

# $\mu$ T-Kernel 3.0 実装仕様書 (CY8CKIT-CY8C6)

**Rev 3.00.06**

January, 2023



**$\mu$ T-Kernel 3.0**



# 目次

1. はじめに .....	3
1.1. 本書について .....	3
1.2. 表記について .....	3
2. 概要 .....	4
2.1. 対象ハードウェア .....	4
2.2. ターゲット名 .....	4
2.3. 関連ドキュメント .....	5
2.4. 開発環境 .....	6
2.5. ディレクトリ構成 .....	6
2.5.1. プロジェクト全体のディレクトリ構成 .....	6
2.5.2. 機種依存定義ディレクトリの構成 .....	9
3. 基本実装 .....	10
3.1. 対象マイコン .....	10
3.2. 実行モードと保護レベル .....	10
3.3. CPU レジスタ .....	10
3.4. 低消費電力モードと省電力機能 .....	11
3.5. コプロセッサ対応 .....	11
4. メモリ .....	12
4.1. メモリモデル .....	12
4.2. マイコンのアドレス・マップ .....	12
4.3. OS のメモリマップ .....	13
4.4. スタック .....	15
4.5. $\mu$ T-Kernel 管理領域 .....	15
5. 割込みおよび例外 .....	17
5.1. マイコンの割込みおよび例外 .....	17
5.2. ベクタテーブル .....	17
5.3. 割込み優先度とクリティカルセクション .....	19
5.3.1. 割込み優先度 .....	19
5.3.2. 多重割込み対応 .....	19
5.3.3. クリティカルセクション .....	20
5.4. OS 内部で使用する割込み .....	20
5.5. $\mu$ T-Kernel/OS の割込み管理機能 .....	21
5.5.1. 割込み番号 .....	21
5.5.2. 割込みハンドラ属性 .....	21
5.5.3. デフォルトハンドラ .....	22
5.6. $\mu$ T-Kernel/SM の割込み管理機能 .....	22

5.6.1.	割込みの優先度 .....	22
5.6.2.	CPU 割込み制御 .....	22
5.6.3.	割込みコントローラ制御 .....	23
5.7.	OS 管理外割込み .....	24
5.8.	その他の例外 .....	24
6.	起動および終了処理 .....	26
6.1.	リセット処理 .....	26
6.2.	ハードウェアの初期化および終了処理 .....	28
6.3.	デバイスドライバの実行および終了 .....	30
7.	タスク .....	31
7.1.	タスク属性 .....	31
7.2.	タスクの処理ルーチン .....	31
7.3.	タスク・コンテキスト情報 .....	31
8.	時間管理機能 .....	33
8.1.	システムタイマ .....	33
8.2.	タイムイベントハンドラ .....	33
8.3.	物理タイマ機能 .....	33
9.	その他の実装仕様 .....	34
9.1.	FPU 関連 .....	34
9.1.1.	FPU の初期設定 .....	34
9.1.2.	FPU 関連 API .....	34
9.1.3.	FPU 使用の際の注意点 .....	35
9.2.	T-Monitor 互換ライブラリ .....	35
9.2.1.	ライブラリ初期化 .....	35
9.2.2.	コンソール入出力 .....	36

## 1. はじめに


本品はユーシーテクノロジー(株)が開発した CY8CKIT-062S2-43012(以下、CY8CKIT-062S2-43012、または CY8CKIT-CY8C6 と記載する)対応の μT-Kernel 3.0 である。

本品は、トロンプフォーラムの配布する μT-Kernel 3.0 をベースに、Infineon 製の評価ボード CY8CKIT-062S2-43012 で動作させるための機種依存部を追加してある。



### 1.1. 本書について

本書は CY8CKIT-062S2-43012 向けの μT-Kernel 3.0 の実装仕様について記載した実装仕様書である。本書の対象となる実装は、ユーシーテクノロジー(株)が公開している μT-Kernel 3.0 BSP(Board Support Package)に含まれている。

以降、単に OS と称する場合は μT-Kernel 3.0 を示し、**本実装**と称する場合は CY8CKIT-062S2-43012 向けのソースコードの実装を示すものとする。

-  ハードウェアに依存しない共通の実装仕様は「μT-Kernel 3.0 共通実装仕様書」を参照のこと。

### 1.2. 表記について

表記	説明
[ ]	[ ]はソフトウェア画面のボタンやメニューを表す。
「 」	「 」はソフトウェア画面に表示された項目などを表す。
	注意が必要な内容を記述する。
	補足やヒントなどの内容を記述する。

## 2. 概要

本書では、μT-Kernel 3.0 BSP に含まれる CY8CKIT-062S2-43012 向けの実装仕様について説明する。

- ① 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ある。

### 2.1. 対象ハードウェア

開発対象のハードウェアおよび OS は以下である。

表 2-1 開発対象のハードウェアと OS

分類	名称	備考
マイコン	CY8C624ABZI-S2D44A0 (ARM Cortex-M4F, Cortex-M0+)	Infineon Technologies AG
OS	μT-Kernel 3.00.06	トロンフォーラム
実機 (マイコンボード)	CY8CKIT-062S2-43012 (CY8CKIT-CY8C6)	Infineon Technologies AG

- ① 本実装の最新版は、ユーシーテクノロジー(株)の GitHub リポジトリにて公開している。

[https://github.com/UCTechnology/mtk3\\_bsp](https://github.com/UCTechnology/mtk3_bsp)

- ① μT-Kernel 3.0 の最新版は以下の GitHub リポジトリにて公開されている。

[https://github.com/tron-forum/mtkernel\\_3](https://github.com/tron-forum/mtkernel_3)

### 2.2. ターゲット名

CY8CKIT-062S2-43012 のターゲット名および識別名は以下とする。

表 2-2 CY8CKIT-062S2-43012 のターゲット名および識別名

分類	名称	対象
ターゲット名	_CY8CKIT_CY8C6_	CY8CKIT-062S2-43012
対象システム	CY8CKIT_CY8C6	CY8CKIT-062S2-43012
対象 CPU	CPU_CY8C624A	CY8CKIT-062S2-43012
対象 CPU アーキテクチャ	CPU_CORE_ARMV7M	ARMv7-M アーキテクチャ
	CPU_CORE_ACM4F	ARM Cortex-M4F コア

