

TOPPERS BASE PLATFORM (CV) V1.1.1

| | |
|-----|-----------------------|
| 担当者 | 竹内良輔 (Educational WG) |
|-----|-----------------------|

変更履歴

| リリース | 日付 | 作成者 | 変更内容 |
|-------|------------|------|------------------------------|
| 0.1.0 | 2017/01/20 | 竹内良輔 | 仮版 |
| 0.2.0 | 2017/04/21 | 竹内良輔 | FPGA 関係の記述を追加 |
| 0.2.1 | 2017/07/07 | 竹内良輔 | ファイルシステムのブロック図を追加 |
| 0.3.0 | 2017/08/25 | 竹内良輔 | FPGA SPI ドライバの記載を追加 |
| 0.4.0 | 2017/12/05 | 竹内良輔 | TOPPERS USB MEDDLEWARE の対応 |
| 1.0.0 | 2018/05/31 | 竹内良輔 | LWIP 用 EMAC ドライバの対応/ADC 対応 |
| 1.1.0 | 2019/03/04 | 竹内良輔 | バージョンアップ、変更履歴詳細に記載 |
| 1.1.1 | 2020/04/18 | 竹内良輔 | lwip のバージョンを 2.1.2 アップ、エラッタ他 |
| | | | |
| | | | |
| | | | |

目次

| | |
|--------------------------------|----|
| 変更履歴 | 2 |
| 1 背景 | 5 |
| 2 目的 | 5 |
| 3 TOPPERS BASE PLATFORM (CV)仕様 | 5 |
| 3.1 プラットフォームのターゲット | 5 |
| 3.2 レイア構造 | 5 |
| 3.3 開発環境 | 7 |
| 4 Device Driver 仕様 | 7 |
| 4.1 Basic Driver | 7 |
| 4.1.1 概要 | 7 |
| 4.1.2 ドライバ一覧 | 7 |
| 4.1.3 GPIO | 7 |
| 4.1.3.1 データ仕様 | 7 |
| 4.1.3.2 インターフェイス仕様 | 8 |
| 4.1.3.3 フローチャート | 8 |
| 4.1.4 DMA | 9 |
| 4.1.4.1 データ仕様 | 9 |
| 4.1.4.2 インターフェイス仕様 | 12 |
| 4.1.4.3 フローチャート | 14 |
| 4.1.5 WATCH DOG | 14 |
| 4.1.5.1 データ仕様 | 14 |
| 4.1.5.2 インターフェイス仕様 | 15 |
| 4.1.6 TIMER | 15 |
| 4.1.7 UART | 15 |
| 4.2 Standard Driver | 15 |
| 4.2.1 概要 | 15 |
| 4.2.2 I2C | 16 |
| 4.2.2.1 データ仕様 | 16 |
| 4.2.2.2 インターフェイス仕様 | 18 |
| 4.2.2.3 フローチャート | 18 |
| 4.2.3 SPI | 20 |
| 4.2.3.1 データ仕様 | 21 |
| 4.2.3.2 インターフェイス仕様 | 24 |
| 4.2.3.3 フローチャート | 24 |
| 4.2.4 SDMMC | 26 |
| 4.2.4.1 データ仕様 | 26 |
| 4.2.4.2 インターフェイス仕様 | 28 |
| 4.2.4.3 設定手順 | 28 |
| 4.2.5 USB OTG | 31 |
| 4.2.5.1 データ仕様 | 31 |
| 4.2.5.2 インターフェイス仕様 | 34 |
| 4.2.5.3 フローチャート | 35 |
| 4.2.6 QSPI | 39 |
| 4.2.6.1 データ仕様 | 39 |
| 4.2.6.2 インターフェイス仕様 | 42 |
| 4.2.6.3 フローチャート | 43 |
| 4.2.7 EMAC | 43 |
| 4.2.7.1 データ仕様 | 43 |
| 4.2.7.2 インターフェイス仕様 | 45 |
| 4.2.7.3 フローチャート | 45 |
| 4.3 FPGA Driver | 47 |



| | | |
|--|------------------------|----|
| 4.3.1 | 概要 | 47 |
| 4.3.2 | FPGA pinmode | 48 |
| 4.3.2.1 | データ仕様 | 48 |
| 4.3.2.2 | インターフェイス仕様 | 48 |
| 4.3.2.3 | 設定手順 | 48 |
| 4.3.3 | FPGA uart | 48 |
| 4.3.3.1 | データ仕様 | 48 |
| 4.3.3.2 | インターフェイス仕様 | 49 |
| 4.3.3.3 | フローチャート | 50 |
| 4.3.4 | FPGA SPI | 50 |
| 4.3.4.1 | データ仕様 | 50 |
| 4.3.4.2 | インターフェイス仕様 | 51 |
| 4.3.4.3 | フローチャート | 51 |
| 4.3.5 | FPGA ADC | 52 |
| 4.3.5.1 | データ仕様 | 52 |
| 5 | タスクモニタ | 52 |
| 5.1 | 概要 | 52 |
| 5.2 | 標準入出力 | 53 |
| 5.3 | 標準デバッグコマンド | 54 |
| 5.4 | デバッグコマンド拡張 | 55 |
| 5.4.1 | データ仕様 | 55 |
| 5.4.2 | インターフェイス仕様 | 55 |
| 6 | API 層 | 55 |
| 6.1 | 概要 | 55 |
| 6.2 | ファイルシステム | 55 |
| 6.2.1 | ファイルライブラリ | 56 |
| 6.2.2 | Storage Device Manager | 57 |
| 6.2.2.1 | データ仕様 | 57 |
| 6.2.2.2 | インターフェイス仕様 | 59 |
| 6.2.3 | FatFs | 60 |
| 6.3 | 時間管理 | 60 |
| 6.3.1 | データ仕様 | 60 |
| 6.3.2 | インターフェイス仕様 | 60 |
| 6.4 | USB ミドルウェア | 60 |
| 6.4.1 | USB ホスト機能 | 60 |
| 6.4.2 | USB デバイス機能 | 61 |
| 6.5 | UI ミドルウェア | 61 |
| 6.5.1 | UI ディレクトリ | 61 |
| 7 | ファイルの構成 | 62 |
| 7.1 | 共通部 | 62 |
| 7.2 | Cyclone V hps ドライバ | 62 |
| 7.3 | FPGA ドライバ | 63 |
| 7.4 | GDIC ドライバ | 63 |
| 8 | 変更履歴 | 63 |
| Appendix A DE0-Nano-SoC Development Kit(Atlas-SoC Kit)/DE10-Nano | | 64 |
| Appendix B DK-DEV-5CSX6N | | 66 |

1 背景

TOPPERS 教育 WG では、基礎講座を中心とした教材開発を行ってきた。マルチ CPU、FPGA、組み込み用 LINUX などの新規の開発技術の対応に伴い上級講座開発することとなった。これに伴い上級講座用の TOPPERS BASE PLATFORM(CV)を開発とその仕様書 (リファレンスマニュアル) を作成することとなった。

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)の仕様について記載する。

TOPPERS BASE PLATFORM(CV) は Intel 社 Cyclone V SoC HPS と教育 WG で作成した FPGA に対応する。

■ターゲットボード

- ・ Cyclone V SoC 開発キット(DK-DEV-5CSX6N) Intel 社
- ・ DE0-Nano-SoC Development Kit(Atlas-SoC Kit) terasIC 社
- ・ DE10-Nano terasIC 社

3 TOPPERS BASE PLATFORM (CV)仕様

本章では、TOPPERS BASE PLATFORM (CV)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(CV) は 2 つのターゲットボード上の TOPPERS FMP-1.3.1/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(FMP-1.3.1 は Version 0.40 まで)。DE0-Nano-SoC には Arduino コネクタが実装されているが、これらは FPGA 側に設定されている。FPGA 用サンプル・ドライバは Arduino コネクタに対応している。Arduino コネクタを使用するには FPGA の設定を行い、FPGA ドライバを使用する必要がある。

Cyclone V SoC 環境では、DRAM (または SRAM) 上でプログラムを実行するのが、基本の実行形態である。また、BOOT のデフォルト設定は SD カードとなっている。Cyclone V SoC の標準的なブートシーケンスは以下の通りである。

- ① リセット→ブート ROM
- ② →プリローダ
- ③ →ブートローダ(uboot) ブートローダを RTOS に入れ替えれば、ここで実行可能
- ④ →OS 等の立ち上げ

本環境では、FPGA の書き換えを行わず工場出荷時のブート手順でプラットフォームが起動する設定とする。加えてブートローダ部分を教育 WG の ROM モニタが起動するように設定し、FMP は ROM モニタのダウンロード機能を用いて実行する形態をとる。もちろん、ROM モニタをビルド済みの FMP カーネルに入れ替えれば、カーネルの自動実行が可能となる。

なお、DK-DEV-5CSX6N ボードではプリローダでウォッチ・ドック・タイマーを起動しているため、OS 起動後、周期的にウォッチ・ドック・タイマーリセットを行う必要がある。

3.2 レイヤ構造

TOPPERS BASE PLATFORM(CV)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。これらのプラットフォームは、Realtime Kernel TOPPERS FMP1.3.1 または 1.4.0 上に構築する。また、FMP 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 (CV)の構造図を示す。Base Driver、Standard Driver、FPGA Driver、GDIC、File Library、USB 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 SD card File system(SDMMC)

- ② SPI/ FPGA SPI
- ③ Wire(I2C)
- ④ UART / FPGA UART
- ⑤ QSPI
- ⑥ USB OTG
- ⑦ MAC

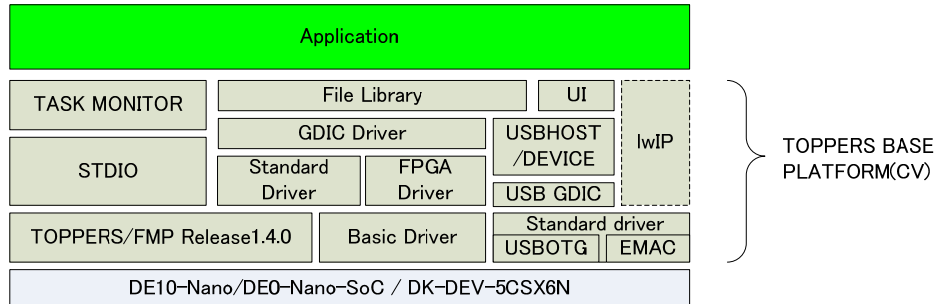


図 3.2.1 TOPPERS BASE PLATFORM(CV)構造図

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)FPGA driver FPGA に依存したサンプル・ドライバ層
- (4)GDIC driver 下位の driver を使用して構築する特定のデバイス用 driver
- (5)TOPPERS BASE PLATFORM で標準化している API 層
- (6)open source のライブラリ等

| PLATFORM(CV) | | ターゲットボード | detail | 対象の講座 |
|------------------|-----------------------|--------------------|-------------------------|-------------------------|
| Device Driver | Basic Driver | gpio | | 上級 1 (Reference Manual) |
| | | dma | | |
| | | watch doc (timer) | | |
| | | (uart) | | |
| | | | | |
| | Standard Driver | i2c | CLCD/senser/eeprom | |
| | | spi | GLCD/SD card | |
| | | sdmmc | SD card | |
| | | qspi | | |
| | | usb OTG | MSC/HID | |
| emac | | ehernet/PHY driver | | |
| FPGA Driver (*1) | pinmode | Arduino GPIO | 上級 2 (Reference Manual) | |
| | uart | Arduino UART | | |
| | spi | Arduino SPI | | |
| | wifi | ESP32 wifi driver | | |
| 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 (*2) | | プロトコル・スタック | | |

*1 : FPGA Driver を使用するには、FPGA の arduino connector 対応が必要

*2 : 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 は Windows7 以降のパソコンで cygwin または MIN-GW をインストールし、コンパイラ環境は GCC-ARM の以下のバージョンをインストールし開発を行っている。

- (1) gcc version 5.4(GCC ARM-2016q2-20160622)
- (2) gcc version 5.4(GCC ARM-2016q2-20160926)

V1.0.1 から Intel SOC EDS Command Shell も開発環境として対応可能とする

開発環境を SOC DES Command Shell に変更する場合は、

target/socfpga_cv_gcc/Makefile.target 中の GCC_TARGET 変数を以下のように変更すること

GCC_TARGET = ~~arm-none-eabi~~

↓

GCC_TARGET = ~~arm-altera-eabi~~

- (1) Quartus Prime 18.1 用 SOC DES Command Shell

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 PLATFORM 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ので、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。

TIMER と UART は、fmp カーネルで使用されている。基本的には fmp 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。

4.1.2 ドライバ一覧

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 I/O ドライバ
- (2) dma ダイナミック・メモリ・アクセスドライバ
- (3) timer タイマードライバ
- (4) uart シリアルドライバ

4.1.3 GPIO

GPIO は汎用の I/O 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。

Cyclone V SoC HPS では 3 つの GPIO ポートを持ち、ポート 1, 2 では 29 ピン、ポート 3 では 23 ピンの GPIO ポートを制御可能となっている。

4.1.3.1 データ仕様

Cyclone V HPS の 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 | debounce | uint32_t | 出力 Pull-Up/Pull-Down 設定 |
| 3 | data | uint32_t | 出力タイプ Open-drain/Push-Pull 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_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 |
| GPIO_MODE_AF | 0x00000002 | アルタネートピン設定 |
| GPIO_MODE_ANALOG | 0x00000003 | アナログピン設定 |

表 4.1.3.2 mode 設定値

② debounce

debounce は GPIO ピンの動作波形を緩やかに設定する機能である。GPIO をスイッチとして使用する場合、ある程度のチャタリングを除去できる。

| 定義 | 値 | 内容 |
|-----------------|------------|---------------|
| GPIO_NODEBOUNCE | 0x00000000 | Debounce 機能無効 |
| GPIO_DEBOUNCE | 0x00000001 | Debounce 機能有効 |

表 4.1.3.3 debounce 設定値

③ data

出力設定の場合、初期値を指定可能、値は0または1

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ウン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。

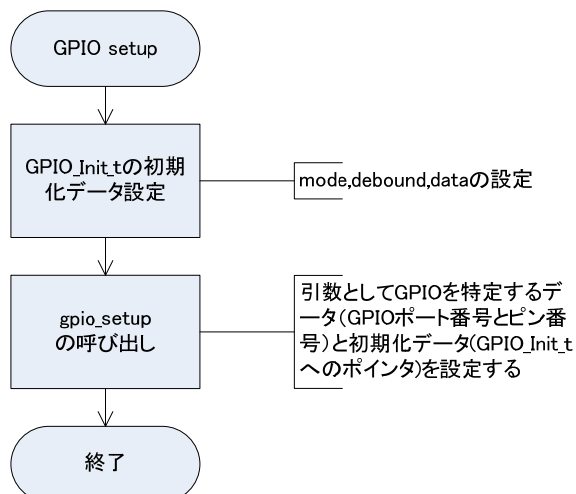


図 4.1.1.3.1 GPIO の設定フローチャート

GPIO の入力データをポーリングで取り出す場合、PORT のデータレジスタはハードウェア上の信号レベルを反映しない。このドライバでは、割り込みない入力設定でも HIGH-LEVEL 割り込み設定で、割り込みマスクを行って初期化する。そのため、入力データは割り込みステータスレジスタのポーリングで取り出すことができる。

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。HPS のデバイスは大きな FIFO を持つ場合が多く、デバイスとメモリ間で高速にデータ通信しても、オーバーラン・エラーやアンダーラン・エラーが発生しない。本プラットフォームでは SPI が 256 フレーム FIFO しか持たないため、FIFO を超えるデータ量が発生した場合用に FIFO モードと DMA モードの選択実行可能なように実装を行った。

HPS では 8 チャンネルの DMA チャンネルを持ち、DMA デバイスドライバはチャンネル単位に DMA 設定を行うことができる。

4.1.4.1 データ仕様

DMA ドライバは初期設定と管理用に表 4.1.4.1 の DMACH_Handle_t 型を持つ。このハンドラはチャンネル初期化に必要な項目と、生成後入れ替え可能な項目がある。設定の項に○が入っている項目が、チャンネルのオープン時に設定が必要な項目である。

| 番号 | 項目 | 型 | 機能 | 設定 |
|----|-------------|-----------|-------------------|----|
| 1 | dmach_id | uint8_t | DMA のチャンネル番号 | |
| 2 | periph_info | uint8_t | ペリフェラル番号 | ○ |
| 3 | periph_can | uint8_t | CAN/FPGA 選択 | ○ |
| 4 | periph_sec | uint8_t | セキュア設定 | ○ |
| 5 | irq_sec | uint8_t | 割り込みセキュア設定 | ○ |
| 6 | evt_req | uint8_t | イベントの指定を行う | ○ |
| 7 | event | uint8_t | 転送モード設定 | ○ |
| 8 | dummy | uint8_t | リザーブ | |
| 9 | phandle | uint32_t | ペリフェラルのハンドラへのポインタ | |
| 10 | pbuff | void* | 転送対象メモリアドレス | |
| 11 | xfersize | uint8_t | 転送データ長 | |
| 12 | xfercount | uint32_t | 転送済カウンタ | |
| 13 | errorcode | uint32_t | DMA エラーコード | |
| 14 | dma_start | ER (*)0 | DMA 設定用のコールバック関数 | |
| 15 | dma_end | void (*)0 | DMA 終了時のコールバック関数 | |

| | | | | |
|----|-----------|-----------|-----------------------|--|
| 16 | flag | uint16_t | DMA コード管理領域：ユーザーの設定不要 | |
| 17 | code_size | uint16_t | DMA コード管理領域：ユーザーの設定不要 | |
| 18 | loop0 | uint16_t | DMA コード管理領域：ユーザーの設定不要 | |
| 19 | loop1 | uint16_t | DMA コード管理領域：ユーザーの設定不要 | |
| 20 | sar | uint16_t | DMA コード管理領域：ユーザーの設定不要 | |
| 21 | dar | uint16_t | DMA コード管理領域：ユーザーの設定不要 | |
| 22 | program | uint8_t* | DMA コード管理領域：ユーザーの設定不要 | |
| 23 | buffer | uint8_t[] | DMA コード管理領域：ユーザーの設定不要 | |

表 4.1.4.1 DMACH_Handle_t 型

① dmach_id

dmach_id はチャンネルのオープン時自動設定されるため、DMACH_Handle_t に直接設定を行う必要はない。

| 定義 | 値 | 内容 |
|----------------|---|---------|
| DMA1_CHANNELID | 1 | チャンネル 1 |
| DMA2_CHANNELID | 2 | チャンネル 2 |
| DMA3_CHANNELID | 3 | チャンネル 3 |
| DMA4_CHANNELID | 4 | チャンネル 4 |
| DMA5_CHANNELID | 5 | チャンネル 5 |
| DMA6_CHANNELID | 6 | チャンネル 6 |
| DMA7_CHANNELID | 7 | チャンネル 7 |
| DMA8_CHANNELID | 8 | チャンネル 8 |

