μT-Kernel3.0 BSP ユーザズ・マニュアル

- Raspberry Pi Pico編 -

Version.01.00.01

2023.05.24

目次

1. 概要

本書は、マイコンボードRaspberry Pi Pico向けのμT-Kernel 3.0 BSP (Board Support Package)の使用方法を説明します。 μT-Kernel 3.0 BSPは、特定のマイコンボード等のハードウェアに対して移植したμT-Kernel 3.0の開発および実行環境一式を提供するものです。

1.1. 対象ハードウェアおよびソフトウェア

本BSPの対象ハードウェアおよびソフトウェアは以下となります。

分類	名称	開発元
マイコンボード	Raspberry Pi Pico	Raspberry Pi Foundation
搭載マイコン	RP2040 (ARM Cortex-M0+コア)	Raspberry Pi Foundation
OS	μT-Kernel3.0 v3.00.06A	TRONフォーラム ※1
BSP	Raspberry Pico向けμT-Kernel3.0 BSP v1.00.00.B5	TRONフォーラム ※2

※1 μT-Kernel 3.0は以下のGitHubのリポジトリから公開 https://github.com/tron-forum/mtkernel_3

※2 μT-Kernel3.0 BSPは以下のGitHubのリポジトリから公開 https://github.com/tron-forum/mtk3_bsp

1.2. 使用する開発環境

開発を行うホストPCのOSはWindowsとします。確認はWindows 11で行いました。 本BSPで使用する開発環境、ツールは以下となります。

(1) C言語ツールチェーン The xPack GNU Arm Embedded GCC Armマイコン向けのC言語のコンパイラ等ツール一式です。Arm 社が公式に提供するGNU Arm Embedded Toolchain から派生したもので、後述のEclipse IDEでの使用に適します。また、GBD(デバッガ)などのるも含みます。
動作検証を行ったバージョンは xpack-arm-none-eabi-gcc-12.2.1-1.1 です。

(2) 開発ツール xPack Windows Build Tools Windows用のビルドツールGnu Makeです。 動作検証を行ったバージョンは xpack-windows-build-tools-4.4.0-1 です。

(3) 統合開発環境 Eclipse IDE for Embedded C/C++ Developers 組込みシステム開発用の統合開発環境Eclipseのパッケージです。Eclipse Embedded CDT プラグインが組み込まれており，xPack のツールと連携することができます。本書では以降はEclipse Embedded IDEと称します。 動作検証を行ったバージョンは eclipse_202303 です。

1.3. 関連ドキュメント

以下に関連ドキュメントを示します。

分類	名称	発行

分類	名称	発行
OS仕様	μT-Kernel 3.0仕様書 Ver.3.00.01	TRONフォーラム TEF020-S004-3.00.01 ※1
デバイスドライバ	μT-Kernel 3.0 デバイスドライバ説明書 Ver.1.00.5	TRONフォーラム ※2 TEF033-W007-221007

※1 TRONフォーラムWebページから公開
https://tron-forum.github.io/

※2 μT-Kernel 3.0正式版に含まれる（以下のGitHubのリポジトリから公開）
https://github.com/tron-forum/mtkernel_3

2. 開発環境の準備

µT-Kernel 3.0 BSPを使用するにあたり、開発環境を準備する手順を説明します。

2.1. C言語ツールチェーンのインストール

Cコンパイラなどの開発ツールをインストールします。これらのツールはEclipse IDEから使用されます。

(1) C言語ツールチェーンのインストール C言語ツールチェーン一式を以下から入手し、Webページの指示に従いインストールします。

The xPack GNU Arm Embedded GCC https://xpack.github.io/arm-none-eabi-gcc/

(2) 開発ツールのインストール 開発ツール一式を以下から入手し、Webページの指示に従いインストールします。

xPack Windows Build Tools https://xpack.github.io/windows-build-tools/

2.2. Eclipse Embedded IDEのインストール

Eclipse Embedded IDEを以下から入手し、Webページの指示に従いインストールします。各種のEclipseのパッケージがありますので「 Eclipse IDE for Embedded C/C++ Developers」を探してダウンロードしてください。

