TOPPERS

TOPPERS BASE PLATFORM (RP) V1.4.4

担当者	竹内良輔

変更履歴

リリース	日付	作成者	変更内容
1.4.4	2023/05/19	竹内良輔	初版

目次

1　背景

TOPPERS BASE PLATFORM の拡張として RP2040(Cortex-M0+)プロセッサに対応する。Raspberry Pi PICO ボードに Arduino コネクタ対応したボードに実装し TOPPERS BASE PLATFORM(RP)を開発することとなった。

2　目的

本リファレンスマニュアルは、ターゲットボードとターゲットボード上に作成したソフトウェアプラットフォーム(TOPPERS BASE PLATFORM(RP))の仕様について記載する。

■ターゲットボード
・PICO Raspberry PI 社
・PICO W Raspberry PI 社

3　TOPPERS BASE PLATFORM (RP)仕様

本章では、TOPPERS BASE PLATFORM (RP)の仕様について記載する。

3.1 プラットフォームのターゲット

TOPPERS BASE PLATFORM(RP) は上記 3 つのボードに対応した TOPPERS ASP-1.9.3 カーネルと FMP-1.4.0 カーネル上で実行するデバイスドライバとミドルウェアに対応する。

TOPPERS 教育 WG の多くの資産を使用するために、TEB003（Arduino Connector ボード）を開発し、このボード上で円滑に動作することを想定する。TEB003 は DIP 半田で部品を実装して使用する。

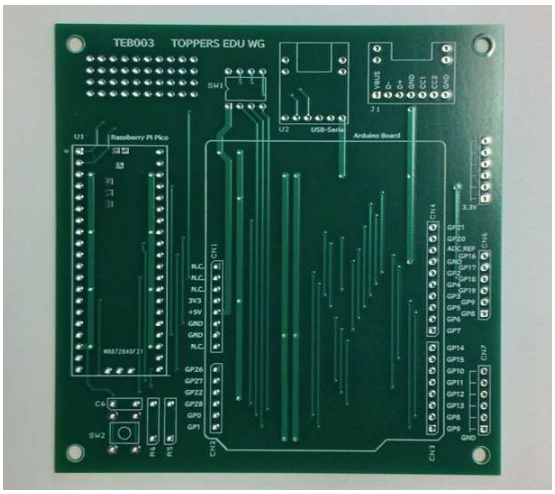

図 3.1.1TEB003 ボード

図 3.1.2 実装し、PICO W と TEB002 を追加

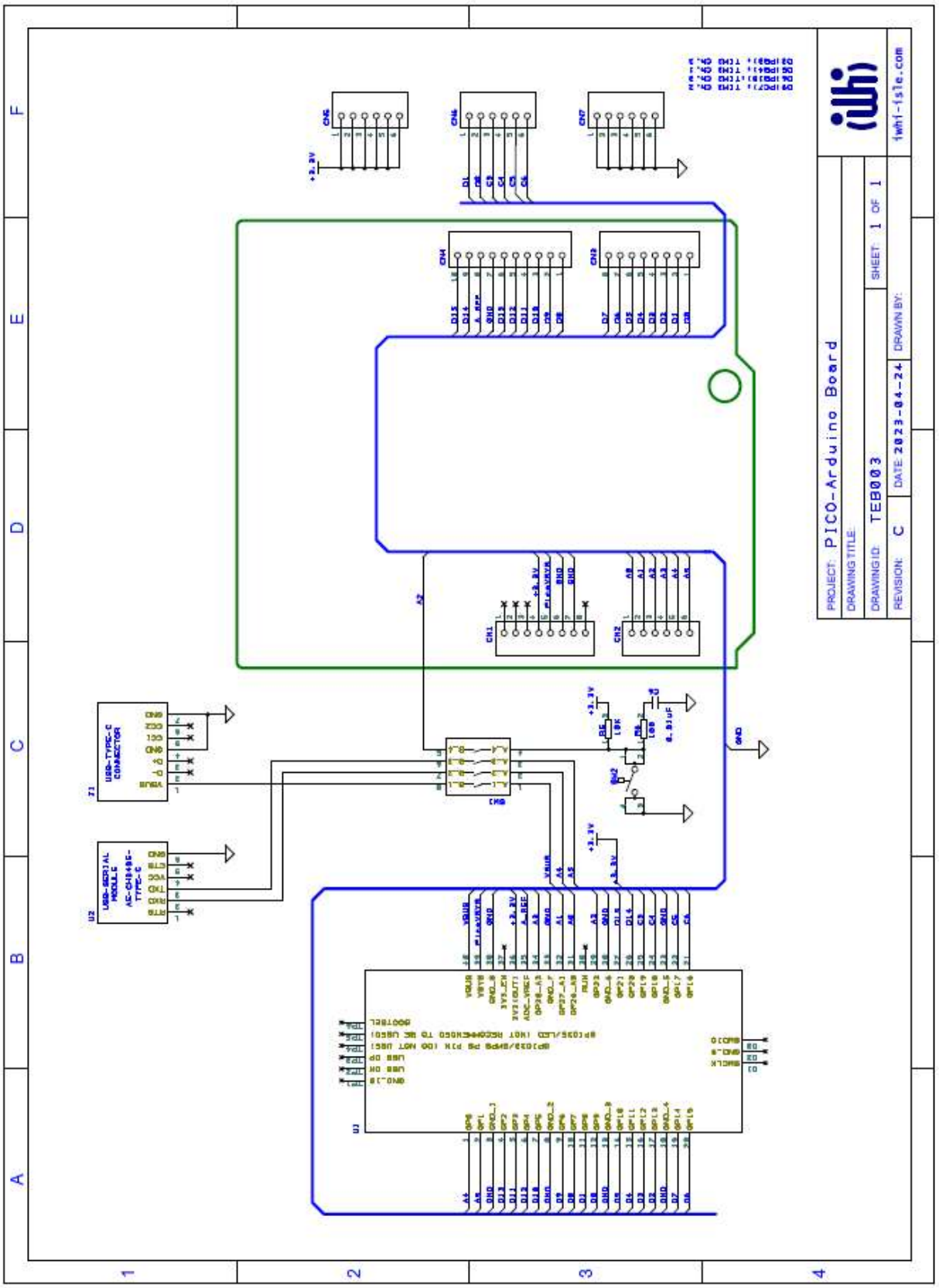

図 3.1.3 TEB003 回路図

名称	型番	単価	個数		部品代
Raspberry Pi PICO	M-16132	730	1		730
細ピンヘッダ 1x20 (20P)	C-04398	20	2		40
細ピンヘッダ 1x3 (3P) (10 個入)	C-04391	5	1		5
ＣＨ３４０Ｅ　ＵＳＢシリアル変換モジュール　Ｔｙｐｅ－Ｃ	K-14745	550	1		550
USBtype-C コネクタ DIP 化キット	K-13080	350	1		350
ARDUINO 用ピンソケット 10P+8P FH-10+8-S	C-17504	80	1		80
ARDUINO 用ピンソケット 8P+6P FH-8+6-S	C-17503	70	1		70
ピンソケット（メス） 1x20 (20P)	C-03077	50	2		100
ピンソケット（メス） 1x6 (6P)	C-03784	20	2		20
ピンソケット（メス） 1x6 (6P)赤	C-10284	20	1		40
ピンソケット（メス） 1x6(6P)青	C-10286	20	1		20
DISP-SW 4	P-00586	60	1		20
プラスチックナット＋連結スペーサー10mm セット	P-01864	100	1		60
タクトスイッチ（黄色）	P-03650	10	1		10
セラミックコンデンサ 0.01μF	P-02281	100	1		100
抵抗 10KΩ 1/6W	R-16103	100	1		100
抵抗 100Ω 1/4W	R-25101	100	1		100
					2395

表 3.1.1 TEB003 部品表(2023 年 5 月時点の価格)

asp/fmp カーネルには、PICO/PICO W のボードの依存はありません。TOPPERS BASE PLATFORM でのボード指定コンパイルスイッチ BOARDTYPE で、正しくボードを指定してください。

BOARDTYPE=PICO : RASPBERRY PI PICO ボードを指定

BOARDTYPE=PICO_W : RASPBERRY PI PICO W ボードを指定

BOARDTYPE と実際のボードが異なった状態で実行した場合、デバッグコマンドにて LED の点灯、消灯を指定しても、LED は正しく動作しません。また、BOARDTYPE=PICO_W でビルドした実行モジュールを PICO ボードで実行した場合、LED や WIFI 制御で cyw43 の通信エラーが発生します。

3.2 レイア構造

TOPPERS BASE PLATFORM(RP)のハードウェアドライバは下層から３層のレイヤ構造を持ち、その上に API 層、ライブラリの I/F 層をもつ。PDIC の API が ASP-1.9.3 カーネルを使用している TOPPERS BASE PLATFORM(ST)とほぼ同じため、GDIC やミドルウェアは、そのまま使用可能である。また、ASP カーネルのもつデバッグ機能以外に、教育ＷＧで提供するタスクモニタを標準に装備し、システムデバッグ用にデバッグコマンドを用いて開発補助を行う。FMP-1.4.0 カーネルに関しては、TOPPERS BASE PLATFORM(CV)のリソース上に、ASP カーネルで作成したドライバを展開して使用する。

図 3.2.1 に TOPPERS BASE PLATFORM (RP)の構造図を示す。Base Driver、Standard Driver、GDIC、UI ミドルウェア等により、アプリケーションから以下の機能が使用可能になる。

① SPI
② I2C
③ UART
④ RTC
⑤ WDOG
⑥ ADC
⑦ USB ホスト・デバイス

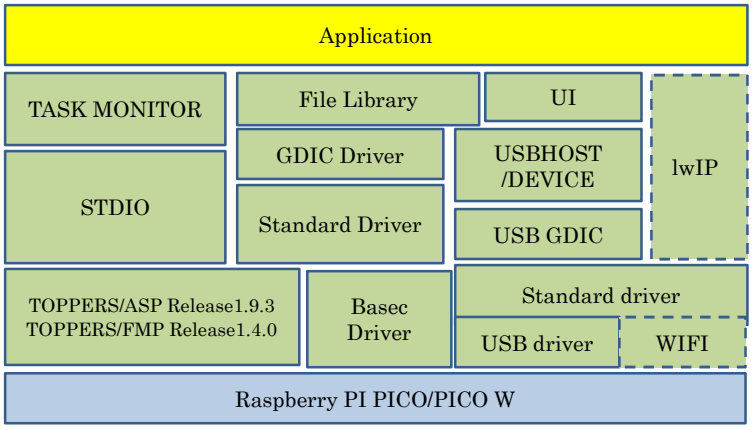

図 3.2.1 TOPPERS BASE PLATFORM(RV)構造図

以下に、レイア構造の概要を示す。

(1)basic driver ハードウェア仕様により API が変わるドライバ層
(2)standard driver 拡張ボードの標準化により、API がある程度標準化されているドライバ層
(3)GDIC driver 下位の driver を使用して構築する特定のデバイス用 driver
(4)TOPPERS BASE PLATFORM で標準化している API 層
(5)open source のライブラリ等

