KEIm SoM
ソフトウェアマニュアル

Ver.1.1.0

株式会社近藤電子工業

はじめに

この度は、KEIm 製品をお買い上げいただき誠にありがとうございます。

本製品をご使用になる前に、本マニュアル及び関連資料を十分ご確認いただき、使用上の注意を守って正しくご使用ください。

取扱い上の注意

- 本書に記載されている内容は、将来予告なく変更されることがあります。本製品のご使用にあたっては、弊社窓口又は弊社ホームページなどで最新の情報をご確認ください。
- 本製品には一般電子機器用部品が使用されています。極めて高い信頼性を要求する装置（航空、宇宙機器、原子力制御機器、生命維持のための医療機器等）には使用しないでください。
- 本製品は国内使用を前提として開発及び製造を行っています。本製品又は本製品を組み込んだ製品を輸出される場合は、お客様の責任において「外国為替及び外国貿易法」及びその他輸出関連法令等を順守し、必要な手続きを行ってください。
- LAN、USB 以外のコネクタへのケーブルの抜き差しは、必ず電源を OFF にした状態で行ってください。
- 水、湿気、ほこり、油煙等の多い場所では使用しないでください。
- 本製品の関連資料の全部又は一部を弊社に無断で使用または複製することを禁止します。
- 本書及び関連資料で取り上げる会社名及び製品名等は、各メーカーの商標または登録商標です。

お問い合わせ先

- 製品に関するお問い合わせは、下記のメールアドレスよりお願いいたします。

keim-support@kd-group.co.jp

目次

図表目次

1. 概要

本 SoftwareDevelopmentKit(以降 SDK)は KEIm SoM のソフトウェア開発用を目的とし、
KEIm SoM の PeripheralsModule の DRIVER および API のソースコードを予め実装しております。

1.1. 開発環境

本 SDK の開発環境を表 1.1.1 に示します。

表 1.1.1 開発環境表

作業用 PC	Windows7 SP1 64bit 以降
Nios® II EDS	Version15.0.0.145 以降
KEIm SoM	KEIm-08SoM
開発ボード	KEIm SoM 開発キット

1.2. 全体構成図

本 SDK の全体構成図を図 1.2.1、全体ファイル構成を表 1.2.1 に示します。

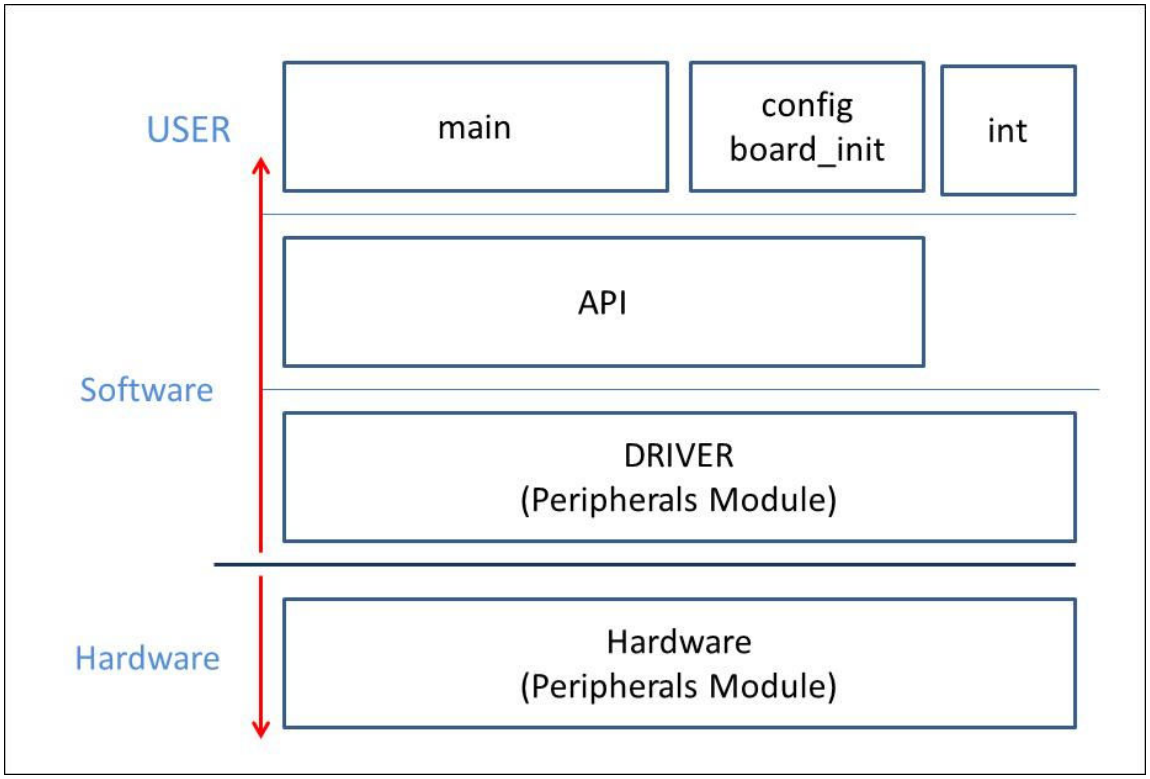


図 1.2.1 全体構成図

表 1.2.1 全体ファイル構成

TOP	階層 1	階層 2	階層 3	補足
keim_sdk_src	API-CORE	inc	EINT-core.h	外部割り込み API 関数定義
			I2C-core.h	I2C API 関数定義
			PIO-core.h	PIO API 関数定義
			PWM-core.h	PWM API 関数定義
			SPI-core.h	SPI API 関数定義
			TIMER-core.h	TIMER API 関数定義
			UART-core.h	UART API 関数定義
			WDT-core.h	WDT API 関数定義
		src	EINT-core.c	外部割り込み API 関数
			I2C-core.c	I2C API 関数
			PIO-core.c	PIO API 関数
			PWM-core.c	PWM API 関数
			SPI-core.c	SPI API 関数
			TIMER-core.c	TIMER API 関数
			UART-core.c	UART API 関数
			WDT-core.c	WDT API 関数
	DRIVER	inc	driver-common.h	ドライバ common 定義
			KEIMSoM-EINT.h	外部割り込みレジスタアクセス関数定義
			KEIMSoM-I2C.h	I2C レジスタアクセス関数定義
			KEIMSoM-PIO.h	PIO レジスタアクセス関数定義 端子定義
			KEIMSoM-PWM.h	PWM レジスタアクセス関数定義
			KEIMSoM-SPI.h	SPI レジスタアクセス関数定義
			KEIMSoM-TIMER.h	TIMER レジスタアクセス関数定義
			KEIMSoM-UART.h	UART レジスタアクセス関数定義
			KEIMSoM-WDT.h	WDT レジスタアクセス関数定義
		src	KEIMSoM-EINT.c	外部割り込みレジスタアクセス関数
			KEIMSoM-I2C.c	I2C レジスタアクセス関数
			KEIMSoM-PIO.c	PIO レジスタアクセス関数
			KEIMSoM-PWM.c	PWM レジスタアクセス関数
			KEIMSoM-SPI.c	SPI レジスタアクセス関数
			KEIMSoM-TIMER.c	TIMER レジスタアクセス関数
			KEIMSoM-UART.c	UART レジスタアクセス関数
			KEIMSoM-WDT.c	WDT レジスタアクセス関数
	board_init.c			ボード初期化
	board_init.h			各ヘッダファイル 割り込み関数定義
	int.c			割り込み処理の関数
	int.h			割り込み処理の関数定義
	config.h			使用機能設定定義
	main.c			メイン関数
keim_sdk_bsp				sopc_info で自動生成するファイル群

1.3. 環境構築方法

Nios II EDS での環境構築の手順を示します。以降は、作業用 PC にあらかじめ Nios II EDS15.0.0.145 がインストールされていることを前提に記載します。

1.3.1. KEIm SoM の SDK 用の workspace フォルダを作成

例：C:¥KEIm_SDK¥workspace

フォルダのパスはユーザーによって異なります。

1.3.2. Nios II EDS の起動

スタートメニューより順に

Altera 15.0.0.145 > Nios II EDS 15.0.0.145 > Nios II 15.0 Software Build Tools for Eclipse をクリックしてください。

Nios II EDS の初期起動画面を図 1.3.1、図 1.3.2 に示します。

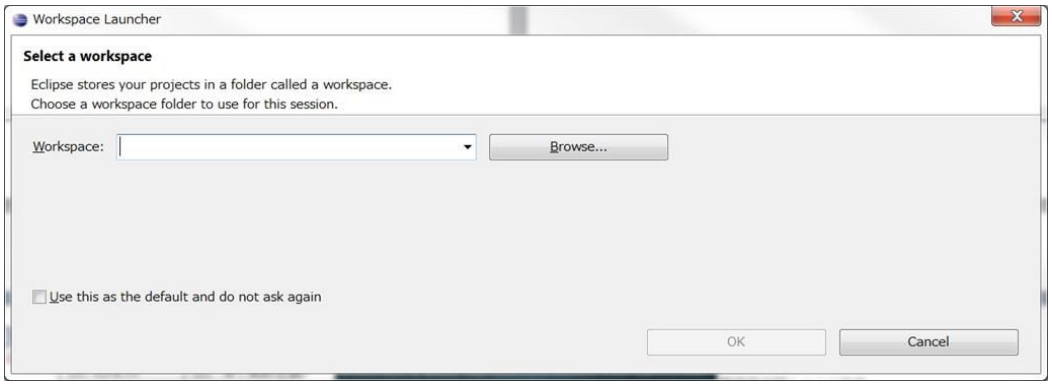


図 1.3.1 初期起動画面

1.3.1 で作成した workspace を指定し”OK”をクリックしてください。

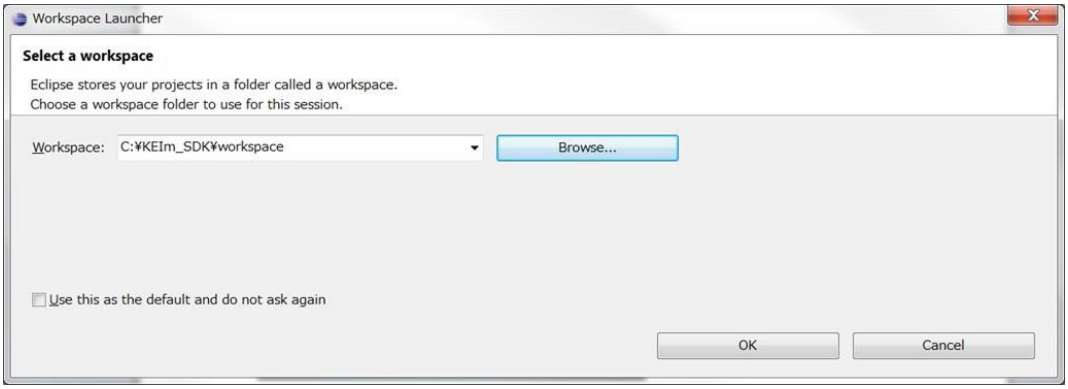


図 1.3.2 workspace 指定

1.3.3. Eclipse 起動後に bsp の作成

弊社 Web サイト(http://www.kd-group.co.jp/info/)より keim08core.sopcinfo ファイルをダウンロードし、bsp を作成します。

bsp 作成の画面を図 1.3.3～図 1.3.5 に示します。

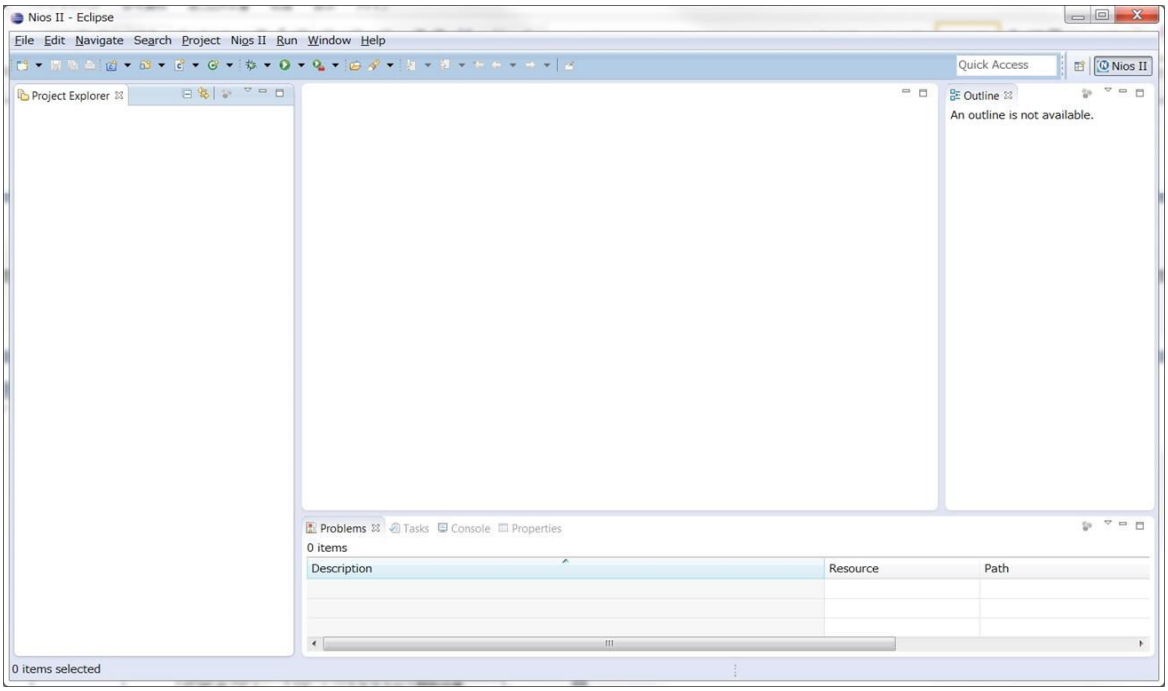
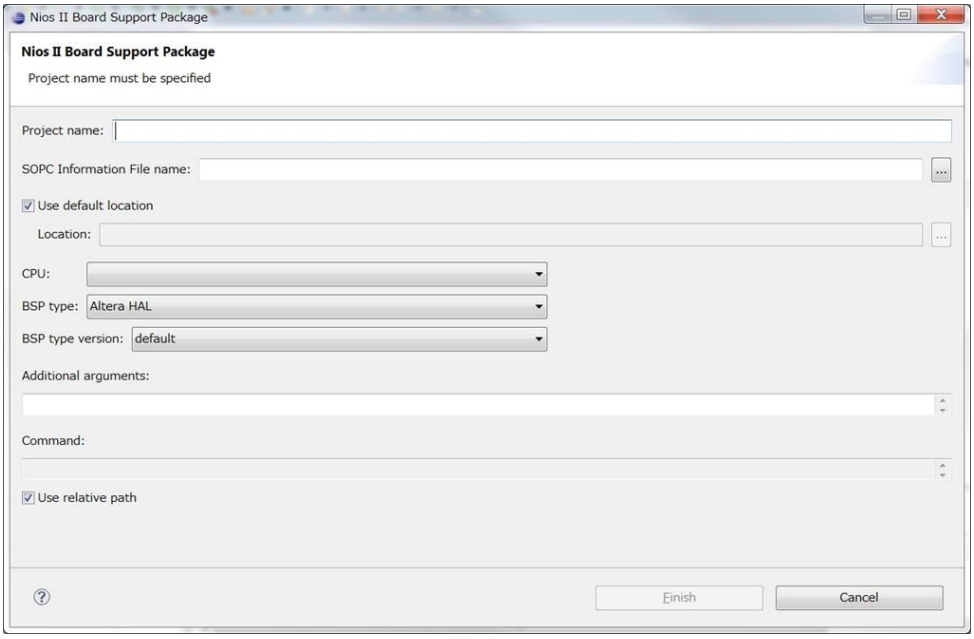


