

TOPPERS基礎実装セミナー

(Raspberry Pi Pico(W)/Pico2(W)版)

新基礎3編:1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) 1日目＞

Copyright (C) 2016-2025 by竹内良輔 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

■ 1日目

- | | |
|------------------------------|-------|
| 1. TOPPERS BASE PLATFORM(RP) | 0.5時間 |
| 2. 開発環境とハードウェアの検証 | 0.5時間 |
| 3. 可変抵抗を使って、ADC入力 | 1.5時間 |
| 4. SPI仕様、SPIデバイスドライバ | 0.5時間 |
| 5. LCDの初期化とピクセル設定 | 1.5時間 |
| 6. LCD描画プログラム実習 | 1.0時間 |
| 7. まとめ | |

概要

- 本教材は組込みプラットフォームについて学ぶ教材です
- ArduinoはArduino IDEを使用して簡単に組込みシステムを構築することができる。通常の組込みシステムでもRTOSを核とした組込みプラットフォームが、Arduino IDEにあたる部分となる
- ラズベリーパイ財団のPICO(W)ボード上に構築した組込みプラットフォーム: TOPPERS BASE PLATFORM(RP)を解説しながら、Joy StickやグラフィックLCDやSDカードを制御するプログラムについて学習し、このシールドを使用してアプリの作成ができることを目指す
- そのために、アプリが動作する組込みプラットフォームの構造と目的について理解する

概要

- RTOSを使用した組込みプラットフォームの題材として学習します。1日目にADC、SPI仕様について学び、それらを使用しJOY STICK、1.54”グラフィックLCDの制御について学習と実習を行う。
- 2日目にSPI通信を使用したSDカード制御とその上位で動作するファイルシステムについて学習と実習を行う。最後にサンプル組込みシステムについて解説する。
- 今回教材として使用するLCDJOY ShieldはArduino上でグラフィック表示、キー処理、データストレージをArduino IDEを用いて行いますが、本教材はRTOSを用いた環境で同等の機能実現を行うと共に、Arduinoではライブラリの形で提供されるデバイスドライバの内容をソースコードを見て学習する内容となっています

TOPPERS BASE PLATFORM(RP)

1. 組込みプラットフォーム概要
2. TOPPERS BASE PLATFORM
3. プラットフォームの部品

組込み規模と開発手法

- 組込み機器適用機種はUSBメモリからプラント機器まで多機種に及ぶ
- 開発規模により開発手法が異なる
- 便宜上、小規模、中規模、大規模に分類する
- この講座では中規模用のプラットフォーム構築技術について講義を行う

開発規模	小規模	中規模	大規模
総ステップ数 ^{注1}	2万行以下	2万行～50万行くらい	50万行以上
デバッグ方法	ICEが主流	ソフトデバッガやICE	シュミレータ等を使用しパソコン上で開発
ベース環境	RTOS等を使用しない	RTOSやミドルウェアでプラットフォームを構築	汎用OSを使用する場合もある
人数 ^{注1}	10人未満	100人未満	100人以上
参考システム	arduino	TOPPERS BASE PLATFORM	LINUX

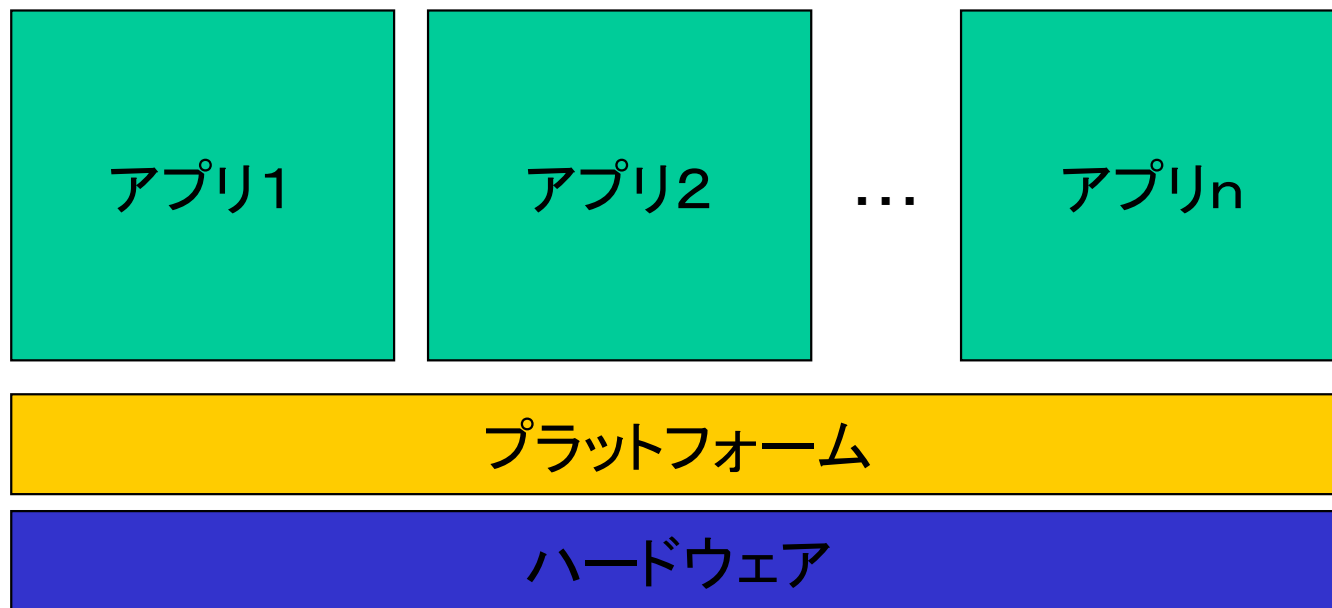
注1) 目安であり、プロジェクトによって異なる

組込みプラットフォーム技術者の必要性

- LINUXや汎用組込みプラットフォームを使ってシステム構築する
 - LINUXはオープンソースであるため、組込み用のハードウェアに特化した場合(特化しなくても)OSの改変やメンテナンス、開発環境のメンテナンスが必要となる。これらを外部に委託した場合コストメリットは低くなる
 - 市販の汎用組込みプラットフォームは開発環境を含んだ高価なものが多い、最終的なカスタマイズや不具合解析は開発者が行わなければならない
- いずれにしても、プラットフォームに特化したスキルを持った技術者は必要
- この講座はLINUXや他のプラットフォームを使用する場合でも必要なスキルを提供する

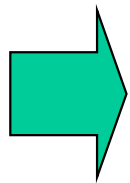
中規模組込みシステム構成

- 中規模組込みシステムを模式化すると、アプリケーションとプラットフォームで構成される
 - アプリケーション 機器の機能を実現するソフトウェア群
 - プラットフォーム 複雑な部分を隠蔽しアプリケーションにサービスを提供するソフトウェア群
- プラットフォームはハードウェアを隠蔽し、アプリケーションに最適化とポータビリティを与える



アプリケーションとプラットフォーム

- アプリケーションは機器の機能実現用のソフトウェア群
 - 多人数でより簡単に、効率よく開発できること
 - 組込み機器の仕様変更に従って、変更可能なこと
 - 大規模開発が可能なこと
- プラットフォームは汎用サービス用のソフトウェア群
 - 複雑部分を隠蔽するため小規模な方がよい
 - ハードウェアに依存した設計、作りこみが必要
 - アプリケーションに汎用的なサービスを提供
 - アプリケーションの改変に影響されない設計



理想的なプラットフォーム設計は、汎用OSの設計に観られる

組込みプラットフォームの設計指針

- 組込みプラットフォームには2つの側面がある
 - 汎用的なコンピュータ制御を行う側面
 - 組込み機器に特化した側面
- 汎用的なコンピュータ制御
 - 汎用OS設計思想が反映される
 - LINUX, Windows, MS-DOS, CPM等
 - ファイルシステムやネットワークは標準的な方が良い
 - 割込み、キャッシュ、ハードウェア制御等の隠蔽
- 組込み機器制御
 - 機器に特化したミドルウェアやハードウェアを効率よく制御する必要がある

組み込みプラットフォームの構成

- 組み込みプラットフォームは一般的には以下の部品で構成される
 - API (Application Program Interface)
 - ミドルウェア (デバイスドライバを含む)
 - ソフトウェアデバッガ (タスクモニタ)
 - デバイスドライバ
 - RTOS
- APIはアプリケーションとのインターフェイス
 - APIはアプリケーションにサービスを提供する関数群
 - 機器固有のサービス化も必要となる
 - アプリケーションのポータビリティを向上させるためRTOSのサービスも隠蔽化することがある

Media Player

- ASPカーネルとBASE PLATFORM(ST)で、STM32F746/769 Discovery上で作成したJPEG表示、MP3演奏システム
- SDカードとUSBメモリに対応



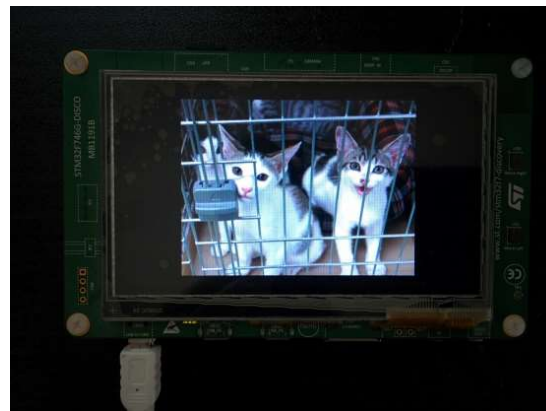
メディアから音楽ファイルやJPEGファイルを選択



RTCを用いた時刻表示



STM32F769 Discovery Media Player



JPEG表示

メディア・アプリ

TOPPERS BASE
PLATFORM(ST)

ASPカーネル

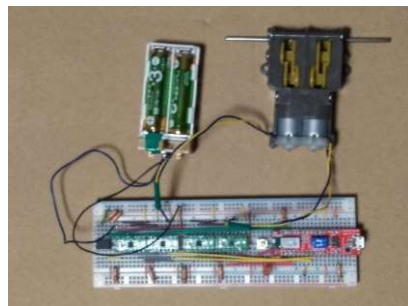
STM32F746/769
Discovery ボード

BLE REMOTE

- BLEを使ったクローラ、模型戦車のリモコンシステム
- モータ制御を行うBLEデバイスボード
 - STM32WB55 USB dongleを核にブレッド・ボード版を作成
 - nRF51822_xxAA(BVMCN5103)を核に実装ボード
- Android用リモコンアプリの作成



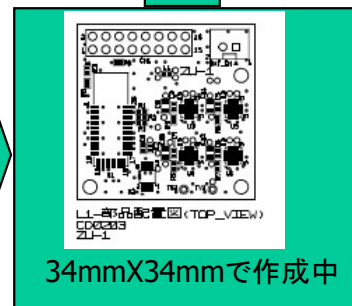
USB dongleブレッド・ボード



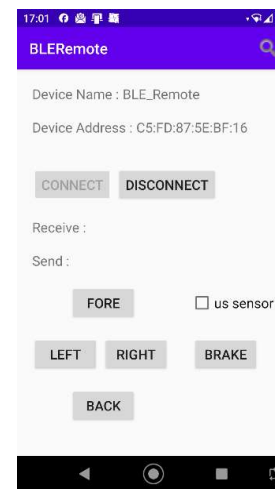
BVMCN5103ブレッド・ボード



リモコン戦車



BVMCN5103実装ボード



Androidリモコンアプリ

BLE デバイス・リモコンアプリ

TOPPERS BASE
PLATFORM(ST)

nRF51SDK
1.0.0

ASPカーネル

STM32WB55
USB dongle

BVMCN5103

TOPPERS BASE PLATFORM(RP)

1. プラットフォーム概要
2. [TOPPERS BASE PLATFORM](#)
3. プラットフォームの部品

TOPPERS BASE PLATFORM(RP)

- 組み込み学習や先行開発用に、ラズベリーパイ財団のRP2040/RP2350コアを使用したエバレーション・ボード上に学習用にASP/FMPカーネルを核とした組み込みプラットフォームTOPPERS BASE PLATFORM(RP)をオープンソースで開発した
- RP2040はCortex-M0+ 125MHzのデュアル・コア
- RP2350はCortex-M33とRISC-V 150MHzの両デュアル・コアでFMPカーネルの実習が可能
- Pico WはWIFIモジュールCYW43439を持ち、WIFI対応のTCP/IPプロトコルスタックの演習が可能



TOPPERS BASE PLATFORM(ST)

- 実際の組み込みプラットフォームは、商品の組み込み機器中で使用されアーキテクチャやソースコードを公開することはありません
- 教育WGでは、組み込み学習や先行開発用に、STマイクロ社のCortex-M0、M4、M7を使用したエバレーション・ボード上に学習用にASPカーネルを核とした組み込みプラットフォームTOPPERS BASE PLATFORM(ST)をオープンソースで開発した
- 組み込みプラットフォームの商品例として、STM32F746/769 Discoveryボード用に「Media Player」STM32WB55 nucleoボード用に「BLE REMOTE」を用意した

対応ボード一覧

- TOPPERS BASE PLATFORM(ST)V1.4.2は20のSTMボードに対応する
- Arduinoコネクタに対応して市販の7つのシールド、TEB001拡張シールド4つに対応して、評価、実験を行うことができる。
- USBコネクタが付いたボードはUSB-OTGに対応して、MSCやHIDの評価、実験が可能
- F091/L476/F723/L4R5/G071ではSDカード動作せず、L073ではメモリ不足でファイルシステム動作せず
- STM32MP157C-DK2はV1.4.2で対応

ボード名	基礎2	Adafruit1.8"TFシールド	TEB001:12Cシールド	BLEシールド2.1	TEB001:EPCシールド	TEB001:uart/lcdシールド	USB-OTG
STM32F401RE nucleo-64	○	○	○	○	○	○	-
STM32F407 Discovery	○	-	-	-	-	-	○
STM32F746 Discovery	-	-	×	×	○	-	○
STM32F446RE nucleo-64	○	○	○	○	○	?	-
STM32F446ZE nucleo-144	○	○	○	×	○	?	○
STM32F746ZG nucleo-144	○	○	○	×	○	?	○
STM32F767ZIT nucleo-144	○	○	○	×	○	?	○
STM32F769N Discovery	-	-	○	-	○	-	○
STM32F091 nucleo-64	○	△	○	○	○	?	-
STM32L073 nucleo-64	○	△	○	○	○	?	-
STM32L476 Discovery	○	-	-	-	-	-	△

対応ボード一覧

ボード名	基礎2	Adafruit1.8" TFTシールド	TEB001:I2Cシールド	BLEシールド2.1	TEB001:EPCシールド	TEB001:uart/lcdシールド	USB-OTG
STM32L476 nucleo-64	○	△	○	○	○		-
STM32F723 Discovery	○	△	○	?	○	?	○
STM32L4R5 nucleo-144	○	△	○	?	○	○	○
STM32G071 nucleo-64	○	△	○	○	○	○	-
STM32H743 nucleo-144	○	△	○	-	○	○	×
STM32G474 nucleo-64	○	△	○	-	○	○	△
STM32G471 nucleo-64	○	△	○	-	○	○	△
STM32WB55 nucleo	×	△	○	不要	○	○	△
STM32MP157C-DK2	×	×	△	-	○	○	×

ブロック図

- ASP-1.9.3カーネルを核に7つのレイアでプラットフォームを構築している
- 部品はポータブル
- Arduinoではすべてライブラリで提供される

BASE PLATFROM V1.3.0		STM32M4xx	STM32F7xx	detail	contents	
Device driver (PDIC)	Basic driver	gpio	gpio		新基礎2	
		dma	dma			
		(timer)	(timer)			
		uart	uart			
		wdog	wdog			
	Standard driver	i2c	i2c	CLCD/senser	新基礎3 Reference Manual	
		spi	spi	GLCD/SDcard		
		adc	adc	Joy stick		
		qspi	qspi	FLASH ROM		
		rtc	rtc	Clock機能		
		usb OTG	usb OTG	host/dev/otg		
	F7 depend driver		ltdc/dsi	GLCD	Application Manual Reference Manual	
			tp			
			sdmmc	SDcard		
			audio/dfsdm	Sound IN/OUT		
			emac	Ethernet		
GDIC	High driver			各種shield用		
api middleware	File System	fatfs	fatfs	FAT	新基礎3	
		file library	file library	C language file		
		stdio	stdio	STDIO		
	USB System	USB HOST/DEV	USB HOST/DEV	MSC/HID		
	(Open source) libr ary		STM-BSP	GUI		
			libjpeg	JPEG		
			libmad	MP3		

Arduino Shieldを使ったドライバ開発

- TOPPERS BASE PLATFORMはArduino用シールドを組み合わることにより、ハードウェアモジュール用のドライバ開発(GDIC)が可能になる
- ポータブルなので不要なドライバは取り外し可能
 - Prototyping Shield
 - 基礎1, 2教材用シールド GPIO, UART, TIMER等
 - I2Cシールド I2C用 LCD、SENSOR、EEPROM
 - LCDJOYシールド 本教材のシールド
 - USB/LCDシールド SPIグラフィックLCD、UART
 - ESP32 WIFI Network
 - 市販のシールド
 - IWHI 1.54” LCDシールド SPI、ADC
 - Adfruit 1.8” TFT Shield SPI、ADC: 生産中止
 - BLE Shield 2.1 BLEペリフェラル: 生産中止

TOPPERS BASE PLATFORM(RV)

- SiFive社HI-Five1/Kendryte社K210用組込み環境
- ASPカーネル上の実装
 - GPIO/RTC/WDOG/SPI/I2Cの開発
- Longan nanoとH-Five1 Rev.Bの対応

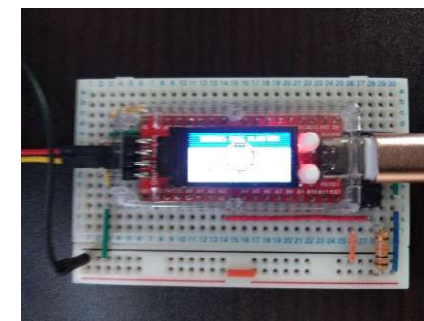
ボード名	SPI	I2C	専用LCD	専用CAMERA	SPI-SDカード	Adafruit1.8" TFTシールド
Hi-Five1 Rev.A/Rev.B	○	△	–	–	–	○
Maix-bits	○	○	○	○	○	○
Maixduino	○	△	○	○	○	○
Longan nano	○	○	○	–	○	?



SiFive社:HI-Five1(RISC-V/32)



Sipeed社:Maixduino(RISC-V/64)



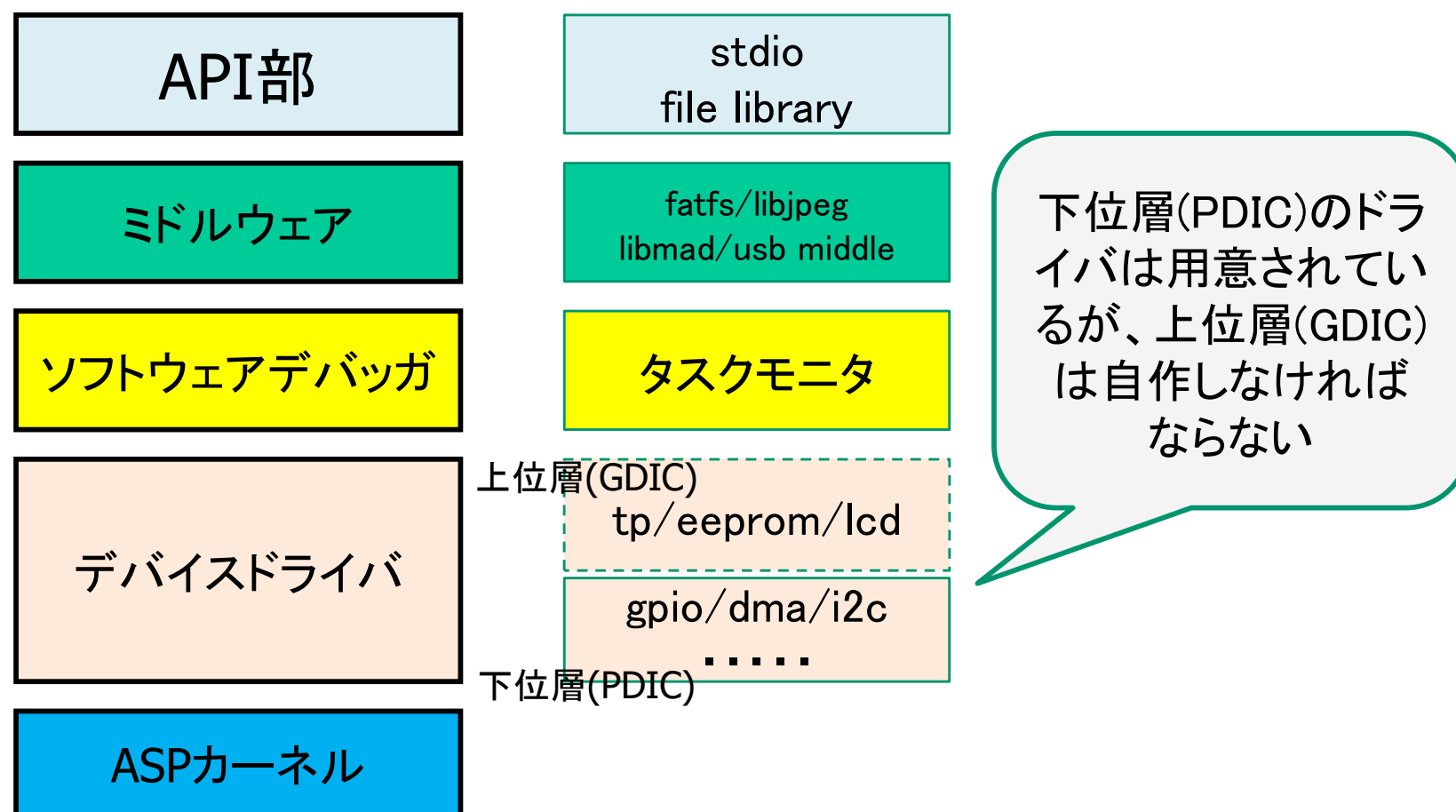
Sipeed社:Longan nano(RISC-V/32)

TOPPERS BASE PLATFORM(ST)

1. プラットフォーム概要
2. TOPPERS BASE PLATFORM
3. プラットフォームの部品

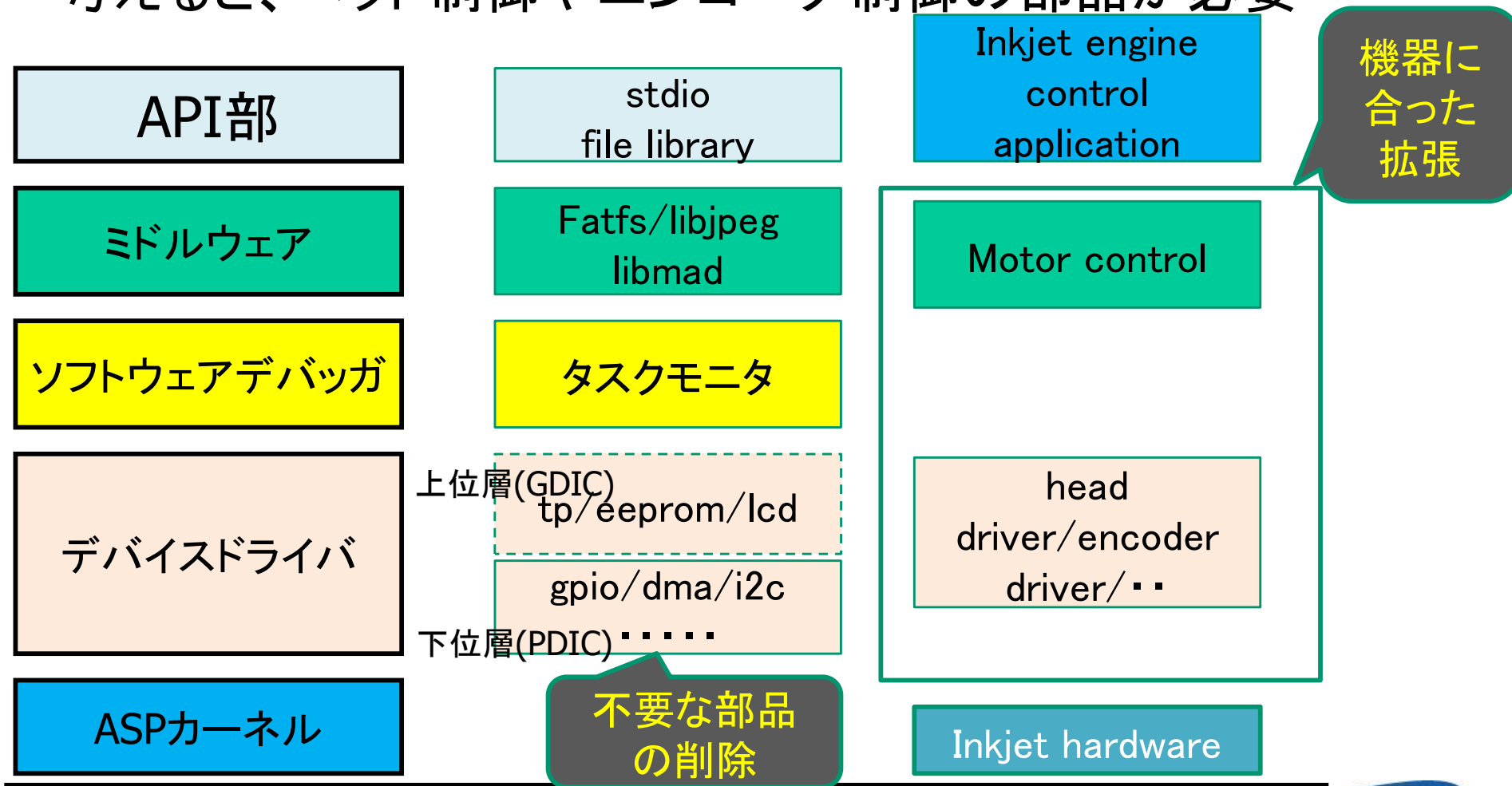
組み込みプラットフォームを構成する部品

- 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



機器を構成するプラットフォーム部品

- 商品を考えるとすべての部品がそろっているわけではない
- 例えば、インクジェット・プリンタの組込みプラットフォームを考えると、ヘッド制御やエンコーダ制御の部品が必要



RTOS:リアルタイム・カーネル

- RTOSはリアルタイム・カーネルと呼ばれるように組み込みプラットフォームの核となる部品です
 - CPUを管理: マルチタスク機構
 - 時間の管理
- Arduino IDEのようなベアメタルシステムではリアルタイム性を保障したり、規模の大きなシステムの管理が難しくなる
- RTOSはマルチタスク機構により、時分割でCPUを多重化するためベアメタル・システムの弱点を克服できる
- ASPカーネルでは割込みをハンドラとして管理可能なので、割込み(非タスクコンテキスト)のシステム管理も可能

用語: ベアメタル(Bare Metal)とは、OSなどを基本システムを組み込んでいない(むき出しの)コンピュータのことを示す

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る
- 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る

GDIC
GDIC
GDIC
PDIC
デバイス

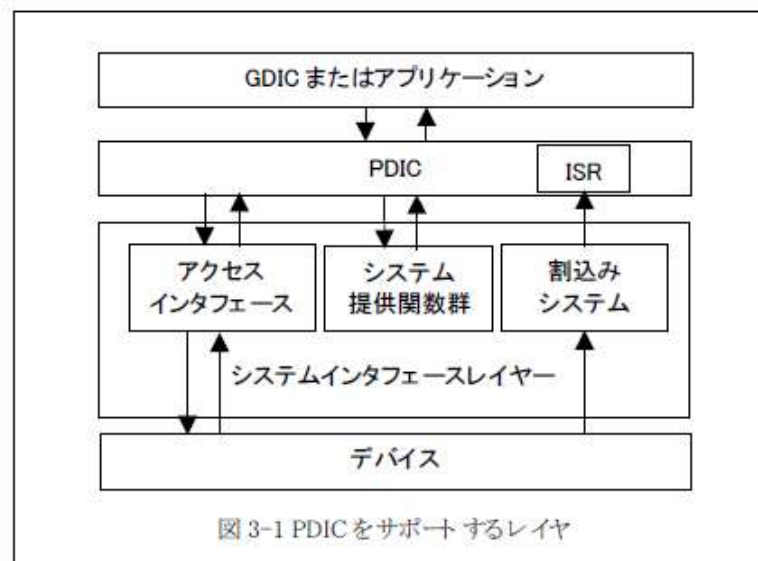
GDIC	
GDIC	高機能なデバイスのPDIC
低機能なデバイスのPDIC	
デバイス	デバイス