PLATFORM(RP)		ターゲットボード	detail	対象の講座
Device Driver	Basic Driver	gpio		基礎１，２，３ 上級１ 上級 USB
		dma		
		watch doc		
		rtc		
		(uart)	カーネル用を使用	
	Standard Driver	i2c	CLCD/senser/eeprom	
		spi	GLCD/SD card	
		adc		
		usb host/device	MSC/HID	
		wifi	CYW43 用	
gdic	high driver		デバイス結合	
api middleware	File System	fatfs	FAT	上級１，２
		file library	C language file	

		stdio	stdio	
	Graphic UI	ui	文字グラフィック表示	
	(Open source) library	USB	USB ミドルウェア	
		lwip (*1)	プロトコル・スタック	

*1：一部パッチによる修正が必要：ソースコードは対応 Web からダウンロードする必要がある

3.3 開発環境

TOPPERS BASE PLATFORM(RP)は Windows10/11 上に MSYS2 をインストールし、GCC コンパイラソースをビルドして開発を行っている。ソースコードを公開しているため、LINUX でも開発可能である。

(1) gcc version 5.4(GCC ARM-2016q2-20160926)
(2) gcc version 10.2.1(GCC ARM-2020q4-major)

4 Device Driver 仕様

ハードウェア用デバイスドライバの仕様について記載を行う。本 TOPPERS BASE PLATFORM(RP)では、３種類のデバイスドライバを提供する。

4.1 Basic Driver

4.1.1 概要

Basic Driver は、ハードウェアを制御する基本的なドライバ群である。制御は簡単な制御手順であるが、初期化や拡張機能は、SoC によってまちまちの実装が行われており、標準的な API では作成できないものが多い。また、Basic Driver は直接ミドルウェアやアプリから制御を行うより、上位のドライバから使用機能として呼び出すケースが多い。逆に Basic Driver は他のドライバの呼び出しは行わない。Basic Driver はハードウェアの依存性が大きいため、PLATFORM を別の SoC にポーティングする場合、別の API の実装となる。

UART は、asp カーネルで使用されている。基本的には asp カーネルのドライバを使用する。差分のみを Basic Driver として記載する。タイマーに関しては、FE310-G000 は machine timer しか持たない、machine timer が asp カーネルのシステムタイマーとして使用しているので、タイマーの使用はサービス・コールに順ずる。

4.1.2 ドライバー覧

Basic Driver として分類するドライバは以下の４つである。

(1) gpio　汎用ＩＯドライバ
(2) dma　DMA ドライバ
(3) rtc　リアルタイム制御ドライバ
(4) wdoc　ウォッチドックタイマードライバ
(5) uart　シリアルインターフェイスドライバ

4.1.3 GPIO

GPIO は汎用のＩＯを制御するドライバである。GPIO はピン設定を入力または出力に設定し、ピンに対してデータを読み込むまたは書き込みことにより、外部のロジックとのデータ交換を行う機能を持つ。機能的には単純であるが、ピンアサイン、ベースの電圧設定、出力モード設定、割込みの対応等、初期化に関して SoC の設計により、設定仕様がまちまちであり、標準的な初期化手順を作ることが難しい。

K210 では GPIO ポートはひとつでピン番号は、0 から 29 までの 30 ピンである。GPIO ピンは pinmux_setup 関数でモードファンクションを設定し、gpio_setup 関数でピンの細かな属性を設定する。

4.1.3.1 データ仕様

GPIO の初期化に用いるデータと構造体について記載する。GPIO の初期化には表 4.1.3.1 の GPIO_Init_t 型を使用する。

番号	項目	型	機能
1	mode	uint32_t	対象のピンアサインの設定モード
2	pull	uint32_t	出力 Pull-Up/Pull-Down 設定
3	over	uint32_t	オーバーライド設定

表 4.1.3.1 GPIO_Init_t 型

① mode
モードはピンアサインの GPIO モードを設定する

定義	値	内容
GPIO_MODE_INPUT	0x00000000	GPIO 入力モード
GPIO_MODE_IT_FALLING	0x10010000	GPIO 入力フォーリングエッジ割込み

GPIO_MODE_IT_RISING	0x10110000	GPIO 入力ライジングエッジ割込み
GPIO_MODE_IT_RISING_FALLING	0x10210000	GPIO 入力両エッジ割込み
GPIO_MODE_LVL_LOW	0x10000000	GPIO 入力 LOW レベル割込み
GPIO_MODE_LVL_HIGH	0x10100000	GPIO 入力 HIGH レベル割込み
GPIO_MODE_OUTPUT	0x00000001	GPIO 出力モード
GPIO_MODE_ANALOG	0x00000002	GPIO アナログ入力設定
GPIO_MODE_FUNCTION	0x00000011	ファンクション設定

表 4.1.3.2 mode 設定値

② pull
pull は GPIO 出力設定。

定義	値	内容
GPIO_NOPULL	0x00000000	プルアップ無効
GPIO_PULLUP	0x00000008	プルアップ有効
GPIO_PULLDOWN	0x00000004	プルダウン有効

表 4.1.3.3 pull 設定値

③ over
オーバーライド設定を行う

定義	値	内容
GPIO_OUTOVER_NORMAL	0x00000000	未設定
GPIO_OUTOVER_INVERT	0x00000100	
GPIO_OUTOVER_LOW	0x00000200	
GPIO_OUTOVER_HIGH	0x00000300	
GPIO_OEOVER_NORMAL	0x00000000	未設定
GPIO_OEOVER_INVERT	0x00001000	
GPIO_OEOVER_DISABLE	0x00002000	
GPIO_OEOVER_ENABLE	0x00003000	
GPIO_INOVER_NORMAL	0x00000000	未設定
GPIO_INOVER_INVERT	0x00010000	
GPIO_INOVER_LOW	0x00020000	
GPIO_INOVER_HIGH	0x00030000	

表 4.1.3.4 over 設定値

4.1.3.2 インターフェイス仕様

GPIO を初期設定するドライバ関数を以下に示す。

関数名	型	引数	機能	備考
pinmix_setup	void	uint8_t gpio uint8_t modefunc	gpio 番号で指定されたポートをモードファンクション設定する	GPIO のモードファンクションは VALUE_SIO(5)である
gpio_setup	void	GPIO_Init_t *init uint8_t gpio	gpio 番号で指定されたポートを初期化する。	

表 4.1.3.6 GPIO 設定関数

4.1.3.3 フローチャート

基本的な GPIO の出力設定のフローチャートを以下に示す。

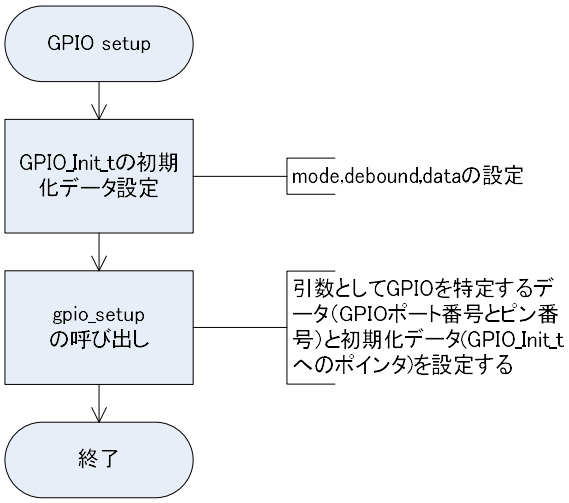

図 4.1.3.3.1 GPIO の設定フローチャート

4.1.4 DMA

DMA は CPU を通さず、直接メモリとデバイスまたはメモリ間でデータ転送を行う機構である。デバイスとメモリ間で高速にデータ通信した場合、オーバーラン・エラーやアンダーラン・エラーが発生するケースが多く、Standard Driver では DMA を使用している。

4.1.4.1 データ仕様

DMA ドライバは初期設定用に DMA_Init_t 型、制御を行うためにハンドラとして使用する DMA_Handle_t 型の二つの型を持つ。

番号	項目	型	機能
1	Channel	uint32_t	DMA のチャネル番号
2	ChainChannel	uint32_t	転送終了後、次に起動する DMA チャネル番号
3	Request	uint32_t	転送に使用するペリフェラルの DMA リクエスト番号
4	Mode	uint32_t	転送モード
5	MultiRequest	uint32_t	ペリフェラルのデータアラインメントを設定する
6	Priority	uint32_t	転送開始のトリガ設定
7	Width	uint32_t	転送データ幅
8	Increment	uint32_t	転送幅転送終了後のアドレスインクリメント設定
9	RingSize	uint32_t	レング転送する場合のデータサイズ
10	SniffData	uint32_t	スニフデータ指定

表 4.1.4.1 DMA_Init_t 型

番号	項目	型	機能
1	base	uint32_t	DMA チャネルコントローラのベースアドレス
2	cbase	uint32_t	DMA チャネル領域の先頭アドレス
3	Init	DMA_Init_t	DMA 初期化情報
4	control	uint32_t	DMA 制御レジスタ保存値
5	xfercallback	void *func	転送終了時のコールバック関数
8	ErrorCode	uint32_t	エラーコード
9	localdata	void*	ローカル領域へのポインタ
10	localvalue	uint32_t	ローカル領域で保存予定のデータ領域

表 4.1.4.2 DMA_Handle_t 型

4.1.4.2 インターフェイス仕様

RTC を設定するドライバ関数を以下に示す。

関数名	型	引数	機能	備考
dma_init	ER	DMA_Handle_t *	DMA チャネルの初期化	
dma_deinit	ER	DMA_Handle_t *	DMA チャネル終了関数	
dma_start	ER	DMA_Handle_t * uint32_t SrcAddress uint32_t DstAddress uint32_t DataLength	指定のアドレスでデータ転送を開始する	
dma_end	ER	DMA_Handle_t *	転送終了	
dma_change_increment	ER	DMA_Handle_t * uint32_t Increment	DMA のインクリメント設定変更	
dma_int	void		DMA 割込みハンドラ	

表 4.1.4.2.1 DMA 設定関数

4.1.5 RTC

RTC(Real-time clock)は時刻を管理するハードウェアである。HI-FIVE1 ボードでは RTC 用のクロックは実装されているが、バックアップ用バッテリは実装されていないため電源を落とすと時刻はリセットされる。RTC は時刻管理以外にアラート機能があり、アラート時刻を設定すると割込みにより、アラート通知を受け取ることができる。

4.1.5.1 データ仕様

RTC では UNIX で使用されている時刻管理構造体 tm と共有して時刻データのやり取りを行えるように表 4.2.6.1.1 として構造体名を変えた tm2 構造体を定義する。この構造体は tm 構造体と混在して使用されても定しくデータの受け渡しを行える。

RTC のアラーム動作設定用に割込みからのコールバック関数を用意する。

番号	項目	型	機能
1	tm_sec	int	秒(0～59)
2	tm_min	int	分(1～60)
3	tm_hour	int	時(0～23)
4	tm_mday	int	月の中の日
5	tm_mon	int	月
6	tm_year	int	年(1970 を 0 とした年数)
7	tm_wday	int	曜日
8	tm_yday	int	年の中の日
9	tm_isdst	int	夏時間： 0 以外の値