図 1.3.3 Eclipse 起動画面

● File->New->Nios II Board Support Package をクリックします。



● Project name:に例として ”keim_sdk_bsp” を入力します。

SOPC Information File name:に keim08core.sopcinfo ファイルを指定してください。

次に Finish をクリックしてください。

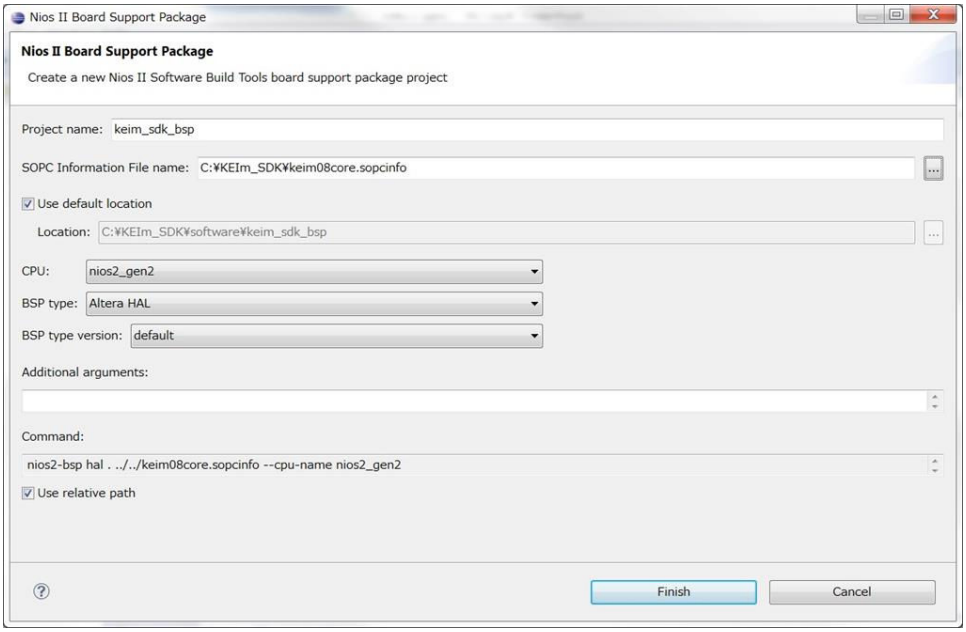


図 1.3.4 Nios II Board Support Package 設定後画面

1.3.4. SDK のソースファイルを Import

弊社 Web サイトより keim_sdk_src_verxx.zip ファイルをダウンロードし、Import を実行してください。

(xx:Versson 番号) 図 1.3.6～図 1.3.8 に示します。

- File->New->Nios II Application をクリックしてください。

  例として Project name:に”keim_sdk”を入力します。

  BSP location:keim_sdk_bsp 選択し Finish をクリックしてください。

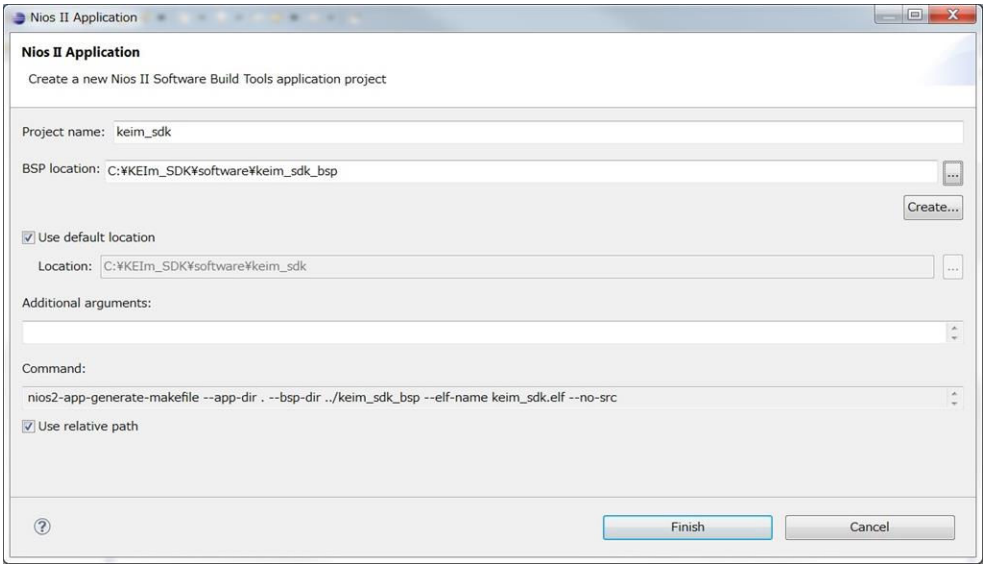


図 1.3.5 sdk Application 設定画面

- File->Import をクリックしてください。

  General->Archive を選択し Next をクリックしてください。

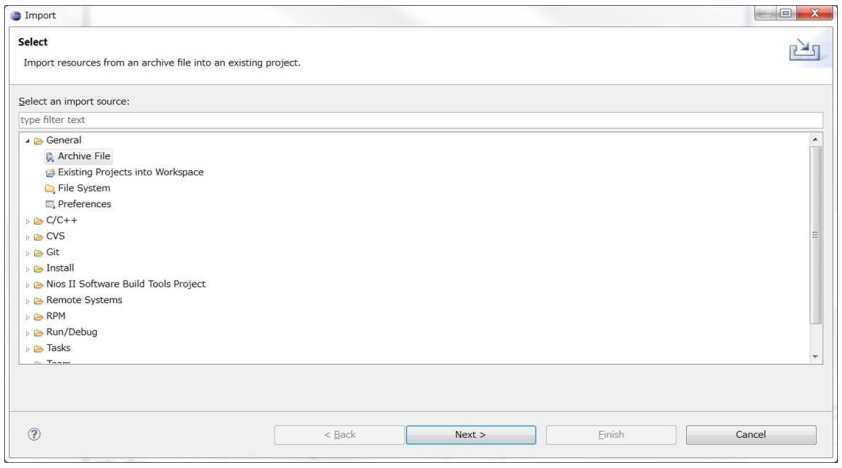


図 1.3.6 sdk Import 画面

From archive file: keim_sdk_src_verxx.zip ファイルを指定してください。

Into folder: keim_sdk を指定し Finish をクリックしてください。

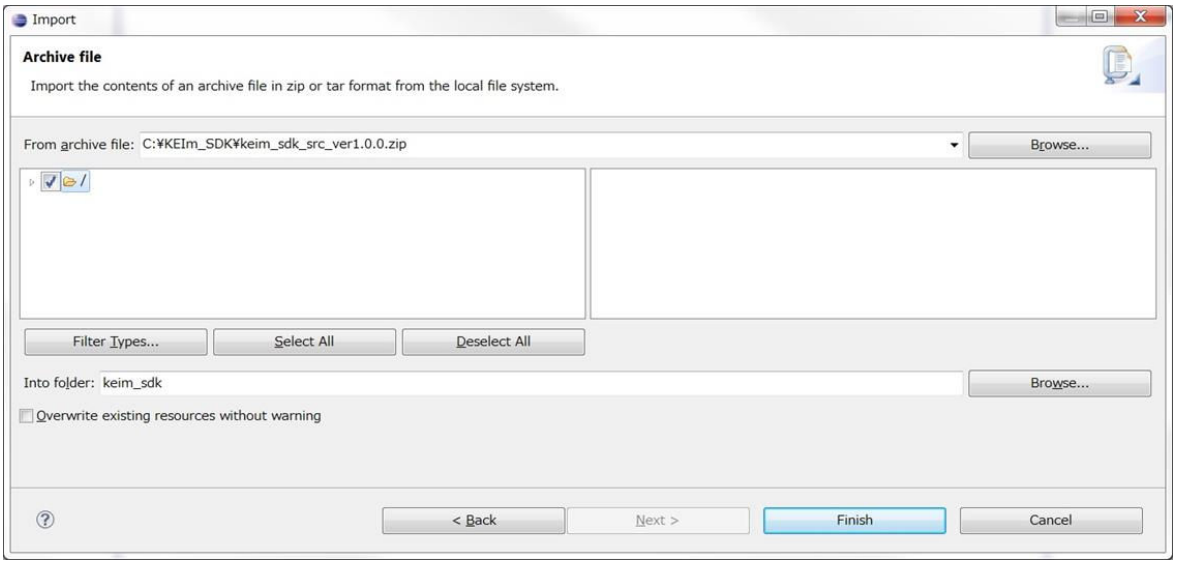


図 1.3.7 sdk ソースファイル Import 指定画面

1.3.5. bsp の Config

BSP Editor で bsp の設定を行います。

図 1.3.9～図 1.3.10 に示します。

● bsp を選択し右クリックしてください。

Nios II->BSP Editor をクリックします。

標準出力の UART のチャンネル設定、SystemClock の TIMER のチャンネル設定、

stack の設定等を変更可能です。

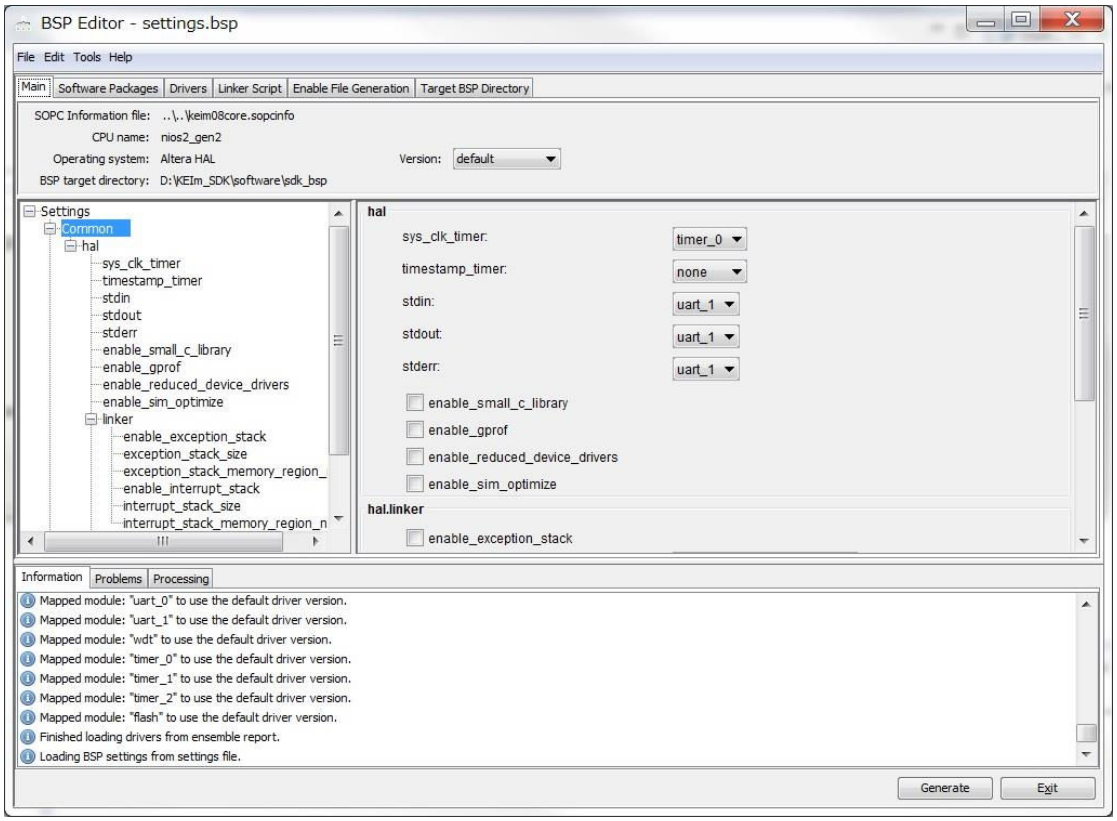


図 1.3.8 BSP Editor 起動画面

● LinkerScript の設定をします。

タブの LinkerScript をクリックします。

Section の使用するメモリを変更可能です。

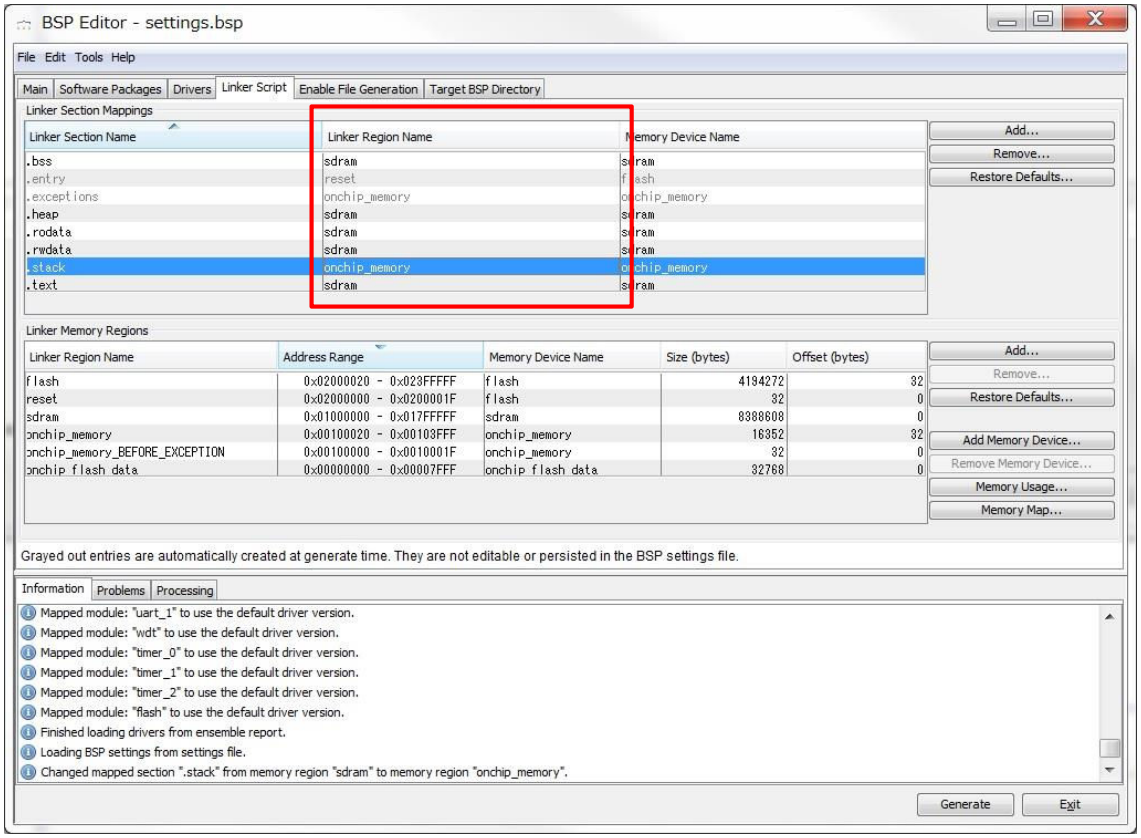


図 1.3.9 LinkerScript 設定画面

stack メモリを sdram から onchip_memory に変更してください。

● BSP Editor で設定後、Generate をクリックしてください。

1.3.6. sdk と bsp を Build

sdk と bsp を Build し、プログラムのバイナリファイル(ELF ファイル)を作成してください。

● Project->Build All をクリックしてください。

Build 実行時を図 1.3.11 に示します。

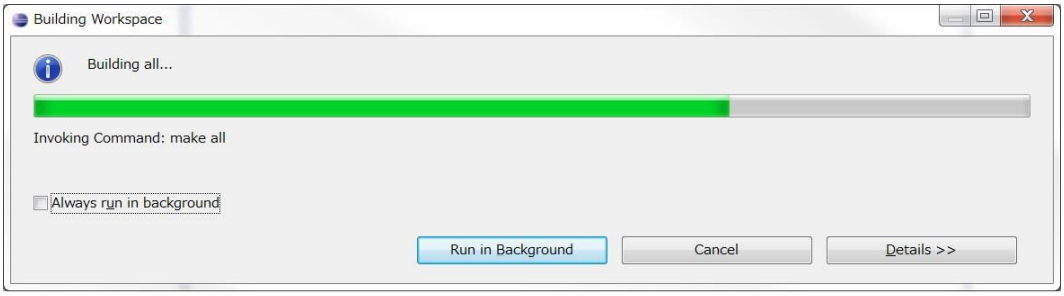


図 1.3.10 Build 実行時

● Build 完了後 keim_sdk.elf ファイルが生成されます。

Build 完了後を図 1.3.12 に示します。

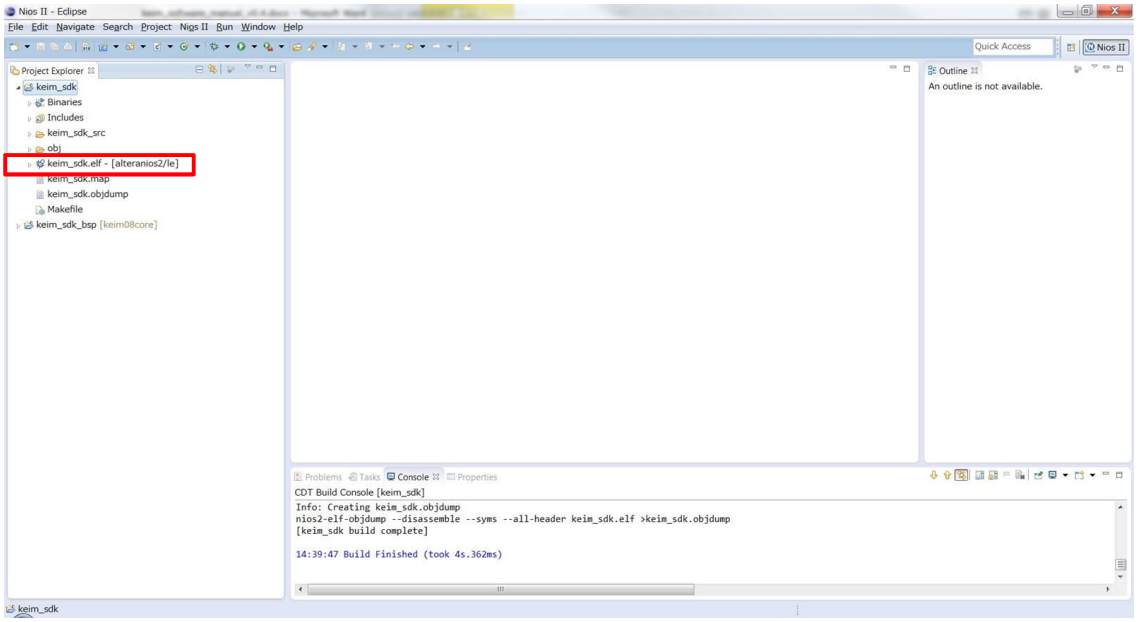


図 1.3.11 Build 完了後画面

1.3.7. コードデバッグおよび動作確認

● KEIm SoM と KEIm SoM 開発キットをセットアップします。

KEIm SoM 開発キットの CN2(JTAG)に USB Blaster の 10pin ケーブルを接続し、PC に USB ケーブルを接続してください。

● Run->DebugConfigurations をクリックし DebugConfigurations を起動します。

DebugConfigurations 起動画面を図 1.3.13 に示します。

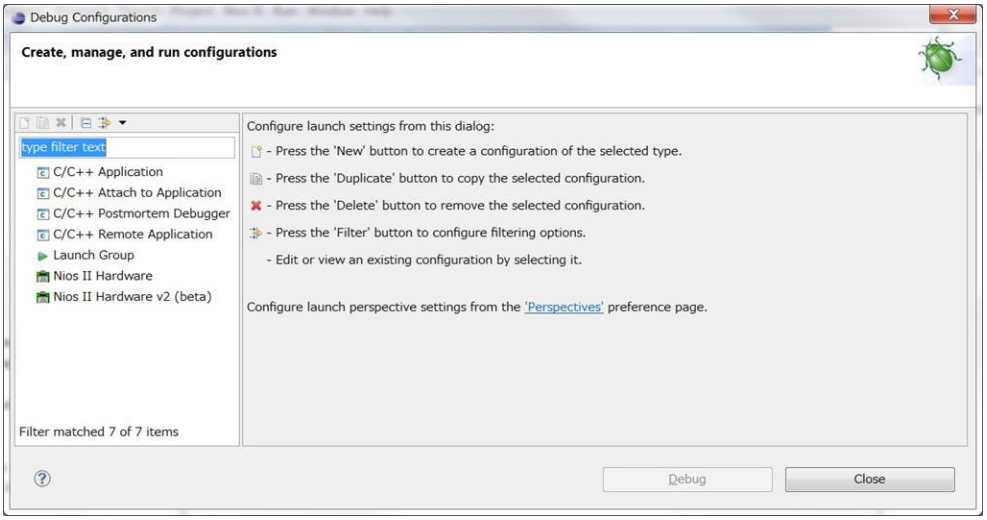


図 1.3.12 DebugConfigurations 画面

● Nios II Hardware をダブルクリックします。

Nios II Hardware Configuration 画面を図 1.3.14 に示します。

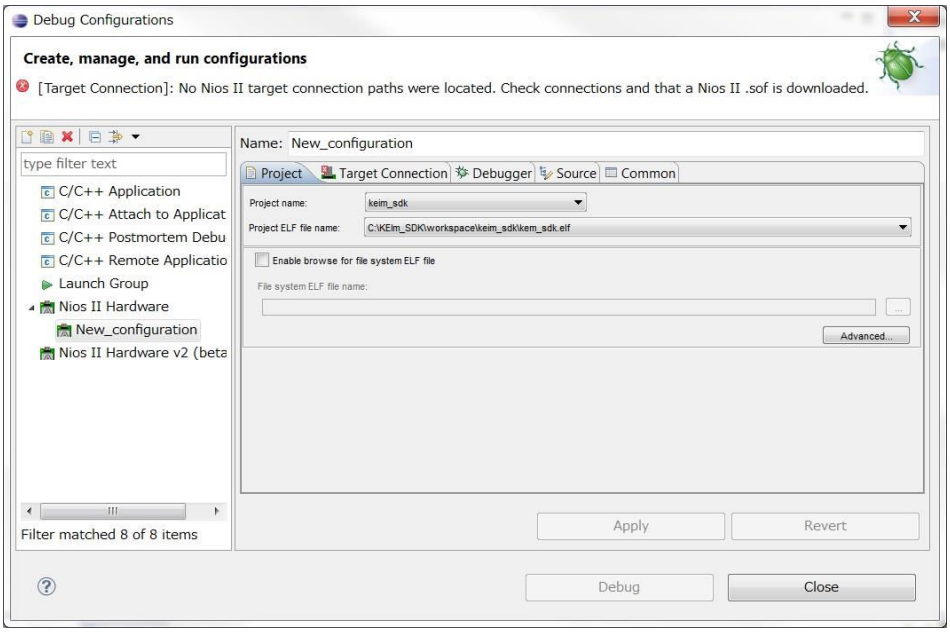


図 1.3.13 Nios II Hardware Configuration 画面

● KEIm SoM 開発キットの SW7(POWER)スイッチを ON にし、電源を投入してください。

● TargetConnection タブをクリックしてください。

TargetConnection 画面を図 1.3.15 に示します。

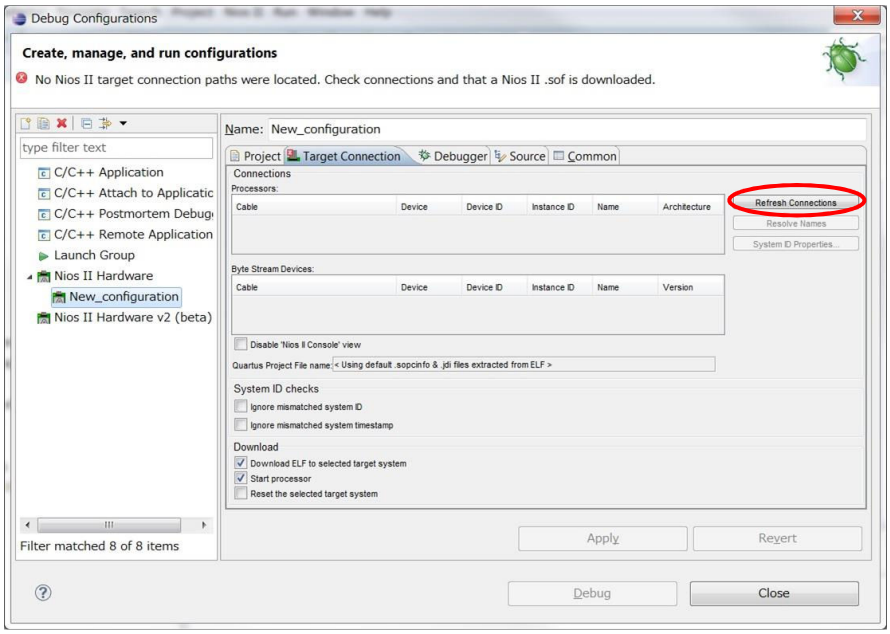


図 1.3.14 TargetConnection 画面

● RefreshConnections をクリックし KEIm SoM と接続できていることを確認してください。

System ID checks の 2 か所、Download の”Reset the selected target system”に

チェックを入れます。TargetConnection 確認画面を図 1.3.16 に示します。

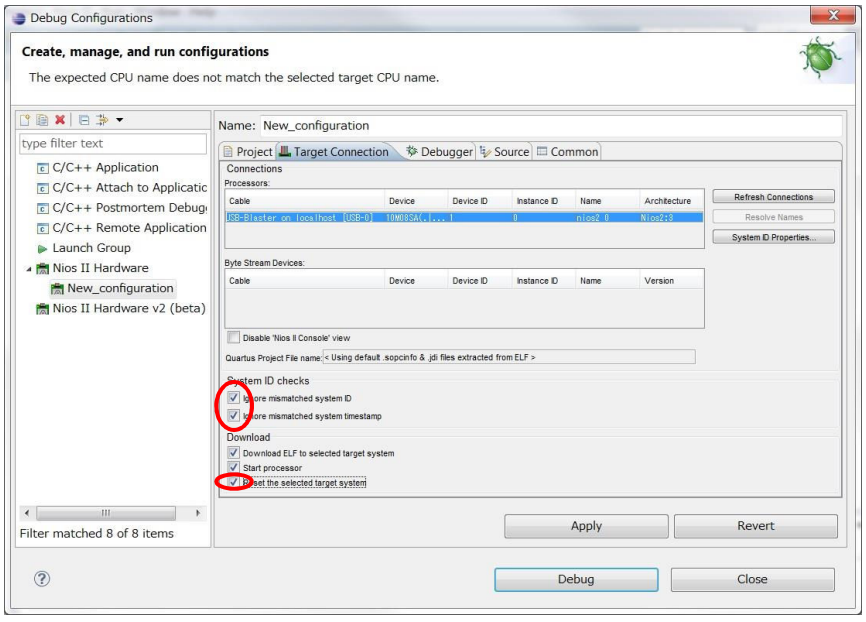


図 1.3.15 TargetConnection 確認画面

- Apply をクリックし、Debug をクリックします。
  “Confirm Perspective Switch”Window が表示、Yes をクリックします。図 1.3.17 にします。

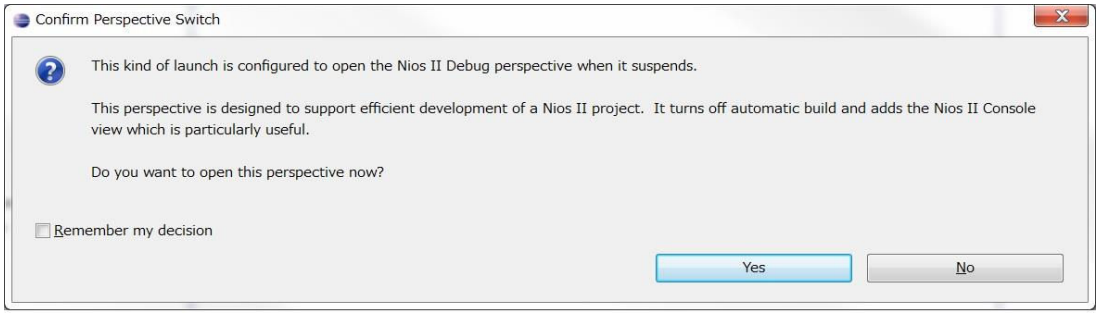


図 1.3.16 Confirm Perspective Switch 画面

- Nios II Debug 画面が表示され、main.c の先頭のプログラムカウンタで break します。
  Nios II Debug 画面を図 1.3.18 に示します。

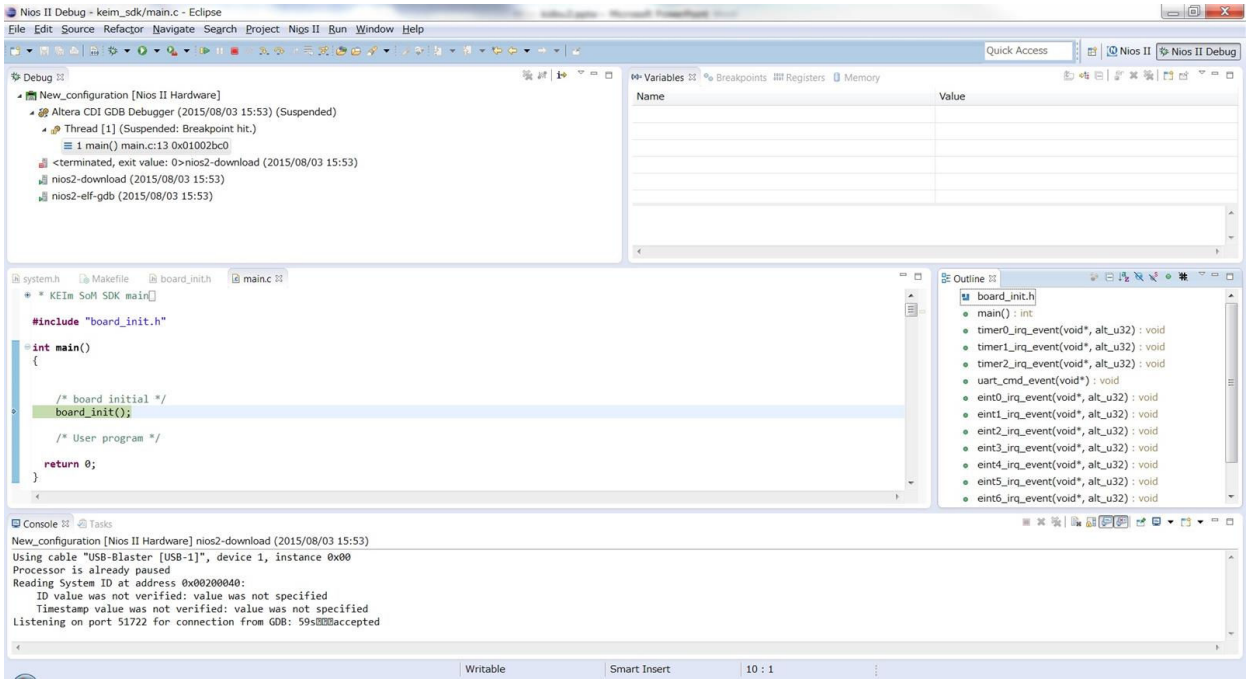


図 1.3.17 Nios II Debug 画面

- Run->Resume をクリックし、プログラム実行します。

- Run->Terminate をクリックしプログラムを停止、Nios II から切断されます。
  Nios II 切断後画面を図 1.3.19 に示します。