DICとはDevice
Interface
Componentの略

図 2. DIC の階層構造

デバイスドライバ

- 機器独自のハードウェアを持つ場合、デバイスドライバは自作する必要がある
 - 独自のハードウェアは10年前まではASICやSoCにより実装された
 - FPGAのコスト低下により、簡単に実装できるようになった
 - SoCFPGAの開発でより、小規模の実装ができるようになった
- デバイスドライバの開発には、ハードウェア技術を熟知する必要がある



IoT開発では、PDICは
UART/I2C/SPIなどを使用し、
個々のデバイスはLSIとして供
給される
個々のデバイス毎のGDICドラ
イバ設計が必要

ソフトウェアデバグ

- 小規模なシステムではICEを使用するが多い
- 多人数で組込み開発を行う場合、すべてのメンバー（特にアプリケーション開発者）にICEを配給できない
- 開発用、テスト時の問題解析用にソフトウェアデバグをプラットフォームに実装した方が、問題解析しやすい
- ソフトウェアデバグに求められる機能
 - タスクやメモリの管理
 - システムの状態監視、管理
 - 機器のテスト機構
 - ストール時の問題解析機能
- プラットフォーム開発者はICE等のデバグ機器は有用な開発手段となる

ミドルウェア

- ミドルウェアとは、RTOSとアプリケーションの中間に位置し専門的な処理を行うソフトウェア群をさす
- ミドルウェアには、機器固有の専門機能を行うもの（専用ミドルウェア）と、多くの機器で汎用的に使用されるもの（汎用ミドルウェア）に分けられる
- 汎用ミドルウェアの例は以下のとおりで、オープンソースとして提供されるものもある
 - TCP/IPプロトコルスタック(TINET)
 - ファイルシステム(FatFs)
 - USBホスト・デバイス
- 専用ミドルウェアは企業秘密に属するものが多く、一般には公開されない

API(Application Program Interface)

- APIはApplication Program Interfaceの略でアプリケーションから使用できるプラットフォームの関数群。プラットフォームを特徴づける機能
- 通常は隠蔽化したAPIを作成する。種々の事情でRTOSやデバイスドライバのI/Fをそのまま使用する場合もある
- APIを用いてプラットフォームを使用する上での手続きや規約を定める
- 組み込み用のAPIは以下の相反する特性を持つ
 - 組み込み用のAPIは機器の特性に特化する
 - アプリケーションの変更に追従する汎用性を求める
- 組み込みシステムもまた、コンピュータシステムなのでUNIX等のコンピュータシステムのAPIを参考に構築する場合が多い

開発環境とハードウェアの検証

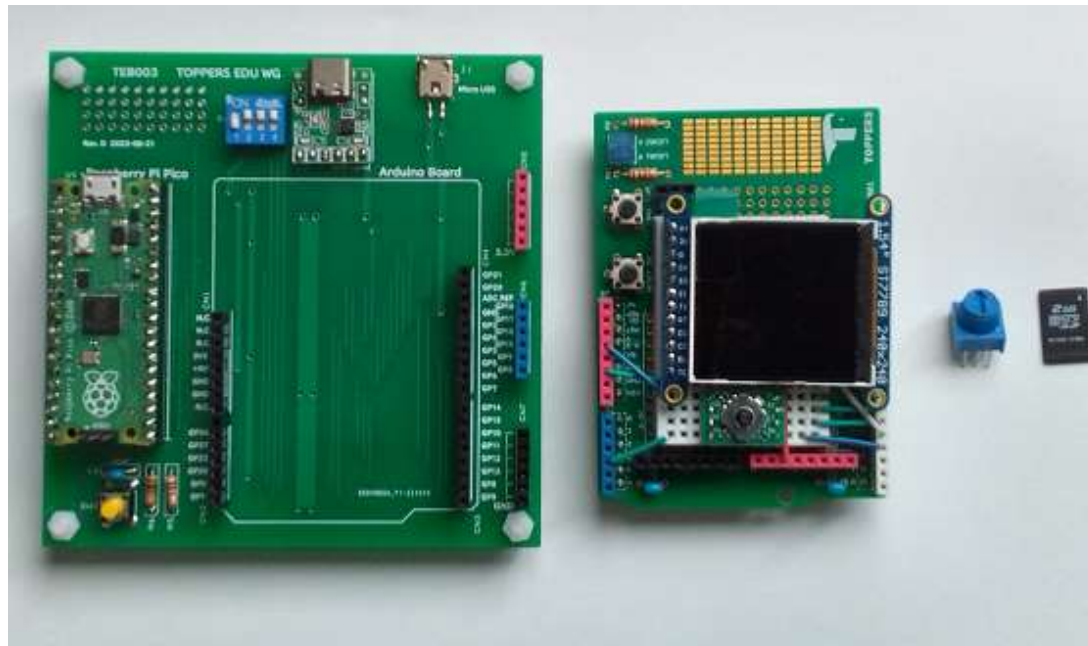
1. 開発環境の構築
2. ASP/プラットフォーム環境構築
3. タスクモニタの実行
4. CPUボードの確認
5. LCDJOYシールドの確認

この章での実施項目

- 開発環境の構築手順の説明
 - パソコンにセットアップ済みです
- ASPカーネル/TOPPERS BASE PLATFORMのポーティング方法の説明
 - 作業が伴います
- MON(タスクモニタの構築)
 - 作業が伴います
- Raspberry Pi Pico/Pico2 + TEB003ボードの説明
 - 座学です
- LCDJOYシールドのハードウェア解説
 - 座学です

ボードの確認とソフトウェアのセットアップ

- 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る
- 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



TEB003ボード

LCDJOYシールド

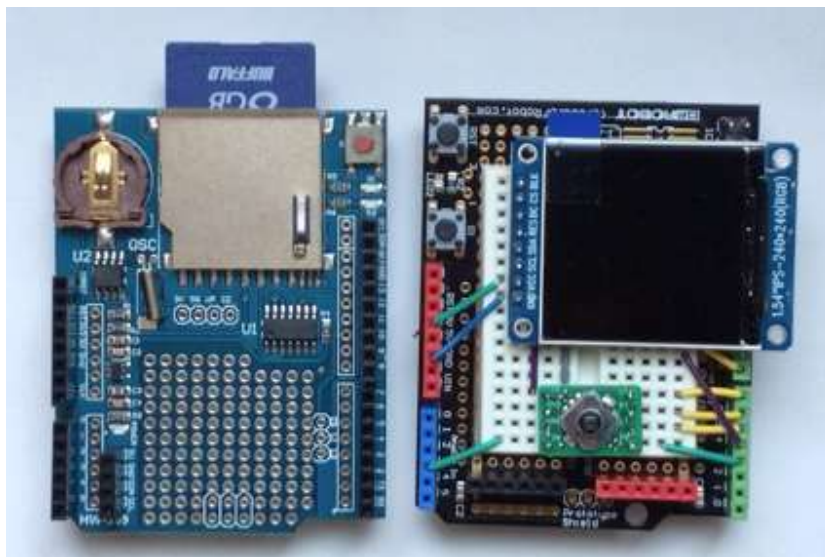
可変抵抗

SPI通信可能なSDカード

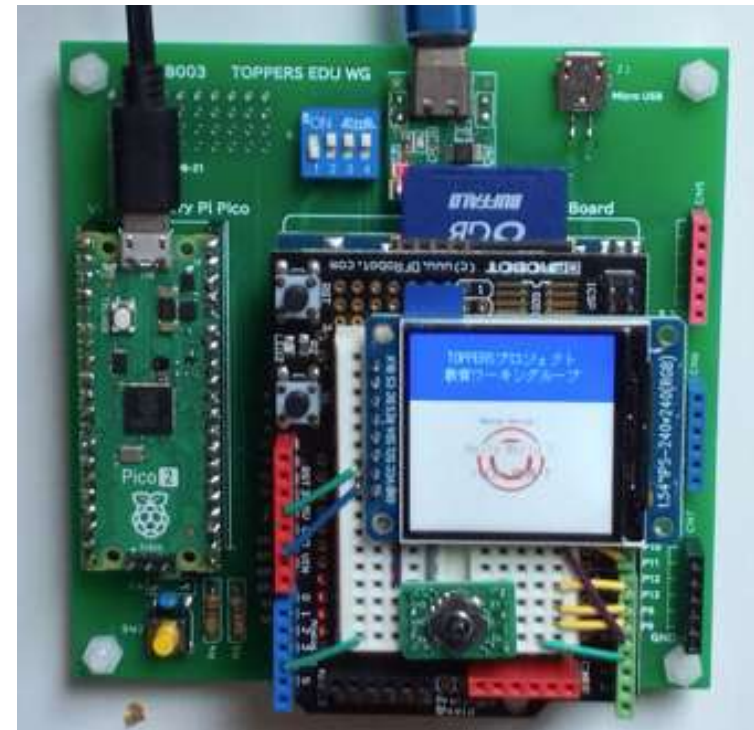
本講座では可変抵抗として
秋月電子通商の「半固定ポ
リウム 10K Ω [103]」を使
用しています

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



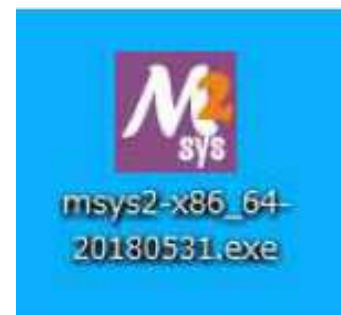
開発環境の構築

- この教材の開発環境を構築するには以下の作業が必要
 - UNIXシェル互換開発環境のインストール
 - CygwinやMSYS2
 - ARM-GCCクロスコンパイラのインストール
 - bin2uf2コマンドのビルドとMSYS2の設定
 - 仮想ターミナルの入出力用にTeraTermのインストール
- 構築手順の詳細はcommonディレクトリ中の「TOPPERS基礎実装セミナー: 開発環境編 (BaseTrainingSeminar_environment-xxxxxxx.pdf)」、またはself_testディレクトリ中の「TOPPERS BASEPLATFORM 開発環境構築 xxxxxx.pdf」に記載

MSYS2のインストール

- MSYS2のダウンロードサイトから最新の「msys2-x86_64-yyyymmdd.exe」をダウンロードする

ホームページ<http://msys2.github.io/>
インストール後、pacmanを使って環境を整備する



組み込みソフトウェア開発に使われるプログラミング言語

- C言語

- ハードウェアを直接操作するプログラミングが可能であるため、組み込みソフトウェア開発では、最も使われている

- アセンブリ言語

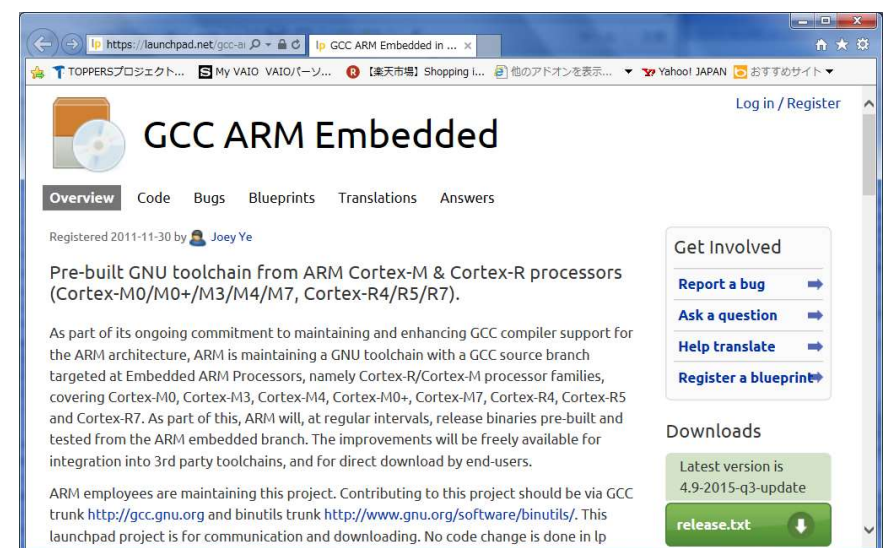
- DSPなどの特殊なプロセッサで使われる場面が多い
- コンパイラが扱えない特殊命令を直接記述して、性能を出す

- C++言語

- 利用は広がっているが、まだ限定的
- オーバーヘッドが大きい
- どのような実行コードになるか見えにくい

GCC ARMのインストール

- GCCのインストールは、GCCのソースコードをダウンロードして、インストールが可能ですが、手順が複雑なため、ここではGCC ARMのバイナリインストールを行います
- <https://launchpad.net/gcc-arm-embedded/+download>からダウンロードしてください



開発環境とハードウェアの検証

1. 開発環境の構築
2. ASP/プラットフォーム環境構築
3. タスクモニタの実行
4. CPUボードの確認
5. LCDJOY Shieldの確認

TOPPERS/ASPカーネルのダウンロード

- TOPPERSプロジェクトのWebからasp-1.9.3の個別パッケージをダウンロードします
- 同様にARM Coretex-M0アーキテクチャ・GCC依存部パッケージasp_arch_arm_m0_gcc-1.9.8をダウンロードします

TOPPERS/ASPカーネル個別パッケージのダウンロード

TOPPERS/ASPカーネルの最新リリースの個別パッケージを配布しています。各ターゲットに対応するカーネル非依存部パッケージ、依存部パッケージ(アーキテクチャ依存部、ターゲット依存部)、シリアルドライバ依存部(PDIC)パッケージを個別にダウンロードできます。

ターゲットごとに必要なソースコードを一つにまとめた簡易パッケージは、[こちら](#)からダウンロードできます。

一般公開以前のバージョン(Release 1.3.0以前)に対応した個別パッケージは、会員向け早期リリースとしての扱いですが、[こちら](#)からダウンロードできます。

ターゲット非依存部

TOPPERS/ASPカーネルターゲット非依存部パッケージ(相当:名古屋大学)

パッケージ	リリース日
asp-1.9.3.tar.gz	2017-04-29
release 1.9.2との違い	
asp-1.9.2.tar.gz	2015-05-30
release 1.9.1との違い	

ARM Cortex-M0アーキテクチャ・GCC依存部パッケージ (担当: TOPPERSプロジェクト教育WG)			
パッケージ	ディレクトリ	対応する非依存部のバージョン	リリース日
asp_arch_arm_m0_gcc-1.9.8.tar.gz	arch/arm_m_gcc target/om13058_gcc target/raspberrypi_pico_gcc target/stm32f091nucleo64_gcc target/stm32l073nucleo64_gcc target/stm32g071nucleo64_gcc target/stm32g0b1nucleo64_gcc target/stm32l0radiscovery_gcc	1.9.*	2025-02-25
asp_arch_arm_m0_gcc-1.9.7.tar.gz	arch/arm_m_gcc target/om13058_gcc target/raspberrypi_pico_gcc target/stm32f091nucleo64_gcc target/stm32l073nucleo64_gcc target/stm32g071nucleo64_gcc target/stm32g0b1nucleo64_gcc target/stm32l0radiscovery_gcc	1.9.*	2024-09-24
asp_arch_arm_m0_gcc-1.9.6.tar.gz	arch/arm-m_gcc target/om13058_gcc target/raspberrypi_pico_gcc target/stm32f091nucleo64_gcc target/stm32l073nucleo64_gcc target/stm32g071nucleo64_gcc target/stm32g0b1nucleo64_gcc target/stm32l0radiscovery_gcc	1.9.*	2023-02-15

TOPPERS BASE PLATFORMのダウンロード

- TOPPERSプロジェクトのWebから最新のTOPPERS BASE PLATFORM(ST/RV/RP)をダウンロードします

TOPPERS BASE PLATFORM とは

TOPPERS BASE PLATFORMは、組み込み教育用にTOPPERSプロジェクト教育ワーキンググループが開発した組み込みソフトウェアプラットフォームです。実装にあたっては、μITRON4.0仕様研究会のデバイスドライバ設計ガイドラインを参考に実装しました。プラットフォームとして、STマイクロエレクトロニクス社のSTM32マイクロコントローラとRISC-VとRaspberry PI PICOをターゲットに作成した[TOPPERS BASE PLATFORM\(ST/RV/RP\)](#)と、上級のハードウェアをターゲットとしたIntel社のCyclone V SoCをターゲットとした[TOPPERS BASE PLATFORM\(CV/RP\)](#)があります。

開発成果物のダウンロード

TOPPERS BASE PLATFORM(ST/RV/RP) : 担当TOPPERS教育WG		
パッケージ	マニュアル	リリース日
asp_baseplatform-1.5.0.tar.gz	V1.5.0 Reference manual(ST) V1.5.0 Reference manual(RV) V1.5.0 Reference manual(RP)	2025-07-29
asp_baseplatformv1.4.6_052224.tar.gz	V1.4.6 Reference manual(ST) V1.4.6 Reference manual(RV) V1.4.6 Reference manual(RP)	2024-05-22

TOPPERS/ASPの解凍

- 開発用のディレクトリ(toppers)にasp-1.9.3.tar.gz、asp_arch_arm_m0_gcc-1.9.8.tar.gzと、教材中のBASE PLATFORM(ST/RV/RP)V1.5.0(asp_baseplatform-1.5.0.tar.gz)を置きます
- asp-1.9.3.tar.gzとasp_arch_arm_m0-gcc-1.9.8.tar.gzの解凍後、asp_baseplatform-1.5.0.tar.gzを解凍します

Pico2の場合、表のarchを解凍してください

Pico2:Cortex-M33

asp_arch_arm_m33_gcc-1.9.3.tar.gz

Pico2:RISC-V

asp_arch_riscv_gcc-1.9.6.tar.gz

```
$ ls
asp_arch_arm_m0_gcc-1.9.8.tar.gz asp-1.9.3.tar.gz
asp_baseplatform-1.5.0.tar.gz
$ tar zxvf asp-1.9.3.tar.gz
$ tar zxvf asp_arch_m0_gcc-1.9.8.tar.gz
$ tar zxvf asp_baseplatform-1.5.0.tar.gz
```

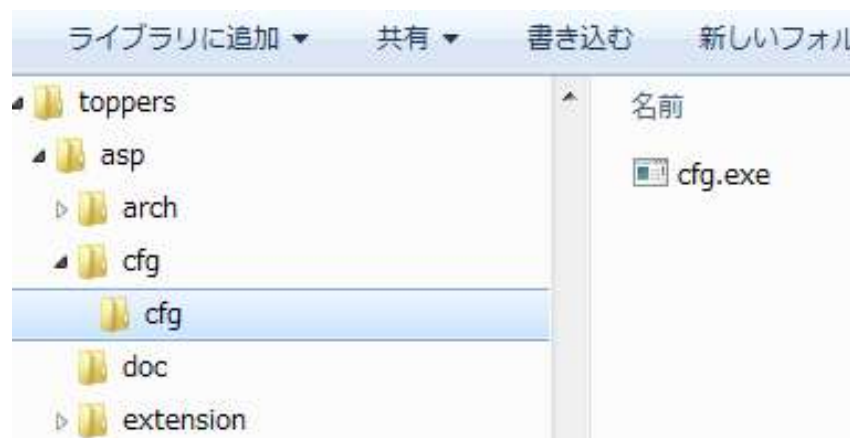
TOPPERS/ASPを構築するディレクトリの位置

- すべて解凍すると右のようなディレクトリが作成される
- デバイスドライバ開発用にはOBJディレクトリのワークスペースで開発作業を行う



コンフィギュレータのセットアップ

- ASPは静的APIをサポートしているため、静的APIをプログラミング言語に変換するコンフィギュレータが用意されています
- Windows版バイナリの最新版をダウンロード解凍して、以下のディレクトリにセットします



```
$ asp/cfg/cfg/cfg.exe
```

nel | Documents | Download | Community | Report | Contacts | FAQ

TOPPERS新世代カーネル用コンフィギュレータ

TOPPERS新世代カーネル用コンフィギュレータの最新リリースを配布しています。コンフィギュレータ自体については[こちら](#)を、構築方法等についてはアーカイブに含まれる README.txt をご覧ください。

下記32bit Linux用バイナリは、次の環境でライブラリを静的リンクしてビルドしたものです。

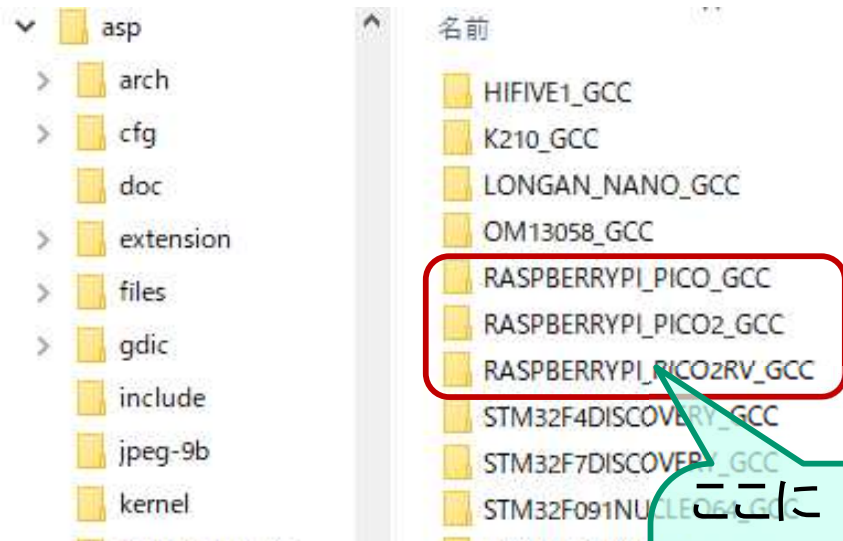
- Ubuntu 12.04 LTE 32bit
- boost 1.46.1
- Xerces C++ 3.1.1

64bitのLinux用のboostライブラリに現状問題があるため、64bitのLinuxでビルドしたcfgは正しく動作しません。64bitのLinuxでcfgを使用する場合は、下記の32bit Linux用バイナリを使用するか、32bitのLinuxでソースからビルドしてスタティックリンクしたバイナリを使用下さい。

最新のリリース			
リリース名	タイプ	サイズ	リリース日
コンフィギュレータ Release 1.9.6	tar.gz (EUC-JE)	150KB	2017-03-31
<u>コンフィギュレータ Release 1.9.6 (Windows用バイナリ)</u>	zip	10.5MB	2017-03-31
<u>コンフィギュレータ Release 1.9.6 (32bit Linux用バイナリ)</u>	gz	7.0MB	2017-03-31
Release 1.9.5との違い			

カーネルライブラリのビルド

- これから作業を行うワークディレクトリ
(OBJ/RASPBERRYPI_PICO_GCC:Pico場合)の下にカーネルライブラリ用ディレクトリ:libkernelにカーネルライブラリ(libkernel.a)を作成する
- configureコマンド実行後は、3つのファイル
Makefile,sample1.cfg,sample1.c,sample1.hが生成される



```
$ cd asp/OBJ/RASPBERRYPI_PICO_GCC
$ mkdir libkernel
$ cd libkernel
$ ../../../../configure -T raspberrypi_pico_gcc -f
$ make libkernel.a
```

ここに
libkernel.aを
作成、これ
を使うとビル
ド時間が短
縮できる

カーネルライブラリのビルド

- Pico2の場合、以下のディレクトリでカーネルライブラリを作成する

Pico2:Cortex-M33

```
$ cd asp/OBJ/RASPBERRYPI_PICO2_GCC  
$ mkdir libkernel  
$ cd libkernel  
$ ../../../../configure -T raspberrypi_pico2_gcc -f  
$ make libkernel.a
```

Pico2:RISC-V

```
$ cd asp/OBJ/RASPBERRYPI_PICO2RV_GCC  
$ mkdir libkernel  
$ cd libkernel  
$ ../../../../configure -T raspberrypi_pico2rv_gcc -f  
$ make libkernel.a
```

開発環境とハードウェアの検証

1. 開発環境の構築
2. ASP/プラットフォーム環境構築
3. タスクモニタの実行
4. CPUボードの確認
5. LCDJOYシールドの確認

ROM実行とRAM実行

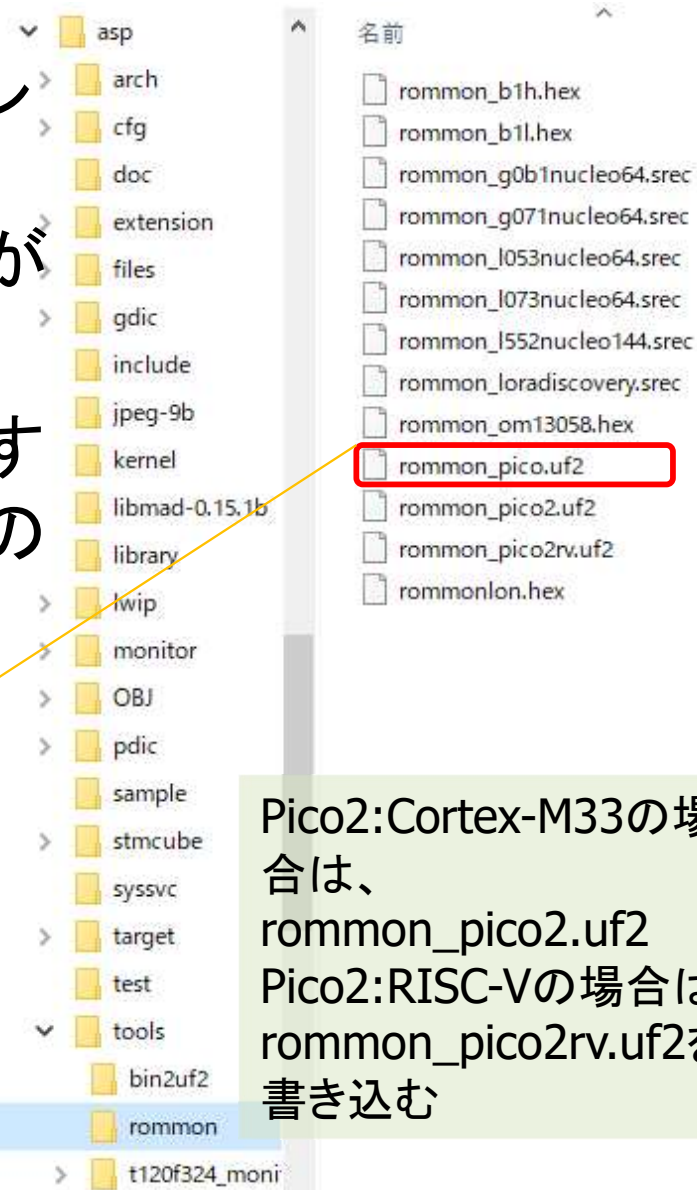
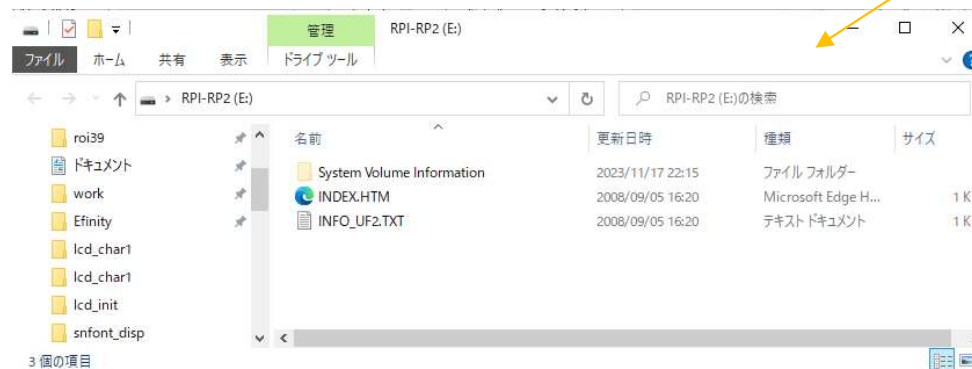
- 本実装環境は、基礎1, 2で使用したROMモニタを用いてRAM実行と、FLASH-ROMにプログラム書き込んで実行するROM実行をビルドの指定で選択できる
- 但し、RAM実行ではSRAM上にプログラムとデータの両方を格納し実行する必要があるためメモリ制約がある
- どちらを選択するかはMakefile中のDBGENV変数の指定に従う、Makefile中に指定がない場合はASPカーネルではデフォルトとしてRAM実行を選択する
- 指定がある場合は、その指定がデフォルト設定となる
- makeコマンドの変数設定でDBGENVを明示的に指定すれば、その設定でビルドすることができる

```
$ make DBGENV=ROM
```