表 4.1.5.1.1 tm2 構造体

4.1.5.2 インターフェイス仕様

RTC を設定するドライバ関数を以下に示す。

関数名	型	引数	機能	備考
rtc_init	void	intptr_exinf	RTC の初期化を行う。引数に意味なし	
rtc_set_time	ER	struct tm2 *ptm	時刻をセットする	
rtc_get_time	ER	struct tm2 *ptm	時刻を取り出す	
rtc_setalarm	ER	struct tm2 *ptm void *func	アラートをセットする	
rtc_handler	void	void	割込みハンドラ	

表 4.1.5.2.1 RTC 設定関数

4.1.5.3 設定手順

初期化は rtc_init 関数を用いて行う。ATT_INI を使用して設定が行えるように引数を用意したが、この引数に意味はない。使用は以下の手順に従う。

① rtc_set_time
時刻の設定を行う。tm2 構造体中に設定に使用するのは以下の 6 つの項目で他の項目は意味を持たない。
(1) tm_year
(2) tm_mon
(3) tm_mday
(4) tm_hour
(5) tm_min
(6) tm_sec

② rtc_get_time
時刻を取り出す。tm2 構造体中に実際に設定される項目は以下の 7 つの項目である。
他の設定も設定したい場合は mktime 関数を用いて設定を行う必要がある。
(1) tm_year
(2) tm_mon
(3) tm_mday
(4) tm_wday
(5) tm_hour
(6) tm_min
(7) tm_sec

③ rtc_setalarm
ptm の時刻と現在時刻の比較を行い、ptm の時刻に達した場合、割込みが発生しコールバック関数が呼び出される。

4.1.6 WATCH DOG

WATCH DOG はシステムに異常が発生した場合、リセットを行う機能である。システムは WATCH DOG TIMER を起動し、一定周期でタイマーのリセットを行うことにより WATCH DOG に正常動作を通知する。

4.1.6.1 データ仕様

WATCH DOG はポート番号により選択できる。ドライバ I/F はポート番号を用いて制御を行う。

番号	項目	値	機能
1	WDOG1_PORTID	1	Watch Dog のポート 0 を示す

表 4.1.6.1.1 WATCH DOG ポート番号

番号	項目	型	機能
1	FeedTime	uint32_t	Feed Timeout
2	Debug	uint32_t	Debug mode request
3	Magic	uint32_t	Scratch magic value

表 4.1.6.1.2 WDOG_Init_t 構造体

4.1.6.2 インターフェイス仕様

WATCH DOG を制御するドライバ関数は以下の通りである。wdog_init でウォッチドックタイマーを起動した場合、リセット以外停止することはできない。

関数名	型	引数	機能	備考

wdog_init	ER	ID portid uint16_t wdogtime	指定ポート ID の WATCH DOG ペリフェラルを初期化する wdogtime はタイムアウト時間で秒単位である	
wdog_deinit	ER	ID portid	停止不可（意味をもたいない）	
wdog_reset	ER	ID portid	タイマーのリセットを行う	

表 4.1.6.2.1 WATCH DOG ドライバ関数

4.1.7 UART

UART 用のデバイスドライバは asp/fmp カーネルで実装済のデバイスドライバを使用しているため、ここでは記載しない。

4.2 Standard Driver

スタンダードドライバは拡張ボード（シールド）の標準化により、ドライバ API がデファクトスタンダードとなっているドライバを指す。但し、ペリフェラルの実装は標準化されたドライバ API 以上の機能を持つものが多く、ハードウェアを最大限に利用するのは拡張インターフェイスにて拡張を行う必要がある。

4.2.1 概要

B TOPPERS BASE PLATFORM として、スタンダードドライバとしたのは、以下の３つのペリフェラルである。ADC 以外はいずれもインターフェイス用のペリフェラルであり、接続先にセンサー、LCD、GLCD、SD card、ネットワークハードウェア等の機器の制御用に用いられる。個々のハードウェアのドライバは別途上位に GDIC ドライバを用意しなければならない。

① I2C
② SPI
③ ADC
④ USB

スタンダードドライバは、基本的にポート ID を指定してハンドラを取り出しハンドラを用いてペリフェラルを制御する構成を取る。

4.2.2 SPI

SPI はシリアル・ペリフェラル・インターフェイスの略で、I2C と同様にペリフェラル間の通信規格である。I2C が高速でも 400kbps であるのに比べ SPI は 1Mbps から 20Mbps まで高速転送が可能である。本実装では 8 つの FIFO フレームを使った実装とした。SS(Slave Select または CS)の設定については AUTO モードと、AUTO オフモードが設定可能である。Arduino コネクタ互換用には、AUTO オフモードを推奨する。SPI は４本の信号で通信を行う。スレーブが複数ある場合は、SS(Slave Select)を LOW にしたスレーブに対して通信を行う。そのため、SS 信号はスレーブの数だけ必要となる。また、双方向通信時は MISO と MOSI は同時にデータ通信するので、同時にデータ交換が行われる形となる。

① SCLK クロック信号
② MISO スレーブからのデータ信号
③ MOSI マスタからのデータ信号
④ SS　　スレーブのセレクト信号

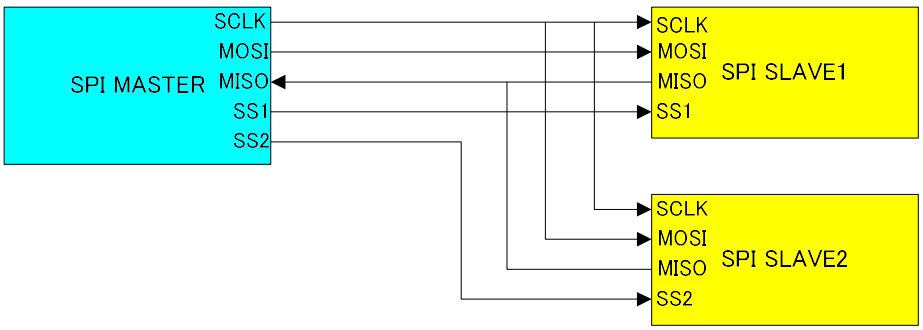

図 4.2.2.1 SPI 接続図

SPI 通信は、クロックの論理（正と負）、クロックに対するデータ設定タイミングにより４つのモードのデータ・タイミングが定義されている。

① モード０：正パルス、前縁ラッチ、後端シフト
② モード１：正パルス、前縁シフト、後端ラッチ
③ モード２：負パルス、前縁ラッチ、後端シフト
④ モード３：負パルス、前縁シフト、後端ラッチ

図 4.2.3.2 はもっとも一般的なモード０の動作タイミングを示す。

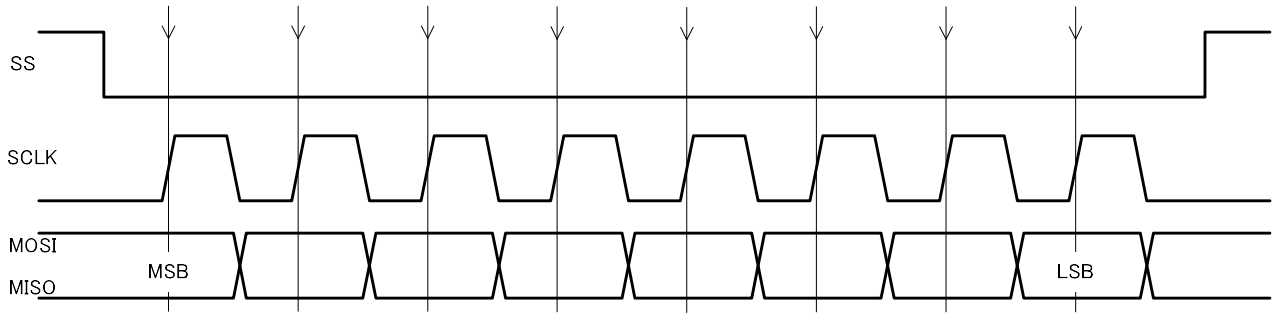

図 4.2.2.2 モード0、正パルス、前縁ラッチ、後端シフトのタイミング図

4.2.2.1 データ仕様

SPI ドライバは初期化用の型として、表 4.2.2.1.1 の SPI コンフィギュレーション型と、ハンドラとして表 4.2.2.1.2 の SPI ハンドラ型を持つ。

番号	項目	型	機能
1	Baudrate	uint32_t	SPI 転送クロック
2	Mode	uint32_t	SPI マスタ、スレーブ等転送モード
3	Direction	uint32_t	SPI 転送方向
4	DataSize	uint32_t	SPI 転送データサイズ
5	CLKPolarity	uint32_t	SPI 転送クロックの極性
6	CLKPhase	uint32_t	SPI 転送クロック位相
7	FrameFormat	uint32_t	SPI フレーム設定
8	DMARxChannel	uint32_t	SPI 受信用 DMA のチャネル番号（-1 で割込み通信）
9	DMATxChannel	uint32_t	SPI 送信用 DMA のチャネル番号（-1 で割込み通信）
10	semid	int	通信用セマフォ ID（0 でセマフォなし）
11	smlock	int	排他制御用セマフォ ID(0 で排他制御なし)

表 4.2.2.1.1 SPI コンフィギュレーション型

semid はセマフォ通信用のセマフォ番号、ゼロで設定なし。このセマフォは割込みとドライバ間の伝達用に使用するため、設定なしの場合、通信遅延が発生する。smlock は、ドライバの排他制御に使用するセマフォ番号を指定する。ゼロの設定で排他制御なしとなる。

番号	項目	型	機能
1	base	uint32_t	SPI ベースアドレス
2	Init	SPI_Init_t	SPI コンフィギュレーション型
3	pTxBuffPtr	uint8_t *	送信データ領域へのポインタ
4	TxXferSize	uint16_t	要求送信サイズ
5	TxXferCount	uint16_t	送信済みサイズ
6	pRxBuffPtr	uint8_t *	受信データ領域へのポインタ
7	RxXferSize	uint16_t	受信要求サイズ
8	RxXferCount	uint16_t	受信済みサイズ
9	hdmatx	DMA_Handle_t *	送信用 DMA ハンドラ
	hdmarx	DMA_Handle_t *	受信用 DMA ハンドラ
10	xmode	uint16_t	転送モード
11	spino	uint16_t	SPI ポート番号
12	status	volatile uint32_t	SPI の状態
13	ErrorCode	volatile uint32_t	SPI エラーコード

表 4.2.2.1.2 SPI ハンドラ型

① Mode
SPI 通信モード

定義	値	内容
SPI_MODE_SLAVE	SPI_SSPCR1_MS	スレーブモード
SPI_MODE_MASTER	0x00000000	マスターモード
SPI_MODE_RESET_ERROR	0x80000000	転送エラー時、状態設定を戻す

表 4.2.2.1.3 Mode 設定値

② Direction
SPI の転送指定。

定義	値	内容
SPI_DIRECTION_2LINES	(0)	2 ライン転送
SPI_DIRECTION_1LINE_RX	(1)	1 ライン受信転送
SPI_DIRECTION_1LINE_TX	(2)	1 ライン送信転送