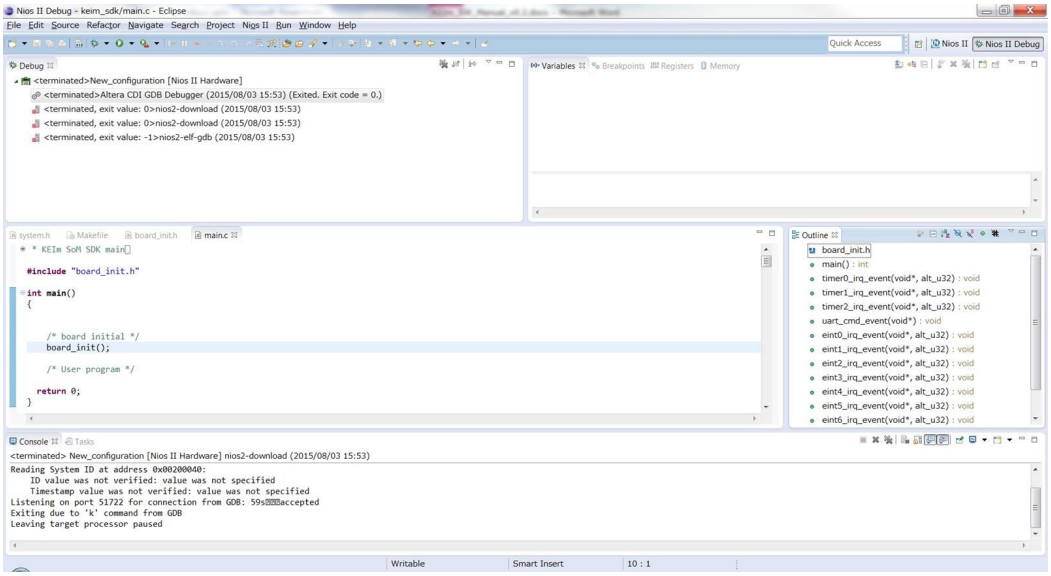


図 1.3.18 Nios II 切断後画面

- ソースコードを修正した後は、1.3.7.sdk と bsp を Build ～ 1.3.8.コードデバッグおよび動作確認を繰り返し実行していきます。

2. 設計

2.1. ボード設定

2.1.1. 概要

使用するデバイスおよび機能を設定します。

2.1.1.1. ファイル構成

表 2.1.1 ファイル構成

ファイル名	内容
config.h	使用機能設定定義

2.1.1.2. 仕様

3. 機能選択

使用する機能を設定するための config 一覧を表 2.1.2 に示します。

表 2.1.1 機能設定 config

No	Config 名	説明
1	CONFIG_FUNC_I2C0	I2C0 機能選択 0:PIO 1:I2C
2	CONFIG_FUNC_SPI0	SPI0 機能選択 0:PIO 1:SPI0
3	CONFIG_FUNC_SPI1	SPI1 機能選択 0:PIO 1:SPI1
4	CONFIG_FUNC_SPI2	SPI2 機能選択 0:PIO 1:SPI2
5	CONFIG_FUNC_SPI3	SPI3 機能選択 0:PIO 1:SPI3
6	CONFIG_FUNC_PWM0	PWM0 機能選択 0:PIO 1:PWM0
7	CONFIG_FUNC_PWM1	PWM1 機能選択 0:PIO 1:PWM1
8	CONFIG_FUNC_PWM2	PWM2 機能選択 0:PIO 1:PWM2
9	CONFIG_FUNC_UART0	UART0 機能選択 0:PIO 1:UART0
10	CONFIG_FUNC_UART1	UART1 機能選択 0:PIO 1:UART1
11	CONFIG_FUNC_TIMER0	TIMER0 機能選択 0:無効 1:有効
12	CONFIG_FUNC_TIMER1	TIMER1 機能選択 0:無効 1:有効
13	CONFIG_FUNC_TIMER2	TIMER2 機能選択 0:無効 1:有効
14	CONFIG_FUNC_LCDC	LCDC 機能選択 0:PIO 1:LCDC
15	CONFIG_FUNC_EINT0	EINT0 機能選択 0:PIO 1:EINT0
16	CONFIG_FUNC_EINT1	EINT1 機能選択 0:PIO 1:EINT1
17	CONFIG_FUNC_EINT2	EINT2 機能選択 0:PIO 1:EINT2
18	CONFIG_FUNC_EINT3	EINT3 機能選択 0:PIO 1:EINT3
19	CONFIG_FUNC_EINT4	EINT4 機能選択 0:PIO 1:EINT4
20	CONFIG_FUNC_EINT5	EINT5 機能選択 0:PIO 1:EINT5
21	CONFIG_FUNC_EINT6	EINT6 機能選択 0:PIO 1:EINT6
22	CONFIG_FUNC_EINT7	EINT7 機能選択 0:PIO 1:EINT7

3.1.1.1.1. 使用機能チャンネル数設定

使用する機能のチャンネルの数を設定するための config 一覧を表 2.1.3 に示します。

表 2.1.2 使用機能チャンネル数設定 config

No	Config 名	説明
1	CONFIG_I2C_NUM	I2C のチャンネル数 default:1
2	CONFIG_SPI_NUM	SPI のチャンネル数 default:3
3	CONFIG_PWM_NUM	PWM のチャンネル数 default:2
4	CONFIG_UART_NUM	UART のチャンネル数 default:2
5	CONFIG_TIMER_NUM	TIMER のチャンネル数 default:3

3.1.1.1.2. 使用デバイス選択

KEIM-SEB に搭載しているデバイスを使用するために設定する config 一覧を表 2.1.4 に示します。

表 2.1.3 使用デバイス config

No	Config 名	説明
1	CONFIG_DEV_I2C_NUM	I2C で使用するデバイス数
2	CONFIG_DEV_I2C_RTC_8564	RTC-8564 使用 0:しない 1:する
3	CONFIG_DEV_I2C_BR24G32	BR24G32 使用 0:しない 1:する
4	CONFIG_DEV_I2C_ADT7410	ADT7410 使用 0:しない 1:する
5	CONFIG_DEV_I2C_MCP23017	MCP23017 使用 0:しない 1:する
6	CONFIG_DEV_SPI_W5500	W5500 使用 0:しない 1:する
7	CONFIG_DEV_SPI_ADC108S022	ADC108S022 使用 0:しない 1:する
8	CONFIG_DEV_SPI_DAC082S085	DAC082S085 使用 0:しない 1:する
9	CONFIG_DEV_SPI_TSC2046	TSC2046 使用 0:しない 1:する
10	CONFIG_DEV_PWM_BUZZER	BUZZER 使用 0:しない 1:する
11	CONFIG_DEV_PWM_BACKLIGHT	BACKLIGHT 使用 0:しない 1:する
12	CONFIG_DEV_PWM_SERVO	SERVO 使用 0:しない 1:する
13	CONFIG_DEV_UART_RS232C	RS232C 使用 0:しない 1:する
14	CONFIG_DEV_UART_USB	USB 使用 0:しない 1:する
15	CONFIG_DEV_LCD	LCD 使用 0:しない 1:する

3.2. 端子設定

3.2.1. 概要

端子の一覧を定義します。

3.2.2. ファイル構成

表 3.2.1 ファイル構成

ファイル名	内容
KEIMSoM-PIO.h	PIO レジスタアクセス定義 端子定義

3.2.3. 仕様

3.2.3.1. 端子一覧

端子の一覧を表 2.2.2 に示します。

表 3.2.2 端子一覧

No	Port	機能 1	機能 2	No	Port	機能 1	機能 2
1	A	PA0	UART0_TXD	33	C	PC0	LCD_D0
2		PA1	UART0_RXD	34		PC1	LCD_D1
3		PA2	UART0_RTS	35		PC2	LCD_D2
4		PA3	UART0_CTS	36		PC3	LCD_D3
5		PA4	UART1_TXD	37		PC4	LCD_D4
6		PA5	UART1_RXD	38		PC5	LCD_D5
7		PA6	UART1_RTS	39		PC6	LCD_D6
8		PA7	UART1_CTS	40		PC7	LCD_D7
9		PA8	SPI0_SCLK	41		PC8	LCD_D8
10		PA9	ETN_SCSn	42		PC9	LCD_D9
11		PA10	SPI0_MOSI	43		PC10	LCD_D10
12		PA11	SPI0_MISO	44		PC11	LCD_D11
13		PA12	SPI1_SCLK	45		PC12	LCD_D12
14		PA13	ADC_SSELn	46		PC13	LCD_D13
15		PA14	SPI1_MOSI	47		PC14	LCD_D14
16		PA15	SPI1_MISO	48		PC15	LCD_D15
17	B	PB0	SPI2_SCLK	49	D	PD0	ETN_INTn
18		PB1	DAC_SSELn	50		PD1	RTCINTn
19		PB2	SPI2_MOSI	51		PD2	TPC_INTn
20		PB3	-	52		PD3	TPC_BUSY
21		PB4	SPI3_SCLK	53		PD4	PUSHSW0
22		PB5	TPC_SSELn	54		PD5	PUSHSW1
23		PB6	SPI3_MOSI	55		PD6	PUSHSW2
24		PB7	SPI3_MISO	56		PD7	PUSHSW3
25		PB8	PWM0	57		PD8	ADCIN1
26		PB9	PWM1	58		PD9	ADCIN2
27		PB10	PWM2	59		PD10	ADCIN3
28		PB11	LCD_DISP	60		PD11	ADCIN4
29		PB12	LCD_CLK	61		PD12	ADCIN5
30		PB13	LCD_HSD	62		PD13	ADCIN6
31		PB14	LCD_VSD	63		PD14	ADCIN7
32		PB15	LCD_DEN	64		PD15	ADCIN8

No	Port	機能 1	機能 2	No	Port	機能 1	機能 2
65	E	PE0	I2C_SCL				
66		PE1	I2C_SDA				

3.3. ボード初期化

3.3.1. 概要

使用するデバイスおよびインタフェースの初期化、端子の設定をします。

3.3.2. ファイル構成

表 3.3.1 ファイル構成

ファイル名	内容
board_init.h	各ヘッダファイル 割り込み関数定義
board_init.c	ボード初期化処理関数

3.3.3. 仕様

・各インタフェースの情報登録と初期化

・使用デバイスの情報登録

・使用インタフェースの端子設定

3.3.3.1. 各インタフェースの情報登録および使用デバイスの情報登録

3.3.3.1.1. I2C

I2C の情報登録および使用デバイス情報登録の詳細を表 2.3.2 に示します。

表 3.3.2 I2C 情報登録

変数		値(例)	説明
device		I2C_x_NAME	I2C のデバイス名 system.h で定義されている名前を指定
address		I2C_x_BASE	I2C のアドレス system.h で定義されている値を指定
clk		80000000	I2C に入力されているクロックを指定
speed		100000	I2C の周波数を指定
irq			割り込みの情報
	ic_id	I2C_x_IRQ_INTERRUPT_CONTROLLER_ID	割り込みの ID system.h で定義されている値を指定
	irq	I2C_x_IRQ	割り込みの番号 system.h で定義されている値を指定
	isr	(void *)xxx	割り込み時の関数名
dev_info			接続先のデバイスの情報
	device_name	“xxx”	デバイス名を文字列で指定
	slave_addr	0xxx	デバイスのスレーブアドレスを 7bit で指定
	config	I2C_ADD16	デイバスのオプション I2C_ADD16：16bit アドレス

3.3.3.1.2. SPI

SPI の情報登録および使用デバイス情報登録の詳細を表 2.3.3 に示します。

表 3.3.3 SPI 情報登録

変数		値(例)	説明
device		SPI_x_NAME	SPI のデバイス名 system.h で定義されている名前を指定
address		SPI_x_BASE	SPI のアドレス system.h で定義されている値を指定
clk		80000000	SPI に入力されているクロックを指定
speed		2000000	SPI の周波数を指定
spi_mode		SPI_MODEx	SPI の転送モードを指定 SPI_MODE0:CPOL=CHPA=0 SPI_MODE1:CPOL=0,CHPA=1 SPI_MODE2:CPOL=1,CHPA=0 SPI_MODE3:CPOL=CHPA=1
irq			割り込みの情報
	ic_id	SPI_x_IRQ_INTERRUPT_CONTROLLER_ID	割り込みの ID system.h で定義されている値を指定
	irq	SPI_x_IRQ	割り込みの番号 system.h で定義されている値を指定
	isr	(void *)xxx	割り込み時の関数名
dev_info			接続先のデバイスの情報
	device_name	“xxx”	デバイス名を文字列で指定
	pio_cs	Pxx	デバイスの CS として使用する PIO のポート番号を指定

3.3.3.1.3. PWM

PWM の情報登録および使用デバイス情報登録の詳細を表 2.3.4 に示します。

表 3.3.4 PWM 情報登録

変数		値(例)	説明
device		PWM_x_NAME	PWM のデバイス名 system.h で定義されている名前を指定
address		PWM_x_BASE	PWM のアドレス system.h で定義されている値を指定
clk		80000000	PWM に入力されているクロックを指定
speed		3906	PWM の周波数を指定
dev_info			接続先のデバイスの情報
	device_name	“xxx”	デバイス名を文字列で指定

3.3.3.1.4. UART

UART の情報登録および使用デバイス情報登録の詳細を表 2.3.5 に示します。

表 3.3.5 UART 情報登録

変数		値(例)	説明
device		UART_x_NAME	UART のデバイス名 system.h で定義されている名前を指定
address		UART_x_BASE	UART のアドレス system.h で定義されている値を指定
clk		80000000	UART に入力されているクロックを指定
baud		2000000	UART の周波数を指定
irq			割り込みの情報
	ic_id	UART_x_IRQ_INTERRUPT_CONTROLLER_ID	割り込みの ID system.h で定義されている値を指定
	irq	UART_x_IRQ	割り込みの番号 system.h で定義されている値を指定
	isr	(void *)uartx_irq_event	受信割り込み時の関数名
	irq_event	(void *)xxx	受信割り込み時のユーザーの関数名 受信割り込み発生時に uartx_irq_event から Jump する関数を指定
dev_info			接続先のデバイスの情報
	device_name	“xxx”	デバイス名を文字列で指定

3.3.3.1.5. TIMER

TIMER の情報登録および使用デバイス情報登録の詳細を表 2.3.6 に示します。

表 3.3.6 TIMER 情報登録

変数		値(例)	説明
device		TIMER_x_NAME	TIMER のデバイス名 system.h で定義されている名前を指定
address		TIMER_x_BASE	TIMER のアドレス system.h で定義されている値を指定
freq		TIMER_x_FREQ	TIMER に入力されているクロックを指定 system.h で定義されている値を指定
timer_mode		xxx	TIMER の動作モードを指定 TIMER_MODE_CONT: 0 に戻っても繰り返しカウントダウン TIMER_MODE_ONCE: 0 になったらカウントストップ
period		xxx	TIMER のカウンタ値を指定
irq			割り込みの情報
	ic_id	TIMER_x_IRQ_INTERRUPT_CONTROLLER_ID	割り込みの ID system.h で定義されている値を指定
	irq	TIMER_x_IRQ	割り込みの番号 system.h で定義されている値を指定
	isr	(void *)xxx	割り込み時の関数名

3.3.3.1.6. WDT

WDT の情報登録および使用デバイス情報登録の詳細を表 2.3.7 に示します。

表 3.3.7 WDT 情報登録

変数	値(例)	説明
device	WDT_NAME	WDT のデバイス名 system.h で定義されている名前を指定
address	WDT_BASE	WDT のアドレス system.h で定義されている値を指定
freq	WDT_FREQ	WDT に入力されているクロックを指定 system.h で定義されている値を指定
period	xxx	WDT のカウンタ値を指定

3.3.3.1.7. EINT

EINT の情報登録および使用デバイス情報登録の詳細を表 2.3.8 に示します。

表 3.3.8 EINT 情報登録

変数			値(例)	説明
address			EINT_BASE	EINT のアドレス system.h で定義されている値を指定
irq_info				EINT の情報
	num		xxx	EINT の番号
	int_mode		MODE_xxx	EINT のモードを指定 MODE_LEVEL、MODE_EDGE
	int_pol		POL_xxx	EINT の極性を指定 POL_HIGH、POL_LOW
	irq			割り込みの情報
		ic_id	0	ID は 0 で固定
		irq	EINT_NUM(EINTx)	割り込みの番号 EINTx に EINT 番号を指定
		isr	(void *)eintx_irq_event	割り込み時の関数名

3.3.3.2. 使用インタフェースの端子設定

表 2.2.2 端子一覧の機能 2 の設定として使用しない場合、端子の初期設定は PIO 入力端子設定となります。

PIO の API を使用して端子の設定を行います。

PIO の API 仕様の詳細に関しましては 2.5.2.7.PIO API 仕様を参照してください。

3.3.3.2.1. I2C0

```
pio_request(PE0, PORT_FUNC); // SCL
pio_request(PE1, PORT_FUNC); // SDA
```

3.3.3.2.2. SPI0

```
pio_request(PA8, PORT_FUNC); // CLK
pio_request(PA9, PORT_PIO); // CS
pio_direction(PA9, PIO_DIR_OUT); // CS output set
pio_output(PA9, 1); // default high
pio_request(PA10, PORT_FUNC); // MOSI
pio_request(PA11, PORT_FUNC); // MISO
```

3.3.3.2.3. SPI1

```
pio_request(PA12, PORT_FUNC); // CLK
pio_request(PA13, PORT_PIO); // CS
pio_direction(PA13, PIO_DIR_OUT); // CS output set
pio_output(PA13, 1); // default high
pio_request(PA14, PORT_FUNC); // MOSI
pio_request(PA15, PORT_FUNC); // MISO
```

3.3.3.2.4. SPI2

```
pio_request(PB0, PORT_FUNC); // CLK
pio_request(PB1, PORT_PIO); // CS
pio_direction(PB1, PIO_DIR_OUT); // CS output set
pio_output(PB1, 1); // default high
pio_request(PB2, PORT_FUNC); // MOSI
```

3.3.3.2.5. SPI3

```
pio_request(PB4, PORT_FUNC); // CLK
pio_request(PB5, PORT_PIO); // CS
pio_direction(PB5, PIO_DIR_OUT); // CS output set
pio_output(PB5, 1); // default high
pio_request(PB6, PORT_FUNC); // MOSI
pio_request(PB7, PORT_FUNC); // MISO
```

3.3.3.2.6. PWM0

```
pio_request(PB8, PORT_FUNC); // PWM
```

3.3.3.2.7. PWM1

```
pio_request(PB9, PORT_FUNC); // PWM
```

3.3.3.2.8. PWM2

```
pio_request(PB10, PORT_FUNC); // PWM
```

3.3.3.2.9. UART0

```
pio_request(PA0, PORT_FUNC); // TXD
pio_request(PA1, PORT_FUNC); // RXD
pio_request(PA2, PORT_FUNC); // RTS
pio_request(PA3, PORT_FUNC); // CTS
```

3.3.3.2.10. UART1

```
pio_request(PA4, PORT_FUNC); // TXD
pio_request(PA5, PORT_FUNC); // RXD
pio_request(PA6, PORT_FUNC); // RTS
pio_request(PA7, PORT_FUNC); // CTS
```

3.3.3.2.11. LCDC

```
pio_request(PB12, PORT_FUNC); // CLK
pio_request(PB13, PORT_FUNC); // HSD
pio_request(PB14, PORT_FUNC); // VSD
pio_request(PB15, PORT_FUNC); // DEN
pio_request(PC0, PORT_FUNC); // D0
pio_request(PC1, PORT_FUNC); // D1
pio_request(PC2, PORT_FUNC); // D2
pio_request(PC3, PORT_FUNC); // D3
pio_request(PC4, PORT_FUNC); // D4
pio_request(PC5, PORT_FUNC); // D5
pio_request(PC6, PORT_FUNC); // D6
pio_request(PC7, PORT_FUNC); // D7
pio_request(PC8, PORT_FUNC); // D8
pio_request(PC9, PORT_FUNC); // D9
pio_request(PC10, PORT_FUNC); // D10
pio_request(PC11, PORT_FUNC); // D11
pio_request(PC12, PORT_FUNC); // D12
pio_request(PC13, PORT_FUNC); // D13
pio_request(PC14, PORT_FUNC); // D14
pio_request(PC15, PORT_FUNC); // D15
```

3.3.3.2.12. EINT0

```
pio_request(PD0, PORT_FUNC); // EINT0
```

3.3.3.2.13. EINT1

```
pio_request(PD1, PORT_FUNC); // EINT1
```

3.3.3.2.14. EINT2

```
pio_request(PD2, PORT_FUNC); // EINT2
```

3.3.3.2.15. EINT3

```
pio_request(PD3, PORT_FUNC); // EINT3
```

3.3.3.2.16. EINT4

```
pio_request(PD4, PORT_FUNC); // EINT4
```

3.3.3.2.17. EINT5

pio_request(PD5, PORT_FUNC); // EINT5

3.3.3.2.18. EINT6

pio_request(PD6, PORT_FUNC); // EINT6

3.3.3.2.19. EINT7

pio_request(PD7, PORT_FUNC); // EINT7

3.4. 割り込み

3.4.1. 概要

各デバイスおよびインタフェースデバイスからの割り込みハンドラーの登録、割り込み許可/禁止を制御します。

3.4.2. ファイル構成

表 3.4.1 ファイル構成

ファイル名	内容
intc.h	割り込み処理の関数定義
intc.c	割り込み処理関数

3.4.3. 仕様

・優先順位は固定です。

・割り込みは全部で 19 本。

そのうち外部割込みは 8 本。

優先順位を表 2.4.2 に示します。

表 3.4.2 割り込み優先順位

IRQ	デバイス	優先順位
0	WDT	高い
1	EINT0	
2	EINT1	
3	EINT2	
4	EINT3	
5	EINT4	
6	EINT5	
7	EINT6	
8	EINT7	
9	Timer_0	
10	Timer_1	
11	Timer_2	
12	SPI_0	
13	SPI_1	
14	SPI_2	
15	SPI_3	
16	UART_0	
17	UART_1	低い
18	I2C_0	

3.4.4. 関数仕様の説明

alt_ic_isr_register()、alt_ic_irq_enable()、alt_ic_irq_disable()関数の詳細は

https://www.altera.co.jp/ja_JP/pdfs/literature/an/an595_j.pdf

を参照してください。

3.4.4.1. irq_request 関数

irq_request 関数の機能と引数の説明を表 2.4.3 に示します。

表 3.4.3 irq_request 関数の機能と引数

void irq_request(struct irq_info *info)		
1	機能	
	この関数はドライバの割り込みを割り込みハンドラーに登録するための関数	
2	入力パラメータ	
	info	board_init で定義したポインタ
	info->ic_id	割り込みの ID 番号
	info->irq	割り込み番号
	info->isr	割り込み発生時のジャンプする関数名
3	戻り値	
	なし	