Eclipse Packagesのダウンロードページ https://www.eclipse.org/downloads/packages/

3. プロジェクトの作成

Eclipse Embedded IDEではプログラムはプロジェクトの単位で開発します。µT-Kernel 3.0 BSPのプロジェクトを以下の手順で作成します。

3.1. GitHubからのプロジェクトの入手

TRONフォーラムのGitHubの以下のレポジトリにµT-Kernel 3.0 BSPが公開されてています。

https://github.com/tron-forum/mtk3_bsp

µT-Kernel 3.0 BSPは対象ハードウェア毎にブランチが作成されています。Raspberry Pi PicoのBSPのブランチ名は「pico_rp2040」です。
また、最新のリリース版は、GitHubのReleaseから取得することができます。「µT-Kernel 3.0 BSP for Raspberry Pi Pico」を選んでダウンロードしてください。
ダウンロードしたZipファイルは任意の場所に展開すると、Eclipse Embedded IDEのプロジェクトになります。

3.2. プロジェクトのファイル構成

µT-Kernel 3.0 BSPのプロジェクトは、以下のディレクトリおよびファイルの構成となっています。基本的にはµT-Kernel 3.0の正式リリース版に準じています。

以下にディレクトリおよびファイルの構成を示します。

ディレクトリまたはファイル名	内容
app_program	アプリケーション・プログラム用のディレクトリ
build_make	Make構築ディレクトリ（BSPでは使用しない）
config	コンフィギュレーション
device	デバイスドライバ
docs	ドキュメント
etc	リンカファイル等
include	各種定義ファイル
kernel	µT-Kernel本体
lib	ライブラリ
.settings	Eclipse Embedded IDEの設定ファイル
.cproject	Eclipse Embedded IDEのプロジェクトファイル
その他ファイル	Eclipse Embedded IDEの各種ファイルなど

ユーザが作成するアプリケーション・プログラムは、ディレクトリ「app_program」に格納します。初期状態では、サンプルコードを記述したapp_main.cファイルが格納されています。通常、ユーザが操作するのは、このディレクトリ「app_program」となります。

3.3. プロジェクトのインポート

前項で入手したプロジェクトをEclipse Embedded IDEに以下の手順で取り込みます。

(1) メニュー[File]から[Import...]を選択するとImportダイアログが開きます。

(2) [General]→[Existing Projects into Workspace]を選択し[Next]ボタンを押します。

(3) [Select root directory]の[Browse]ボタンを押し、前項のµT-Kernel 3.0 BSPプロジェクトのディレクトリを指定します。

(4) [Projects]に[mtk3bsp_pico_rp2040]が表示されるので、それをチェックします。

(5) [Finish]ボタンを押下すると、プロジェクトがワークスペースに取り込まれます。インポートが完了すると、Eclipse Embedded IDEのProject Explorerに[mtk3bsp_pico_rp2040]が表示されます。