表 4.2.2.1.4 Direction 設定値

③ DataSize
SPI の転送サイズ指定

定義	値	内容
SPI_DATASIZE_4BIT	SPI_SSPCR0_DSS_4	4 ビット転送
SPI_DATASIZE_5BIT	SPI_SSPCR0_DSS_5	5 ビット転送
SPI_DATASIZE_6BIT	SPI_SSPCR0_DSS_6	6 ビット転送
SPI_DATASIZE_7BIT	SPI_SSPCR0_DSS_7	7 ビット転送
SPI_DATASIZE_8BIT	SPI_SSPCR0_DSS_8	8 ビット転送
SPI_DATASIZE_9BIT	SPI_SSPCR0_DSS_9	9 ビット転送
SPI_DATASIZE_10BIT	SPI_SSPCR0_DSS_10	10 ビット転送
SPI_DATASIZE_11BIT	SPI_SSPCR0_DSS_11	11 ビット転送
SPI_DATASIZE_12BIT	SPI_SSPCR0_DSS_12	12 ビット転送
SPI_DATASIZE_13BIT	SPI_SSPCR0_DSS_13	13 ビット転送
SPI_DATASIZE_14BIT	SPI_SSPCR0_DSS_14	14 ビット転送
SPI_DATASIZE_15BIT	SPI_SSPCR0_DSS_15	15 ビット転送
SPI_DATASIZE_16BIT	SPI_SSPCR0_DSS_16	16 ビット転送

表 4.2.2.1.5 DataSize 設定値

④ CLKPolarity
SPI 転送クロック極性定義。

定義	値	内容
SPI_POLARITY_LOW	0x00000000	極性 LOW
SPI_POLARITY_HIGH	SPI_SSPCR0_SP0	極性 HIGH

表 4.2.2.1.6 CLKPolarity 設定値

⑤ CLKPhase
SPI 転送クロック位相定義。

定義	値	内容
SPI_PHASE_1EDGE	0x00000000	前縁ラッチ
SPI_PHASE_2EDGE	SPI_SSPCR0_SPH	後縁ラッチ

表 4.2.2.1.7 CLKPhase 設定値

⑥ FrameFormat
SPI のフレームフォーマットを設定する。

定義	値	内容

SPI_MOTOROLA_MODE	SPI_SSPCR0_MOTOROLA	モトローラ・フォーマット
SPI_TI_MODE	SPI_SSPCR0_TI	テキサスインスツルメンツ･モード
SPI_MICROWARE_MODE	SPI_SSPCR0_MICROWARE	マイクロウェア・モード

表 4.2.2.1.8 FrameFormat 設定値

⑦ DMARxChannl
SPI の受信 DMA のチャネル番号を設定する。

⑧ DMATxChannel
SPI の送信 DMA のチャネル番号を設定する。

⑨ semid
DMA 転送終了割込みをドライバに伝えるためのセマファ ID。

⑩ semlock
タスク間で、関数をロックするためのセマフォ ID。

4.2.2.2 インターフェイス仕様

SPI を制御するドライバ関数は以下の通りである。SS の設定は、GPIO を使って別途制御しなければならない。TOPPERS BASE PLATFORM(ST)の SPI 設定は、AUTO モードオフなので、SPI 用のサンプルプログラムは、この設定になっている。

関数名	型	引数	機能	備考
spi_init	SPI_Handle_t*	ID portid SPI_Init_t *spii	指定ポート ID の SPI ペリフェラルを初期化し、ハンドラへのポインタを返す	
spi_deinit	ER	SPI_Handle_t* hspi	SPI を未使用状態に戻す	
spi_reset	ER	SPI_Handle_t* hspi	SPI をリセットする	
spi_transmit	ER	SPI_Handle_t* hspi uint32_t slave_select uint8_t *tx_buf uint32_t len	SPI 送信を行う。転送終了まで関数内で待ちとなる。	
spi_receive	ER	SPI_Handle_t* hspi uint32_t slave_select uint8_t *rx_buf uint32_t len	SPI 受信を行う。転送終了まで関数内で待ちとなる。	
spi_transrecv	ER	SPI_Handle_t* hspi uint32_t slave_select uint8_t *tx_buf uint8_t *rx_buf uint32_t len	SPI 送受信を行う。転送終了まで関数内で待ちとなる。	
spi_isr	void	intptr_t exinf	SPI 割込みサービスルーチン	

表 4.2.2.2.1 SPI ドライバ関数

4.2.2.3 フローチャート

SPI ドライバは、FIFO による通信を指定するように設計しています。SPI の初期設定は、spi_init 関数を指定して対象ポート番号と初期設定したコンフィギュレーション構造体のポインタを指定します。

ペリフェラルには受信のみ、送信のみ、送受信があり、それぞれの転送関数を用意しています。転送待ちは各関数内で行われます。実際はほとんどの場合、spi_transsrev ですべての転送を行えます。転送の終了待ちを行う場合、spi_wait 関数を呼び出せば関数内で終了待ちを行います。SPI ハンドラ内の ErrorCode は表 4.2.2.3.1 のように制御中に発生したエラーを格納する。

定義	値	内容
SPI_ERROR_NONE	0x00000000	エラーなし
SPI_ERROR_OVR	0x00000004	オーバーランエラー
SPI_ERROR_TIMEOUT	0x00000020	ステート変更タイムアウト

表 4.2.2.3.1 ErrorCode の内容

図 4.2.2.3.1 に SPI の初期化フローチャートを記載します。SPI 割込み、通信と排他制御用セマフォの静的 API を用いて登録する。

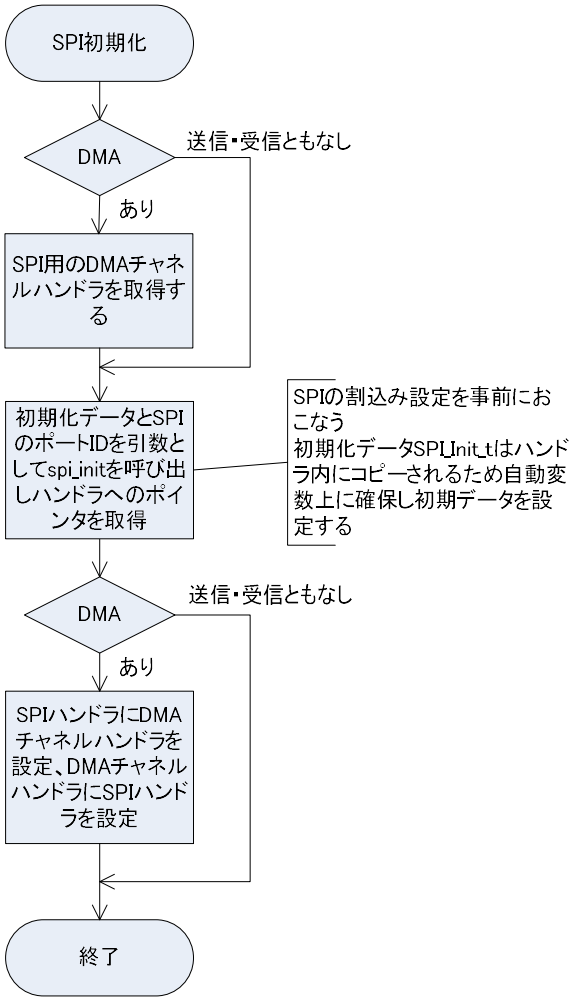

図 4.2.2.3.1 SPI 初期化フローチャート

図 4.2.2.3.2 に SPI の送受信のフローチャートを記載します。複数のスレーブの対応を行う場合は、SS の設定を行わなければならない。送受信の処理は、送信のみの通信や受信のみの通信であっても、spi_tranrecv 関数で処理を代用できる。送信のみの通信でこの関数を使用した場合、受信データとして 0xFF が受信され、受信のみの通信でこの関数を代用した場合、送信データとして 0xFF をパディングした方が安全である。

送受信の場合、送信と同期して受信データが受信領域にセットされる。転送待ちは関数内で行う。戻り値が E_OK 以外はエラーが発生している。エラーの詳細は SPI ハンドラの errorcode にセットされる。

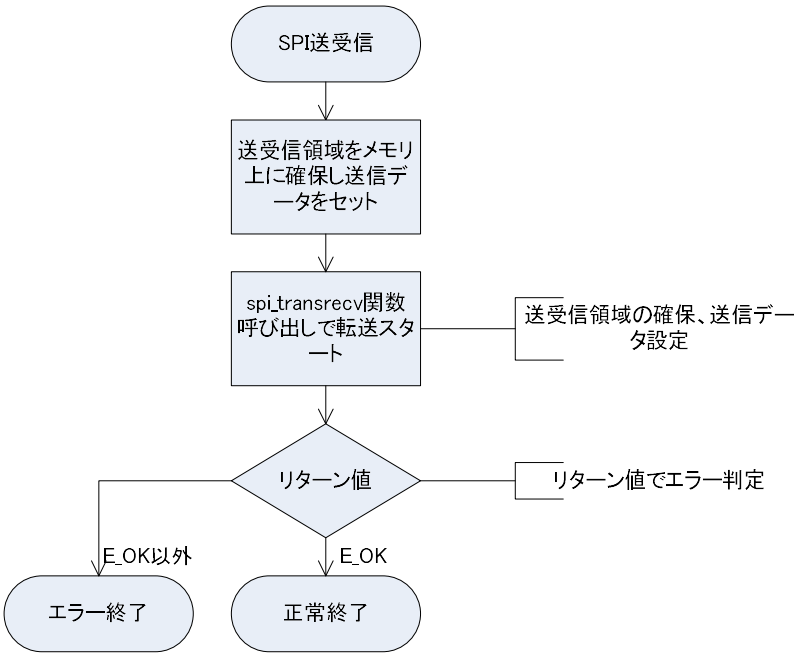

図 4.2.2.3.2 SPI 送受信フローチャート

4.2.3 I2C

I2C（アイ・スクエア・シー）は周辺機との通信用にフィリップス社が開発した低速なシリアルバスである。メインボード側がマスタ、周辺機側がスレーブとなり、スレーブアドレスをキーにデータの送受信を行う。基本的な通信速度は 100kbit/sec の通常モードと 10kbit/sec の低速モードがあるが、基本以上、または、基本以下の速度で通信を行う場合も多い。スレーブアドレスは通常は 7 ビットであるが、拡張として１０ビットのスレーブアドレスも通信可能となっている。

I2C は SCL（クロック）と SDA（データ）の２つの線で通信を行う、周辺機が複数ある場合はこの２つの線を共有する形となる。マスタ側が常に制御権を持っており基本のクロック SCL はマスタ側が設定する。但し、スレーブ側で待ちが必要な場合は、スレーブ側 SCL 信号を Low に落として待ち状態を作る。送信を行う場合は、送信側がクロックに合わせて SDA 上にデータ信号を乗せる。最後の８ビット目で受信側が SDA を Low にした場合は ACK となり、Hi のままならば NACK となる。スレーブアドレス 7bit の一般的なデータ転送を図 4.2.3.1 に示す。