3.4.4.2. irq_enable 関数

irq_enable 関数の機能と引数の説明を表 2.4.4 に示します。

表 3.4.4 irq_enable 関数の機能と引数

int irq_enable(struct irq_info *info)		
1	機能	
	この関数は割り込みハンドラーに登録した割り込みの許可関数	
2	入力パラメータ	
	info	board_init で定義したポインタ
	info->ic_id	割り込みの ID 番号
	info->irq	割り込み番号
3	戻り値	
	return alt_ic_irq_enable(info->ic_id, info->irq)	

3.4.4.3. irq_disable 関数

irq_disable 関数の機能と引数の説明を表 2.4.5 に示します。

表 3.4.5 irq_disable 関数の機能と引数

void irq_disable(struct irq_info *info)		
1	機能	
	この関数は割り込みハンドラーに登録した割り込みの禁止関数	
2	入力パラメータ	
	info	board_init で定義したポインタ
	info->ic_id	割り込みの ID 番号
	info->irq	割り込み番号
3	戻り値	
	なし	

3.5. API

3.5.1. 概要

ユーザーからの各デバイスに対するアクセスを制御します。

3.5.2. 仕様

3.5.2.1. UART

3.5.2.1.1. uart_xfer 関数

3.5.2.1.1.1. **関数仕様説明**

uart_xfer 関数の仕様を表 2.5.1、フローチャートを図 2.5.1 に示します。

表 3.5.1 uart_xfer 関数仕様

char *uart_xfer(char *device_name, char *data, alt_u8 type, alt_u8 nbyte)		
1	機能	
	UART で接続されているターゲットデバイスに対して データのリード/ライトを行う関数	
2	入力パラメータ	
	device	ターゲットのデバイス名 board_init で定義したデバイス名と同じものを指定します。
	data	リード/ライトするデータを格納するポインタ
	type	送信:0 受信:1
	nbyte	転送するバイト数
3	戻り値	
	0	データ転送完了
	ENODEV	指定したデバイス名が不一致

3.5.2.1.1.2. **フローチャート**

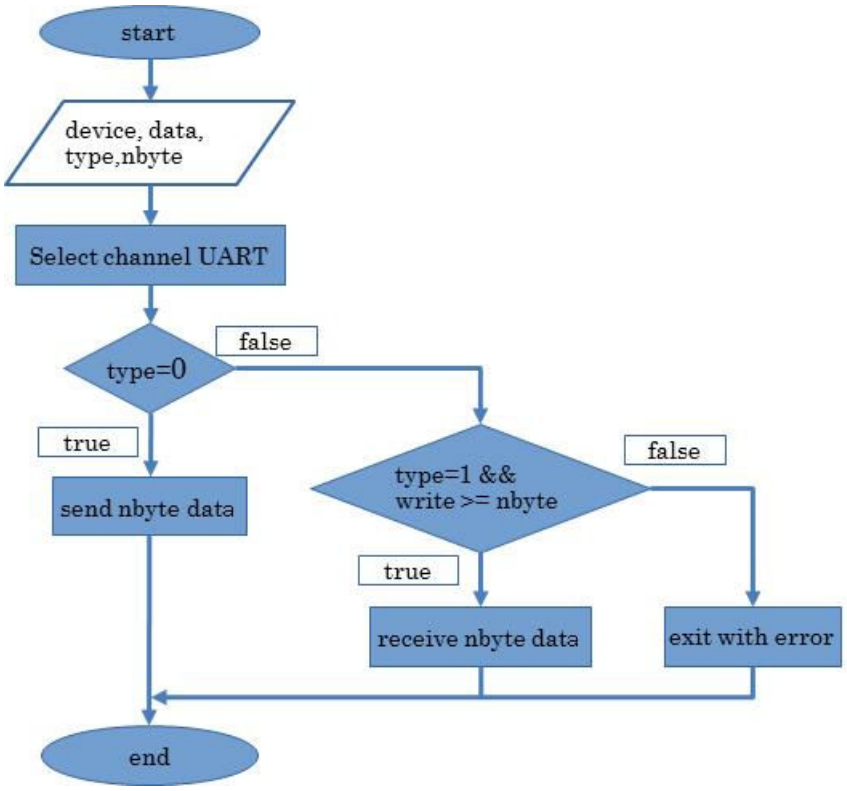


図 3.5.1 uart_xfer 関数フローチャート

3.5.2.1.1.3. **使用例**

データをターゲットに転送する例

入力パラメータ

device：”UART0”

data ：データを格納するポインタ

type ：0

nbyte ：5

プログラムには以下のようなコードを記述します。

```
{
        char msg[]=”12345”;
        uart_xfer(“UART0”, msg, 0, sizeof(msg));
}
```

3.5.2.1.1.4. **使用時の注意事項**

UART は NiosII の BSP の設定で 1ch だけ<stdio.h>を使用した標準出力に設定した場合、
設定した ch は uart_xfer()関数は使用できません。

3.5.2.1.2. uart_baud 関数

3.5.2.1.2.1. **関数仕様説明**

uart_baud 関数の仕様を表 2.5.2、フローチャートを図 2.5.2 に示します。

表 3.5.2 uart_baud 関数仕様

char *uart_baud(char *device, alt_u32 baud)		
1	機能	
	UART のボーレートを変更する関数	
2	入力パラメータ	
	device	デバイスファイル “/dev/uart_x”(x=0,1…)
	baud	ボーレートの指定
3	戻り値	
	0	ボーレート設定完了
	ENODEV	指定したデバイスファイルが不一致

3.5.2.1.2.2. **フローチャート**

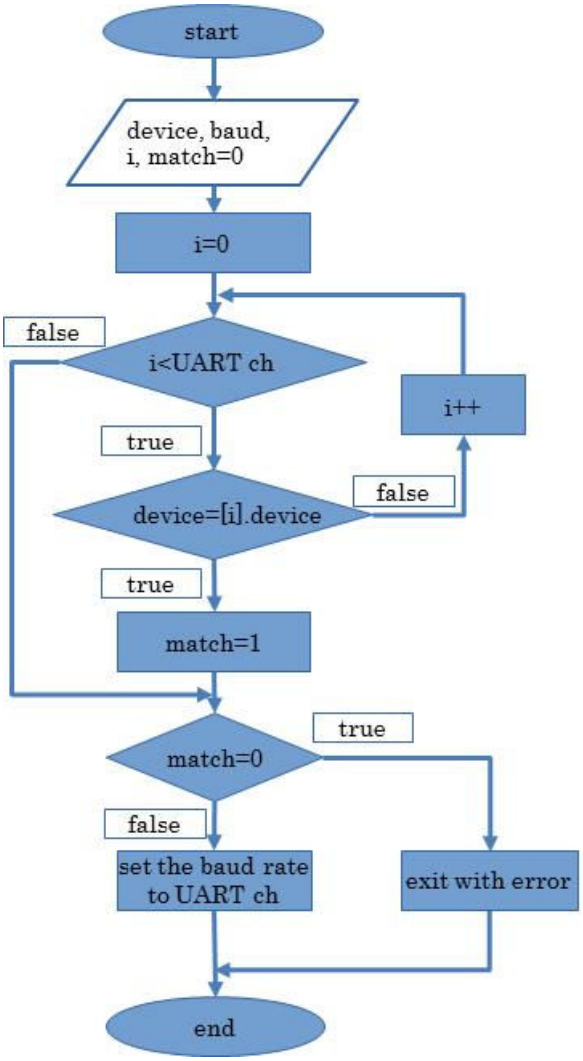


図 3.5.2 uart_baud 関数フローチャート

3.5.2.1.2.3. **使用例**

ボーレートを 115200bps に変更する例

入力パラメータ:

baud :115200

device :"/dev/uart_0"

プログラムには以下のようなコードを記述します。

```
{
        uart_baud("/dev/uart_0", 115200);
}
```

3.5.2.2. I2C

3.5.2.2.1. i2c_xfer 関数

3.5.2.2.1.1. **関数仕様説明**

i2c_xfer 関数の仕様を表 2.5.3、フローチャートを図 2.5.3 に示します。

表 3.5.3 i2c_xfer 関数仕様

alt_u8 i2c_xfer(char *device_name, alt_u8 *addr, alt_u8 *data, alt_u8 dir, alt_u8 nbyte)		
1	機能	
	I2C で接続されているターゲットデバイスに対して データのリード/ライトを行う関数	
2	入力パラメータ	
	device_name	ターゲットのデバイス名 board_init で定義したデバイス名と同じものを指定する
	addr	レジスタアドレスを格納するポインタ
	data	リード/ライトするデータを格納するポインタ
	dir	方向ビット ライト:0 リード:1
	nbyte	転送するバイト数
3	戻り値	
	0	ACK
	1	NACK
	ENODEV	指定したデバイス名が不一致

3.5.2.2.1.2. **フローチャート**

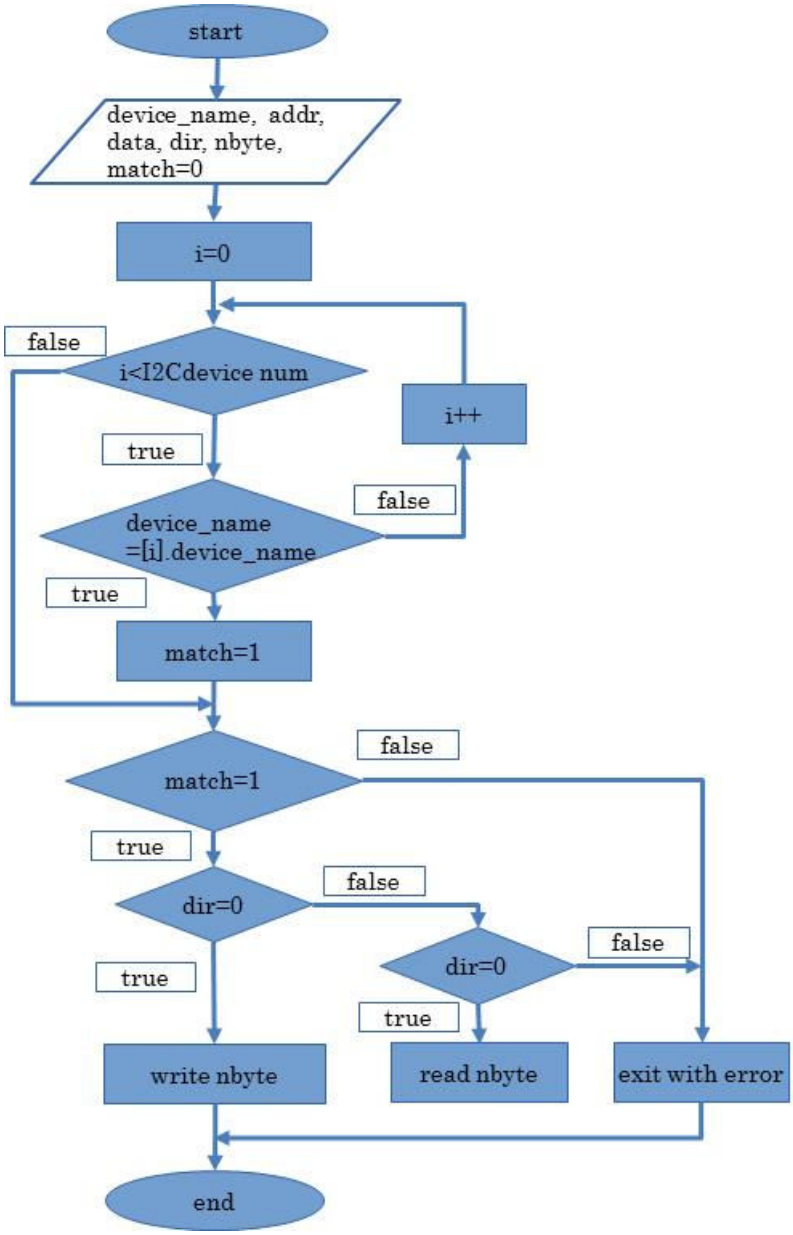


図 3.5.3 i2c_xfer 関数フローチャート

3.5.2.2.1.3. **使用例**

RTC_8564 デバイスに 2byte データのリード/ライトをする例

プログラムには以下のようなコードを記述します。

```
{
        unsigned char i2c_bufwr[] = {0x35,0x55};
        unsigned char i2c_bufre[] = {0,0};
        unsigned char addr[2]       = {2,0};
        /* Set second,minute, data for RTC */
        i2c_xfer("RTC_8564", addr, i2c_bufwr, 0, 7 );
        /* Read second,minute,hour,day data of RTC */
        i2c_xfer("RTC_8564", addr, i2c_bufre, 1, 7);
}
```

3.5.2.2.2. i2c_speed 関数

3.5.2.2.2.1. **関数仕様説明**

i2c_speed 関数の仕様を表 2.5.4、フローチャートを図 2.5.4 に示します。

表 3.5.4 i2c_speed 関数仕様

alt_u8 i2c_speed(char *device, alt_u32 speed)		
1	機能	
	I2C の転送レートを変更する関数	
2	入力パラメータ	
	device	デバイスファイル “/dev/i2c_x”(x=0,1…)
	speed	変更する転送レート
3	戻り値	
	0	設定更新完了
	ENODEV	指定したデバイスが不一致

3.5.2.2.2.2. **フローチャート**

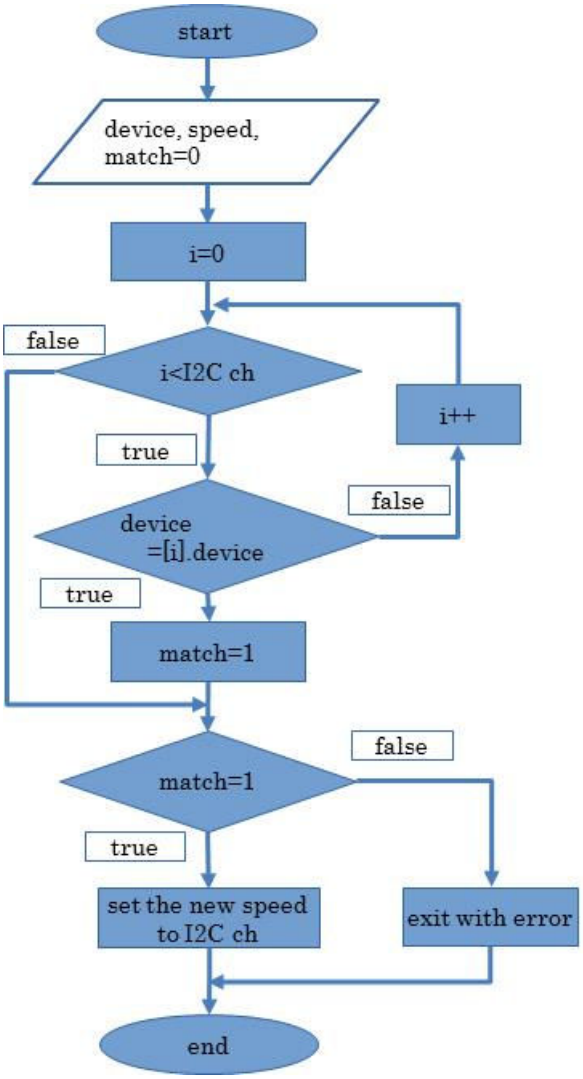


図 3.5.4 i2c_speed 関数フローチャート

3.5.2.2.2.3. **使用例**

I2C0 の転送レートを 400kbps に変更する例

プログラムには以下のようなコードを記述します。

```
{
        /* Change speed to 400K and read data */
        i2c_speed("/dev/i2c_0", 400000);
}
```

3.5.2.3. SPI

3.5.2.3.1. spi_xfer 関数

3.5.2.3.1.1. **関数仕様説明**

spi_xfer 関数の仕様を表 2.5.5、フローチャートを図 2.5.5 に示します。

表 3.5.5 spi_xfer 関数仕様

alt_u8 spi_xfer(char *device_name, alt_u8 *data, alt_u8 dir, alt_u8 nbyte)		
1	機能	
	SPI で接続されているターゲットデバイスに対して データのリード/ライトを行う関数	
2	入力パラメータ	
	device_name	ターゲットのデバイス名 board_init で定義したデバイス名と同じものを指定します。
	data	リード/ライトするデータを格納するポインタ
	dir	方向ビット ライト:0 リード:1
	nbyte	転送するバイト数
3	戻り値	
	0	データ転送完了
	ENODEV	指定したデバイス名が不一致

3.5.2.3.1.2. **フローチャート**

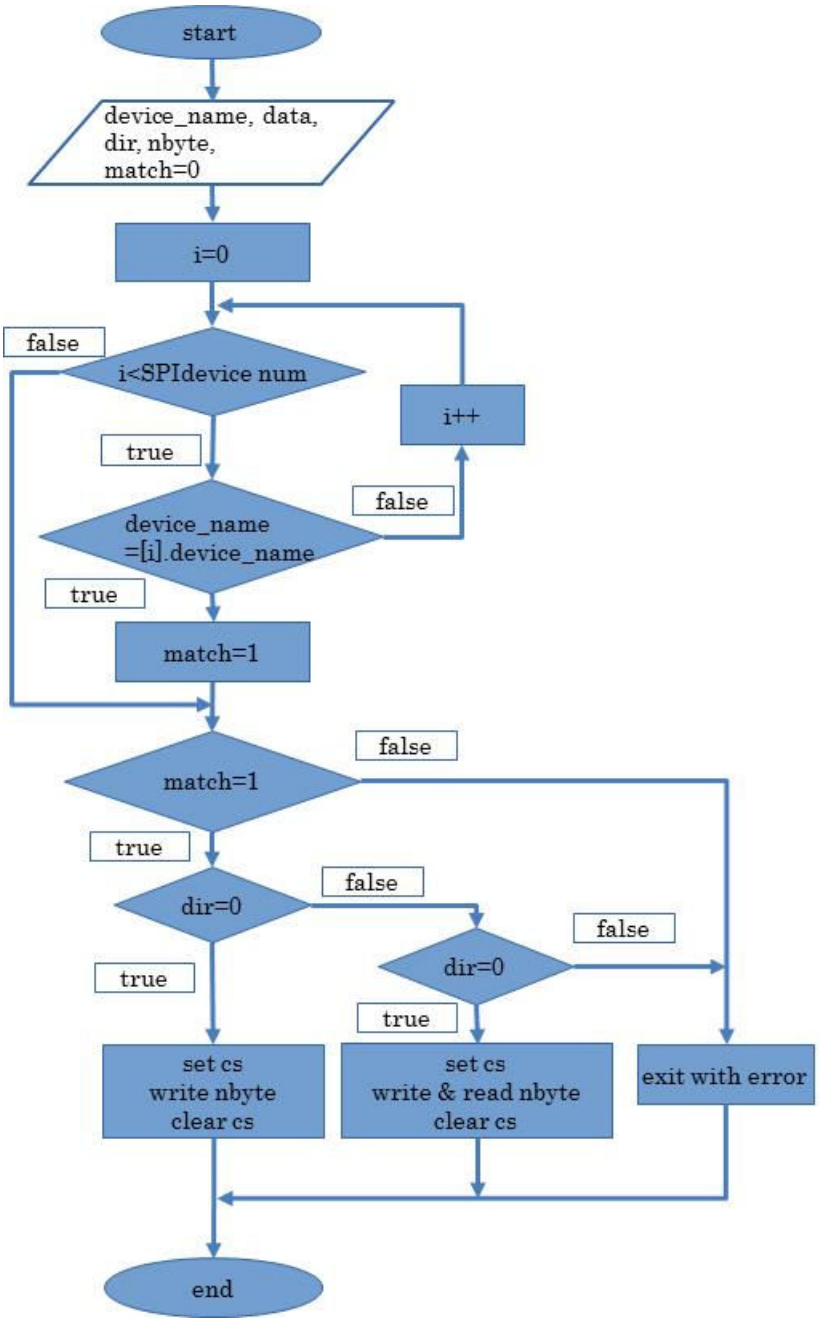


図 3.5.5 spi_xfer 関数フローチャート

3.5.2.3.1.3. **使用例**

SPI0 に接続されている W5500 に 3byte のデータをライトし 4byte データをリードする例

W5500 のアクセスはアドレス 2byte、コマンド 1byte、あとはデータ nbyte のプロトコルになっています。
上記の例はあるアドレスに対してリードを行うものです。

プログラムには以下のようなコードを記述します。

```
{
        alt_u8 eth_data[7];
        alt_u8 valid_data[4];

        /* addrss 0x0001 set */
        eth_data[0] = 0x00;
        eth_data[1] = 0x01;
        /* cmd 0x04 set */
        eth_data[2] = 0x02;

        spi_xfer("W5500", eth_data, 1, 7);

        /* valid read data */
        valid_data[0] = eth_data[3];
        valid_data[1] = eth_data[4];
        valid_data[2] = eth_data[5];
        valid_data[3] = eth_data[6];
}
```

3.5.2.3.2. spi_speed 関数

3.5.2.3.2.1. **関数仕様説明**

i2c_speed 関数の仕様を表 2.5.6、フローチャートを図 2.5.6 に示します。

表 3.5.6 spi_speed 関数仕様

alt_u8 spi_speed(char *device, alt_u32 speed)		
1	機能	
	SPI の CLK の周波数を変更する関数	
2	入力パラメータ	
	device	デバイスファイル “/dev/spi_x”(x=0,1…)
	speed	変更する CLK の周波数
3	戻り値	
	0	設定更新完了
	ENODEV	指定したデバイスが不一致

3.5.2.3.2.2. **フローチャート**

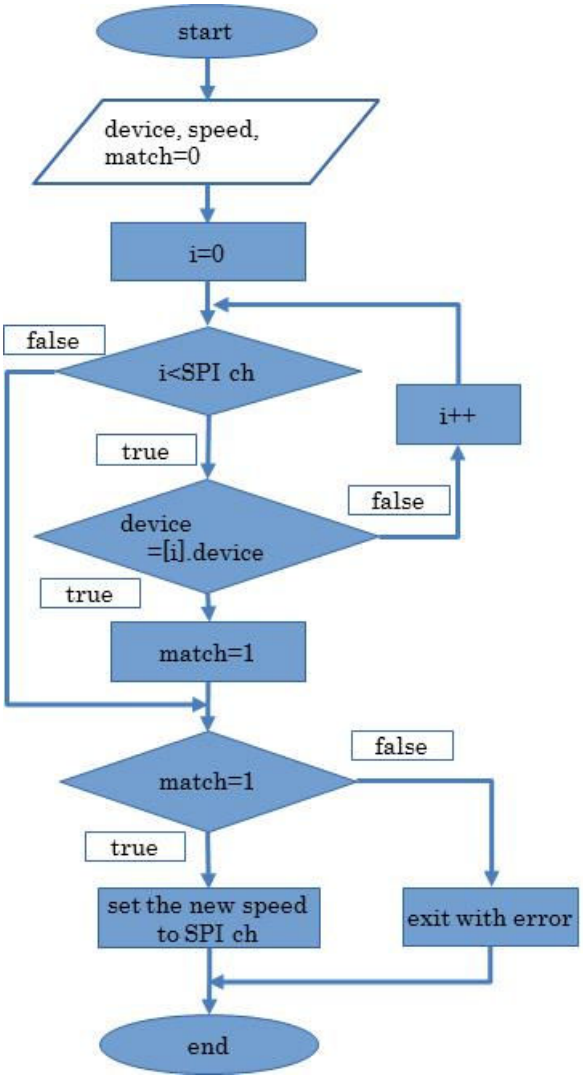


図 3.5.6 spi_speed 関数フローチャート

3.5.2.3.2.3. **使用例**

I2C0 の転送レートを 400kbps に変更する例

プログラムには以下のようなコードを記述します。

```
{
        /* Change speed to 400K and read data */
        i2c_speed("/dev/i2c_0", 400000);
}
```