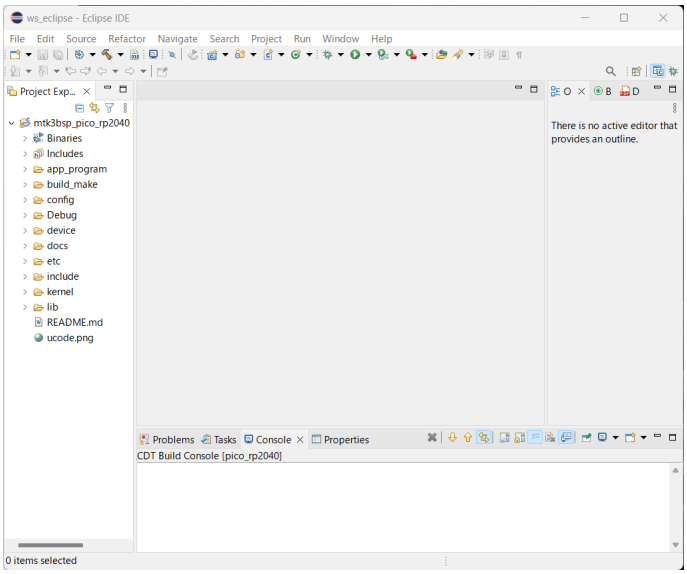

図1 Eclipse Embedded IDEに取り込んだプロジェクト

3.4. プロジェクトのビルド

µT-Kernel 3.0 BSPプロジェクトをビルドします。 Project Explorer上で[mtk3bsp_pico_rp2040]が選択されている状態で、メニュー[Project]→[Build project]を選択します。もしくはProject Explorer上の[mtk3bsp_pico_rp2040]を右クリックし、メニュー[Build project]を選択します。
プログラムのビルドが実行され、[Console]に「Build Finished. 0 errors」が表示されれば、正常にビルドが完了しました。
ビルドでエラーが発生する場合は、C言語ツールチェーンまたは開発ツールの実行パスが正しく設定されていない可能性があります。プロジェクト[mtk3bsp_pico_rp2040]が選択されている状態で、メニュー[File]から[Properties]を選択し、項目[MCU]の[ARM Toolchains Path]および[Build Tool Path]で実行パスの設定を行います。

4. アプリケーションの作成

4.1. プログラムのソースファイル

本プロジェクトでは、アプリケーション・プログラムのソースファイルは、プロジェクトのディレクトリ下の[app_program]ディレクトリに置くことを前提としています。
初期状態では、サンプルコードを記述したapp_main.cファイルが置かれています。ユーザは本ファイルの内容を変更することにより、プログラムを記述することができます。また、アプリケーションが複数のファイルから構成される場合は、このディレクトリの中にすべてを置いてください。

4.2. ワーキングセットの選択

Eclipse Embedded IDEでは、プログラムの各種設定のセットをワーキングセットと呼びます。ワーキングセットは、一つのプロジェクトに複数作ることができます。
本プロジェクトは初期状態では、「Release」と「Debug」の二つのワーキングセットが作られています。通常、デバッグ時には「Debug」を使用します。
ワーキングセットは、メニュー[Project]→[Build configurations]→[Set Active]から選択することができます。
「Release」と「Debug」の設定内容の相違を以下に示します。なお、ワーキングセットの各種設定は、アプリケーション・プログラムに応じて変更してください。

項目	Debug	Release
最適化レベル	None(-O0)	Optimize (-O1)
デバッグ・レベル	Maximum(-g3)	Default(-g)

4.3. プログラムのビルド

Project Explorer上で[mtk3bsp_pico_rp2040]が選択されている状態で、メニュー[Project]→[Build Project]を選択すると、プログラムがビルド（コンパイル、リンク）され、実行コード・ファイル
「mtk3bsp_pico_rp2040.elf」が生成されます。
実行コード・ファイルは、プロジェクトのディレクトリの下の、ワーキングセットと同名のディレクトリの中に生成されます。

5. アプリケーションの実行

生成したアプリケーション・プログラムの実行コードをマイコンボード上で実行する方法を説明します。

5.1. デバッグ環境の準備

Eclipse Embedded IDEを実行しているパソコンとマイコンボード（Raspberry Pi Pico）をデバッグプローブで接続します。デバッグプローブを介して、パソコンからマイコンボードへのプログラムの転送およびデバッグ実行を行うことができます。
デバッグプローブは、PicoprobeおよびRaspberry Pi Debug Probeを使用して検証しました。 Picoprobeはハードウェアに Raspberry Pi Picoを使用します。デバッグ対象のマイコンボードとは別にRaspberry Pi Picoを用意します。Picoprobeの使用方法はRaspberry Pi Ltdの以下のWebを参考にしてください。

https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html#debugging-using-another-raspberry-pi-pico

Raspberry Pi Debug Probeは独立した製品です。機能的にはPicoprobeと変わりません。Raspberry Pi Ltdの以下のWebを参考にしてください。

https://www.raspberrypi.com/documentation/microcontrollers/debug-probe.html#about-the-debug-probe

また、パソコンから仮想COMポート(シリアル通信ポート)として、マイコンボードとの接続が認識されます。パソコン上でターミナルエミュレータ（例：TeraTerm ※1）を実行することにより、マイコンボードと通信を行うことができるようになります。
µT-Kernel 3.0 BSPでは、デバッグ用シリアル出力をこの仮想COMポートに送信します。また、µT-Kernel 3.0のシリアル通信デバイスを使用してパソコンと通信を行うことも可能です。デバッグ用シリアル出力のシリアルポート設定は以下とします。