表 4.1.4.2 ChannelID

② periph_info

DMA がペリフェラルに対応する場合、ペリフェラル番号を設定する。4 から 7 は CAN モジュールと FPGA の共用であるため、どちらを指定するか periph_can で再設定する必要がある。

| 定義 | 値 | 内容 |
|-------------------------------|----|--------------------|
| DMA_PERIPH_FPGA_0 | 0 | FPGA0 |
| DMA_PERIPH_FPGA_1 | 1 | FPGA1 |
| DMA_PERIPH_FPGA_2 | 2 | FPGA2 |
| DMA_PERIPH_FPGA_3 | 3 | FPGA3 |
| DMA_PERIPH_FPGA_4_OR_CAN0_IF1 | 4 | FPGA4 または CAN0_IF1 |
| DMA_PERIPH_FPGA_5_OR_CAN0_IF2 | 5 | FPGA5 または CAN0_IF2 |
| DMA_PERIPH_FPGA_6_OR_CAN1_IF1 | 6 | FPGA6 または CAN1_IF1 |
| DMA_PERIPH_FPGA_7_OR_CAN1_IF2 | 7 | FPGA7 または CON1_IF2 |
| DMA_PERIPH_I2C0_TX | 8 | メモリから I2C0 |
| DMA_PERIPH_I2C0_RX | 9 | I2C0 からメモリ |
| DMA_PERIPH_I2C1_TX | 10 | メモリから I2C1 |
| DMA_PERIPH_I2C1_RX | 11 | I2C1 からメモリ |
| DMA_PERIPH_I2C2_TX | 12 | メモリから I2C2 |
| DMA_PERIPH_I2C2_RX | 13 | I2C2 からメモリ |
| DMA_PERIPH_I2C3_TX | 14 | メモリから I2C3 |
| DMA_PERIPH_I2C3_RX | 15 | I2C3 からメモリ |
| DMA_PERIPH_SPI0_MASTER_TX | 16 | メモリから SPIM0 |
| DMA_PERIPH_SPI0_MASTER_RX | 17 | SPIM0 からメモリ |
| DMA_PERIPH_SPI0_SLAVE_TX | 18 | メモリから SPIS0 |
| DMA_PERIPH_SPI0_SLAVE_RX | 19 | SPIS0 からメモリ |
| DMA_PERIPH_SPI1_MASTER_TX | 20 | メモリから SPIM1 |
| DMA_PERIPH_SPI1_MASTER_RX | 21 | SPIM1 からメモリ |

| | | |
|--------------------------|----|------------------------|
| DMA_PERIPH_SPI1_SLAVE_TX | 22 | メモリから SPIS1 |
| DMA_PERIPH_SPI1_SLAVE_RX | 23 | SPIS1 からメモリ |
| DMA_PERIPH_QSPI_FLASH_TX | 24 | メモリから QSPI |
| DMA_PERIPH_QSPI_FLASH_RX | 25 | QSPI からメモリ |
| DMA_PERIPH_STM | 26 | System Trace Macrocell |
| DMA_PERIPH_UART0_TX | 28 | メモリから UART0 |
| DMA_PERIPH_UART0_RX | 29 | UART0 からメモリ |
| DMA_PERIPH_UART1_TX | 30 | メモリから UART1 |
| DMA_PERIPH_UART1_RX | 31 | UART1 からメモリ |

表 4.1.4.3 periph_info 設定値

- ③ periph_can
periph_info が CAN モジュールと FPGA の共用の場合、どちらかを再定義する。0:FPGA、1:CAN
- ④ periph_sec
セキュア設定を行うかどうかの指定、0:行わない、1:行う
- ⑤ ireq_sec
割込みのセキュア設定を行うかどうかの指定、0:行わない、1:行う
- ⑥ evt_req
イベント設定を行う。このドライバでは DMA_EVENT_INTREQ を設定してください。

| 定義 | 値 | 内容 |
|------------------|---|-----------------|
| DMA_EVENT_NONE | 0 | イベントを発生させない |
| DMA_EVENT_REQ | 1 | イベントを要求する |
| DMA_EVENT_INTREQ | 2 | 割込みをイベントとして要求する |

表 4.1.4.4 evt_req 設定値

- ⑦ event
発生イベントを指定する。本ドライバでは DMA チャンネル ID に対応したイベントを指定してください。

| 定義 | 値 | 対応 |
|-----------------|---|--------------------|
| DMA_EVENT_0 | 0 | DMA1_CHANNELID |
| DMA_EVENT_1 | 1 | DMA2_CHANNELID |
| DMA_EVENT_2 | 2 | DMA3_CHANNELID モード |
| DMA_EVENT_3 | 3 | DMA4_CHANNELID |
| DMA_EVENT_4 | 4 | DMA5_CHANNELID |
| DMA_EVENT_5 | 5 | DMA6_CHANNELID |
| DMA_EVENT_6 | 6 | DMA7_CHANNELID |
| DMA_EVENT_7 | 7 | DMA8_CHANNELID |
| DMA_EVENT_ABOUT | 8 | |

表 4.1.4.5 Mode 設定値

- ⑧ phandle
DMA がペリフェラルに対応する場合、DMA チャンネルハンドラをオープン後、ペリフェラルのハンドラを登録する。
- ⑨ pbuff
ペリフェラルでデータ転送の対象となるメモリ領域の先頭ポインタ、dma_start 関数で設定する。
- ⑩ xfersize



⑪ ペリフェラルでデータ転送の対象となるメモリ領域のバイトサイズ、`dma_start` 関数で設定する。

⑫ `xfercount`

DMA 転送時の転送カウンタ、`xfersize` と一致すれば、転送終了。

⑬ `errorcode`

DMA 転送エラー発生時のエラーコード

| 定義 | 値 | 内容 |
|--------------------------------------|------------|--------------------------|
| <code>DMA_ERROR_NONE</code> | 0x00000000 | エラーなし |
| <code>DMA_ERROR_CH_UNDEFINST</code> | 0x00000001 | チャンネル・インストラクション未定義 |
| <code>DMA_ERROR_CH_OPEINVALID</code> | 0x00000002 | チャンネル・オペランド不良 |
| <code>DMA_ERROR_CH_EVTERR</code> | 0x00000004 | チャンネル・イベントエラー |
| <code>DMA_ERROR_CH_PERIPERR</code> | 0x00000008 | チャンネル・ペリフェラルエラー |
| <code>DMA_ERROR_CH_RDWRERR</code> | 0x00000010 | チャンネル・READ/WRITE エラー |
| <code>DMA_ERROR_CH_MFIFOERR</code> | 0x00000020 | チャンネル・MFIFO エラー |
| <code>DMA_ERROR_CH_DATAUNAVL</code> | 0x00000040 | チャンネル・データアベラブルエラー |
| <code>DMA_ERROR_CH_INSTFETERR</code> | 0x00000080 | チャンネル・インストラクション・フェッチエラー |
| <code>DMA_ERROR_CH_DWRITEERR</code> | 0x00000100 | チャンネル・データ WRITE エラー |
| <code>DMA_ERROR_CH_DREADERR</code> | 0x00000200 | チャンネル・データ READ エラー |
| <code>DMA_ERROR_CH_DBGINST</code> | 0x00001000 | チャンネル・デバッグインストラクションエラー |
| <code>DMA_ERROR_CH_LOCKUPERR</code> | 0x00002000 | チャンネル・ロックアップエラー |
| <code>DMA_ERROR_MG_UNDEFINST</code> | 0x00010000 | マネージャー・インストラクション未定義 |
| <code>DMA_ERROR_MG_OPEIVALID</code> | 0x00020000 | マネージャー・オペランド不良 |
| <code>DMA_ERROR_MG_DMAGOERR</code> | 0x00040000 | マネージャー・DMAGO エラー |
| <code>DMA_ERROR_MG_EVNTERR</code> | 0x00080000 | マネージャー・パーミッションエラー |
| <code>DMA_ERROR_MG_INSTFETERR</code> | 0x00800000 | マネージャー・インストラクション・フェッチエラー |
| <code>DMA_ERROR_MG_DBGINST</code> | 0x01000000 | マネージャー・デバッグインストラクションエラー |

表 4.1.4.6 errorcode 設定値

⑭ `dma_start`

ペリフェラル用 DMA 初期化、スタート設定コールバック関数

⑮ `dma_end`

DMA 転送終了時に呼び出されるコールバック関数、呼び出しには、イベントリクエスト設定を割り込み(`evt_req=DMA_EVENT_INTREQ`)、イベント定義(`event`)をチャンネルに対応したイベントに設定しなければならない。

4.1.4.2 インターフェイス仕様

DMA を制御するドライバ関数を以下に示す。HPS 用の DMA はメモリ上に DMA コードを記載し、`dma_channel_exe` 関数により、DMA を起動する形で実行される。`dma_program_start` から `dma_program_end` までの関数は、DMA チャンネルハンドラ内の `buffer` 領域に DMA コードを記載するためのプログラムである。

| 関数名 | 型 | 引数 | 機能 | 備考 |
|-------------------------------|------|---|------------------|----|
| <code>dma_init</code> | void | <code>intptr_t exinf</code> | DMA デバイスの初期化を行う。 | |
| <code>dma_deinit</code> | void | void | DMA を未使用状態に戻す | |
| <code>dma_channel_open</code> | ER | ID <code>channelid</code> <code>DMACH_Handler_t *hdmach</code> | DMA チャンネルをスタート | |

| | | | | |
|----------------------|------|--|--|--|
| | | | させる。 | |
| dma_channel_close | ER | DMACH_Handler_t *hdmach | DMA チャンネルを停止させる。 | |
| dma_periph_transfer | ER | DMACH_Handler_t *hdmach void *addr uint32_t size | ペリフェラルの DMA 設定を行う。dma_start を設定する必要がある | |
| dma_memory_to_memory | ER | DMACH_Handler_t *hdmach void *dst void *src uint32_t size | メモリ to メモリのデータ転送 | |
| dma_zero_to_memory | ER | DMACH_Handler_t *hdmach void *buf uint32_t size | メモリに ZERO フィルを行う | |
| dma_channel_exec | ER | DMACH_Handler_t *hdmach | DMA コードを実行する | |
| dma_channel_state | ER | DMACH_Handler_t *hdmach | DMASTATE を取り出す | |
| dma_hps_isr | void | intprpt_t exinf | ストリーム DMA の割込みハンドラ | |
| dma_program_start | ER | DMACH_Handler_t *hdmach | DMA コード:作成開始要求 | |
| dma_program_move | ER | DMACH_Handler_t *hdmach uint8_t chan_reg uint32_t val | DMA コード: MOVE 設定 | |
| dma_program_mode | ER | DMACH_Handler_t *hdmach uint8_t mode uint8_t inst | DMA コード: モード設定 | |
| dma_program_lpstart | ER | DMACH_Handler_t *hdmach uint8_t iterations | DMA コード:ループスタート | |
| dma_program_lpend | ER | DMACH_Handler_t *hdmach uint8_t mode uint8_t iteration | DMA コード:ループ終了 | |
| dma_program_periph | ER | DMACH_Handler_t *hdmach uint8_t mode bool_t in | DMA コード:ペリフェラル設定 | |
| dam_program_end | ER | DMACH_Handler_t *hdmach bool_t wmb | DMA コード: 終了要求 | |

表 4.1.4.7 DMA 設定関数

DMA STATE の内容を表 4.1.4.8 に示す。

| 定義 | 値 | 内容 |
|-------------------------------|----|--------------|
| DMA_STATE_STOPPED | 0 | 停止状態 |
| DMA_STATE_EXECUTING | 1 | 実行状態 |
| DMA_STATE_CACHE_MISS | 2 | キャッシュミスが発生した |
| DMA_STATE_UPDATING_PC | 3 | PC のアップデート |
| DMA_STATE_WFE | 4 | イベント待ち状態 |
| DMA_STATE_AT_BARRIER | 5 | At Barrier |
| DMA_STATE_WFP | 7 | ペリフェラル待ち状態 |
| DMA_STATE_KILLING | 8 | 中断中 |
| DMA_STATE_COMPLETING | 9 | 終了状態 |
| DMA_STATE_FAULTING_COMPLETING | 14 | エラーあり、終了状態 |
| DMA_STATE_FAULTING | 15 | エラー状態 |

表 4.1.4.8 DMA STATE 設定値

4.1.4.3 フローチャート

DMA 処理は DMA 全体の初期化と DMA チャンネルを使用した転送処理に分けられる。HPS の場合、DMA チャンネルは 8 つ用意されており、DMA オープン時の DMA チャンネルハンドラへの設定により、使用用途が決定する。DMA モジュールの初期化は `dma_init` 関数にて行う。DMA モジュールを使用しない場合は、`dma_deinit` 関数にて停止する。

初期化終了後、DMA チャンネルを使用する場合は、DMA チャンネル ID と DMA チャンネルハンドラ（領域は自分で確保）へのポインタを渡して `dma_channel_open` 関数を呼び出す。正常終了ならば、指定の DMA チャンネルは使用可能となる。DMA チャンネルが不要となった場合は、`dma_channel_close` 関数にて DMA チャンネルを解放する。

本 DMA モジュールを使用可能なペリフェラルは表 4.1.4.3 `periph_info` で設定可能なペリフェラルのみである。SPI モジュールを DMA 用いて使用する場合の設定手順をフローチャートで示す。SPI は送信、受信の両方に DMA を使用可能であるが、説明を簡略化するために受信で DMA を使用する手順の説明を行う。この場合、送信は FIFO を用いた割込み送信となる。DMA 用の割込みサービスルーチンは `device.cfg` で設定済みなので新たに設定を行う必要はない。

SPI 受信 DMA 初期設定フローチャートを図 4.1.4.3.1 に示す。この設定で SPI の送信、受信処理を行えば受信側が DMA 実行となる。

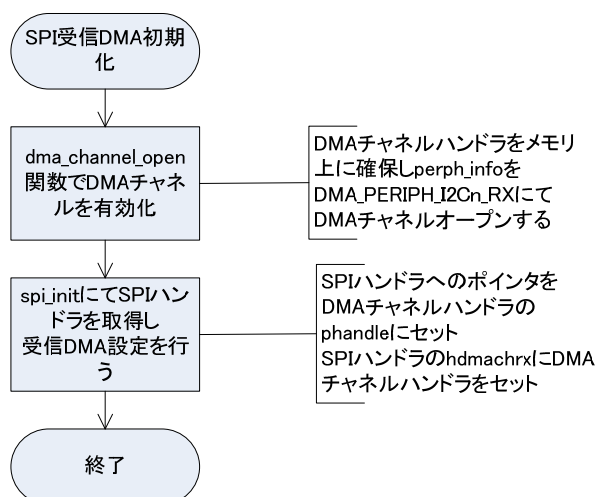


図 4.1.4.3.1 SPI-DMA 初期化

現状、BASE PLATFORM(CV)では SPI 以外のドライバは DMA モードに対応していない注 1) `periph_info` に対応してペリフェラルを DMA 化するには、`dma_start` と `dma_end` 関数を自作しなければならない。

注 1) USB OTG モジュールはモジュール自体に DMA 機能を持つ。

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。Hps は複数の WATCH DOG 機能を持つが、このドライバは L4 Watch Dog Timer の制御を行う。

4.1.5.1 データ仕様

L4 WATCH DOG は Hps 上で 2 つ用意されている。2 つ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 | L4 Watch Dog のポート 0 を示す |

| | | | |
|---|--------------|---|-------------------------|
| 2 | WDOG2_PORTID | 2 | L4 Watch Dog のポート 1 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。タイムアウトクロックは 2147483648 クロックに固定、regtim は 0~15 の範囲でレギュラータイムアウト値、inittim は 0~15 の範囲でイニシャルタイムアウト値を設定する。

| 関数名 | 型 | 引数 | 機能 | 備考 |
|-------------|----|--|-----------------------------------|----|
| wdog_init | ER | ID portid | 指定ポート ID の WATCH DOG ペリフェラルを初期化する | |
| wdog_deinit | ER | ID portid | WATCH DOG を未使用状態に戻す | |
| wdog_start | ER | ID portid uint8_t regtim uint8_t inittim | WATCH DOG スタートさせる | |
| wdog_stop | ER | ID portid | WATCH DOG を停止させる | |
| 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 TIMER

TIMER は一般的に使用される周期タイマー割込み以外に特殊な機能を持つものもある。例えば PWM 出力の設定のようなクロック生成にも使用されるため、ドライバの API を統一することは難しい。RTOS のシステムタイマーとして使用する場合は、fmp カーネル内に記述があるため、それを参照して頂きたい。

4.1.7 UART

UART 用のデバイスドライバは 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

BASE PLATFORM (CV) として、スタンダードドライバとしたのは、以下の 5 つのペリフェラルである。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用意しなければならない。(SPI 用は SPI xDriver を用意している)

- ① I2C
- ② SPI
- ③ SDMMC
- ④ USB OTG
- ⑤ QSPI

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 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 4.2.2.1 に示す。

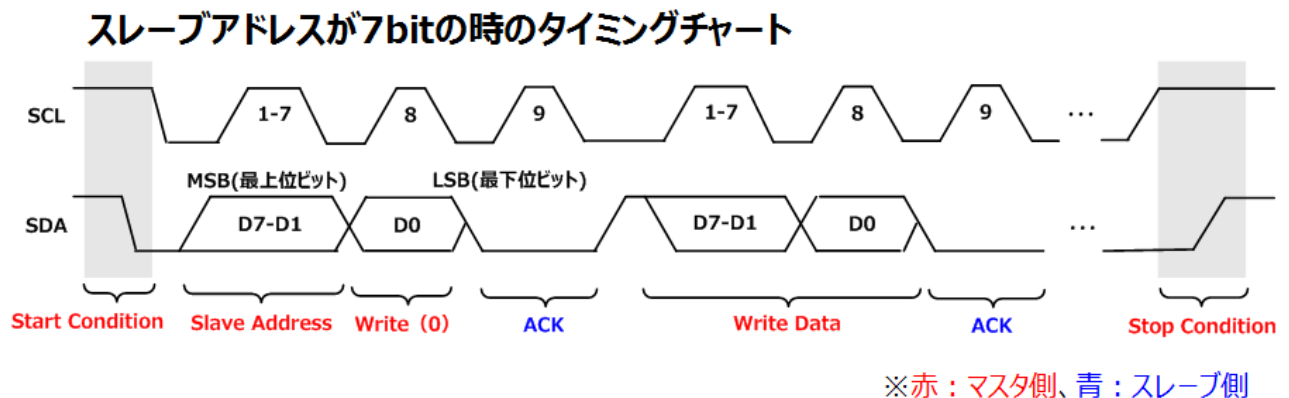


図 4.2.2.1 スレーブアドレス 7bit のデータ転送

4.2.2.1 データ仕様

I2C ドライバは初期化用の型として、表 4.2.2.1.12 の I2C コンフィギュレーション型と、ハンドラとして表 4.2.2.1.2 の I2C ハンドラ型を持つ。

| 番号 | 項目 | 型 | 機能 |
|----|-----------------|----------|-------------------------|
| 1 | ClockSpeed | uint32_t | 通信クロックスピード(bps) |
| 2 | AddressingMode | uint32_t | アドレスモード(7bit or 10bit) |
| 3 | OwnAddress1 | uint32_t | スレーブアドレス 1 (スレーブの場合のみ) |
| 4 | GeneralCallMode | uint32_t | ジェネラルコールモード設定 |
| 5 | SdaHoldClock | uint16_t | SDA のホールドクロック数 |
| 6 | restart_enable | uint8_t | マスタ リスタート有効設定 |
| 7 | fs_spklen | uint8_t | FS スパイク除去幅 |
| 8 | nack_enable | uint8_t | スレーブモード NACK 生成有効設定 |
| 9 | semid | int | 通信用セマフォ ID (0 でセマフォなし) |
| 10 | 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 HPS I2C コンフィギュレーション型

コンフィギュレーション型は、TOPPERS BASE PLATFORM(ST)に合わせた設計を行ったが、かなりの差異がある。これは、hps 用 I2C ペリフェラルの設計が ST マイクロエレクトロニクス社の設計とは大きく異なるためである。

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を使用するため、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 | 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 | pBuffPtr | uint8_t * | 通信データ領域へのポインタ |
| 4 | XferSize | uint16_t | 通信バイト数 |
| 5 | XferCount | volatile uint16_t | 通信済みバイト数 |
| 6 | XferFifo | volatile uint16_t | 転送 FIFO カウンタ |
| 7 | writcallback | void (*) 0 | 書き込みコールバック関数 |
| 8 | readcallback | void (*) 0 | 読み込みコールバック関数 |
| 9 | errorcallback | void (*) 0 | エラー発生時コールバック関数 |
| 10 | Status | ID | I2C ドライバの状態 |
| 11 | ErrorCode | volatile uint32_t | I2C エラーコード |
| 12 | PortID | volatile uint32_t | I2C ポート ID |
| 13 | Status | uint8_t | I2C リスタートモード(内部使用) |
| 14 | ErrorCode | uint8_t | I2C ストップモード(内部使用) |

表 4.2.2.1.2 I2C ハンドラ型

- ① ClockSpeed
転送スピードを周波数で設定する

- ② AddressingMode
I2C のアドレッシング・モードを設定する。

| 定義 | 値 | 内容 |
|--------------------------|------------|-----------|
| I2C_ADDRESSINGMODE_7BIT | 0x00000000 | 7 ビットモード |
| I2C_ADDRESSINGMODE_10BIT | 0x00000001 | 10 ビットモード |