例)ROM実行の指定

ROMモニタuf2ファイルをPicoに書き込む

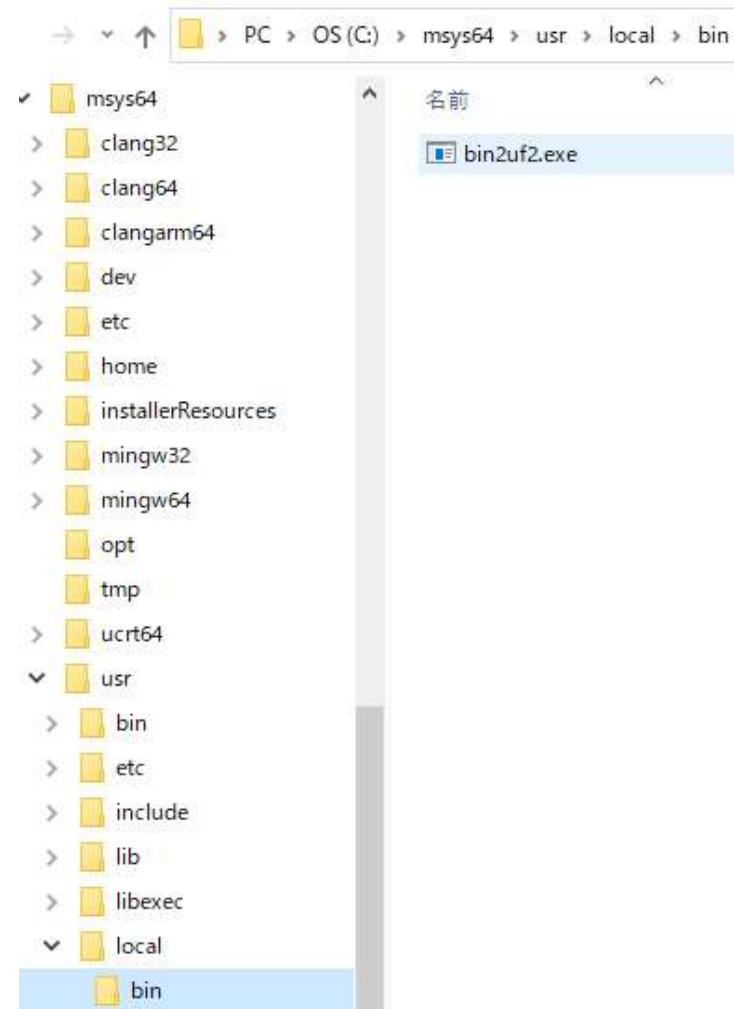
- 解凍したasp/tools/rommonディレクトリに、PICO用ROMモニタのuf2ファイル(rommon_pico.uf2)がある
- これを、BOOT-SWオンで起動するPicoのストレージディレクトリのドロップして、書き込みを行う



Pico2:Cortex-M33の場合は、
rommon_pico2.uf2
Pico2:RISC-Vの場合は、
rommon_pico2rv.uf2を
書き込む

bin2uf2コマンドの作成

- フラッシュROMに書込むには書込みデータを.uf2ファイルに変換する必要がある
- .binファイルを.uf2ファイルに変換するコマンドbin2uf2をasp/tools/bin2fu2にソースとMakefileを置いている
- このディレクトリに移動して、makeコマンドを実行するとbin2uf2.exeができる
- bin2uf2.exeをmsys64/usr/local/binに置けば、コマンドとして使用できる



MONのビルド

- ワークスペースで、MONプログラムのビルドを行う
- ROMモニタを使用してRAM実行版をビルドする
- どちらがデフォルト設定となっているかは、MakefileのDBGENVの設定による
 - DBGENV=ROMでROM実行版
 - DBGENV=RAMでRAM実行版
- 以下のmakeコマンドでRAM実行版を指定する

```
$ cd ../MON  
$ make DBGENV=RAM
```

Pico/Pico2のビルド方法

- make時のオプション設定でPico/Pico2の切り替えが可能

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時のオプション設定が面倒な場合、Makefile.configにBOARDNO設定を追加することで変更が可能

```
1 #  
2 # ターゲット略称の設定  
3 #  
4 BOARDNO = 1
```

この行を追加
Pico2:Cortex-M33に設定

LCDJOYシールドの切り替え方法

- make時のオプション設定でPico/Pico2の切り替えが可能

make LOGSHIELD=0	AdafruitのLCDを指定(デフォルト)
make LOGSHIELD=1	ロギング・シールドを使用したシールドを指定

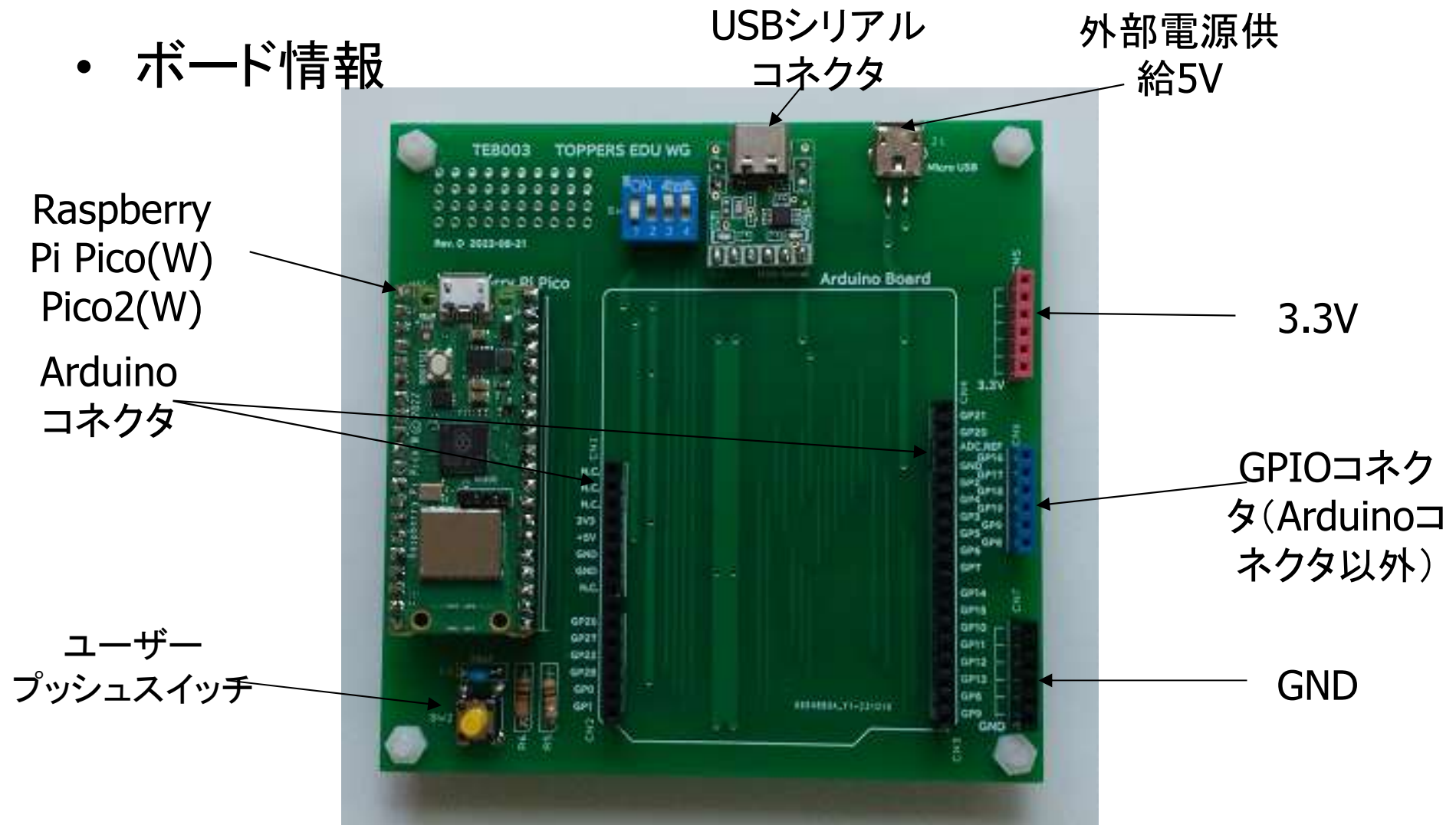
- make時のオプション設定が面倒な場合、Makefile.configにLOGSHIELD設定を追加することで変更が可能

```
38 #  
39 # ターゲット略称の設定  
40 #  
41 ifeq ($(LOGSHIELD),)  
42 LOGSHIELD=1  
43 endif
```

ロギング・シールドを使ったLCDJOY
シールドをデフォルトに設定

TEB003ボードの構成

- ボード情報



リセットピンはありません。タスクモニタでコマンドリセットが可能です(後述)

シリアルターミナルの起動

- TEB003のUSBシリアルコネクタをパソコンをUSBケーブルでつなぐとシリアルターミナル用のCOMポートが有効となる
- TeraTermを起動しシリアル設定を行い、RICOのUSBをパソコンにつなぐとROMモニタの表示が行われる

TeraTermの設定

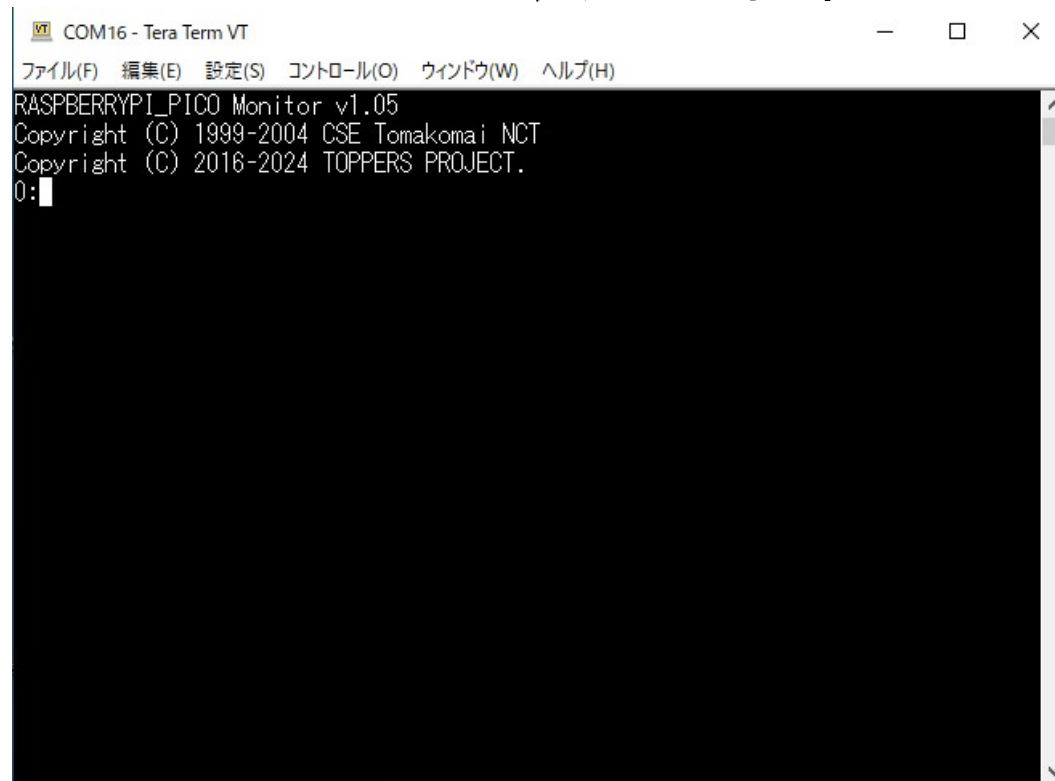
115200baud

8bit

Non parity

Stop bit1

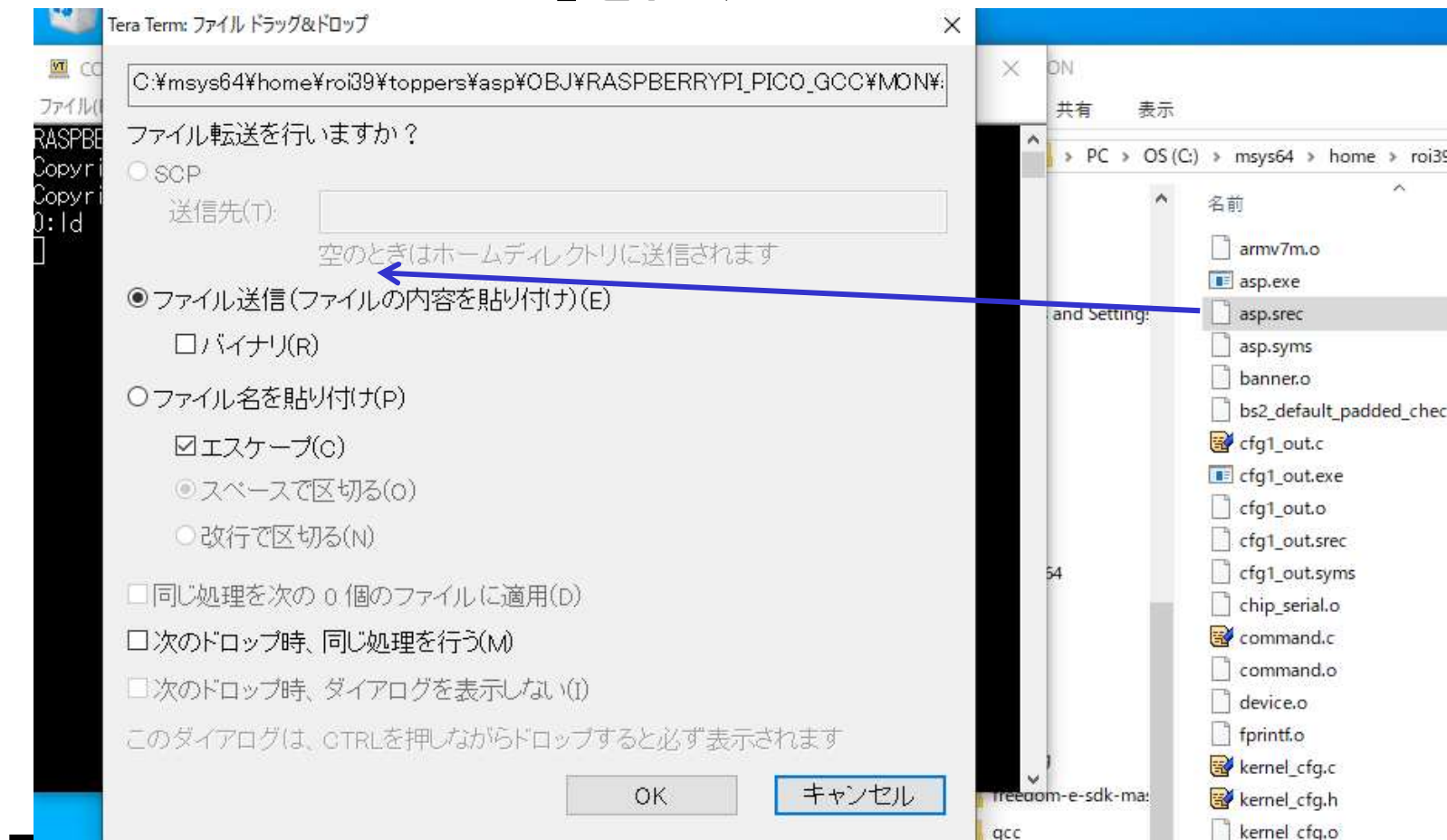
None flow



The screenshot shows a TeraTerm 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)', 'ウインドウ(W)', and 'ヘルプ(H)'. The main text area displays 'RASPBERRYPI_PICO Monitor v1.05', 'Copyright (C) 1999-2004 CSE Tomakomai NCT', 'Copyright (C) 2016-2024 TOPPERS PROJECT.', and a cursor at '0:'. The window has standard Windows controls (minimize, maximize, close) in the top right corner.

プログラムのダウンロード

- ldコマンドダウンロード設定後
- asp.srecをTera Termにドラックすると転送開始のダイアログがでるので「OK」を押す



MONを実行する

- ダウンロード終了後、goコマンドで実行するとMONが実行する
- 以下の手順でMAIN_TASKをACTIVATEするとプログラムが実行する

```
mon>display task
```

実装タスクの表示

```
mon>set task 6
```

MAIN_TASKを選択

```
mon>task activate
```

MAIN_TASKをACTIVATEに

```
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) 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 (Mar  4 2025, 13:42:59)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>display task
cur id pri state  pc      stack  inistack inisize
*  001  3 WAITING  200060dd 20011f04 20011b90 1024
  *  mon  4 RUNNING  20011a2c 20011590 1536
  003  4 DORMANT  2000022d 00000000 20010590 4096
  004  4 DORMANT  2000022d 00000000 2000f590 4096
  005  4 DORMANT  2000022d 00000000 2000e590 4096
  006  4 DORMANT  2000070d 00000000 2000d590 4096
mon>set task 6
006(DORMANT)>task activate
execute act_tsk(6) :: result = E_OK !
Sample program starts (exinf = 0).
task1 is running (001). |
006(WAITING)>task1 is running (002). |
task1 is running (003). |
task1 is running (004). |
```

開発環境とハードウェアの検証

1. 開発環境の構築
2. ASP/プラットフォーム環境構築
3. タスクモニタの実行
4. [CPUボードの確認](#)
5. LCDJOYシールドの確認

RP2040概要

- RP2040はARM Cortex-M0+2コアとしたSoCで、ラズベリーパイ財団が独自に開発しました
- 製品型番: RP2040/SC0914
- パッケージ: 56pin QFNパッケージ
- CPU: デュアルコア ARM Cortex-M0+ @133MHz
- 内部RAM: 264KB SRAM
- Flashメモリ: なし、Quad SPIによる外付け
- I/O電圧: 3.3V
- 電源電圧: 3.3V

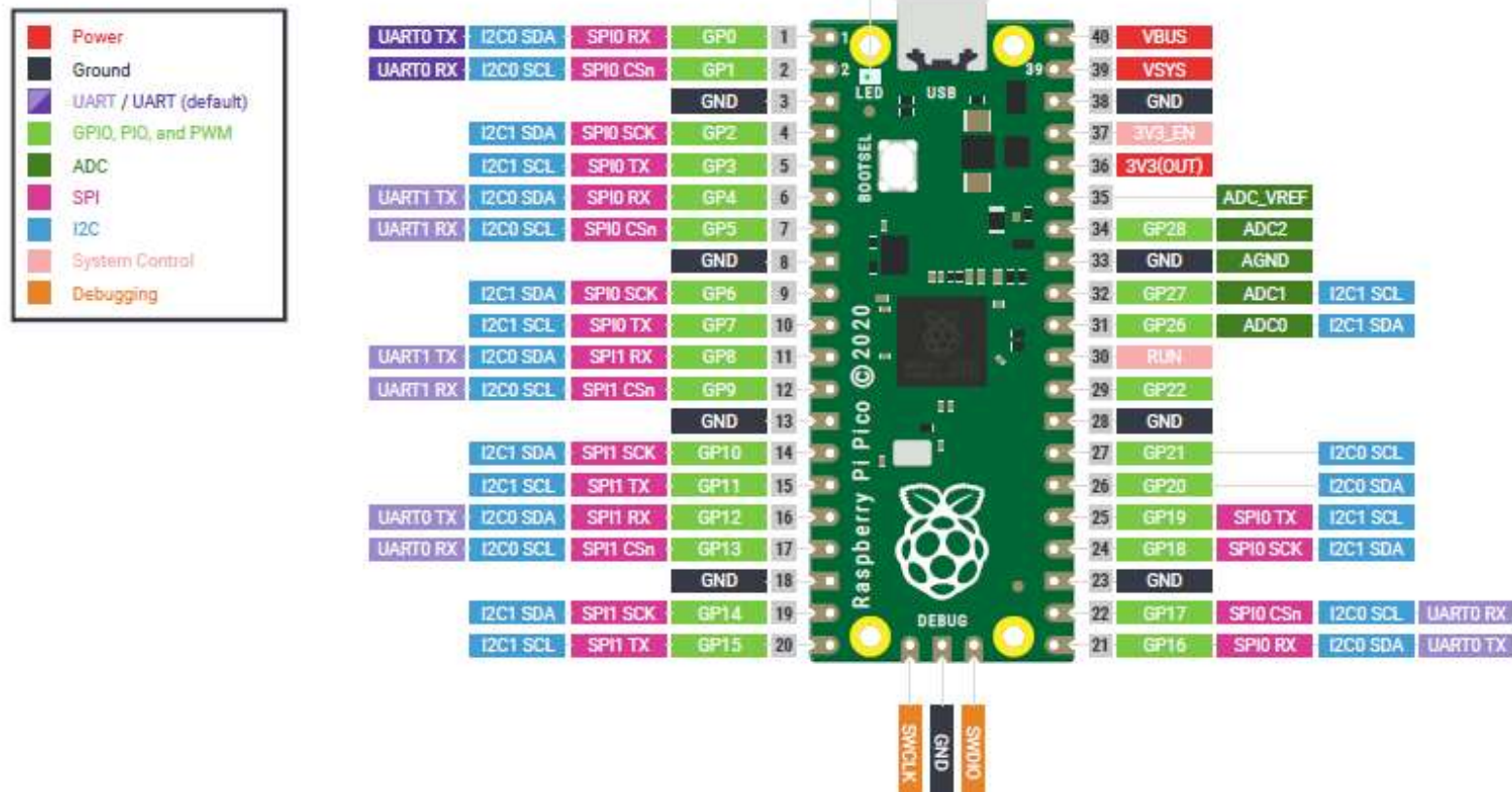


RP2040概要

- インターフェースとして以下を持ちます
 - 1 × USB 1.1 (デバイス・ホストモードに対応)
 - 30 × GPIO (内4本がアナログ対応)
 - 2 × UART
 - 2 × I2C
 - 2 × SPI
 - 16 × PWM (8 × A/Bチャネル、Bは入力対応)
 - 8 × PIO
 - 5 × ADC 12bit 500kサンプル/秒 (ひとつは内部温度センサーに接続)
 - 1 × タイマー (4 × アラート機能付き)
 - 1 × RTC
 - 1 × ウォッチドッグタイマー
 - 1 × 内部温度センサー
 - 1 × 3ピンARMシリアルワイヤデバッグ(SWD)ポート

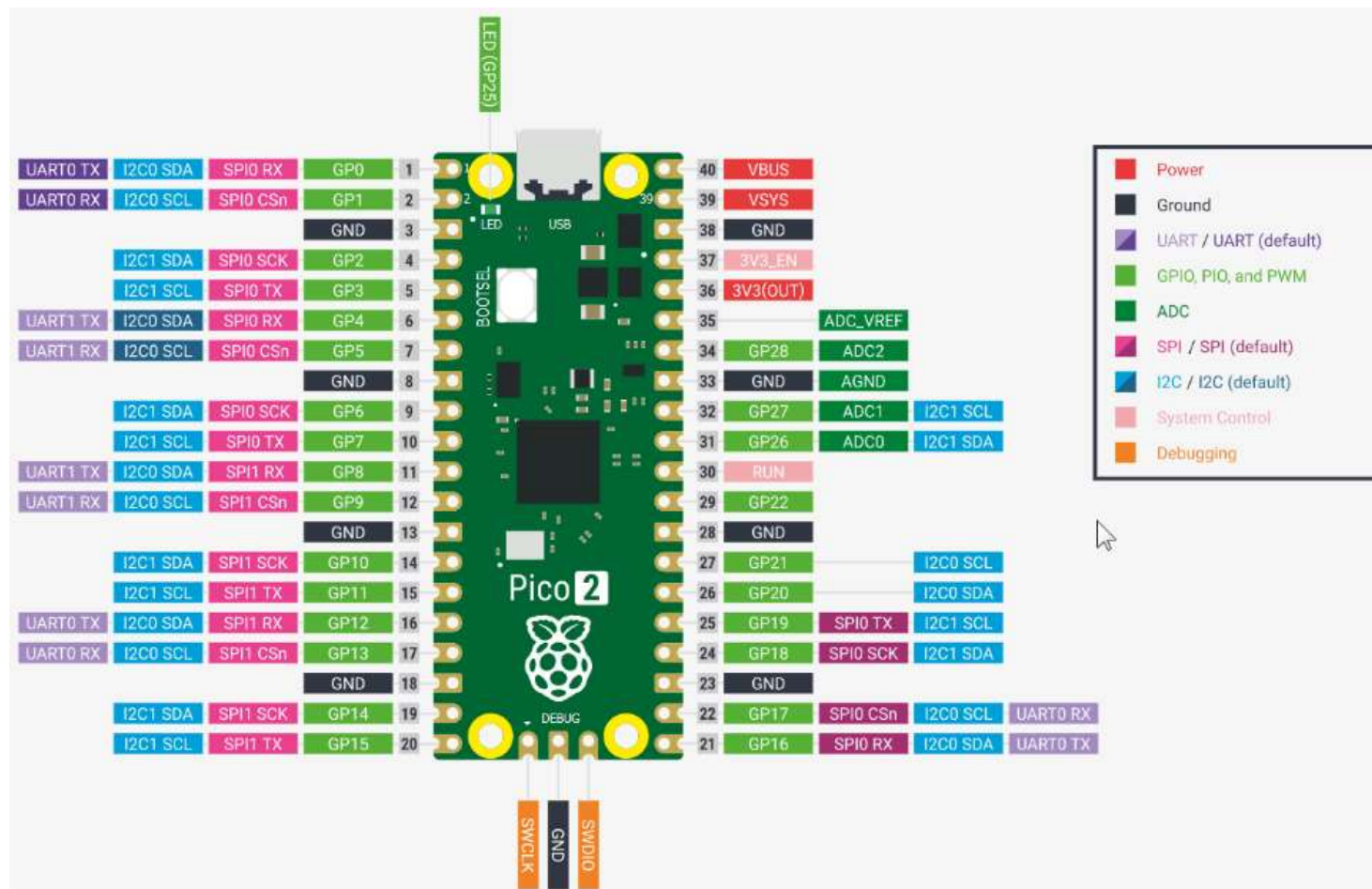
Picoのピン設定

- Raspberry Pi Picoのピンレイアウトは以下の通り
- TEB003ではGP0,1,22をUARTとプッシュスイッチとして使用
- 給電はUSBとVBUSを使用できます



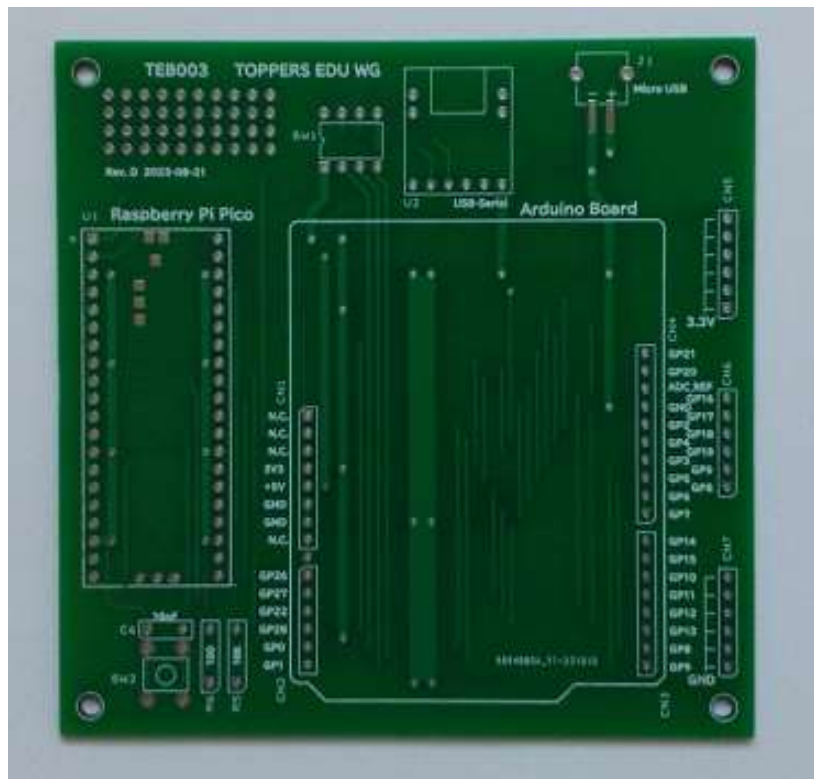
Pico2のピン設定

- Raspberry Pi Pico2のピンレイアウトはPicoと同様
- Raspberry Pi Pico2はTEB003ボード上で動作させます



TEB003ボード

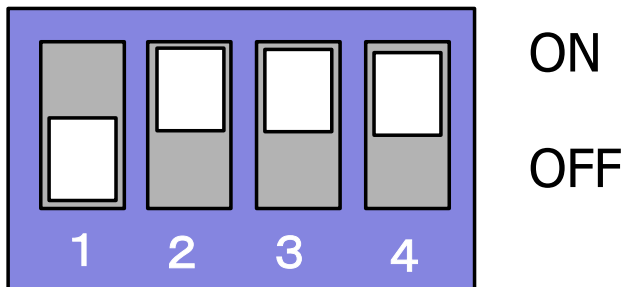
- Raspberry Pi Pico(W)またはPico2(W)を装着し、Arduinoコネクタが利用できる
- UART0に対応したUSBシリアル変換器を持つ
- 外部5V電源により駆動可能