3.5.2.4. PWM

3.5.2.4.1. pwm_start 関数

3.5.2.4.1.1. **関数仕様説明**

pwm_start 関数の仕様を表 2.5.7、フローチャートを図 2.5.7 に示します。

表 3.5.7 pwm_start 関数仕様

alt_u8 pwm_start(char *device)		
1	機能	
	PWM のパルス出力を開始する関数	
2	入力パラメータ	
	device	デバイスファイル “/dev/pwm_x”(x=0,1…)
3	戻り値	
	0	パルス出力開始完了
	ENODEV	指定したデバイスが不一致

3.5.2.4.1.2. **フローチャート**

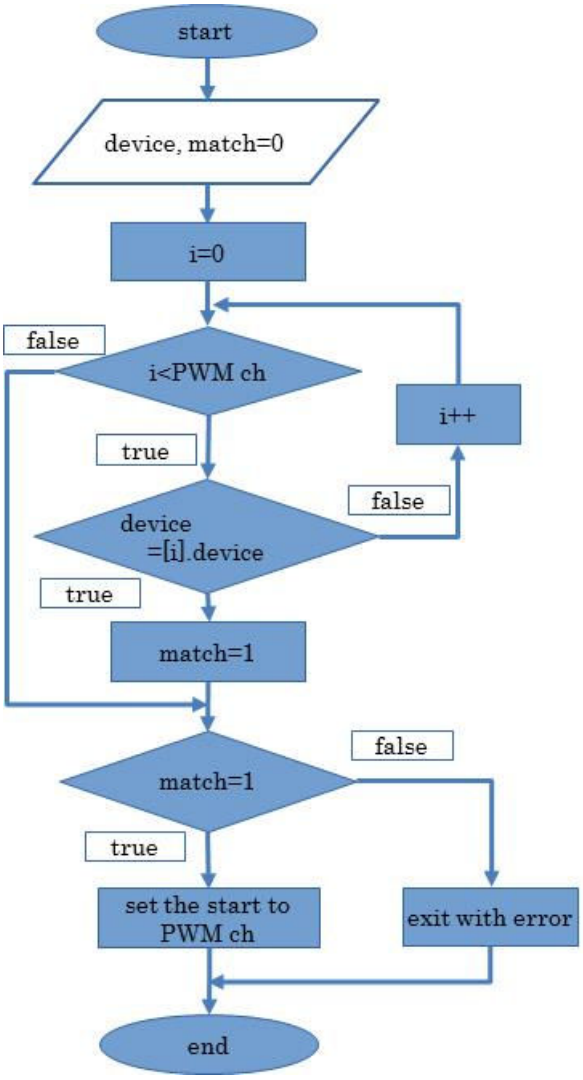


図 3.5.7 pwm_start 関数フローチャート

3.5.2.4.1.3. **使用例**

PWM0 のパルス出力を開始する例

プログラムには以下のようなコードを記述します。

```
{
        pwm_start("/dev/pwm_0");
}
```

3.5.2.4.2. pwm_stop 関数

3.5.2.4.2.1. **関数仕様説明**

pwm_stop 関数の仕様を表 2.5.8、フローチャートを図 2.5.8 に示します。

表 3.5.8 pwm_stop 関数仕様

alt_u8 pwm_stop(char *device)		
1	機能	
	PWM のパルス出力を停止する関数	
2	入力パラメータ	
	device	デバイスファイル “/dev/pwm_x”(x=0,1…)
3	戻り値	
	0	パルス出力停止完了
	ENODEV	指定したデバイスが不一致

3.5.2.4.2.2. **フローチャート**

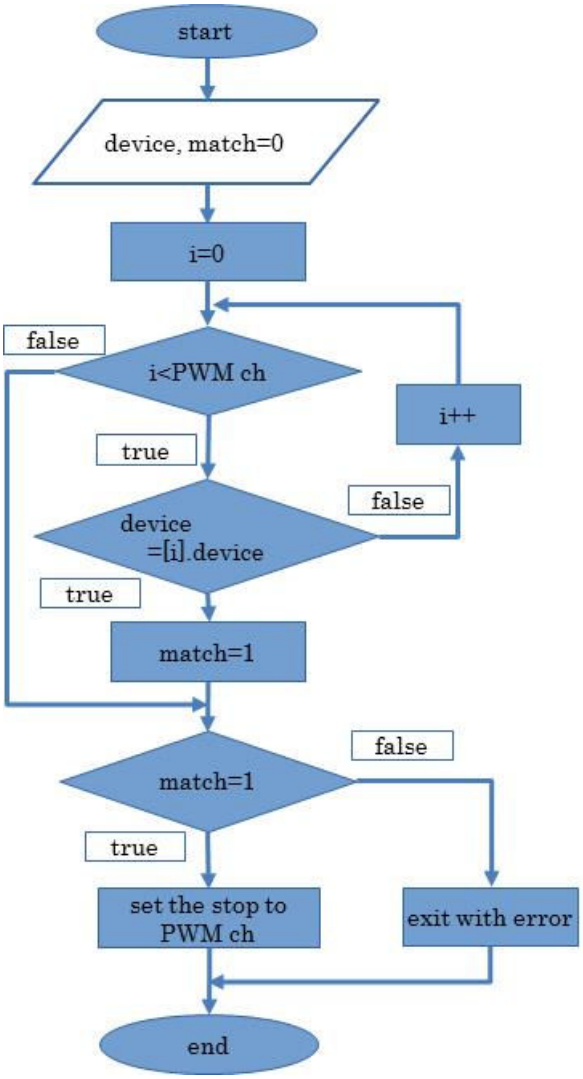


図 3.5.8 pwm_stop 関数フローチャート

3.5.2.4.2.3. **使用例**

PWM0 のパルス出力を停止する例

プログラムには以下のようなコードを記述します。

```
{
        pwm_stop("/dev/pwm_0");
}
```

3.5.2.4.3. pwm_speed 関数

3.5.2.4.3.1. **関数仕様説明**

pwm_speed 関数の仕様を表 2.5.9、フローチャートを図 2.5.9 に示します。

表 3.5.9 pwm_speed 関数仕様

lt_u8 pwm_speed(char *device, alt_u32 speed)		
1	機能	
	PWM のパルス周波数を変更する関数	
2	入力パラメータ	
	device	デバイスファイル “/dev/pwm_x”(x=0,1…)
	speed	パルス周波数を指定
3	戻り値	
	0	パルス周波数変更完了
	ENODEV	指定したデバイスが不一致

3.5.2.4.3.2. **フローチャート**

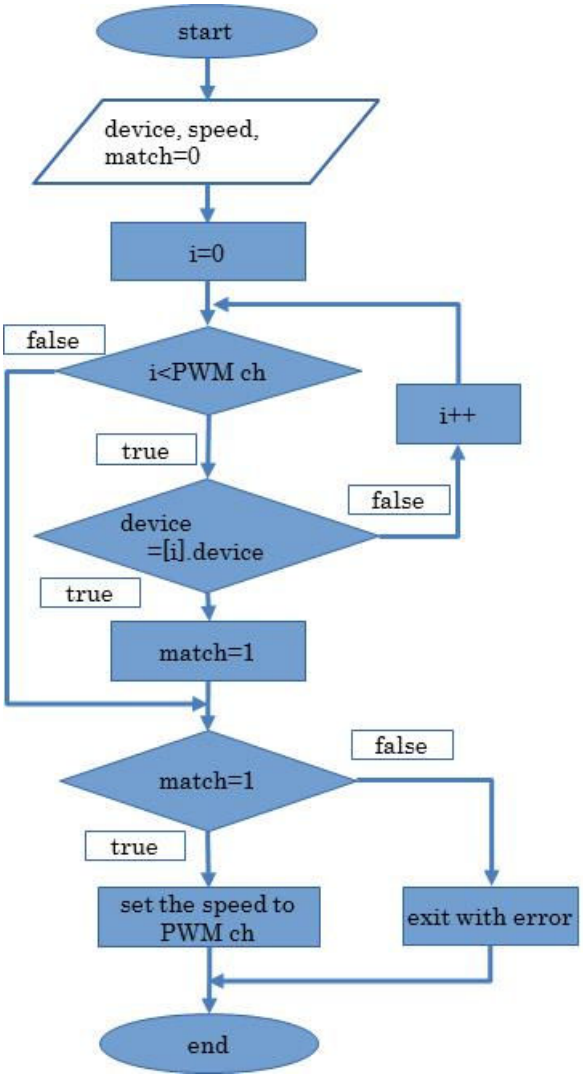


図 3.5.9 pwm_speed 関数フローチャート

3.5.2.4.3.3. **使用例**

PWM0 のパルス周波数を 3.906KHz(256us)に変更する例

プログラムには以下のようなコードを記述します。

```
{
        pwm_speed("/dev/pwm_0", 3906);
}
```

3.5.2.4.4. pwm_duty 関数

3.5.2.4.4.1. **関数仕様説明**

pwm_duty 関数の仕様を表 2.5.10、フローチャートを図 2.5.10 に示します。

表 3.5.10 pwm_duty 関数仕様

alt_u8 pwm_duty(char *device, alt_u8 duty)		
1	機能	
	PWM の duty を変更する関数	
2	入力パラメータ	
	device	デバイスファイル “/dev/pwm_x”(x=0,1…)
	duty	duty を指定 (0～100(%))
3	戻り値	
	0	duty 変更完了
	ENODEV	指定したデバイスが不一致

3.5.2.4.4.2. **フローチャート**

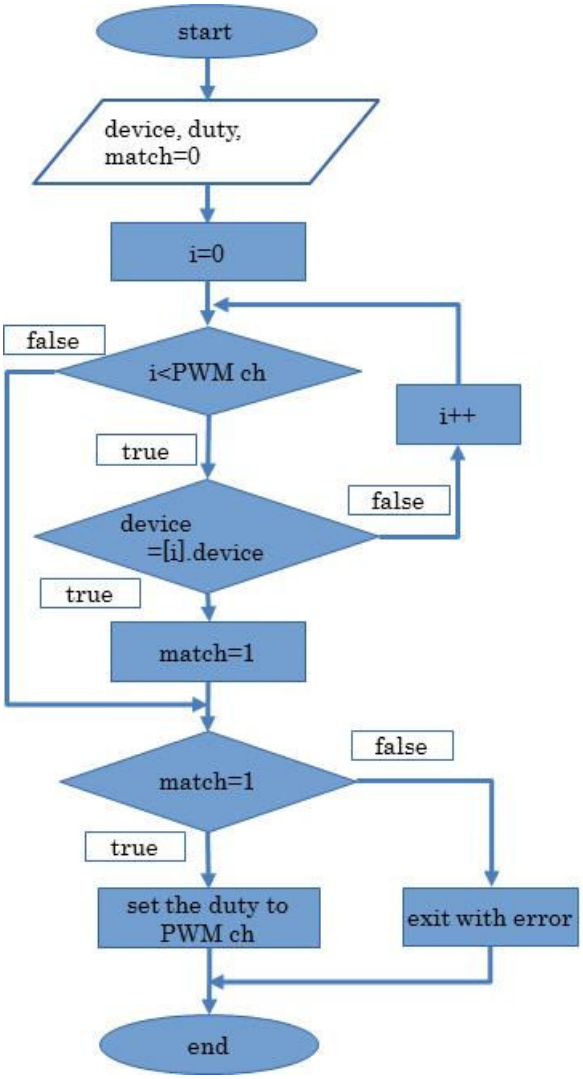


図 3.5.10 pwm_duty 関数フローチャート

3.5.2.5. TIMER

3.5.2.5.1. timer_start 関数

3.5.2.5.1.1. **関数仕様説明**

timer_start 関数の仕様を表 2.5.11、フローチャートを図 2.5.11 に示します。

表 3.5.11 timer_start 関数仕様

alt_u8 timer_start(char *device)		
1	機能	
	TIMER のカウンタを開始する関数	
2	入力パラメータ	
	device	デバイスファイル “/dev/timer_x”(x=0,1…)
3	戻り値	
	0	カウンタ開始完了
	ENODEV	指定したデバイスが不一致

3.5.2.5.1.2. **フローチャート**

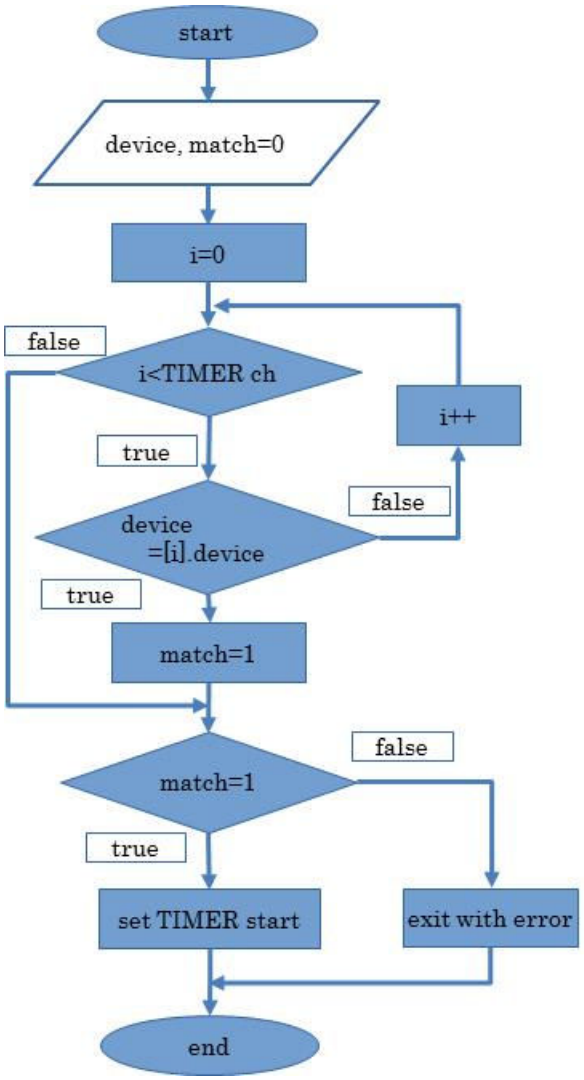


図 3.5.11 timer_start 関数フローチャート

3.5.2.5.1.3. **使用例**

TIMER0 のカウントを開始する例

プログラムには以下のようなコードを記述します。

```
{
        timer_start("/dev/timer_0");
}
```

3.5.2.5.2. timer_stop 関数

3.5.2.5.2.1. **関数仕様説明**

timer_stop 関数の仕様を表 2.5.12、フローチャートを図 2.5.12 に示します。

表 3.5.12 timer_stop 関数仕様

alt_u8 timer_stop(char *device)		
1	機能	
	TIMER のカウンタを停止する関数	
2	入力パラメータ	
	device	デバイスファイル “/dev/timer_x”(x=0,1…)
3	戻り値	
	0	カウンタ停止完了
	ENODEV	指定したデバイスが不一致

3.5.2.5.2.2. **フローチャート**

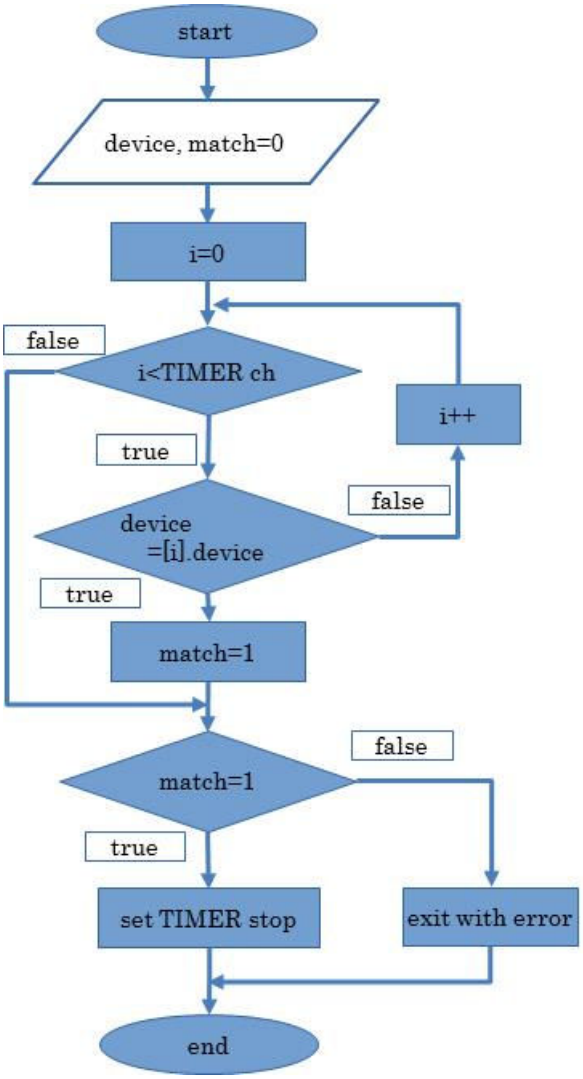


図 3.5.12 timer_stop 関数フローチャート

3.5.2.5.2.3. **使用例**

TIMER0 のカウントを停止する例

プログラムには以下のようなコードを記述します。

```
{
        timer_stop("/dev/timer_0");
}
```

3.5.2.5.3. timer_settime 関数

3.5.2.5.3.1. **関数仕様説明**

timer_settime 関数の仕様を表 2.5.13、フローチャートを図 2.5.13 に示します。

表 3.5.13 timer_settime 関数仕様

alt_u8 timer_settime(char *device, alt_u32 time)		
1	機能	
	TIMER の時間を設定する関数	
2	入力パラメータ	
	device	デバイスファイル “/dev/timer_x”(x=0,1…)
	time	時間(単位:us)を指定
3	戻り値	
	0	カウンタ停止完了
	ENODEV	指定したデバイスが不一致

3.5.2.5.3.2. **フローチャート**

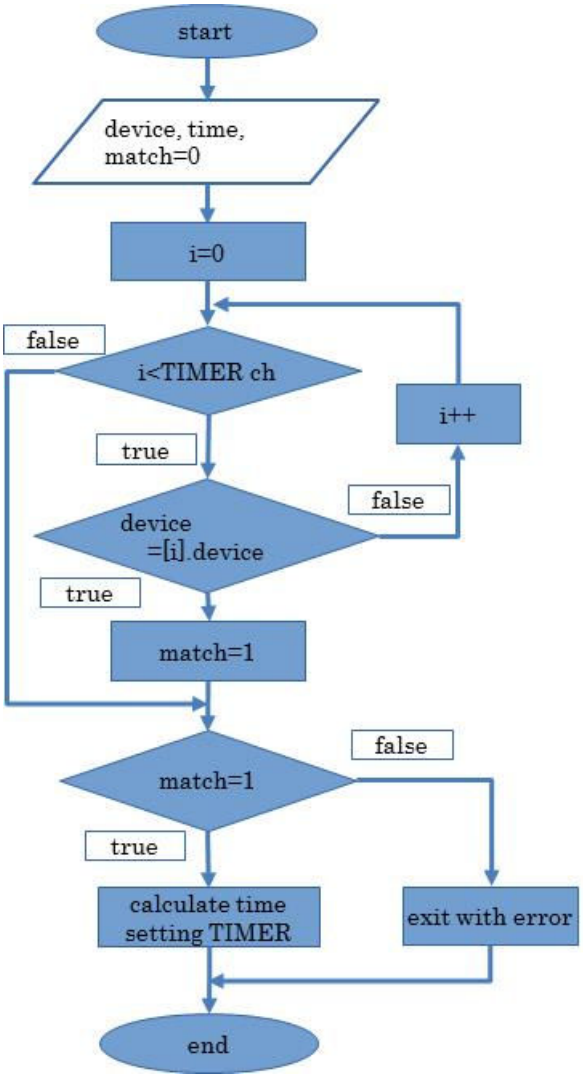


図 3.5.13 timer_settime 関数フローチャート

3.5.2.5.3.3. **使用例**

TIMER0 の時間を 10ms に設定する例

プログラムには以下のようなコードを記述します。

```
{
        timer_settime("/dev/timer_0", 10000);
}
```

3.5.2.5.4. timer_config 関数

3.5.2.5.4.1. **関数仕様説明**

timer_config 関数の仕様を表 2.5.14、フローチャートを図 2.5.14 に示します。

表 3.5.14 timer_config 関数仕様

alt_u8 timer_config(char *device, alt_u32 config)		
1	機能	
	TIMER の動作モードを設定する関数	
2	入力パラメータ	
	device	デバイスファイル “/dev/timer_x”(x=0,1…)
	config	動作モードを指定 モードの詳細は 2.3.3.1.5 TIMER の timer_mode の項目を参照ください。
3	戻り値	
	0	カウンタ停止完了
	ENODEV	指定したデバイスが不一致

3.5.2.5.4.2. **フローチャート**

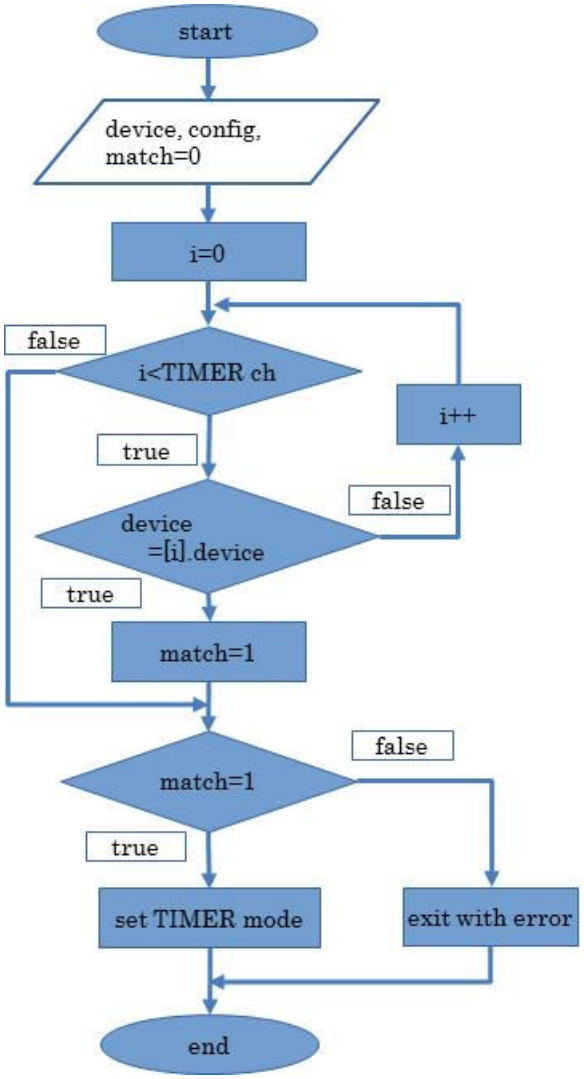


図 3.5.14 timer_config 関数フローチャート

3.5.2.5.4.3. **使用例**

TIMER0 をカウンタ値 0 になったらストップするモードに設定する例

プログラムには以下のようなコードを記述します。

```
{
        timer_config("/dev/timer_0", TIMER_MODE_ONCE);
}
```