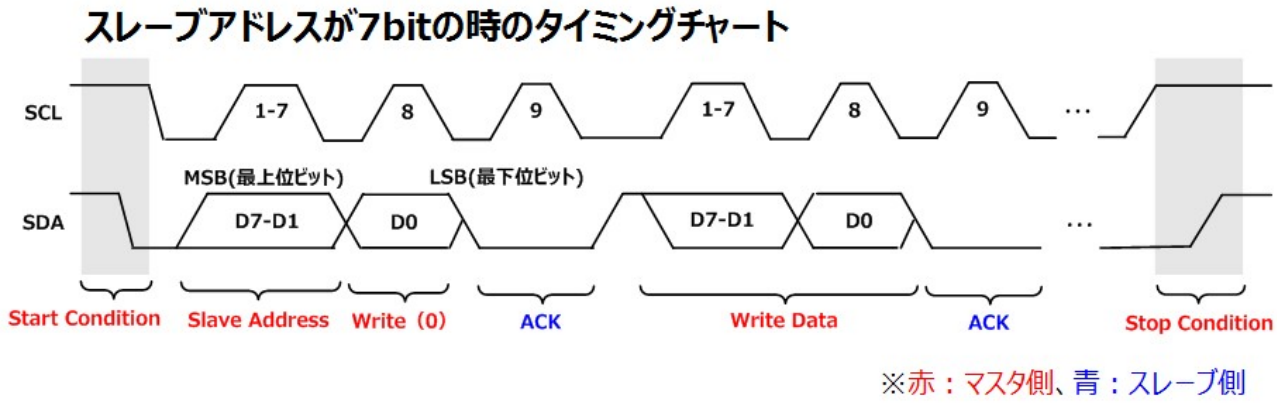

図 4.2.3.1 スレーブアドレス 7bit のデータ転送

4.2.3.1 データ仕様

I2C ドライバは初期化用の型として、表 4.2.3.1.1 の I2C コンフィギュレーション型と、ハンドラとして表 4.2.3.1.2 の I2C ハンドラ型を持つ。SoC に I2C 用の IP はないため、本ドライバは GPIO のソ

フト制御で実装している。SDA のデータ入力が不定のため、送信のみ実行可能となっている。

番号	項目	型	機能
1	Bardrate	uint32_t	I2C のクロック値
2	SpeedMode	uint32_t	I2C コアのスピード設定モード
3	OwnAddress	uint32_t	I2C スレーブの場合のスレーブアドレス
4	AddressingMode	uint32_t	I2C アドレッシングモード
5	GeneralCallMode	uint32_t	I2C ジェネラルコールのアドレッシングモード
6	semid	int	終了割込みをドライバに伝えるためのセマフォ ID
7	smlock	int	排他制御用セマフォ ID(0 で排他制御なし)

表 4.2.3.1.1 I2C コンフィギュレーション型

本ドライバは SCL と SDA に GPIO を指定してソフトウェア制御で I2C 制御を行う設定となっているため、コンフュギュレーション型は、ロック制御を行うセマフォ以外の設定はない。smlock は、ドライバの排他制御に使用するセマフォ番号を指定する。ゼロの設定で排他制御なしとなる。

番号	項目	型	機能
1	base	uint32_t	I2C ベースアドレス
2	Init	I2C_Init_t	I2C コンフィギュレーション型
3	txsize1	uint32_t	I2C ポート ID
4	txsize2	uint32_t	ナノ秒単位のクロック待ち時間
5	rxsize	uint16_t	SCL のピン番号
6	txcount1	uint16_t	SDA のピン番号
7	txcount2	uint16_t	GPIO の PULLUP 設定保存領域
8	rxcount	uint16_t	GPIO の IOF 設定保存領域
9	txbuf1	uint8_t *	
10	txbuf2	uint8_t *	
11	rxbuf	uint8_t *	
12	slave_trans	bool_t	
13	writecallback	void(*)()	送信終了コールバック
14	readcallback	void(*)()	
15	finishcallback	void(*)()	
16	errorcallback	void(*)()	
17	i2cno	uint32_t	I2C ポート番号
18	status	uint32_t	現在の実行状態
19	ErrorCode	uint32_t	I2C エラーコード

表 4.2.3.1.2 I2C ハンドラ型

4.2.3.2 インターフェイス仕様

I2C を制御するドライバ関数は以下の通りである。

関数名	型	引数	機能	備考
i2c_init	I2C_Handler*	ID portid I2C_Init_t *ii2c	指定ポート ID の I2C ペリフェラルを初期化し、ハンドラへのポインタを返す	
i2c_deinit	ER	I2C_Handler* hi2c	I2C を未使用状態に戻す	
i2c_slaveactive	ER	I2C_Handler* hi2c	スレーブモードを設定する	
i2c_slavewrite	ER	I2C_Handler* hi2c	スレーブモードのデータ送信、コールバック関数で送信	
i2c_slaveread	ER	I2C_Handler* hi2c uint8_t data	スレーブモードのデータ受信、コールバック関数で受信	

i2c_masterwrite (i2c_memwrite)	ER	I2C_Handler* hi2c uint16_t DevAddr uint16_t MemAddr uint16_t MemAddSize uint8_t *pData uint16_t Size	マスターモードのデータ送信、MemAddSize をゼロにするとアドレス設定を行わない	
i2c_masterread (i2c_memread)	ER	I2C_Handler* hi2c uint16_t DevAddr uint16_t MemAddr uint16_t MemAddSize uint8_t *pData uint16_t Size	マスターモードのデータ受信、MemAddSize をゼロにするとアドレス設定を行わない	

表 4.2.3.2.1 I2C ドライバ関数

4.2.3.3 フローチャート

I2C の初期設定は、i2c_init 関数を指定して対象ポート番号と初期設定したコンフィギュレーション構造体のポインタを指定します。本実装ではマスタ機能のみの実装となっています。基本的に、このドライバではタスク上でデータの送受信を行います。また、スタート、ストップ、ACK 処理はソフトウェア処理にて処理しますので、特別にデータを管理する項目を用意する必要はありません。

図 4.2.3.3.1 に初期化のフローチャートを示します。i2c_init で取得した I2C ハンドラへのポインタは以後、I2C の制御用に使用する。

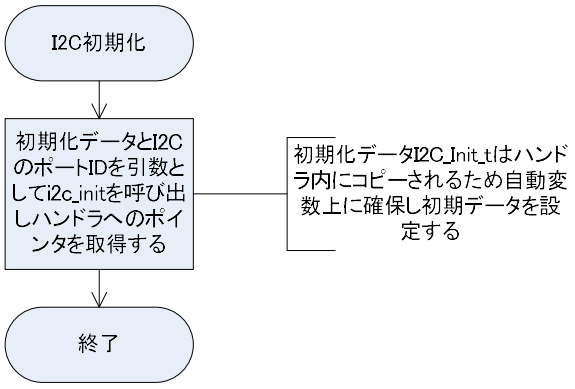

図 4.2.3.3.1 初期化フローチャート

図 4.2.3.3.2 にマスタのデータ送信のフローチャート、図 4.2.3.3.3 に受信のフローチャートを示します。送信ならば i2c_masterwrite(i2c_memwrite)、受信ならば i2c_masterread(i2c_memread)関数を呼び出せば指定サイズの送受信が行えます。送受信の結果は、関数の戻り値確認できます。E_OK 以外の戻り値の場合エラー処理を行ってください。ペリフェラルには、データアドレスを持つもの（EEPROM や RTC など）があり、送受信のとき、データアドレスの設定を行う必要があります。この場合、MemAddr でデータアドレス、MemAddSize でデータアドレスのバイトサイズを指定します。データアドレスの設定を行わない場合は、MemAddrSize をゼロして、送受信関数を呼び出します。

I2C ペリフェラルを終了させたい場合は、引数として I2C ハンドラへのポインタを指定して i2c_deinit 関数を呼び出せば、ペリフェラルとハンドラは未使用状態に戻ります。ErrorCode はアプリケーションのプログラム互換用に項目を用意しており、エラーコードは設定されません。

定義	値	内容
I2C_ERROR_NONE	0x00000000	エラーなし

表 4.2.3.3.1 ErrorCode の内容

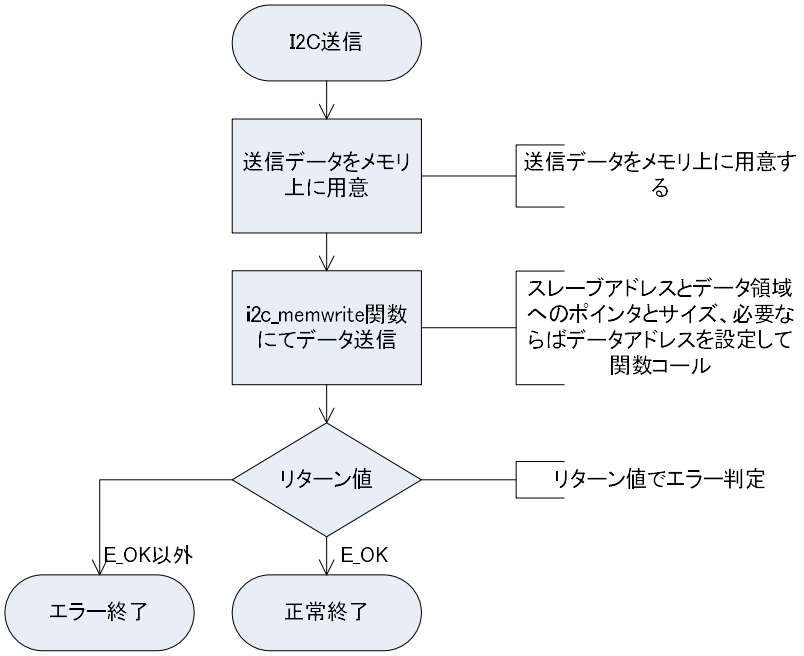

図 4.2.3.3.2 I2C 送信フローチャート

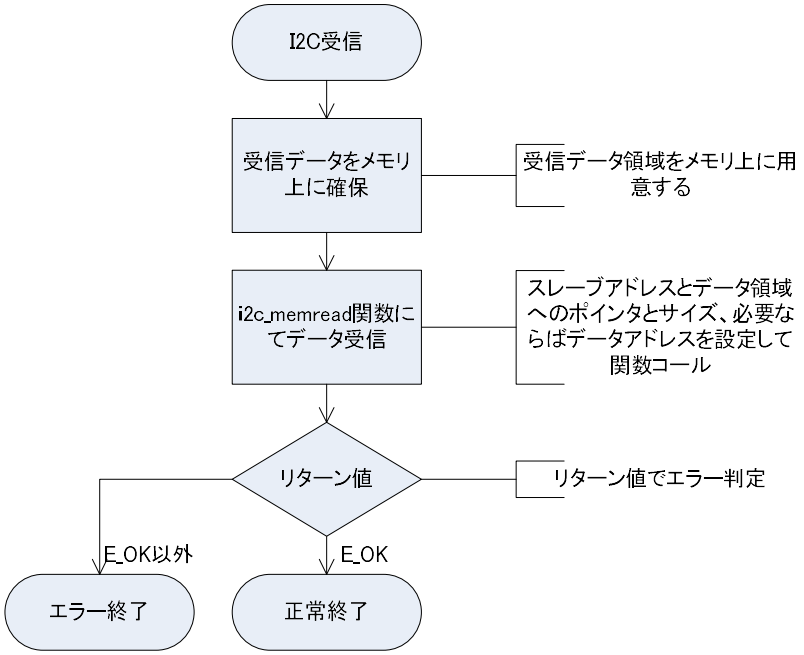

図 4.2.3.3.3 I2C 受信フローチャート

4.2.4 ADC

ADC(アナログ・データ・コントローラ)は、アナログ入力データをデジタル・データに変換するドライバである。

4.2.4.1 データ仕様

ADC ドライバは初期化用の型として、表 4.2.4.1.1 の ADC コンフィギュレーション型と、ハンドラとして表 4.2.4.1.4 の ADC ハンドラ型を持つ。ADC の設定後、チャネルの設定用に表 4.2.4.1.2 の ADC チャネル設定型がある。