表 4.2.2.1.3 AddressingMode 設定値

- ③ OwnAddress
自分のアドレス、マスタならばゼロを設定(設定されない)、スレーブならばスレーブアドレスを設定する。

- ④ GeneralCallMode
ジェネラルコールモード設定、無効ならばゼロを設定、有効ならば 1 を設定。

- ⑤ SdaHoldClock
SDA のホールドクロックを周波数で設定、レンジは 0~65535。

- ⑥ restart_enable
マスタ・リスタート設定(マスタのみ)、通常は有効。

| 定義 | 値 | 内容 |
|---------------------|-----------------------|---------|
| I2C_RESTART_DISABLE | 0x00000000 | リスタート無効 |
| I2C_RESTART_ENABLE | I2C_CON_IC_RESTART_EN | リスタート有効 |

表 4.2.2.1.4 restart_enable 設定値

- ⑦ fs_spklen
FS スパイクの除去幅を設定、レンジは 0~255。

- ⑧ nack_enable

スレーブモードのみ、NACK 有効無効設定、無効ならばゼロを設定、有効ならば 1 を設定。

4.2.2.2 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_slavewrite | ER | I2C_Handler* hi2c uint8_t *pData uint16_t Size | スレーブモードのデータ送信 | |
| i2c_slaveread | ER | I2C_Handler* hi2c uint8_t *pData uint16_t Size | スレーブモードのデータ受信 | |
| 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 | |
| i2c_int_enable | void | I2C_Handler* hi2c | I2C 割込み許可 | |
| i2c_int_disable | void | I2C_Handler* hi2c | I2C エラー禁止 | |
| i2c_hps_isr | void | intptr_t exinf | I2C イベント割込みサービスルーチン | |

表 4.2.2.2.1 I2C ドライバ関数

4.2.2.3 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。BASE PLATFORM を使用する環境ではマスタとして使用しますので、マスタからペリフェラルとの通信方法について記載します。基本的に、このドライバでは割込みを使用してデータの送受信を行います。また、スタート、ストップ、ACK 処理はペリフェラル側で処理しますので、マスタの場合、スレーブアドレスと送受信するデータ領域へのポインタと転送サイズを指定します。スレーブの場合、初期化でスレーブアドレスをセットして、送受信関数でバッファの設定を行います。PLATFORM はスレーブなることはないと思われますのでスレーブの説明は行いません。

図 4.2.2.3.1 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。

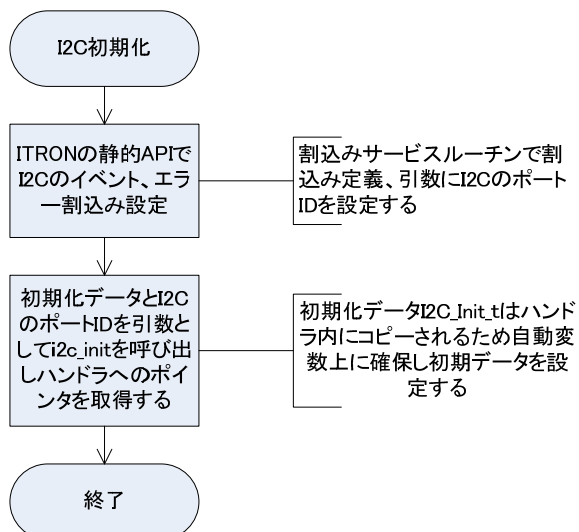


図 4.2.2.3.1 初期化フローチャート

図 4.2.2.3.2 にマスタのデータ送信のフローチャート、図 4.2.2.3.3 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、`MemAddrSize` 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。

I2Cの割込みはイベント割込みとエラー割込みがあり、イベント割込みはI2C内部のデータ遷移用に使用され、エラー発生時のみエラー割込みが発生します。エラー内容は、ハンドラの `ErrorCode` に設定される。`ErrorCode` がゼロの場合はエラーなしで、エラーが発生した場合の `ErrorCode` の値は表 4.2.2.3.1 の `ErrorCode` の内容に示す。

| 定義 | 値 | 内容 |
|--------------------------------|------------|---------------|
| <code>I2C_ERROR_NONE</code> | 0x00000000 | エラーなし |
| <code>I2C_ERROR_RXUNDR</code> | 0x00000001 | 受信アンダーラン |
| <code>I2C_ERROR_RXOVR</code> | 0x00000002 | 受信オーバーラン |
| <code>I2C_ERROR_TXABOUT</code> | 0x00000004 | 送信アボート |
| <code>I2C_ERROR_TXOVR</code> | 0x00000008 | 送信オーバーラン |
| <code>I2C_ERROR_TIMEOUT</code> | 0x00000020 | 転送タイムアウト（未実装） |
| <code>I2C_ERROR_SIZE</code> | 0x00000040 | 転送サイズエラー（未実装） |

表 4.2.2.3.1 ErrorCode の内容

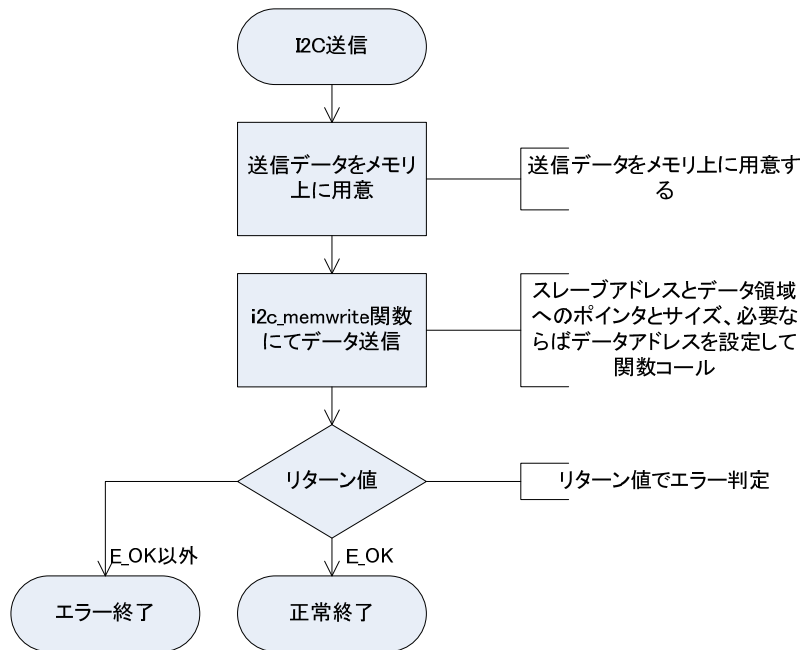


図 4.2.2.3.2 I2C 送信フローチャート

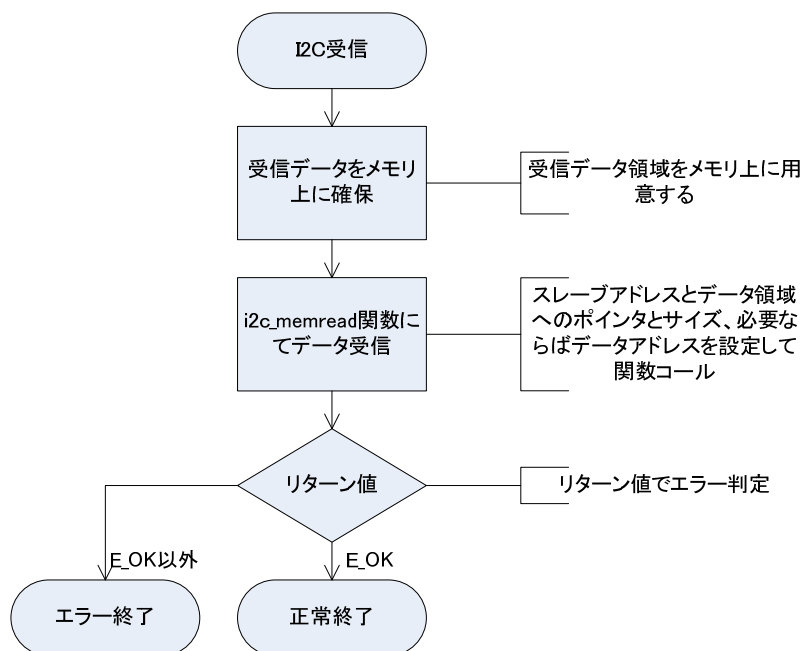


図 4.2.2.3.3 I2C 受信フローチャート

4.2.3 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。特に受信で使用する場合、HPS では FIFO フレームが 256 しかないため、それ以上のデータ受信ではポーリングや割込みではオーバーラン・エラーとなってしまう場合が多い、そのため、本デバイスドライバでは、割込み FIFO と DMA 転送を選択できる設計とした。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

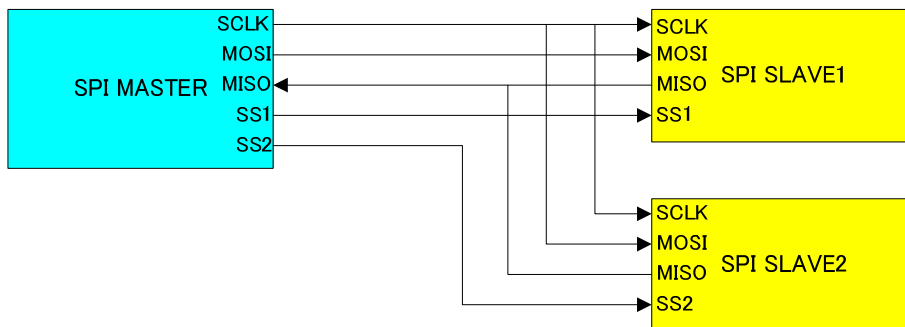


図 4.2.3.1 SPI 接続図

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。

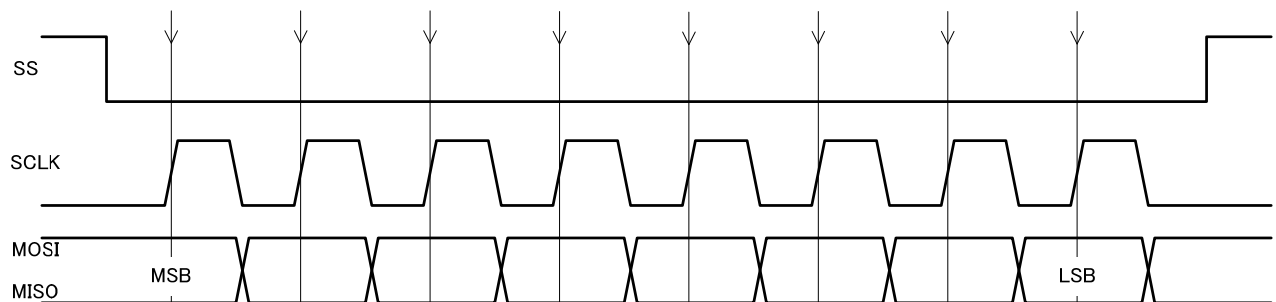


図 4.2.3.2 モード 0、正パルス、前縁ラッチ、後端シフトのタイミング図

4.2.3.1 データ仕様

SPI ドライバは初期化用の型として、表 4.2.3.1.1 の SPI コンフィギュレーション型と、ハンドラとして表 4.2.3.1.2 の SPI ハンドラ型を持つ。

| 番号 | 項目 | 型 | 機能 |
|----|---------------|----------|-------------------------------|
| 1 | DataSize | uint32_t | SPI 転送データサイズ |
| 2 | FrameMode | uint32_t | SPI 転送モード設定 |
| 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 | XferMode | uint32_t | SPI デフォルト転送モード |
| 6 | Speed | uint32_t | SPI 転送スピード設定 |
| 7 | RxSampleDelay | uint32_t | SPI Master : 受信サンプルデレイ |
| 8 | LoopBack | uint32_t | SPI Master : ループバックモード設定 |
| 9 | SlaveOut | uint32_t | SPI Slave : アウトプット・イネーブル |
| 10 | CtlFrameSize | uint32_t | SPI Microware : コントロールフレームサイズ |
| 11 | MwMode | uint32_t | SPI Microware : 転送モード |

| | | | |
|----|-------------|----------|----------------------------------|
| 12 | MwDir | uint32_t | SPI Microware : 転送方向 |
| 13 | MwHandshake | uint32_t | SPI Microware Master : ハンドシェイク設定 |
| 14 | semid | int | 通信用セマフォ ID (0 でセマフォなし) |
| 15 | 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 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を使用するため、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 | clock_freq | uint32_t | 送信データ領域へのポインタ |
| 4 | op_mode | uint16_t | マスタ、スレーブ設定 |
| 5 | xmode | uint16_t | 現在のマスタ SPI 転送モード |
| 6 | ptxbuff | uint8_t * | 送信データ領域へのポインタ |
| 7 | txxfersize | uint32_t | 送信フレーム数 |
| 8 | txxfersize | uint32_t | 送信済みフレーム数 |
| 9 | prxbuff | uint8_t * | 受信データ領域へのポインタ |
| 10 | rxxfersize | uint32_t | 受信フレーム数 |
| 11 | rxxfersize | uint32_t | 受信済みフレーム数 |
| 12 | hdmactx | DMACH_Handle_t* | 送信用 DMA ハンドラ |
| 13 | hdmachrx | DMACH_Handle_t* | 受信用 DMA ハンドラ |
| 14 | errorcode | volatile uint32_t | SPI エラーコード |

表 4.2.3.1.2 SPI ハンドラ型

- ① DataSize
SPI データ転送ビットサイズ、レンジは 4~16。

- ② FrameMode
SPI 通信方式のモード設定。

| 定義 | 値 | 内容 |
|-------------------|------------|-----------------|
| SPI_FRM_SPI | 0x00000000 | モトローラ SPI モード |
| SPI_FRM_SSP | 0x00000010 | TI 転送モード |
| SPI_FRM_MICROWIRE | 0x00000020 | Microwire 転送モード |

表 4.2.3.1.3 FrameMode 設定値

- ③ DataSize
SPI のフレーム中のビットサイズを指定する。レンジは 4 以上、16 以下。

- 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 | SPIM_CTLR0_SCPOL | 極性 HIGH |

表 4.2.3.1.4 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 | SPIM_CTLR0_SCPH | 後縁ラッチ |

表 4.2.3.1.5 CLKPhase 設定値

⑥ XferMode

SPI のデフォルトの転送モードを設定する。マスタの場合、転送関数にて転送モードの切り替えが可能である。スレーブの場合は切り替え不可である。

| 定義 | 値 | 内容 |
|---------------|------------|----------|
| SPI_XMOD_TXRX | 0x00000000 | 転送モード送受信 |
| SPI_XMOD_TX | 0x00000100 | 転送モード送信 |
| SPI_XMOD_RX | 0x00000200 | 転送モード受信 |

表 4.2.3.1.6 XferMode 設定値

⑦ Speed

SPI 転送周波数を設定する。

⑧ RxSampleDelay

マスタの場合、受信のサンプルデレイを設定する。レンジは 0 から 4 まで。

⑨ LoopBack

マスタの場合、ループバック設定を行える。

| 定義 | 値 | 内容 |
|-----------------|----------------|----------|
| SPI_LOOPBACKOFF | 0x00000000 | TI モード無効 |
| SPI_LOOPBACKON | SPIM_CTLR0_SRL | TI モード有効 |

表 4.2.3.1.7 LoopBack 設定値

⑩ SlaveOut

スレーブの場合、アウトプット有効無効設定、マスタからのブロードキャスト送信に対する応答動作を指定する。

| 定義 | 値 | 内容 |
|---------------------|-------------------|----------|
| SPI_SLAVEOUTDISABLE | 0x00000000 | アウトプット無効 |
| SPI_SLAVEOUTENABLE | SPIS_CTLR0_SLV_OE | アウトプット有効 |

表 4.2.3.1.8 SlaveOut 設定値

⑪ CtlFrameSize

FrameMode が Microwire 設定のみ有効、コントロールフレームサイズを設定する。レンジは 1 から 16 まで。

⑫ MwMode

FrameMode が Microwire 設定のみ有効、転送モードを設定する。

| 定義 | 値 | 内容 |
|-----------------------|-----------------|----------|
| SPI_MW_NON_SEQUENTIAL | 0x00000000 | 非シーケンシャル |
| SPI_MW_SEQUENTIAL | SPIM_MWCR_MWMOD | シーケンシャル |

表 4.2.3.1.9 MwMode 設定値

⑬ MwDir

FrameMode が Microwire 設定のみ有効、通信方向を指定する。

| 定義 | 値 | 内容 |
|---------------|---------------|----|
| SPI_MW_DIR_RX | 0x00000000 | 受信 |
| SPI_MW_DIR_TX | SPIM_MWCR_MDD | 送信 |

表 4.2.3.1.10 MwDir 設定値

⑭ MwHandshake

FrameMode が Mirocwire 設定で、マスタの場合のみ有効、ハンドシェイク設定を行う。

| 定義 | 値 | 内容 |
|-------------------------|---------------|----|
| SPI_MWHANDSHAKE_DISABLE | 0x00000000 | 無効 |
| SPI_MWHANDSHAKE_ENABLE | SPIM_MWCR_MHS | 有効 |

表 4.2.3.1.12 MwHandshake 設定値

4.2.3.2 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。

| 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_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_hps_isr | void | intptr_t exinf | SPI 割り込みサービスルーチン | |

表 4.2.3.2.1 SPI ドライバ関数

4.2.3.3 フローチャート

SPI ドライバは、FIFO による通信と DMA による通信を指定でき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。ハンドラ内の hdmactx に送信用 DMA チャンネルハンドラを設定した場合のみ、送信が DMA モードで行われます。また、ハンドラ内の hdmachrx に受信用 DMA ハンドラを設定した場合のみ、受信が DMA モードで行われます。したがって、FIFO と DMA を混在させた通信も可能です。DMA を使用する場合、転送領域をキャッシュのアラインになるように設定してください。

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_transrecv 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。マスタにてシリアル通信を行う場合、対象のペリフェラルの SS 信号の操作を、slave_select を用いて制御する。(スレーブの場合、この引数は意味をもちません)

SPI ハンドラ内の errorcode は表 4.2.3.3.1 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_TXOVR | SPIS_ISR_TXOIS | TX FIFO オーバーフロー・エラー |
| SPI_ERROR_RXFUND | SPIS_ISR_RXUIS | RX FIFO アンダーフロー・エラー |
| SPI_ERROR_RXFOVR | SPIS_ISR_RXOIS | RX FIFO オーバーラン・エラー |
| SPI_ERROR_DMA | 0x00010000 | DMA 転送エラー |

| | | |
|-------------------|------------|--------------|
| SPI_ERROR_TIMEOUT | 0x00020000 | ステート変更タイムアウト |
|-------------------|------------|--------------|

表 4.2.3.3.1 ErrorCode の内容

図 4.2.3.3.1 に SPI の初期化フローチャートを記載します。SPI 割り込み、通信と排他制御用セマフォの静的 API を用いて登録する。

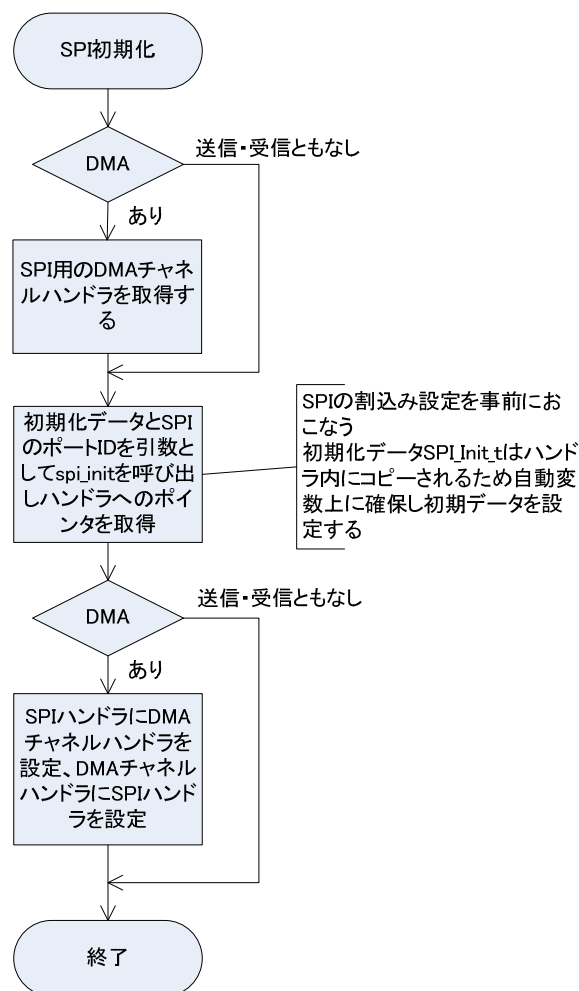


図 4.2.3.3.1 SPI 初期化フローチャート

図 4.2.3.3.2 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、送信のみのスレーブや受信のみのスレーブであっても、spi_tranrecv 関数で処理を代行できる。送信のみのスレーブ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、受信のみのスレーブでこの関数を代行した場合、設定値が送信されるが、スレーブ側では受信しない。