識別名は以下のファイルで定義される。

```
include/sys/sysdepend/cy8ckit_cy8c6/machine.h
```

## 2.3. 関連ドキュメント

OS の標準的な仕様は「μT-Kernel 3.0 仕様書」に記載される。

ハードウェアに依存しない共通の実装仕様は、「μT-Kernel 3.0 共通実装仕様書」に記載される。

また、対象とするマイコンを含むハードウェアの仕様は、それぞれの仕様書などのドキュメントに記載される。

以下に関連するドキュメントを記す。

表 2-3 関連ドキュメント

分類	名称	発行
OS	μT-Kernel 3.0 仕様書 (Ver. 3.00.00)	トロンフォーラム TEF020-S004-03.00.00
	μT-Kernel 3.0 共通実装仕様書 (Ver.1.00.08)	トロンフォーラム TEF033-W002-211115
T-Monitor	T-Monitor 仕様書	トロンフォーラム TEF020-S002-01.00.01
デバイスドライバ	μT-Kernel 3.0 デバイスドライバ説明書 (Ver.1.00.02)	トロンフォーラム TEF033-W007-210331
ターゲットボード	取扱説明書 CY8CKIT-062S2-43012 PSoC 62S2 Wi-Fi BT Pioneer Kit Guide	Infineon Technologies AG Doc. 002-28109 Rev. *G (2021/03/26)
	ユーザーガイド KitProg3 user guide	Doc. 002-24616 Rev. *P (2022/05/02)
	回路図 CY8CMOD-062S2-43012 PSoC 6 (2M) with CYW43012 Carrier Module	Rev 03 2020/01/31
	回路図 CY8CKIT-062S2-43012 PSoC 62S2 Wi-Fi BT Pioneer Kit	Rev 05 2019/09/18
搭載マイコン	データシート PSoC 6 MCU: CY8C62x8, CY8C62xA Datasheet PSoC 62 MCU	Doc. 002-23185 Rev. *Q (2022/04/12)
	テクニカルリファレンスマニュアル PSoC 6 MCU: CY8C6xx8, CY8C6xxA Architecture Technical Reference Manual (TRM) PSoC 61, PSoC 62 MCU	Doc. 002-24529 Rev. *G (2020/12/02)
	テクニカルリファレンスマニュアル PSoC 6 MCU: CY8C6xx8, CY8C6xxA Registers Technical Reference Manual (TRM) PSoC 62 MCU	Doc. 002-24591 Rev. *E (2020/07/15)
	ModusToolbox ユーザーガイド ModusToolbox user guide	Doc. 002-29893 Rev. *N (2022/04/07)

## 2.4. 開発環境

本実装では、開発環境として ModusToolbox を使用する。

ModusToolbox は Infineon 製マイコン向けの統合開発環境であり、コンフィギュレーションツール、低レベルドライバ、ミドルウェアライブラリなどが含まれている。

また、すべての ModusToolbox ツールと統合されたフローを提供する Eclipse IDE も含まれている。このため、基本的な開発作業は Eclipse の UI を通して行うことができ、ModusToolbox の各種ツールも Eclipse の UI から起動することができる。

本実装では ModusToolbox 用の Eclipse である Eclipse IDE for ModusToolbox を用いて開発を行う。以降、Eclipse IDE for ModusToolbox を ModusToolbox と記述する。

本実装では ModusToolbox が生成した起動処理ルーチンやデバイスの初期化ルーチンを利用しており、システム構成を変更する場合は ModusToolbox のツールを利用して作業することを想定している。

なお、ModusToolbox が提供する機能を μ T-Kernel のアプリケーションから利用できることを保証するものではない。特に各機能がスレッドセーフな実装になっているかなどは不明であるので、μ T-Kernel のアプリケーションから ModusToolbox が提供する機能を利用する場合は排他制御をかけるなどして、十分に注意して利用する必要がある。

- ❶ ModusToolbox の詳細については、「ModusToolbox ユーザーガイド」を参照のこと。
- ❷ ModusToolbox が生成した起動処理ルーチンの具体的な利用方法については、「6. 起動および終了処理」で詳しく説明する。

## 2.5. ディレクトリ構成

### 2.5.1. プロジェクト全体のディレクトリ構成

本実装では ModusToolbox で生成したプロジェクトに対して、μ T-Kernel 3.0 を Git のサブモジュールとして追加して利用する。このため、プロジェクト全体のディレクトリ構成は以下の通りとなる。

<ベースディレクトリ>	
└─ mtb_workspace/	
└─ <b>uTK3/</b>	<b>※青:本実装用のディレクトリ、ファイル</b>
└─ mtb_shared/	ModusToolbox 用のワークスペース
└─ <APP-PRJ#1>/	μ T-Kernel 3.0 用プロジェクトのベース
:	ModusToolbox 用の共通モジュール
└─ <b>&lt;APP-PRJ#N&gt;/</b>	アプリケーションプロジェクト#1
└─ mtkernel_3/	アプリケーションプロジェクト#N
└─ .gitmodules	μ T-Kernel 3.0 本体(サブモジュール)
└─ app_program/	サブモジュール用設定ファイル
└─ Makefile	μ T-Kernel 3.0 用アプリケーション
└─ cy8c6xxa_cm4_dual.ld	ビルド構成を指定するためのファイル
└─ .cyignore	本実装専用のリンクスクリプト
└─ makefile.init	ビルド対象外を指定するためのファイル
└─ main.c	ビルド設定指定ファイル
	ModusToolbox が生成したサンプルコード

└─ libs/	ModusToolbox 設定ファイル
└─ .mtbLaunchConfigs/	プログラム起動用の設定ファイル
└─ .settings/	ModusToolbox 用の設定ファイル
└─ build/	ビルド用ディレクトリ
└─ deps/	ModusToolbox 用の設定ファイル
└─ images/	README.md 用の画像ファイル
└─ .cproject	ModusToolbox 用の設定ファイル
└─ .gitignore	ModusToolbox 用の Git の設定ファイル
└─ .project	ModusToolbox 用の設定ファイル
└─ LICENSE	プロジェクトのライセンス
└─ README.md	プロジェクト生成直後に表示されるファイル

以下、主なディレクトリとファイルの概要について説明する。

#### (a) <ベースディレクトリ>

ModusToolbox 起動時にワークスペースの親ディレクトリとして指定する。

場所は任意である。下図では `C:\mtb` がベースディレクトリとなる。

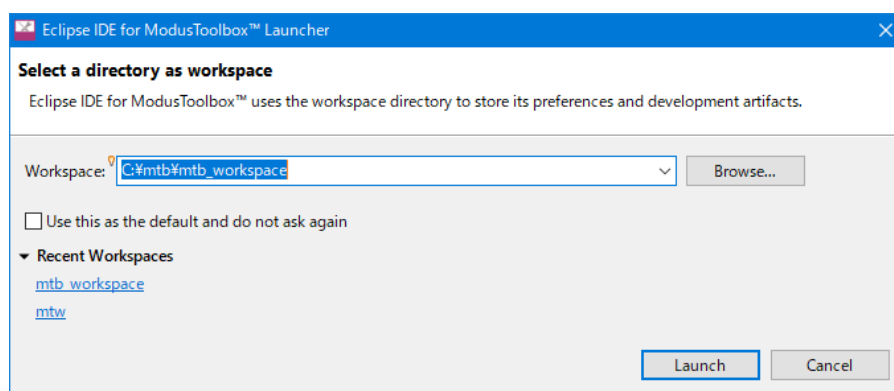


図 1 ワークスペース・ランチャーでワークスペースを指定

#### (b) `mtb_workspace/`

ModusToolbox を起動する際に指定する作業ディレクトリである。

#### (c) `uTK3/`

μT-Kernel 3.0 用プロジェクトのベースである。Project Creator でサンプルプロジェクトを生成する際に **Applicaton(s) Root Path** として指定する。名称は任意である。デフォルトは `mtb_workspace/` であるので、そのまま `mtb_workspace/` にプロジェクトを生成しても構わないが、今回は ModusToolbox が生成するプロジェクトと区別するため別のディレクトリ `uTK3/` を生成した。