番号	項目	型	機能

1	ClockPrescaler	uint32_t	ADC クロックプリスケーラ値
2	FIFOMode	uint32_t	ADC のリゾリューション
3	ConvertCount	uint32_t	ADC 結果データアライン設定
4	RxDMAChannel	int32_t	ADC の DMA 受信チャネル番号(-1 で DMA を使用しない)

表 4.2.4.1.1 ADC コンフィギュレーション型

番号	項目	型	機能
1	Channel	uint32_t	ADC チャンネル番号
2	GpioPin	uint32_t	ADC-GPIO ピン番号

表 4.2.4.1.2 ADC チャネル設定型

番号	項目	型	機能
1	base	uint32_t	ADC ペリフェラルのベースアドレス
2	Init	ADC_Init_t	ADC 初期化設定パラメータ
3	hdmarx	DMA_Handle_t*	ADC 受信 DMA のハンドラ
4	RxBuf	uint32_t*	コンバージョン受信データ領域へのポインタ
5	Count	int32_t	ADC コンバージョンカウンタ
6	xfercallback	void (*)()	転送完了時のコールバック関数
7	errorcallback	void (*)()	ADC エラー発生時のコールバック関数
8	dstaddr	uint32_t	ADC デスティネーションアドレス
9	translen	uint32_t	ADC 転送数
10	status	volatile uint32_t	ADC の状態
11	ErrorCode	volatile uint32_t	ADC エラーコード

表 4.2.4.1.3 ADC ハンドラ型

① ClockPrescaler
ADC クロック分周設定。0 から 65535 までの分周値

② FIFOMode
ADCFIFO モード設定。

定義	値	内容
ADC_VALUE_SHIFT	ADC_FCS_SHIFT	設定により結果値を右シフトする
ADC_ERROR_CONVERT	ADC_FCS_ERR	エラーの場合、結果を FIFO に置く

表 4.2.4.1.4 FIFOmode 設定値

③ ConvertCount
ADC 変換の変換数を指定する

④ RxDMAChannel
ADC 変換結果転送に DMA を使用する場合、チャネル番号をド設定。

⑤ Channel
ADC のチャネル番号を設定（0～3)。

⑥ GpioPin
ADC チャネルの GPIO のピン番号を設定。

チャネル番号定義	GPIO ピン	
0	26	
1	27	
2	28	

表 4.2.4.1.5 チャネルピン番号

4.2.4.2 インターフェイス仕様

ADC を制御するドライバ関数は以下の通りである。入力ピンの設定は、GPIO を使って別途制御しなければならない。

関数名	型	引数	機能	備考
adc_init	ADC_Handler*	ID portid ADC_Init_t *pini	指定ポート ID の ADC ペリフェラルを初期化し、ハンドラへのポインタを返す	
adc_deinit	ER	ADC_Handler* hadc	ADC を未使用状態に戻す	
adc_start_int	ER	ADC_Handler* hadc	ADC 割込みモード開始	
adc_end_int	ER	ADC_Handler* hadc	ADC 割込みモード終了	
adc_start_dma	ER	ADC_Handler* hadc uint32_t *pdata uint32_t length	ADC DMA モード開始	
adc_end_dma	ER	ADC_Handler* hadc	ADC DMA モード終了	
adc_setupchannel	ER	ADC_Handler* hadc ADC_ChannelConf_t* sConfig	ADC チャネル設定	
adc_set_int_buffer	ER	ADC_Handler* hadc uint32_t *pdata	ADC 割込み用のバッファポインタをセットする	
adc_getvalue	int32_t	ADC_Handler* hadc	ADC の現在の変換結果を取り込む	
adc_int_handler	void	void	ADC 割込みハンドラ	

表 4.2.4.2.1 ADC ドライバ関数

4.2.4.3 フローチャート

実行手順に影響を与えるモードは、以下の３つである。

(1) FIFOMode

変換データのフォーマットを設定する。

(2) ConvertCount

連続変換の場合、AD 変換回数を指定する。

(3) RxDmaChannel

変換結果の転送に DMA を使用する場合、DMA のチャネル番号を指定する。

変換の手順は、サンプルプログラム ADC を参照のこと。EXEC_MODE の設定で以下の変換を行う。

① EXEC_MODE=0 DMA CIRCULAR CONTINUOUS
② EXEC_MODE=1 DMA CIRCULAR CONTINUOUS FREE RUN
③ EXEC_MODE=2 割込み CONTINUOUS COUNT 設定
④ EXEC_MODE=3 割込み NOT CONTINUOUS

4.2.5 USB

USB モジュールは、ホスト、デバイスの USB 管理を行うドライバである。USB のメインコントロールレジスタの HOST_NDEVICE 設定により、USB ホスト、デバイスを選択が可能である。USB ホストとして動作させるためには USB ホストクラスドライバ、USB デバイスとして動作させるためには USB デバイスライブラリを上位のモジュールとして用意する必要がある。

4.2.5.1 データ仕様

USB ホストドライバと USB デバイスドライバで管理用のデータ定義が異なる。USB デバイス用の

型として、表 4.2.7.1.1～3 の USB デバイス初期設定定義型、USB デバイスエンドポイント定義型、USB デバイスハンドラを定義する。USB ホスト用の型として表 4.2.7.1.4～6 の USB ホスト初期設定定義型 USB ホストエンドポイント定義、USB ホストハンドラ型を持つ。

番号	項目	型	機能
1	dev_endpoints	uint32_t	デバイス用のエンドポイントの数(1-126)
2	speed	uint32_t	USB コアスピード
3	sof_enable	uint32_t	デバイスモードでの SOF 割込みの有効無効設定

表 4.2.7.1.1 USB デバイス初期設定定義型

番号	項目	型	機能
1	num	uint8_t	エンドポイント番号(0-15)
2	is_in	uint8_t：1	エンドポイントのディレクション(1:IN 0:OUT)
3	is_stall	uint8_t：1	スティール状態(1:スティール状態)
4	type	uint8_t：2	通信タイプ
5	next_pid	uint8_t：1	エンドポイントの次の PID
6	endpoint_control	uint32_t*	エンドポイント制御領域へのポインタ
7	buffer_control	uint32_t*	エンドポイントバッファ制御領域へのポインタ
8	data_buffer	uint8_t*	データ領域へのポインタ
9	maxpacket	uint32_t	最大パケットサイズ
10	xfer_buff	uint8_t*	転送バッファポインタ
11	xfer_len	uint32_t	現在転送サイズ
12	request_count	uint32_t	指定の転送サイズ

表 4.2.7.1.2 USB デバイスエンドポイント定義型

番号	項目	型	機能
1	base	uint32_t	USB デバイスレジスタベースアドレス
2	dpram	USB_DEV_Dpram_t*	USB デバイス PMA ベースアドレス
3	Init	USB_DEV_Init_t	USB デバイス初期設定定義
4	IN_ep	USB_DEV_EPTypedef	エンドポイント IN の定義領域
5	OUT_ep	USB_DEV_EPTypedef	エンドポイント OUT の定義領域
6	*Setup	volatile uint8_t*	Setup パケットバッファ
7	usb_address	volatile uint8_t	USB アドレス
8	usb_suspend	uint8_t	USB サスペンドモード
9	ep0_out_length	uint8_t	受信した EP0.OUT の長さ
10	usbno	uint8_t	USB デバイス番号（常に 0）
11	devsetupstagecallback	void(*)()	デバイスセットアップステージコールバック関数
12	devdataoutstagecallback	void(*)()	デバイスデータアウトステージコールバック関数
13	devdatainstagecallback	void(*)()	デバイス SOF 割込みコールバック関数
14	devsofcallback	void(*)()	デバイスデータインステージコールバック関数
15	devresetcallback	void(*)()	デバイスリセットコールバック関数
16	devsuspendcallback	void(*)()	デバイスサスペンドコールバック関数
17	devresumecallback	void(*)()	デバイスレジュームコールバック関数
18	devconnectcallback	void(*)()	デバイスコネクトコールバック関数
19	devdisconnectcallback	void(*)()	デバイスディスコネクトコールバック関数
	devlpmcallback	void(*)()	デバイス LPM コールバック関数
	pDev	void *	上位デバイスハンドラ格納領域

表 4.2.7.1.3 USB デバイスハンドラ定義型

番号	項目	型	機能
1	host_channels	uint32_t	ホストチャネルの数(未使用)
2	sof_enable	uint32_t	デバイスモードでの SOF 割込みの有効無効設定

表 4.2.7.1.4 USB ホスト初期設定定義型

番号	項目	型	機能
1	dev_addr	uint8_t	USB デバイスアドレス(1-255)
2	ch_num	uint8_t	HOST チャネル番号(1-10)
3	ep_num	uint8_t	エンドポイント番号(1-15)
4	ep_is_in	uint8_t	エンドポイントのディレクション
5	speed	uint8_t	USB Host のスピード
6	do_ping	uint8_t	HS モードの PING プロトコルの有無
7	process_ping	uint8_t	PING プロトコル実行中フラグ(1 で実行中)
8	ep_type	uint8_t	エンドポイントタイプ(0-3)
8	max_packet	uint16_t	最大パケットサイズ(64B 以内)
9	data_pid	uint8_t	初期設定データの PID
10	xfer_buff	uint8_t *	送受信バッファへのポインタ
11	xfer_len	uint16_t	現在転送サイズ
12	xfer_count	uint16_t	指定の転送サイズ
13	remain_len	uint16_t	残り転送サイズ
14	toggle_in	uint8_t	IN 転送のトグルフラグ
15	toggle_out	uint8_t	OUT 転送のトグルフラグ
16	endpoint_control	uint32_t *	エンドポイント制御領域へのポインタ
17	buffer_control	uint32_t *	エンドポイントバッファ制御領域へのポインタ
18	data_buffer	uint8_t *	データ領域へのポインタ
19	next_pid	uint8_t	次の PID
20	urb_state	volatile uint8_t	URB ステート
21	state	volatile uint8_t	HOST チャネルのステート