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。

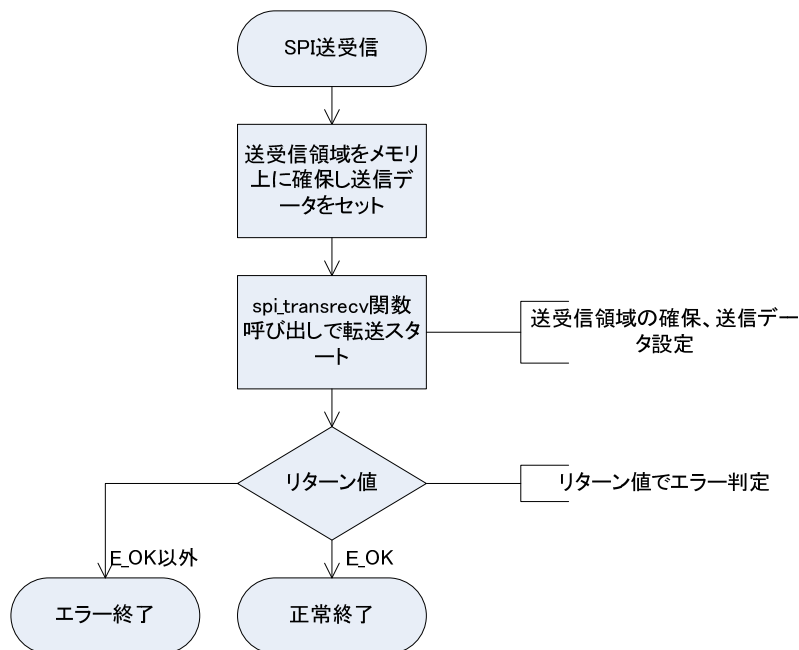


図 4.2.3.3.2 SPI 送受信フローチャート

4.2.4 SDMMC

SDMMC(SD/SDIO/MMC card host interface)はSDカードメモリやSDIOとのインターフェイスを行うペリフェラルである。マイクロSDカードソケットは各ボードに実装されているためSDカード、SDHCカードとのインターフェイスを行うように設計している。SDカードとの通信には送受信ともFIFOを使用する設定と受信のみDMAを使用する設定がある(SDMMC_READ_INNER_DMAコンパイルスイッチによる選択)。しかし、SDMMC非常に大きなFIFOがあり、DMAを使用しない場合、キャッシュ操作やアラインを考慮する必要なくデバイス制御を行えるため、FIFOモード(デフォルト)を推奨する。

4.2.4.1 データ仕様

SDMMCドライバはSDカード用に設計している。表4.2.4.1.1にSDMMCドライバ用のハンドラ型SDMMC_Handle_tを示す。ハンドラの設定は管理、参照用であり、ユーザーによる設定変更は許可しない。表4.2.4.1.2にSDカード情報を設定するSDMMC_CardInfo_t型を定義する。また、受信DMAを使用する場合、SDMMC_DMA_BUF_DESC_t型のDMA管理領域を確保しなければならないが、ここでは詳細を記載しない。

| 番号 | 項目 | 型 | 機能 |
|----|------------|----------|------------------------|
| 1 | base | uint32_t | SDMMCレジスタベースアドレス |
| 2 | ClockMode | uint32_t | 指定クロックモード(未使用) |
| 3 | BusWide | uint32_t | 指定バス幅 |
| 4 | RetryCount | uint32_t | リトライ回数 |
| 5 | cardtype | uint32_t | カード種類(保存用) |
| 6 | RCA | uint32_t | RCA値(保存用) |
| 7 | CSD[4] | uint32_t | CSD値(保存用) |
| 8 | CID[4] | uint32_t | CID値(保存用) |
| 9 | SCR[2] | uint32_t | SCR値(保存用) |
| 10 | CCC | uint32_t | CCC値(保存用) |
| 11 | block_size | uint32_t | READ/WRITEのブロックサイズ |
| 12 | high_speed | bool_t | 50MHzのhigh speedの許可不許可 |

| | | | |
|----|------------|-----------------------|----------------------------------|
| 13 | dma_enable | bool_t | DMAの有効無効 |
| 14 | pBuffPtr | uint8_t* | SDMMC 通信バッファへのポインタ |
| 15 | pCurDesc | SDMMC_DMA_BUF_DESC_t* | 現在の DMA_BUF へのポインタ (DMA 使用時のみ有効) |
| 16 | istatus | volatile uint32_t | 割込み状態 |
| 17 | dstatus | volatile uint32_t | DMA 実行状態 |
| 18 | status | volatile uint32_t | 転送状態 |
| 19 | SdCmd | volatile uint32_t | 転送指定コマンド |
| 20 | ErrorCode | volatile uint32_t | エラーコード |
| 21 | semid | ID | 通信用セマフォ値 |

表 4.2.4.1.1 SDMMC_Handle_t 型

| 番号 | 項目 | 型 | 機能 |
|----|-----------|----------|----------|
| 1 | capacity | uint64_t | 容量(バイト) |
| 2 | blocksize | uint32_t | ブロックサイズ |
| 3 | maxsector | uint32_t | ブロックの数 |
| 4 | RCA | uint16_t | SD RCA 値 |
| 5 | cardtype | uint8_t | カード種類 |
| 6 | status | uint8_t | ステータス |

表 4.3.5.1.2 SDMMC_CardInfo_t 型

① BusWidth

SDMMC の使用データバス幅を指定する。

| 定義 | 値 | 内容 |
|-------------------|---|-------------|
| SDMMC_BUS_WIDE_1B | 1 | 1 ビット |
| SDMMC_BUS_WIDE_4B | 4 | 4 ビット (設定値) |
| SDMMC_BUS_WIDE_8B | 8 | 8 ビット |

表 4.3.5.1.3 BusWidth 設定値

② RetryCount

エラー発生時のリトライカウントを設定する。32 回を設定している。

③ cardtype

カード情報取得後判定したカード種類

| 定義 | 値 | 内容 |
|------------------|---|------------------|
| SD_CARD_V11 | 0 | SD CARD V1.1 |
| SD_CARD_V20 | 1 | SD CARD V2.0 |
| SD_CARD_HC | 2 | SDHC CARD |
| MMC_CARD | 3 | MMC CARD(未対応) |
| SD_IO_CARD | 4 | SD IO CARD(未対応) |
| HS_MM_CARD | 5 | HS MMC CARD(未対応) |
| SD_IO_COMBO_CARD | 6 | SD IO CARD(未対応) |
| MMC_CARD_HC | 7 | MM HC CARD(未対応) |

表 4.3.5.1.4 cardtype 設定値

ドライバの返り値を ER コードで設定しているが、SDMMC 独自エラーが発生した場合の拡張定義を表 4.3.5.1.6 に示す。

| 定義 | 値 | 内容 |
|---------|-----|---------|
| E_SDCOM | -80 | コマンドエラー |

表 4.3.5.1.5 エラーコード拡張

4.2.4.2 インターフェイス仕様

SDMMC を設定するドライバ関数を以下に示す。ドライバは表 4.3.5.2.1 に示す基本動作を行うドライバと、表 4.3.5.2.2 の SD カードの初期設定を行うドライバがある。初期設定ドライバは後述の Storage Device Manager からカード検知時、一定の手順で呼び出しを行う。

| 関数名 | 型 | 引数 | 機能 | 備考 |
|---------------------|------------------|--|--|----|
| sdmmc_init | void | intptr_t exinf | SDMMC ハード初期化 | |
| sdmmc_sense | bool_t | int id | 指定 ID の SD card をセンスする。ある場合 true を返す | |
| sdmmc_open | SDMMC_Handle_t * | int id | 指定 ID の SDMMC ドライバをオープンする。戻り値はハンドラへのポインタ | |
| sdmmc_close | ER | SDMMC_handle_t *hsd | SDMMC ドライバをクローズする | |
| sdmmc_erase | ER | SDMMC_handle_t *hsd uint64_t startaddr uint64_t endaddr | SD card のイレースを行う | |
| sdmmc_blockread | ER | SDMMC_handle_t *hsd uint32_t *pbuff uint32_t ReadAddr uint32_t blocksize uint32_t num | SD card からブロック単位で READ する | |
| sdmmc_blockwrite | ER | SDMMC_handle_t *hsd uint32_t *pbuff uint32_t WriteAddr uint32_t blocksize uint32_t num | SD card へブロック単位で WRITE する | |
| sdmmc_wait_transfar | ER | SDMMC_handle_t *hsd uint32_t Timeout | READ/WRITE 転送待ちを行う | |
| sdmmc_hps_handler | void | void | 割込みハンドラ | |

表 4.3.5.2.1 SDMMC 基本ドライバ関数

| 関数名 | 型 | 引数 | 機能 | 備考 |
|---------------------|----|---|------------------------|----|
| sdmmc_getcardinfo | ER | SDMMC_handle_t *hsd SDMMC_CardInfo_t *pCinfo | コネクした SD card の情報を取り出す | |
| sdmmc_checkCID | ER | SDMMC_handle_t *hsd | CID を取得する | |
| sdmmc_setaddress | ER | SDMMC_handle_t *hsd | 相対アドレス(RCA)を取得する。 | |
| sdmmc_sendCSD | ER | SDMMC_handle_t *hsd | CSD を取得する。 | |
| sdmmc_select_card | ER | SDMMC_handle_t *hsd uint32_t addr | カードセレクトコマンドを発行する | |
| sdmmc_configuration | ER | SDMMC_handle_t *hsd | コンフィギュレーション設定を行う | |
| sdmmc_set_widebus | ER | SDMMC_handle_t *hsd | バス幅を設定する | |
| sdmmc_getstatus | ER | SDMMC_handle_t *hsd | SD カードステータスを取り込む。 | |

表 4.3.5.2.2 SDMMC 初期化ドライバ関数

4.2.4.3 設定手順

SDMMC のハードウェアの初期化は電源投入時、リセット後に sdmmc_init 関数を用いて行う。SD カードの場合、スロットにメディアがあるかないかで初期化手順が異なる。メディアの管理は後述の Storage Device Manager が行う。挿抜処理が可能なメディアの場合、Storage Device Manager は



500ms 周期に `mci_ses_por` 関数（仮想関数で SDMMC の場合、`sdmmc_init` が呼び出される）を呼び出してメディアの有無をチェックする。これ以降は、図 4.3.5.3.1 の初期化フローに従い、メディアが挿入された場合は、初期化処理を行う。初期化処理中にエラーが発生した場合は、メディア使用不可の設定が行われ、すべて正常に終了した場合、メディアが使用可能になる。使用可能な状態の場合、ファイルシステムを用いて、ファイル処理が可能になる。メディアが抜かれた場合は、`mci_cls_por` 関数（仮想関数で SDMMC の場合、`sdmmc_close` が呼び出される）を呼び出し、メディアを使用不可にする。

メディアの初期化が終了したと、FATFs のドライバ層を経由して、メディアに対するブロック単位の **READ/WRITE** と **IO** アクセスが発生する。図 4.3.5.3.2 に FATFs のドライバからのブロックリードのフローを示す。ブロックライトは実行関数を `mci_wri_blk(sdmmc_block_write)` に置き換えればよい。

なお、ブロックリード、ライトのバッファ領域のポインタが 4 バイトアラインとなっているが、FATFs のブロックリード、ライトドライバは 1 バイトアラインの領域にデータの読み書きを指定する。そのため、SDMMC ドライバのブロックリード、ライトは 1 バイトアラインの読み書きを有効にしている。

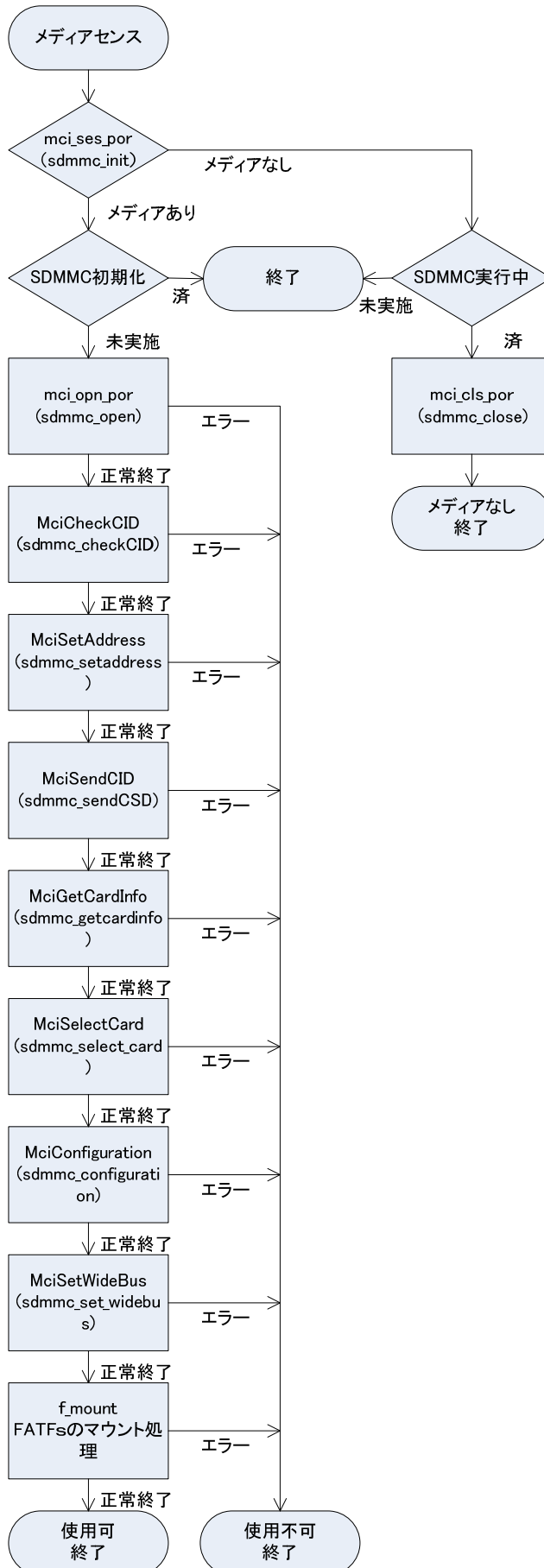


図 4.3.5.3.1 初期化フロー

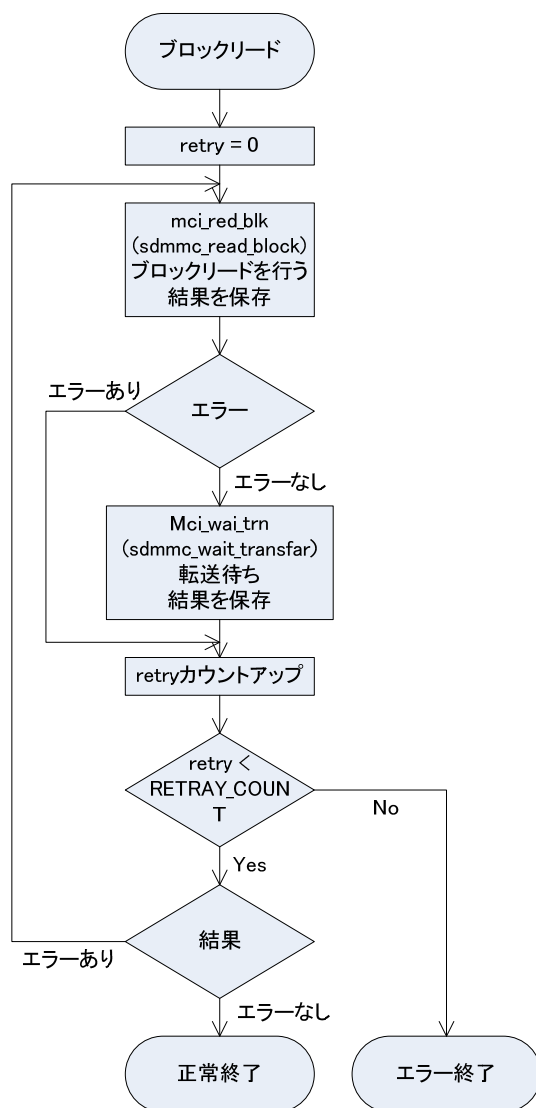


図 4.3.5.3.2 ドライバブロックリード

4.2.5 USB OTG

USB OTG は、ホスト、デバイスの USB 管理を行うドライバである。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 OTG ドライバは初期化用の型として、表 4.2.5.1.1 の USB OTG 初期設定定義型と、ハンドラとして表 4.2.5.1.4 の USB OTG ハンドラ型を持つ。内部の型としてエンドポイントを管理するための表 4.2.5.1.2 のエンドポイント定義型と、ホストチャネルを管理するための表 4.2.5.1.3 のホストクラス定義型をもつ。エンドポイントとホストクラス定義は、USB の上位層のモジュールで設定を行う必要がある。この実装では、TOPPERS USB ミドルウェアが設定を行っている。

初期設定定義型の有効無効設定は、1 が有効、0 が無効の設定となる。

| 番号 | 項目 | 型 | 機能 |
|----|---------------|----------|-----------------------|
| 1 | usb_otg_mode | uint32_t | USB OTG の実行モード指定 |
| 2 | dev_endpoints | uint32_t | デバイス用のエンドポイントの数(1-15) |
| 3 | host_channels | uint32_t | ホストチャネルの数(1-15) |

| | | | |
|---|------------|----------|--------------------------|
| 4 | speed | uint32_t | USB コアスピード |
| 5 | dma_enable | uint32_t | USB DMA 機能の有効無効設定 |
| 6 | phy_iface | uint32_t | PHY の指定 |
| 7 | sof_enable | uint32_t | デバイスモードでの SOF 割込みの有効無効設定 |
| 8 | spnlockid | uint32_t | スピンロック用の ID 指定 |

表 4.2.5.1.1 USB OTG 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 | data_pid_start | uint8_t:1 | スタートの PID(0 または 1) |
| 6 | even_off_frame | uint8_t:1 | フレームのパリティ(0:EVEN 1:ODD) |
| 7 | tx_fifo_num | uint16_t | 送信 FIFO 番号 |
| 8 | maxpacket | uint16_t | 最大パケットサイズ(64KB 以内) |
| 9 | xfer_buff | uint8_t * | 送受信バッファへのポインタ |
| 10 | dma_addr | uint32_t | DMA アドレス(4 バイトのアライン値) |
| 11 | xfer_len | uint32_t | 現在転送サイズ |
| 12 | xfer_count | uint32_t | 指定の転送サイズ |

表 4.2.5.1.2 エンドポイント定義型

| 番号 | 項目 | 型 | 機能 |
|----|--------------|-----------|------------------------------|
| 1 | dev_addr | uint8_t | デバイスアドレス(1-255) |
| 2 | ch_num | uint8_t | ホストチャネル番号(1-15) |
| 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(1:IN 0:OUT) |
| 5 | speed | uint8_t | USB ホストのスピード |
| 6 | do_ping | uint8_t | HS モード時の PING プロトコルの有効無効設定 |
| 7 | process_ping | uint8_t | PING プロトコル実行状態 |
| 8 | ep_type | uint8_t | エンドポイントの型 |
| 9 | max_paket | uint16_t | 最大パケットサイズ(64KB 以内) |
| 10 | data_pid | uint8_t | 初期設定データ PID |
| 11 | xfer_buff | uint8_t * | 通信バッファ領域へのポインタ |
| 12 | xfer_len | uint32_t | 現在の通信サイズ |
| 13 | xfer_count | uint32_t | 指定通信サイズ |
| 14 | toggle_in | uint8_t | IN 通信のトグルフラグ |
| 15 | toggle_out | uint8_t | OUT 通信のトグルフラグ |
| 16 | dma_addr | uint32_t | DMA モードでのデータアドレス(4 バイトのアライン) |
| 17 | err_count | uint32_t | エラー発生数 |
| 18 | urb_state | uint8_t | URB ステータス |
| 19 | state | uint8_t | ホストステータス |