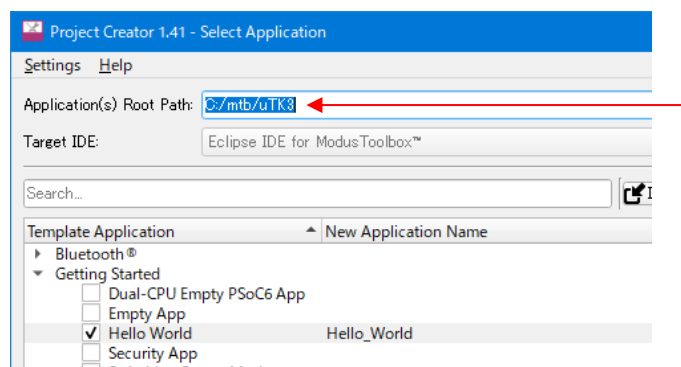


図 2 Application(s) Root Path の指定



(d) `mtb_shared/`

Project Creator で生成するサンプルプロジェクトで利用する各種モジュールが格納されるディレクトリである。

(e) `<APP-PRJ#x>/`

Project Creator で生成するサンプルプロジェクトである。

サンプルプロジェクト選択時に **New Application Name** の列で名称を指定する。

下図ではデフォルトの `Hello_World` のまま変更していない。

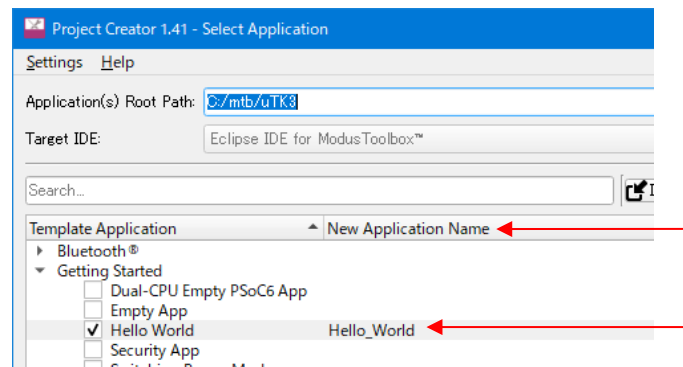


図 3 New Application Name の指定

(f) `mtkernel_3/`

μ T-Kernel 3.0 本体である。Git のサブモジュール機能を利用して取り込む。

(g) `.gitmodules`

`mtkernel_3/`用の設定ファイルである。サブモジュール取り込み時に自動生成される。

(h) `app_program/`

μ T-Kernel 3.0 用のアプリケーションプログラムを記述する。

(i) `Makefile`

ModusToolbox でビルドする際のコンフィギュレーションを指定する。

本ファイルにはコンパイルなどの詳細情報は記載されていない。コンパイルなどの詳細な設定は、自動生成される `build/compile_commands.json` に記載されている。

(j) `cy8c6xxa_cm4_dual.ld`

本実装用のリンカスクリプトである。

(k) `.cyignore`

ビルド対象外とするファイル、ディレクトリを指定する。

(l) `makefile.init`

ビルド用設定ファイルである。

(m) `main.c`

Project Creator が生成するソースコードであり、`main()`関数が含まれる。

μ T-Kernel 3.0 では使用しない。

(n) `libs/`

Device Configurator など設定(調整)して自動生成されたコードなどが含まれる。

(o) `.mtbLaunchConfigs/` ~ `README.md`

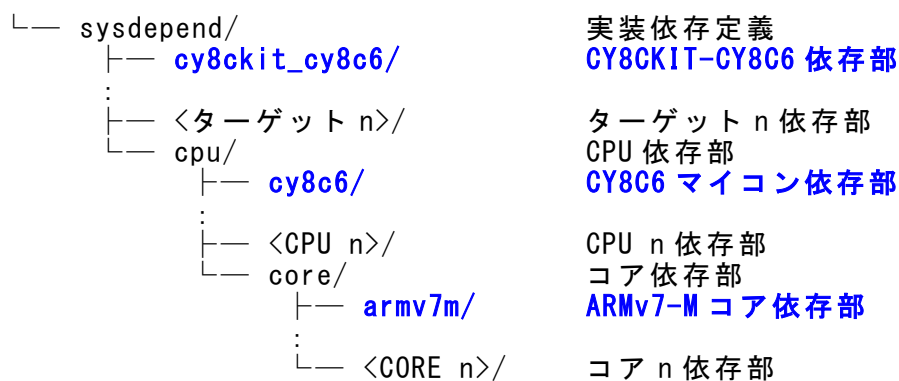
Project Creator によって自動的に生成される。

詳細については ModusToolbox の資料を確認のこと。

## 2.5.2. 機種依存定義ディレクトリの構成

サブモジュールとして追加した `mtkernel_3/` の下に μT-Kernel 3.0 のソースコードが含まれる。μT-Kernel 3.0 のソースコードは共通部(機種に依存しない μT-Kernel 3.0 に共通のコード)と機種依存部(機種に合わせて実装されたコード)に分けられており、機種依存部は機種依存定義ディレクトリ `sysdepend/` の下にまとめられている。

更に `sysdepend/` の中の **太字** で書かれた箇所が、本実装のディレクトリとなる。



「**ARMv7-M コア依存部**」は、ARMv7-M コアに共通するコードであり、他の共通のコアを有するマイコンでも使用される。

「**CY8C6 マイコン依存部**」は、コア依存部以外の本マイコンに固有のコードである。

「**CY8CKIT-CY8C6 依存部**」は、コア依存部およびマイコン依存部以外の **CY8CKIT-CY8C6** のハードウェアに固有のコードである。

### 3. 基本実装

#### 3.1. 対象マイコン

実装対象のマイコンの基本的な仕様を以下に記す。

表 3-1 CY8CKIT-062S2-43012 の基本仕様

項目	内容
CPU コア	ARM Cortex-M4F, ARM Cortex-M0+ ※本実装では OS は ARM Cortex-M4F のみを対象とする。
ROM	2MB (Application Flash)
RAM	1MB (SRAM) ※本実装では OS は SRAM1 のみを使用する。

❶ メモリマップの詳細については「4. メモリ」を参照のこと。

#### 3.2. 実行モードと保護レベル

ARM Cortex-M4F コアは、プログラムの動作モードとして、スレッドモードとハンドラモードの二つのモードをもち、それぞれに特権モードまたはユーザモードの実行モードを割り当てることができる。

本実装では、マイコンの実行モードは、特権モードのみを使用する。

OS が提供する保護レベルは、マイコンの実行モードが特権モードのみなので、すべて保護レベル 0 とみなす。カーネルオブジェクトに対して保護レベル 1～3 を指定しても保護レベル 0 を指定されたものとして扱う。

プロファイル TK\_MEM\_RNG0～TK\_MEM\_RNG3 はすべて 0 が返される。

#### 3.3. CPU レジスタ

本マイコンは内部レジスタとして、汎用レジスタ (R0～R12)、SP(R13)、LR(R14)、PC(R15)、xPSR を有する。

OS の API(tk\_set\_reg、tk\_get\_reg)を用いて実行中のタスクのコンテキストのレジスタ値を操作できる。

API で使用するマイコンのレジスタのセットは以下のように定義される。

##### (1) 汎用レジスタ

```
typedef struct t_regs {
    VW    r[13];    /* 汎用レジスタ      R0-R12 */
    void  *lr;      /* リンクレジスタ   R14    */
} T_REGS;
```