DIP-SWの設定

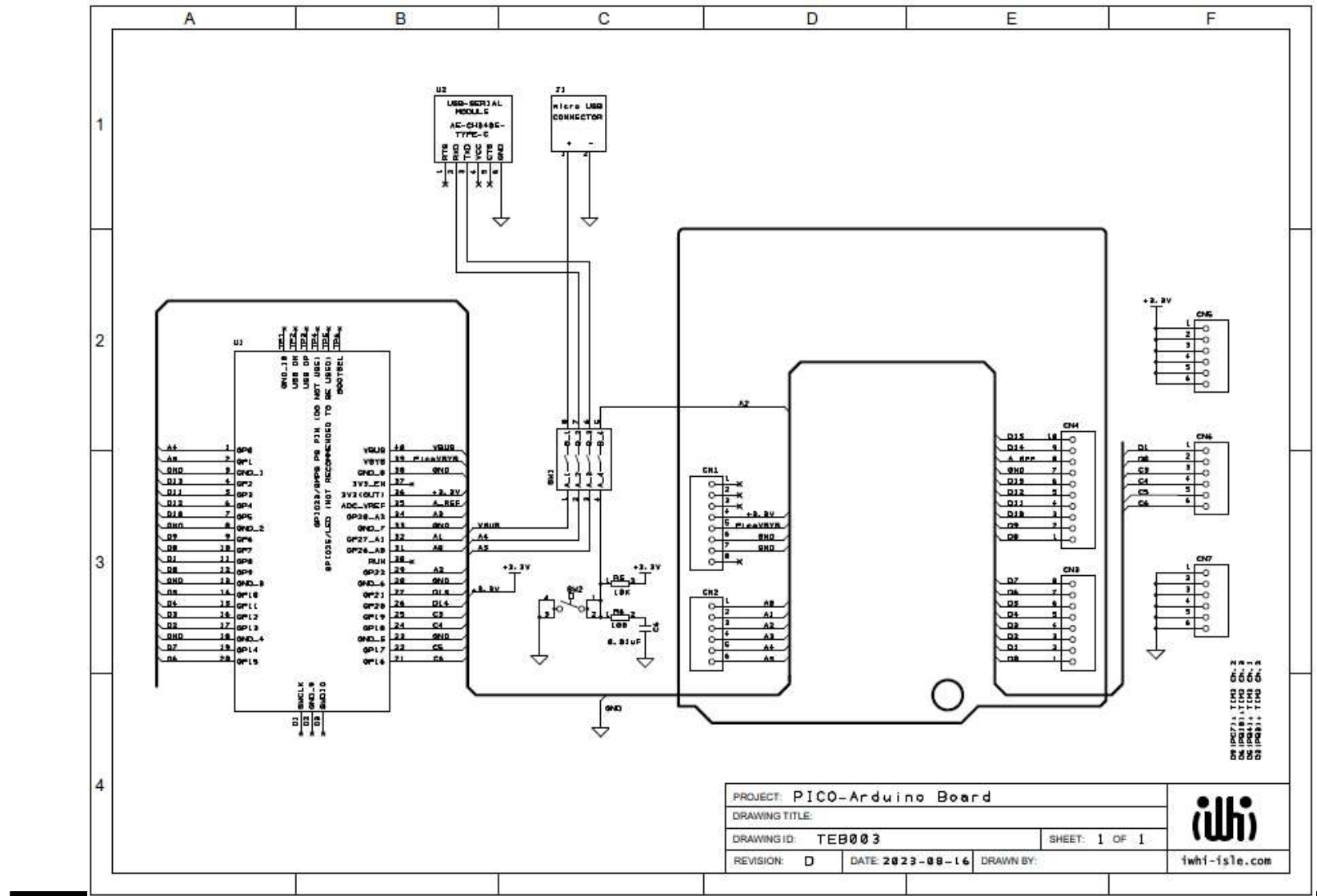
- TEB003のDIP-SWは以下の設定です
- DIP-SWの1は通常OFFですが、USBホストを実行する場合、Pico/Pico2のUSBのVBUSは給電となるため、DIP-SWの1をONにして、外部給電に切り替えます

番号	ON/OFFの設定	有効機能
1	ONで外部電源をVBUSに接続	外部電源より給電
2	ONでUSBシリアルのRXをGP0に接続	USBシリアル有効化
3	ONでUSBシリアルのTXをGP1に接続	USBシリアル有効化
4	ONでプッシュスイッチをGP22に接続	プッシュスイッチ有効化



TEB003回路図

- Pico/Pico2のGPIOをArduinoコネクタの仕様に準じて接続

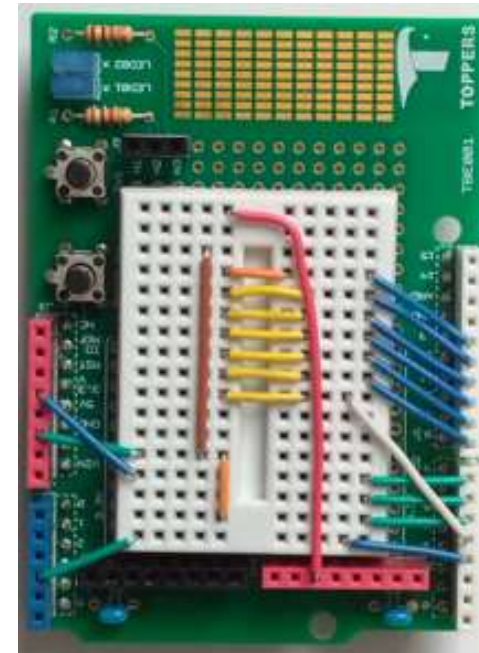


開発環境とハードウェアの検証

1. 開発環境の構築
2. ASP/プラットフォーム環境構築
3. タスクモニタの実行
4. CPUボードの確認
5. [LCDJOYシールドの確認](#)

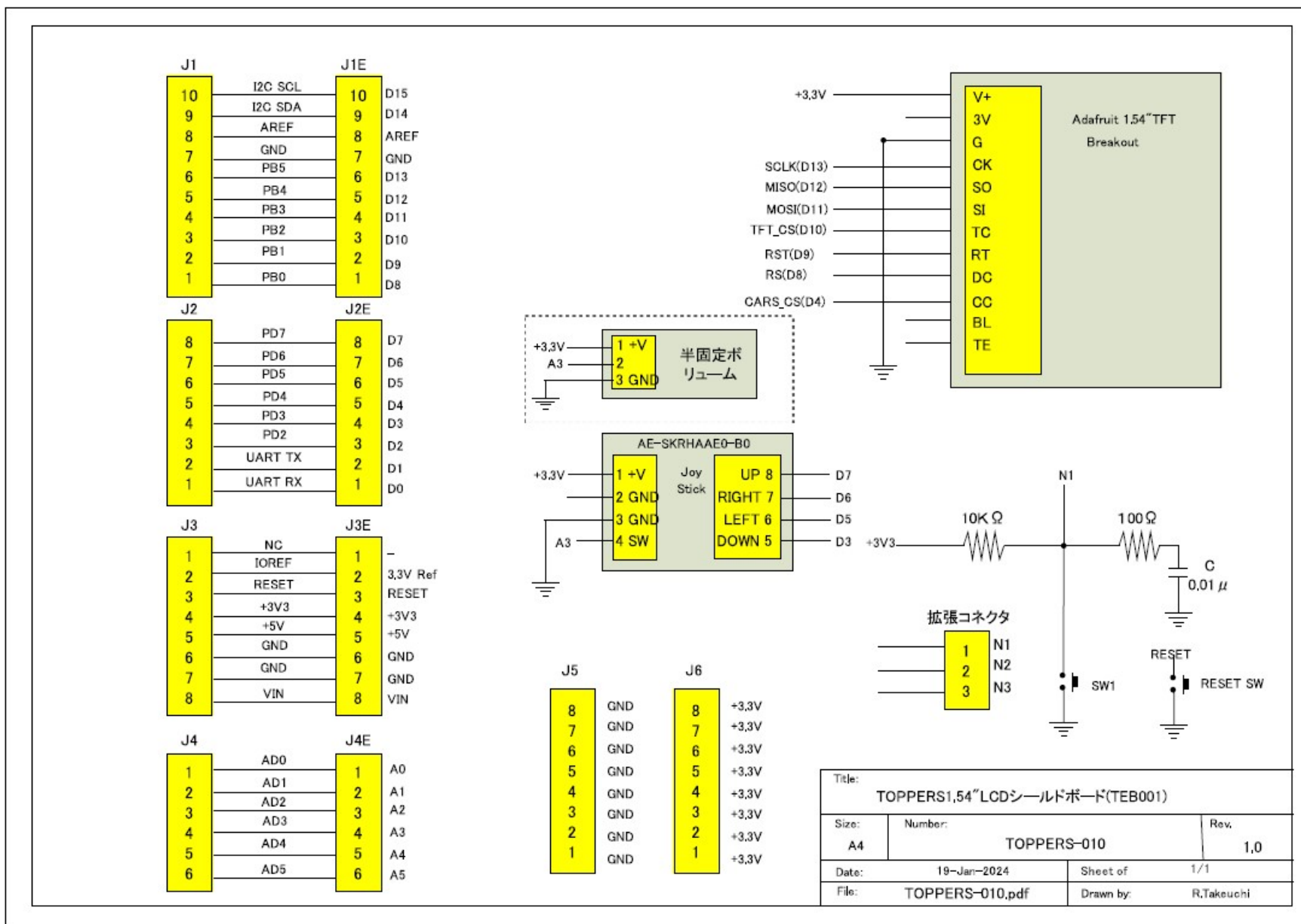
LCDJOYシールド概要

- LCDJOYシールドは、プロトタイプシールドに以下の部品を装着して、LCD/SDカード/JOY STICKの機能に対応している
 - Adafruit 1.54” TFT BreakOutボード
 - AE-SKRHAAE010-B0ジョイスティック
- 作成するには回路図に従ったジャンパーケーブルの実装を行う必要がある



プロトタイプシールドの配線

LCDJOYシールド回路図



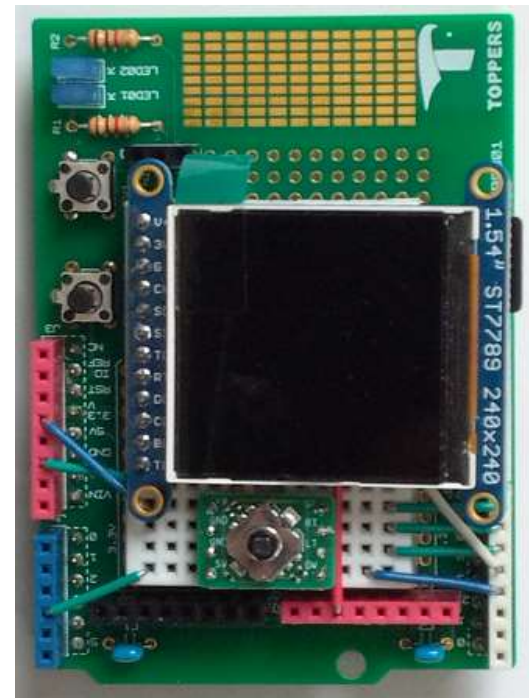
LCDとSDカードの制御

- SPIの通信ラインがLCDは直接、SDカードはHC4050Mを通して降圧して接続している
- 下位層のドライバをSPIとして、2つのデバイスを排他制御しながら上位層のドライバで制御すればLCDとSDカードの制御が可能

Arduino	信号機能	LCD	SDカード
D4	CS2		CARD_CS
D10	CS1	TFT_CS	
D11	MOSI	SO	DAT0
D12	MISO	SI	CARD-MISO
D13	SCLK	SCK	CARD-SCK

Joy Stickの制御

- 回路上のA3(オンオフスイッチ)の電圧は、Joy Stickの押下に対応して、電圧降下が発生する
 - ADCの講座では、Joy Stickを可変抵抗に入れ替えて電圧測定を行う
- 上下、左右の位置は以下のGPIOの入力により検知が可能となる
 - UP D7:GPIO14
 - RIGHT D6:GPIO13
 - LEFT D5:GPIO10
 - DOWN D3:GPIO12



LCDとJoy Stickモジュールをセット

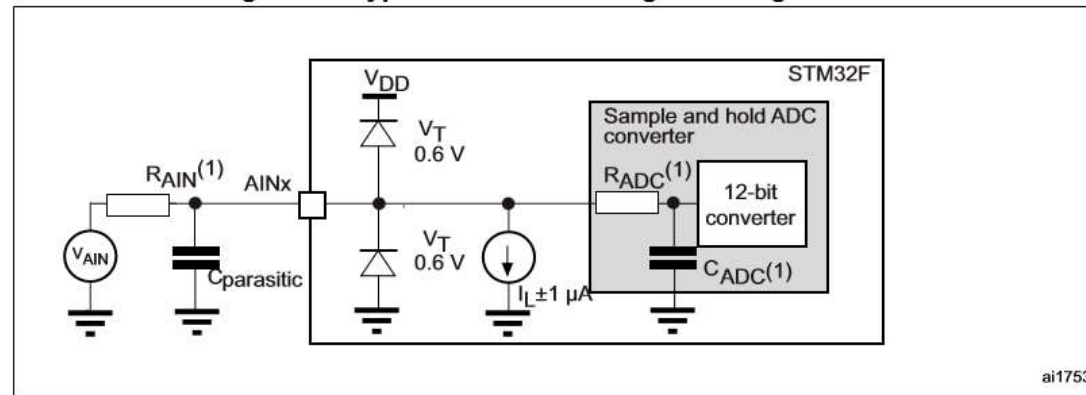
可変抵抗を使ってADC入力

1. [RP2040-ADC仕様](#)
2. ADCドライバの説明
3. adc1プログラム
4. adc1のビルドと確認
5. 位置取り込みプログラム作成

アナログ-デジタル変換回路

- アナログ-デジタル変換回路 (A/Dコンバータ: Analog-to-digital converter: ADC) とは、アナログ電気信号をデジタル電気信号に変換する電子回路を指す
- ADCは、使用目的によりいろいろな回路がある
- 一般にマイコンで使用するADCは逐次比較型と呼ばれ、アナログ電圧をデジタル値に変換するものである
- 入力のアナログ電圧は時系列に変化するため、サンプリングによりデジタル値を取り出す

Figure 41. Typical connection diagram using the ADC

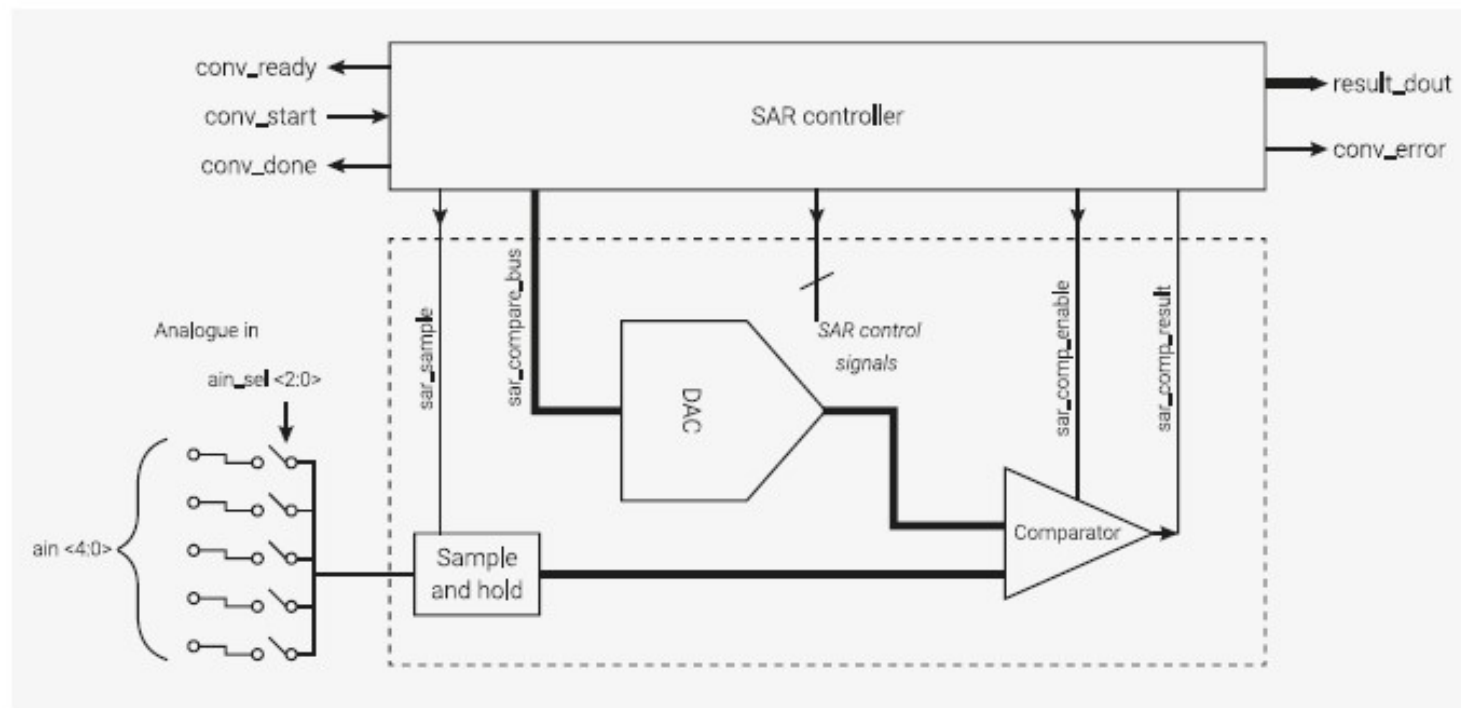


STM32F401のデータブックより
AINxは12ビットのデジタル値に変換される

1. Refer to [Table 66](#) for the values of R_{AIN} , R_{ADC} and C_{ADC} .
2. $C_{parasitic}$ represents the capacitance of the PCB (dependent on soldering and PCB layout quality) plus the pad capacitance (roughly 5 pF). A high $C_{parasitic}$ value downgrades conversion accuracy. To remedy this, f_{ADC} should be reduced.

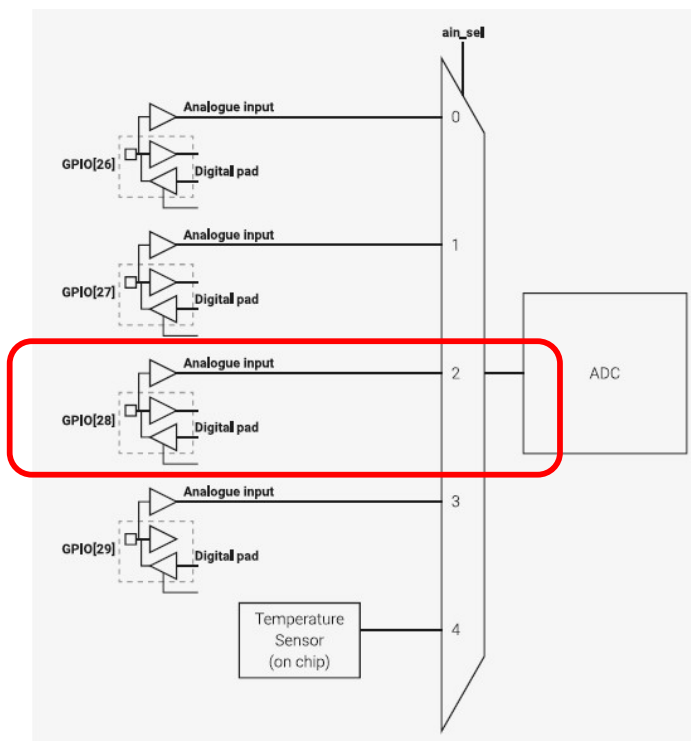
RP2040ADCロジック

- RP2040は右図のADCロジックは1つ(ADC1としている)で、5つのチャネルを選択可能
- サンプリングクロックは48MHz(USB用)
- Picoの解説ではADC_VREF+は3.3V~3.0V、GNDに接続しているので、測定範囲は0V~3.3Vとなる



可変抵抗のアナログ入力

- ADCの実習ではJoy Stickを可変抵抗に載せ替えて検証を行う、アナログ入力ピンはArduino connectorのA3ピン(GPIO28)なので、ADCの2チャンネルに対応している
- ADC設定、チャンネル設定を行えば可変抵抗を調整して電圧をデジタル値に変換することができる



可変抵抗を使ってADC入力

1. RP2040-ADC仕様
2. [ADCドライバの説明](#)
3. adc1プログラム
4. adc1のビルドと確認
5. 位置取り込みプログラム作成

ADC1を動作させるためのリソース

- ADC1を制御するために、関連ハードウェアの設定が必要
- TOPPERS BASE PLATFORM(RP) ADCドライバでは、ADC割込みと変換データ転送用にDMAを使用する
- 割込み設定とDMA設定はADCドライバ内で行う、静的APIにて割込み設定が必要
- ADCドライバは、`pdic/rp2040/adc.c`に記載
- DMAドライバは、`pdic/rp2040/device.c`に記載
 - これらをリンクする必要がある

Pico2ではrp2350

番号1はPico、番号2は
Pico2:Cortex-M33、
RISC-Vの場合、番号2-16となる

使用割込み	デファイン名	番号1	番号2	管理種別
ADC割込み	IRQ_VECTOR_ADC_FIFO	38	51	割込みハンドラ
DMA割込み	IRQ_VECTOR_DMA0	27	26	割込みハンドラ

ADCドライバの構成

- Arduinoでは、アナログピンを指定してanalogRead関数にてデータ読み込めば、読み込んだ時点での変換データが読み込める
- TOPPERS BASE PLATFORM(RP)ではより複雑なサンプリングやトリガを設定できるように初期設定を行わなければならない

書式	機能	返回值
void adc_int_handler(void);	ADC割込みハンドラ	なし
ADC_Handle_t *adc_init(ID portid ADC_Init_t *pini);	ADC初期化関数	ハンドラへのポインタ
ER adc_deinit(ADC_Handle_t *ph);	ADCドライバ終了	E_OK
ER adc_start_int(ADC_Handle_t *ph);	ADC割込み開始設定	E_OK
ER adc_end_int(ADC_Handle_t *ph);	ADC割込み終了	E_OK
ER adc_start_dma(ADC_Handle_t *ph, uint32_t *pdata, uint32_t length);	ADC DMA開始設定	E_OK
ER adc_end_dma(ADC_Handle_t *ph);	ADC DMA終了設定	E_OK
ER adc_setupchannel(ADC_Handle_t *ph, ADC_Channel_Conf_t *pconf);	ADC チャンネル設定	E_OK
ER adc_selectchannel(ADC_Handle_t *ph, uint8_t channel);	ADCウオッチドック設定	E_OK

可変抵抗を使ってADC入力

1. RP2040-ADC仕様
2. ADCドライバの説明
3. [adc1プログラム](#)
4. adc1のビルドと確認
5. 位置取り込みプログラム作成

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（一日目）
 - 教材ディレクトリ/base3/program

ADC

adc1 : 基本ADコンバータプログラム

SPI/LCD

spi1 : spi LOOP-BACKテストプログラム

lcd_init : LCD初期化プログラム

lcd_char1 : LCD文字表示ベース

lcd_text1 : LCD文字列表示ベース

lcd_text4 : LCD文字列漢字表示(ファイル)対応

演習環境の準備

- TOPPERS BASE PLATFORMをインストールしたaspディレクトリに並列して演習用のbase3ディレクトリをコピーする
- base3/program/adc1がadc1プログラムのワーク領域となる
- 実習はbase3/my_programディレクトリを作成しadc1とcommand.cとMakefile.configをコピーして行う

		名前	更新日時	種類	サイズ
▼	toppers				
>	asp				
>	base				
▼	base3				
	filetest1				
>	my_program				
>	program				
>	fmp_1.4.1				
>	jsp-1.4.4.1-full				
>	senior_1.4.0				
>	senior1				
>	senior2				
		adc1	2025/03/04 14:35	ファイル フォルダ	
		CupRamenTimer	2025/03/04 15:56	ファイル フォルダ	
		lcd_char1	2025/03/04 15:25	ファイル フォルダ	
		lcd_init	2025/03/04 14:14	ファイル フォルダ	
		lcd_text1	2025/03/04 15:40	ファイル フォルダ	
		lcd_text4	2025/03/04 15:39	ファイル フォルダ	
		lcdjoyshield	2025/03/04 16:03	ファイル フォルダ	
		sd_file1	2025/03/04 16:25	ファイル フォルダ	
		sd_ini1	2025/03/04 16:25	ファイル フォルダ	
		spi1	2025/03/04 15:11	ファイル フォルダ	
		command.c	2025/03/04 13:59	C ファイル	7 KB
		Makefile.config	2025/03/04 14:10	CONFIG ファイル	1 KB

adc1解説

- adc1は単純なADコンバータとなる
 - ADCロジックは複雑なコンバージョンが可能
 - チャンネル切り替えで、複数のチャンネル入力が可能
 - サンプルング周期を設定可能
- adc1仕様
 - A3:ADC1チャンネル2を入力
 - サンプルング周期を設定してDMAを用いて10ms間の連続入力し、最後にログ出力を行う
 - 平均入力値から電圧を計算してログ出力する
 - この動作を100ms待ち1000回繰り返す

adc1ワーク領域の構成

- adc1のプログラムはprogram/adc1内にある
- ビルドの補助をしたり、タスクモニタの拡張を行うプログラムがbase3/programの下にある
- ADCのデバイスドライバはasp/pdic/rp2040内にあり、adc.c以外にも多くのプログラムを使用している

ファイル名	ファイルの内容	設定場所
adctest.h	adc1プログラムのインクルードファイル	/base3p/program/adc1
adctest.cfg	adc1プログラムコンフィギュレーションファイル	/base3p/program/adc1
adctest.c	adc1プログラムのソースファイル	/base3p/program/adc1
Makefile	adc1プログラムのメイクファイル	/base3p/program/adc1
Makefile.config	Makefileからインクルードされる環境設定ファイル	/base3p/program
command.c	DEVICEコマンド拡張ソースファイル	/base3p/program
adc.h	ADコンバータドライバインクルードファイル	/asp/pdic/rp2040
adc.c	ADコンバータドライバソースファイル	/adc/pdic/rp2040

adctest.h: インクルードファイル

- チャンネルの定義とADC,DMAの設定定義を行う

```
/*
 * ADC1チャンネル定義
 */
#define ADC1_CHANNEL2      2
#define ADC1_CHANNEL2_PINNO 28

#define THREAH_FIFO  6

#define DMA_INTNO    DMA_INT0
#define INHNO_DMAADC (IRQ_VECTOR_DMA0+DMA_INTNO) /* 割込みハンドラ番号 */
#define INTNO_DMAADC (IRQ_VECTOR_DMA0+DMA_INTNO) /* 割込み番号 */
#define INTPRI_DMAADC -3 /* 割込み優先度 */
#define INTATR_DMAADC TA_EDGE /* 割込み属性 */

#define INHNO_ADC    IRQ_VECTOR_ADC_FIFO /* 割込みハンドラ番号 */
#define INTNO_ADC    IRQ_VECTOR_ADC_FIFO /* 割込み番号 */
#define INTPRI_ADC    -2 /* 割込み優先度 */
#define INTATR_ADC    0 /* 割込み属性 */
```

ADCチャンネル定義

割込みの定義

adctest.cfg:コンフィギュレーションファイル

- adc1プログラムで使用するOSリソースを静的APIで定義
 - セマフォ、タスク、割り込み定義
- ADCで使用するADCとDMAの割り込みは最適な設定をユーザーが定義する

```
#include "device.h"
#include "adctest.h"
```

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

```
CRE_SEM(ADCDMA_SEM, { TA_TPRI, 0, 1 });
```

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

```
ATT_ISR({TA_NULL, DMA_INTNO, INTNO_DMAADC, dma_isr, 1 });
CFG_INT(INTNO_DMAADC, { TA_ENAINT | INTATR_DMAADC, INTPRI_DMAADC });
DEF_INH(INHNO_ADC, { TA_NULL, adc_int_handler });
CFG_INT(INTNO_ADC, { TA_ENAINT | INTATR_ADC, INTPRI_ADC });
```

転送終了通知用セマフォを
ADCDMA_SEMで設定

プログラム実行用の
タスクを定義

割り込みハンドラの定義
静的APIにて、ADCとDMA
の割り込みハンドラとサービ
スルーチンを定義

adctest.c: ソースファイル1

- 10ms分のコンバート結果を保存する領域の定義
 - ADC1の周波数は48MHzの1000分周(分周値は変更可能)
 - サンプリングクロックは1msで48回(可変)
 - 10ms分の転送領域を計算して確保
- コンバージョン終了時のコールバック関数の作成

```
#define NUM_ADCBUF 48*10 /* 10ms */  
  
volatile uint32_t uhADCxConvertedValue[NUM_ADCBUF];  
  
/*  
 * ADC転送終了コールバック関数  
 */  
static void  
Adcconvert_callback(ADC_Handle_t* hadc)  
{  
    syslog_4(LOG_NOTICE, "0(%d) 1(%d) 2(%d) 3(%d)", uhADCxConvertedValue[0], uhADCxConvertedValue[1],  
            uhADCxConvertedValue[2], uhADCxConvertedValue[3]);  
    SVC_PERROR(isig_sem(ADCDMA_SEM));  
}
```