表 4.2.7.1.5 USB ホストクラス定義型

番号	項目	型	機能
1	base	uint32_t	USB-OTG ペリフェラルのベースアドレス
2	dpram	USB_HOST_Dpram_t*	USB ホスト PMA 領域へのポインタ
3	Init	USB_HOST_Init_t	USB ホスト初期化設定パラメータ
4	hc[15]	USB_HOST_HCTypeDef	ホストチャネル領域
5	enabled	uint8_t	デバイスイネーブル
6	connected	uint8_t	デバイス接続中
7	intep0_ch	uint8_t	EP0 割込みチャネル番号
8	usbno	uint8_t	USB デバイス番号
9	hostsofcallback	void (*)()	ホスト用 SOF 割込みコールバック関数
10	hostconnectcallback	void (*)()	ホストコネクト時コールバック関数
11	hostdisconnectcallback	void (*)()	ホストディスコネクト時コールバック関数
12	hostchangeurbcallback	void (*)()	ホスト URB 変更時コールバック関数
13	pHost	void *	上位ホストハンドラ格納領域

表 4.2.7.1.6 USB ホストハンドラ型

4.2.7.2 インターフェイス仕様

USB デバイスを制御するドライバ関数を表 4.2.7.2.1 に、USB ホストを制御するドライバ関数を表 4.2.7.2.2 に記載する。

関数名	型	引数	機能	備考
usbd_init	USB_DEV_Handler*	ID portid USB_DEV_Init_t *pini	指定ポート ID の USB デバイスペリフェラルを初期化	
usbd_deinit	ER	USB_DEV_Handler* husb	USB デバイスモジュールの無効化	
usbd_devconnect	ER	USB_DEV_Handler* husb	デバイス接続設定	
usbd_devdisconnect	ER	USB_DEV_Handler* husb	デバイス切断設定	
usbd_setDevAddress	ER	USB_DEV_Handler* husb uint8_t address	確定したデバイスアドレスを設定	
usbd_activateEndpoint	ER	USB_DEV_Handler* husb USB_DEV_EPTypeDef *ep	エンドポイントを有効化設定	
usbd_deactivateEndpoint	ER	USB_DEV_Handler* husb USB_DEV_EPTypeDef*ep	エンドポイントを無効化	
usbd_epstartreceive	ER	USB_DEV_Handler* husb USB_DEV_EPTypeDef*ep	エンドポイント受信を開始する	
usbd_epstartsend	ER	USB_DEV_Handler* husb USB_DEV_EPTypeDef*ep	エンドポイント送信を開始する	
usbd_epsetStall	ER	USB_DEV_Handler* husb USB_DEV_EPTypeDef*ep	エンドポイントをスティール状態に移行する	
usbd_epclearStall	ER	USB_DEV_Handler* husb USB_DEV_EPTypeDef*ep	エンドポイントのスティール状態を解除する	
usbd_setupepdata	ER	USB_DEV_Handler* husb uint16_t ep_addr uint8_t ch_num	エンドポイントのデータ領域を設定する	
usb_isr	void	intptr_t exinf	USB 割込みサービスルーチン	

表 4.2.7.2.1 USB デバイスドライバ関数

USB OTG を制御するドライバ関数は以下の通りである。

関数名	型	引数	機能	備考
usbh_init	USB_OTG_Handler*	ID portid USB_HOST_Init_t *pini	指定ポート ID の USB ホストペリフェラルを初期化	
usbh_deinit	ER	USB_HOST_Handler* husb	USB ホストモジュールの無効化	
usbh_coreinit	ER	USB_HOST_Handler* husb	USB ホストコアの初期化	
usbh_setupint	ER	USB_HOST_Handler* husb	USB ホスト割込み設定	
usbh_enableglobalint	ER	USB_HOST_Handler* husb	未処理	
usbh_hostinit	ER	USB_HOST_Handler* husb	USB ホスト初期化する	
usbh_starthost	ER	USB_HOST_Handler* husb	未処理	
usbh_resethost	ER	USB_HOST_Handler* husb	未処理	
usbh_drivevbus	ER	USB_HOST_Handler* husb uint8_t state	ホスト用 VBUS のオンオフ設定を行う	
usbh_gethostspeed	uint32_t	USB_HOST_Handler* husb	ホストのコアスピードを取り出す	
usbh_getcurrentspeed	uint32_t	USB_HOST_Handler* husb	常に 0	
usbh_hc_init	ER	USB_HOST_Handler* husb uint8_t chnum uint8_t epnum	ホストチャネルの初期化	

usbh_hc_startxfer	ER	USB_HOST_Handler* husb uint8_t chnum	ホストチャネルの送信要求	
usbh_hc_halt	ER	USB_HOST_Handler* husb uint8_t chnum	ホストチャネルを HALT 状態にする	
usbh_stophost	ER	USB_HOST_Handler* husb	ホストコア停止	
usb_isr	void	intptr_t exinf	USB 割込みサービスルーチン	

表 4.2.7.2.2 USB ホストドライバ関数

4.2.7.3 フローチャート

USB モジュールはホスト機能とデバイス機能をもつペリフェラルである。ホスト機能をデバイスの接続を待って以下の機能を実行する。

(1) ホスト処理

サンプルプログラム USBH を参照のこと。

(2) デバイス処理

サンプルプログラム USBD を参照のこと。

5 タスクモニタ

本章では、標準入出力機能付きのタスクモニタの仕様に関して記載する。

5.1 概要

タスクモニタはタスク上で動作し、デバッグ用のコマンドを用いてプラットフォーム部の機能確認やテストを行う。デバックコマンドは設定によりコマンド追加が可能である。これによりアプリケーションでも、アプリケーション用デバッグコマンドを追加することができる。タスクモニタの入出力は標準入出力に対して行う、デフォルトの標準入出力は FMP カーネルにて実装されているシリアルデバイスであるが、入出力の切り替えにより、telnet の端末等に切り替えが可能である。

5.2 標準入出力

タスクモニタの入出力は標準入出力に対して行う。標準入出力は FILE 型を定義し、入力、出力、エラーの 3 つの FILE へのポインタを以下の名称で定義することで実現する。

① stdin
② stdout
③ stderr

FILE 型は表 5.2.1 の構成となる。FILE 型は fread や fwrite でファイルにアクセスする場合のハンドラとして使用される。

番号	項目	型	機能
1	_flags	int	ファイル用フラグ
2	_file	int	ファイル番号
3	_func_in	int	1byte 入力コールバック関数
4	_func_ins	int	n bytes 入力コールバック関数
5	_func_out	void	1byte 出力コールバック関数
6	_func_outs	int	n bytes 出力コールバック関数
7	_func_flush	int	データフラッシュコールバック関数
8	_dev	void *	デバイス構造体へのポインタ

表 5.2.1 FILE 型

標準入出力では、以下の関数をサポートする。

関数名	型	引数	機能	備考
fgetc	int	FILE *fp	ファイルから 1byte 読み込み	
fgets	int	char *c FILE *fp	ファイルから文字列読み込み	
fputc	int	int c FILE *fp	ファイルに 1byte 書き込み	
fputs	int	const char *str FILE *	ファイルに文字列書き込み	
putchar	int	int c	1byte 書き込み	
puts	int	const char *str	文字列を標準出力に書き込み	
printf	int	const char const …	標準出力へのプリント	結果は項目数
sprintf	int	char *c const char const …	バッファへプリント	結果は項目数
scanf	int	const char const …	標準入力からスキャン	結果は項目数
sscanf	int	char *c const char const …	スキャンしバッファにセット	結果は項目数
fflush	int	FILE *fp	ファイルのフラッシュ	

fread	size_t	void *buf size_t len size_t num FILE *fp	ファイルからデータ読み込み	結果は num 数
fwrite	size_t	const void *buf size_t len size_t num FILE *fp	ファイルへデータ書き込み	結果は num 数
fprintf	int	FILE *fp const char *const …	ファイルへプリント	結果は項目数
putc	int	int c FILE *fp	fputc と同様	
getchar	int		標準入力から 1byte 読み込み	
getc	int	FILE *fp	fgetc と同様	

表 5.2.2 サポートしている標準入出力関数

5.3 標準デバッグコマンド

タスクモニタは、標準のデバッグコマンドとして以下のコマンドをサポートする。タスクモニタのデバッグコマンドは第１（カテゴリ)、第２の２つのコマンドで機能を指定する形をとる。また、コマンドを設定する場合、最初の１文字以降を省略可能である。省略名で同一のコマンドがある場合、はじめにディスパッチするコマンドが選択される。

第１コマンド	第２コマンド	引数	機能
DISPLAY	BYTE	start address[hex]	バイト単位でメモリ DUMP する
	HALF	start address[hex]	２バイト単位でメモリ DUMP する
	WORD	start address[hex]	４バイト単位でメモリ DUMP する
	TASK	-	タスクの状態を表示する
	REGISTER	-	CPU レジスタの内容を表示する
SET	BYTE	set address[hex]	バイト単位で、メモリ内容を変更する
	HALF	set address[hex]	２バイト単位で、メモリ内容を変更する
	WORD	set address[hex]	４バイト単位で、メモリ内容を変更する
	COMMAND	mode[1 or 2]	デフォルト 2、1 の場合最初の 1 文字のみ比較
	SERIAL	portno	標準入出力のシリアルポート番号を変更
	TASK	taskid	TASK コマンドの対象タスクを指定する
TASK	ACTIVATE	-	タスクの起動要求(act_tsk)
	TERMINATE	-	タスクを終了する(ter_tsk)
	SUSPEND	-	タスクの待ち要求(sus_tsk)
	RESUME	-	タスクの待ち再開(rsm_tsk)
	RELEASE	-	タスクの待ち解除(rel_wai)
	WAKEUP	-	タスクの起床(wup_tsk)
	PRIORITY	priority	タスクの優先度を変更する
LOG	MODE	[logmask][lowmask]	syslog の表示モードを変更する
	TASK	[time]	タスクの実行状態表示(指定が必要)
	PORT	[no][logno][portaddress]	ポートアクセスログ
HELP	Arg1		コマンドヘルプ

表 5.3.1 標準デバッグコマンド

DEVICE ドライバをサポートした場合、以下のコマンドを追加でサポートする。

第１コマンド	第２コマンド	引数	機能
DEV	CLK	0～9	指定のデバイスの現在のクロックを表示
	VER	-	バージョンを表示、PICO W の場合、cyw43 のバージョン等を表示

	RDT	year month day	日にちを設定、または、表示
	RTM	hour min sec	時間を設定、または、表示
	RAL	count	アラームカウントを設定
	RST	-	ソフトリセットする
	LED	number on/off	LED を点灯消灯
	DIP	number on/off	ソフト DIP SW のオンオフ
	CUP	command	1 文字を get_cup_commnad()に送る

表 5.3.2 DEVICE デバッグコマンド

ファイルライブラリが追加された場合、以下のコマンドを追加でサポートする。

第 1 コマンド	第 2 コマンド	引数	機能
VOLUME	FORMAT	drive	ドライブのフォーマット(未サポート)
	DIR	path	ディレクトリの表示
	MKDIR	path	ディレクトリの作成
	RMDIR	path	ディレクトリの消去
	ERASE	path	ファイルの消去

