TOPPERS BASE PLATFORM (RV) V1.4.1

担当者	竹内良輔

変更履歴

リリース	日付	作成者	変更内容
0.1.0	2019/04/25	竹内良輔	初版
0.1.1	2019/10/22	竹内良輔	Maix シリーズの対応
0.1.2	2019/11/29	竹内良輔	Maix 用カメラと I2C ドライバ追加
0.1.3	2019/12/28	竹内良輔	Maix 用 SD カード対応
1.4.1	2020/04/18	竹内良輔	正式リリース用に版を改定

目次

1　背景

TOPPERS BASE PLATFORM の拡張として RISC-V プロセッサに対応する。組込み用途で開発された RISC-V プロセッサを使用したエバレーション・ボードから最適なものを選択して組込みソフトウェアプラットフォームとして TOPPERS BASE PLATFORM(RV)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(RV))の仕様について記載する。

■ターゲットボード
・HI-FIVE1 SiFive 社
・Maix ボード Sipeed 社

3　TOPPERS BASE PLATFORM (RV)仕様

本章では、TOPPERS BASE PLATFORM (RV)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(RV) は HI-FIVE1 ボード(SoC FE310-G000)及び Maix シリーズの K210 上の TOPPERS ASP-1.9.3 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。

3.2　レイア構造

TOPPERS BASE PLATFORM(RV)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。FE310-G000 は他の一般的な SoC と比べ、RAM 領域が小さく、デバイス IP が少ないため、他の TOPPERS BASE PLATFORM に比べミドルウェアの対応が少ない。しかし、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。

図 3.2.1 に TOPPERS BASE PLATFORM (RV)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 Hi-Five1:Wire(I2C):仮実装、送信のみ可能/Maix:対応正式対応
③ UART
④ RTC
⑤ WDOG

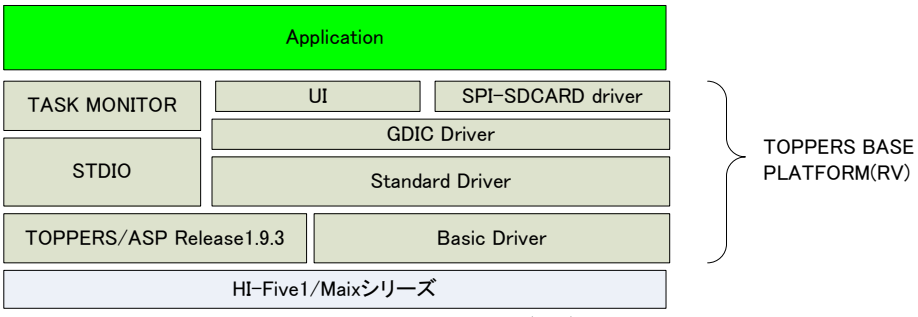

図 3.2.1 TOPPERS BASE PLATFORM(RV)構造図

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

PLATFORM(CV)		ターゲットボード	detail	対象の講座
Device Driver	Basic Driver	gpio		なし
		rtc		
		watch doc		
		(timer)	RTOS の機能	
		(uart)	RTOS 上に実装	
	Standard Driver	spi	GLCD/SD card	
		i2c	CLCD	
gdic	high driver		デバイス結合	
api middleware	File System	stdio	標準入出力	
	Graphic UI	ui	文字グラフィック表示	

注意：Maix シリーズ用のデバイスドライバはサンプル実装です。

3.3 開発環境

TOPPERS BASE PLATFORM(RV)は Windows10 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、LINEX でも開発可能である。
(1) gcc version 8.2(GNU MCU Eclipse: riscv-none-embed 8.2.0-2.1-20190425-1021)
(2) gcc version 8.2(newlib-3.0.0.20180831)

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(RV)では、３種類のデバイスドライバを提供する。**Maix シリーズのデバイスハードウェアの正式な仕様書はないためソフトウェアドライバ仕様は記載しない。**

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) rtc　　リアルタイム制御ドライバ
(3) wdoc　　ウォッチドックタイマードライバ
(4) 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。

FE310-G000 では GPIO ポートはひとつでピン番号は、0 から 31 までの 32 ピンである。GPIO ピンは IP 用のピンとして設定しているものがあり、すべてのピンを GPIO として使用できるわけではない。

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	otype	uint32_t	出力タイプ Push-Pull 設定のみ設定可能
4	oxor	uint32_t	出力 XOR 設定
5	iof	uint32_t	IOF 選択

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_LEVEL_LOW	0x10000000	GPIO 入力 LOW レベル割込み
GPIO_MODE_LEVEL_HIGH	0x10100000	GPIO 入力 HIGH レベル割込み
GPIO_MODE_OUTPUT	0x00000001	GPIO 出力モード

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	0x00000001	プルアップ有効

表 4.1.3.3 pull 設定値

③ otype

プルアップタイプのみ設定可能

定義	値	内容
GPIO_OTYPE_PP	0x00000000	プルアップタイプ

表 4.1.3.4 otype 設定値

④ oxor

出力データを XOR するか否かの設定

定義	値	内容

GPIO_OUTPUT_NORMAL	0x00000000	出力データをそのまま設定
GPIO_OUTPUT_XOR	0x00000001	出力データを XOR して設定

表 4.1.3.5 oxor 設定値

⑤ iof

IOF を使用するか、否かの設定

定義	値	内容
GPIO_NOIOF	0x00000000	IOF を使用しない：ソフト制御
GPIO_IOF0	0x00000001	IOF0 を使用する
GPIO_IOF1	0x00000002	IOF1 を使用する

表 4.1.3.6 iof 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
gpio_setup	void	uint32_t base GPIO_Init_t *init uint32_t pin	base アドレスと pin 番号で指定されたポートを初期化する。	アルタネートの設定デバイスピンの初期化の場合もある