変換データ格納領域

転送終了時コールバック関数

adctest.c: ソースファイル2

- エラー発生時コールバック関数を作成

```
/*  
 * ADCエラーコールバック  
 */  
static void  
adcerror_callback(ADC_Handle_t* hadc)  
{  
    syslog_1(LOG_EMERG, "ADC Error callback(%08x)", hadc);  
}
```

エラー時コールバック関数

adctest.c: ソースファイル3

- 平均、電圧計算関数の作成

```
/*
 * 平均値の計算
 */
static float
make_sum(int *ptmp1, int *ptmp2)
{
    float v, sum;
    uint32_t i;
    int tmp1, tmp2;

    for(i = 0, sum = 0.0 ; i < NUM_ADCBUF ; i++)
        sum += uhADCxConvertedValue[i];
    sum /= NUM_ADCBUF;
    v = (sum * 3.3) / 4096.0;
    tmp1 = (int)v;
    if(v < 0.0)
        tmp2 = (tmp1 - v) * 100;
    else
        tmp2 = (v - tmp1) * 100;
    *ptmp1 = tmp1;
    *ptmp2 = tmp2;
    return sum;
}
```

adctest.c:メイン関数

- ADC1の初期化をadc_init関数で行う
- 初期化後、コールバック関数を登録

```
/*  
 * ADCの初期化  
 */  
aInit.ClockPrescaler = 1000;  
aInit.FIFOMode      = 0;  
aInit.ConvertCount = 1;  
aInit.RxDmaChannel = 0;  
if((hadc = adc_init(ADC1_PORTID, &aInit)) == NULL){  
    syslog_0(LOG_NOTICE, "## adc_init ERROR ##");  
    slp_tsk();  
}  
hadc->xfercallback = ADC_ConvCpltCallback2;  
hadc->errorcallback = ADC_ErrorCallback;  
  
/*  
 * ADCチャネル設定  
 */  
sConfig.Channel = ADC1_CHANNEL2;  
sConfig.GpioPin = ADC1_CHANNEL2_PINNO;  
if((ercd = adc_setupchannel(hadc, &sConfig)) != E_OK){  
    syslog_1(LOG_NOTICE, "## adc_setupchannel2 ERROR(%d) ##", ercd);  
    slp_tsk();  
}
```

1000分周を設定

継続モード設定

DMAを有効

割り込み用のコールバック関数の設定

チャネル2を設定

adctest.c: メイン関数

- 2つのチャネルの設定を行う
- 可変抵抗値の取り込み
 - チャネル2を設定し、DMAをスタートする
- セマフォを使って、終了割込みを待つ

```
/*
 * コンバージョンスタート
 */
for(i = 0 ; i < 1000 ; i++){
    /*
     * コンバージョンスタート#2
     */
    memset((void *)uhADCxConvertedValue, 0, NUM_ADCBUF*4);
    adc_selectchannel(hadc, ADC1_CHANNEL2);
    get_tim(&tim1);
    if((ercd = adc_start_dma(hadc, (uint32_t*)uhADCxConvertedValue, NUM_ADCBUF)) != E_OK){
        syslog_1(LOG_NOTICE, "### adc_start_dma ERROR(%d) ##", ercd);
        slp_tsk();
    }
    while(hadc->status == ADC_STATUS_BUSY){
        twai_sem(ADCDMA_SEM, 100);
    }
    get_tim(&tim2);
}
```

可変抵抗用チャンネル

adctest.c: メイン関数

- 終了後、DMAを停止
- 平均値、電圧を計算して表示する
- 100ms待つ

```
/*
 * コンバージョンエンド#2
 */
ercd = adc_end_dma(hadc);
if(ercd != E_OK){
    syslog_1(LOG_NOTICE, "## adc_end_dma ERROR(%d) ##", ercd);
    slp_tsk();
}
sum = make_sum(&vint, &dint);
syslog_5(LOG_NOTICE, "ch1 dma(%d) time(%d) avr1(%d) v1(%d.%02d)", i, tim2-tim1, (int)sum, vint, dint);
dly_tsk(100);
}
```

可変抵抗を使ってADC入力

1. RP2040-ADC仕様
2. ADCドライバの説明
3. adc1プログラム
4. [adc1のビルドと確認](#)
5. 位置取り込みプログラム作成

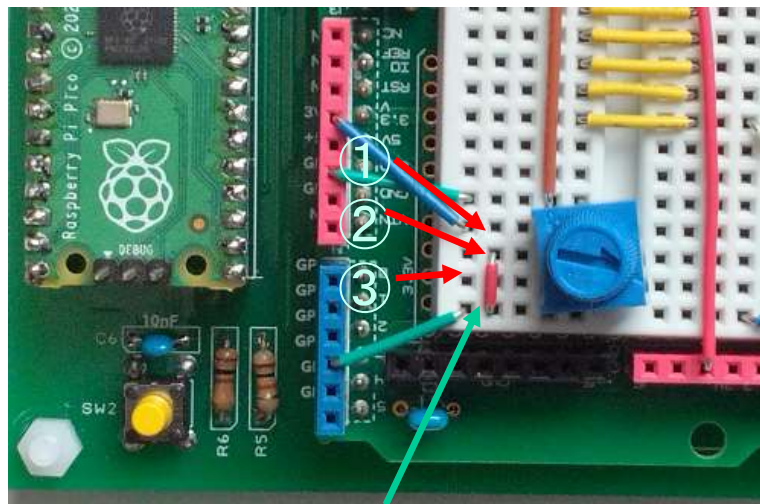
演習 : adc1 のワーク領域作成

- base3の下に、my_programディレクトリを作成
- my_programに以下のファイルとディレクトリをコピー
 - command.c
 - Makefile.config
 - adc1ディレクトリ
- コピーしたadc1に入り、ビルドする

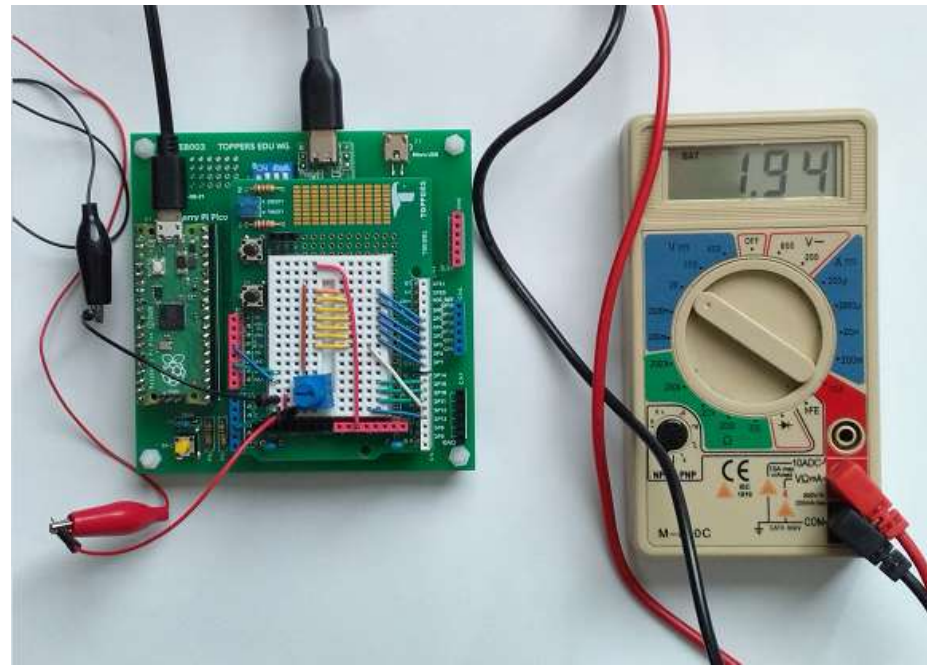
```
$ cd base3/my_program/adc1  
$ make
```

演習：可変抵抗の電圧確認

- Joy Stickを外し、A3ピンがGNDと3.3Vの中央に来るようにジャンパーピンを付けます
- 以下のピン位置になるように可変抵抗を付けます
 - 1ピン:3.3V
 - 2:ピン:A3(ジャンパーピンを付けたところ)
 - 3:GND



ジャンパーピン



筆者の環境でテスターで電圧測定(GNDとA3)

演習：可変抵抗の電圧確認

- adc1をビルド実行して、可変抵抗ダイアルの位置による電圧を確認する
- ADC1のコンバージョンビットは12ビット: avr2(0~4095)、比例計算で電圧:v1を求める
 - 3V3が3.29V
 - GNDが0V



可変抵抗ダイヤル	コンバージョン値	比例計算値	テストの測定値
上にいっぱい	7	0.00	0.00
中央	2161	1.74	1.94
下にいっぱい	4093	3.29	3.58

演習: DMAを使用しないAD変換

- 割込みによるAD変換プログラムを作成する
- 変換データはDMAを使用しない
 - DMAは共有なので、他のデバイスで使用可能
- 学習内容
 - ADC割込みモード設定
- 手順
 - adc1をコピーしadc2を作成、これを修正して作る
 - コンフィギュレーションファイルのDMA割込みを削除
 - 割込み開始停止設定

演習: コンフィギュレーションファイルの改造

- ADC用セマフォADC_DMA_SEMをADC_SEMに変更
- DMAの割込みをコメント化で削除

```
#include "device.h"
#include "adctest.h"

ATT_INI({ TA_NULL, 0, device_info_init });
ATT_INI({ TA_NULL, sw_handle, setup_sw_func });

CRE_SEM(ADC_SEM, { TA_TPRI, 0, 1 });

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

/*
ATT_ISR({TA_NULL, DMA_INTNO, INTNO_DMAADC, dma_isr, 1 });
CFG_INT(INTNO_DMAADC, { TA_ENAINT | INTATR_DMAADC, INTPRI_DMAADC });
*/

DEF_INH(INHNO_ADC, { TA_NULL, adc_int_handler });
CFG_INT(INTNO_ADC, { TA_ENAINT | INTATR_ADC, INTPRI_ADC });
```

演習:コンバート回数をカウントする変数を追加

- ADC割込みの回数をカウントする変数convert_countを追加する

```
#define NUM_ADCBUF 48*10                                /* 10ms */  
  
volatile uint32_t uhADCxConvertedValue[NUM_ADCBUF];  
volatile uint32_t convert_count;
```

演習:コールバック関数変更

- コールバック関数に以下の処理を追加する
 - 変換データの取り込み
 - 取り込んだ変換値がマイナスならば無効とする(FIFOオーバー)
 - 割込み終了設定

```
/*
 * コンバージョン終了コールバック関数
 */
static void
adcconvert_callback(ADC_Handle_t* hadc)
{
    int32_t value;

    for( ; convert_count < NUM_ADCBUF ; convert_count++){
        value = adc_getvalue(hadc);
        if(value < 0)
            break;
        uhADCxConvertedValue[convert_count] = value;
    }
    if(convert_count >= NUM_ADCBUF)
        isig_sem(ADC_SEM);
}
```

演習: 割込みモード設定

- 割込み用に初期化を変更
 - ConvertCountを設定
 - DMAのチャンネルを-1に設定(DMA未使用)

```
175  /*
176   * ADCの初期化
177   */
178  aInit.ClockPrescaler = 1000;
179  aInit.FIFOMode      = 0;
180  aInit.ConvertCount = NUM_ADCBUF;
181  aInit.RxDmaChannel = -1;
182  if((hadc = adc_init(ADC1_PORTID, &aInit)) == NULL){
183      syslog_0(LOG_NOTICE, "## adc_init ERROR ##");
184      slp_tsk();
185  }
186  hadc->xfercallback = adcconvert_callback;
187  hadc->errorcallback = adcerror_callback;
```

ConvertCountを設定
DMAを無効にする(マイナスの設定)

演習 : adctest.c (割り込み用のadc関数に変える)

```
202  for(i = 0 ; i < 50000 ; i++){
203      /*
204      * コンバージョンスタート#2
205      */
206      memset((void *)uhADCxConvertedValue, 0, NUM_ADCBUF*4);
207      convert_count = 0;
208      adc_selectchannel(hadc, ADC1_CHANNEL2);
209      get_tim(&tim1);
210      if((ercd = adc_start_int(hadc)) != E_OK){
211          syslog_1(LOG_NOTICE, "## adc_start_int ERROR(%d) ##", ercd);
212          slp_tsk();
213      }
214      while(hadc->status == ADC_STATUS_BUSY){
215          twai_sem(ADC_SEM, 100);
216      }
217      get_tim(&tim2);
218
219      /*
220      * コンバージョンエンド#2
221      */
222      ercd = adc_end_int(hadc);
223      if(ercd != E_OK){
224          syslog_1(LOG_NOTICE, "## adc_end_int ERROR(%d) ##", ercd);
225          slp_tsk();
226      }
```

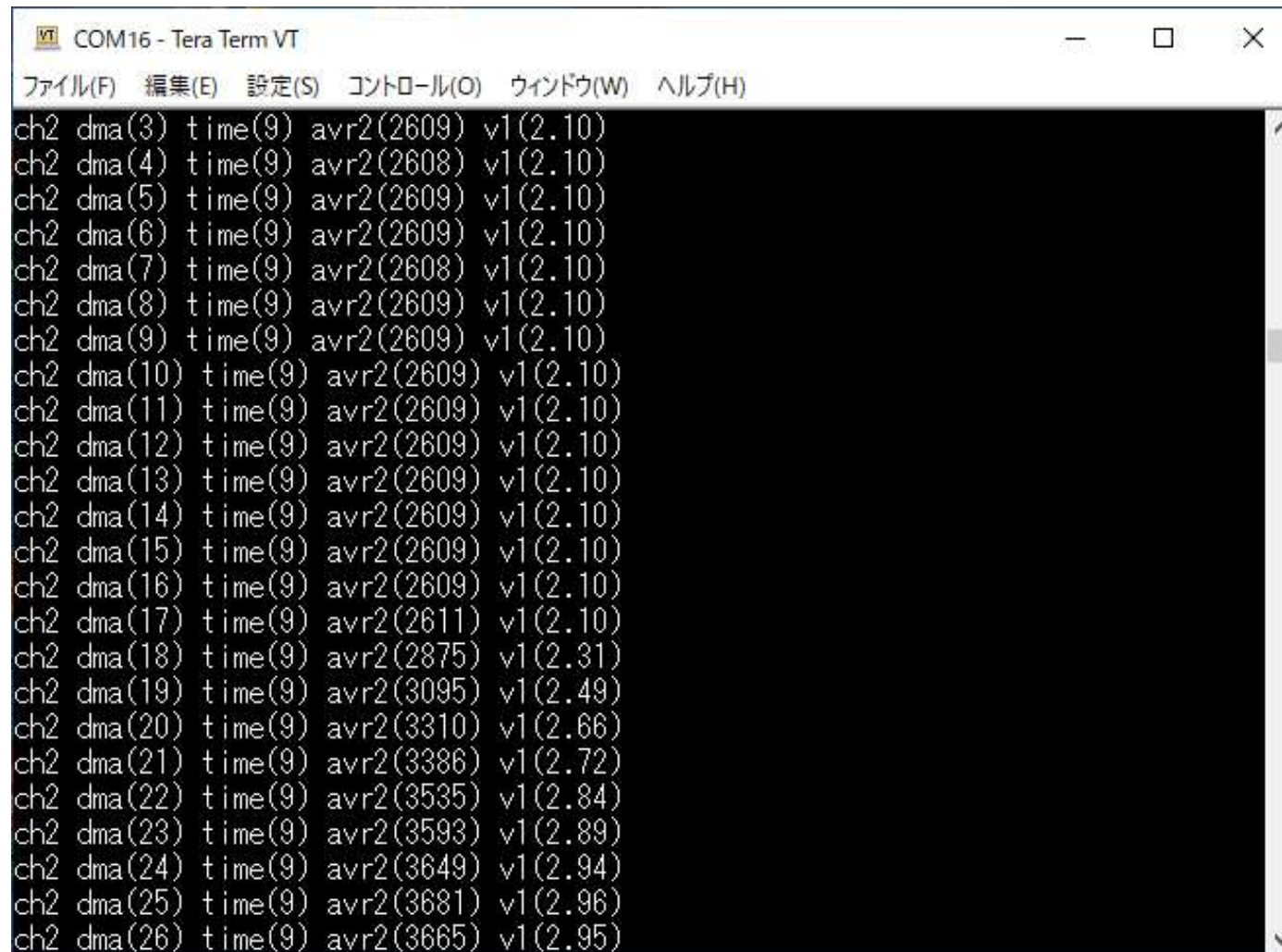
conver_countをゼロに

スタートをadc_start_intに変更

終了をadc_end_intに変更

演習: adc2を実行

- adc2を実行すると、adc1と同様に電圧表示ができる



The screenshot shows a Tera Term VT window titled "COM16 - Tera Term VT". The window contains a list of data for DMA channels 2 through 26. Each line represents a channel and contains five fields: channel number, dma number, time, avr2 value, and v1 value. The data is as follows:

Channel	DMA	Time	avr2	v1
ch2	dma(3)	time(9)	avr2(2609)	v1(2.10)
ch2	dma(4)	time(9)	avr2(2608)	v1(2.10)
ch2	dma(5)	time(9)	avr2(2609)	v1(2.10)
ch2	dma(6)	time(9)	avr2(2609)	v1(2.10)
ch2	dma(7)	time(9)	avr2(2608)	v1(2.10)
ch2	dma(8)	time(9)	avr2(2609)	v1(2.10)
ch2	dma(9)	time(9)	avr2(2609)	v1(2.10)
ch2	dma(10)	time(9)	avr2(2609)	v1(2.10)
ch2	dma(11)	time(9)	avr2(2609)	v1(2.10)
ch2	dma(12)	time(9)	avr2(2609)	v1(2.10)
ch2	dma(13)	time(9)	avr2(2609)	v1(2.10)
ch2	dma(14)	time(9)	avr2(2609)	v1(2.10)
ch2	dma(15)	time(9)	avr2(2609)	v1(2.10)
ch2	dma(16)	time(9)	avr2(2609)	v1(2.10)
ch2	dma(17)	time(9)	avr2(2611)	v1(2.10)
ch2	dma(18)	time(9)	avr2(2875)	v1(2.31)
ch2	dma(19)	time(9)	avr2(3095)	v1(2.49)
ch2	dma(20)	time(9)	avr2(3310)	v1(2.66)
ch2	dma(21)	time(9)	avr2(3386)	v1(2.72)
ch2	dma(22)	time(9)	avr2(3535)	v1(2.84)
ch2	dma(23)	time(9)	avr2(3593)	v1(2.89)
ch2	dma(24)	time(9)	avr2(3649)	v1(2.94)
ch2	dma(25)	time(9)	avr2(3681)	v1(2.96)
ch2	dma(26)	time(9)	avr2(3665)	v1(2.95)

可変抵抗を使ってADC入力

1. RP2040-ADC仕様
2. ADCドライバの説明
3. adc1プログラム
4. adc1のビルドと確認
5. 位置取り込みプログラム作成

演習: Joy Stickの位置を表示するプログラム

- Joy Stickの位置を表示するプログラムを作成する
- 学習内容
 - GPIO入力での位置判定
 - PUSH状態をADC値を使って変換する
- 手順
 - 可変抵抗をJOY STICKに戻す、追加したジャンパーピンを外す
 - adc2を修正してjoy2を作成する
 - SWピンは周期的に変換データを取り込む
 - 変換データからスティックの座標を判定する
 - ADCは12ビットなので、0～4095までの範囲(中間位置は2048)
 - 1920未満をLOW、以上をHIGHと判定
 - SWピン以外はGPIO入力で位置を取り込む
 - UP:D7 RIGHT:D6 LEFT:D5 DOWN:D4

演習：Joy Stickの位置を表示するプログラム

- Joy Stickの位置は、uint32_t型の整数で示す
 - 対応のビットがオンならばその位置にあるものとする
- プッシュ(SW)と他の位置は同時にオンとなる場合がある
- プッシュはアナログ値で判定(1970を閾値)
- JS定義とピン定義はadc1.hに定義済み

位置	JS定義	ピン	ピン定義
プッシュ(SW)	JS_PUSH	A3	–
上	JS_UP	D7	UP_PORT
下	JS_DOWN	D3	DOWN_PORT
左	JS_LEFT	D6	LEFT_PORT
右	JS_RIGHT	D5	RIGHT_PORT

演習：Joy Stickの位置を表示するプログラム

- adc2を修正してJoy Stickの位置を表示するプログラムを作成する
- 作成方法
 - adc2をmy_programにコピーして名前をjoy2に変更
 - joy2内の以下のプログラムをリネーム
 - adctest.c → joy.c
 - adctest.cfg → joy.cfg
 - adctest.h → joy.h
 - Makefileの166行目adctestをjoyに書き換え
 - joy.cfgの22行目adctest.hをjoy.hに書き換え
 - joy.cの58行目adctest.hをjoy.hに書き換え
- joyディレクトリ上でmakeを行って確認

演習 : Makefileの修正

- デジタルピンのGPIOアクセスを円滑に行うため、pinmodeドライバを追加する
- pinmodeドライバには、以下の機能がある
 - pinmodeドライバは、デジタルピンをCPUのピンに変換
 - デジタルピン番号でGPIOのREAD/WRITEを行う
- システムサービスに関する定義にpinmode.oを追加する

```
#
# システムサービスに関する定義
#
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 adc.o pinmode.o $(CXXRTS)
SYSSVC_CFLAGS := $(SYSSVC_CFLAGS)
SYSSVC_LIBS := $(SYSSVC_LIBS)
INCLUDES := $(INCLUDES) -I$(SRCDIR)/pdic/$(PDIC_DIR)
```

演習: joy.cの修正(インクルードファイル追加)

- joy.cのインクルードファイルにpinmode.hを追加する

```
#include <kernel.h>
#include <t_syslog.h>
#include <t_stdlib.h>
#include <string.h>
#include <sil.h>
#include "syssvc/serial.h"
#include "syssvc/syslog.h"
#include "kernel_cfg.h"
#include <target_syssvc.h>
#include "device.h"
#if BOARDNO != 0
#include "dtime.h"
#endif
#include "adc.h"
#include "pinmode.h"
#include "joy.h"

/*
 * サービスコールのエラーのログ出力
 */
```

演習: joy.cの修正 (GPIO初期化関数を追加)

- GPIO初期化関数pinModeを追加する
 - この関数lcdjoyshield/lcdshield.c等にあるので、コピーペーストでかなわない
- 関数の解説はSPIの章で行う

```
/*
 * GPIO設定関数
 */
static void
pinMode(uint8_t no, uint8_t mode)
{
    Arduino_PortControlBlock *ppin = getGpioTable(DIGITAL_PIN, no);
    GPIO_Init_t GPIO_Init_Data = {0};

    if(ppin == NULL)
        return;
    pinmux_setup(*ppin, IO_BANK0_GPIO_CTRL_FUNCSEL_VALUE_SIO);
    GPIO_Init_Data.mode = mode;
    if(mode == GPIO_MODE_OUTPUT)
        GPIO_Init_Data.pull = GPIO_PULLDOWN;
    else
        GPIO_Init_Data.pull = GPIO_NOPULL;
    gpio_setup(&GPIO_Init_Data, *ppin);
}
```

演習: joy.cの修正 (main_task)

- SW変数を追加

```
166 void
167 main_task(intptr_t exinf)
168 {
169     ADC_Init_t    aInit;
170     ADC_ChannelConf_t sConfig;
171     ADC_Handle_t *hadc;
172     ER_UINT        ercd;
173     SYSTIM tim1, tim2;
174     float sum;
175     int i, sw;
176     int vint, dint;
```

演習: joy.cの修正 (main_task)

- Joy Stickの位置に対応したGPIOを入力モードで初期設定する

```
196  /*
197   * UP PORT(D7-14)
198   */
199  pinMode(UP_PORT, GPIO_MODE_INPUT);
200
201  /*
202   * RIGHT PORT(D6-13)
203   */
204  pinMode(RIGHT_PORT, GPIO_MODE_INPUT);
205
206  /*
207   * LEFT PORT(D5-10)
208   */
209  pinMode(LEFT_PORT, GPIO_MODE_INPUT);
210
211  /*
212   * DOWN PORT(D3-12)
213   */
214  pinMode(DOWN_PORT, GPIO_MODE_INPUT);
```

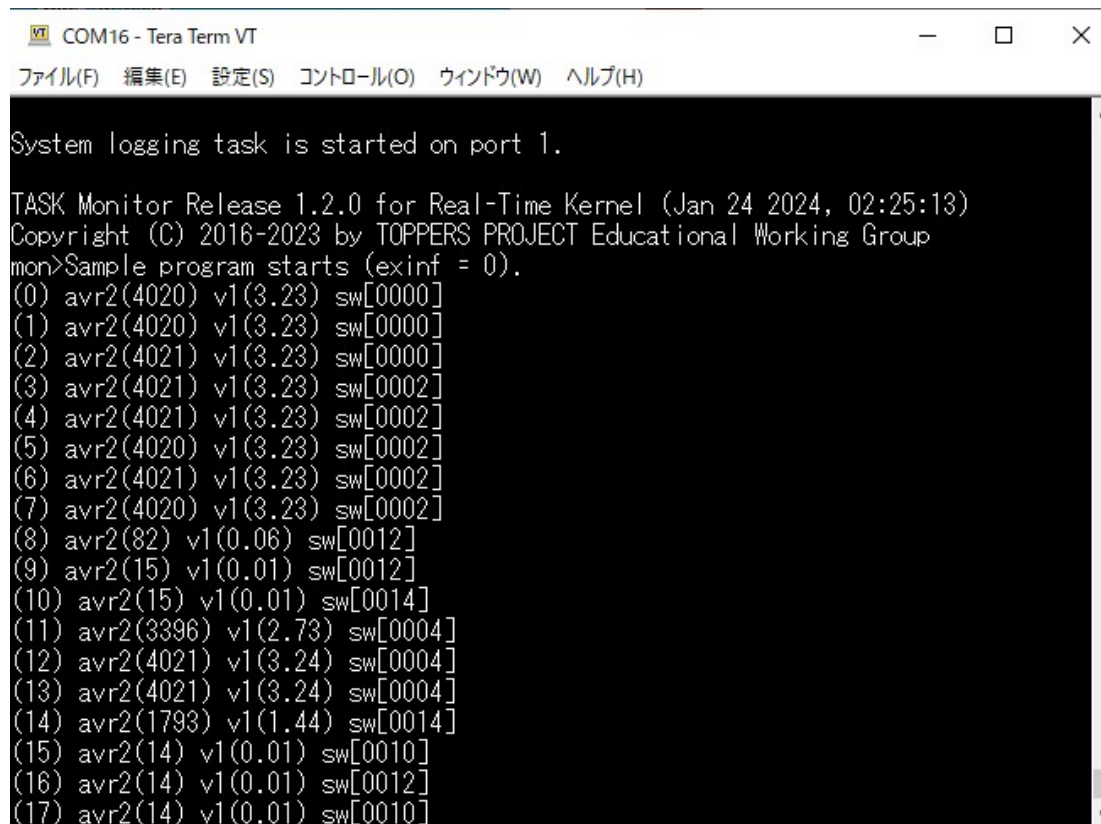
演習: joy.cの修正(位置データを取り込む)

- digitalRead関数を使って、位置情報を取り込む
 - 0:オン 1:オフとなっているので注意
- SW(PUSH)は、ADC値を閾値で判定する

```
263     ercd = adc_end_int(hadc);
264     if(ercd != E_OK){
265         syslog_1(LOG_NOTICE, "## adc_end_int ERROR(%d) ##", ercd);
266         slp_tsk();
267     }
268     sum = make_sum(&vint, &dint);
269     sw = digitalRead(RIGHT_PORT);
270     sw |= digitalRead(LEFT_PORT) << 1;
271     sw |= digitalRead(DOWN_PORT) << 2;
272     sw |= digitalRead(UP_PORT) << 3;
273     sw ^= 0xf;
274     if(sum < 1920)
275         sw |= JS_PUSH;
276     syslog_5(LOG_NOTICE, "(%d) avr2(%d) v1(%d.%02d) sw[%04x]", i, (int)sum, vint, dint, sw);
277     dly_tsk(100);
278 }
```