## (2) 例外時に保存されるレジスタ T\_EIT

```
typedef struct t_eit {
    void *pc;          /* プログラムカウンタ R15 */
    UW xpsr;           /* プログラムステートレジスタ */
    UW taskmode;       /* タスクモード(仮想レジスタ) */
} T_EIT;
```

## (3) 制御レジスタ T\_CREGS

```
typedef struct t_cregs {
    void *ssp;         /* System stack pointer R13_svc */
    // void *usp;      /* User stack pointer R13_usr */
} T_CREGS;
```

OS の API によって操作されるのは、実際にはスタック上に退避されたレジスタの値である。よって、実行中のタスクに対して操作することはできない(OS 仕様上、自タスクへの操作、またはタスク独立部からの API 呼出しは禁止されている)。

**taskmode** レジスタは、マイコンの物理的なレジスタを割り当てず、OS 内の仮想的なレジスタとして実装する。本レジスタは、メモリへのアクセス権限(保護レベル)を保持する仮想レジスタである。ただし、本実装ではプログラムは特権モードでのみ実行されるので、常に値は 0 となる。

### 3.4. 低消費電力モードと省電力機能

省電力機能はサポートしていない。プロファイル TK\_SUPPORT\_LOWPPOWER は FALSE である。よって、マイコンの低消費電力モードに対応する機能は持たない。

省電力機能の API(low\_pow、off\_pow)は、kernel/sysdepend/iote\_stm32l4/power\_save.c に空関数として記述されている。本関数に適切な省電力処理を記述し、プロファイル TK\_SUPPORT\_LOWPPOWER に TRUE を指定すれば、OS の省電力機能(tk\_set\_pow)が使用可能となる。

### 3.5. コプロセッサ対応

本マイコンは IEEE754 規格に準拠した FPU(浮動小数点ユニット)を内蔵する。コンフィギュレーション USE\_FPU に TRUE を指定すると、OS で FPU 対応の機能が有効となり、FPU にコプロセッサ番号 0 が割り当てられる。

FPU が有効の場合、以下の機能が有効となる。

- ・ 起動時の FPU の有効化                      「9.1.1. FPU の初期設定」参照
- ・ TA\_FPU 属性のタスク                      「7.3. タスク・コンテキスト情報」参照
- ・ コプロセッサレジスタ操作 API              「9.1.2. FPU 関連 API」参照

## 4. メモリ

### 4.1. メモリモデル

Cortex-M4F は 32bit のアドレス空間を有する。MMU(Memory Management Unit：メモリ管理ユニット)は有さず、単一の物理アドレス空間のみである。

本実装では、プログラムは一つの実行オブジェクトに静的にリンクされていることを前提とする。OS とユーザプログラム（アプリケーションなど）は静的にリンクされており、関数呼び出しが可能とする。

メモリは Cortex-M4F と Cortex-M0+で共通である。このため、メモリ領域は Cortex-M4F と Cortex-M0+の両方が利用できるように割り当てる必要がある。

### 4.2. マイコンのアドレス・マップ

以下にマイコンのアドレス・マップを記す。

詳細はマイコンのデータシートを参照のこと。

表 4-1 搭載マイコンのアドレス・マップ(全体)

アドレス	名称	備考
0x00000000 ～ 0x1FFFFFFF	プログラムコード領域	プログラムコード領域 データも配置可能であり、アドレス 0 から始まる例外ベクタテーブルも含まれる。
0x20000000 ～ 0x3FFFFFFF	データ領域	データ領域 本マイコン(PSoC6)ではサポートされていない。
0x40000000 ～ 0x5FFFFFFF	ペリフェラルレジスタ領域	ペリフェラルレジスタ領域（コード実行不可） この領域へのビットバンドアクセスは、本マイコン(PSoC 6)ではサポートされていない。
0x60000000 ～ 0x9FFFFFFF	外部 RAM 領域	Quad-SPI または Serial Memory Interface で接続されるメモリ領域。（コード実行可能）
0xA0000000 ～ 0xDFFFFFFF	外部デバイス領域	本マイコン(PSoC 6)では未使用である。
0xE0000000 ～ 0xE0FFFFFF	システムレジスタ領域	CPU コア内のペリフェラルレジスタにアクセスするための領域
0xE0100000 ～ 0xFFFFFFF	デバイス固有レジスタ領域	デバイス固有のシステムレジスタにアクセスするための領域

ペリフェラルのビットバンドアクセスは、本マイコンではサポートされていない。

本マイコンではデータ領域(0x20000000～)は利用せず、プログラムコード領域(0x00000000～)にある SRAM を利用する。

プログラムコード領域(0x00000000～)のアドレス・マップは以下のとおりである。

表 4-2 プログラムコード領域のアドレス・マップ

アドレス	メモリタイプ	サイズ
0x00000000 ～ 0x0000FFFF	監視 ROM Cortex-M0+によって保護コンテキスト 0 で実行される。	64KB
0x08000000 ～ 0x080FFFFFFF	内蔵 SRAM	1MB
0x10000000 ～ 0x101FFFFFFF	アプリケーションフラッシュ (ROM)	2MB
0x14000000 ～ 0x14007FFF	補助フラッシュメモリ EEPROM エミュレーションに使用可能 (本実装では利用していない。)	32KB
0x16000000 ～ 0x16007FFF	監督用フラッシュ (本実装では利用していない。)	32KB

#### 4.3. OS のメモリマップ

本実装では、アプリケーションフラッシュ (ROM) および内蔵 SRAM を使用する。

OS を含む全てのプログラムのコードはアプリケーションフラッシュ (ROM) に配置され、実行される。

例外ベクタテーブルは、リセット時はアプリケーションフラッシュ (ROM) 上にあるが、OS の初期化処理にて内蔵 SRAM に転送される。ただし、コンフィグレーション `USE_STATIC_IVT` を有効(1)に設定することにより、SRAM への再配置を禁止することができる(初期設定では無効(0))。再配置を禁止した場合、API による割込みハンドラの登録が不可となる。

以下にアプリケーションフラッシュ (ROM) および内蔵 SRAM のメモリマップを示す。

表中でアドレスに(ー)が記載された箇所はデータのサイズにより C 言語の処理系にてアドレスが決定される(OS ではアドレスの指定は行っていない)。

表 4-3 アプリケーションフラッシュ (ROM) のメモリマップ : 0x10000000～0x101FFFFFFF

アドレス	セクション名	種別	内容
0x10000000 ～ (ー)	.cy_m0p_image	Cortex-M0+用 プログラム	Cortex-M0+用のプログラム イメージ
(ー)～(ー)	.vector	例外ベクタ テーブル	例外や割込みのベクタテー ブル SRAM に再配置される。
0x10002000 ～ (ー)	.text	プログラム コード	プログラムコードが配置され る。

アドレス	セクション名	種別	内容
(-) ~ (-)	.ARM.exidx	ARM EXIDX	libgcc が使用する C++ の例外処理用コード等が含まれるセクション
(-) ~ (-)	.copy.table .zero.table	定数データ	定数データなどが配置される。