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。

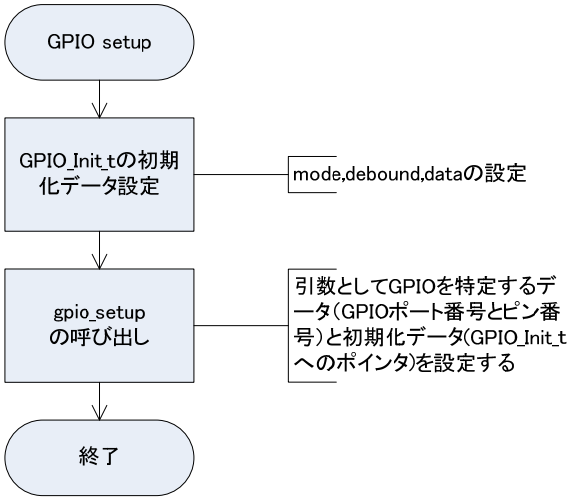

図 4.1.3.3.1 GPIO の設定フローチャート

4.1.4 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。

4.1.4.1 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 4.1.4.1.1 tm2 構造体

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
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 *pt	時刻をセットする	
rtc_get_time	ER	struct tm2 *pt	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 4.1.4.2.1 RTC 設定関数

4.1.4.3 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。

4.1.5 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。

4.1.5.1 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 4.1.5.1.1 WATCH DOG ポート番号

4.1.5.2 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 4.1.5.2.1 WATCH DOG ドライバ関数

4.1.6 UART

UART 用のデバイスドライバは asp 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

BASE PLATFORM (RV)として、スタンダードドライバとしたのは、以下の SPI ペリフェラルのみである。I2C は GPIO を使用したソフトウェアドライバである。現状は SDA のデータ受信値が不安定なため送信のみ可能となっている。

① SPI
② I2C

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

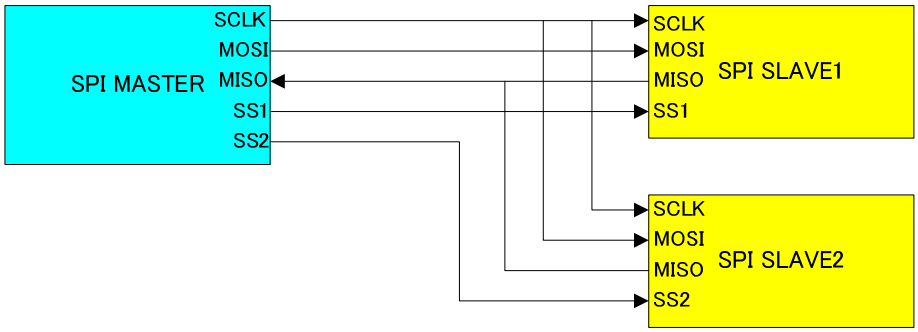

図 4.2.2.1 SPI 接続図

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。

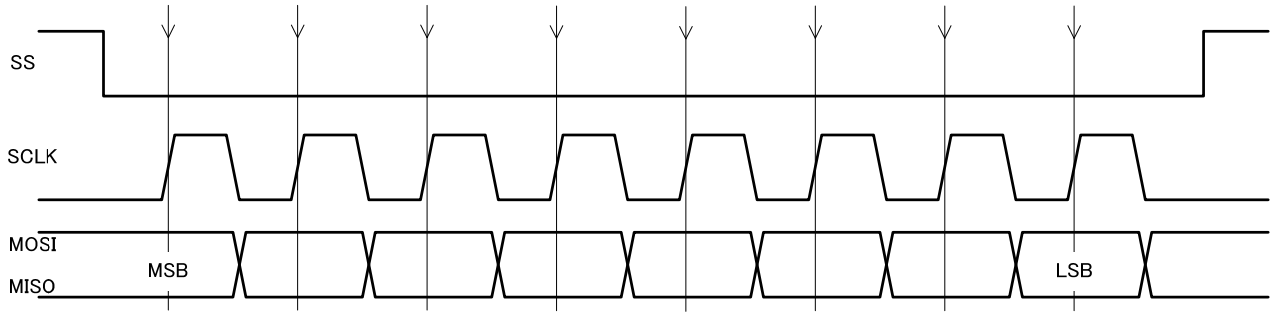

図 4.2.2.2 モード０、正パルス、前縁ラッチ、後端シフトのタイミング図

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	Direction	uint32_t	SPI 転送データ方向
2	DataSize	uint32_t	SPI 転送データサイズ
3	CLKPolarity	uint32_t	SPI 転送クロックの極性
4	CLKPhase	uint32_t	SPI クロック位相
5	CSMode	uint32_t	SPI CS 設定モード設定
6	Protocol	uint32_t	SPI プロトコル設定
7	CSDelay	uint32_t	SPI CS to SCK delay
8	SCKDelay	uint32_t	SPI SCK to CS delay
9	MinScInActTime	uint32_t	SPI Minimun CS Inactive time
10	MaxifTime	uint32_t	SPI Maximum Interface Delay
11	Prescaler	uint32_t	SPI クロック分周設定
12	SignBit	uint32_t	SPI MSB/LSB 設定
13	semid	int	通信用セマフォ ID（0 でセマフォなし）
14	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			
4	pTxBuffPtr	uint8_t *	送信データ領域へのポインタ
5	TxXferSize	uint16_t	要求送信サイズ
6	TxXferCount	uint16_t	送信済みサイズ
7	pRxBuffPtr	uint8_t *	受信データ領域へのポインタ
8	RxXferSize	uint16_t	受信要求サイズ
9	RxXferCount	uint16_t	受信済みサイズ
10	xmode	uint32_t	転送モード
11	xfercallback	void (*)(SPI_Handle_t *)	転送終了時コールバック関数
12	status	volatile uint32_t	送信用 DMA ハンドラ
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 Direction
SPI 通信方向設定のモード設定。