表 4.2.5.1.2 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 | Init | USB_OTG_Init_t | USB-OTG 初期化設定パラメータ |
| 3 | hc[15] | USB_OTG_HCTypedef | ホストチャネル領域 |
| 4 | IN_ep[15] | USB_OTG_EPTypedef | IN エンドポイント領域 |
| 5 | OUT_ep[15] | USB_OTG_EPTypedef | OUT エンドポイント領域 |
| 6 | Setup[12] | uint32_t | セットアップパケット保存領域 |
| 7 | BESL | uint32_t | BESL データの保存領域 |

| | | | |
|----|-------------------------|------------|------------------------|
| 8 | lpm_state | uint8_t | LPM の状態 |
| 9 | lpm_active | uint8_t | LPM 機能有効無効設定 |
| 10 | hostsofcallback | void (*) 0 | ホスト用 SOF 割込みコールバック関数 |
| 11 | hostconnectcallback | void (*) 0 | ホストコネクト時コールバック関数 |
| 12 | hostdisconnectcallback | void (*) 0 | ホストディスコネクト時コールバック関数 |
| 13 | hostchangeurbcallback | void (*) 0 | ホスト URB 変更時コールバック関数 |
| 14 | devsetupstagecallback | void (*) 0 | デバイスセットアップステージコールバック関数 |
| 15 | devdataoutstagecallback | void (*) 0 | デバイスデータアウトステージコールバック関数 |
| 16 | devdatainstagecallback | void (*) 0 | デバイスデータインステージコールバック関数 |
| 17 | devsofcallback | void (*) 0 | デバイス SOF 割込みコールバック関数 |
| 18 | devresetcallback | void (*) 0 | デバイスリセットコールバック関数 |
| 19 | devsuspendcallback | void (*) 0 | デバイスサスペンドコールバック関数 |
| 20 | devresumecallback | void (*) 0 | デバイスレジュームコールバック関数 |
| 21 | devisooutcallback | void (*) 0 | デバイス ISOOUT コールバック関数 |
| 22 | devisoincallback | void (*) 0 | デバイス ISOIN コールバック関数 |
| 23 | devconnectcallback | void (*) 0 | デバイスコネクトコールバック関数 |
| 24 | devdisconnectcallback | void (*) 0 | デバイスディスコネクトコールバック関数 |
| 25 | devlpmcallback | void (*) 0 | デバイス LPM コールバック関数 |
| 26 | pHost | void * | 上位ホストハンドラ格納領域 |
| 27 | pDev | void * | 上位デバイスハンドラ格納領域 |

表 4.2.5.1.4 USB OTG ハンドラ型

① usb_otg_mode

USB OTG の設定モードを指定する。

| 定義 | 値 | 内容 |
|---------------------|---|---------------|
| USB_OTG_MODE_DEVICE | 0 | USB DEVICE 固定 |
| USB_OTG_MODE_HOST | 1 | USB HOST 固定 |
| USB_OTG_MODE_DRD | 2 | OTG モード |

表 4.2.5.1.5 usb_otg_mode 設定値

② speed

USB の転送スピードを設定する。

| 定義 | 値 | 内容 |
|------------------------|---|--------------------------|
| USB_SPEED_HIGH | 0 | HIGH スピード指定 |
| USB_SPEED_HIGH_IN_FULL | 3 | FULL スピード許可の HIGH スピード設定 |
| USB_SPEED_LOW | | LOW スピード |
| USB_SPEED_FULL | | FULL スピード |

表 4.2.5.1.6 speed 設定値

③ phy_iface

PHY 種別設定を行う。

| 定義 | 値 | 内容 |
|------------------|---|--------------|
| USB_PHY_ULPI | 1 | ULPI-PHY |
| USB_PHY_EMBEDDED | 2 | EMBEDDED-PHY |

表 4.2.5.1.7 phy_iface 設定値

④ spnlockid

スピンロックを行う場合、スピンロック ID を設定する。ゼロでスピンロックを行わない。

4.2.5.2 インターフェイス仕様

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 |
|--------------------------|-------------------|--|-------------------------------|----|
| usbo_init | USB_OTG_Handle_t* | ID portid USB_OTG_Init_t *pini | 指定ポート ID の USB-OTG ペリフェラルを初期化 | |
| usbo_deinit | ER | USB_OTG_Handle_t* husb | USB モジュールの無効化 | |
| usbo_getcurrentframe | uint32_t | USB_OTG_Handle_t* husb | 現在実行中のフレーム番号を取りだす | |
| usbo_setcurrentmode | ER | USB_OTG_Handle_t* husb | USB-OTG のモード設定を行う | |
| usbo_coreinit | ER | USB_OTG_Handle_t* husb | USB-OTG コアの初期化 | |
| usbo_enableglobalint | ER | USB_OTG_Handle_t* husb | USB-OTG グローバル割込みを有効に設定 | |
| usbo_disableglobalint | ER | USB_OTG_Handle_t* husb | USB-OTG グローバル割込みを無効に設定 | |
| usbo_flushTxFifo | ER | USB_OTG_Handle_t* husb uint32_t num | 送信 FIFO をフラッシュする | |
| usbo_flushRxFifo | ER | USB_OTG_Handle_t* husb | 受信 FIFO をフラッシュする | |
| usbo_initFifo | ER | USB_OTG_Handle_t* husb | 送信 FIFO 初期化設定 | |
| usbo_hc_init | ER | USB_OTG_Handle_t* husb uint8_t ch_num uint8_t eptnum | ホストチャネルを初期化する | |
| usbo_hc_startxfer | ER | USB_OTG_Handle_t* husb uint8_t ch_num | ホストチャネル送信要求 | |
| usbo_hc_halt | ER | USB_OTG_Handle_t* husb uint8_t ch_num | ホストチャネルを HALT 状態にする | |
| usbo_resetport | ER | USB_OTG_Handle_t* husb | ホストポートをリセットする | |
| usbo_drivevbus | ER | USB_OTG_Handle_t* husb uint8_t state | ホスト用 VBUS のオンオフ設定を行う | |
| usbo_hostinit | ER | USB_OTG_Handle_t* husb | ホストの初期化を行う | |
| usbo_starthost | ER | USB_OTG_Handle_t* husb | ホストを開始する | |
| usbo_stophost | ER | USB_OTG_Handle_t* husb | ホストを停止する | |
| usbo_gethostspeed | uint32_t | USB_OTG_Handle_t* husb | ホストのコアスピードを取り出す | |
| usbo_hcd_irqhandler | void | USB_OTG_Handle_t* husb | USB-OTG ホスト割込み処理 | |
| usbo_devconnect | ER | USB_OTG_Handle_t* husb | USB デバイス接続要求 | |
| usbo_devdisconnect | ER | USB_OTG_Handle_t* husb | USB デバイス切断要求 | |
| usbo_activateEndpoint | ER | USB_OTG_Handle_t* husb USB_OTG_EPTypedef *ep | エンドポイントアクティベート要求 | |
| usbo_disactivateEndpoint | ER | USB_OTG_Handle_t* husb USB_OTG_EPTypedef *ep | エンドポイントディスアクティベート要求 | |
| usbo_epsetStall | ER | USB_OTG_Handle_t* husb USB_OTG_EPTypedef *ep | エンドポイントをステイール状態に設定 | |
| usbo_epclearStall | ER | USB_OTG_Handle_t* husb USB_OTG_EPTypedef *ep | エンドポイントのステイール状態解除 | |

| | | | | |
|---------------------|---------|---|----------------------|-----|
| usbo_setDevAddress | ER | USB_OTG_Handle_t* husb uint8_t address | デバイスアドレスを設定する | |
| usbo_getDevSpeed | uint8_t | USB_OTG_Handle_t* husb | USB デバイススピードを取り出す | |
| usbo_ep0_outstart | ER | USB_OTG_Handle_t* husb uint8_t *psetup | エンドポイント 0-OUT 開始要求 | |
| usbo_ep0startxfer | ER | USB_OTG_Handle_t* husb USB_OTG_EPTypeDef* ep | エンドポイント0の転送開始要求 | |
| usbo_epstartxfer | ER | USB_OTG_Handle_t* husb USB_OTG_EPTypeDef* ep | エンドポイントの転送開始要求 | |
| usbo_devinit | ER | USB_OTG_Handle_t* husb | USB デバイス初期化 | |
| usbo_stopdevice | ER | USB_OTG_Handle_t* husb | USB デバイス停止 | |
| usbo_init_lpm | ER | USB_OTG_Handle_t* husb | LPM の初期化 | 未実装 |
| usbo_pcd_irqhandler | void | USB_OTG_Handle_t* husb | USB デバイスの割り込み処理 | |
| usb_otg_isr | void | intptr_t exinf | USB-OTG 割り込みサービスルーチン | |

表 4.2.5.2.1 USB-OTG ドライバ関数

4.2.5.3 フローチャート

USB-OTG 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) ホスト初期化

USB-OTG ハードウェア、上位モジュールの初期化を行い、USB ホストをスタートする。USB の状態遷移は割り込み関数からのコールバックで実行される。

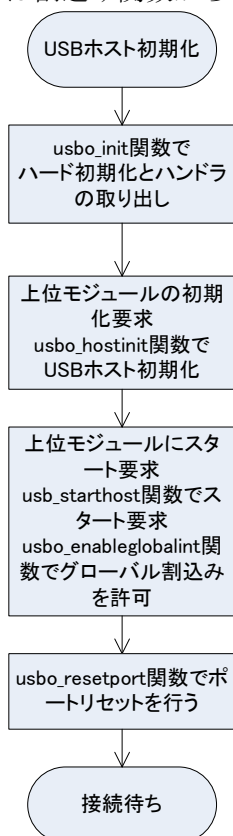


図 4.2.5.3.1 USB ホスト初期化

(2) ホスト接続とエナミュネーション

USB ホストの初期化が終了すると、デバイスの接続待ち状態に移行する。状態遷移は割込み関数中のコールバック関数で上位モジュールに伝達される。USB ホストで使用されるコールバック関数は以下の4つである

- ① **hostsofcallback**
SOF のタイミングで割込みを発生する。USB は 1ms 単位に SOF を発行するため 1ms 間隔にコールバックされる。
- ② **hostconnectcallback**
ホストのポート変化割込み(USB_OTG_GINTSTS_HPRTINT)が発生し、ポート接続チェックを行いコールバックする。
- ③ **hostdisconnectcallback**
ホストのディスコネクト割込み(USB_OTG_GINTSTS_DISCINT)が発生時、コールバックする。
- ④ **hostchangeurbcallback**
ホストチャンネル割込み(USB_OTG_GINTSTS_HCINT)が発生時、コールバックする。

接続の検知は **hostconnectcallback** 関数呼び出しから開始する。コールバック関数はイベント通知のみで、処理はタスクレベルで実行される。

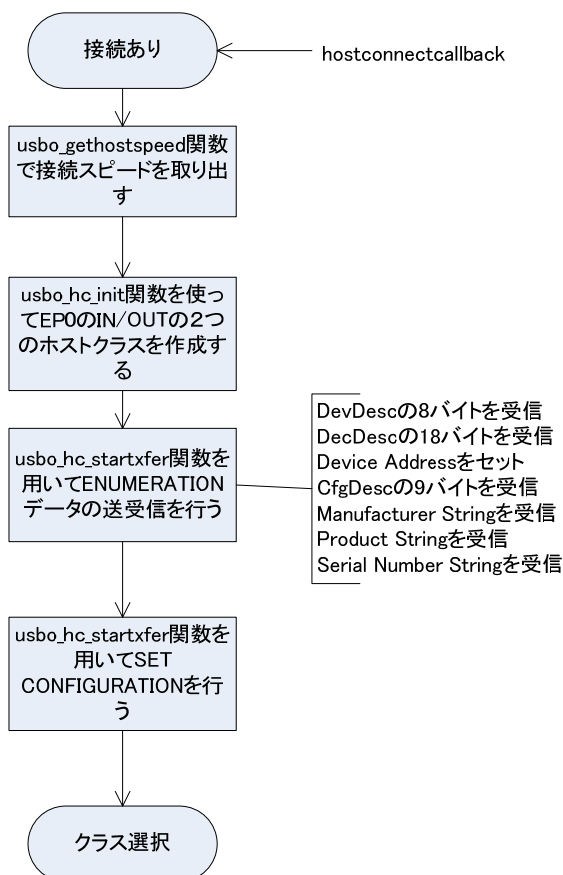


図 4.2.5.3.2 接続処理

(3) ホストクラスの選択と確認

上位モジュールでエナミュネーションにより取得した **bInterfaceClass** からクラスモジュールを選択する。対応するクラスモジュールがない場合は処理中止となる。クラスモジュールが決定された場合は、クラスモジュールで定められた手順で **Init** (通信用のエンドポイントの設定等) → **Requests** (クラスで使用するパラメータの取得等) を実行し、正常終了すればクラスモジュールの通常プロセスを

実行する。これらの処理はクラスモジュール毎に異なるので、統一したフローチャートで記載はできない。ここで状態遷移に用いるコールバック関数は `hostchangeurbcallback` 関数である。

(4)ホスト切断

切断が行われた場合、USB ホストの割込みから `hostdisconnectcallback` が行われる。このコールバックにより、上位モジュールが呼ばれ上位モジュールの切断手順が行われる。手順は以下のフローチャートの通りである。クラスモジュールの `Deinit` 移行はタスクレベルで行われ、終了後接続待ちとなる。

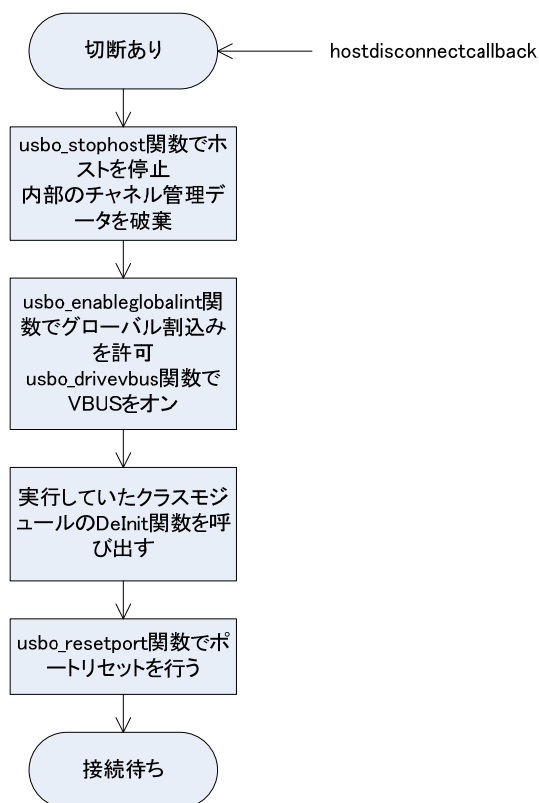


図 4.2.5.3.3 切断処理

デバイス側はホスト機能に対応する動作を行う。また、設定によっては、USB 割込み内で処理を完結可能である。状態の遷移は USB デバイスからのコールバックによって行われる。

- ① `devsetupstagecallback`
OUT エンドポイントの SETUP パケット通知割込み(USB_OTG_DOEPINT_STUP)からのコールバック
- ② `devdataoutstagecallback`
OUT エンドポイントの転送割込み(USB_OTG_DOEPINT_XFRC)からのコールバック
- ③ `devdatainstagecallback`
IN エンドポイントの送信終了割込み(USB_OTG_DIEPINT_XFRC)からのコールバック
- ④ `devsofcallback`
SOF 割込み(USB_OTG_GINTSTS_SOF)からのコールバック
- ⑤ `devresetcallback`
ENUMERATION 終了割込み(USB_OTG_GINTSTS_ENUMDNE)からのコールバック
- ⑥ `devsuspendcallback`
SUSPEND 割込み(USB_OTG_GINTSTS_USBSUSP)からのコールバック
- ⑦ `devresumecallback`
RESUME 割込み(USB_OTG_GINTSTS_WKUINT)からのコールバック
- ⑧ `devisooutcallback`
ISO-OUT 未完結割込み(USB_OTG_GINTSTS_PXFR_INCOMPISOOUT)からのコールバック
- ⑨ `devisoincallback`

ISO-IN 未完結割込み(USB_OTG_GINTSTS_IISOIXFR)からのコールバック

- ⑩ devconnectcallback
コネクション割込み(USB_OTG_GINTSTS_SRQINT)からのコールバック
- ⑪ devdisconnectcallback
ディスコネクション割込み(USB_OTG_GINTSTS_OTGINT)からのコールバック
- ⑫ devlpmcallback
LPM をサポートする場合の状態遷移コールバック

(5) デバイス初期化

USB デバイスの初期化手順を以下の表に示す。

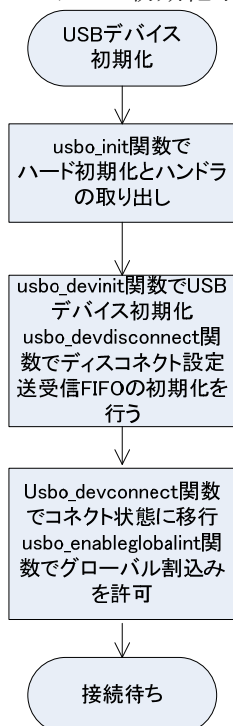


図 4.2.5.3.4 USB デバイスの初期化

(6) デバイス接続とエナミュネーション

USB の接続は、SUSPEND、RESUME のコールバックが何度か続いた後、リセットコールバック (devresetcallback)により、接続を開始する。これも、接続の状態で何度と発生する可能性がある。まず、エナミュレーションを行うためにエンドポイント 0 の IN/OUT を作成する。作成後、以下のコールバックが発生し、内容に従って、デバイス情報の通信を行う。

- ① devsetupstagecallback SETUP ステージの要求
- ② devdataoutstagecallback データ受信要求
- ③ devdatainstagecallback データ送信要求

最後に、SET CONFIGURATION を受信したら、USB クラスの通信に移行する。

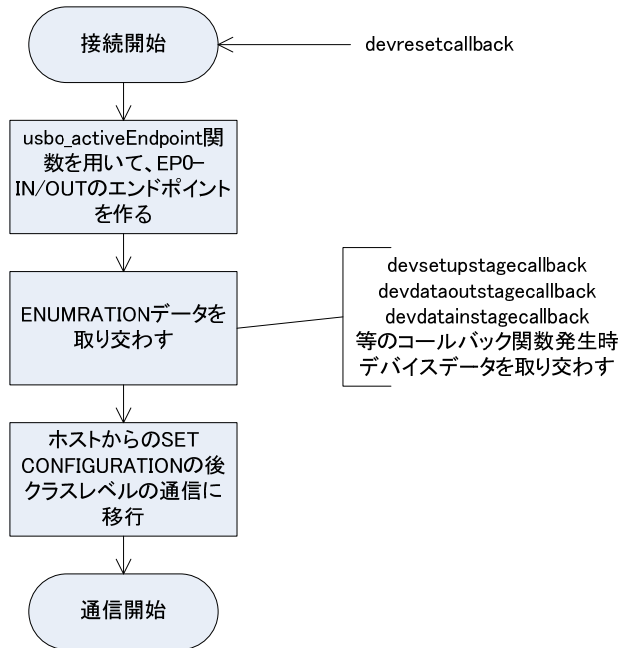


図 4.2.5.3.5 USB デバイスの接続

(7)SUSPEND と RESUME

USB デバイスの場合、通信確立後も SUSPEND と RESUME の要求により、必要な電源操作を行わなければならない。これらの移行は以下のコールバック関数によって要求される。切断、接続時も、これらのコールバック関数が呼ばれるので注意が必要である。

- ① devsuspendcallback 省エネモード要求
- ② devresumecallback 省エネ復帰要求

(8)デバイス切断

USB デバイスの切断は切断コールバック (devdisconnectcallback) の呼び出しにより行う。このコールバックの前に SUSPEND もコールバックも発生する。

