH_PORT-16));
    push_switch[1].pin = PSW_PIN;
```

stick_senseプログラムを修正

- stick_sense関数でチャタリング防止した第1，第2ボタンの状態を取り込むように修正

```
uint32_t
stick_sense(void)
{
   static uint32_t sw_save = 256;
   uint32_t sw;
   Arduino_PortControlBlock *ppin = getGpioTable(ANALOG_PIN, (PUSH_PORT-16));

   sw  = digitalRead(RIGHT_PORT);
   sw |= digitalRead(LEFT_PORT) << 1;
   sw |= digitalRead(DOWN_PORT) << 2;
   sw |= digitalRead(UP_PORT) << 3;
   sw |= push_switch[0].current_value << 4;
   sw |= push_switch[1].current_value << 5;
   sw ^= 0xf;
   if(sw != sw_save){
      syslog_2(LOG_DEBUG, "## sw[%02x] sw_save[%02x] ##", sw, sw_save);
      sw_save = sw;
      sw |= JS_EVENT;
   }
   return sw;
}
```

不要なので削除

TOPPERS

メイン関数：不要な変数を削除

- PC送信判定にpositionの保存は不要なのでsave_position変数を削除

```
void
main(void)
{
    SPI_Init_t spi_initd;
    LCD_Handler_t   *hlcd;
    SPI_Handle_t  *hspi;
    USB_DEV_Init_t USB_Data_Init;
    USB_DEV_Handle_t *husb;
    TUSBD_ERCODE result;
    uint32_t flgptn;
    char pstring[16];
    uint8_t save_dev_state = 0;
    uint32_t save_position = 0;
```

不要なので削除

stick_senseプログラムを修正

- 第1，第2ボタンの状態通知はtimer_1ms関数でおこなうので、ポジションの変化のみを通知すればよい
- timer_1msはタイマ割込みなので、ポジションの送信は割込み禁止の排他制御を行います

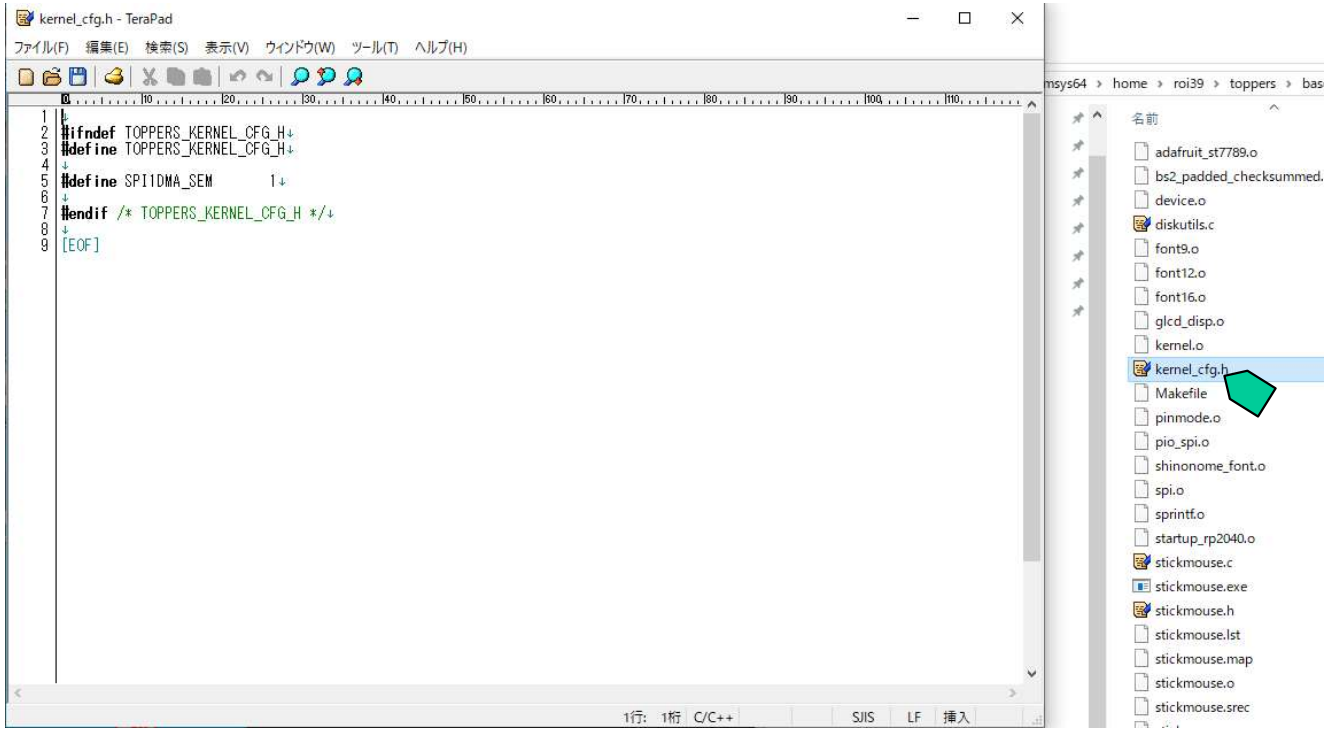
```
if(usb_mode == 1 && (flgptn & POSTION_MASK) != 0){
    loc_cpu();
    GetPointerData(HID_Buffer, flgptn);
    if((result = USBD_HID_SendReport(&USBD_Device, HID_Buffer, 4)) == TUSBD_E_OK)
        memset(&HID_Buffer, 0, sizeof(HID_Buffer));
    unl_cpu();
    save_position = flgptn;
}
if(USBD_Device.dev_state != save_dev_state){
    syslog_2(LOG_NOTICE, "dev_state(%d) result(%d)", USBD_Device.dev_state, result);
    save_dev_state = USBD_Device.dev_state;
}
```

不要なので削除

TOPPERS

ビルドして実行

- ビルドして実行するとダブルクリックが有効となっています
- SDK等で与えられたドライバではなく、新しいデバイス管理機能を追加しないと対応できない場合もあります