定義	値	内容
SPI_DIRECTION_2LINES	0x00000000	２ラインモード
SPI_DIRECTION_1LINE	SPI_FMT_DIR_TX	片方向モード

表 4.2.2.1.3 Direction 設定値

② DataSize
SPI のフレーム中のビットサイズを指定する。

③ 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_CLK_POL	極性 HIGH

表 4.2.2.1.4 CLKPolarity 設定値

④ 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_CLK_PHA	後縁ラッチ

表 4.2.2.1.5 CLKPhase 設定値

⑤ CSMode
SPI の CS 設定モードを設定する。

定義	値	内容
SPI_CSM_AUTO	SPI_CSMODE_AUTO	CS 自動制御
SPI_CSM_HOLD	SPI_CSMODE_HOLD	CS をホールドする
SPI_CSM_OFF	SPI_CSMODE_OFF	CS 制御を行わない

表 4.2.2.1.6 XferMode 設定値

⑥ Protocol
本 SPI はクワッド転送が可能なためプロトコルを設定する。

定義	値	内容
SPI_PROTOCOL_4LINES	SPI_PROTO_Q	4 ライン転送
SPI_PROTOCOL_2LINES	SPI_PROTO_D	2 ライン転送
SPI_PROTOCOL_1LINE	SPI_PROTO_S	1 ライン転送

表 4.2.2.1.7 Protocol 設定値

⑦ CSDelay
AUTO モードの場合、CS から SCLK までのデレィをクロックで設定する。レンジは 0 から 255 まで。

⑧ SCKDelay
AUTO モードの場合、SCLK から CS オフまでデレィをクロックで設定する。レンジは 0 から 255 まで。

⑨ MinScInActTime
AUTO モードの場合、CS オフから CS オンまでの最小遅延クロックを指定する。レンジは 0 から 255 まで。

⑩ MaxifTime
AUTO モード以外の設定で、CS をオフしなかった場合の 2 つのフレーム間の遅延クロックを設定する。レンジは 0 から 255 まで。

⑪ Prescaler
SCLK の周波数をシステムクロックの分周設定で指定する。

⑫ SignBit
SPI データ MSB/LSB 設定値。

定義	値	内容

SPI_DATA_MSB	SPI_FMT_ENDIAN_MSB	転送データ MSB
SPI_DATA_LSB	SPI_FMT_ENDIAN_LSB	転送データ LSB

表 4.2.2.1.8 SignBit 設定値

4.2.2.2 インターフェイス仕様

AUTO モードオフの場合、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_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。