3.5.2.6. WDT

3.5.2.6.1. wdt_start 関数

3.5.2.6.1.1. **関数仕様説明**

wdt_start 関数の仕様を表 2.5.15、フローチャートを図 2.5.15 に示します。

表 3.5.15 wdt_start 関数仕様

alt_u8 wdt_start(char *device)		
1	機能	
	WDT のカウンタを開始する関数	
2	入力パラメータ	
	device	デバイスファイル “/dev/wdt”
3	戻り値	
	0	カウンタ開始完了
	ENODEV	指定したデバイスが不一致

3.5.2.6.1.2. **フローチャート**

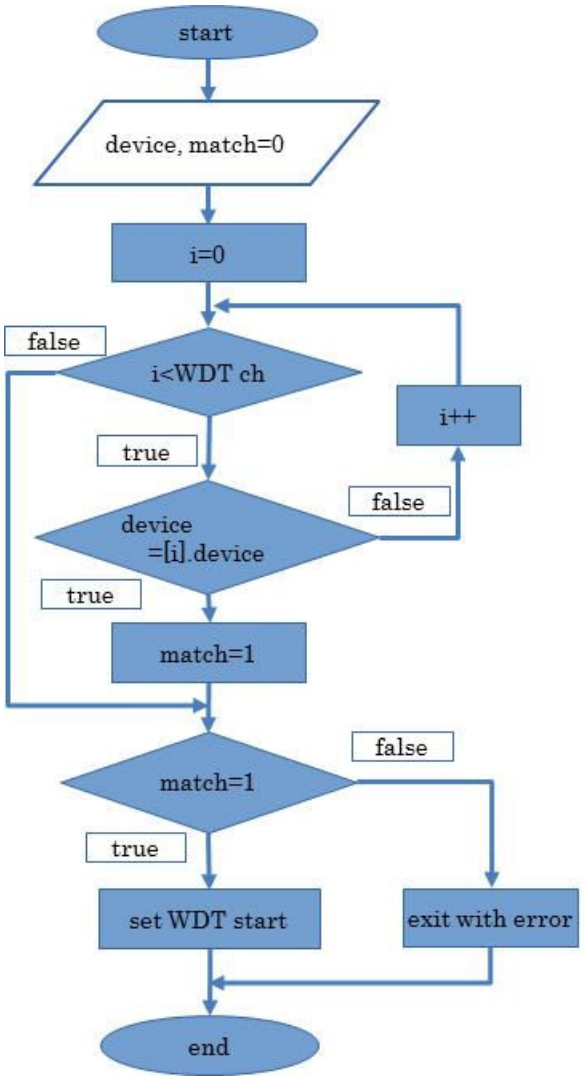


図 3.5.15 wdt_start 関数フローチャート

3.5.2.6.1.3. **使用例**

WDT のカウンタを開始する例

プログラムには以下のようなコードを記述します。

```
{
        wdt_start("/dev/wdt");
}
```

3.5.2.6.2. wdt_stop 関数

3.5.2.6.2.1. **関数仕様説明**

wdt_stop 関数の仕様を表 2.5.16、フローチャートを図 2.5.16 に示します。

表 3.5.16 wdt_stop 関数仕様

alt_u8 wdt_stop(char *device)		
1	機能	
	WDT のカウンタを停止する関数	
2	入力パラメータ	
	device	デバイスファイル “/dev/wdt”
3	戻り値	
	0	カウンタ停止完了
	ENODEV	指定したデバイスが不一致

3.5.2.6.2.2. **フローチャート**

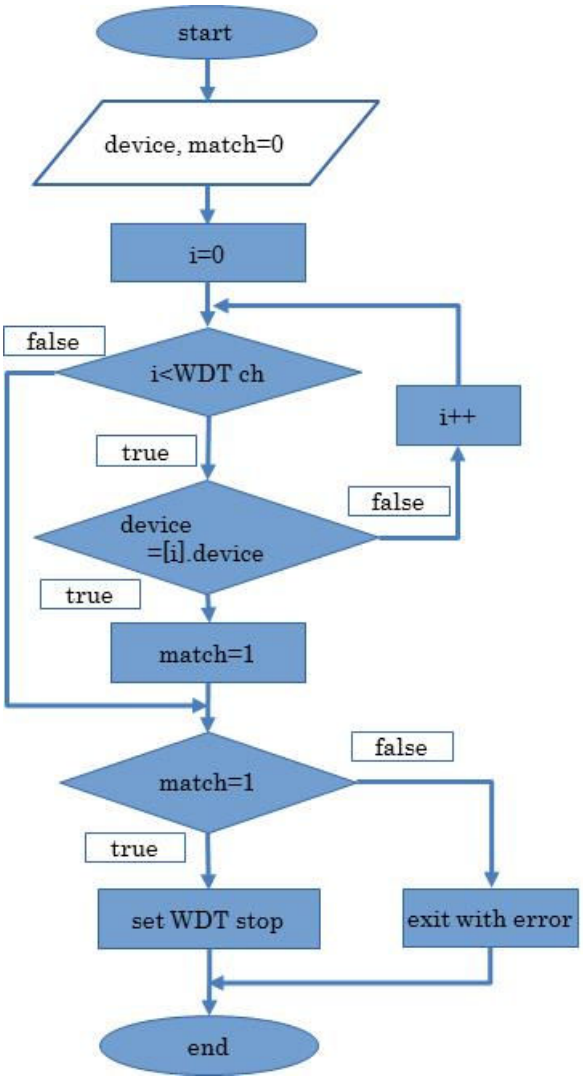


図 3.5.16 wdt_stop 関数フローチャート

3.5.2.6.2.3. **使用例**

WDT のカウンタを停止する例

プログラムには以下のようなコードを記述します。

```
{
        wdt_stop("/dev/wdt");
}
```

3.5.2.6.3. wdt_clr 関数

3.5.2.6.3.1. **関数仕様説明**

wdt_clr 関数の仕様を表 2.5.17、フローチャートを図 2.5.17 に示します。

表 3.5.17 wdt_clr 関数仕様

alt_u8 wdt_clr(char *device)		
1	機能	
	WDT のカウンタ値を初期化して再スタートする関数	
2	入力パラメータ	
	device	デバイスファイル “/dev/wdt”
3	戻り値	
	0	再スタート完了
	ENODEV	指定したデバイスが不一致

3.5.2.6.3.2. **フローチャート**

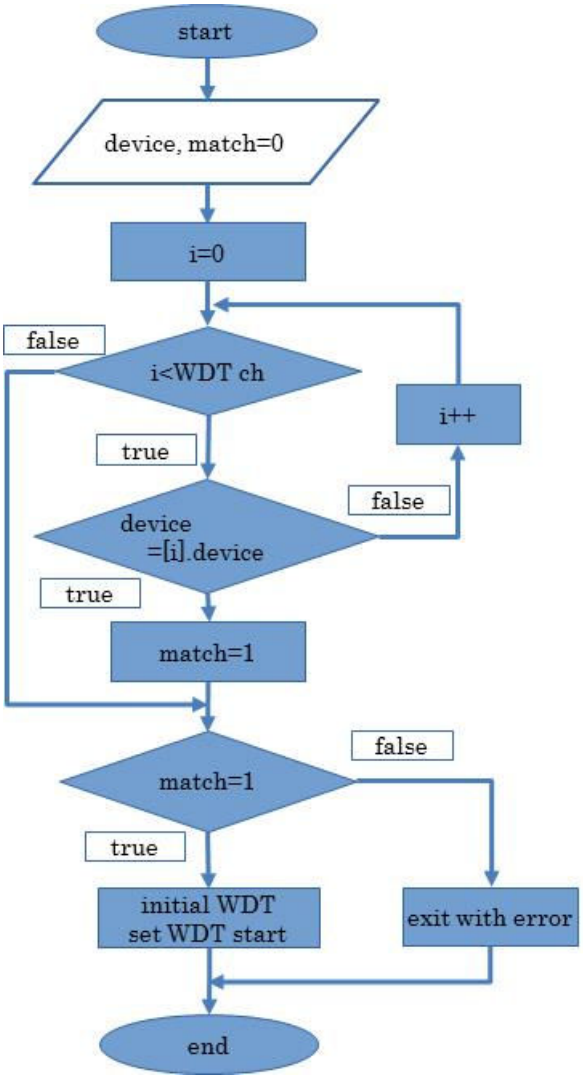


図 3.5.17 wdt_clr 関数フローチャート

3.5.2.6.3.3. **使用例**

WDT のカウンタを初期化して再スタートする例

プログラムには以下のようなコードを記述します。

```
{
        wdt_clr("/dev/wdt");
}
```

3.5.2.7. PIO

3.5.2.7.1. pio_direction 関数

3.5.2.7.1.1. **関数仕様説明**

pio_direction 関数の仕様を表 2.5.18、フローチャートを図 2.5.18 に示します。

表 3.5.18 pio_direction 関数仕様

void pio_direction(alt_u8 port_no, alt_u8 dir)		
1	機能	
	PIO の入力/出力を設定する関数	
2	入力パラメータ	
	port_no	ポート番号を指定 表 3 端子一覧の機能 1 の欄を指定
	dir	入力/出力を指定 PIO_DIR_IN:入力 PIO_DIR_OUT:出力
3	戻り値	
	なし	

3.5.2.7.1.2. **フローチャート**

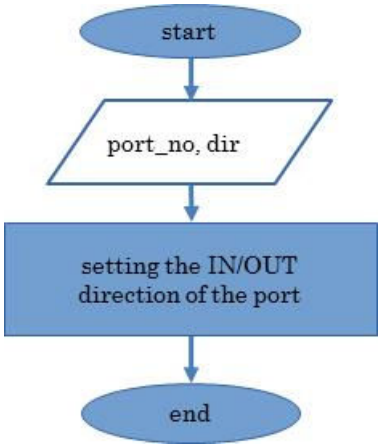


図 3.5.18 pio_direction 関数フローチャート

3.5.2.7.1.3. **使用例**

port PA0 を PIO 出力に設定する例

プログラムには以下のようなコードを記述します。

```
{
        pio_direction(PA0, PIO_DIR_OUT);
}
```

3.5.2.7.2. pio_input 関数

3.5.2.7.2.1. **関数仕様説明**

pin_input 関数の仕様を表 2.5.19、フローチャートを図 2.5.19 に示します。

表 3.5.19 pin_input 関数仕様

alt_u16 pio_input(alt_u8 port_no)		
1	機能	
	PIO のポート入力信号をモニタする関数	
2	入力パラメータ	
	port_no	ポート番号を指定 表 2.2.2 端子一覧の機能 1 の欄を指定
3	戻り値	
	入力ポートの状態	

3.5.2.7.2.2. **フローチャート**

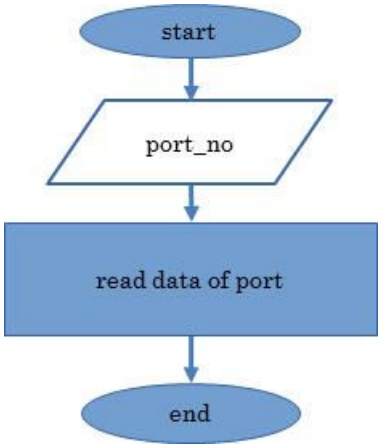


図 3.5.19 pin_input 関数フローチャート

3.5.2.7.2.3. **使用例**

port PA0 をモニタする例

プログラムには以下のようなコードを記述します。

```
{
        alt_u16  data;
        data = pio_input(PA0, PIO_DIR_OUT);
}
```

3.5.2.7.3. pio_output 関数

3.5.2.7.3.1. **関数仕様説明**

pio_output 関数の仕様を表 2.5.20、フローチャートを図 2.5.20 に示します。

表 3.5.20 pio_output 関数仕様

void pio_output(alt_u8 port_no, alt_u8 bit)		
1	機能	
	PIO の出力を制御する関数	
2	入力パラメータ	
	port_no	ポート番号を指定 表 2.2.2 端子一覧の機能 1 の欄を指定
	bit	信号を制御 0:Low 出力 1:High 出力
3	戻り値	
	なし	

3.5.2.7.3.2. **フローチャート**

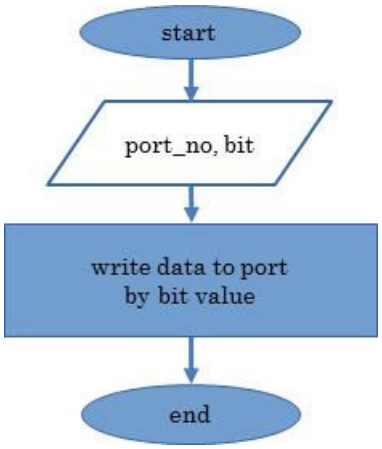


図 3.5.20 pio_output 関数フローチャート

3.5.2.7.3.3. **使用例**

port PA0 から 1 を出力する例

プログラムには以下のようなコードを記述します。

```
{
        pio_output(PA0, 1);
}
```

3.5.2.7.4. pio_request 関数

3.5.2.7.4.1. **関数仕様説明**

pio_request 関数の仕様を表 2.5.21、フローチャートを図 2.5.21 に示します。

表 3.5.21 pio_request 関数仕様

void pio_output(alt_u8 port_no, alt_u8 port)		
1	機能	
	端子の機能を切り替える関数	
2	入力パラメータ	
	port_no	ポート番号を指定 表 2.2.2 端子一覧の機能 1 の欄を指定
	port	機能切り替え PORT_PIO:PIO PORT_FUNC:Funtion
3	戻り値	
	なし	

3.5.2.7.4.2. **フローチャート**

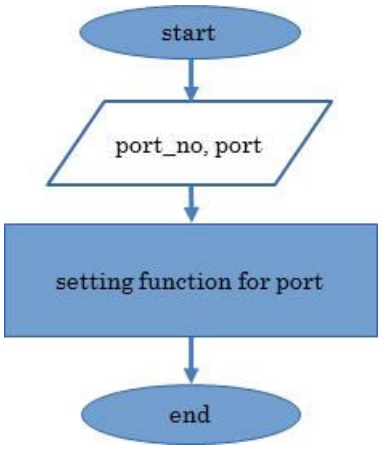


図 3.5.21 pio_request 関数フローチャート

3.5.2.7.4.3. **使用例**

port PA0 を UART0_TXD に設定する例

プログラムには以下のようなコードを記述します。

```
{
        pio_request(PA0, PORT_FUNC);
}
```

3.5.2.8. EINT

3.5.2.8.1. eint_enable 関数

3.5.2.8.1.1. **関数仕様説明**

eint_enable 関数の仕様を表 2.5.22、フローチャートを図 2.5.22 に示します。

表 3.5.22 eint_enable 関数仕様

void eint_enable(alt_u8 num)		
1	機能	
	EINT 割り込みを許可する関数	
2	入力パラメータ	
	num	EINT の番号を指定
3	戻り値	
	なし	

3.5.2.8.1.2. **フローチャート**

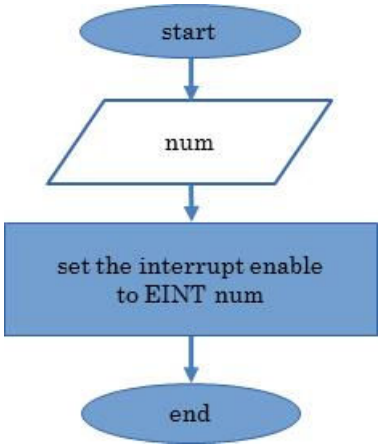


図 3.5.22 eint_enable 関数フローチャート

3.5.2.8.1.3. **使用例**

EINT0 の割り込みを許可する例

プログラムには以下のようなコードを記述します。

```
{
        eint_enable(EINT0);
}
```

3.5.2.8.2. eint_disable 関数

3.5.2.8.2.1. **関数仕様説明**

eint_disable 関数の仕様を表 2.5.23、フローチャートを図 2.5.23 に示します。

表 3.5.23 eint_disable 関数仕様

void eint_disable(alt_u8 num)		
1	機能	
	EINT 割り込みを禁止する関数	
2	入力パラメータ	
	num	EINT の番号を指定
3	戻り値	
	なし	

3.5.2.8.2.2. **フローチャート**

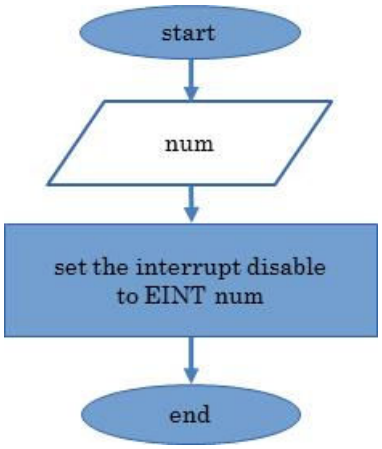


図 3.5.23 eint_disable 関数フローチャート

3.5.2.8.2.3. **使用例**

EINT0 の割り込みを禁止する例

プログラムには以下のようなコードを記述します。

```
{
        eint_disable(EINT0);
}
```

3.5.2.8.3. eint_status 関数

3.5.2.8.3.1. **関数仕様説明**

eint_status 関数の仕様を表 2.5.24、フローチャートを図 2.5.24 に示します。

表 3.5.24 eint_status 関数仕様

alt_u8 eint_status(alt_u8 num)		
1	機能	
	EINT 割り込みの状態をモニタする関数	
2	入力パラメータ	
	num	EINT の番号を指定
3	戻り値	
	割り込み状態 0:割り込み無 1:割り込み有	

3.5.2.8.3.2. **フローチャート**

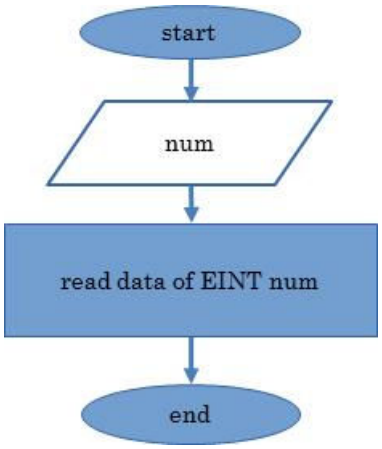


図 3.5.24 eint_status 関数フローチャート

3.5.2.8.3.3. **使用例**

EINT0 の割り込み状態をモニタする例

プログラムには以下のようなコードを記述します。

```
{
        alt_u8 data;
        data = eint_status(EINT0);
}
```

3.5.2.8.4. eint_clear 関数

3.5.2.8.4.1. **関数仕様説明**

eint_clear 関数の仕様を表 2.5.25、フローチャートを図 2.5.25 に示します。

表 3.5.25 eint_clear 関数仕様

void eint_clear(alt_u8 num)		
1	機能	
	EINT 割り込みをクリアする関数	
2	入力パラメータ	
	num	EINT の番号を指定
3	戻り値	
	なし	

3.5.2.8.4.2. **フローチャート**

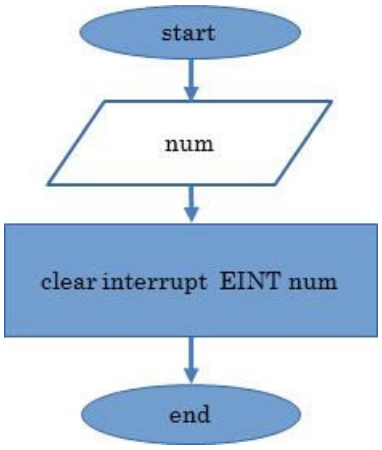


図 3.5.25 eint_clear 関数フローチャート

3.5.2.8.4.3. **使用例**

EINT0 の割り込みをクリアする例

プログラムには以下のようなコードを記述します。

```
{
        eint_clear(EINT0);
}
```

3.6. ドライバ

3.6.1. 概要

ペリフェラルモジュールの初期設定およびレジスタのリード/ライトを制御します。

3.6.2. ペリフェラルモジュール一覧

ペリフェラルモジュール一覧を表 2.6.1 に示します。

表 3.6.1 インタフェース一覧

NO	インタフェース	説明
1	UART	シリアルインタフェース
2	I2C	I2C 準拠インタフェース
3	SPI	シリアルペリフェラルインタフェース
4	PWM	パルス Duty 変動信号
5	TIMER	タイマー
6	WDT	ウォッチドッグタイマー
7	PIO	汎用入出力信号 端子機能切り替え
8	EINT	外部割り込み

3.6.3. 仕様

3.6.3.1. UART

3.6.3.1.1. UART_init 関数

3.6.3.1.1.1. **関数仕様説明**

UART_init 関数の仕様を表 2.6.2、フローチャートを図 2.6.1 に示します。

表 3.6.2 UART_init 関数仕様

void UART_init(struct UART_info *board_info)			
1	機能		
	UART の初期化をする関数		
2	入力パラメータ		
	board_info	address	UART チャンネルのベースアドレス
		baud	UART チャンネルのボーレート
		clk	UART チャンネルのシステムクロック
		irq	UART チャンネルの割り込み情報
		fifo	受信データを保存するキュー
3	戻り値		
	なし		

3.6.3.1.1.2. **フローチャート**

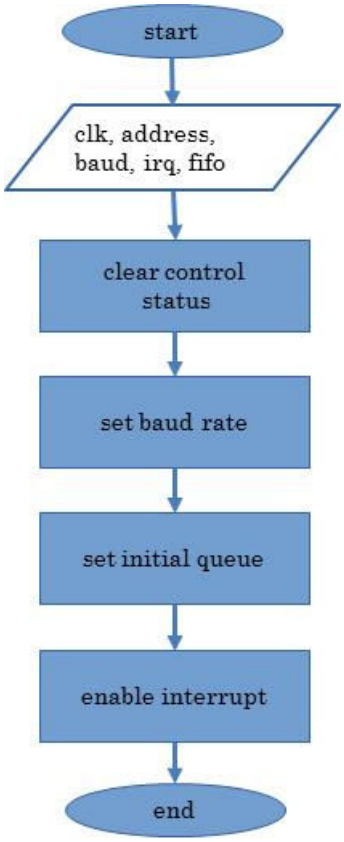


図 3.6.1 UART_init 関数フローチャート

3.6.3.1.2. UART_write 関数

3.6.3.1.2.1. **関数仕様説明**

UART_write 関数の仕様を表 2.6.3、フローチャートを図 2.6.2 に示します。

表 3.6.3 UART_write 関数仕様

void UART_write(struct UART_info *board_info, alt_u8 data)			
1	機能		
	UART の送信データを書き込む関数		
2	入力パラメータ		
	board_info	address	UART チャンネルのベースアドレス
	data	送信データ	
3	戻り値		
	なし		