表 4-4 内蔵 SRAM のメモリマップ : 0x08000000 ~ 0x080FFFFF

アドレス	セクション名	種別	サイズ	内容
0x08000000 ~ 0x08001FFF		Cortex-M0+ 用	8KB	
0x08002000 ~ (-)	.ramVectors	例外ベクタ テーブル	736B	例外や割込みのベクタ テーブル OS の初期化後に有効
(-) ~ (-)	.data	データ領域		プログラムの変数領域 (初期値あり)
(-) ~ (-)	.cy_sharedmem	コア共有領域		両方のコアからのアクセ スが必要なオブジェ クトを配置する。
(-) ~ (-)	.noinit	非初期化 変数領域		プログラムの変数領域 (初期値なし)
(-) ~ (-)	.bss	BSS 領域		プログラムの変数領域 (0 初期化)
(-) ~ (-)	.utk_mgt	μT-Kernel 管理領域	512KB	OS 内部で利用する動的 メモリに割り当てる 領域
(-) ~ (-)	.heap	ヒープ領域		ModusToolbox 用
0x080FE800 ~ 0x080FF7FF	.stack_dummy		4KB	初期化スタック領域 初期化処理時のみに使 用する。
0x080FF800 ~ 0x080FFFFF		システム用	2KB	ModusToolbox 用

❗ 各セクションの並び(やサイズ)はリンカスクリプト `cy8c6xxa_cm4_dual.ld` によって指定する。

💡 灰色のエリアは Cortex-M0+ や ModusToolbox が用意するプログラムで利用しているので μT-Kernel 3.0 から直接利用することはできない。

#### 4.4. スタック

本マイコンには、MSP(Main Stack Pointer)と PSP(Process Stack Pointer)の二種類のスタックが存在する。ただし、本実装では MSP のみを使用しており、PSP は使用しない。本実装で使用するスタックには共通仕様に従い以下の種類がある。

- (1) タスクスタック
- (2) 例外スタック
- (3) テンポラリスタック

なお、本実装では OS 初期化処理のみ例外スタックを使用する。初期化終了後は、割込みハンドラなどのタスク独立部もタスクスタックを使用し、例外スタックは使用しない。

よって、タスクスタックのサイズには、タスク実行中に発生した割込みによるスタックの使用サイズも加味しなくてはならない。

#### 4.5. μ T-Kernel 管理領域

OS の API の処理において以下のメモリが μ T-Kernel 管理領域から動的に確保される。

- ・ メモリプールのデータ領域
- ・ メッセージバッファのデータ領域
- ・ タスクのスタック

μ T-Kernel 管理領域は、μ T-Kernel のメモリ管理機能で使用する領域であり、本実装では BSS 領域とヒープ領域の間に配置してある。

サイズは 512KB であり、リンカスクリプト `cy8c6xxa_cm4_dual.ld` で指定している。μ T-Kernel 管理領域のサイズを変更する場合は、リンカスクリプトの設定値を変更する。

##### `cy8c6xxa_cm4_dual.ld` : μ T-Kernel 管理領域用のセクション指定

---

```
/* .utk_mgt section doesn't contains any symbols. It is only used
 * for linker to calculate size of uT-Kernel MANAGEMENT SECTION. */
.utk_mgt (NOLOAD):
{
    . = ALIGN(4);
    __utk_mgt_start__ = .;
    KEEP(*(.utk_mgt*))
    . = . + 512 * 1024;
    __utk_mgt_end__ = .;
} > ram
```

---

また、μ T-Kernel 管理領域はセクションとして専用の領域を確保してある。このため、起動時の `Reset_Handler()` ではシステムメモリ領域を管理する変数(`kn1_lowmem_top`、`kn1_lowmem_limit`)に対してセクション `utk_mgt` の先頭アドレスと終端のアドレスを直接指定する実装にしてある。



kernel/sysdepend/cpu/cy8c6/reset\_hdl.c : μT-Kernel 管理領域の先頭と終端の設定

---

```
EXPORT void Reset_Handler(void)
{
    :
    #if USE_IMALLOC
        /* set SYSTEM MEMORY AREA */
        knl_lowmem_top = (UW*)&__utk_mgt_start__;
        knl_lowmem_limit = (UW*)&__utk_mgt_end__;
    #endif
    :
}
```

---

動的メモリ管理はコンフィギュレーションの **USE\_IMALLOC** の設定値で有効(1)/無効(0)を設定可能である。

デフォルトでは **USE\_IMALLOC** に 1 が設定されており、動的メモリ管理を行う。

動的メモリ管理を利用しない場合は **USE\_IMALLOC** に 0 を設定し、合わせてリンカスクリプト **cy8c6xxa\_cm4\_dual.ld** で μT-Kernel 管理領域のセクションを確保しないように変更する必要がある。

## 5. 割込みおよび例外

### 5.1. マイコンの割込みおよび例外

本マイコンには以下の例外が存在する。なお、OS 仕様上は例外、割込みをまとめて、割込みと称している。

表 5-1 例外番号と種別

例外番号	例外の種別	備考
1	リセット	
2	NMI(ノンマスカブル割込み)	
3	ハードフォールト	
4	メモリ管理フォールト	
5	バスフォールト	
6	用法フォールト	
7～10	予約	
11	SVCall(スーパーバイザコール)	OS で使用
12～13	予約	
14	PendSV	OS で使用
15	SysTick	OS で使用
16～183	IRQ 割込み#0～#167	OS の割込み管理機能で管理

### 5.2. ベクタテーブル

本マイコンでは、前述の各種例外に対応する例外ハンドラのアドレスを設定したベクタテーブルを有する。

本実装では、リセット時のベクタテーブルは `kernel/sysdepend/cpu/cy8c6/vector_tbl.c` に `vector_tbl` として定義される。

`kernel/sysdepend/cpu/cy8c6/vector_tbl.c` : ROM に配置されるベクタテーブル

```
void (* const vector_tbl[])() __attribute__((section(".vector"))) = {
    (void(*)()) (INITIAL_SP),          /* 0: Top of Stack */
    Reset_Handler,                    /* 1: Reset Handler */
    NMI_Handler,                      /* 2: NMI Handler */
    HardFault_Handler,                /* 3: Hard Fault Handler */
    MemManage_Handler,                /* 4: MPU Fault Handler */
    BusFault_Handler,                 /* 5: Bus Fault Handler */
    UsageFault_Handler,               /* 6: Usage Fault Handler */
    0,                                /* 7: Reserved */
    0,                                /* 8: Reserved */
    0,                                /* 9: Reserved */
    0,                                /* 10: Reserved */
}
```

```

    Svcall_Handler,          /* 11: Svcall */
    0,                      /* 12: Reserved */
    0,                      /* 13: Reserved */
    knl_dispatch_entry,     /* 14: Pend SV */
    knl_systim_inthdr,      /* 15: Systick */

    /* External Interrupts */
    Default_Handler,        /* IRQ 0 */
    Default_Handler,        /* IRQ 1 */
    Default_Handler,        /* IRQ 2 */
    :
    Default_Handler,        /* IRQ 167 */
};

```

---

ROM に配置されたベクタテーブルは(OS 稼働前の初期化処理において)RAM にコピーされ、以降は RAM にコピーされたベクタテーブルが使用される。

本実装では、RAM 上のベクタテーブルはリンカスクリプト `cy8c6xxa_cm4_dual.ld` を利用してセクション `.ramVectors` として領域を確保している。

---

#### `cy8c6xxa_cm4_dual.ld` : RAM 上のベクタテーブルのセクション指定

---

```

. ramVectors (NOLOAD) : ALIGN(8)
{
    __ram_vectors_start__ = .;
    KEEP(*(.ram_vectors))
    __ram_vectors_end__   = .;
} > ram