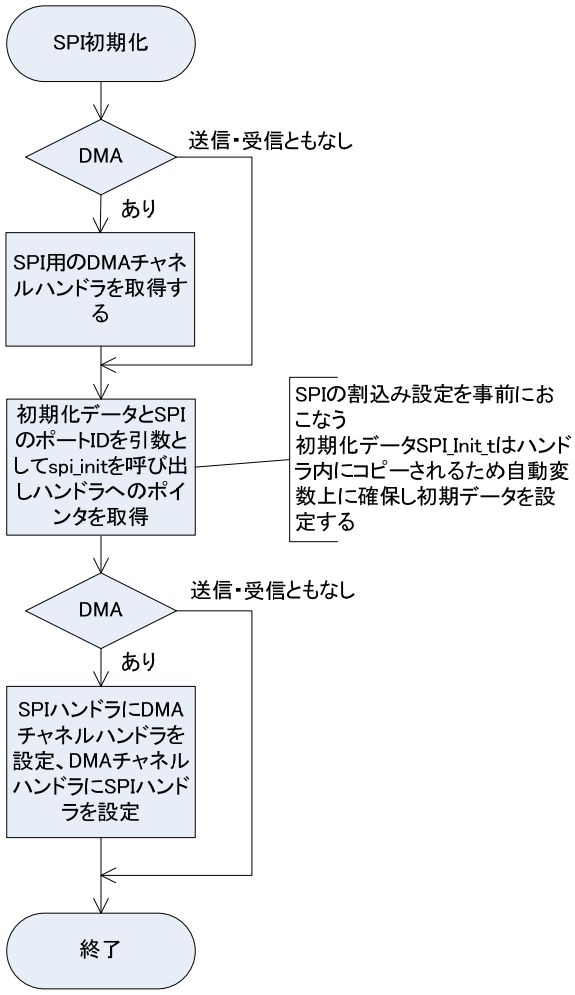

図 4.2.2.3.1 SPI 初期化フローチャート

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。

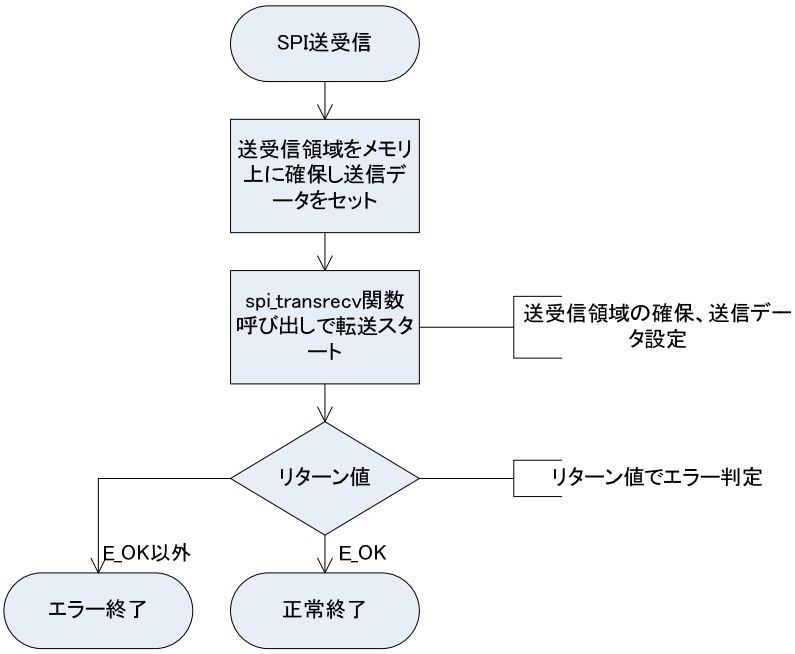

図 4.2.2.3.2 SPI 送受信フローチャート

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。

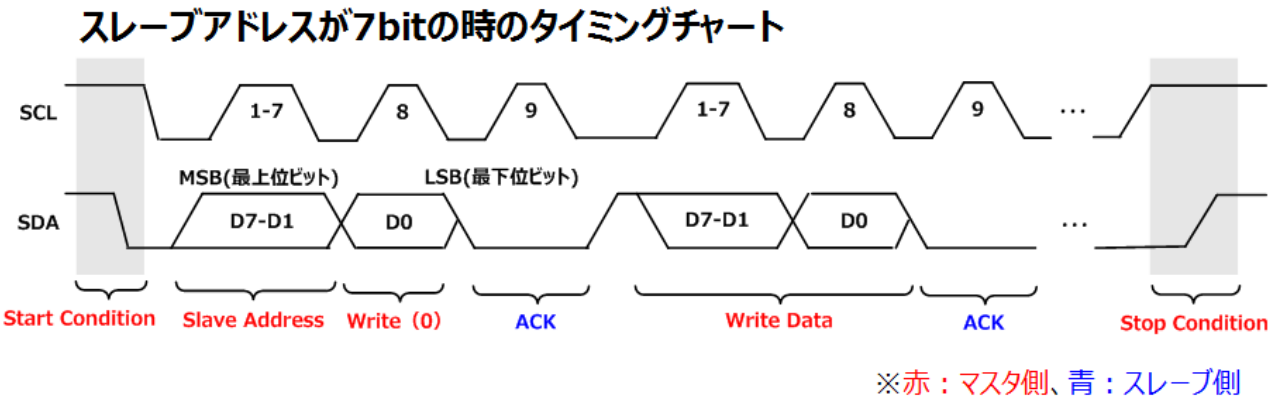

図 4.2.3.1 スレーブアドレス 7bit のデータ転送

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	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	portid	uint32_t	I2C ポート ID
4	clkCount	uint32_t	ナノ秒単位のクロック待ち時間
5	sclpin	uint16_t	SCL のピン番号
6	sdapin	uint16_t	SDA のピン番号
7	gpiopull	uint32_t	GPIO の PULLUP 設定保存領域
8	gpioiof1	uint32_t	GPIO の IOF 設定保存領域
9	sdaoe	uint32_t	現在の SDA のディレクション設定
10	ErrorCode	volatile uint32_t	I2C エラーコード(エラーは設定されない)

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_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_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。

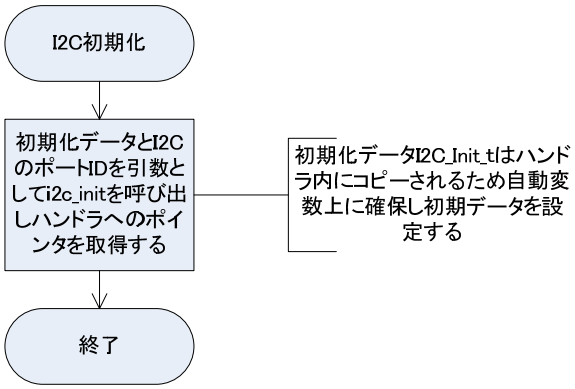

図 4.2.3.3.1 初期化フローチャート

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_memwrite、受信ならば 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