3.6.3.1.2.2. **フローチャート**

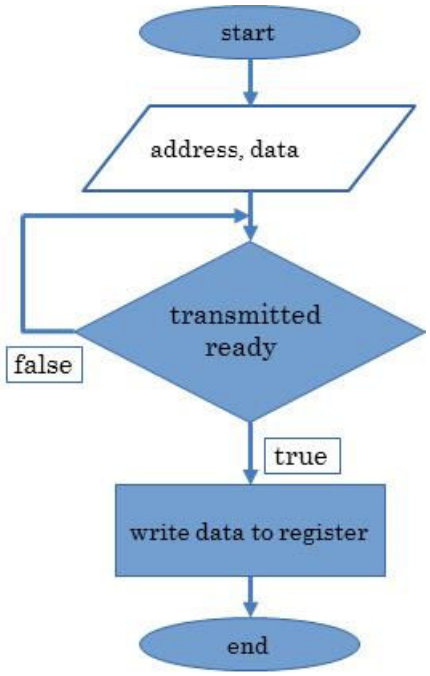


図 3.6.2 UART_wrire 関数フローチャート

3.6.3.1.3. UART_read 関数

3.6.3.1.3.1. **関数仕様説明**

UART_read 関数の仕様を表 2.6.4、フローチャートを図 2.6.3 に示します。

表 3.6.4 UART_read 関数仕様

alt_u8 UART_read(struct UART_info *board_info)			
1	機能		
	UART の受信データを読み出す関数		
2	入力パラメータ		
	board_info	address	UART チャンネルのベースアドレス
3	戻り値		
	なし		

3.6.3.1.3.2. **フローチャート**

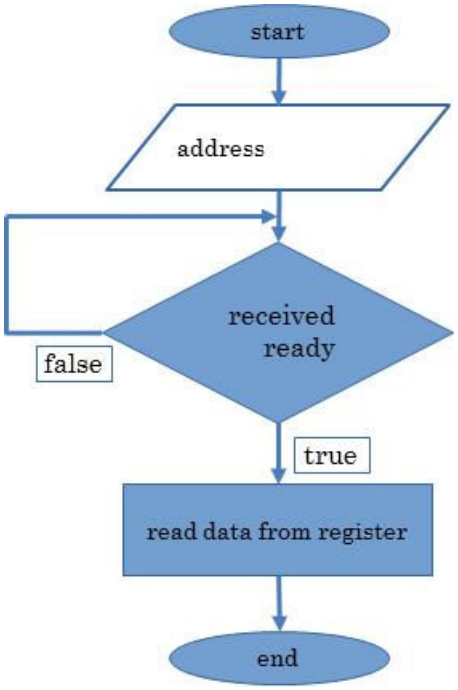


図 3.6.3 UART_read 関数フローチャート

3.6.3.1.4. handler_interrupt 関数

3.6.3.1.4.1. **関数仕様説明**

handler_interrupt 関数の仕様を表 2.6.5、フローチャートを図 2.6.4 に示します。

表 3.6.5 handler_interrupt 関数仕様

void handler_interrupt(struct UART_info *board_info)			
1	機能		
	受信割り込み発生時受信データ処理およびユーザー関数にジャンプする関数		
2	入力パラメータ		
	board_info	irq	割り込み情報
3	戻り値		
	なし		

3.6.3.1.4.2. **フローチャート**

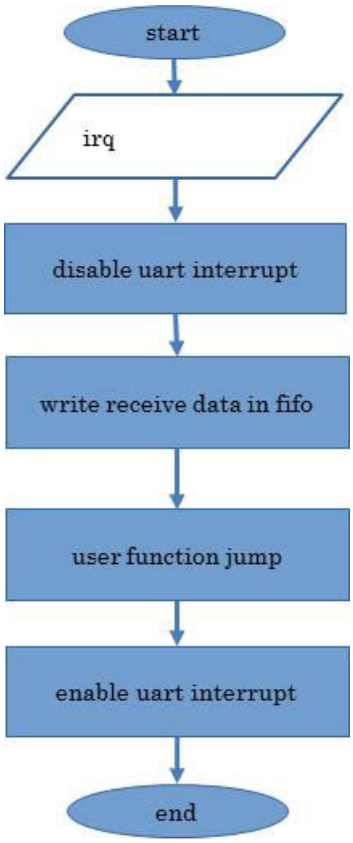


図 3.6.4 handler_interrup 関数フローチャート

3.6.3.1.5. uart_rx_fifo 関数

3.6.3.1.5.1. **関数仕様説明**

uart_rx_fifo 関数の仕様を表 2.6.6、フローチャートを図 2.6.5 に示します。

表 3.6.6 uart_rx_fifo 関数仕様

void uart_rx_fifo(struct UART_info *board_info)			
1	機能		
	UART の送信データを書き込む関数		
2	入力パラメータ		
	board_info	fifo	受信データを保存するキュー
3	戻り値		
	なし		

3.6.3.1.5.2. **フローチャート**

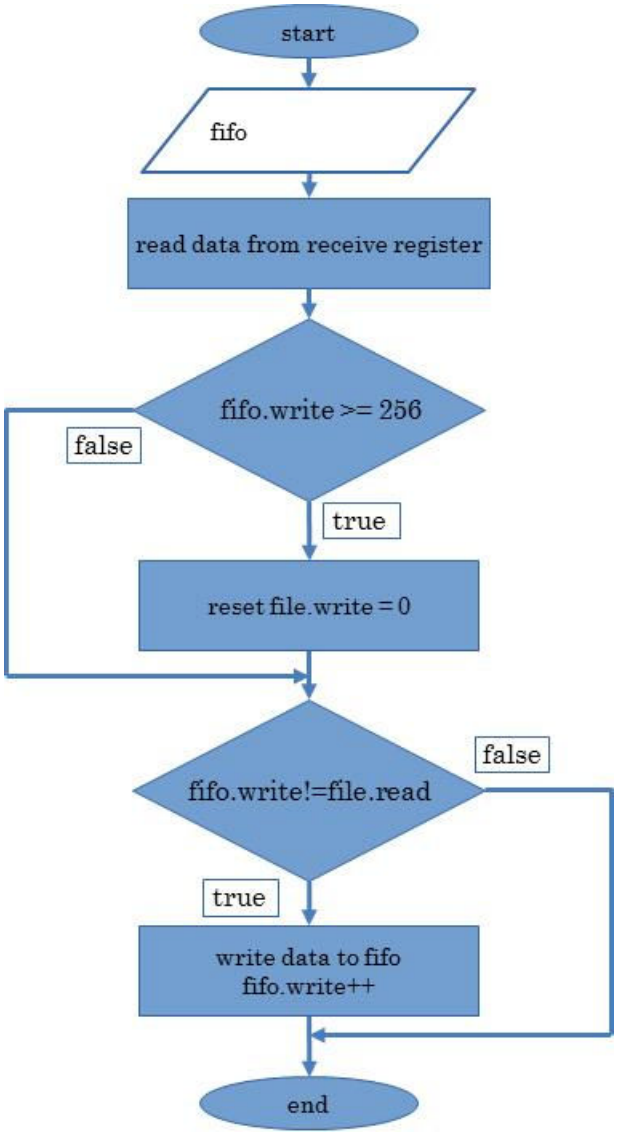


図 3.6.5 uart_rx_fifo 関数フローチャート

3.6.3.2. I2C

3.6.3.2.1. I2C_init 関数

3.6.3.2.1.1. **関数仕様説明**

I2C_init 関数の仕様を表 2.6.7、フローチャートを図 2.6.6 に示します。

表 3.6.7 I2C_init 関数仕様

void I2C_init(struct I2C_info *board_info)			
1	機能		
	I2C の初期化をする関数		
2	入力パラメータ		
	board_info	address	I2C チャンネルのベースアドレス
		clk	I2C の動作クロック
		speed	I2C の周波数
3	戻り値		
	なし		

3.6.3.2.1.2. **フローチャート**

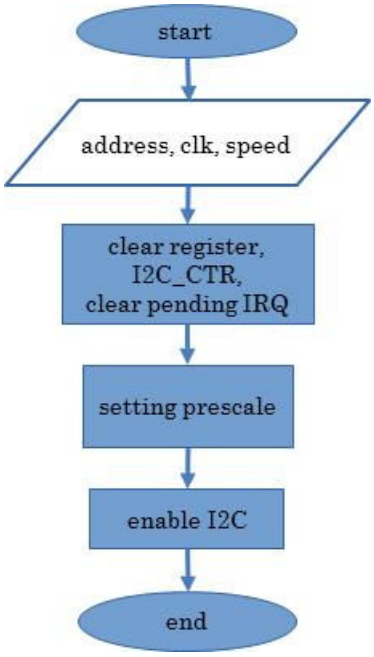


図 3.6.6 I2C_init 関数フローチャート

3.6.3.2.2. I2C_start 関数

3.6.3.2.2.1. **関数仕様説明**

I2C_start 関数の仕様を表 2.6.8、フローチャートを図 2.6.7 に示します。

表 3.6.8 i2c_start 関数仕様

alt_u8 I2C_start(struct I2C_info *board_info, char *device_name, alt_u8 dir)				
1	機能			
	I2C の開始ビットを設定しスレーブアドレスと方向ビットを送信する関数			
2	入力パラメータ			
	board_info	address	I2C チャンネルのベースアドレス	
		dev_info	device_name	ターゲットのデバイス名
	dir	送信/受信を指定 0:受信 1:送信		
3	戻り値			
	0:ACK 1:NACK			

3.6.3.2.2.2. **フローチャート**

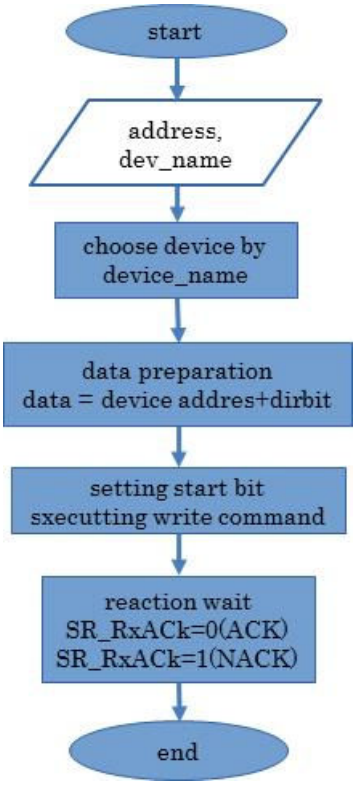


図 3.6.7 I2C_start 関数フローチャート

3.6.3.2.3. I2C_read 関数

3.6.3.2.3.1. **関数仕様説明**

I2C_read 関数の仕様を表 2.6.9、フローチャートを図 2.6.8 に示します。

表 3.6.9 I2C_read 関数仕様

alt_u8 I2C_read(struct I2C_info *board_info, alt_u8 last)			
1	機能		
	I2C の受信転送をする関数		
2	入力パラメータ		
	board_info	address	I2C チャンネルのベースアドレス
	last	最後のデータ設定 0:Next データ有 1:最後データ	
3	戻り値		
	1byte の受信データ		

3.6.3.2.3.2. **フローチャート**

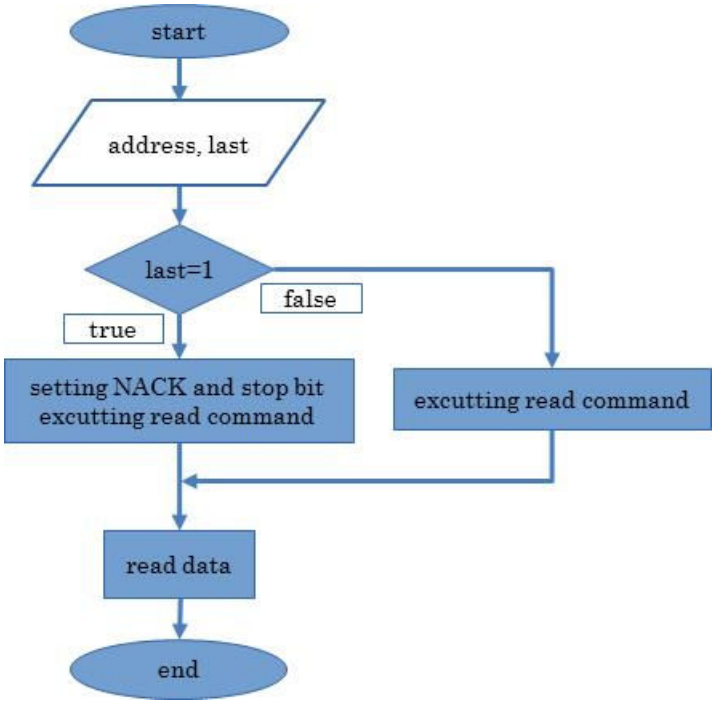


図 3.6.8 I2C_read 関数フローチャート

3.6.3.2.4. I2C_write 関数

3.6.3.2.4.1. **関数仕様説明**

I2C_write 関数の仕様を表 2.6.10、フローチャートを図 2.6.9 に示します。

表 3.6.10 I2C_write 関数仕様

alt_u8 I2C_write(struct I2C_info *board_info, alt_u8 data, alt_u8 last)			
1	機能		
	I2C の受信転送をする関数		
2	入力パラメータ		
	board_info	address	I2C チャンネルのベースアドレス
	data	送信データ	
	last	最後のデータ設定 0:Next データ有 1:最後データ	
3	戻り値		
	0:ACK 1:NACK		

3.6.3.2.4.2. **フローチャート**

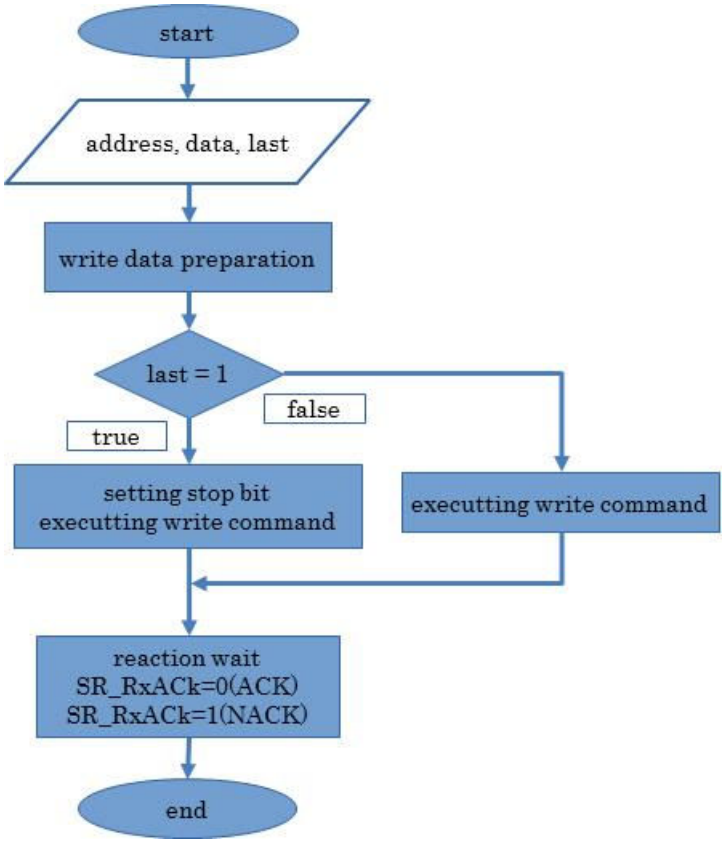


図 3.6.9 I2C_write 関数フローチャート

3.6.3.3. SPI

3.6.3.3.1. SPI_init 関数

3.6.3.3.1.1. **関数仕様説明**

SPI_init 関数の仕様を表 2.6.11、フローチャートを図 2.6.10 に示します。

表 3.6.11 SPI_init 関数仕様

void SPI_init(struct SPI_info *board_info)			
1	機能		
	SPI の初期化をする関数		
2	入力パラメータ		
	board_info	address	SPI チャンネルのベースアドレス
		clk	SPI の動作クロック
		speed	SPI の周波数
		mode	SPI の転送モード
3	戻り値		
	なし		

3.6.3.3.1.2. **フローチャート**

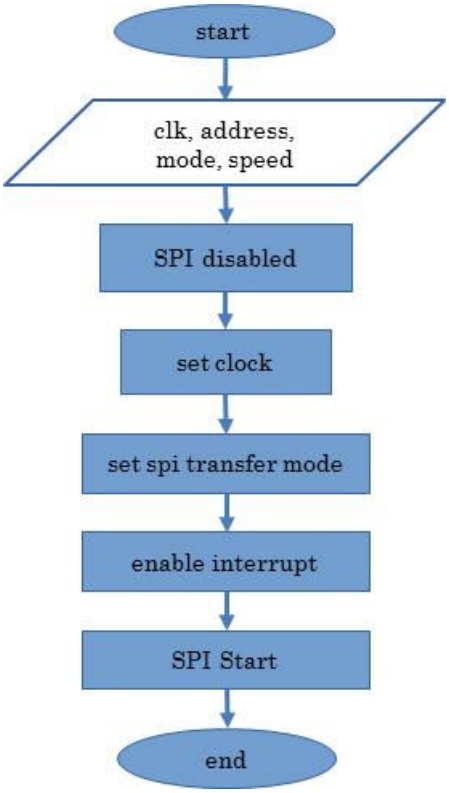


図 3.6.10 SPI_init 関数フローチャート

3.6.3.3.2. SPI_write 関数

3.6.3.3.2.1. **関数仕様説明**

SPI_write 関数の仕様を表 2.6.12、フローチャートを図 2.6.11 に示します。

表 3.6.12 SPI_write 関数仕様

alt_u8 SPI_write(struct SPI_info *board_info, const alt_u8 *write_data, alt_u16 write_length)			
1	機能		
	SPI の送信データのバイト数を指定して書き込む関数		
2	入力パラメータ		
	board_info	address	SPI チャンネルのベースアドレス
	write_data	送信データポインタ	
	write_length	送信データバイト数	
3	戻り値		
	0:転送完了		

3.6.3.3.2.2. **フローチャート**

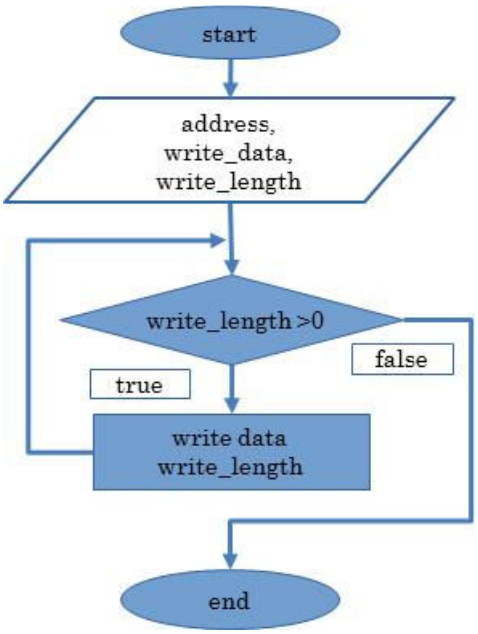


図 3.6.11 SPI_write 関数フローチャート

3.6.3.3.3. SPI_read 関数

3.6.3.3.3.1. **関数仕様説明**

SPI_read 関数の仕様を表 2.6.13、フローチャートを図 2.6.12 に示します。

表 3.6.13 SPI_read 関数仕様

alt_u8 SPI_read(struct SPI_info *board_info, alt_u8 *read_data, alt_u16 read_length)			
1	機能		
	SPI の受信データのバイト数を指定して読み出す関数		
2	入力パラメータ		
	board_info	address	SPI チャンネルのベースアドレス
	read_data	受信データポインタ	
	read_length	受信データバイト数	
3	戻り値		
	0:転送完了		

3.6.3.3.3.2. **フローチャート**

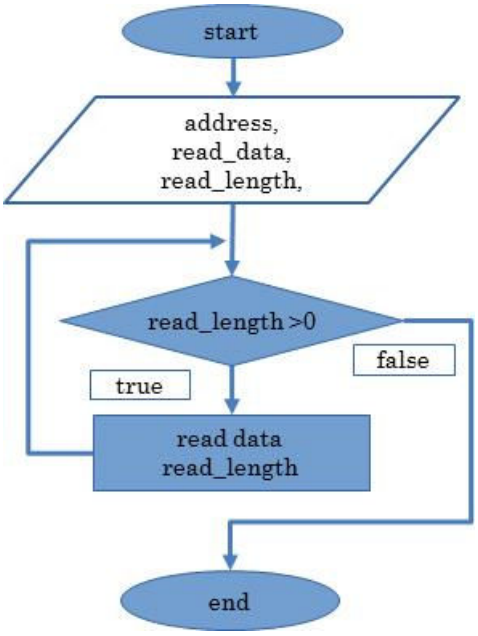


図 3.6.12 SPI_read 関数フローチャート

3.6.3.3.4. spi_ready_wait 関数

3.6.3.3.4.1. **関数仕様説明**

spi_ready_wait 関数の仕様を表 2.6.14、フローチャートを図 2.6.13 に示します。

表 3.6.14 spi_ready_wait 関数仕様

void spi_ready_wait(alt_u32 addr)		
1	機能	
	SPI のデータ転送可能状態までウェイトする関数	
2	入力パラメータ	
	addr	SPI チャンネルのベースアドレス
3	戻り値	
	なし	

3.6.3.3.4.2. **フローチャート**

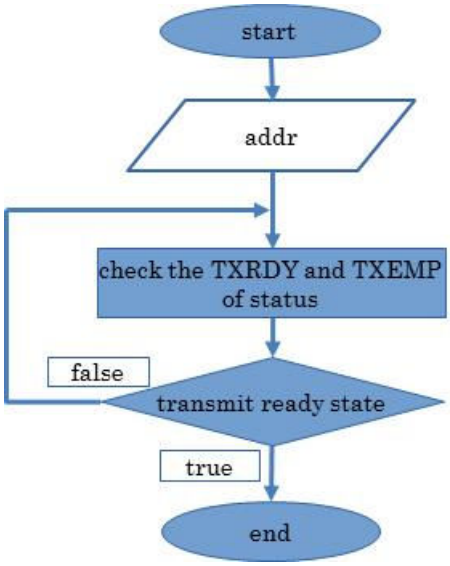


図 3.6.13 spi_ready_wait 関数フローチャート

3.6.3.3.5. SPI_clear_cs 関数

3.6.3.3.5.1. **関数仕様説明**

SPI_clear_cs 関数の仕様を表 2.6.15、フローチャートを図 2.6.14 に示します。

表 3.6.15 SPI_clear_cs 関数仕様

void SPI_clear_cs(struct SPI_info *board_info)				
1	機能			
	SPI の CS 信号を High に設定する関数			
2	入力パラメータ			
	board_info	dev_info	pio_cs	CS の PIO 番号
3	戻り値			
	なし			

3.6.3.3.5.2. **フローチャート**

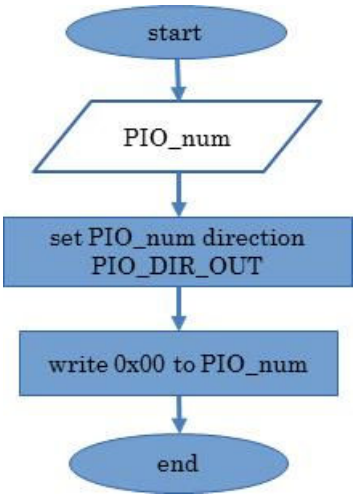


図 3.6.14 SPI_clear_cs 関数フローチャート

3.6.3.3.6. SPI_set_cs 関数

3.6.3.3.6.1. **関数仕様説明**

SPI_set_cs 関数の仕様を表 2.6.16、フローチャートを図 2.6.15 に示します。

表 3.6.16 SPI_set_cs 関数仕様

void SPI_set_cs(struct SPI_info *board_info)				
1	機能			
	SPI の CS 信号を Low に設定する関数			
2	入力パラメータ			
	board_info	dev_info	pio_cs	CS の PIO 番号
3	戻り値			
	なし			