```

---

本実装では割込みベクタのコピーなどの初期化処理は `ModusToolbox` によって生成される初期化処理を利用している。このため、RAM 上のベクタテーブルの定義としては上記のセクションやシンボルを利用する必要がある。また、RAM 上のベクタテーブルのシンボルは `__ramVectors` として `.ramVectors` に配置されるように実装してある。

一方、μT-Kernel の割込管理機能では、`exchdr_tbl` をシンボルとして利用する必要がある。このため、`__ramVectors` を `exchdr_tbl` として参照できるようにするために、`kernel/sysdepend/cpu/cy8c6/interrupt.c` において以下のように実装を変更してある。

---

#### `kernel/sysdepend/cpu/cy8c6/interrupt.c` : RAM 上のベクタテーブルのシンボルを変換

---

```

/*
 * Exception handler table (RAM)
 */
//EXPORT UW exchdr_tbl[N_SYSVEC + N_INTVEC] __attribute__((section (".data_vector")));

IMPORT UW __ramVectors[];
#define exchdr_tbl __ramVectors

```

---

なお、他のマイコンの実装では `interrupt.c` は ARMv7-M 共通であり、ARMv7-M 用のディレクトリに配置されている。

一方、本実装では CY8C6 専用(`ModusToolbox` を使用)の実装にするため、`interrupt.c`

を CY8C6 用のディレクトリに用意し、そちらがビルド対象になるように調整する必要がある。

kernel/sysdepend/cpu/core/armv7m/interrupt.c	ARMv7-M 共通→ビルド対象外
↓	
kernel/sysdepend/cpu/cy8c6/interrupt.c	CY8C6 専用←ビルド対象

結果として ARMv7-M 共通の `interrupt.c` は本実装では不要となるが、他のマイコンでも共通に利用するファイルであるので削除することができない。そこで、ModusToolbox のターゲットファイル選択用のコンフィギュレーションファイル `.cyignore` において ARMv7-M 共通の `interrupt.c` を無視する様に設定してある。

### 5.3. 割り込み優先度とクリティカルセクション

#### 5.3.1. 割り込み優先度

Cortex-M4F は、割り込み優先度を 8bit(0~255)の 256 段階に設定できる(優先度の数字の小さい方が優先度は高い)。本実装では AIRCR(アプリケーション割り込みおよびリセット制御レジスタ)の PRIGROUP を 3 に設定している。つまり、横取り優先度が 4 ビット、サブ優先度が 4 ビットに設定される。

ただし、実際の割り込み優先度はハードウェアの実装に依存する。本マイコンでの実装は、割り込み優先度が 3bit になっているので、設定可能な割り込み優先度は 8 段階(0~7)である。

以上より、本マイコンの外部割り込みには優先度 0~7 を割り当て可能である。ただし、優先度 0 は OS からマスク不可のため、ユーザプログラムからの使用は許されない。また、優先度 1 と 7 は OS 内の割り込み処理で使用しているため、ユーザプログラムからの使用は許されない。結果的に、ユーザプログラムから使用可能な外部割り込みの優先度は 2~6 の 5 段階となる。

なお、次の例外は優先度 0 より高い優先度に固定されている。

- ・ リセット (優先度 : -3)
- ・ NMI (優先度 : -2)
- ・ ハードフォールト (優先度 : -1)

#### 5.3.2. 多重割り込み対応

本マイコンの割り込みコントローラ NVIC(Nested Vector Interrupt Controller)は、ハードウェアの機能として、多重割り込みに対応している。割り込みハンドラの実行中に、より優先度の高い割り込みが発生した場合は、実行中の割り込みハンドラに割り込んで優先度の高い割り込みハンドラが実行される。

### 5.3.3. クリティカルセクション

本実装では、クリティカルセクションは BASEPRI レジスタに最高外部割込み優先度 INTPRI\_MAX\_EXTINT\_PRI を設定することにより実現する。

INTPRI\_MAX\_EXTINT\_PRI は、本 OS が管理する割込みの最高の割込み優先度であり、include/sys/sysdepend/cpu/cy8c6/sysdef.h にて以下のように定義される。

include/sys/sysdepend/cpu/cy8c6/sysdef.h : 最高外部割込み優先度の定義

```
/*
 * Interrupt Priority Levels
 */
#define INTPRI_MAX_EXTINT_PRI      1      /* Highest Ext. interrupt level */
:
```

クリティカルセクション中は、INTPRI\_MAX\_EXTINT\_PRI 以下の優先度の割込みはマスクされる。

PRIMASK および FAULTMASK レジスタは使用しない。よって、リセット、NMI、ハードフォールトはクリティカルセクション中でもマスクされない。

### 5.4. OS 内部で使用する割込み

本 OS の内部で使用する割込みには、以下のように本マイコンの割込みまたは例外が割り当てられる。該当する割込みまたは例外は OS 以外で使用してはならない。

表 5-2 OS 内部で使用する割込み

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	1
ディスパッチ要求	14	PendSV	7
強制ディスパッチ要求	14	PendSV	7

各割込み・例外の優先度は include/sys/sysdepend/cpu/cy8c6/sysdef.h で以下のように定義される。

include/sys/sysdepend/cpu/cy8c6/sysdef.h : 各割込み・例外の優先度の定義

```
/*
 * Interrupt Priority Levels
 */
#define INTPRI_MAX_EXTINT_PRI      1      /* Highest Ext. interrupt level */
#define INTPRI_SVC                  0      /* SVCall */
#define INTPRI_SYSTICK              1      /* SysTick */
//                                2      /* TCPWM (Timer, Counter, PWM) */
//                                3      /* Sample Drivers (Ser, ADC, I2C) */
#define INTPRI_PENDSV              7      /* PendSV */
```

ただし、本実装では SVC には対応せず、システムコールは関数呼び出しのみである。よって SVC 割込みは使用しない。SVC 割込みは、将来の拡張に備えて優先度の定義のみが存在する。

システムタイマ割込みは最高優先度(1)、ディスパッチ要求は最低優先度(7)を使用している。

ユーザプログラムは、割込み優先度 2~6 を使用しなければならない。

ディスパッチ要求は、本実装ではクリティカルセクションの最後に、タスクのディスパッチが必要な場合に発行される。

また、強制ディスパッチ要求が、OS の起動時とタスクの終了時に発行される。本実装では、ディスパッチ要求と強制ディスパッチ要求は同一の割込み(PendSV)を使用する。

## 5.5. μT-Kernel/OS の割込み管理機能

μT-Kernel/OS の割込み管理機能は、割込みハンドラの管理を行う。

本実装では、対象とする割込み(割込みハンドラが定義可能な割込み)は外部割込み(IRQ0~167)のみとし、その他の例外については対応しない(その他の例外については「5.8. その他の例外」を参照のこと)。

### 5.5.1. 割込み番号

OS の割込み管理機能が使用する割込み番号は、マイコンの外部割込みの番号と同一とする。例えば、IRQ0 の割込み番号は 0 である。

### 5.5.2. 割込みハンドラ属性

Cortex-M では、C 言語の関数による割込みハンドラの記述がハードウェアの仕様として可能となっている。そのため、TA\_HLNG 属性と TA\_ASM 属性のハンドラで大きな差異はなくなっている。