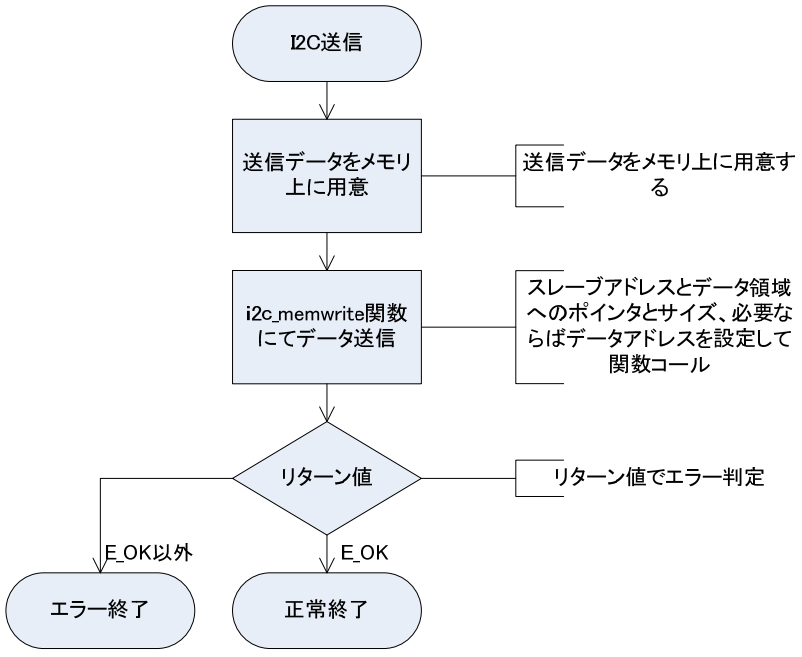

図 4.2.3.3.2 I2C 送信フローチャート

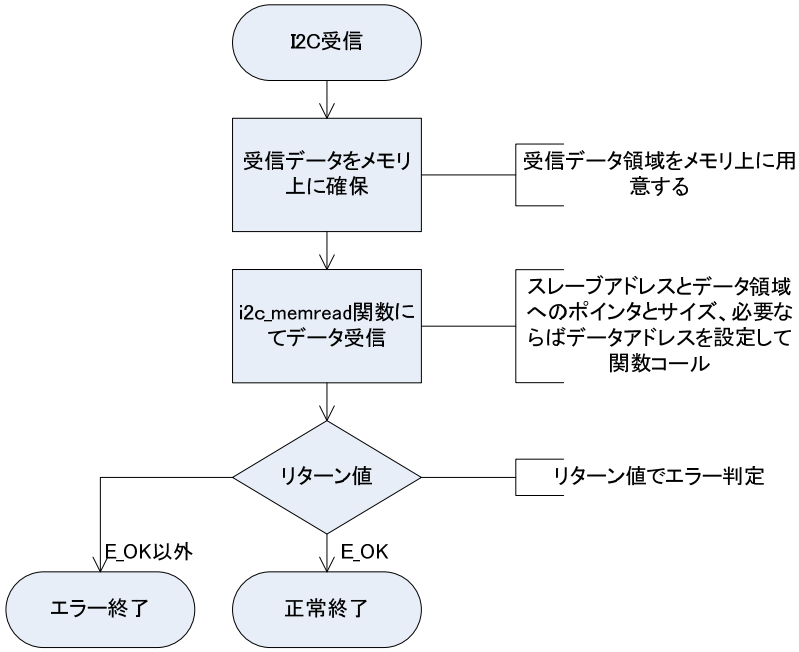

図 4.2.3.3.3 I2C 受信フローチャート

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

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。

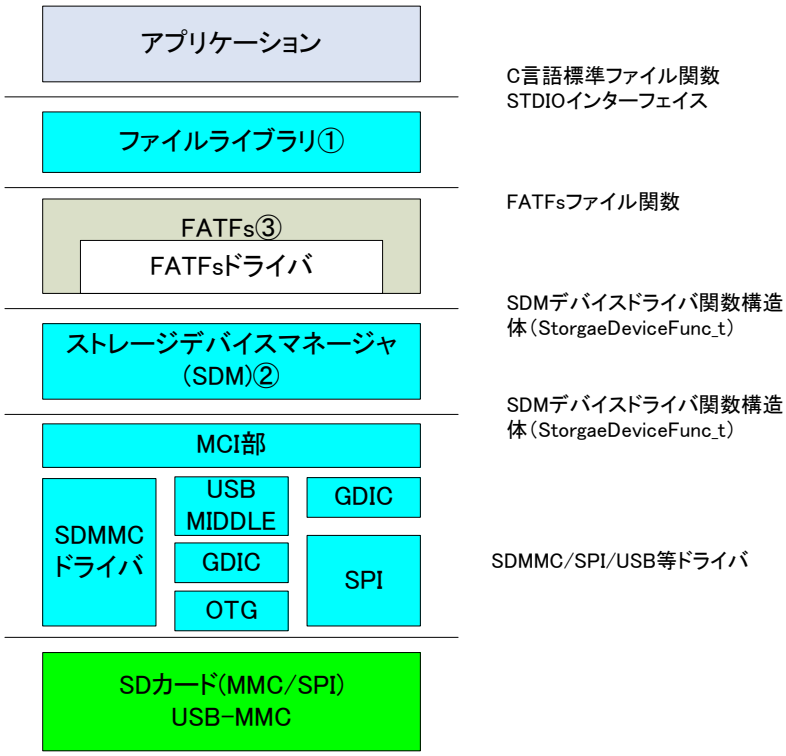

図 6.2.1 ファイルシステムのレイア構造

6.2.1 ファイルライブラリ

ファイルライブラリは標準入出力関数に合わせて、表 6.2.1.1 関数を提供する。これらはＣ言語上のファイル関数とファイル関係の POSIX 仕様であるが、以下の規定に従って関数の選択を行った。

・Ｃ言語標準のもの（C89,C99）
・POSIX.1-2001 でよく使われるもの
・LINUX 固有であるが、通常使用に必要なもの

関数名	型	引数	機能	備考
open	int	const char *pathname int flags	ファイルのオープン、ファイルディスクリプタを返す	エラー時-1 を返す
close	int	int fd	ファイルのクローズ	成功０、失敗-1
fstat	int	int fd struct stat *buf	ファイルの状態を取り出す。読み出しサイズを返す	-1 でエラー
lseek	off_t	int fd off_t offset int whence	ファイルの読み書き位置を設定する。	-1 でエラー
read	long	int fd void *buf long count	ファイルからデータを読み出す。読み出しサイズを返す。	0 で終端、-1 でエラー
write	long	int fd const void *buf long count	ファイルにデータを書き込む。書き込みサイズを返す。	-1 でエラー
mmap	void *	void *start size_t length int prot int flags int fd off_t offset	ファイルをメモリ上に配置する	一部ファイルシステムのみサポート
mumap	int	void *start size_t length	mmap の解除	
fopen	FILE*	const char *name const char *attr	ファイルオープン	
fclose	int	FILE *fp	ファイルクローズ	
fseek	int	FILE *fp long offset int whence	ファイルの読み書き位置を設定する	
stat	int	const char *name struct stat *buf	ファイルの状態の取出し	成功０、失敗-1
lstat	int	const char *name struct stat *buf	ファイルの状態の取出し	成功０、失敗-1
access	int	const char *name int mode	ファイルの状態の取出し	成功時 0、その他は-1
mkdir	int	const char *name mode_t mode	ディレクトリの作成	成功時 0、その他は-1
rmdir	int	const char *name	ディレクトリの削除	成功時 0、その他は-1
chmod	int	const char *name mode_t mode	ファイルモードの変更	成功０、失敗-1
remove	int	const char *name	ファイル削除	成功０、失敗-1
unlink	int	const char *name	ファイル削除	
rename	int	const char *oldpath const char *newpath	ファイル名の変更	成功０、失敗-1
opendir	void *	const char *path	path に対応したディレクトリストリームをオープンし、ストリームのポインタを返す。	エラー時 NULL
closedir	int	void *dir	ディレクトリストリームのクローズ	成功０、失敗-1
readdir	struct dirent*	void *dir	順番にディレクトリから dirent 構造体を読み、dirp で指されたバッファに格納する。	POSIX とは I/F 仕様が異なる
statfs	int	const char *path	マウントされたファイルシステ	成功０、失敗-1