項目	設定内容
通信速度	115200bps
データ	8bit
パリティ	none
ストップビット	1bit

※1 Tera Term Home Page https://ttssh2.osdn.jp/

5.2. デバッグ実行

アプリケーション・プログラムのデバッグ実行は、以下の手順で行います。なお、(1)から(3)までの操作は最初に1回のみ行うだけです。
デバッグ実行を行うことにより、プログラムの実行コードはマイコンのFlash ROMに書き込まれます。よって以降は、Eclipse Embedded IDEを使用せずに単独でマイコンボードに電源を入れれば、プログラムは実行されます。

(1) アプリケーション・プログラムのビルドを行った後にメニュー[Run]→[Debug configurations]を選択します。

(2) 開いたダイアログから項目[GDB OpenOCD Debugging]を選択し、[New configuration]ボタンを押します。「mtk3bsp_pico_rp2040」が新規構成として追加されます。

(3) 追加された構成「mtk3bsp_pico_rp2040」を選択して、以下の設定を行っていきます。すでに正しい値が設定されている場合は変更の必要はありません。その他の設定についても作成するプログラムの必要に応じて変更してください。

- [Main]タブ [Project:]プロジェクトを指定します。 [C/C++ Application:]にビルドしたELFファイルを指定します。

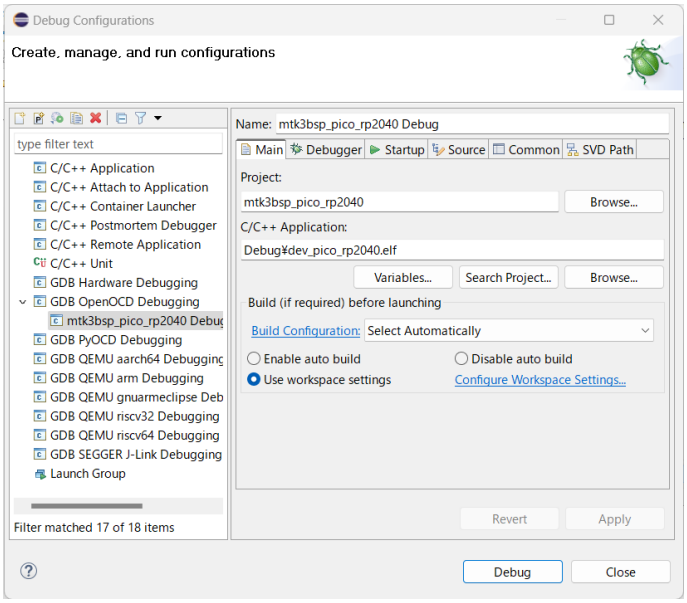

図2 [Debug configurations]の[Main]タブ

- [Debugger]タブ [OpenOCD Setup]の[Start OpenOCD locally]をチェックし、[Excutable path:]をインストールしたツールに合わせて設定します。 OpenOCDのConfig optionsは以下を設定とします。