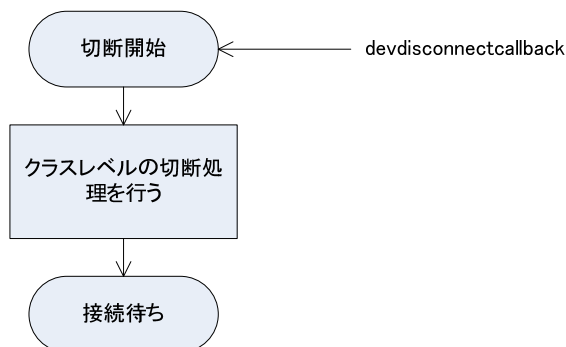


図 4.2.5.3.6 USB デバイスの切断

4.2.6 QSPI

QSPI は、フラッシュ ROM 等に用いられる SPI ドライバである。

4.2.6.1 データ仕様

QSPI ドライバは初期化用の型として、表 4.2.6.1.1 の QSPI コンフィギュレーション構造体と、ハンドラとして表 4.2.6.1.2 の QSPI ハンドラ型を持つ。QSPI コンフィギュレーション構造体は QSPI 対応のフラッシュ ROM の処理属性を設定する。

| 番号 | 項目 | 型 | 機能 |
|----|----------------------|------------------|----------------|
| 1 | mauf_id | uint16_t | マニファクチャ ID |
| 2 | type_capacity | uint16_t | キャパシティタイプ ID |
| 3 | clk_phase | uint32_t | クロックフェーズ |
| 4 | clk_pol | uint32_t | QSPI 転送クロックの極性 |
| 5 | max_frq_mhz | uint32_t | 最大周波数(MHZ) |
| 6 | tshsl_ns | uint32_t | tshsl(ns) |
| 7 | tsd2d_ns | uint32_t | tsd2f(ns) |
| 8 | tchsh_ns | uint32_t | tchsh(ns) |
| 9 | tslch_ns | uint32_t | tslch(ns) |
| 10 | page_size | uint32_t | ページサイズ |
| 11 | addr_size | uint32_t | アドレスサイズ |
| 12 | inst_type | uint8_t | インストラクションタイプ |
| 13 | read_op_code | uint32_t | 読み出し用オペコード |
| 14 | read_addr_xfer_type | uint8_t | 読み出しアドレス転送タイプ |
| 15 | read_data_xfer_type | uint8_t | 読み出しデータ転送タイプ |
| 16 | read_dummy_cycles | uint32_t | 読み出しダミーサイクル値 |
| 17 | write_op_code | uint32_t | 書き込み用オペコード |
| 18 | write_addr_xfer_type | uint8_t | 書き込みアドレス転送タイプ |
| 19 | write_data_xfer_type | uint8_t | 書き込みデータ転送タイプ |
| 20 | write_dummy_cycles | uint32_t | 書き込みダミーサイクル値 |
| 21 | erase_count | uint8_t | 消去オペレーション数 |
| 22 | erase_size[4] | uint32_t | 消去最小サイズ |
| 23 | erase_cmds[4] | uint8_t | 消去コマンド |
| 24 | erase_sector_idx | uint8_t | 消去セクタインデックス |
| 25 | support_chip_erase | bool_t | チップ消去機能の有無 |
| 26 | device_size | uint32_t | デバイスサイズ |
| 27 | micron_multi_die | bool_t | |
| 28 | die_size | uint32_t | ダイサイズ |
| 29 | init_func | QSPI_INIT_FUNC_T | 初期化関数 |
| 30 | wait_func | QSPI_WAIT_FUNC_T | 待ち関数 |
| 31 | friendly_name | const char * | デバイス名へのポインタ |

表 4.2.6.1.1 QSPI コンフィギュレーション構造体

| 番号 | 項目 | 型 | 機能 |
|----|-----------|---------------------|-------------------|
| 1 | base | uint32_t | QSPI モジュールベースアドレス |
| 2 | clk_req | uint32_t | QSPI モジュール実行クロック |
| 3 | Init | QSPI_Init_t | QSPI 初期設定値保存領域 |
| 4 | pBuffPtr | uint32_t * | QSPI 通信バッファへのポインタ |
| 5 | XferSize | uint32_t | QSPI 通信サイズ |
| 6 | XferCount | volatile uint32_t | QSPI 通信カウンタ |
| 7 | istatus | volatile uint32_t | QSPI 割込みステータス |
| 8 | semid | int | 通信用セマフォ ID |
| 9 | errorcode | volatile uint32_t * | QSPI エラーコード |

表 4.2.6.1.2 QSPI ハンドラ型

① `manuf_id`

マニファクチャ ID を指定する。

| 定義 | 値 | 内容 |
|----|---|----|
|----|---|----|

| | | |
|---------------------------------|------|-------------|
| QSPI_STIG_RDID_JEDECID_MICRON | 0x20 | Micron |
| QSPI_STIG_RDID_JEDECID_SPANSION | 0x01 | Spansion |
| QSPI_STIG_RDID_JEDECID_MACRONIX | 0xC2 | Macronix 拡張 |

表 4.2.6.1.3 `manuf_id` 設定

② `type_capacity`

キャパシティタイプを指定する。

例) Micron N25Q128 0x18Ba

Micron N25Q00AA 0x21Ba

Micron N25Q512A 0x20BA

③ `clk_phase`

クロックフェーズを指定する。0:アクト、1:インアクト。

④ `clk_pol`

QSPI 転送クロック極性定義。0:HIGH、1:LOW。

⑤ `inst_type`

インストラクションタイプを指定する。

| 定義 | 値 | 内容 |
|------------------|---|---------|
| QSPI_MODE_SINGLE | 0 | シングルモード |
| QSPI_MODE_DUAL | 1 | デュアルモード |
| QSPI_MODE_QUAD | 2 | クアッドモード |

表 4.2.6.1.4 `inst_type` 設定値

⑥ `read_op_code`

デバイス依存の読み出しフラッシュコマンド設定。

⑦ `read_addr_xfer_type/read_data_xfer_type`

デバイス依存の読み出しのアドレスタイプとデータタイプを指定する。設定は `inst_type` と同じ属性。

⑧ `read_dummy_cycles`

デバイス依存の読み出しダミーサイクルを指定する。レンジは 0 から 31 まで。

⑨ `write_op_code`

デバイス依存の書き込みフラッシュコマンド設定。

⑩ `write_addr_xfer_type/write_data_xfer_type`

デバイス依存の読み出しのアドレスタイプとデータタイプを指定する。設定は `inst_type` と同じ属性。

⑪ `write_dummy_cycles`

デバイス依存の読み出しダミーサイクルを指定する。レンジは 0 から 31 まで。

⑫ `erase_count`

消去コマンド数を指定。(4 以下)

⑬ `erase_sizes`

消去最小単位を指定。

⑭ `erase_cmds`

消去用のフラッシュコマンドを指定

⑮ erase_sector_idx
セクタ消去用のインデックスを指定。

⑯ init_func
フラッシュデバイスの初期化関数を設定。NULL ならば初期化処理を行わない。

⑰ wait_func
書き込み、または、消去時の待ち関数を指定。NULL ならば待ち処理を行わない。

4.2.6.2 インターフェイス仕様

QSPI を用いてフラッシュ ROM の制御するドライバ関数は以下の通りである。このドライバ関数に加えて init_func や wait_func からフラッシュコマンドを発行するためのドライバを表 4.2.6.2.2 に示す。

| 関数名 | 型 | 引数 | 機能 | 備考 |
|----------------------------|----------------|---|---|----|
| qspi_init | QSPI_Handle_t* | ID portid QSPI_Init_t *spii | 指定ポート ID の QSPI ペリフェラルを初期化し、ハンドラへのポインタを返す | |
| qspi_deinit | ER | QSPI_Handle_t* hqspi | QSPI を未使用状態に戻す | |
| qspi_read | ER | QSPI_Handle_t* hqspi void *dest uint32_t src uint32_t size | 非ダイレクトモードでフラッシュ ROM のデータを読み出す | |
| qspi_write | ER | QSPI_Handle_t* hqspi uint32_t dest const void *src uint32_t size | フラッシュ ROM にデータの書き込みを行う | |
| qspi_erase | ER | QSPI_Handle_t* hqspi uint32_t address uint32_t size | フラッシュ ROM の消去。 | |
| qspi_erase_exec | ER | QSPI_Handle_t* hqspi uint32_t address uint32_t idx | インデックス単位にフラッシュ ROM を消去する | |
| qspi_direct_disable | void | QSPI_Handle_t* hqspi | ダイレクト読み出し無効 | |
| qspi_direct_enable | void | QSPI_Handle_t* hqspi uint32_t map_addr | ダイレクト読み出し有効 | |
| qspi_ahb_remap_address_get | uint32_t | QSPI_Handle_t* hqspi | 現在のダイレクト読み出しのベースオフセットを取り出す | |
| spi_hps_isr | void | intptr_t exinf | QSPI 割込みサービスルーチン | |

表 4.2.6.2.1 QSPI ドライバ関数

| 関数名 | 型 | 引数 | 機能 | 備考 |
|-----------------------------|------|---|-----------------|----|
| qspi_chip_select_config_get | ER | QSPI_Handle_t* hqspi uint32_t *pcs uint8_t pcmode | 現在の CS モードの取り出し | |
| qspi_chip_select_config_set | ER | QSPI_Handle_t* hqspi uint32_t cs uint8_t cs_mode | CS モード設定 | |
| qspi_mode_bit_disable | void | QSPI_Handle_t* hqspi | モードビットを無効化 | |
| qspi_mode_bit_enable | void | QSPI_Handle_t* hqspi | モードビットを有効化 | |
| qspi_mode_bit_config_set | ER | QSPI_Handle_t* hqspi const uint32_t mode_bit | フラッシュ ROM の消去。 | |

| | | | | |
|-------------------------|----|--|------------------------------|--|
| qspi_sram_partition_set | ER | QSPI_Handle_t* hqspi const uint32_t part_size | SRAM 中の読み出しパーティ ション数を設定する | |
|-------------------------|----|--|------------------------------|--|

表 4.2.6.2.2 QSPI 拡張ドライバ関数

4.2.6.3 フローチャート

QSPI ドライバは、FIFO による通信のみをサポートする。QSPI の初期設定は、qspi_init 関数を指定して対象ポート番号（1のみ）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。コンフィギュレーション構造体は実装されたフラッシュ ROM の属性にあった設定を行わなければならない。QSPI の初期化フローチャートを図 4.2.6.3.1 に示す。

初期化後は、非ダイレクト読み込み、書き込み、消去機能が有効となる。ダイレクト読み込み機能は qspi_direct_enable 関数の実行で有効となる。フラッシュ ROM が参照できるメモリアドレスは HPS の仕様で固定となっており、アドレスが 0xffa00000 から 1MB である。

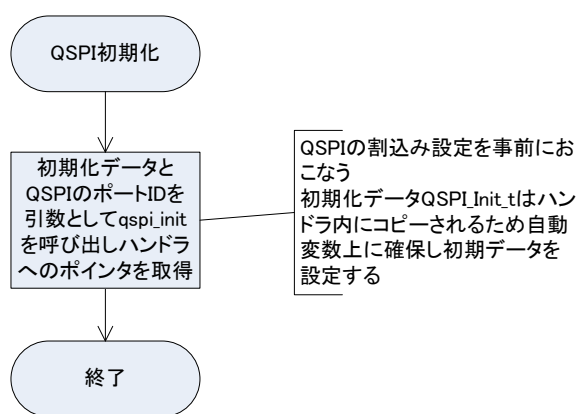


図 4.2.6.3.1 QSPI 初期化

4.2.7 EMAC

EMAC は、有線インサートネット用ドライバである。EMAC ドライバはパケットのインターフェイスとして LWIP 用 PBUF を想定して、設計している。パケット・インターフェイスを修正すれば、他の TCP/IP プロトコル・スタックに対応可能である。

4.2.7.1 データ仕様

EMAC ドライバは初期化用の型として、表 4.2.7.1.1 の EMAC コンフィギュレーション構造体と、ハンドラとして表 4.2.7.1.2 の EMAC ハンドラ型を持つ。EMAC コンフィギュレーション構造体は DESCRIPTOR キューと送受信バッファ等の設定を定義する。EMAC ドライバは、この設定に従って DESCRIPTOR を設定する。

| 番号 | 項目 | 型 | 機能 |
|----|---------------|----------|-----------------------|
| 1 | packetbufsize | uint32_t | パケットバッファ長 |
| 2 | rxquecount | uint32_t | 受信キューの数 |
| 3 | txquecount | uint32_t | 送信キューの数 |
| 4 | quebuffer | void * | キューバッファ領域へのポインタ |
| 5 | databuffer | void * | データ領域へのポインタ |
| 6 | semid | ID | mdio 用セマフォ ID(LOCK 用) |

表 4.2.7.1.1 EMAC コンフィギュレーション構造体

| 番号 | 項目 | 型 | 機能 |
|----|------|----------|-------------------|
| 1 | base | uint32_t | EMAC モジュールベースアドレス |

| | | | |
|----|-----------------------------|---------------------|--------------------------|
| 2 | Init | EMAC_Init_t | EMAC 初期設定値保存領域 |
| 3 | tx_wait | uint8_t | 送信終了待ちフラグ |
| 4 | intno | uint8_t * | EMAC 割り込み番号 |
| 5 | macid | uint8_t | PORTID 番号 (保存用) |
| 6 | phy_addr | uint8_t | PHY アドレス番号 |
| 7 | phy_features | uint32_t | PHY の機能ステータス |
| 8 | linkvalue | uint32_t | 要求 LINK 値 |
| 9 | tx_top_desc | volatile uint32_t * | 送信キュー(DESCRIPTOR)の先頭アドレス |
| 10 | tx_cur_desc | volatile uint32_t * | 現在の送信キューのアドレス |
| 11 | rx_top_desc | volatile uint32_t * | 受信キュー(DESCRIPTOR)の先頭アドレス |
| 12 | rx_cur_desc | volatile uint32_t * | 現在の受信キューのアドレス |
| 13 | emacevent | void (*) 0 | EMAC 状態のコールバック関数 |
| 14 | phy_opt_init | void (*) 0 | PHY のオプション設定関数 |
| 15 | ifShortPkts | uint32_t | ショートパケットの受信回数 |
| 16 | mmc_tx_irq_n | uint32_t | MMC 割り込み MMCTIS の発生回数 |
| 17 | mmc_rx_irq_n | uint32_t | MMC 割り込み MMCRIS の発生回数 |
| 18 | mmc_rx_csum_offload_irq_n | uint32_t | MMC 割り込み MMCCSUM の発生回数 |
| 19 | irq_receive_pmt_irq_n | uint32_t | PMT 割り込みの発生回数 |
| 20 | irq_tx_path_in_lpi_mode_n | uint32_t | LPI 割り込み TLPEN の発生回数 |
| 21 | irq_tx_path_exit_lpi_mode_n | uint32_t | LPI 割り込み TLPEX の発生回数 |
| 22 | irq_rx_path_in_lpi_mode_n | uint32_t | LPI 割り込み RLPIEN の発生回数 |
| 23 | irq_rx_path_exit_lpi_mode_n | uint32_t | LPI 割り込み RLPIEX の発生回数 |
| 24 | normal_irq_n | uint32_t | TX/RX NORMAL 割り込みの発生回数 |
| 25 | fatal_bus_error_irq | uint32_t | ABNORMAL/FBI の発生回数 |
| 26 | tx_undeflow_irq | uint32_t | ABNORMAL/UNF の発生回数 |
| 27 | tx_jabber_irq | uint32_t | ABNORMAL/TJT の発生回数 |
| 28 | tx_early_irq | uint32_t | ABNORMAL/ETI の発生回数 |
| 29 | tx_process_stopped_irq | uint32_t | ABNORMAL/TPS の発生回数 |
| 30 | rx_normal_irq_n | uint32_t | TX/RX NORMAL/RIE の発生回数 |
| 31 | rx_early_irq | uint32_t | TX/RX NORMAL/ERI の発生回数 |
| 32 | rx_overflow_irq | uint32_t | ABNORMAL/OVF の発生回数 |
| 33 | rx_buf_unav_irq | uint32_t | ABNORMAL/RU の発生回数 |
| 34 | rx_process_stopped_irq | uint32_t | ABNORMAL/RPS の発生回数 |
| 35 | rx_watchdog_irq | uint32_t | ABNORMAL/RWT の発生回数 |

表 4.2.7.1.2 EMAC ハンドラ型

① packetbufsize

パケットバッファサイズを指定する。キャッシュアラインサイズ出なければならない。

② rxquecount

受信用のキュー(DESCRIPTOR)の数を指定する。

③ txquesize

送信用のキュー(DESCRIPTOR)の数を指定する。

④ quebuffer

EMAC で使用するキュー(DESCRIPTOR)領域のポインタを指定する。この領域はキャッシュアライン・アドレスでなければならない。quebuffer のバイトサイズは以下の計算による。

$$\text{que_size} = (\text{rxquecount} + \text{txreqcount}) \times 4$$

⑤ databuffer

EMAC で使用する通信用のバッファ領域のポインタを指定する。この領域はキャッシュアライン・アドレスでなければならない。databuffer のバイトサイズは以下の計算による。
 $data_size = (rxquecount + txquecount) \times packetbufsize$

⑥ semid

PHY との通信時、排他制御を行うためのセマフォ ID。ゼロなら排他制御を行わない。

4.2.7.2 インターフェイス仕様

EMAC を用いてイーサネット通信を行うための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 |
|--------------------|----------------|--|---------------------------------------|----|
| emac_init | EMAC_Handle_t* | ID portid EMAC_Init_t *ini | 指定ポートIDのEMACペリフェラルを初期化し、ハンドラへのポインタを返す | |
| emac_deinit | ER | EMAC_Handle_t* hmac | EMAC を未使用状態に戻す | |
| emac_send | ER | EMAC_Handle_t* hmac struct emac_pkt *pktp | パケットのデータを送信する | |
| emac_rcv_length | int | EMAC_Handle_t* hmac | 受信パケット長を取り出す | |
| emac_rcv | ER | EMAC_Handle_t* hmac struct emac_pkt *pktp | 受信パケットを取り出す | |
| emac_start | ER | EMAC_Handle_t* hmac uint32_t link | EMAC と PHY をスタートする | |
| emac_stop | ER | EMAC_Handle_t* hmac int disable | EMAC を停止する | |
| emac_link_detect | uint16_t | EMAC_Handle_t* hmac | LINK 情報を取り出す | |
| emac_link_mode_set | ER | EMAC_Handle_t* hmac uint32_t link | LINK 情報を EMAC に設定する | |
| emac_reset | ER | EMAC_Handle_t* hmac uint8_t *mac_addr | EMAC をリセットする | |
| emac_hps_isr | void | intptr_t exinf | EMAC 割り込みサービスルーチン | |

表 4.2.7.2.1 EMAC ドライバ関数

4.2.7.3 フローチャート

EMAC ドライバは、TCP/IP プロトコル・スタックから呼び出されることを前提に設計している。TCP/IP プロトコル・スタックとの依存性が高い部分は、送受信のパケットの受け渡し処理である。このドライバでは、LWIP 用の pbuf を使用してパケットを受け渡すことを前提に設計している。

EMAC の初期化フローを図 4.2.7.3.1 に示す。ここで特に注意が必要なのは、EMAC モジュールはリセット後、割り込みマスクが許可に以降するため、必ず、EMAC の割り込みコントローラは割り込み禁止の状態システム起動しなければならない。emac_init にて EMAC の初期化を行う。このとき、EMAC_Init_t 構造体にて、キューや送受信バッファ領域の設定を行う。Socfpga_cv は、ノンキャッシュメモリはないため、キューや送受信制御にキャッシュコントロールを伴う。これらの領域はキャッシュアライン・アドレスの設定でなければならない。初期化後、emac_reset 関数にて EMAC のリセットを行う。この関数は通信中に EMAC のハードエラー等が発生した場合、EMAC のリセットに使用かうである。リセット後、emac_start 関数にて LINK 情報を設定して、PHY のスタート、EMAC のスタートを設定する。emac_start を実行しても、LINK していない場合、通信動作が開始できないばかりでなく、PHY の設定が AUTO NEGOTIATION の場合、EMAC の通信設定も確定しない。emac_link_detect 関数にて接続情報を取得できる。AUTO NEGOTIATION の場合、接続情報を EMAC に反映するために、emac_link_mode_set 関数を使用する。

送信処理のフローを図 4.2.7.3.2 に示す。pbuf に送信データをつめて、emac_send 関数を呼び出せばよい。

受信処理のフローを図 4.2.7.3.3 に示す。受信データの有無は、`emac_rcv_length` 関数で受信データサイズを取り出せる。受信データがある場合は、サイズ分の `pbuf` を用意して `emac_rcv` 関数にて、受信データの取得を行う。受信の有無はコールバック関数のイベントでも通知されるので、常に `emac_rcv_length` 関数でポーリングを行う必要はない。