		struct statfs2*stat	ムについての情報を返す。	

表 6.2.1.1 ファイルライブラリ関数

6.2.2 Storage Device Manager

Storage Device Manager は、ストレージデバイスとその下で実行される複数のローカルファイルシステムを管理するモジュールである。

6.2.2.1 データ仕様

Storage Device Manager は以下の４つの型で構成される。最初の２つは関数テーブルであり、表 6.2.2.1.1 の StorageDeviceFunc_t はローカルファイルシステムのデバイスを複数のストレージに対応させるために、関数テーブル化に用いるデバイスファンクションテーブルの型である。実際使用しているローカルファイルシステムが FATFs であるため、FATFs のデバイス I/F の関数テーブルとなっている。表 6.2.2.1.2 の StorageDeviceFileFunc_t 型は、ファイルライブラリを作成するために、ローカルファイルシステムのファイル関数をテーブル化するための型である。

番号	項目	型	機能
1	_sdev_sens	int (*)()	デバイスセンス関数
2	_sdev_diskinit	int (*)()	デバイスの初期化関数
3	_sdev_diskstatus	int(*)()	デバイスの状態取出し関数
4	_sdev_diskread	int(*)()	デバイスブロックリード関数
5	_sdev_diskwrite	int(*)()	デバイスブロックライト関数
6	_sdev_diskioctl	int(*)()	デバイス IOCTL 関数

表 6.2.2.1.1 StroageDeviceFunc_t 型

番号	項目	型	機能
1	_sdevff_opendir	void *(*)()	opendir 関数
2	_sdevff_closedir	int (*)()	closedir 関数
3	_sdevff_readdir	int (*)()	readdir 関数
4	_sdevff_mkdir	int (*)()	mkdir 関数
5	_sdevff_rmdir	int (*)()	rmdir 関数
6	_sdevff_unlink	int (*)()	unlink 関数
7	_sdevff_rename	int (*)()	rename 関数
8	_sdevff_chmod	int (*)()	chmod 関数
9	_sdevff_stat	int (*)()	stat 関数
10	_sdevff_statfs	int (*)()	statfs 関数
11	_sdevff_open	int (*)()	open 関数
12	_sdevff_close	int (*)()	close 関数
13	_sdevff_fstat	int (*)()	fstat 関数
14	_sdevff_lseek	off_t (*)()	lseek 関数
15	_sdevff_read	long (*)()	read 関数
16	_sdevff_write	long (*)()	write 関数
17	_sdevff_mmap	void *(*)()	mmap 関数

表 6.2.2.1.2 StorageDeviceFunc_t 型

あとの２つはストレージを管理する型で、表 6.2.2.1.3 StorageDevice_t 型はストレージデバイス自体の情報管理用の型で、表 6.2.2.1.4 StorageDeviceHead_t 型は StorageDevice_t 型のインスタンスを管理するヘッダ部である。表 6.2.2.1.5 は StorageDevice_t 型の_sdev_attribute のビット指定値を示す。SDEV_INSERTCHK がオンになっていないデバイスでは、メディアの挿抜チェックを行わない。この場合、初期化時のメディアの状態でメディアの有無を決定する。

番号	項目	型	機能

1	_sdev_attribute	uint16_t	デバイスの状態
2	_sdev_devno	uint8_t	デバイス番号
3	_sdev_port	uint8_t	デバイス種別
4	_sdev_maxsec	uint32_t	デバイスの最大セクタ数
5	_sdev_secsize	uint32_t	デバイスのセクタサイズ
6	_sdev_instimer	uint16_t	挿入タイマー
7	_sdev_inswait	uint16_t	挿入時待ち時間
8	_sdev_notice	void (*)()	検知コールバック関数
9	_sdev_local[4]	void *	ローカルエリア
10	pdevf	StorageDeviceFunc_t	デバイス関数テーブル
11	pdevff	StorageDeviceFileFunc_t*	ファイル関数テーブル

表 6.2.2.1.3 StroageDevice_t 型

番号	項目	型	機能
1	_num_activedev	uint16_t	登録済ストレージデバイスの数
2	_sdev_active	uint8_t	デバイスマネージャの有効無効
3	_default_device	uint8_t	デフォルトデバイス番号
4	_get_datetime	uint32_t (*)()	時刻取出し関数
5	_psd	StorageDevice_t *	ストレージデバイス配列

表 6.2.2.1.4 StroageDeviceHead_t 型

定義	値	内容
SDEV_ACTIVE	(1<<15)	デバイスがアクティブの状態
SDEV_INSERTCHK	(1<<14)	挿入・排出の設定あり
SDEV_CHKREMOVE	(1<<13)	排出検査を行う
SDEV_ONEEXIT	(1<<12)	一度以上排出があった
SDEV_EMPLOY	(1<<8)	デバイス動作中
SDEV_ERROR	(1<<7)	デバイスエラー
SDEV_DEVNOTUSE	(1<<0)	デバイス使用不可
SDEV_NOTUSE	255	使用不可のビット定義