  -f interface/cmsis-dap.cfg -f target/rp2040.cfg

  [GDB Client Setup]の[Start GDB session]をチェックし、[Commands]に以下を設定します。

  set mem inaccessible-by-default off

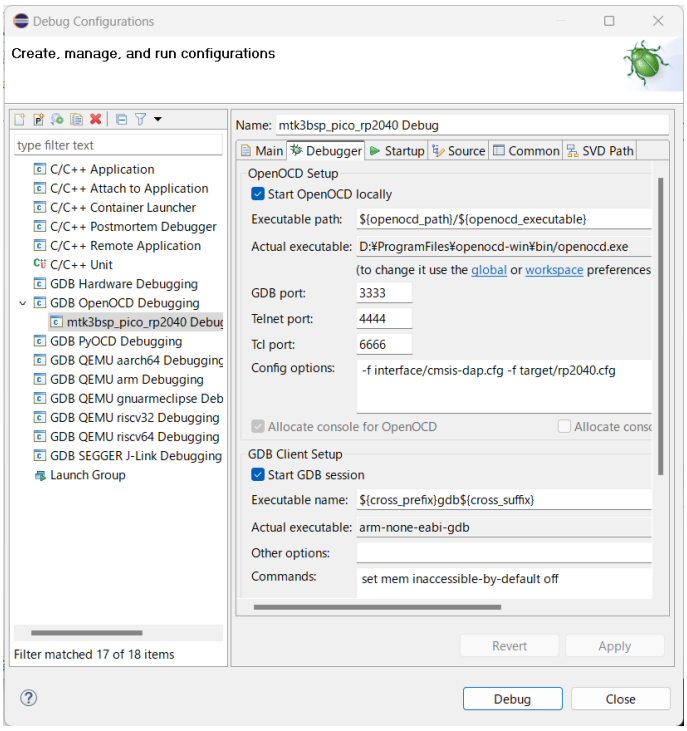

図3 [Debug configurations]の[Debugger]タブ

- 「Startup」タブ [Set breakpoint at:]の設定にデバッグ開始時にブレーク(一時停止)させる関数名を設定します。初期値の「main」だとμT-Kernel 3.0 の内部でブレークしてしまうので、アプリケーション・プログラムに合わせて変更します。通常はアプリケーション・プログラムの開始関数である「usermain」とします。

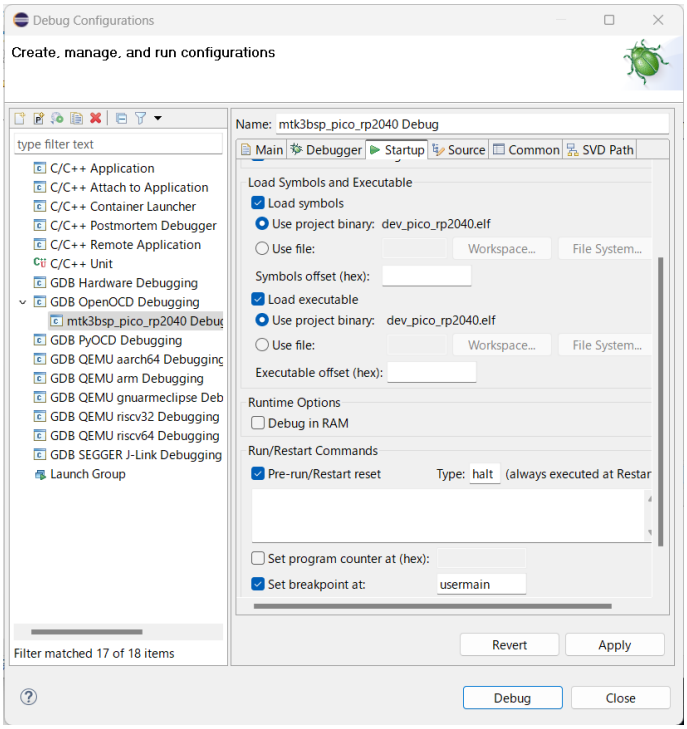

図4 [Debug configurations]の[Startup]タブ

(4) [Debug]ボタンを押すとプログラムがマイコンボードに転送され、Eclipse Embedded IDEはデバッグ画面に変わります。

(5) [Resume]ボタンを押しプログラムを実行すると設定したブレークポイントで停止します。再び[Resumeボタン]を押すとサンプル・プログラムが実行されます。サンプル・プログラムは、マイコンボード上のLED

を点滅させます。また、デバッグ出力に「User program started」の文字列を出力しますので、パソコンでターミナルエミュレータを実行していれば、その画面上に文字列が表示されます。

6. ハードウェアに関わる実装仕様

Raspberry Pi Pico向けのμT-Kernel 3.0の ハードウェアに関わる実装仕様を記します。

6.1. 基本実装仕様

6.1.1. マイコン関連

実装対象のマイコンであるRP2040の基本的な仕様を以下に示します。

項目	内容
CPUコア	ARM Cortex-M0+ (デュアルコア)
最大CPU動作周波数	133MHz
ROM	2MB(外部フラッシュメモリ)
RAM	264KB(64KB*4bank* + *4KB*2bank)
内蔵周辺モジュール	タイマー、I2C、SPI、PWM、UARTなど