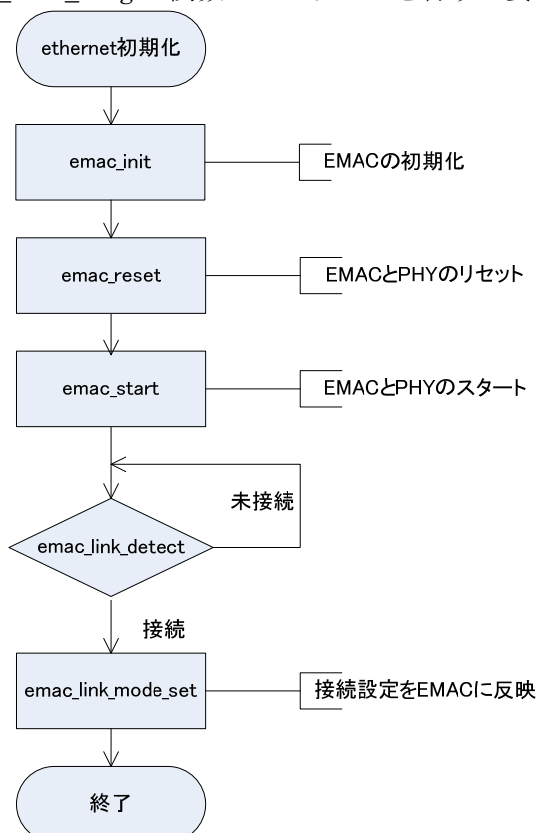


図 4.2.7.3.1 QSPI 初期化

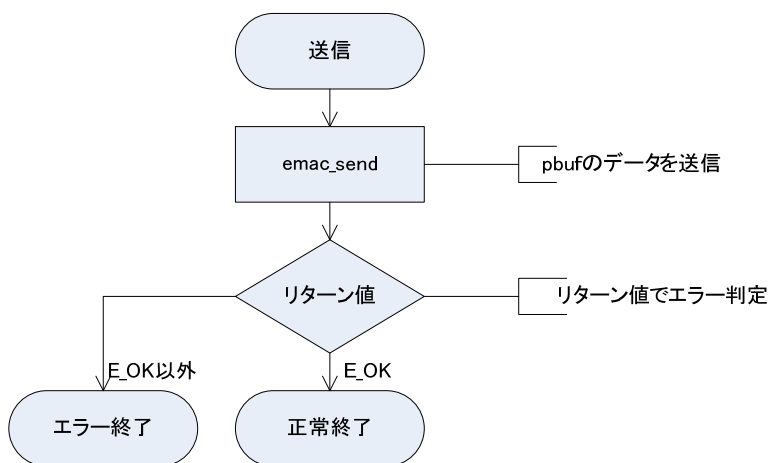


図 4.2.7.3.2 EAC 送信

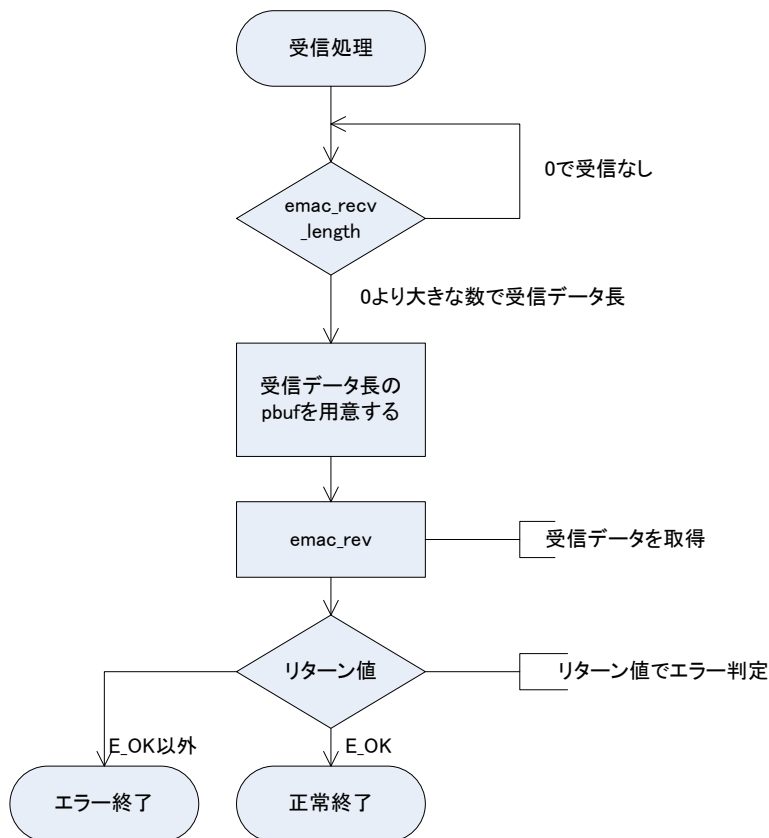


図 4.2.7.3.3 EMAC 受信

図 4.2.7.3.4 に EMAC の終了処理を示す。

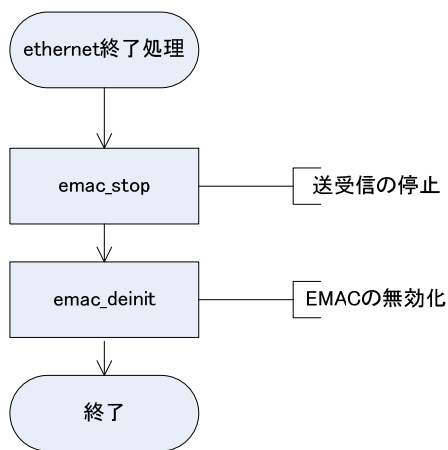


図 4.2.7.3.4 EMAC 終了処理

4.3 FPGA Driver

4.3.1 概要

FPGA ドライバは、Arduino connector をもつ DE0-Nano-SoC ボードのみ対応している。このボード用に QSYS でサンプル実装したデバイスドライバの仕様について記載する。ドライバとして以下のドライバをサポートする。このドライバを使用するには、Arduino connector に対応した FPGA 設計と FPGA の初期化が必要である。

- (1) FPGA pinmode Arduino connector に対応した GPIO 設定

- (2) FPGA uast Arduino D0,D1 ピンを使用した FPGA-UART デバイスドライバ
- (3) FPGA spi Arduino D10,D11,D12,D13 ピンを使用した FPGA-SPI デバイスドライバ
- (4) FPGA adc Arduino A0,A1,A2,A3,A5 入力をデジタル変換

4.3.2 FPGA pinmode

Arduino connector の デジタル・ピンに対応した GPIO 設定を行う。

4.3.2.1 データ仕様

(1) mode

モードは GPIO の設定を指定する。

| 定義 | 値 | 内容 |
|--------------|---|---------------|
| INPUT | 0 | GPIO 入力モード |
| INPUT_PULLUP | 1 | GPIO 入力モードと同等 |
| OUTPUT | 2 | GPIO 出力モード |

表 4.3.2.1 mode 設定値

4.3.2.2 インターフェイス仕様

GPIO に割り当てられたデジタル・ピンの初期化、制御を行う。

| 関数名 | 型 | 引数 | 機能 | 備考 |
|--------------|------|-------------------------------|----------------------------|----|
| pinMode | void | uint8_t pinno uint8_t mode | デジタル・ピンの GPIO の初期設定を行う | |
| digitalRead | int | uint8_t pinno | デジタル・ピンのデータを読み取る | |
| digitalWrite | void | uint8_t pinno int sw | デジタル・ピンの出力値を設定する。sw は0または1 | |

表 4.3.2.2 GPIO 設定関数

4.3.2.3 設定手順

pinMode 関数にて、入力、出力の初期化を行った後、digitalRead 関数、または digitalWrite 関数に GPIO の入力、出力を行う。FPGA 上の GPIO のベースアドレスが変更となった場合は、デファイン宣言 TADR_PIO0_BASE を外部からのデファイン宣言に置き換えることが可能である。

4.3.3 FPGA uart

FPGA uart は、QSYS の UART(altera_avalon-uart)を使用してシリアル通信ドライバを提供する。Arduino connector 上の実装を想定するため、ハードウェアフロー制御用の CTS/RTS 処理は altera_avalon-uart で提供しているモジュールを使用せず、デジタル・ピンの GPIO を用いて実装する。また、ボーレート等の初期設定は QSYS で設定するため、設定用のインターフェイスはない。

4.3.3.1 データ仕様

FPGA uart で使用するハンドラ型：FPGAUART_Handle_t を表 4.3.3.1.1 に示す。FPGA uart のハードウェア設定構造体として FPGAUART_PortControlBlock 型をもつ。この型の設定を表 4.3.3.1.2 に示す。加えて FPGA uart のバッファ領域を管理する型として queue_t 型を使用する。この型を表 4.3.3.1.2 に示す。queue_t 型はバッファ位置アイテムが 16 ビットであるため最大のバッファサイズは 64K byte -1 となる。

| 番号 | 項目 | 型 | 機能 |
|----|------------|-------------------------------------|------------------------|
| 1 | base | uint32_t | FPGA uart レジスタのベースアドレス |
| 2 | pcb | const FPGAUART_PortControlBlock* | FPGA uart コンフィギュレーション型 |
| 3 | ctrl | uint32_t | コントロール・レジスタの保存領域 |
| 4 | rxque | queue_t* | 受信データ管理キューへのポインタ |
| 5 | txque | queue_t | 送信データ管理キューへのポインタ |
| 6 | rxcallback | void (*)(FPGAUART_Handl_t*) | 受信発生時のコールバック関数 |

表 4.3.3.1.1 FPGAUART ハンドラ型

本ドライバは CTS/RTS を用いたハードウェアフロー制御に対応する。rtspin、ctspin は Arduino connecter 上のデジタル・ピンのピン番号。

| 番号 | 項目 | 型 | 機能 |
|----|--------|----------|-------------------------------------|
| 1 | Base | uint32_t | FPGA uart レジスタのベースアドレス |
| 2 | intno | uint32_t | HPS 上の割り込み番号 |
| 3 | rtspin | uint32_t | RTS 用 GPIO ピン番号(0 の場合、RTS 機能を使用しない) |
| 4 | ctspin | uint32_t | CTS 用 GPIO ピン番号(0 の場合、CTS 機能を使用しない) |

表 4.3.3.1.2 SPI コンフィギュレーション型

| 番号 | 項目 | 型 | 機能 |
|----|--------|----------|---------------|
| 1 | size | uint16_t | バッファ領域のバイトサイズ |
| 2 | top | uint16_t | データ設定位置管理番号 |
| 3 | tail | uint16_t | データ取り出し位置管理番号 |
| 4 | limit | uint16_t | スレッシユ値 (未使用) |
| 5 | buffer | uint8_t* | バッファ領域へのポインタ |

表 4.3.3.1.3 queue_t 型

4.3.3.2 インターフェイス仕様

FPGA uart を制御するドライバ関数は以下の通りである。fpgauart_cts_sens ハードウェアフロー制御時に有効。この関数は送信可能チェックを行い、送信可能で、送信データが残っている場合、送信再開を行う。

| 関数名 | 型 | 引数 | 機能 | 備考 |
|-------------------|-------------------|--|--|----|
| fpgauart_init | FPGAUART_Handler* | ID portid | 指定ポート ID の UART を初期化し、ハンドラへのポインタを返す | |
| fpgauart_deinit | ER | FPGAUART_Handler* huart | UART を未使用状態に戻す | |
| fpgauart_send | int | FPGAUART_Handler* huart unsigned char *buf int len | UART 送信キューに送信データをセットし、送信可能ならば送信を行う。返り値はキューに設定したデータサイズ。 | |
| fpgauart_recev | int | FPGAUART_Handler* huart unsigned char *buf int len | UART 受信キューから受信データを取り出す。取り出したデータサイズを返す。 | |
| fpgauart_resize | int | FPGAUART_Handler* huart | 受信バッファに蓄積された受信データサイズを返す。 | |
| fpgauart_cts_sens | bool_t | FPGAUART_Handl_t* huart | 送信可能用センス true なら可能 | |
| fpgauart_isr | void | intptr_t exinf | FPGA uart 割り込みサービスルーチン | |

表 4.2.3.2.1 SPI ドライバ関数

4.3.3.3 フローチャート

まず、FPGA の割込み設定を有効にするために、`fpgauart.cfg` をコンフィギュレーションファイルに取り込む必要がある。割込み番号は `fpgauart.h` に記載があるため、これを書き換えればよい。図 4.3.3.3.1 に初期化フローを示す。初期化は `fpgauart_init` 関数にポート番号を引数として呼び出し、ハンドラのポインタを取り出す。ハンドラのポインタが `NULL` の場合、初期化エラーが発生したことを示す。そのあと、ハンドラの `rxcallback` に受信用コールバック関数を設定する。コールバック関数が呼び出されるとセマフォやイベントフラグを用いて受信プログラムにイベントを発生するように設計する。図 4.3.3.3.2 に受信のフローを記載する。受信イベントが発生すると、`fpgauart_resize` 関数を用いて受信データサイズを取り出し、`fpgauart_recev` 関数を用いてキューバッファから受信データを取り出す。図 4.3.3.3.3 に送信フローを示す。`fpgauart_send` 関数を用いて送信データをキューバッファにセットする。返り値に実際にセットされたバイト数が返される。キューバッファにオーバーフローが発生した場合は設定数より少ない値が返されるため、時間をおいて、送信できなかったデータは再送信する必要がある。

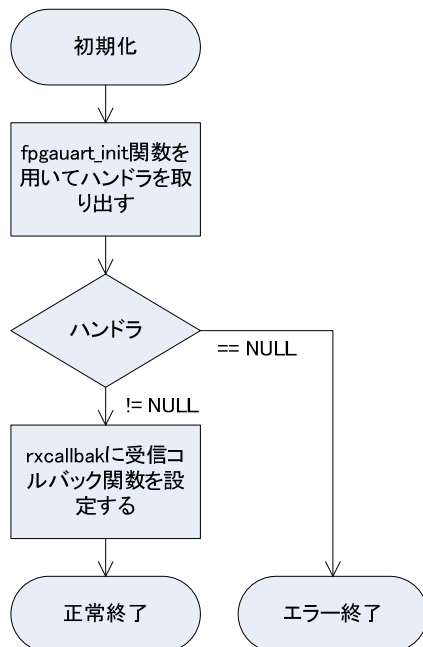


図 4.3.3.3.1 初期化

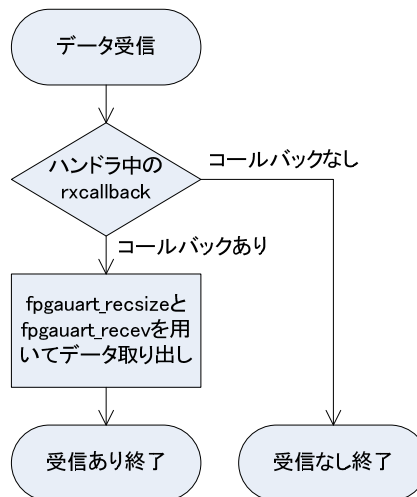


図 4.3.3.3.2 データ受信

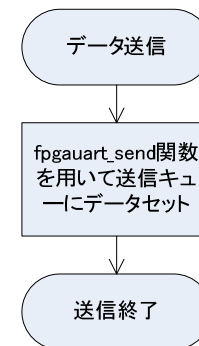


図 4.3.3.3.3 データ送信

4.3.4 FPGA SPI

FPGA SPI は、QSYS の SPI(3 Wire serial)を使用して SPI 通信ドライバを提供する。初期設定は QSYS で設定するため、コンフィギュレーション型はない。

4.3.4.1 データ仕様

FPGA SPI で使用するハンドラ型：FPGASPI_Handle_t を表 4.3.4.1.1 に示す。SPI 通信は割込みを使用し FIFO を用いてデータ通信を行うため、大容量の高速データ受信には不向きである。ハンドラ型は割込みハンドラとの通信用データを確保するために使用する。また、割込みとのイベント通信用に `semid`、排他制御用に `semlock` を初期化後に設定した方が、効率的に通信を行うことができる。

`flags` 転送時の SS の制御を指定する。

`AVALON_SPI_COMMAND_MERGE`：転送終了後も SS を落とさない。

`AVALON_SPI_COMMAND_TOGGLE_SS_N`：強制 SS 設定モードを指定しない。

| 番号 | 項目 | 型 | 機能 |
|----|----|---|----|
|----|----|---|----|

| | | | |
|----|---------|-----------|-----------------------|
| 1 | base | uint32_t | FPGA SPI レジスタのベースアドレス |
| 2 | portid | uint32_t | FPGA SPI ポート ID |
| 3 | txBuff | uint8_t * | 送信データ格納領域のポインタ |
| 4 | rxBuff | uint8_t * | 受信データ格納領域のポインタ |
| 5 | size | uint32_t | 転送データのバイトサイズ |
| 6 | txCount | uint32_t | 送信済みデータカウント |
| 7 | flags | uint32_t | 受信済みデータカウント |
| 8 | timeout | uint32_t | タイムアウト時間(ms) |
| 9 | semid | int | 転送イベント通知用セマフォ ID |
| 10 | semlock | int | 排他制御用セマフォ ID |

表 4.3.4.1.1 FPGASPI ハンドラ型

4.3.4.2 インターフェイス仕様

FPGA uart を制御するドライバ関数は以下の通りである。fpgauart_cts_sens ハードウェアフロー制御時に有効。この関数は送信可能チェックを行い、送信可能で、送信データが残っている場合、送信再開を行う。

| 関数名 | 型 | 引数 | 機能 | 備考 |
|-------------------|------------------|---|--|----|
| fpgaspi_init | FPGASPI_handler* | ID portid | 指定ポート ID の SPI ハンドラを初期化し、ハンドラへのポインタを返す | |
| fpgaspi_deinit | ER | FPGASPI_Handler* hspi | SPI を未使用状態に戻す | |
| fpgaspi_transrecv | ER | FPGASPI_Handler* hspi uint32_t slave uint8_t *tx_buf uint8_t *rx_buf uint32_t len | slave に対して、tx_buf から送信し、rx_buf に受信する。転送長は len。バッファがない場合は NULL を指定する。 | |
| spim_fpga_isr | void | intptr_t exinf | FPGA SPI 割込みサービスルーチン | |

表 4.3.4.2.1 FPGASPI ドライバ関数

4.3.4.3 フローチャート

まず、FPGASPI の割込み設定を有効にするために、ユーザーのコンフィギュレーションファイルに割込みサービスルーチンで INTNO_F2S_FPGA_IRQ32 を有効にする必要がある（割込み番号は QSYS の設定で変更が可能）。図 4.3.4.3.1 に初期化フローを示す。初期化は fpgaspi_init 関数にポート番号を引数として呼び出し、ハンドラのポインタを取り出す。ハンドラのポインタが NULL の場合、初期化エラーが発生したことを示す。semid と semlock にセマフォを設定後、fpgaspi_transrecv 関数を用いてデータ転送を行う。転送のフローチャートを図 4.3.4.3.2 に示す。バッファがない場合はバッファへのポインタに NULL をセットする。

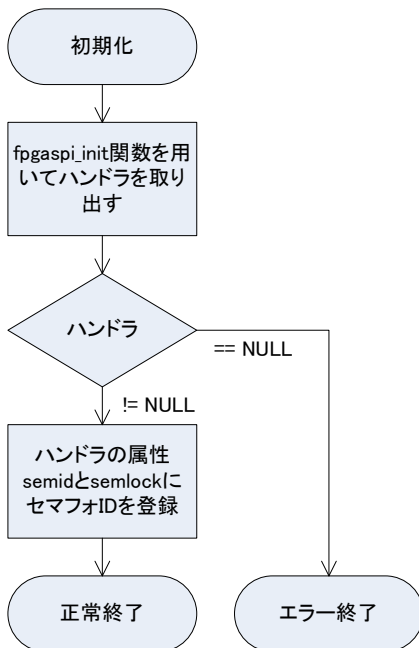


図 4.3.4.3.1 初期化

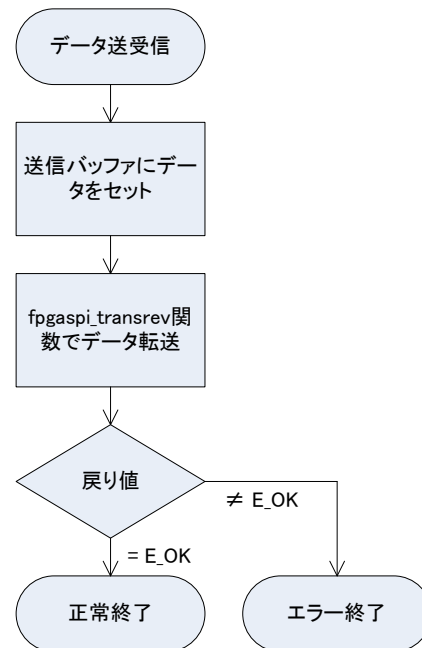


図 4.3.4.3.2 データ転送

4.3.5 FPGA ADC

FPGA ADC は、QSYS の ADC Controller for DE-series Boards を FPGA IP として使用し、AD ピンのアナログ値をデジタル変換して ADC コントローラのレジスタに取り込みを行う。初期設定は QSYS で設定するため、コンフィギュレーション型はない。この機能は DE ボードの固有機能である。