表 6.2.2.1.5 _sdev_attribute 設定値

6.2.2.2 インターフェイス仕様

表 6.2.2.2.1 はストレージデバイスマネージャーで使用する関数である。sdev_init 関数にてストレージデバイスマネージャーは有効となり、sdev_terminate 関数で無効となる。この間で、ストレージデバイスの管理を行う。まず、SDMSetupDevice 関数でストレージデバイスの登録を行う。デバイス番号を指定して、この関数を呼び出すと、ppsdev にストレージデバイス(StoregeDevice_t)がセットされて戻る。使用者は、戻されたストレージデバイスに以下の属性等をセットする。

① デバイス関数テーブル：デバイスドライバテーブル
② ファイル管理テーブル：ローカルファイルシステムに対応したファイル関数テーブル
③ 属性：挿抜処理の有無
④ ローカルデータ：下位のデバイス等の情報を_sdev_local に設定する

ストレージデバイスに挿抜検知は SDMSense_task で行うが、デバイスに挿抜検知がない場合は、タスクレベルで検知を行う必要はない。この場合、SDEV_SENSE_ONETIME をコンパイルスイッチで定義してビルドすれば関数として設定され、デバイスの登録後、SDMSense_task(0)を関数コールすれば、メディアの有り無し処理を行う。SDMSense_task では、センス関数として psdev->pdevf->_sdevf_sense にてメディアのセンスを行うため、この関数を設定しておく必要がある。

SDMSense_task を使用しない場合は、SDMEmploy 関数を使って直接メディアの有無を設定することができる。

関数名	型	引数	機能	備考
sdev_init	void	intptr_t exinf	ストレージデバイスマネージ	

			ャーを初期化する	
sdev_terminate	void		ストレージデバイスマネージャーを終了する	
SDMSetupDevice	ER	int16_t devno StorageDevice_t **ppsdev	指定のデバイスを追加する	
SDMDeviceNo	ER_ID	const char **ppathname	パス名からデバイス番号を取り出す。パス名のデバイス名はスキップする	
SDMGetStorageDevice	Storage Device_t *	int devno	デバイス番号からストレージデバイスへのポインタを取り出す	
SDMEmploy	ER	StorageDevice_t *psdev bool_t sw	デバイスの動作中(true)、停止中(false)を設定する	
SDMSence_task	void	intptr_t exinf	デバイスセンスタスク（関数の場合あり）	

表 6.2.2.2.1 ストレージデバイスマネージャー関数

定義	内容
SDEV_SENSE_ONETIME	SDMSense_task を関数として使用
NUM_STORAGEDEVICE	ストレージデバイスの数(デフォルトは 4)
DEAFULT_DEVNO	デフォルトのデバイス番号(通常は 0)

表 6.2.2.2.2 コンパイルスイッチ設定値

6.2.3 FatFs

PLATFORM V1.X では、FAT12,16,32 用のローカルファイルシステムとして FatFs R0.07a を RTOS 対応のため一部修正して使用している。また、FatFs のデバイスドライバ(diskio.c)に関しては、複数のストレージに対応可能なようにデバイスファンクションテーブルを呼び出す形で改造を行っている。

6.3 時間管理

RTC デバイスが有効な場合、時間管理関数が有効となる。時間管理では、２つの型を用いて時刻の管理を行う。

6.3.1 データ仕様

表 6.3.1.1 の tm(tm2)構造体は時刻の管理を行う構造体である。tm 構造体はライブラリの time.h 中に定義されている時刻用構造体と同一のものである。tm2 は tm と同一の構造体で、standard device でローカルに定義しているものである。RTC デバイスの時刻処理は tm2 構造体を使用する。もうひとつが types.h 等で定義されている time_t 型で 1970 年 1 月 1 日からの経過時間を秒で表す。

番号	項目	型	機能
1	tm_sec	int	秒(0～59)
2	tm_min	int	分(0～59)
3	tm_hour	int	時(0～23)
4	tm_mday	int	月中の日(1～31)
5	tm_mon	int	月(1～12)
6	tm_year	int	年：西暦、但し 0 は 1970 年
7	tm_wday	int	週中の日(0～6)
8	tm_yday	int	年中の日(1～366)
9	tm_isdst	int	

表 6.2.1.1 tm 構造体

6.3.2 インターフェイス仕様

tm 構造体と time_t 型との変換する関数として表 6.3.2.1 に関数を用意する。

関数名	型	引数	機能	備考
mktime	time_t	struct tm *ptm	tm 構造体を time_t に変換する	
gmtime_r	struct tm *	const time_t *pt struct tm *ptm	time_t を tm 構造体に変換する	

表 6.3.2.1 時刻管理関数

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。

6.4.1 UI ディレクトリ

UI ディレクトリ以下に東雲フォント環境を用意する。UI 機能を取り込むには Makefile のミドルウェアの設定に以下のインクルードを行えばよい。

include $(SRCDIR)/ui/snfont_disp/Makefile.config

フォント	1 バイトフォント高さ	漢字フォント高さ	備考
東雲フォント	9,12,16	12,16	漢字フォントはファイルで提供

表 6.4.1.1 UI ディレクトリ

東雲フォントの漢字ビットマップデータは以下のヘッダを持つ、ファイルといて参照する。

以下に漢字フォントビットマップファイルのヘッダを示す。数値はリトルエンディアンとなる。

番号	項目		型	機能
1	Magic	0	char [4]	“FONT”
2	Name	4	char [32]	フォント名
3	Fsize	36	unsigned int	ファイルサイズ
4	Csize	40	unsigned int	ビットマップデータのコード部のバイト数
5	Isize	44	unsigned int	ビットマップデータのイメージ部のバイト数
6	off_cnv_table1	48	unsigned int	UTF-8N 2 バイトコード検索テーブルへのオフセット
7	siz_cnv_table1	52	unsigned int	UTF-8N 2 バイトコード検索テーブルへのコード数
8	off_cnv_table2	56	unsigned int	UTF-8N 3-1 バイトコード検索テーブルへのオフセット
9	siz_cnv_table2	60	unsigned int	UTF-8N 3-1 バイトコード検索テーブルへのコード数
10	off_cnv_table3	64	unsigned int	UTF-8N 3-2 バイトコード検索テーブルへのオフセット
11	siz_cnv_table3	68	unsigned int	UTF-8N 3-2 バイトコード検索テーブルへのコード数
12	off_2byte_font	72	unsigned int	UTF-8N 2 バイトコード＋イメージへのオフセット
13	siz_2byte_font	76	unsigned int	UTF-8N 2 バイトコード＋イメージのアイテム数
14	off_3byte_font	80	unsigned int	UTF-8N 3 バイトコード＋イメージへのオフセット
15	siz_3byte_font	84	unsigned int	UTF-8N 3 バイトコード＋イメージのアイテム数