演習: joy2を実行

- joy2を実行してJoy Stickの位置により、swの値が正しく変化することを確認
- ADCの入力の通り、GPIOの入力は閾値未満を0、閾値以上を1としている



```
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)

System logging task is started on port 1.

TASK Monitor Release 1.2.0 for Real-Time Kernel (Jan 24 2024, 02:25:13)
Copyright (C) 2016-2023 by TOPPERS PROJECT Educational Working Group
mon>Sample program starts (exinf = 0).
(0) avr2(4020) v1(3.23) sw[0000]
(1) avr2(4020) v1(3.23) sw[0000]
(2) avr2(4021) v1(3.23) sw[0000]
(3) avr2(4021) v1(3.23) sw[0002]
(4) avr2(4021) v1(3.23) sw[0002]
(5) avr2(4020) v1(3.23) sw[0002]
(6) avr2(4021) v1(3.23) sw[0002]
(7) avr2(4020) v1(3.23) sw[0002]
(8) avr2(82) v1(0.06) sw[0012]
(9) avr2(15) v1(0.01) sw[0012]
(10) avr2(15) v1(0.01) sw[0014]
(11) avr2(3396) v1(2.73) sw[0004]
(12) avr2(4021) v1(3.24) sw[0004]
(13) avr2(4021) v1(3.24) sw[0004]
(14) avr2(1793) v1(1.44) sw[0014]
(15) avr2(14) v1(0.01) sw[0010]
(16) avr2(14) v1(0.01) sw[0012]
(17) avr2(14) v1(0.01) sw[0010]
```

SPI仕様、SPIデバイスドライバ

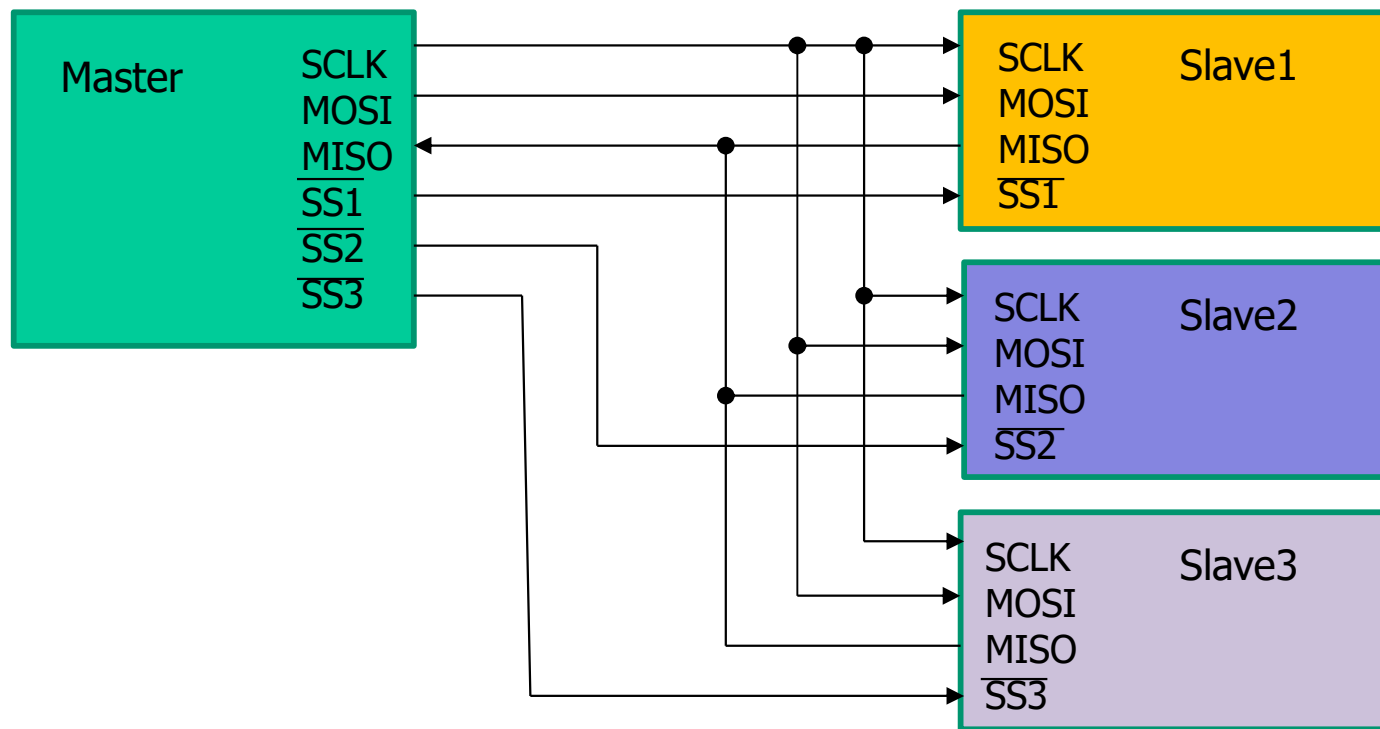
1. [SPI仕様](#)
2. SPIデバイスドライバ
3. LOOP-BACKプログラム

SPI仕様

- SPI(シリアル・ペリフェラル・インターフェース)はモトローラ社(現NXP)が提唱した周辺デバイス用のシリアル通信規格である
 - TI社、MicroChip社互換の拡張規格を提唱している
- SPIは4つの信号により構成される
 - SCLK(クロック): Serial Clock
 - MISO(マスター入力/スレーブ出力): Master In Slave Out
 - MOSI(マスター出力/スレーブ入力): Master Out Slave In
 - SS(スレーブセレクト): Slave Select
- 送信専用の場合、MISOは不要
- 受信専用の場合、MOSIは不要
- 一対一でも、信号を保障するためにSSは制御した方が良い

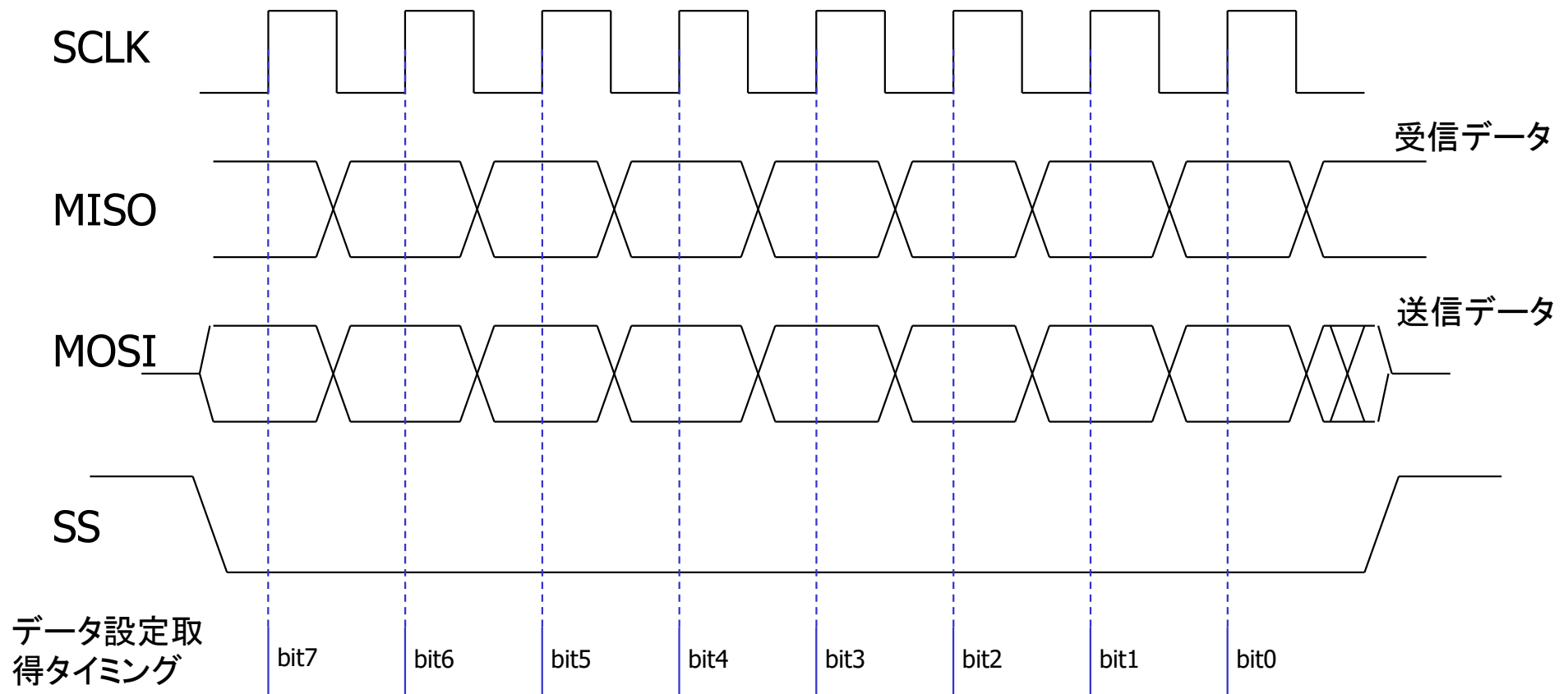
一般的な接続図

- 下図はMasterに対して3つのスレーブと接続した場合の接続図です
- SSは通信が有効なスレーブに対してLow Activeとする



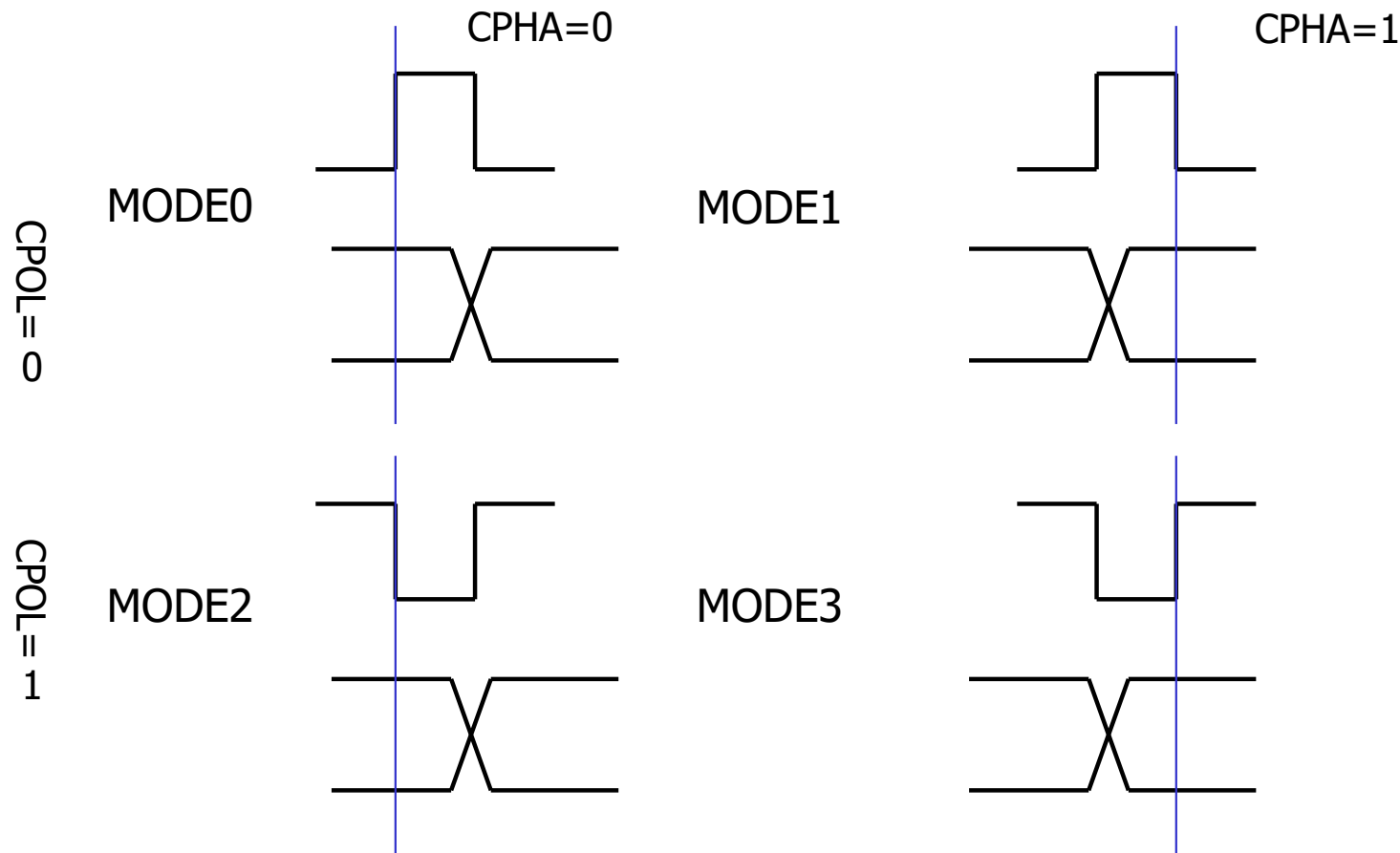
信号とデータ

- データは1クロックに対して1ビットを確定できる
- データ長は2～16ビットを指定できる(一般的には8ビット)
- MODE0(CPOL=0,CPHA=0)の8ビットタイミング



転送モード

- クロック位相とクロック極性によりで4つのモードがある
- クロック極性: CPOL(Clock Polarity)、クロック位相: CPHA(Clock Phase)



SPIの特徴

- 転送速度は1MHzから数十MHzまでで、I2CやUARTと比べて高速
- 全二重通信が可能
- 複雑な周辺デバイスの場合、SPIの上位に通信プロトコルを定義する必要がある

SPI仕様、SPIデバイスドライバ

1. SPI仕様
2. SPIデバイスドライバ
3. LOOP-BACKプログラム

SPI1を動作させるためのリソース

- SPI1を制御するために、関連ハードウェアの設定が必要
- TOPPERS BASE PLATFORM(RP) SPIドライバでは、SPI割込みとデータ転送用に二つのDMAチャネルを使用する
- 割込み設定とDMA設定はSPIドライバ内で行う、静的APIで割込み設定が必要
- SPIドライバは、`pdic/rp2040/spi.c`に記載
- DMAドライバは、`pdic/rp2040/device.c`に記載
 - これらをリンクする必要がある

使用割込み	デファイン名	番号1	番号2	管理種別
SPI1割込み	IRQ_VECTOR_SPI0	34	47	割込みサービス・ルーチン
DMA割込み	IRQ_VECTOR_DMA0	27	26	割込みハンドラ

SPIドライバの構成

- SPIはマスターモードのみで、送信、受信、送受信用に3つの関数をもつ
- 転送待ちはコンパイラ変数SPI_WAIT_TIMEに1以上の数を設定した場合は不要となる
- この場合、(SPI_WAIT_TIME*length)msが最大待ち時間となる

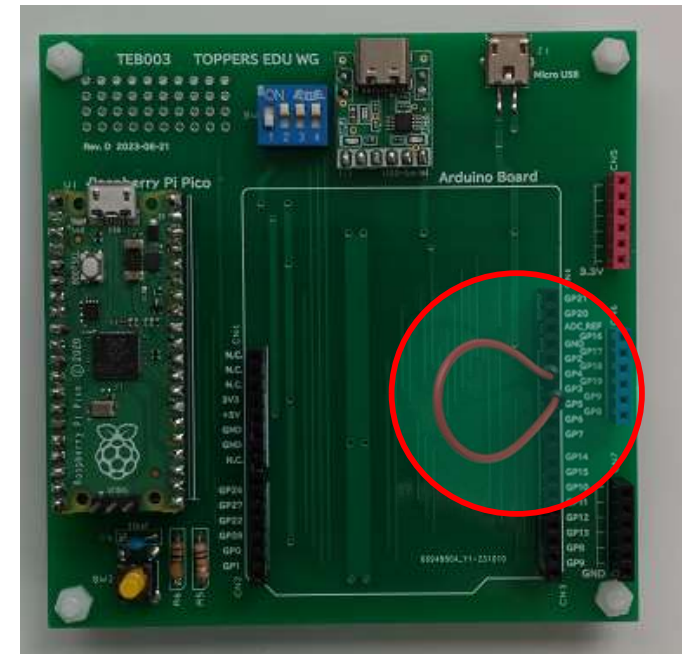
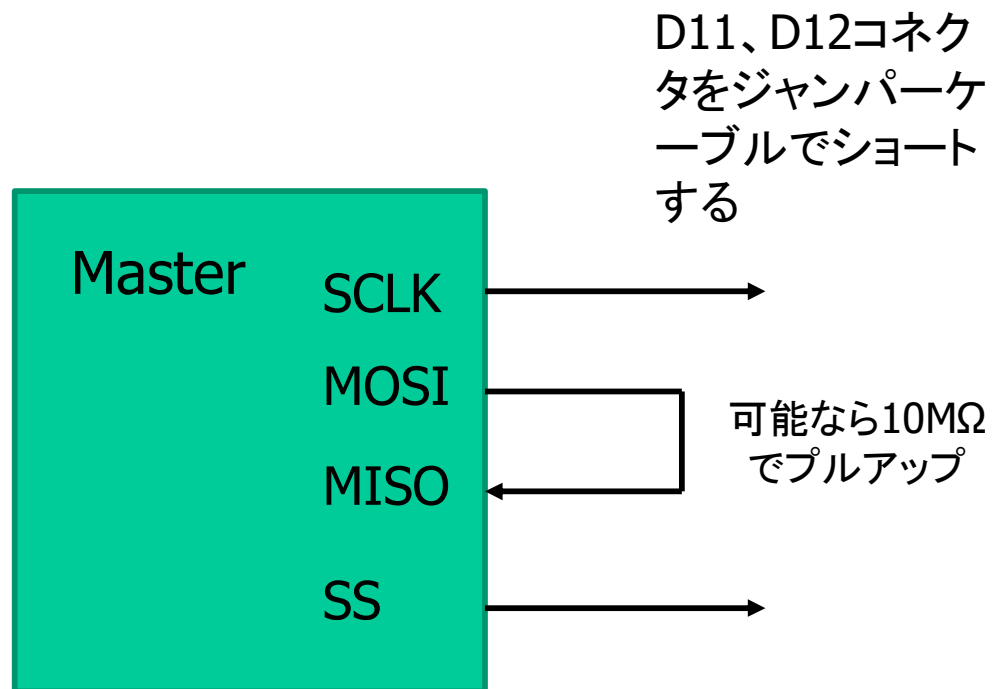
書式	機能	返り値
void spi_isr(intptr_t exinf);	SPI割込みサービスルーチン	なし
SPI_Handle_t *spi_init(ID portid SPI_Init_t *spii);	SPI初期化関数	ハンドラへのポインタ
ER spi_deinit(SPI_Handle_t *hsapi);	SPIドライバ終了	E_OK
ER spi_transmit(SPI_Handle_t *hsapi, uint8_t *pdata, uint16_t length);	SPI送信	E_OK
ER spi_receive(SPI_Handle_t *hsapi, uint8_t *pdata, uint16_t length);	SPI受信	E_OK
ER spi_tarnsrev(SPI_Handle_t *hsapi, uint8_t *ptxdata, uint8_t *prxdata, uint16_t length);	SPI送受信	E_OK
ER spi_wait(SPI_Handle_t *hsapi, uint32_t timeout);	SPI転送終了待ち	E_OK

SPI仕様、SPIデバイスドライバ

1. SPI仕様
2. SPIデバイスドライバ
3. [LOOP-BACKプログラム](#)

LOOP-BACKハードウェア

- SPIのデバイスドライバテスト用に、以下のようにMOSIとMISOを直接接続すれば、送信データをそのまま、受信することが可能となる
- この場合、SCLK、SSは考慮する必要はない



spi1解説

- spi1は以下の処理を連続して行う
 - SPIで文字列を転送し送受信データを表示
 - SPIで文字列を送信する
- プログラムの構成
 - program/spi1のディレクトリ中のファイル構成
 - spi1test.c
 - spi1test.h
 - spi1test.cfg
 - Makefile
 - MAIN_TASK(main_task)のみで実行
- 手順
 - program/spi1をmy_programにコピーしビルド、実行

spi1.h: インクルードファイル

- SPIと送受信DMAの割込みの設定定義を行う

SPI割込み定義

```
#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 */
```

DMA割込みの定義

```
#define INHNO_DMASPI IRQ_VECTOR_DMA0    /* 割込みハンドラ番号 */
#define INTNO_DMASPI IRQ_VECTOR_DMA0    /* 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 */
```

```
/*
```

```
 * 関数のプロトタイプ宣言
```

```
*/
```

```
#ifndef TOPPERS_MACRO_ONLY
```

```
extern void main_task(intptr_t exinf);
```

```
extern void device_info_init(intptr_t exinf);
```

```
extern void sw_handle(void);
```

```
#endif /* TOPPERS_MACRO_ONLY */
```

spi1.cfg:コンフィギュレーションファイル

- spi1プログラムで使用するOSリソースを静的APIで定義
通信用セマフォ、排他制御セマフォ、タスク、割込み定義

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

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

```
CRE_SEM(SPI1DMA_SEM, { TA_TPRI, 0, 1 });  
CRE_SEM(SPI1LOCK_SEM, { TA_TPRI, 1, 1 });
```

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

```
ATT_ISR({TA_NULL, SPI1_PORTID, INTNO_SPI, spi_isr, 1 });  
CFG_INT(INTNO_SPI, { TA_ENAINT | INTATR_SPI, INTPRI_SPI });  
ATT_ISR({TA_NULL, DMA_INTNO, INTNO_DMAADC, dma_isr, 1 });  
CFG_INT(INTNO_DMASPI, { TA_ENAINT | INTATR_DMASPI, INTPRI_DMASPI });
```

転送終了通知用セマフォを
SPI1DMA_SEMで設定
排他制御セマフォを
SPI1LOCK_SEMで設定

プログラム実行用の
タスクを定義

割込みサービスルーチンの
定義静的APIにて、SPIと
DMAの割込み設定

spi1.c:ソースファイル1

- 送受信のデータ領域の設定を行う
 - aTxBuffer: 送信バッファ: 文字列を初期値で設定
 - aRxBuffer: 受信バッファ

```
#define SVC_PERROR(expr)    svc_perror(__FILE__, __LINE__, #expr, (expr))

#define COUNTOF(__BUFFER__) (sizeof(__BUFFER__) / sizeof(*(__BUFFER__)))

#define BUFFERSIZE          (COUNTOF(aTxBuffer) - 1)           /* 送受信バッファサイズ */

uint8_t aTxBuffer[] = "SPI123456789012345678901";              /* 送信バッファ */
uint8_t aRxBuffer[BUFFERSIZE];                                  /* 受信バッファ */
```

送信バッファ

受信バッファ

spi1.c:ソースファイル2

- SPI1の初期化をspi_init関数で行う

```
syslog_0(LOG_NOTICE, "SPI SEND/RECV START");  
/*  
 * SPI 双方向転送 1MHz  
 */  
spi_initd.Baudrate      = 1000*1000;  
spi_initd.Direction     = SPI_DIRECTION_2LINES;  
spi_initd.CLKPhase      = SPI_PHASE_1EDGE;  
spi_initd.CLKPolarity   = SPI_POLARITY_HIGH;  
spi_initd.DataSize      = SPI_DATASIZE_8BIT;  
spi_initd.FrameFormat   = SPI_MOTOROLA_MODE;  
spi_initd.DMARxChannel   = 0;  
spi_initd.DMATxChannel   = 1;  
spi_initd.Mode          = SPI_MODE_MASTER;  
spi_initd.sem            = SPI1DMA_SEM;  
spi_initd.semlock       = SPI1LOCK_SEM;  
  
if((hspi = spi_init(SPI1_PORTID, &spi_initd)) == NULL){  
    /* SPI初期化エラー */  
    syslog_0(LOG_NOTICE, "spi_init 初期化エラー !");  
    slp_tsk();  
}
```

転送周波数を設定

SPIモード設定

転送データ長を設定

DMAチャンネルを設定

セマフォを登録

SPI初期化

spi1.c:ソースファイル3

- 文字列を送受信後、送信バッファと受信バッファを表示
- DMAレジスタを表示後、SPI停止

```
if((ercd = spi_transrecv(hspi, (uint8_t*)aTxBuffer, (uint8_t *)aRxBuffer, BUFFERSIZE)) != E_OK){
    /* Transfer error in transmission process */
    syslog_1(LOG_NOTICE, "spi_transrecv 転送エラー(%d) !", ercd);
}
#if SPI_WAIT_TIME == 0
    if((ercd = spi_wait(hspi, 100)) != E_OK){
        syslog_1(LOG_NOTICE, "spi_transrecv 転送待ちエラー(%d) !", ercd);
    }
#endif
dly_tsk(100);
for(i = 0 ; i < BUFFERSIZE ; i++){
    syslog_3(LOG_NOTICE, "## (%d) T[%02x] R[%02x] ##", i, aTxBuffer[i], aRxBuffer[i]);
}
.....
hdmatx = hspi->hdmatx;
syslog_2(LOG_NOTICE, "## INST RX[%08x] TX[%08x] ##", hdmarx->cbase, hdmatx->cbase);
syslog_2(LOG_NOTICE, "## CR  RX[%08x] TX[%08x] ##",
    sil_rew_mem((uint32_t *) (hdmarx->cbase+TOFF_DMA_CH_CTRL_TRIG)),
    sil_rew_mem((uint32_t *) (hdmatx->cbase+TOFF_DMA_CH_CTRL_TRIG)));
.....
if(spi_deinit(hspi) != E_OK){
    syslog_0(LOG_NOTICE, "spi_deinit 終了エラー(%d) !");
}
syslog_0(LOG_NOTICE, "SPI SEND/RECV END");
```

spi1.c:ソースファイル4

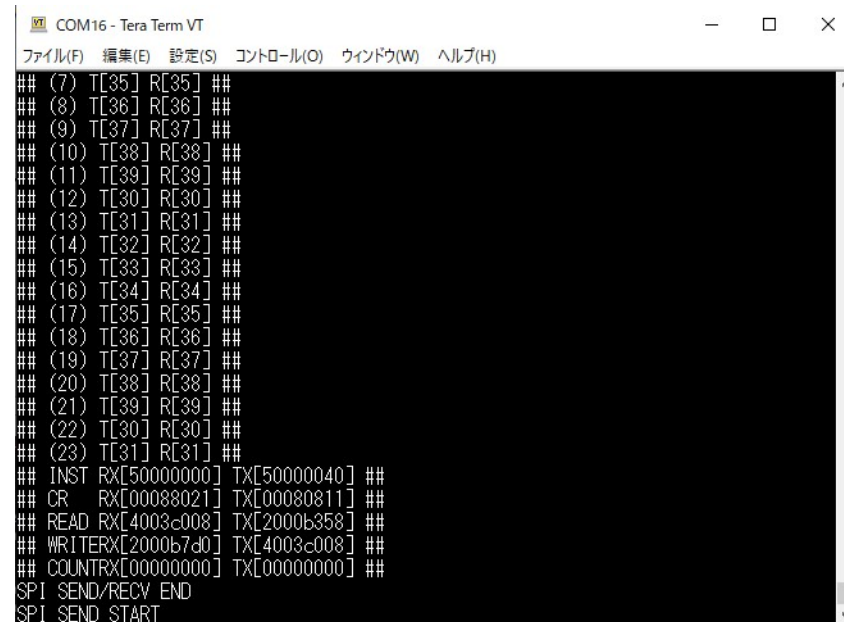
- 設定を片方向転送に変更して、送信を行う

```
/*
 * 1LINEモードで送信
 */
if((ercd = spi_transmit(hspi, (uint8_t*)aTxBuffer, BUFFERSIZE)) != E_OK){
    /* Transfer error in transmission process */
    syslog_1(LOG_NOTICE, "spi_transmit 転送エラー(%d) !", ercd);
}
#if SPI_WAIT_TIME == 0
    if((ercd = spi_wait(hspi, 100)) != E_OK){
        syslog_1(LOG_NOTICE, "spi_transmit 転送待ちエラー(%d) !", ercd);
    }
#endif
dly_tsk(100);
syslog_0(LOG_NOTICE, "SPI SEND END");
spi_deinit(hspi);

syslog(LOG_NOTICE, "Sample program ends.");
slp_tsk();
```

演習: spi1を実行して、送受信を確認

- ジャンパーケーブルでMOSI(D11)/MISO(D12)を接続
- spi1をビルド、ダウンロード、実行して受信バッファに送信データが転送されていることを確認する
- 学習内容
 - SPIの転送手順の設定確認
 - ジャンパーケーブルを外すと正しく転送されない



```
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)
## (7) T[35] R[35] ##
## (8) T[36] R[36] ##
## (9) T[37] R[37] ##
## (10) T[38] R[38] ##
## (11) T[39] R[39] ##
## (12) T[30] R[30] ##
## (13) T[31] R[31] ##
## (14) T[32] R[32] ##
## (15) T[33] R[33] ##
## (16) T[34] R[34] ##
## (17) T[35] R[35] ##
## (18) T[36] R[36] ##
## (19) T[37] R[37] ##
## (20) T[38] R[38] ##
## (21) T[39] R[39] ##
## (22) T[30] R[30] ##
## (23) T[31] R[31] ##
## INST RX[50000000] TX[50000040] ##
## CR RX[00088021] TX[00080811] ##
## READ RX[4003c008] TX[2000b358] ##
## WRITERX[2000b7d0] TX[4003c008] ##
## COUNTRX[00000000] TX[00000000] ##
SPI SEND/RCV END
SPI SEND START
```