kernel_cfg.hのダブルクリックでエデッタが開く

TOPPERS

ベアメタル組込み実装の勘所

1. これまでの実習のまとめ
2. RTOS実装との比較
3. ベアメタル組込み実装の勘所

これまでの学習内容とベアメタル実装

- スタートアップルーチン、割込み設定（学習内容）
  - メーカー提供のSDKを見れば実装可能
- 基本的な開発方針はメイン関数にどんどん、機能追加していく
  - 達成したい機能、非機能条件（制約）はわかっても、対応方法がわからない？
  - ハードウェアに対するプログラミング？
    - 現代のIOT設計を考えると、ハードウェアIPは
      DMA,PWR,SPI,I2C,UART,TIMERを使って通信を行い
      個別の機能は上位のドライバで実装するケースが多い

1999年トロン協会で公開した「μITRON4.0仕様研究会のデバイスドライバ設計ガイドライン」では
DMA等のベーシックドライバをPDICと呼び、機能実行する上位のドライバをGDICと呼ぶ
DICとは、Device Interface Componentの略

必要なデバイスドライバを揃える

- メイン関数よりも、デバイスドライバの方が大きく複雑
  - PDICドライバは、ハードウェア依存
  - GDICドライバは、機能も規模も大きなものがある
- 省エネ機構、ベーシックデバイスドライバ
  - RTOSではサポートされない
  - システムではフルスペックが要求される
    - ポーリング方式ではなく、割込み＋DMA等
  - 同じチップメーカーのSoCでもバージョンアップしていく
- 各メーカーで公開するデバイスドライバの最新ソースを使用する
  - Pico SDKを参照して作成
  - Arduino IDEなどの開発環境
  - mbed、TOPPERS BASE PLATFORM等のオープンソース

ベアメタル組込み実装の勘所

1. これまでの実習のまとめ
2. RTOS実装との比較
3. ベアメタル組込み実装の勘所

プログラムサイズを最小にできる

- ベアメタル版とTOPPERS ASP版のlcd_text4とサイズ面で比較する
- ROMサイズで78%、RAMサイズで26.8%の容量となる
- 特に問題なのは、Cortex-M0+に実装しようとした場合、ベアメタル実装ではRAMサイズが16KBでも実装可能であるが、RTOSでは32KB以上は必要

セクション	Asp版lcd_text4（バイト）	lcd_text6（バイト）	比率（%）
.text	63,888	38,440	60.17
.rodata	89,296	81,157	90.88
.data	4,208	3,084	73.29
.bss	27,064	5,280	19.51
ROM	157,392	122,681	77.95
RAM	31,272	8,368	26.76

ベアメタル組込み実装の勘所

1. これまでの実習のまとめ
2. I2Cシールドをベアメタルで実装してみる
3. ベアメタル組込み実装の勘所

ベアメタル組込み実装の勘所

- 「これまでの実習のまとめ」では、開発環境やプログラミングをベースにまとめを行った
- 実際、システム設計を行うと種々の配慮が必要となる
  - デバイス・ドライバの入手、作成
  - 規模の大きなシステムは複雑になりすぎる
  - アップデートの必要性
  - ICEに頼り切った開発は危険
  - ウォッチドックタイマの使用

デバイス・ドライバの入手、開発

- DMA/I2C/SPI等のデバイスIPはSoCのIP設計で作られるので、同じMPUでも、SoCメーカーによって、まったく別物である
- 同じSoCメーカーでも、半年単位でバージョンアップしていくので、SoCを変えると動作しなくなるケースもある
  - PDICドライバの開発が困難
- PDIC関数のAPIを、統一化して上位のドライバ（GDIC）が共通に使える工夫が必要
  - TOPPERS BASE PLATFORM（ST）と(CV)でベーシックドライバのAPIを共通化して、SPIやUSB OTGの上位のドライバやミドルウェアを共通化している