表 6.4.1.2 東雲フォントファイルのヘッダ部構造体

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
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	
syssvc	malloc,calloc,free 関数	

表 7.1.1 共通部ディレクトリ

7.2 PDIC(Base/Standard)ドライバ

FE310 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,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
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.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	

表 7.2.1 FE310-G000 ドライバファイル

K210 用 Base/Standard 用ドライバ部は pdic/k210 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 ドライバ・ソースファイル	Base
device.cfg	RTC,WDOG の RTOS リソースファイル	Base
device.h	GPIO,DMA,WDOG,RTC ドライバ・インクルードファイル	Base
dvp.c	CAMERA ドライバ・ソースファイル	
dvp.cfg	CAMARA ドライバ RTOS リソースファイル	
dvp.h	CAMARA 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	制約あり
i2c.h	I2C ドライバ・インクルードファイル	制約あり
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	
spi_reg.c	レガシーSPI ドライバ・ソースファイル	
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	

表 7.2.2 K210 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
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_st7735	SPI インターフェイスの Adafruit 1.8”LCD に対して、グラフィック API を提供する	
adafruit_ILI9341	SPI インターフェイスの Adafruit 2.4”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	
bleshield2.1	Arduino BLE Shield2.1 に対して、BLE-API を提供する	
sipeed_ov2640	K210 用 ov2640 カメラ機能 API を提供	K210 専用
sipeed_st3389	SPI インターフェイスの K210 用 st3389LCD に対して、グラフィック API を提供	K210 専用

表 7.3.1 GDIC ディレクトリ

8 変更履歴

Version 1.0.0 以降の変更履歴を記載する。

version	date	functions

Appendix A HI-FIVE1

HI-FIVE1 の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	PIN_AH7(FPGA)	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	13	-
	2	A1	12	-
	3	A2	11	-
	4	A3	10	-
	5	A4	9	-
	6	A5	-	-

表 A.1.1 左 Arduino connector 設定

CN No.	Pin No.	Pin 名	MCU pin	機能
CN5 digital	10	D15	13	(SCL)
	9	D14	12	(SDA)
	8	AREF	-	AVDD

	7	GND	-	Ground
	6	D13	5	CLK
	5	D12	4	MISO
	4	D11	3	MOSI
	3	D10	2	SS(CS0)
	2	D9	1	GPIO7
	1	D8	0	GPIO6
CN9 degital	8	D7	23	GPIO5
	7	D6	22	GPIO4
	6	D5	21	GPIO3
	5	D4	20	GPIO2
	4	D3	19	GPIO1
	3	D2	18	GPIO0
	2	D1	17	TX
	1	D0	16	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
LCD	Adafruit 1.8 TFT	ROM	グラフィック LCD 表示テストプログラム
LCD2	Adafruit 2.8 TFT	ROM	グラフィック LCD 表示テストプログラム
bleshield21	BLE SHIELD2.1	ROM	BLE ペリフェラルテストプログラム
ulcdshield	TEB001 UART/LCD シールド	ROM	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
WDOG	-	ROM	ウォッチドックタイマーテスト
I2C	TEB001 I2C シールド	ROM	AMQ0802 の表示のみ可能

Appendix B Maixdino

Maixdino のボード依存仕様を記載する。

(1)Arduino connectors 定義

CN No.	Pin No.	Pin 名	IO pin	機能
CN6 power	1	NC	-	-
	2	IOREF	-	3.3V Ref
	3	RESET	PIN_AH7(FPGA)	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	IO33	-
	2	A1	IO32	-
	3	A2	IO35	-
	4	A3	IO34	-
	5	A4	IO39	-
	6	A5	IO36	-

表 B.1.1 左 Arduino connector 設定

CN No.	Pin No.	Pin 名	IO pin	機能
CN5	10	D15	IO30	(SCL)

digital	9	D14	IO31	(SDA)
	8	AREF	-	AVDD
	7	GND	-	Ground
	6	D13	IO3	CLK
	5	D12	IO10	MISO
	4	D11	IO11	MOSI
	3	D10	IO12	SS(CS0)
	2	D9	IO13	GPIO7
	1	D8	IO14	GPIO6
CN9 degital	8	D7	IO15	GPIO5
	7	D6	IO32	GPIO4
	6	D5	IO24	GPIO3
	5	D4	IO23	GPIO2
	4	D3	IO22	GPIO1
	3	D2	IO21	GPIO0
	2	D1	IO5	TX
	1	D0	IO4	RX

表 B.1.2 右 Arduino connector 設定

(2)SD カード接続ピン

Pin No.	機能
IO26	SPI-MISO
IO27	SPI-SCK
IO28	SPI-MOSI
IO29	SPI-SS

(3)サンプルプログラム

Program name	shield	environment	detail
DEMO		RAM	カメラ、グラフィック LCD 表示デモ
I2C	I2C シールド	RAM	I2C 表示センステスト、Mainduno では電気的に動作せず
SDCARD		RAM	SPI-SDCARD のファイルシステム・テストプログラム
TEST	-	RAM	ウォッチドックタイマーテスト