3.6.3.3.6.2. **フローチャート**

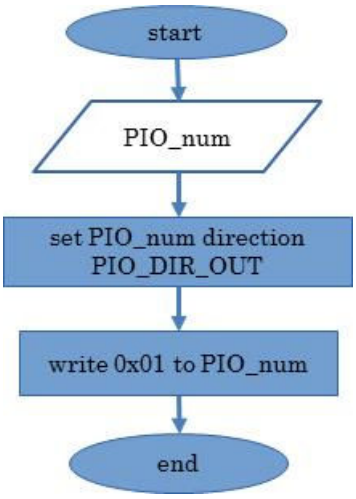


図 3.6.15 SPI_set_cs 関数フローチャート

3.6.3.4. PWM

3.6.3.4.1. PWM_init 関数

3.6.3.4.1.1. **関数仕様説明**

PWM_init 関数の仕様を表 2.6.17、フローチャートを図 2.6.16 に示します。

表 3.6.17 PWM_init 関数仕様

void PWM_init(struct PWM_info *board_info)			
1	機能		
	PWM を初期化する関数		
2	入力パラメータ		
	board_info	address	PWM チャンネルのベースアドレス
		clk	PWM の動作 CLK
		speed	PWM のパルス周波数
3	戻り値		
	なし		

3.6.3.4.1.2. **フローチャート**

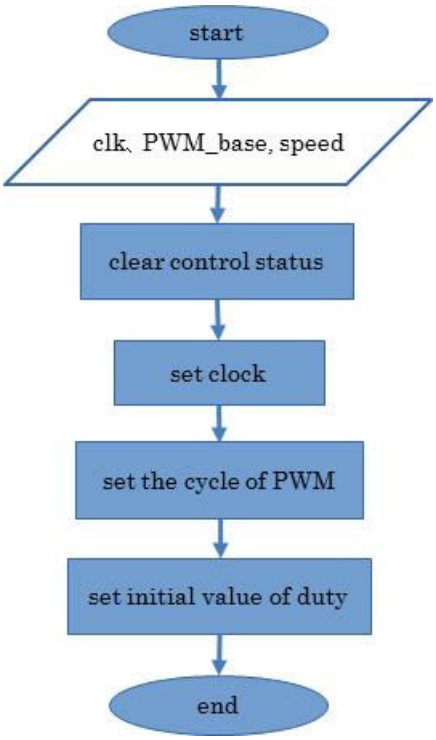


図 3.6.16 PWM_init 関数フローチャート

3.6.3.4.2. PWM_write 関数

3.6.3.4.2.1. **関数仕様説明**

PWM_write 関数の仕様を表 2.6.18、フローチャートを図 2.6.17 に示します。

表 3.6.18 PWM_write 関数仕様

void PWM_write(struct UART_info *board_info)			
1	機能		
	PWM のレジスタにデータを書き込む関数		
2	入力パラメータ		
	board_info	address	PWM チャンネルのベースアドレス
	offset	PWM のレジスタのオフセット	
	data	書き込みデータ	
3	戻り値		
	なし		

3.6.3.4.2.2. **フローチャート**

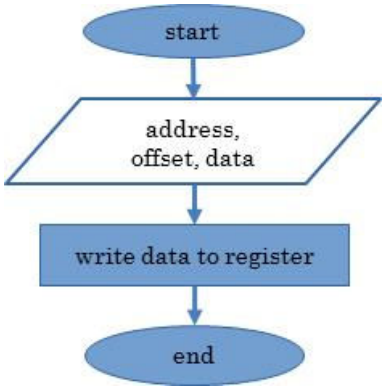


図 3.6.17 PWM_write 関数フローチャート

3.6.3.4.3. PWM_read 関数

3.6.3.4.3.1. **関数仕様説明**

PWM_read 関数の仕様を表 2.6.19、フローチャートを図 2.6.18 に示します。

表 3.6.19 PWM_read 関数仕様

alt_u16 PWM_read(struct PWM_info *board_info, alt_u8 offset)			
1	機能		
	PWM のレジスタからデータを読み出す関数		
2	入力パラメータ		
	board_info	address	PWM チャンネルのベースアドレス
	offset	PWM レジスタのオフセット	
3	戻り値		
	読み出したデータ		

3.6.3.4.3.2. **フローチャート**

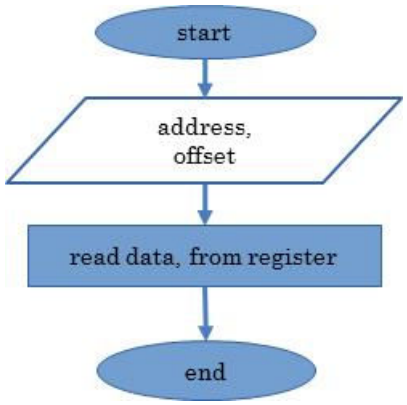


図 3.6.18 PWM_read 関数フローチャート

3.6.3.5. TIMER

3.6.3.5.1. TIMER_init 関数

3.6.3.5.1.1. **関数仕様説明**

TIMER_init 関数の仕様を表 2.6.20、フローチャートを図 2.6.19 に示します。

表 3.6.20 TIMER_init 関数仕様

void TIMER_init(struct TIMER_info *board_info)			
1	機能		
	TIMER を初期化する関数		
2	入力パラメータ		
	board_info	address	TIMER チャンネルのベースアドレス
		period	カウンタ値の設定
3	戻り値		
	なし		

3.6.3.5.1.2. **フローチャート**

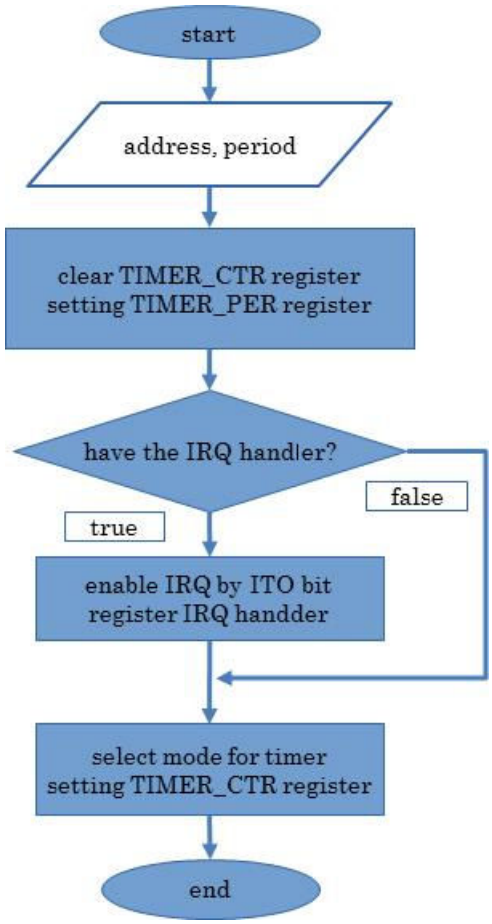


図 3.6.19 TIMER_init 関数フローチャート

3.6.3.5.2. TIMER_write 関数

3.6.3.5.2.1. **関数仕様説明**

TIMER_write 関数の仕様を表 2.6.21、フローチャートを図 2.6.20 に示します。

表 3.6.21 TIMER_write 関数仕様

void TIMER_write(struct TIMER_info *board_info, alt_u8 add, alt_u16)			
1	機能		
	TIMER のレジスタにデータを書き込む関数		
2	入力パラメータ		
	board_info	address	TIMER チャンネルのベースアドレス
	add	TIMER レジスタのオフセット	
	data	書き込みデータ	
3	戻り値		
	なし		

3.6.3.5.2.2. **フローチャート**

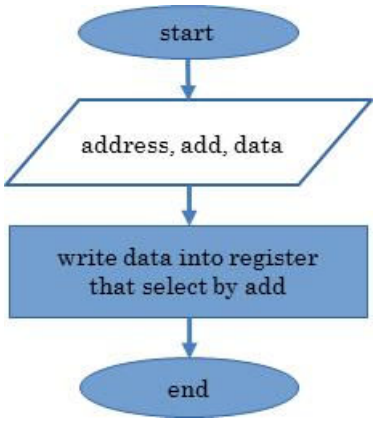


図 3.6.20 TIMER_write 関数フローチャート

3.6.3.5.3. TIMER_read 関数

3.6.3.5.3.1. **関数仕様説明**

TIMER_read 関数の仕様を表 2.6.22、フローチャートを図 2.6.21 に示します。

表 3.6.22 TIMER_read 関数仕様

alt_u16 TIMER_read(struct TIMER_info *board_info, alt_u8 add)			
1	機能		
	TIMER のレジスタのデータを読み出す関数		
2	入力パラメータ		
	board_info	address	TIMER チャンネルのベースアドレス
	add	TIMER レジスタのオフセット	
3	戻り値		
	読み出したデータ		

3.6.3.5.3.2. **フローチャート**

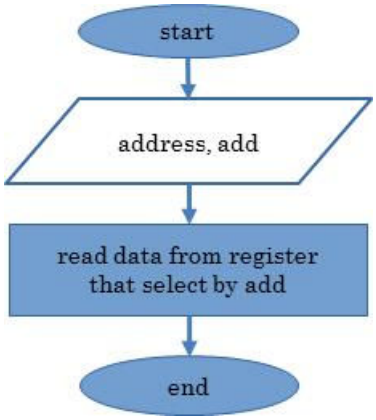


図 3.6.21 TIMER_read 関数フローチャート

3.6.3.6. WDT

3.6.3.6.1. WDT_init 関数

3.6.3.6.1.1. **関数仕様説明**

WDT_init 関数の仕様を表 2.6.23、フローチャートを図 2.6.22 に示します。

表 3.6.23 WDT_init 関数仕様

void WDT_init(struct WDT_info *board_info)			
1	機能		
	WDT を初期化する関数		
2	入力パラメータ		
	board_info	address	WDT のベースアドレス
		period	カウンタ値の設定
3	戻り値		
	なし		

3.6.3.6.1.2. **フローチャート**

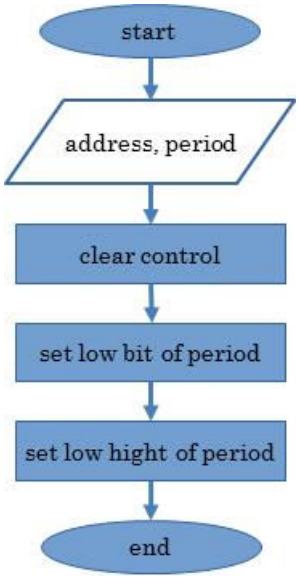


図 3.6.22 WDT_init 関数フローチャート

3.6.3.6.2. WDT_write 関数

3.6.3.6.2.1. **関数仕様説明**

WDT_write 関数の仕様を表 2.6.24、フローチャートを図 2.6.23 に示します。

表 3.6.24 WDT_write 関数仕様

void WDT_write(struct WDT_info *board_info, alt_u8 add, alt_u16 data)			
1	機能		
	WDT のレジスタにデータを書き込む関数		
2	入力パラメータ		
	board_info	address	WDT のベースアドレス
	add	WDT のレジスタのオフセット	
	data	書き込みデータ	
3	戻り値		
	なし		

3.6.3.6.2.2. **フローチャート**

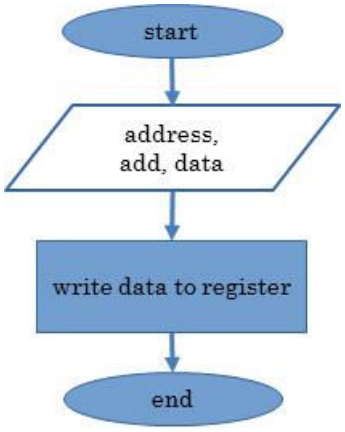


図 3.6.23 WDT_write 関数フローチャート

3.6.3.6.3. WDT_read 関数

3.6.3.6.3.1. **関数仕様説明**

WDT_read 関数の仕様を表 2.6.25、フローチャートを図 2.6.24 に示します。

表 3.6.25 WDT_read 関数仕様

alt_u16 WDT_read(struct WDT_info *board_info, alt_u8 add)			
1	機能		
	WDT のレジスタからデータを読み出す関数		
2	入力パラメータ		
	board_info	address	WDT のベースアドレス
	add	WDT のレジスタのオフセット	
3	戻り値		
	読み出したデータ		

3.6.3.6.3.2. **フローチャート**

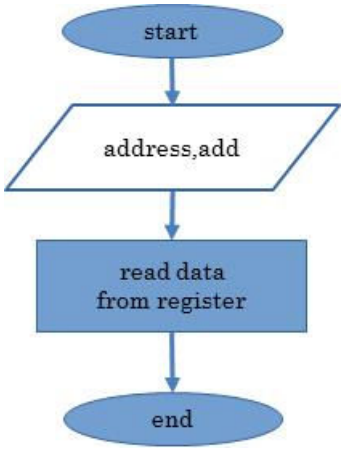


図 3.6.24 WDT_read 関数フローチャート

3.6.3.7. PIO

3.6.3.7.1. PIO_data_write 関数

3.6.3.7.1.1. **関数仕様説明**

PIO_data_write 関数の仕様を表 2.6.26、フローチャートを図 2.6.25 に示します。

表 3.6.26 PIO_data_write 関数仕様

void PIO_data_write(alt_u8 port_no, alt_u16 data)		
1	機能	
	PIO の出力レジスタにデータを書き込む関数	
2	入力パラメータ	
	port_no	PIO 番号
	data	出力するデータ値
3	戻り値	
	なし	

3.6.3.7.1.2. **フローチャート**

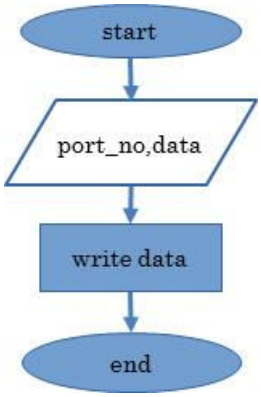


図 3.6.25 PIO_data_write 関数フローチャート

3.6.3.7.2. PIO_dir_write 関数

3.6.3.7.2.1. **関数仕様説明**

PIO_dir_write 関数の仕様を表 2.6.27、フローチャートを図 2.6.26 に示します。

表 3.6.27 PIO_dir_write 関数仕様

void PIO_dir_write(alt_u8 port_no, alt_u16 data)		
1	機能	
	PIO の方向制御のレジスタにデータを書き込む関数	
2	入力パラメータ	
	port_no	PIO 番号
	data	入力/出力を指定 0:入力 1:出力
3	戻り値	
	なし	

3.6.3.7.2.2. **フローチャート**

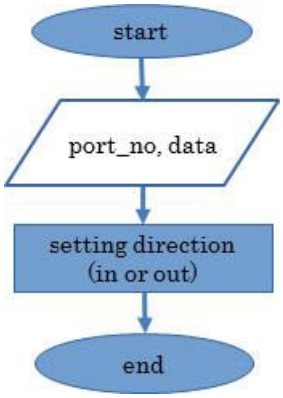


図 3.6.26 PIO_dir_write 関数フローチャート

3.6.3.7.3. PIO_data_read 関数

3.6.3.7.3.1. **関数仕様説明**

PIO_data_read 関数の仕様を表 2.6.28、フローチャートを図 2.6.27 に示します。

表 3.6.28 PIO_data_read 関数仕様

alt_u16 PIO_data_read(alt_u8 port_no)		
1	機能	
	PIO の入力をモニタする関数	
2	入力パラメータ	
	port_no	PIO 番号
3	戻り値	
	モニタしたデータ	

3.6.3.7.3.2. **フローチャート**

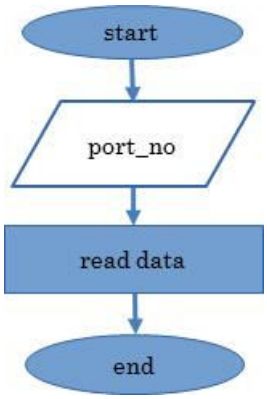


図 3.6.27 PIO_data_read 関数フローチャート

3.6.3.7.4. PIO_dir_read 関数

3.6.3.7.4.1. **関数仕様説明**

PIO_dir_read 関数の仕様を表 2.6.29、フローチャートを図 2.6.28 に示します。

表 3.6.29 PIO_dir_read 関数仕様

alt_u16 PIO_dir_read(alt_u8 port_no)		
1	機能	
	PIO の方向制御のレジスタからデータを読み出す関数	
2	入力パラメータ	
	port_no	PIO 番号
3	戻り値	
	読み出したデータ	

3.6.3.7.4.2. **フローチャート**

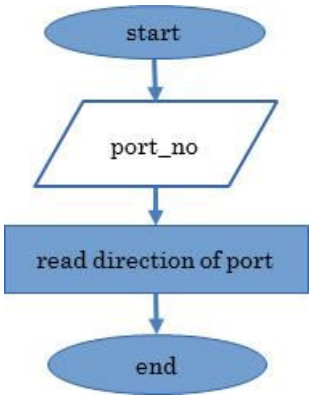


図 3.6.28 PIO_dir_read 関数フローチャート

3.6.3.7.5. PIO_mux_write 関数

3.6.3.7.5.1. **関数仕様説明**

PIO_mux_write 関数の仕様を表 2.6.30、フローチャートを図 2.6.29 に示します。

表 3.6.30 PIO_mux_write 関数仕様

void PIO_mux_write(alt_u8 port_no, alt_u16 data)		
1	機能	
	PIO の機能選択のレジスタにデータを書き込む関数	
2	入力パラメータ	
	port_no	PIO 番号
	data	機能選択を指定 0:PIO 1:FUNC
3	戻り値	
	なし	

3.6.3.7.5.2. **フローチャート**

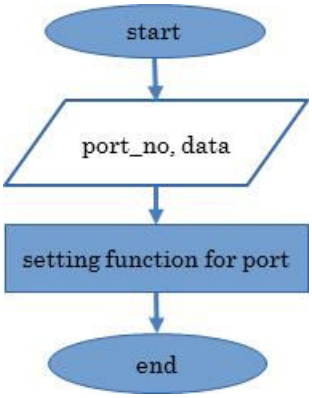


図 3.6.29 PIO_mux_write 関数フローチャート

3.6.3.7.6. PIO_mux_read 関数

3.6.3.7.6.1. **関数仕様説明**

PIO_mux_read 関数の仕様を表 2.6.31、フローチャートを図 2.6.30 に示します。

表 3.6.31 PIO_mux_read 関数仕様

alt_u16 PIO_mux_read(alt_u8 port_no)		
1	機能	
	PIO の方向制御のレジスタからデータを読み出す関数	
2	入力パラメータ	
	port_no	PIO 番号
3	戻り値	
	読み出したデータ	

3.6.3.7.6.2. **フローチャート**

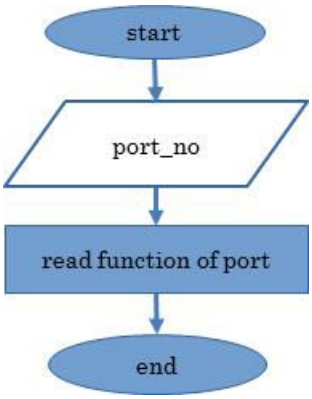


図 3.6.30 PIO_mux_read 関数フローチャート

3.6.3.8. EINT

3.6.3.8.1. EINT_init 関数

3.6.3.8.1.1. **関数仕様説明**

EINT_init 関数の仕様を表 2.6.32、フローチャートを図 2.6.31 に示します。

表 3.6.32 EINT_init 関数仕様

void EINT_init(struct EINT_info *board_info, alt_u8 num)				
1	機能			
	EINT を初期化する関数			
2	入力パラメータ			
	board_info	address	EINT のベースアドレス	
		dev_info	num	EINT 番号
			int_mode	割り込みエッジ/レベル指定
			int_pol	割り込み High/Low 指定
			irq	EINT の割り込み情報
	num	EINT の番号		
3	戻り値			
	なし			

3.6.3.8.1.2. **フローチャート**

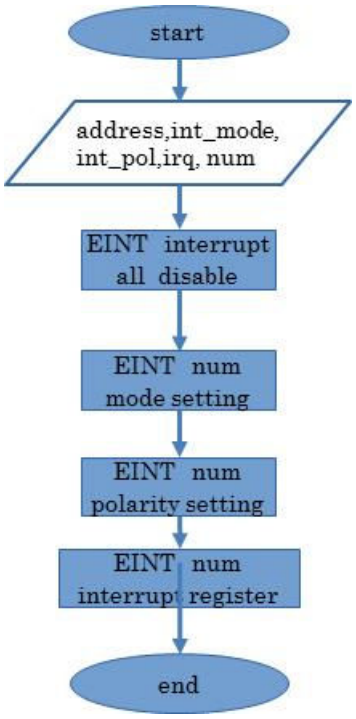


図 3.6.31 EINT_init 関数フローチャート

3.6.3.8.2. EINT_write 関数

3.6.3.8.2.1. **関数仕様説明**

EINT_write 関数の仕様を表 2.6.33、フローチャートを図 2.6.32 に示します。

表 3.6.33 EINT_write 関数仕様

void EINT_write (struct EINT_info *board_info, alt_u8 addr, alt_u16 data)			
1	機能		
	EINT のレジスタにデータを書き込む関数		
2	入力パラメータ		
	board_info	address	EINT のベースアドレス
	addr	EINT のレジスタのオフセット	
	data	書き込みデータ	
3	戻り値		
	なし		

3.6.3.8.2.2. **フローチャート**

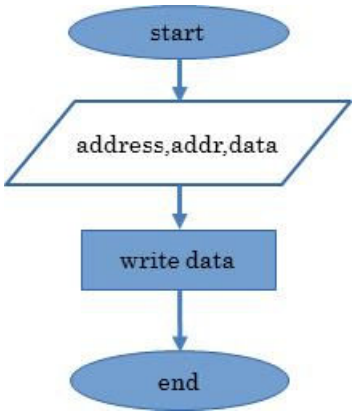


図 3.6.32 EINT_write 関数フローチャート

3.6.3.8.3. EINT_read 関数

3.6.3.8.3.1. **関数仕様説明**

EINT_read 関数の仕様を表 2.6.34、フローチャートを図 2.6.33 に示します。

表 3.6.34 EINT_read 関数仕様

alt_u16 EINT_read(struct EINT_info *board_info, alt_u8 addr)			
1	機能		
	EINT のレジスタのデータを読み出す関数		
2	入力パラメータ		
	board_info	address	EINT のベースアドレス
	addr	EINT のレジスタのオフセット	
3	戻り値		
	読み出したデータ		

3.6.3.8.3.2. **フローチャート**

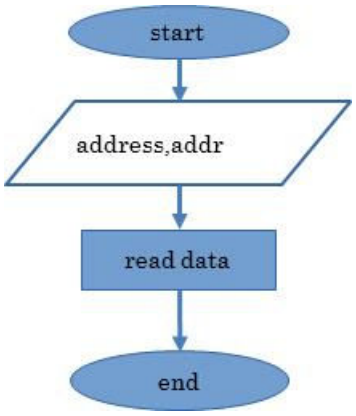


図 3.6.33 EINT_read 関数フローチャート

4. 更新履歴

Ver.	更新日付	内容
1.0.0	2015/09/14	新規作成
1.1.0	2016/04/14	・1.1.1 開発環境表を修正 ・各種データ入手元の記載を”CD-ROM”から”弊社 Web サイトよりダウンロード”に変更