表 5.3.3 ファイルデバッグコマンド

5.4 デバッグコマンド拡張

タスクモニタは、コマンドを拡張する機能を持つ。コマンドの拡張は第 1 （カテゴリ）コマンド単位で追加される。

5.4.1 データ仕様

コマンド追加には 2 つの型を使用する。COMMAND_INFO 型は第 2 コマンドの設定を行い、COMMAND_LINK 型は、複数の COMMAND_INFO 型をまとめて登録カテゴリを指定する。COMMAND_LINK 型の pcnext はデバッグコマンドのリンクに使用する、そのため、COMMAND_LINK は値付きの変数で作成しなければならない。

番号	項目	型	機能
1	command	const char *	第 2 コマンド名
2	func	int_t (*)()	第 2 コマンド関数へのポインタ

表 5.4.1.1 COMMAND_INFO 型

番号	項目	型	機能
1	pcnext	COMMAND_LINK *	COMMAND_LINK のチェーン用
2	num_command	int	第 2 コマンドの数
3	command	const char *	第 1 （カテゴリ）コマンド名
4	func	int_t (*)()	カテゴリコマンドの実行関数（通常は NULL）
5	help	const char *	カテゴリの HELP メッセージ
6	pcinfo	COMMAND_INFO *	COMMAND_INFO の配列へのポインタ

表 5.4.1.2 COMMAND_LINK 型

5.4.2 インターフェイス仕様

デバッグコマンドの追加は、COMMAND_LINK のインスタンスへのポインタを引数に以下の関数コールにて追加される。

関数名	型	引数	機能	備考
setup_command	int	COMMAND_LINK *	コマンドカテゴリを追加する	

表 5.4.2.1 デバッグコマンド追加定関数

6　API 層

API 層はアプリケーションに対して標準的なインターフェイスを提供する層である。この層はハードウェアや RTOS の仕様に影響されず、一意のインターフェイスを提供することにより、アプリケーションを汎用的に作成することができる。また、C 言語の規約や POSIX のように汎用的な API に準拠すれば、LINUX 等のオープンソースのライブラリを未修整で使用することができる。

6.1　概要

TOPPERS BASE PLATFORM でサポートするストレージ機能で使用するファイルシステムと時刻の管理を行う時間管理の２つの API について説明を行う。

FE310-G000 環境では、使用可能な RAM サイズが小さくファイルシステムは動作しない。

6.2　ファイルシステム

ファイルシステムは TOPPERS BASE PLATFORM で提供するストレージ機能である。ファイルシステムは以下の３つのモジュールで構成される。

① ファイルライブラリ
　C 言語標準のファイル関数をサポートするライブラリ
② ストレージデバイスマネージャー
　ストレージデバイスの管理モジュール
③ FATFs
　赤松武史氏が開発し、フリーソフトウェアとして公開されている、FAT 仕様準拠のローカルファイルシステム

TOPPERS BASE PLATFORM アプリケーション、ハードウェアを除く部分を供給している。
内容は、TOPPERS BASE PLATFORM(ST)を参照してください。

6.4　UI ミドルウェア

UI ミドルウェアとして、グラフィック LCD に対して、図形描画、文字描画を行うインターフェイスを追加する。図形描画の API は GDIC/adafuit_st7735 の描画関数を使用する。文字描画は、TOPPERS ECHONET WG で作成した東雲フォントの文字環境に対応できるように設計した。内容は、TOPPERS BASE PLATFORM(ST)を参照してください

6.5　LWIP ミドルウェア

TCP/IP スタック lwip の拡張手順について記載する。現状 lwip は以下のバージョンに対応している。RTOS 上で実行するには問題があるソースに関しては PATCH を用意している。

・lwip-2.1.3
　3 つのパッチ
・contrib-2.1.0
　3 つのパッチ

ASP/FMP カーネル用のドライバを contrib-2.1.0/ports/toppers に追加している。WIFI ドライバは lwip の ETHERNET を gdic の wifi(cyw43)ドライバにて接続を行う。

lwip のポーティング方法は、TOPPERS BASE PLATFORM(CV)を参照してください。

6.6　RTOS 隠蔽機能

pdic ディレクトリの直下に base_platform.h インクルードファイルを置く、このファイルは TOPPERS BASE PLATFORM 内の以下のモジュールから参照される。このファイル内に RTOS との API のマクロ化定義を行い。マクロ定義を書き換えることにより、種々の RTOS の API に対応することができる。このマクロを参照するディレクトリは 7.1 共通部に記載。内容は、TOPPERS BASE PLATFORM(ST)を参照してください

7　ファイルの構成

TOPPERS BASE PLATFORM のソースファイル構造について記載する。共通部はファイルシステムやタスクモニタ等共通となる部分について記載する。BASE PLATFORM のソフトウェア部品は asp のベースディレクトリ上に配置する。

7.1　共通部

共通部のディレクトリ構成を表 7.1.1 に示す。

ディレクトリ	内容	備考
files	ファイルシステムのソースとインクルードファイル	RTOS 隠蔽化対応
monitor	タスクモニタと標準入手力のソースとインクルードファイル	
gdic	GDIC ドライバ	RTOS 隠蔽化対応
lwip	lwIP V2.1.3 のソースファイル置き場所	
pdic	PDIC ドライバ、RISC/V ボード依存ドライバを持つ	RTOS 隠蔽化対応
syssvc	malloc,calloc,free 関数	
ui	GLCD に文字、グラフィックデータ表示	RTOS 隠蔽化対応
usb	USB ホスト、デバイスのミドルウェア	RTOS 隠蔽化対応

表 7.1.1 共通部ディレクトリ

pdic の直下に RTOS 隠蔽用のインクルードファイル base_platform.h を置く。

7.2　PDIC(Base/Standard)ドライバ

RP2040 用 Base/Standard 用ドライバ部は pdic/fe310 にソースファイルがある。

ファイル	内容	備考
device.c	GPIO,DMA,WDOG,RTC,LED,SW ドライバ・ソースファイル	Base
device.cfg	LED,SW の RTOS リソースファイル	Base
device.h	GPIO,DMA,LED,SW ドライバ・インクルードファイル	Base
adc.c	ADC ドライバ・ソースファイル	
adc.h	ADC ドライバ・インクルードファイル	
i2c.c	I2C ドライバ・ソースファイル	
i2c.h	I2C ドライバ・インクルードファイル	
spi.c	SPI ドライバ・ソースファイル	
spi.h	SPI ドライバ・インクルードファイル	
pinmode.c	Arduino コネクタピン配列関数変換プログラム・ソースファイル	
pinmode.h	Arduino コネクタピン配列関数変換プログラム・インクルードファイル	
pio_spi.c	Cyw43 用 SPI ドライバ・ソースファイル	PICO W 用
pio_spi.h	Cyw43 用 SPI ドライバ・インクルードファイル	PICO W 用
pwm.c	PWM ドライバ・ソースファイル	
pwm.h	PWM ドライバ・インクルードファイル	
usb_device.c	USB デバイスドライバ・ソースファイル	
usb_device.c	USB デバイスドライバ・インクルードファイル	
usb_host.c	USB ホストドライバ・ソースファイル	
usb_host.h	USB ホストドライバ・インクルードファイル	
pinmode.h	Arduino の GPIO ピン設定・ソースファイル	
pinmode.c	Arduino の GPIO ピン設定・インクルードファイル	
rp2040.h	RP2040 デバイスアドレス定義ファイル	

表 7.2.1 PICO ドライバファイル

7.3　GDIC ドライバ

ディレクトリ gdic 以下に GDIC ドライバを持つ。GDIC ドライバは PDIC に依存性し、デバイスに依存した機能を提供する。

ディレクトリ	内容	備考

usb_device	usb_device 上に位置し、USB ミドルウェアにてデバイス機能を提供する	
usb_host	usb_host 上に位置し、USB ミドルウェアにてホスト機能を提供する	PICO(W)用
spi_driver	SPI インターフェイスの SD カード用ドライバ、ファイルシステムに SD カードドライバを提供する	
adafruit_st7789	SPI インターフェイスの Adafruit 1.54”LCD に対して、グラフィック API を提供する	
aqm1284_st7565	SPI インターフェイスの AQM1284 LCD に対して、グラフィック API を提供	
aqm0804_st07032	I2C インターフェイスの AQM0804LCD に対してキャラクタ API を提供	
cyw43	Cyw43 通信用ドライバ群	PICO W 用

表 7.3.1 GDIC ディレクトリ

8 変更履歴

Version 1.4.4 以降の変更履歴を記載する。

version	date	functions

Appendix A Raspberry Pi PICO(W) TEB003(Arduino connector)ボード

TEB003 ボード依存仕様を記載する。

(1)Arduino connectors 定義

CN No.	Pin No.	Pin 名	MCU pin	機能
CN6 power	1	NC	-	-
	2	IOREF	-	3.3V Ref
	3	RESET	-	-
	4	+3V3	VCC3P3	3.3V input/output
	5	+5V	VCC5	5V output
	6	GND	GND	Ground
	7	GND	GND	Ground
	8	VIN	VCC9	Power input
CN8 analog	1	A0	GP26	-
	2	A1	GP27	-
	3	A2	GP22	-
	4	A3	GP28	-
	5	A4	GP0	-
	6	A5	GP1	-

表 A.1.1 左 Arduino connector 設定

CN No.	Pin No.	Pin 名	MCU pin	機能
CN5 digital	10	D15	GP21	(SCL)
	9	D14	GP20	(SDA)
	8	AREF	AREF	AVDD
	7	GND	GND	Ground
	6	D13	GP2	CLK
	5	D12	GP4	MISO
	4	D11	GP3	MOSI
	3	D10	GP5	SS(CS0)
	2	D9	GP6	GP6
	1	D8	GP7	GP7
CN9 degital	8	D7	GP14	GP14
	7	D6	GP15	GP15
	6	D5	GP10	GP10
	5	D4	GP11	GP11
	4	D3	GP12	GP12
	3	D2	GP13	GP13
	2	D1	GP8	TX
	1	D0	GP9	RX

表 A.1.2 右 Arduino connector 設定

(2)サンプルプログラム

Program name	shield	environment	detail	RTOS
ADC	TEB002 シールド	RAM	ADC テストプログラム	ASP
CORE1	-	RAM	CORE1 テストプログラム	ASP
I2C_SLAVE	-	RAM	I2C スレーブテストプログラム	ASP
i2cshield	TEB001 I2C シールド	RAM	I2C シールドを使った I2C マスターテストプログラム	
iwhishield	TEB002 シールド	RAM	グラフィック LCD 表示テストプログラム	
PWM	-	RAM	LED を使った PWM テストプログラム	ASP
SPI	-	RAM	SPI ループバックテストプログラム	ASP
TIMER	-	RAM	タイマーモジュールテストプログラム	ASP
ulcdshield	TEB001 UART/LCD シールド	RAM	LWLP の DHCP クライアント、ping テストが可能な実装 但し、ディレクトリ lwip に LWIP のソース設定(lwip-2.0.3/cntrib-2.0.1)が必要 また、設定ソースに数か所のパッチが必要	
USBD	-	RAM	USB デバイステストプログラム	FMP
USBH	-	RAM	USB ホストテストプログラム	FMP
WDOG	-	RAM	ウォッチドックタイマーテスト	ASP
WIFI	(TEB002 シールド)	ROM	WIFI テストプログラム(PICO W 専用)	FMP