本BSPではCPUコアは1個だけ使用します。メモリはROM(外部フラッシュメモリ)およびRAMを使用します。

6.1.2. 割込み定義

割込み関連の定義を以下に記します。

名称	値	意味
N_INTVEC	32	外部割込み数
N_SYSVEC	16	例外数
INTPRI_BITWIDTH	2	割込み優先度の幅(ビット数)
INTPRI_MAX_EXTINT_PRI	1	外部割込みの最高優先度
INTPRI_SVC	0	SVC例外の優先度
INTPRI_SYSTICK	0	システムタイマ例外の優先度
INTPRI_PENDSV	3	PendSV例外の優先度

割込み優先度の範囲は0～3です。ただし、ユーザプログラムから使用可能な範囲は1～3です。

6.1.3. OS内部で使用する割込み

OSの内部で使用する割込みには、以下のようにマイコンの割込みまたは例外が割り当てられます。

種類	割込み番号	割り当てられる割込み・例外	優先度
システムタイマ割込み	15	SysTick	0

種類	割込み番号	割り当てられる割込み・例外	優先度
ディスパッチ要求	14	PendSV	3
強制ディスパッチ要求	14	PendSV	3

6.1.4. クロックの設定

マイコンのクロックは以下のように設定されます。

項目	値	備考
SYSCLK	125MHz	システムクロック
CLK_PERI_FREQ	125MHz	周辺クロック
CLK_USB_FREQ	48MHz	USBクロック
CLK_ADC_FREQ	48MHz	A/DCクロック
CLK_RTC_FREQ	46875Hz	RTCクロック

6.1.5. 周辺モジュールの初期設定

(1) リセットの解除 OS起動時の初期化で以下の周辺モジュールのリセットが解除されます。

- コンフィギュレーションUSE_SDEV_DRVが無効(0)の場合

周辺モジュール名	用途
IO_BANK0	I/O端子の機能選択
PADS_BANK0	I/O端子の各種設定
UART0	デバッグ用シリアル入出力(T-Monitor)
PWM	物理タイマ

- コンフィギュレーションUSE_SDEV_DRVが有効(1)の場合

周辺モジュール名	用途
IO_BANK0	マイコン端子の機能選択
PADS_BANK0	マイコン端子の各種設定
UART0	デバッグ用シリアル入出力(T-Monitor)
PWM	物理タイマ
ADC	A/Dコンバータ
I2C0	I2C通信

(2) I/O端子の設定 OS起動時の初期化で以下のようにI/O端子が設定されます。

● コンフィギュレーションUSE_SDEV_DRVが無効(0)の場合

I/Oポート名	機能名	用途
P0	UART0_TX	デバッグ用シリアル入出力(T-Monitor)
P1	UART0_RX	同上
P25	出力ポート	オンボードLED

● コンフィギュレーションUSE_SDEV_DRVが有効(1)の場合

I/Oポート名	機能名	用途
P0	UART0_TX	デバッグ用シリアル入出力(T-Monitor)
P1	UART0_RX	同上
P8	I2C0_SDA	I2C通信
P9	I2C0_SCL	I2C通信
P25	出力ポート	オンボードLED
P26	ADC0	アナログ入力
P27	ADC1	アナログ入力
P28	ADC2	アナログ入力

6.2. 物理タイマ機能

6.2.1. 使用するハードウェアタイマ

物理タイマ機能は、マイコン内蔵の以下のPWMを以下のように使用します。物理タイマとして使用していないPWMは他の用途に使用可能です。

物理タイマ番号	対応するタイマ	ビット幅
1	PWM ch.0	16
2	PWM ch.1	16
3	PWM ch.2	16
4	PWM ch.3	16
5	PWM ch.4	16
6	PWM ch.5	16
7	PWM ch.6	16
8	PWM ch.7	16

6.2.2. タイマの設定

初期設定では、物理タイマのクロックはシステムタイマの周波数（125MHz）を未分周で使用します。 以下のファイルの設定値を変更することにより周波数の分周値を変更できます。この値はPWMのCHx_DIVレジスタに設定されます。

```
include\sys\sysdepend\cpu\rp2040\sysdef.h
```

```
#define PTMR_DIV_CH0    (1<<4)
#define PTMR_DIV_CH1    (1<<4)
#define PTMR_DIV_CH2    (1<<4)
#define PTMR_DIV_CH3    (1<<4)
#define PTMR_DIV_CH4    (1<<4)
#define PTMR_DIV_CH5    (1<<4)
#define PTMR_DIV_CH6    (1<<4)
#define PTMR_DIV_CH7    (1<<4)
```