TA\_HLNG 属性の割込みハンドラは、割込みの発生後、OS の割込み処理ルーチンを経由して呼び出される。OS の割込み処理ルーチンでは以下の処理が実行される。

#### (1) タスク独立部の設定

処理開始時にシステム変数 knl\_taskindp をインクリメントし、終了時にデクリメントする。本変数が 1 以上のとき、タスク独立部であることを示す。

#### (2) 割込みハンドラの実行

割込み PSR(IPAR)レジスタの値を参照し、テーブル knl\_hll\_inthdr に登録されている割込みハンドラを実行する。

TA\_ASM 属性の割込みハンドラは、マイコンの割込みベクタテーブルに直接登録される。よって、割込み発生時には、OS の処理を介さずに直接ハンドラが実行される。よって、

TA\_ASM 属性の割り込みハンドラからは、原則として API などの OS 機能の使用が禁止される。ただし、上記の OS 割り込み処理ルーチンで実行されるのと同様の処理を行うことにより、OS 機能の使用が可能となる。

### 5.5.3. デフォルトハンドラ

割り込みハンドラが未登録の状態で、割り込みが発生した場合はデフォルトハンドラが実行される。デフォルトハンドラは、`kernel/sysdepend/cpu/core/armv7m/exc_hdr.c` の `Default_Handler` 関数として実装されている。

デフォルトハンドラは、コンフィギュレーション `USE_EXCEPTION_DBG_MSG` を有効にすることにより、デバッグ情報を出力する(初期設定は有効である)。

デバッグ情報は、T-Monitor 互換ライブラリによるコンソールに出力される(「9.2.2. コンソール入出力」参照)。

必要に応じてユーザがデフォルトハンドラを記述することにより、未定義割り込み発生時の処理を行うことができる。デフォルトハンドラは `weak` 属性を付けて宣言してあるので、ユーザが同名の関数を作成してリンクすることにより上書きすることができる。

## 5.6. μT-Kernel/SM の割り込み管理機能

μT-Kernel/SM の割り込み管理機能は、CPU の割り込み管理機能および割り込みコントローラ(NVIC)の制御を行う。

### 5.6.1. 割り込みの優先度

割り込みの優先度は、2~6 を使用可能である。優先度 1 および 7 は OS が使用しているため、原則として指定してはならない。

### 5.6.2. CPU 割り込み制御

CPU 割り込み制御は、マイコンの BASEPRI レジスタのみを制御して実現する。PRIMASK および FAULTMASK レジスタは使用しない。

#### (1) CPU 割り込みの禁止(DI)

CPU 割り込みの禁止(DI)は、BASEPRI レジスタに最高外部割り込み優先度 `INTPRI_MAX_EXTINT_PRI` を設定し、それ以下の優先度の割り込みを禁止する。

#### (2) CPU 割り込みの許可(EI)

割り込みの許可(EI)は、BASEPRI レジスタの値を DI 実行前に戻す。

#### (3) CPU 内割り込みマスクレベルの設定(SetCpuIntLevel)

CPU 内割り込みマスクレベルの設定(SetCpuIntLevel)は、BASEPRI レジスタを指定

した割込みマスクレベルに設定する。

指定したマスクレベルより低い優先度の割込みはマスクされる。マスク可能な割込みの優先度は 1~7 である。よって、0 を指定した場合はすべての割込みがマスクされる。また、7 を指定した場合はすべての割込みが許可される。

μT-Kernel 3.0 仕様に基づき、INTLEVEL\_DI を指定した場合はすべての割込みがマスクされる。また、INTLEVEL\_EI を指定した場合はすべての割込みが許可される。

#### (4) CPU 内割込みマスクレベルの参照(GetCpuIntLevel)

CPU 内割込みマスクレベルの取得(GetCpuIntLevel)は、BASEPRI レジスタの設定値を参照し、設定されている割込みマスクレベルを返す。

なお、すべての割込みがマスクされていない場合(すべての優先度の割込みを許可)は、INTLEVEL\_EI の値を返すものとする。

### 5.6.3. 割込みコントローラ制御

マイコン内蔵の割込みコントローラ(NVIC)の制御を行う。

各 API の実装を以下に記す。

#### (1) 割込みコントローラの割込み許可(EnableInt)

割込みコントローラ(NVIC)の割込みイネーブルセットレジスタ(ISER)を設定し、指定された割込みを許可する。

同時に割込み優先度レジスタ(IPR)に指定された割込み優先度を設定する。

#### (2) 割込みコントローラの割込み禁止(DisableInt)

割込みコントローラ(NVIC)の割込みイネーブルクリアレジスタ(ICER)を設定し、指定された割込みを禁止する。

#### (3) 割込み発生のカリア(ClearInt)

割込みコントローラ(NVIC)の割込み保留クリアレジスタ(ICPR)を設定し、指定された割込みが保留されていればクリアする。

#### (4) 割込みコントローラの EOI 発行(EndOfInt)

本マイコンでは EOI の発行は不要である。よって、EOI 発行(EndOfInt)は何も実行しない関数として定義されている。

#### (5) 割込み発生のカリア(CheckInt)

割込み発生のカリア(CheckInt)は、割込みコントローラ(NVIC)の割込み保留クリアレジスタ(ICPR)を参照することにより実現する。



(6) 割込みモード設定(SetIntMode)

本実装では非対応である。

割込みモード設定(SetIntMode)は何も実行しない関数として定義されている。

(7) 割込みコントローラの割込みマスクレベル設定(SetCtrlIntLevel)

割込みコントローラ(NVIC)に本機能はないため、未実装である。

(8) 割込みコントローラの割込みマスクレベル取得(GetCtrlIntLevel)

割込みコントローラ(NVIC)に本機能はないため、未実装である。

## 5.7. OS 管理外割込み

最高外部割込み優先度 `INTPRI_MAX_EXTINT_PRI` より優先度の高い外部割込みは、OS 管理外割込みとなる。

管理外割込みは OS 自体の動作よりも優先して実行される。よって、OS から制御することはできない。また、管理外割込みの中で OS の API などの機能を使用することも原則としてできない。

本実装では `INTPRI_MAX_EXTINT_PRI` は優先度 1 と定義されている。よって、優先度 0 以上の例外が管理外割込みとなる。マイコンのデフォルトの設定では、リセット、NMI、ハードフォールト、メモリ管理の例外がこれに該当する。

## 5.8. その他の例外

外部割込み以外の例外は、本実装では OS は管理しない。

これらの例外には、暫定的な例外ハンドラを定義している。暫定的な例外ハンドラは、`kernel/sysdepend/cpu/core/armv7m/exc_hdr.c` で以下の関数として実装されている。

表 5-3 例外ハンドラ

例外番号	例外の種別	関数名
2	NMI(ノンマスカブル割込み)	NMI_Handler
3	ハードフォールト	HardFault_Handler
4	メモリ管理	MemManage_Handler
5	バスフォールト	BusFault_Handler
6	用法フォールト	UsageFault_Handler
11	SVC(スーパーバイザコール)	Svcall_Handler
12	デバッグモニタ	DebugMon_Handler

暫定的な例外ハンドラは、プロファイル `USE_EXCEPTION_DBG_MSG` を有効にすることにより、デバッグ用の情報を出力する(初期設定は有効である)。