LCDの初期化とピクセル設定

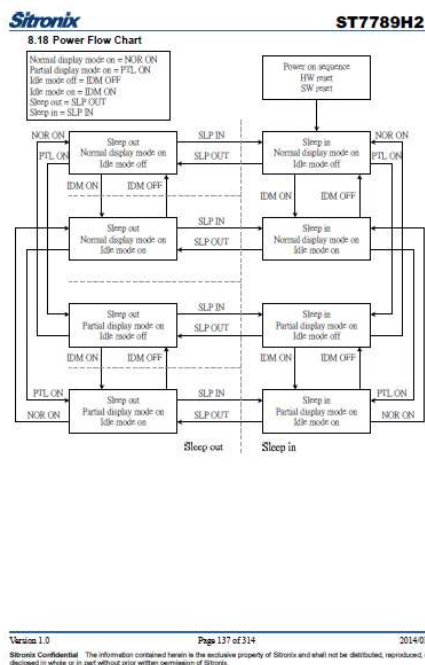
1. [Adafruit ST7789グラフィックI/F](#)
2. グラフィックLCD通信ドライバ
3. LCDの初期化
4. ピクセル設定

ST7789によるハードウェア制御

- Sitronix社ST7789はIMピンの設定により、MPUとの間に種々のデータ転送が可能である
- 本シールドではSPI(送信準拠)の4-Pin Serial Interface IIを使用している
- 信号ピンは以下の5ピンを使用している
 - RESX: ハードリセット(RST:D9)
 - CSX: スレーブセレクト信号(CS:D10またはD4)
 - WRX: コマンド/データセレクト(RS:D8)
 - SCL: クロック信号(SCLK:D13)
 - SDA: データ信号(MOSI:D11)
- WRXは通信データがコマンドか、データかの判別に使用

ST7789ハードウェア設計

- ST7789のデータブック「Sitornix ST7789H2 Datasheet」を見れば基本制御を理解することができる
- シールドがST7789を、どのようにハードウェア実装しているかは、正しい回路図がなければ理解できない
 - 細かいコマンド設定はハードウェア設計者が指定する



Sitronix

ST7789H2

9 COMMAND

9.1 System Function Command Table 1

Instruction	GDA	MVA	DON	DITA	DT	DE	DS	DA	D3	D2	D1	D0	Hex	Function
NOP	0	1	1	-	0	0	0	0	0	0	0	0	(0x)	No operation
SWRESET	0	1	1	-	0	0	0	0	0	0	1	0	(01)	Software reset
	1	1	1	-	0	0	0	0	0	0	1	0	(0A)	Read display ID
	1	1	1	-	-	-	-	-	-	-	-	-	-	Dummy read
R0D0	1	1	1	-	D17	D16	D15	D14	D13	D12	D11	D10	(01 read)	D1 read
	1	1	1	-	D27	D26	D25	D24	D23	D22	D21	D20	(02 read)	D2 read
	1	1	1	-	D37	D36	D35	D34	D33	D32	D31	D30	(03 read)	D3 read
	0	1	1	-	0	0	0	0	1	0	0	1	(0A)	Read display status
	1	1	1	-	-	-	-	-	-	-	-	-	-	Dummy read
R0E1	1	1	1	-	BETON	MY	MX	ML	RGB	M1	STPA	-	-	-
	1	1	1	-	TB3	FPFZ	FPFI	FPFO	OMCA	PFLN	SLOUT	SLOCAT	-	-
	1	1	1	-	ST18	ST14	MON	ST12	ST11	SDON	TECON	OC12	-	-
	1	1	1	-	GD31	GD30	TEMP	ST4	ST3	ST2	ST1	ST0	-	-
	0	1	1	-	0	0	0	0	0	1	0	1	(0A)	Read display power
	1	1	1	-	BETON	EMON	PFLN	SLOUT	HOSION	SDON	S	0	-	-
	1	1	1	-	-	-	-	-	-	-	-	-	-	Dummy read
RDD MACOT1	0	1	1	-	0	0	0	0	0	1	0	1	(0B)	Read display gamma
	1	1	1	-	-	-	-	-	-	-	-	-	-	Dummy read
RDD COLMOD1	0	1	1	-	0	0	0	0	0	1	1	0	(0C)	Read display color
	1	1	1	-	-	-	-	-	-	-	-	-	-	Dummy read
R0D0M	1	1	1	-	0	DE	DB	DE	0	02	D1	DD	-	-
	0	1	1	-	0	0	0	0	0	1	0	1	(0D)	Read display image
	1	1	1	-	-	-	-	-	-	-	-	-	-	Dummy read
R0D0M	1	1	1	-	VSDON	0	MYON	0	0	OC2	OC1	OD0	-	-
	0	1	1	-	0	0	0	0	0	1	1	0	(0E)	Read display signal
	1	1	1	-	-	-	-	-	-	-	-	-	-	Dummy read

Version 1.0

Page 154 of 314

2014/01

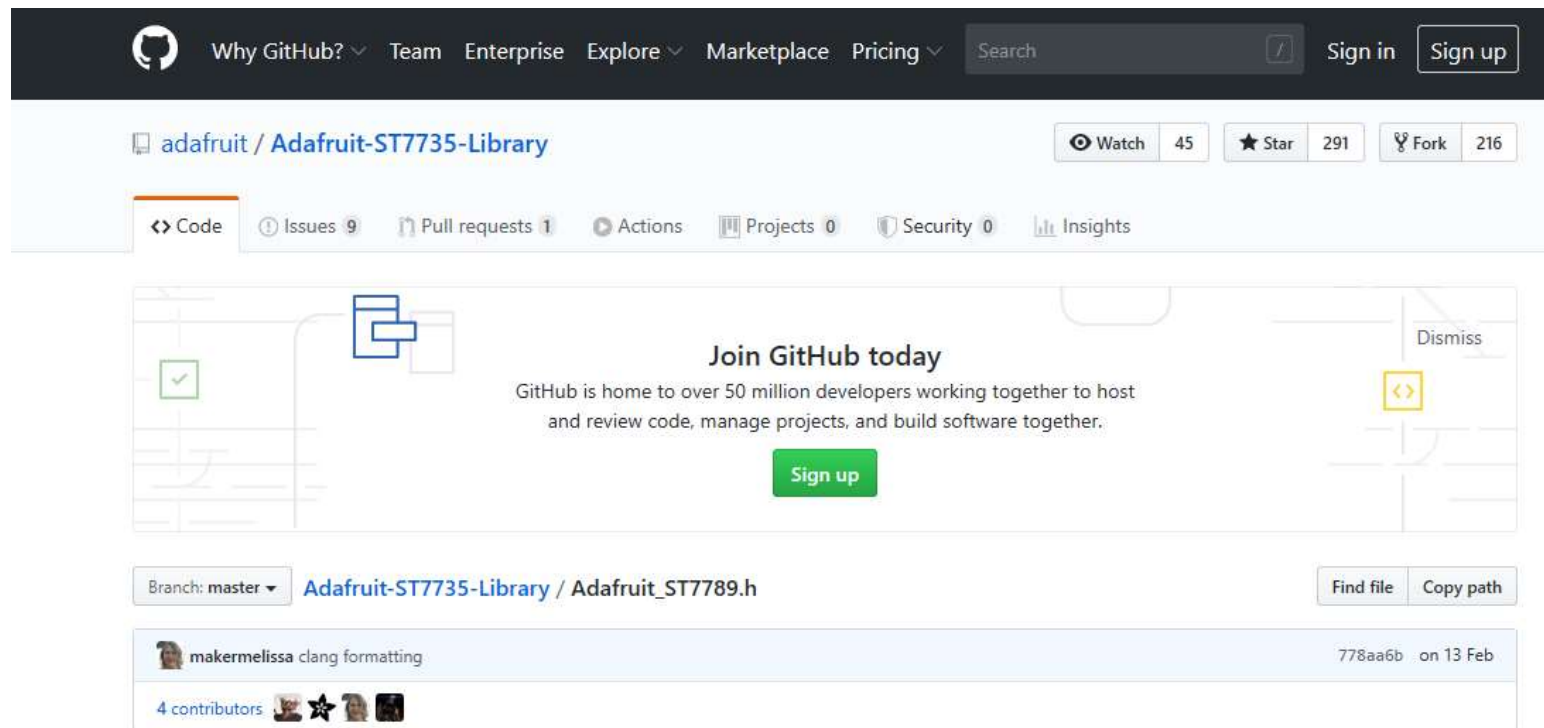
Sitronix Confidential The information contained herein is the exclusive property of Sitronix and shall not be distributed, reproduced,

電源ONシーケンス図

コマンド一覧表(抜粋)

LCD制御上位ドライバの作成

- Adafruit社のGitHubからST7735/ST7789用のライブラリを使用してドライバを作成

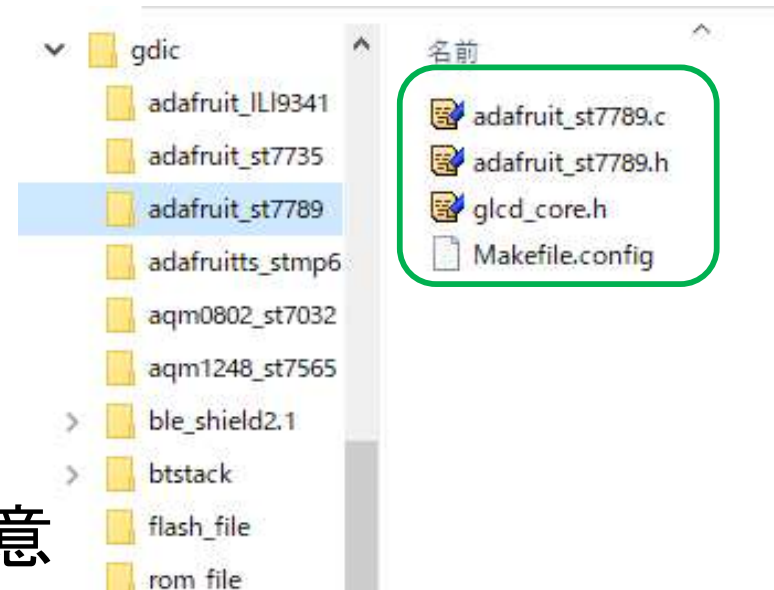


LCDの初期化とピクセル設定

1. Adafruit ST7789グラフィックI/F
2. [グラフィックLCD通信ドライバ](#)
3. lcd_initプログラム
4. ピクセル設定

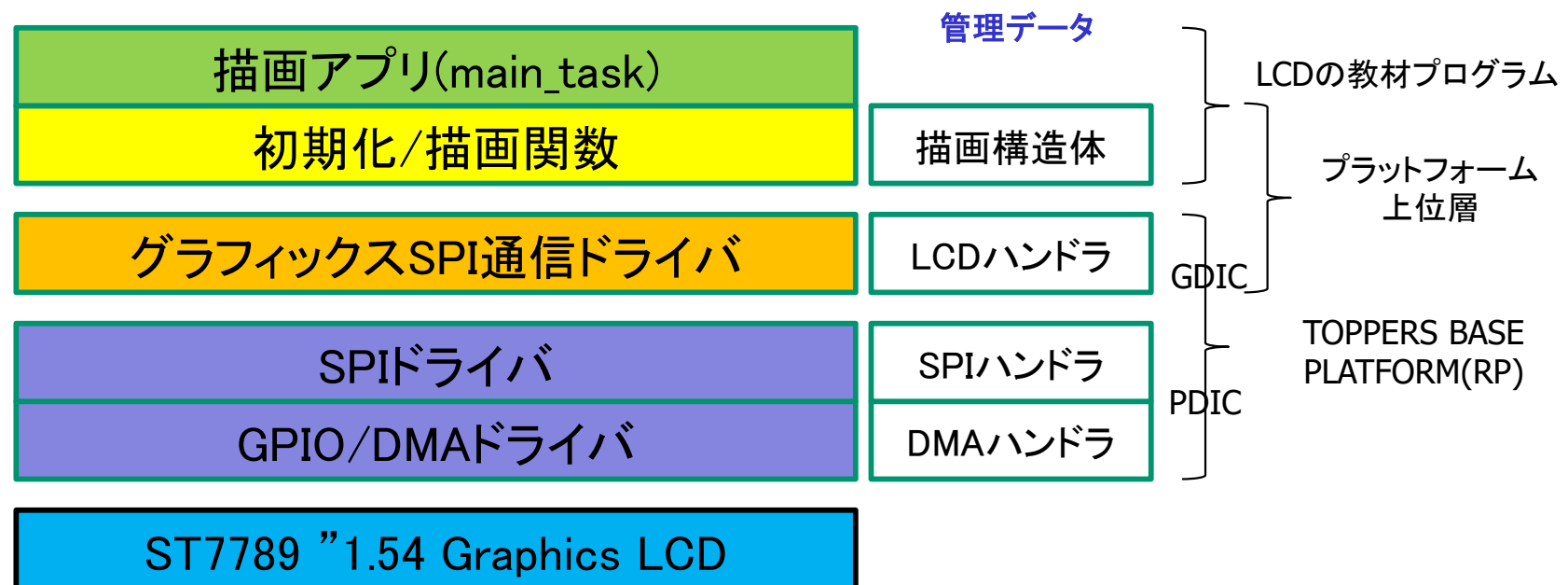
LCDドライバの設定手順

- 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初期設定
 - 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
- 参考プログラムをlcd_initに用意
 - 以降の説明はlcd_init/lcd_init.cを用いて説明



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ドライバーはCPUの差異には依存しない
 - gdicドライバーはpdicドライバーのAPIに依存する
- アプリからは、SPIドライバは見え、初期化/描画関数(adafruit_st7789ドライバ)を用いてLCD表示を行う



Arduino準拠 GPIO初期化関数

- Arduino IDE仕様に準拠したGPIO出力関数をローカルに用意する
- 以下の関数は、PLATFORMのdevice.c/pinmode.cで用意
 - gpio_setup/getGpiotable/pinClock

```
static void
pinMode(uint8_t no, uint8_t mode)
{
    Arduino_PortControlBlock *ppin = getGpioTable(DIGITAL_PIN, no);
    GPIO_Init_t GPIO_Init_Data = {0};

    if(ppin == NULL)
        return;
    pinmux_setup(*ppin, IO_BANK0_GPIO_CTRL_FUNCSEL_VALUE_SIO);
    GPIO_Init_Data.mode = mode;
    if(mode == GPIO_MODE_OUTPUT)
        GPIO_Init_Data.pull = GPIO_PULLDOWN;
    else
        GPIO_Init_Data.pull = GPIO_NOPULL;
    gpio_setup(&GPIO_Init_Data, *ppin);
}
```

Arduiboデジタルピン情報
からGPIOベースアドレスと
ピン番号を取り出す

Arduiboデジタルピン番号に対
応したGPIOクロックを起動

GPIOの初期設定
を行う

GPIO設定

- 使用するGPIOの初期設定を出力モードで行う
 - RESX(RESX):ハードリセット(RST:D9)
 - CSX(CSX):スレーブセレクト信号(CS:D10)
 - D/CX(WRX):コマンド/データセレクト(RS:D8)

```
/*  
 * 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);
```

SDカード用のCS

SPI設定

- LCDJOYシールドの仕様に合わせてSPIドライバの初期化を行う
- 初期化が成功すれば、hspiにハンドラのポインタが設定

```
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.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();  
}
```

Adafruit_st7789(GDIC)ドライバを取り込む

- Makefileでadafruit_st7789ディレクトリ上のMakefile.configをインクルードしドライバを取り込み

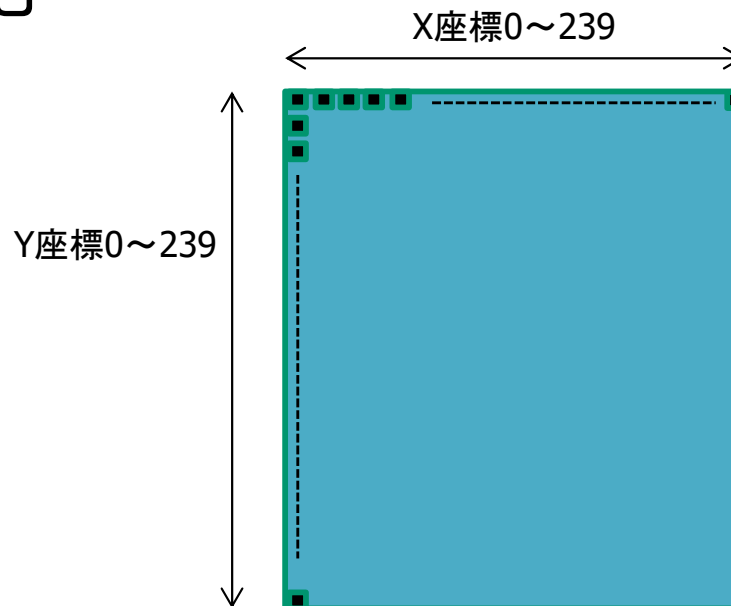
```
#  
# Makefile のプロセッサ依存 (adafruit st7789:LCD用)  
#  
# コンパイルオプション  
#  
INCLUDES := $(INCLUDES) -I$(SRCDIR)/gdic/adafruit_st7789  
#  
# ADAFRUIT ST7789 LCD DRIVER(GDIC)に関する定義  
#  
SYSSVC_DIR := $(SYSSVC_DIR):$(SRCDIR)/gdic/adafruit_st7789  
SYSSVC_COBJS := $(SYSSVC_COBJS) adafruit_st7789.o
```

Makefile.configの内容

```
#  
# ミドルウェア Makefile のインクルード  
#  
include $(SRCDIR)/monitor/Makefile.config  
  
#  
# GDIC Makefile のインクルード  
#  
include $(SRCDIR)/gdic/adafruit_st7789/Makefile.config
```

表示データ書込み

- LCDは幅240ピクセル、高さ240ピクセルのカラーグラフィック表示が可能
- 1ピクセルは2バイト(16ビット)
 - RGB R:5bit G:6bit B:5bitの割り当てとなる
 - ピクセルの色を指定するには対応座標に2バイトの色データを書き込む



SPI通信ドライバ

- adfruit_st7789.c中にSPIを使用し、LCDとの通信を行う上位ドライバを3つ用意した
- コマンドとデータの種別はドライバ内のRSの設定により選択する
- 3つのドライバの組み合わせで、SR7789のコマンド一覧に対応したコマンドを送信することができる

書式	機能	返回值
ER lcd_writecommand(LCD_Handle_t *hlcd, unsigned char c);	LCDに1バイトコマンドを送信する	E_OKまたはSPIエラー
ER lcd_writedata(LCD_Handle_t *hlcd, uint8_t c);	LCDに1バイトデータを送信する	E_OKまたはSPIエラー
ER lcd_writedata2(LCD_Hanale_t *hlcd, uint16_t c, int cnt);	LCDに2バイトデータをカウント分送信する	E_OKまたはSPIエラー

LCDコマンド送信ドライバ

- RS信号をLOWとしてSPIからデータ送信するとLCDはデータをコマンドと判定して受け取る

```
#define cs_set(sw) digitalWrite(hlcd->cs_pin, sw)
#define rs_set(sw) digitalWrite(hlcd->rs_pin, sw)

ER lcd_writecommand(LCD_Handle_t *hlcd, uint8_t c)
{
    ER ercd = E_OK;

    if(hlcd->spi_lock != 0){
        if((ercd = wai_sem(hlcd->spi_lock)) != E_OK)
            return ercd;
    }
    rs_set(PORT_LOW);    /* LCD-RS CMD */
    cs_set(PORT_LOW);    /* LCD-SS LOW */

    aTxBuffer[0] = c;
    ercd = spi_transmit(hlcd->hspi, (uint8_t*)aTxBuffer, 1);

    cs_set(PORT_HIGH);   /* LCD-SS HIGH */
    if(hlcd->spi_lock != 0)
        sig_sem(hlcd->spi_lock);
    return ercd;
}
```

RSをLOWとして、コマンド送信設定

データ c をコマンドとして送信

LCDデータ送信ドライバ

- RS信号をHIGHとしてSPIからデータ送信するとLCDはデータをデータと判定して受け取る

```
ER lcd_writedata(HLCD_Handle_t *hlcd, uint8_t c)
{
    ER ercd = E_OK;

    if(hlcd->spi_lock != 0){
        if((ercd = wai_sem(hlcd->spi_lock)) != E_OK)
            return ercd;
    }
    rs_set(PORT_HIGH); /* LCD-RS DATA */
    cs_set(PORT_LOW); /* LCD-SS LOW */
    sil_dly_nse(100);

    aTxBuffer[0] = c;
    ercd = spi_transmit(hlcd->hspi, (uint8_t*)aTxBuffer, 1);

    sil_dly_nse(100);
    cs_set(PORT_HIGH); /* LCD-SS HIGHT */
    if(hlcd->spi_lock != 0)
        sig_sem(hlcd->spi_lock);
    return ercd;
}
```

RSをHIGHとして、データ送信設定

データ c をデータとして送信

初期化テーブルを設定する関数

- LCD初期化テーブルに従ってST7789の設定を行う関数
 - DELAYは0x80(7bitがオンならば、待ち時間指定)

```
static void displayInit(LCD_Handler_t *hlcd, const uint8_t *addr)
{
    uint8_t numCommands, numArgs;
    uint16_t ms;

    numCommands = *addr++;          /* コマンド数を取り出し */
    while(numCommands--){
        SVC_ERROR(lcd_writecommand(hlcd->hspi, *addr++));
        numArgs = *addr++;          /* 引数の数を取り出し */
        ms = numArgs & DELAY; /* 待ち時間を取り出し */
        numArgs &= ~DELAY;          /* 待ち時間をマスク */
        while(numArgs--){          /* 引数を送信 */
            SVC_ERROR(lcd_writedata(hlcd->hspi, *addr++));
        }
        if(ms != 0){                /* 待ち時間指定があれば、待ちを行う */
            ms = *addr++;            /* 待ち時間を取り出す(ms) */
            if(ms == 255)             /* 最大値ならば 500ms */
                ms = 500;
            DELAY_TASK(ms);
        }
    }
}
```

コマンド送信

データ送信

設定待ち設定
サービスコールに依存しないように
マクロ化している

LCD初期化関数

- LCDの初期化は、初期化関数lcd_initを使用
 - widthは横のピクセル数、heightは縦のピクセル数
 - commandInitはハードウェア初期化関数

```
void lcd_init(LCD_Handle_t *hlcd, uint16_t width, uint16_t height)
{
    commandInit(hlcd, width, height);

    displayInit(hlcd, Gcmd);
    lcd_setRotation(hlcd, hlcd->rotation);
}
```

LCDハードリセット

初期化テーブルを用いた初期化

表示の回転を設定

LCDハンドラ

- LCDのハードウェア設定情報を効率良く管理する構造体を定義する
- SPIハンドラとLCDの物理的な管理データを持つ

```
/*
 * LCDハンドラ構造体定義
 */
typedef struct
{
    SPI_Handle_t    *hspi;        /* spiハンドラ */
    uint16_t        _width;       /* 幅ピクセル数 */
    uint16_t        _height;      /* 高さピクセル数 */
    uint16_t        colstart;
    uint16_t        rowstart;
    uint16_t        lcd_width;
    uint16_t        lcd_height;
    uint16_t        lcd_colstart;
    uint16_t        lcd_rowstart;
    uint8_t         rotation;
    uint8_t         dummy[3];
    ID              spi_lock;
    uint8_t         cs_pin;
    uint8_t         rs_pin;
    uint8_t         rst_pin;
    uint8_t         cs2_pin;
}LCD_Handler_t;
```

描画情報の管理構造体

- LCD描画に必要な(最低限度)データを管理する構造体を定義する
 - hlcd LCDハンドラへのポインタ
 - TextColor foreground色
 - BackColor background色
 - pFont フォント構造体へのポインタ: 文字表示で使用

```
/*  
 * 描画構造体  
 */  
typedef struct  
{  
    LCD_Handler_t        *hlcd;  
    uint16_t              TextColor;  
    uint16_t              BackColor;  
    void                  *pFont;  
}LCD_DrawProp_t;
```

LCDの初期化とピクセル設定

1. Adafruit ST7789グラフィックI/F
2. グラフィックLCD通信ドライバ
3. [lcd_initプログラム](#)
4. ピクセル設定

lcd_init解説

- lcd_initはLCDの初期化とピクセル描画を行う
 - LCDの初期化
 - 黒でバックグラウンドフィル
 - 緑で中央にピクセル描画
 - 白でバックグラウンドフィル
- 構成ファイル
 - lcd_init.h: ヘッダーファイル
 - lcd_init.c: C言語ソース
 - lcd_init.cfg: コンフィギュレーションファイル
 - Makefile: メイクファイル

lcd_init.h: インクルードファイル

- 使用するSPIデバイスの割込み設定と受信と送信のDMAの割込み設定を行う

```
#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 */
```

SPI1割込み設定

```
#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 */
```

DMA割込み設定

lcd_init.h: インクルードファイル

- LCDのGPIOボードとして、4つのポートとデジタルピン番号を定義
 - CS_PORT D10またはD4 (使用しない場合はHIGH)
 - RST_PORT D9
 - RS_PORT D8
 - CS2_PORT D4またはD10 (使用しない場合はHIGH)

```
#define PORT_HIGH 1
#define PORT_LOW 0

#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
```

デジタルピンの定義
シールドの差異を条件コンパイル

lcd_init.cfg: コンフィギュレーションファイル

- lcd_initプログラムで使用するOSリソースを静的APIで定義
 - セマフォ、タスク、割込み定義
- SPIとDMAの2つの割込みは最適な設定をユーザーが定義する

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

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

プログラム実行用の
タスクを定義

```
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 });
```

転送終了通知用セマフォと
排他制御用セマフォを定義

```
ATT_ISR({TA_NULL, SPI1_PORTID, INTNO_SPI, spi_isr, 1 });  
CFG_INT(INTNO_SPI, { TA_ENAINT | INTATR_SPI, INTPRI_SPI });  
ATT_ISR({TA_NULL, DMA_INTNO, INTNO_DMAADC, dma_isr, 1 });  
CFG_INT(INTNO_DMASPI, { TA_ENAINT | INTATR_DMASPI, INTPRI_DMASPI });
```

SPIとDMAの割込み
設定を定義

lcd_init.c:ソースファイル

- メインタスクにデジタルピンとSPI1の初期化を行う

```
void main_task(intptr_t exinf)
{
    SPI_Init_t spi_initd;
    SPI_Handle_t *hspi;
    LCD_Handler_t *hlcd;
    :
    syslog_0(LOG_NOTICE, "LCD TEST START");
    pinMode(CS_PORT, GPIO_MODE_OUTPUT);
    pinMode(RS_PORT, GPIO_MODE_OUTPUT);
    pinMode(RST_PORT, GPIO_MODE_OUTPUT);
    pinMode(CS2_PORT, GPIO_MODE_OUTPUT);
    digitalWrite(CS2_PORT, PORT_HIGH);
    spi_initd.Baudrate    = 1000*1000;
    spi_initd.Direction  = SPI_DIRECTION_2LINES;
    :
    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();
    }
}
```

デジタルピンの初期化

SPIドライバの初期化
ハンドラを取り出す

spi_init関数呼び出し
エラーの場合停止

lcd_init.c: ソースファイル

- hlcdハンドラ、描画構造体設定後、lcd_initで初期化
- バックスクリーンを黒にして、緑ピクセル表示
- バックスクリーンを白にして停止

```
hlcd = &LcdHandle;
hlcd->hspi = hspi;
hlcd->rotation = 3;
hlcd->spi_lock = SPI1LOCK_SEM;
hlcd->cs_pin = CS_PORT;
hlcd->rs_pin = RS_PORT;
hlcd->rst_pin = RST_PORT;
hlcd->cs2_pin = CS2_PORT;
DrawProp.hlcd = hlcd;
lcd_init(hlcd, LCD_WIDTH, LCD_HEIGHT);
DrawProp.BackColor = ST7789_BLACK;
DrawProp.TextColor = ST7789_WHITE;

lcd_fillRect(hlcd, 0, 0, hlcd->_width, hlcd->_height, DrawProp.BackColor);
dly_tsk(1000);
lcd_drawPixel(hlcd, hlcd->_width/2, hlcd->_height/2, ST7789_GREEN);
dly_tsk(1000);

DrawProp.TextColor = ST7789_BLACK;
DrawProp.BackColor = ST7789_WHITE;
lcd_fillRect(hlcd, 0, 0, hlcd->_width, hlcd->_height, DrawProp.BackColor);

syslog_0(LOG_NOTICE, "LCD TEST END");
slp_tsk();
spi_deinit(hspi);
```

演習: lcd_initをビルドして動作確認

- lcd_initをビルド、実行させ、LCDの初期化が正しく行われることを確認する
- 学習内容
 - 一般的なSPI通信のグラフィックLCDの初期化確認
- 手順
 - program/lcd_initをmy_program/lcd_initにコピーする
 - my_program/lcd_initでビルドする
 - device rst(return)コマンドでボードリセット
 - ROMモニタを使って、ダウンロード実行する
 - LCDのバックグラウンド表示とピクセル表示が行われることを確認