4.3.5.1 データ仕様

FPGA ADC は、FPGA のアドレスに以下のようにマッピングされる。このレジスタを直接読み込むことにより、変換データを取り込むことができる。

| 番号 | 項目 | 型 | オフセット | 機能 |
|----|-------------|----------|----------|---------------------|
| 1 | ADC_CH0_REG | uint32_t | 0(read) | CH0/A0 のアナログデータ読み込み |
| 2 | ADC_UPDATE | uint32_t | 0(write) | データ書き込みで ADC スタート設定 |
| 3 | ADC_CH1_REG | uint32_t | 4(read) | CH1/A1 のアナログデータ読み込み |
| 4 | ADC_CH2_REG | uint32_t | 8(read) | CH2/A2 のアナログデータ読み込み |
| 5 | ADC_CH3_REG | uint32_t | 12(read) | CH3/A3 のアナログデータ読み込み |
| 6 | ADC_CH4_REG | uint32_t | 16(read) | CH4/A4 のアナログデータ読み込み |
| 7 | ADC_CH5_REG | uint32_t | 20(read) | CH5/A5 のアナログデータ読み込み |

表 4.3.5.1.1 FPGA ADC レジスタマップ

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ク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 | ファイルへプリント | 結果は項目 |

| | | | | |
|---------|-----|--------------------------|-------------------|---|
| | | const char *const ... | | 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

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 5.3.2 ファイルデバッグコマンド

RTC 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 |
|--------|--------|----------------|----------------|
| RTC | DATE | year month day | 日にちを設定する |
| | TIME | hour min sec | 時間を設定する |
| | CLOCK | - | 現在の日にちと時間を表示する |

表 5.3.3 RTC 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うことができる。

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。

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。

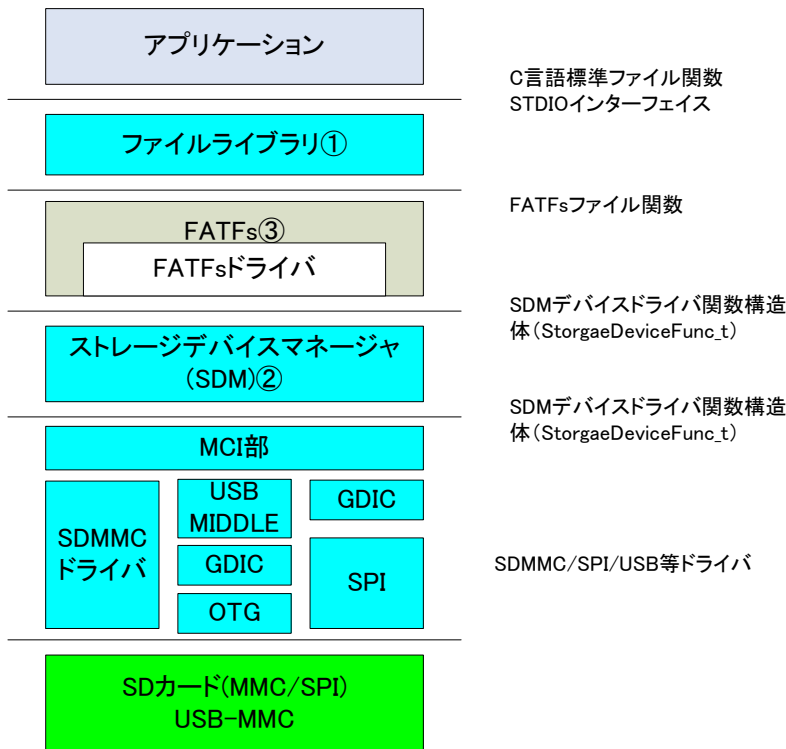


図 6.2.1 ファイルシステムのレイア構造

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 | 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 |

| | | | | |
|----------|----------------|--|--|---------------------|
| | | int flags int fd off_t offset | | |
| mmap | 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 struct statfs2 *stat | マウントされたファイルシステムについての情報を返す。 | 成功 0、失敗 -1 |

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 (*)0 | デバイスセンス関数 |
| 2 | _sdev_diskinit | int (*)0 | デバイスの初期化関数 |
| 3 | _sdev_diskstatus | int(*)0 | デバイスの状態取出し関数 |
| 4 | _sdev_diskread | int(*)0 | デバイスブロックリード関数 |
| 5 | _sdev_diskwrite | int(*)0 | デバイスブロックライト関数 |
| 6 | _sdev_diskioctl | int(*)0 | 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 *(*)0 | opendir 関数 |
| 2 | _sdevff_closedir | int (*)0 | closedir 関数 |
| 3 | _sdevff_readdir | int (*)0 | readdir 関数 |
| 4 | _sdevff_mkdir | int (*)0 | mkdir 関数 |
| 5 | _sdevff_rmdir | int (*)0 | rmdir 関数 |
| 6 | _sdevff_unlink | int (*)0 | unlink 関数 |
| 7 | _sdevff_rename | int (*)0 | rename 関数 |
| 8 | _sdevff_chmod | int (*)0 | chmod 関数 |
| 9 | _sdevff_stat | int (*)0 | stat 関数 |
| 10 | _sdevff_statfs | int (*)0 | statfs 関数 |
| 11 | _sdevff_open | int (*)0 | open 関数 |
| 12 | _sdevff_close | int (*)0 | close 関数 |
| 13 | _sdevff_fstat | int (*)0 | fstat 関数 |
| 14 | _sdevff_lseek | off_t (*)0 | lseek 関数 |
| 15 | _sdevff_read | long (*)0 | read 関数 |
| 16 | _sdevff_write | long (*)0 | write 関数 |
| 17 | _sdevff_mmap | void *(*)0 | 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も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 (*)0 | 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 | <code>_get_datetime</code> | <code>uint32_t (*)0</code> | 時刻取出し関数 |
| 5 | <code>_psd</code> | <code>StorageDevice_t *</code> | 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 |
|-----------------------------|----------------------------|---------------|
| <code>SDEV_ACTIVE</code> | <code>(1<<15)</code> | デバイスがアクティブの状態 |
| <code>SDEV_INSERTCHK</code> | <code>(1<<14)</code> | 挿入・排出の設定あり |
| <code>SDEV_CHKREMOVE</code> | <code>(1<<13)</code> | 排出検査を行う |
| <code>SDEV_ONEXIT</code> | <code>(1<<12)</code> | 一度以上排出があった |
| <code>SDEV_EMPLOY</code> | <code>(1<<8)</code> | デバイス動作中 |
| <code>SDEV_ERROR</code> | <code>(1<<7)</code> | デバイスエラー |
| <code>SDEV_DEVNOTUSE</code> | <code>(1<<0)</code> | デバイス使用不可 |
| <code>SDEV_NOTUSE</code> | <code>255</code> | 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(`StorageDevice_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 |
|----------------------------------|--------------------------------|---|--|----|
| <code>sdev_init</code> | <code>void</code> | <code>intptr_t exinf</code> | ストレージデバイスマネージャーを初期化する | |
| <code>sdev_terminate</code> | <code>void</code> | | ストレージデバイスマネージャーを終了する | |
| <code>SDMSetupDevice</code> | <code>ER</code> | <code>int16_t devno</code> <code>StorageDevice_t **ppsdev</code> | 指定のデバイスを追加する | |
| <code>SDMDeviceNo</code> | <code>ER_ID</code> | <code>const char **ppathname</code> | 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 | |
| <code>SDMGetStorageDevice</code> | <code>StorageDevice_t *</code> | <code>int devno</code> | 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 | |
| <code>SDMEmploy</code> | <code>ER</code> | <code>StorageDevice_t *psdev</code> <code>bool_t sw</code> | デバイスの動作中(<code>true</code>)、停止中(<code>false</code>)を設定する | |
| <code>SDMSence_task</code> | <code>void</code> | <code>intptr_t exinf</code> | 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 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 USB ミドルウェア

USB ミドルウェアとして、TOPPERS USB MIDDLEWARE を対応する。これは USB OTG デバイス用の USB ホスト、デバイス機能、USB デバイスに対して USB デバイス機能を提供する。

6.4.1 USB ホスト機能

USB ホスト機能は、マルチデバイスを対応して HUB クラスに対して複数のクラスを同時管理可能である USB ホストは以下の 5 つのクラスをサポートする。

| クラス名 | ID | 機能 | アプリとの通信機能 | 備考 |
|------------|------|----------------|-----------------------|-------|
| HUB | 0x09 | HUB のサポート | | |
| HID | 0x03 | MOUSE/KEYBOAED | コールバック関数設定 | |
| MSC | 0x08 | USB メモリ | API による関数コール | |
| PRT | 0x07 | プリンタ | API による関数コール/コールバック関数 | |
| CDC/SERIAL | 0x02 | USB シリアル | API による関数コール/コールバック関数 | |
| BLUETOOTH | 0xE0 | Bluetooth クラス | API による関数コール/コールバック関数 | 1.3.1 |

表 6.4.1.1 USB ホストクラス

6.4.2 USB デバイス機能

USB デバイス機能は、指定のクラスに対してのデバイス機能を提供する。

| クラス名 | ID | 機能 | 備考 |
|------|------|----------------|----|
| HID | 0x03 | MOUSE/KEYBOAED | |
| MSC | 0x08 | USB メモリ | |

表 6.4.1.1 USB デバイスクラス

6.5 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。

6.5.1 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 6.5.1.1 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、ファイルとして参照する。TOPPERS BASE PLATFORM(CV)で使用するには、SD カードのルートディレクトリ上に東雲フォントビットマップファイルコピーすれば、アクセス可能になる。フォントビットマップファイルは ui/snfont_disp/shinonome_font12.fnt(高さ 12)、ui/snfont_disp/shinonome_font16.fnt(高さ 16)である。漢字フォントをプログラムのデータ領域に置きたい場合は、USE_ROMFONT コンパイルスイッチを有効にすれば、romfont というグローバル変数として定義される。ROM ファイル機能を使って ROM ファイルに変換すればファイルとしてアクセスができる。

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 6.5.1.2 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 fmp 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 | |
| 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 | |
| sysvc | malloc, calloc, free 関数 | |
| usb | USB ホスト、デバイスのミドルウェア | |
| lwip | lwip プロトコル・スタック、Web より lwip-2.1.2 と contrib-2.1.0 をダウンロードセットし、patch ディレクトリ中のソースパッチを行ってください | ソースなし |

表 7.1.1 共通部ディレクトリ

7.2 Cyclone V hps ドライバ

Cyclone V hps 用ドライバ部は pdic/hps_cv 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 | |
| mcicmd.h | ファイルシステム用インクルードファイル | |
| qspi.c | QSPI ドライバ・ソースファイル | |
| qspi.h | QSPI 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 | |
| sdmmc.h | SD-card ドライバ・ソースファイル | |
| sdmmc.cfg | SD-card ドライバの RTOS リソースファイル | |
| sdmmc.c | SD-card インクルード・ソースファイル | |
| socfpga_cv.h | HPS 用 SIL 記載インクルードファイル | |
| usb_otg.c | USB-OTG ドライバ・ソースファイル | |
| usb_otg.h | USB-OTG ドライバ・インクルードファイル | |
| emac.c | LWIP 用 EMAC ドライバ・ソースファイル | |

| | | |
|----------|-----------------------------|--|
| emac.h | LWIP 用 EMAC ドライバ・インクルードファイル | |
| phyreg.h | PHY レジスタ定義・インクルードファイル | |

表 7.2.1 Cyclone V hps ドライバファイル

7.3 FPGA ドライバ

FPGA ドライバ部は pdic/fpga にソースファイルがある。FPGA の実装は QSYS の設定に従うため、IP のベースアドレスは、実装により修正を行う必要がある。ベースアドレスの定義は fpga.h インクルードファイル中に記載している。

| ファイル | 内容 | 備考 |
|--------------|---|-------|
| fpga.h | FPGA の定義用インクルードファイル | |
| pinmode.c | Arduino デジタル・ピンの GPIO 設定ドライバ・ソースファイル | |
| pinmode.h | Arduino デジタル・ピンの GPIO 設定ドライバ・インクルードファイル | |
| fpgauart.c | FPGA UART ドライバ・ソースファイル | |
| fpgauart.cfg | FPGA UART ドライバ・コンフィギュレーションファイル | |
| fpgauart.h | FPGA UART ドライバ・インクルードファイル | |
| fpgaspi.c | FPGA SPI ドライバ・ソースファイル | |
| fpgaspi.h | FPGA SPI ドライバ・インクルードファイル | |
| fpgai2c.c | FPGA I2C ドライバ・ソースファイル SDA の受信 BAIDIRECTION 回路をうまく合成できず送信のみ可能 | 1.4.1 |
| fpgai2c.h | FPGA I2C ドライバ・インクルードファイル | 1.4.1 |

表 7.3.1 FPGA ドライバファイル

7.4 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_otg | usb_otg(DWC2-OTG ドライバ)上に位置し、USB ミドルウェアに OTG 機能を提供する | |
| 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 | fpga 用 |
| 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 | fpga 用 1.4.1 |
| 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 | fpga 用 1.3.1 |
| 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 | fpga 用 1.4.1 |
| rom_file | ROM 上のデータを読み出し専用ファイルとしてアクセスする機能を提供する | 1.4.1 |

表 7.4.1 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 |
|---------|------------|--|
| 1.1.0 | 06/25/2018 | リードオンリーの fopen で open エラーとならない不具合対応(プログラムのみ) |
| | 07/05/2018 | usb host SERIAL/PRT 送受信を並行処理できるよう修正 |

| | | |
|-------|------------|--|
| | | BLUETOOTH CLASS の対応 |
| | 08/24/2018 | I2C スレーブアドレス切り替え時の障害対応、I2C メモリ READ 変更 |
| | 09/12/2018 | SOC EDS Command shell を開発環境として追加 |
| | 10/17/2018 | ui ディレクトリ追加、gdc/aqm1284_st7565 を追加 |
| | 11/07/2018 | Watch doc ドライバを追加/gpio 割込みの不具合対応 |
| 1.1.1 | 04/09/2019 | gdc adafruit_ILI9341 を追加 |
| | 04/17/2019 | gdc aqm0802_st7032/fpgai2c ソフトドライバの仮対応 |
| | 07/05/2019 | lwip のバージョンを 2.1.2 アップ、ネットアプリの不具合対応 |
| | 11/07/2019 | TLSF2.4.6 アロケート・フリーを繰り返すとアロケート領域の先頭 8 バイトが破壊される問題の暫定対策 |
| | 12/03/2019 | ROM ファイル機能を GDIC に追加。ui/snfont_disp に漢字フォントをプログラムデータとして取り込み機能を追加 |
| | 03/31/2020 | メモリヒープの libc/LOCAL HEAP の対応 |
| | | |
| | | |
| | | |

Version 1.1.0 以降のエラッタの内容を記載

| version | 判明日 | エラー内容 |
|----------|------------|---|
| 1.1.0 以降 | 04/18/2020 | FPGAUART の CTS 入力をバイディレクション GPIO を用いて行っているが、FPGA の設定で正しく入力を反映できない。そのため、現状は常に CTS 送信可能と判定する。FPGA 設定の関係で当面对応を行わない |
| | | |
| | | |
| | | |

Appendix A DE0-Nano-SoC Development Kit(Atlas-SoC Kit)/DE10-Nano

DE0-Nano-Soc/DE10-Nano のボード依存仕様を記載する。このコネクタを使用するには QSYS の設定と FPGA のコンフィギュレーションを行う必要がある。

(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 | - | - |
| | 2 | A1 | - | - |
| | 3 | A2 | - | - |
| | 4 | A3 | - | - |
| | 5 | A4 | - | - |
| | 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 | PIN_AG11(FPGA) | (SCL) |
| | 9 | D14 | PIN_AH9(FPGA) | (SDA) |
| | 8 | AREF | - | AVDD |

| | | | | |
|----------------|---|-----|----------------|----------------|
| | 7 | GND | - | Ground |
| | 6 | D13 | PIN_AH12(FPGA) | CLK |
| | 5 | D12 | PIN_AH11(FPGA) | MISO |
| | 4 | D11 | PIN_AG16(FPGA) | MOSI |
| | 3 | D10 | PIN_AF15(FPGA) | SS |
| | 2 | D9 | PIN_AE15(FPGA) | GPIO7(TFT-RST) |
| | 1 | D8 | PIN_AF17(FPGA) | GPIO6(TFT-RS) |
| CN9 digital | 8 | D7 | PIN_AH8(FPGA) | GPIO5 |
| | 7 | D6 | PIN_AG8(FPGA) | GPIO4 |
| | 6 | D5 | PIN_U13(FPGA) | GPIO3 |
| | 5 | D4 | PIN_U14(FPGA) | GPIO2 |
| | 4 | D3 | PIN_AG9(FPGA) | GPIO1 |
| | 3 | D2 | PIN_AG10(FPGA) | GPIO0 |
| | 2 | D1 | PIN_AF13(FPGA) | TX |
| | 1 | D0 | PIN_AG13(FPGA) | RX |

表 A.1.2 右 Arduino connector 設定

(2) LTC Connector

| CN No. | Pin No. | Pin 名 | MCU pin | 機能 |
|--------|---------|-------|----------------|-------------------|
| J17 | 1 | | VCC9 | Power input |
| | 2 | | VCC3C3 | 3.3V |
| | 3 | | GND | Ground |
| | 4 | | SCK_SCL | CLK/SCL |
| | 5 | | HPS_SPIM1_MISO | MISO |
| | 6 | | HPS_SPIM1_SS | SS |
| | 7 | | MOSI_SDA | MOSI/SDA |
| | 8 | | GND | Ground |
| | 9 | | HPS_I2C1_SDAT | SDA |
| | 10 | | VCC3P3 | 3.3V |
| | 11 | | HPS_I2C1_SCLK | SCL |
| | 12 | | GND | Ground |
| | 13 | | GND | Ground |
| | 14 | | HPS_LTC_GPIO | Select SPIM1/I2C1 |

表 A.1.3 LTC connector 設定

SPI のデバッグを LTC Connector を用いループバック接続して行った。

(3) サンプルプログラム

| Program name | shield | environment | detail |
|--------------|------------------|-------------|--|
| media | - | RAM | SD カードと USB メモリ(MMC)のテストプログラム |
| sdcard | - | RAM | SD カード用のファイルシステムのテストプログラム |
| FPGALCD | Adafruit 1.8 TFT | RAM | グラフィック LCD 表示テストプログラム 但し、FPGA に arduino connector 用の設計が必要 |
| MAC | - | 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/ctnrib-2.0.1)が必要 また、設定ソースに数か所のパッチが必要 |
| USBH | - | RAM | USB ホストテストプログラム |
| ADXL345 | - | RAM | ADXL345 との I2C 通信 |

このボードを使用して開発したデバイスドライバ
GPIO, SDMMC, SPI, I2C, DMA, USB-OTG

Appendix B DK-DEV-5CSX6N

DK-DEV-5CSX6N 開発キット依存仕様を記載する。

(1)Arduino connectors 定義

コネクタなし

(2)サンプルプログラム

| Program name | shield | environment | detail |
|--------------|--------|-------------|-------------------------------|
| I2C_DEMO | - | RAM | ボードに付属の LCD/EEPROM 等のテストプログラム |
| QSI | - | RAM | QSPI テストプログラム |

このボードを使用して開発、デバッグしたデバイスドライバ

I2C, QSPI

I2C はスレーブ側のポートが、アクティブとなっておらずスレーブのデバッグをしていない