6.2.3. 物理タイマ割込み

物理タイマはその内部処理でPWMの割込みを使用します。PWMの割込みは以下のように定義されます。

項目	値	備考
割込み番号	4	PWM_IRQ_WRAP割込み
優先度	2	INTPRI_PTIMの定義を変更することにより変更可能

6.2.4. デバイスドライバ

本BSPは、TRONフォーラムが提供するμT-Kernel 3.0のサンプル・デバイスドライバを、Raspberry Pi Pico向けに移植し実装しています。

種別	対象I/Oデバイス	デバイス名
シリアル通信デバイスドライバ	UART0	sera
	UART1	serb
A/D変換デバイスドライバ	ADC	adca
I2C通信デバイスドライバ	I2C0	iica
	I2C1	iicb

6.3. BSPライブラリ

BSPライブラリはμTK3.0 BSP独自のユーティリティ・ライブラリです。これはμT-Kernel 3.0の仕様には含まれません。

6.3.1. GPIO制御関数

GPIOを制御する以下のライブラリ関数を提供します。

(1) GPIO設定関数

項目	内容
書式	ER gpio_set_pin(UINT no, UINT mode)
引数	UINT no : GPIOの番号 UINT mode : 設定するI/Oモード
戻り値	エラーコード
機能	引数noで指定されたポートを入力または出力に設定します GPIO_MODE_OUT : 出力 GPIO_MODE_IN : 入力

(2) GPIO出力関数

項目	内容
書式	ER gpio_set_val(UINT no, UINT val)
引数	UINT no : GPIOの番号 UINT mode : 出力するデータ
戻り値	エラーコード
機能	引数noで指定された出力ポートから引数valで指定されたデータを出力します 0: L出力 1: H出力

(3) GPIO入力関数

項目	内容
書式	UINT gpio_get_val(UINT no)
引数	UINT no : GPIOの番号
戻り値	エラーコード
機能	引数noで指定された入力ポートからのデータを戻り値として返します

6.3.2. PWM制御関数

PWMを制御する以下のライブラリ関数を提供します。

(1) PWMカウンタ上限値設定

項目	内容
書式	ER pwm_set_top(UINT no, UW top)
引数	UINT no : PWMチャンネル番号 UW top : カウンタ上限値

項目	内容
戻り値	エラーコード
機能	引数noで指定されたPWMチャンネルのカウンタの上限値を引数topで指定された値に設定します この値はPWMのCHx_TOPレジスタに設定されます

(2) PWMレベル設定

項目	内容
書式	ER pwm_set_cc(UINT no, UW cc)
引数	UINT no : PWMチャンネル番号 UW cc : カウンタコンペア値
戻り値	エラーコード
機能	引数noで指定されたPWMチャンネルのカウンタコンペア値を引数ccで指定された値に設定します この値はPWMのCHx_CCレジスタに設定されます

(3) PWMイネーブル関数

項目	内容
書式	ER pwm_set_enabled(UINT no, BOOL enable)
引数	UINT no : PWMチャンネル番号 BOOL enable : !0 PWM有効、0 PWM無効
戻り値	エラーコード
機能	引数noで指定されたPWMチャンネルを引数enableの値に応じで有効(enable!=0)または無効(enable=0)にします

6.4. T-Monitor互換ライブラリ

6.4.1. コンソール入出力

T-Monitor互換ライブラリのAPIによるコンソール入出力の仕様を以下に示します。

項目	内容
使用デバイス	内蔵UART (UART0)
ボーレート	115200bps
データ形式	data 8bit, stop 1bit, no parity

7. 更新履歴

版数	日付	内容
1.00.01	2023.05.24	・タイトルを「マニュアル」から「ユーザズ・マニュアル」に変更 ・内容および構成の全面見直し
1.00.00	2022.11.30	・初版