演習：SPI通信速度変更

- SPIのクロックを変更する
- 初期化項目のBardrateで設定する
- Bardrateを変えて速度の検証を行う
 - 初期設定：1000*1000(1MHz)
 - 高速設定：10*1000*1000(10MHz)
 - 低速：200*1000(200KHz)
- 学習内容
 - SPIのクロックにより表示速度が変わることを確認
- 手順
 - プログラムを修正しビルド、ダウンロード、実行

LCDの初期化とピクセル設定

1. Adafruit ST7789グラフィックI/F
2. グラフィックLCD通信ドライバ
3. lcd_initプログラム
4. ピクセル設定

描画ウィンドウ設定

- ピクセルにデータを書込むとき、常に座標設定をしながら書込みを行うと転送データ量が多くなる
- 方形のWindowを設定を一度設定後、データ転送を行うと連続してWindowにデータ転送が可能になる

```
void lcd_setAddrWindow(LCD_Handle_t *hlcd, uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1)
{
    SVC_ERROR(lcd_writecommand(hlcd->hspi, ST7789_CASET)); /* カラムアドレスセット */
    SVC_ERROR(lcd_writedata(hlcd->hspi, 0x00));             /* X開始位置 */
    SVC_ERROR(lcd_writedata(hlcd->hspi, x0+hlcd->colstart));
    SVC_ERROR(lcd_writedata(hlcd->hspi, 0x00));             /* X終了位置 */
    SVC_ERROR(lcd_writedata(hlcd->hspi, x1+hlcd->colstart));
    SVC_ERROR(lcd_writecommand(hlcd->hspi, ST7789_RASET)); /* ロウアドレスセット */
    SVC_ERROR(lcd_writedata(hlcd->hspi, 0x00));             /* Y開始位置 */
    SVC_ERROR(lcd_writedata(hlcd->hspi, y0+hlcd->rowstart));
    SVC_ERROR(lcd_writedata(hlcd->hspi, 0x00));             /* Y終了位置 */
    SVC_ERROR(lcd_writedata(hlcd->hspi, y1+hlcd->rowstart));

    SVC_ERROR(lcd_writecommand(hlcd->hspi, ST7789_RAMWR)); /* write to RAM */
}
```

ピクセル設定

- LCD上にピクセルを設定するための関数lcd_drawPixel関数を示す
 - ピクセルの色はRGB:565に対応した定義を用意する

```
// Color definitions
#define ST7789_BLACK  0x0000
#define ST7789_BLUE   0x001F
#define ST7789_RED    0xF800
#define ST7789_GREEN  0x07E0
#define ST7789_CYAN   0x07FF
#define ST7789_MAGENTA 0xF81F
#define ST7789_YELLOW 0xFFE0
#define ST7789_WHITE  0xFFFF
void lcd_drawPixel(LCD_Handler_t *hlcd, int16_t x, int16_t y, uint16_t color)
{
    if((x < 0) || (x >= hlcd->_width) || (y < 0) || (y >= hlcd->_height))
        return;
    lcd_setAddrWindow(hlcd, x,y,x+1,y+1);
    SVC_ERROR(lcd_writedata2(hlcd, hlcd->hspl, color, 1));
}
```

ライン描画アルゴリズム

- Xの増分(ΔX)とYの増分(ΔY)を使用し、長い線を引く方向を基軸とし、短い線を引く方向を副軸とする
- 基軸が1ピクセル移動する毎に、副軸の増分を加算する
- 増分の加算値が主軸の増分を超えた場合、副軸に1ピクセル移動する

(x_0, y_0)から(x_1, y_1)に線を引く
Xの増分 ΔX 、Yの増分を ΔY
増分を保存する変数を、sumDeltaとする
 ΔX は $x_1 - x_0$ の絶対値
 ΔY は $y_1 - y_0$ の絶対値

ΔX が ΔY より大きい場合(Xが基軸、Yが副軸)
Xが1ピクセル移動する毎に、
sumDelta += ΔY
sumDeltaが ΔX 以上になれば、
sumDelta -= ΔX として、Y座標を1ピクセル移動する

ライン描画関数:前半

- 三角関数を使わないライン描画関数を以下に示す
- 詳細はコメント見ながら、ソースで理解してください

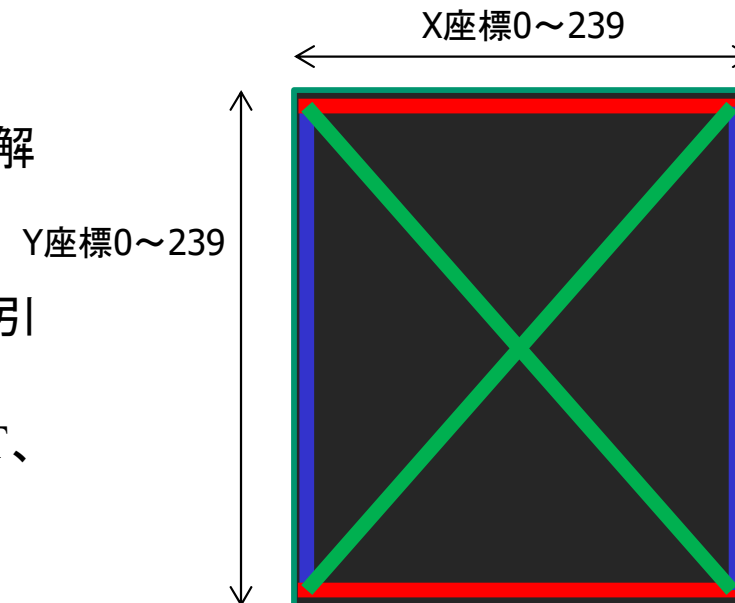
```
void lcd_drawLine(LCD_DrawProp_t *pDrawProp, uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2)
{
    LCD_Handler_t *hlcd = pDrawProp->hlcd;
    int16_t deltax = 0, deltay = 0, x = 0, y = 0, xinc1 = 0, xinc2 = 0,
    yinc1 = 0, yinc2 = 0, den = 0, num = 0, num_add = 0, num_pixels = 0,
    curpixel = 0;
    deltax = ABS(x2 - x1);          /* x1とx2の間のx方向ピクセル数 */
    deltay = ABS(y2 - y1);          /* y1とy2の間のy方向ピクセル数 */
    x = x1;                          /* xにx座標始点設定 */
    y = y1;                          /* yにy座標始点設定 */
    if(x2 >= x1){                    /* x終点が起点より大きいまたは等しい場合、x増数設定 */
        xinc1 = 1;
        xinc2 = 1;
    }
    else{                            /* x終点が起点より小さい場合、x減数設定 */
        xinc1 = -1;
        xinc2 = -1;
    }
    if(y2 >= y1){                    /* y終点が起点より大きいまたは等しい場合、y増数設定 */
        yinc1 = 1;
        yinc2 = 1;
    }
    else{                            /* y終点が起点より小さい場合、y減数設定 */
        yinc1 = -1;
        yinc2 = -1;
    }
}
```

ライン描画関数:後半

```
if(deltax >= deltay){                                /* x方向の増数がy方向より大きい場合 */
    xinc1 = 0;                                       /* x方向の分数増加分をゼロ */
    yinc2 = 0;                                       /* y方向の絶対増加分をゼロ */
    den = deltax;
    num = deltax / 2;
    num_add = deltay;
    num_pixels = deltax;                            /* There are more x-values than y-values */
}
else{                                                 /* x方向の増数がy方向より小さい場合 */
    xinc2 = 0;                                       /* x方向の絶対増加分をゼロ */
    yinc1 = 0;                                       /* y方向の分数増加分をゼロ */
    den = deltay;
    num = deltay / 2;
    num_add = deltax;
    num_pixels = deltay; /* There are more y-values than x-values */
}
for(curpixel = 0; curpixel <= num_pixels; curpixel++){
    lcd_drawPixel(hlcd, x, y, DrawProp.TextColor); /* ピクセル描画 */
    num += num_add;                                  /* 分数増減値を加算領域に加算 */
    if(num >= den){                                  /* 加算領域値が限界値を超えた場合 */
        num -= den;                                  /* 加算領域値から限界値を引く */
        x += xinc1;                                  /* x座標にx方向の分数増加分を加算 */
        y += yinc1;                                  /* y座標にy方向の分数増加分を加算 */
    }
    x += xinc2;                                      /* x座標にx方向の絶対増加分を加算 */
    y += yinc2;                                      /* y座標にy方向の絶対増加分を加算 */
}
}
```

演習: ライン描画を行う

- lcd_initを改造し、Line描画関数を用いてピクセル表示の後、赤、青、緑の線を描画するプログラムを作成
- 表示する線は、以下の6本
 - 赤: (0,0)-(239,0) (0,239)-(239,239)
 - 青: (0,0)-(0,239) (239,0)-(239,239)
 - 緑: (0,0)-(239,239) (239,0)-(0,239)
- 学習内容
 - 簡単な線描画のアルゴリズムの理解
- 手順
 - line_draw関数を使って指定の線を引く
 - 縦横のピクセル数はLCD_HEIGHT、LCD_WIDTHを使用



演習:ライン描画解答

- 緑のピクセル描画の後に、DrawProp.TextColorの色を変えながらライン描画を行い、dly_tskで5000msの待ちを入れる

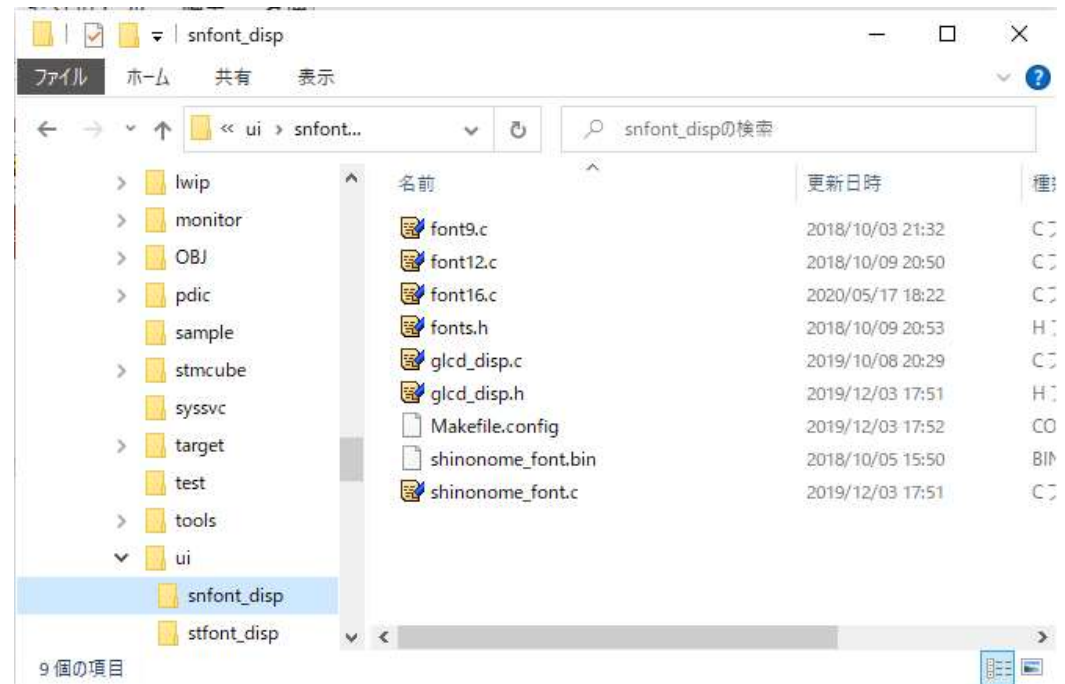
```
lcd_drawPixel(hlcd, hlcd->_width/2, hlcd->_height/2, ST7789_GREEN);  
dly_tsk(1000);  
  
DrawProp.TextColor = ST7789_BLUE;  
lcd_drawLine(&DrawProp, 0, 0, 0, LCD_HEIGHT-1);  
lcd_drawLine(&DrawProp, LCD_WIDTH-1, 0, LCD_WIDTH-1, LCD_HEIGHT-1);  
DrawProp.TextColor = ST7789_RED;  
lcd_drawLine(&DrawProp, 0, 0, LCD_WIDTH-1, 0);  
lcd_drawLine(&DrawProp, 0, LCD_WIDTH-1, LCD_WIDTH-1, LCD_HEIGHT-1);  
DrawProp.TextColor = ST7789_GREEN;  
lcd_drawLine(&DrawProp, 0, 0, LCD_WIDTH-1, LCD_HEIGHT-1);  
lcd_drawLine(&DrawProp, LCD_WIDTH-1, 0, 0, LCD_HEIGHT-1);  
dly_tsk(5000);  
  
DrawProp.TextColor = ST7789_BLACK;
```

LCD描画プログラム実習

1. [Fontの説明](#)
2. Fontを使ってテキスト表示
3. UIミドルウェア

グラフィックLCDの文字表示

- グラフィックLCDに文字を表示する場合、フォントを使用する
 - ビットマップフォント: ビットデータを使用したフォント
 - アウトラインフォント: フォントの外形データを使用したフォント
- 本実習では、オープンなビットマップフォントを用いる
 - asp/ui/snfont_disp
- 9/12/16ポイントのアスキーデータと12/16の漢字フォントを持つ



ビットマップフォントの構成

- フォントデータはfont_head_tとfont_tの2つの構造体で管理される
- font_head_t構造体は、フォントヘッダーと呼ばれフォントの種別を管理する、以下で構成
 - フォントの名称、属性、基本の縦横サイズ
 - フォントテーブル(font_t)へのポインタ

```
typedef struct _font_head_s {  
    const char  font_name[32];  
    uint8_t     file_attribute;  
    uint8_t     font_attribute;  
    uint8_t     font_width;  
    uint8_t     font_height;  
    const font_t **font_table;  
    void        *pdata;  
} font_head_t;
```

```
font_head_t Embedded_Font9 = {  
    "embedded_font9",  
    0x00,  
    0x00,  
    6,  
    9,  
    Font9,  
    NULL  
};
```

ビットマップフォントの構成

- font_t構造体は実際のフォントイメージをコード単位で管理する
- font_head_t中font_tableは文字コードに対応したフォントイメージのポインターの配列を指す

```
typedef struct _font_s {  
    uint8_t    width;  
    uint8_t    height;  
    uint8_t    stride;  
    uint8_t    iheight;  
    uint8_t    image[8];  
} font_t;
```

```
static const font_t *Font9[] = {  
    (font_t *)Font9_32, (font_t *)Font9_01, (font_t *)Font9_02, (font_t *)Font9_03,  
    (font_t *)Font9_04, (font_t *)Font9_05, (font_t *)Font9_06, (font_t *)Font9_07,  
    (font_t *)Font9_08, (font_t *)Font9_09, (font_t *)Font9_10, (font_t *)Font9_11,  
    (font_t *)Font9_12, (font_t *)Font9_13, (font_t *)Font9_14, (font_t *)Font9_15,  
    :  
};
```

ビットマップフォントの構成

- 配列ポインタの65番が大文字Aとなる
 - widthが幅、6ピクセル
 - heightが高さ、9ピクセル
 - strideが1バイト(ラインのバイト数)
 - iheightがラインの数、9ライン
 - imageの場所にフォントのイメージデータが対応する

```
typedef struct _font_s {  
    uint8_t    width;  
    uint8_t    height;  
    uint8_t    stride;  
    uint8_t    iheight;  
    uint8_t    image[8];  
} font_t;
```

```
static const uint8_t Font9_65[] = {  
    6, 9,  
    1, 9,  
    0x70, //   ###  
    0x88, // #   #  
    0x88, // #   #  
    0x88, // #   #  
    0xF8, // #####  
    0x88, // #   #  
    0x88, // #   #  
    0x00, //  
    0x00 //  
};
```

大文字'A'の
データ

フォントの描画方法

- アスキーコードに対応した文字データを、データが1 (ON) の場合、foreground色(TextColor)、データが0 (OFF) の場合、background色(BackColor)で、LCD上にピクセル単位で書き込むことによって、文字を表示する



LCD描画プログラム実習

1. Fontの説明
2. Fontを使ってテキスト表示
3. UIミドルウェア

演習：一文字表示

- 一文字表示するプログラムを作成する
- 学習内容
 - 一般的なビットマップフォントの表示方法の学習
- 手段
 - lcd_char1ディレクトリ中にA,B,Cを一文字ずつ表示するプログラムを用意している
 - 但し、LCDに文字データを書き込むプログラム(DrawChar)が未完成である
 - DrawCharを正しく完成させる
- 引数：(x0, y0) 表示文字の左上座標
- pf 表示するfont_tデータを指すポインタ

```
static void DrawChar(LCD_DrawProp_t *pDrawProp, uint16_t x0, uint16_t y0, const font_t *pf)
{
    LCD_Handler_t *hlcd = DrawProp.hlcd;
    :
}
```

メインタスクからの呼び出し

- メインタスクでは、DrawProp.pFontに表示したいフォントのsFont構造体をセットし、lcd_DisplayCharにて表示を行う
- lcd_DisplayCharはアスキーコードに対応した文字データの位置を計算して、DrawChar関数を呼び出す

```
void lcd_DisplayChar(LCD_DrawProp_t *pDrawProp, uint16_t x0, uint16_t y0, uint8_t Ascii)
{
    font_head_t *phead = (font_head_t *)pDrawProp->pFont;
    const font_t **pf  = phead->font_table;

    DrawChar(pDrawProp, x0, y0, pf[Ascii]);
}
```

```
DrawProp.pFont = &Font16;
lcd_DisplayChar(&DrawProp, 10, 10, 'A');
lcd_DisplayChar(&DrawProp, 30, 10, 'B');
lcd_DisplayChar(&DrawProp, 50, 10, 'C');
```

Font16のビットマップ構成

- フォントデータ中のビットをビット演算で判定する
- フォント・イメージ中の幅のバイト数をstrideと呼ぶ、フォントの幅は8ビット、高さは16ライン
- フォントのstrideは以下の計算
 - $\text{stride} = 1\text{byte} = (8+7) / 8$
 - 既にstrideに1が設定済
- 各ラインのポインタはstrideを使って計算する

```
static const uint8_t Font16_65[] = {  
    8, 16,  
    1, 16,  
    0x00, //  
    0x10, // #  
    0x10, // #  
    0x28, // # #  
    0x28, // # #  
    0x28, // # #  
    0x44, // # #  
    0x44, // # #  
    0x44, // # #  
    0x7C, // #####  
    0x82, // # #  
    0x82, // # #  
    0x82, // # #  
    0x82, // # #  
    0x00, //  
    0x00 //  
};
```

演習: 文字表示の解答

- フォントの幅、高さを取得し、描画ウィンドウを設定する
- 水平方向に一ラインずつ、文字データを取り出しデータビットが1の場合、TextColorをセット、0の場合、BackColorをピクセル単位に設定する
- 描画ウィンドウ設定で、1ピクセルずつ座標設定は不要
- pcharを各フォントラインの先頭ポインタ、wを水平ビット位置、hを垂直ラインとすると、ビットの0/1判定は以下の通り

```
if((pchar[w/8] & (1 << (7 - (w % 8)))) != 0){  
    lcd_drawPixel(hlcd, (x0 + w), (y0 + h), pDrawProp->TextColor);  
}  
else{  
    lcd_drawPixel(hlcd, (x0 + w), (y0 + h), pDrawProp->BackColor);  
}
```

演習: 文字表示の解答

```
static void DrawChar(LCD_DrawProp_t *pDrawProp, uint16_t x0, uint16_t y0, const font_t *pf)
{
    LCD_Handler_t *hlcd = DrawProp->hlcd;
    uint32_t h, w;
    const uint8_t *pchar;

    if(pf->stride == 0 || pf->iheight == 0)
        return;
    for(h = 0 ; h < pf->iheight ; h++){
        pchar = (pf->image + pf->stride * h);
        for(w = 0 ; w < pf->width ; w++){
            if((pchar[w/8] & (1 << (7 - (w % 8)))) != 0){
                lcd_drawPixel(hlcd, (x0 + w), (y0 + h), pDrawProp->TextColor);
            }
            else{
                lcd_drawPixel(hlcd, (x0 + w), (y0 + h), pDrawProp->BackColor);
            }
        }
    }
}
```

フォントの幅、高さを取り出し
描画ウィンドウを設定

フォント中のビットを
バイナリ演算で判定

lcd_drawPixel関数を使って
LCDに書込み

演習:テキスト表示

- 文字列「Hello World !」を表示するプログラムを完成させる
- 学習内容
 - 一般的な文字表示手順の学習
- 手段
 - lcd_text1ディレクトリ中に文字列Hello World !プログラムを用意している
 - 但し、文字列を解析し一文字表示関数を呼び出すプログラム(lcd_DisplayString)が未完成である
 - lcd_DisplayStringを正しく完成させる
- 引数: (x0, y0) 表示文字列の左上座標
- Text: アスキー文字列へのポインタ

```
static void lcd_DisplayString(LCD_DrawProp_t *pDrawProp,uint16_t x0,  
                             uint16_t y0, const char *Text)  
{  
    :  
}
```

演習: 文字列表示の解答

- フォントヘッダーとフォントイメージのポインターの配列を取り出す
- 変数の定義と初期設定、カラム位置の調整

```
void lcd_DisplayString(LCD_DrawProp_t *pDrawProp, uint16_t x0, uint16_t y0, char *Text)
{
    LCD_Handler_t *hlcd = pDrawProp->hlcd;
    font_head_t *phead = (font_head_t *)pDrawProp->pFont;
    const font_t **pf = phead->font_table;
    uint16_t ref_column = x0;
    uint8_t c;
    int wid, len = 1;

    /*
     * 開始カラム位置をスクリーン内に収める
     */
    if(ref_column >= hlcd->width){
        ref_column = 0;
    }
}
```

演習: 文字列表示の解答

- Textから文字コードを取り出しフォントの文字データ位置を計算してlcd_DisplayChar関数で表示、カラム位置を右に移動

```
/*
 * 一文字描画を使って文字列を描画する
 */
while (((c = *Text) != 0) && pf[c] != 0
        && (ref_column + (wid = GetFontWidth(phead, c, &len))) < hlcd->_width){
    lcd_DisplayChar(pDrawProp, ref_column, y0, c);
    ref_column += wid;          /* 次のカラムに移動 */
    Text += len;               /* 次の文字に移動 */
}
```

文字列中の文字分だけ、文字データ位置を計算して
lcd_DisplayChar関数を呼び出す

LCD描画プログラム実習

1. Fontの説明
2. Fontを使ってテキスト表示
3. UIミドルウェア

演習: UIミドルウェアを使う

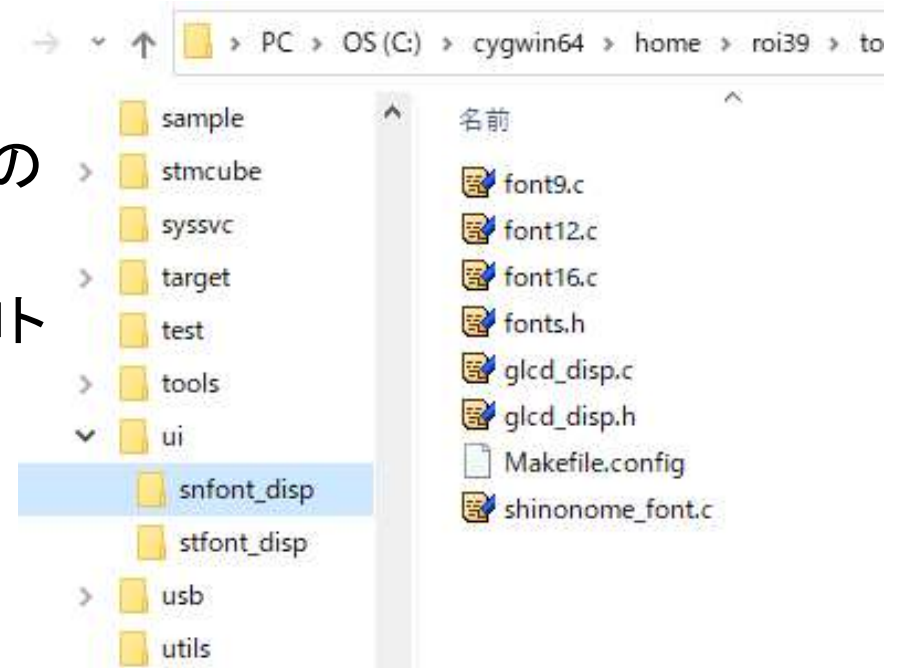
- テキスト表示をUIミドルウェアを使って表示する
 - グラフィック、文字描画をミドルウェアが肩代わりする
- 学習内容
 - ミドルウェアを使って文字列表示を行う
- 手段
 - テキスト表示に対応したlcd_text1を改造してUIミドルウェアに対応する
- TOPPERS BASE PLATFORMでは2つのUIミドルウェアの対応を行っている
 - snfont_disp: 東雲フォントを使った文字表示
 - stfont_disp: STマイクロエレクトロニクス社のフォントで文字表示
- 最終的にファイルシステムから漢字を表示させるため、snfont_dispを使用
 - ファイルシステムの対応は2日目

演習：UIミドルウェアを使う

- 作成したlcd_text1をUIミドルウェアに対応、後で比較を行えるようlcd_text3として作成する
- 作成方法
 - lcd_text1をmy_programにコピーして名前をlcd_text3に変更

snfont_disp

- asp/ui/snfont_dsp中に文字列表示用のミドルウェアのソースがある
- glcd_disp.hの文字列表示関数のプロトタイプ宣言がある
 - Modeで表示属性が指定できる
- CENTER_MODEで中央に表示



```
/*
 * テキスト表示ラインモード定義
 */
typedef enum
{
    CENTER_MODE = 0x01, /* Center mode */
    RIGHT_MODE,          /* Right mode */
    LEFT_MODE            /* Left mode */
}Line_ModeTypdef;

extern void lcd_DisplayStringAt(LCD_DrawProp_t *pDrawProp, uint16_t Xpos, uint16_t Ypos, uint8_t *Text,
                               Line_ModeTypdef Mode);
```

表示位置方法を指定

演習 : Makefileの修正

- UIミドルウェアに対応するためにMakefile.configを取り込む
 - SYSSVC_DIRとINCLUDESのui/snfont_disp参照は不要
- 不要なフォントファイルの取り込みを削除
 - フォントファイルはミドルウェアで取り込むため

```
188 #
189 # ミドルウェア Makefile のインクルード
190 #
191 include $(SRCDIR)/ui/snfont_disp/Makefile.config
192 include $(SRCDIR)/monitor/Makefile.config
    :
207
    #
    # フォントファイルの取り込み
    #
    SYSSVC_DIR := $(SYSSVC_DIR) $(SRCDIR)/ui/snfont_disp
    INCLUDES := $(INCLUDES) -I$(SRCDIR)/ui/snfont_disp
    SYSSVC_COBJS := $(SYSSVC_COBJS) font16.o font12.o

208 #
209 # システムサービスに関する定義
210 #
```

フォントの取り込みは
不要なので削除

演習: ソースの修正

- lcd_text.cへの修正
- インクルードファイルFonts.hをglcd_disp.hに変更
- 以下の関数は未使用となる
 - GetFontWidth(), lcd_getFontTable(), DrawChar(), lcd_DisplayChar(), lcd_DisplayString()
 - これらのソースは#ifdef _GLCD_DISP_H_ ~ #endifで自動でコンパイルされない
- lcd_DisaplayStringをlcd_DisplayStringAtに修正

```
#include "adafruit_st7789.h"
#include "glcd_disp.h"
#include "lcd_text.h"
:
:
DrawProp.pFont = &Font16;
lcd_DisplayStringAt(&DrawProp, 0, 80, "Hello World !", CENTER_MODE);
DrawProp.TextColor = ST7789_BLUE;
DrawProp.pFont = &Font12;
lcd_DisplayStringAt(&DrawProp, 0, 110, "Hello World !", CENTER_MODE);
```

まとめ

基礎3実習1日目のまとめ

- MSYS2/Cygwinを用いたGCCのARM開発環境の構築
- ADCの理解とドライバ/制御方法の理解
- SPI仕様とハードウェアの理解
- LCD上位(GDIC)、描画プログラムの理解
- グラフィックLCDに対する一般的な描画方法
 - 初期化
 - ピクセル表示
 - 線描画
 - 文字列表示

1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	: 2日目
sd_ini3	: 2日目
sd_cmd1	: 2日目
sd_cmd2	: 2日目
sd_file2	: 2日目
sd_file3	: 2日目

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(RP) REFERENCE MANUAL
- 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

- TOPPERS BASE PLATFORM(ST)
- 開発環境とハードウェアの検証
- JOY STICKを使って、ADC入力
- SPI仕様、SPIデバイスドライバ
- LCDの初期化とピクセル設定
- LCD描画プログラム実習
 - 竹内 良輔(TOPPERSプロジェクト教育WG主査)

教材に関するご意見は,
secretariat@toppers.jp
までお寄せください.