- SDKのドライバがそのまま使えないケースもある

規模の大きなシステムは複雑になりすぎる

- ベアメタル実装ではアセンブラ化しやすいC言語で実装する必要があるため2万行以下のコードサイズが適正
  - オブジェクト指向言語やパーサー言語はRTOS以上にメモリを消費するため、本来の目的からそれる
- 開発手法は構造化設定等が適正
- 小規模のプログラムを十分なテストや検証により、作り込んでいく形を取るケースが多い
  - 開発、検証に見合うロットが必要となる

アップデータの必要性

- プログラムの信頼性維持のため、アップデータが必要となるケースがある
  - 通常のワンチップマイコンではICEを使ってプログラム書き込みを行う
  - 外部通信可能なケースではネットワークパケットでアップデートデータを送り、プログラムアップデートを要求されるケースもある
- STマイクロエレクトロニクスのSoCは、DFU形式でICEを使わずUARTやUSB DEVICEからアップデートが可能
- ベアメタル実装の小規模なシステムではFLASH-ROM実行を行うため、特殊なシステム設計が必要
  - FLASH-ROM上に2つのシステムを配置し、パケットデータをFLASH-ROMに書いていく
  - この場合、FLASH-ROM上のインストラクションフェッチができないため、RAM上での実行プログラムが必要

ICEに頼り切った開発は危険

- ICEを開発やデバッグに使用するのは有用
- 問題は、機能テストやフィールドテストでICEは使用できないケースが多い
- ある程度のデバッグシステムを用意しておく必要がある
  - 本セミナのようなsyslog_nやsys_printfのようなログ出力機能
  - メモリやレジスタのDUMP機能
  - システムの状態やモジュールの状態を表示する機能
  - テストコマンド等による直接ハードウェア、ソフトウェア検証機能等

ウォッチドックタイマの活用

- スタンドアロンのIoTデバイスとして使用する場合、何かの要因でシステムが停止状態になった場合でも、継続して使用可能な手段が必要
- ウォッチドックタイマを使用すれば、システム停止時、リセットから復帰が可能になります

まとめ

小規模組込み開発のまとめ

- 小規模な組込み開発では少数の開発者でファームウェア、ソフトウェアの開発を行うため、ハードウェアの知識や開発環境を構築する知識も必要となる
- 開発言語はC言語を用いることが多いが、C言語で記述できない部分は、アセンブラ言語で開発する必要がある
- 開発環境はプロセッサメーカーの開発環境を用いることが多く、開発前にノウハウの蓄積が必要
- ハードウェアの制御は、ポーリングを用いるのもの、割込みを用いて制御を行うものがあり、適切な方式を選択する必要がある
- 割込みに制御はプロセッサ毎に異なる。特別な設定を行う必要があり、処理を熟知して開発を行う必要がある

小規模組込み開発のまとめ

- アプリケーション部分の開発については、オブジェクト指向やモデル設計を用いれば、可視的なソフトウェア開発ができる。
- このような開発は実行ROM、RAMサイズの増加を伴うことが多いので注意が必要
- NPO法人　組込みソフトウェア管理者・技術者育成研究会では、いろいろは組込みソフトウェア開発手法の紹介を行ってますので、組込み開発プロセスの参考にしてください。http://www.sessame.jp/

参考文献

- RP2040 Datasheet（ラズベリーパイ財団）
- Raspberry Pi Pico Datasheet（ラズベリーパイ財団）
- Sitornix ST7789H2 Datasheet V0.2（Sitornix社）
- Adafruit ST7735/ST7789 LCD用Arduinoサンプルドライバ（Adafruit社）
- TOPPERS BASE PLATFORM
  リファレンス・マニュアル
  TOPPERSプロジェクト 教育WG
- Interface 2015年8月号

著者リスト

- 開発環境の説明
- LCDJOYシールドを動かす
- スティック・マウスを動かす
- ベアメタル組込み実装の勘所
  - 竹内良輔（TOPPERSプロジェクト教育WG）