必要に応じてユーザが例外ハンドラを記述することにより、各例外に応じた処理を行う

ことができる。暫定的な例外ハンドラは **weak** 属性を付けて宣言してあるので、ユーザが同名の関数を作成してリンクすることにより上書きすることができる。

各例外ハンドラはベクタテーブルに登録されているので、例外発生時に **OS** を介さず直接実行される。例外の優先度が、優先度 **0** 以上の場合、その例外は **OS** 管理外例外となる。それ以外の優先度の場合は、**ASM** 属性の割込みハンドラと同等である。

❶ **CY8C6** では例外番号 **12** は「予約」であるため、デバッグモニタは非対応である。

## 6. 起動および終了処理

### 6.1. リセット処理

リセット処理は、マイコンのリセットベクタに登録され、マイコンのリセット時に実行される。

本実装では、リセット後の起動処理の一部を ModusToolbox が生成する初期化ルーチンを利用して実現している。このため、ARMv7-M に共通の処理ではなく、CY8C6 に依存した処理として実装する必要がある。

具体的には、CY8C6 依存のディレクトリに追加した `reset_hdr.c` 内の `Reset_Handler` 関数をリセット処理において呼び出す実装になっている。

ARMv7-M 共通	<code>kernel/sysdepend/cpu/core/armv7m/reset_hdl.c</code>
	↓
CY8C6 依存	<code>kernel/sysdepend/cpu/cy8c6/reset_hdr.c</code>

関数名は同じなので、マイコンのリセットベクタに登録するエントリに変更はない。

リセットベクタの実装については「5.2. ベクタテーブル」を参照のこと。

なお、ARMv7-M 共通の `reset_hdl.c` は本実装では利用しないが、共通のコードであるために削除することができない。そこで、ModusToolbox の機能を利用してビルド対象外に設定してある。

本実装での起動時の処理のフローは以下のとおりである。

`Reset_Handler()` : `kernel/sysdepend/cpu/cy8c6/reset_hdr.c`

```

├─→ knl_startup_hw() : kernel/sysdepend/cy8ckit_cy8c6/hw_setting.c
│   └─ 割込み禁止 (ModusToolbox のリセット処理に合わせて CPSID 1 を実行)
├─→ startup_clock() : kernel/sysdepend/cpu/cy8c6/cpu_clock.c
│   └─ ※空関数(クロックの初期化は SystemInit() で実施)
├─ クロック供給対象モジュール選択(設定用データ modclk_tbl[] は空)
├─ I/O ピンの機能選択(設定用データ pinfunc_tbl[] は空)
├─ セクションの初期化
│   └─ ベクタテーブルを RAM にコピー      Load Vector Table from ROM to RAM
│   └─ VTOR の設定                        Set Vector Table offset to SRAM
│   └─ DATA セクションの初期化          Load .data to ram
│   └─ BSS セクションの初期化            Initialize .bss
├─→ SystemInit()
│   └─ Cy_PDL_Init(CY_DEVICE_CFG)         ModusToolbox : 初期化処理ルーチン
│   └─ Cy_SystemInit()                   周辺デバイス用の低レベルドライバの定義
│   └─ SystemCoreClockUpdate()           PSoC Creator が生成する初期化関数
│   └─ Cy_IPC_Sema_Init(CY_IPC_CHAN_SEMA, 0u1, NULL)   コアクロックの取得
│   └─ Cy_IPC_Pipe_Config(systemIpcPipeEpArray)
│   └─ Cy_IPC_Pipe_Init(&systemIpcPipeConfigCm4)
│       └─ ※Cy_IPC_*() 関数では、プロセッサ間通信用の機能を初期化する。
├─ Imalloc() 用の領域を初期化             Set System memory area
├─ 割込み関連レジスタの設定
│   └─ 割込み(例外)優先度の設定         Configure exception priorities
│   └─ STIR 使用許可                     Allow user to access SCB_STIR

```

```

├→ Cy_SystemInitFpuEnable()          ModusToolbox : FPU の初期化
├→ cybsp_init()                      ModusToolbox : ペリフェラルの初期化
└→ main() : kernel/sysinit/sysinit.c

```

Reset\_Handler 関数の処理手順を以下に示す。

(1) ハードウェア初期化(knl\_startup\_hw)

リセット時の必要最小限のハードウェアの初期化を行う。

通常はクロックの設定と IO ピンの設定を本関数において行うが、本実装では何も設定していない。本実装では ModusToolbox の機能を利用して設定する。

(2) 例外ベクタテーブルの移動

例外ベクタテーブルを ROM から RAM にコピーし、VTOR をコピー先(RAM)のアドレスに変更する。これにより例外ベクタテーブルを変更可能にする。

ただし、コンフィギュレーションで USE\_STATIC\_IVT に 1 が指定された場合は、例外ベクタテーブルの移動は行われしない。この場合、実行中の OS からの割込みハンドラの登録はできなくなる(初期値では USE\_STATIC\_IVT は 0 に設定されている)。

(3) 変数領域の初期化

C 言語のグローバル変数領域(.data セクション)の初期化を行い、初期値付き変数の設定および、その他の変数のゼロクリアを行う。

コンフィギュレーションで USE\_NOINIT に 1 を指定した場合は、.noinit セクションの 0 クリアを省略し、.bss セクションのみ 0 クリアする(初期値では USE\_NOINIT は 0 に設定されている)。

本実装では、.noinit セクション、.bss セクションの順に並んでいることを前提として実装しているので、リンカスクリプトを調整する場合は注意が必要である。メモリマップについては「4.3. OS のメモリマップ」を参照のこと。

(4) ModusToolbox によるシステム初期化ルーチンの実行

ModusToolbox が提供するシステム初期化ルーチン SystemInit() を実行する。

SystemInit() は C 言語で記述されているので、予め「(3) 変数領域の初期化」が実行されている必要がある。

(5) システムメモリ領域の確保

システムメモリ領域(μT-Kernel 管理領域)の確保を行う。

ただし、コンフィギュレーション USE\_IMALLOC が指定されていない場合、本処理は実行されない。詳細については「4.5. μT-Kernel 管理領域」を参照のこと。

(6) 割込みコントローラ(NVIC)の設定

割込みコントローラの基本設定を行う。

## (7) ModusToolbox による FPU 初期化ルーチンの実行

ModusToolbox が提供する FPU 初期化ルーチン `Cy_SystemInitFpuEnable()` を実行する。

## (8) ModusToolbox による CPU とボードのデバイス初期化ルーチンの実行

ModusToolbox が提供する CPU とボードの周辺デバイス初期化ルーチン `cybsp_init()` を実行する。

`cybsp_init()` では、Device Configurator によって指定されたデバイスの初期化処理を実行する。

## (9) OS 初期化処理の実行

OS の初期化処理 `main()` を実行する。

`main()` の中で μT-Kernel を実行してタスクに処理が移るので、`main()` 関数からは戻ってこない。

## 6.2. ハードウェアの初期化および終了処理

OS の起動時にはマイコンおよび周辺デバイスの初期化処理が必要となる。

μT-Kernel 3.0 では、以下の関数によってマイコンおよび周辺デバイスの初期化、終了処理が実行されることになっている。

表 6-1 ファイル : `kernel/sysdepend/cy8ckit_cy8c6/hw_setting.c`

関数名	内容
<code>kn1_startup_hw</code>	ハードウェアのリセット リセット時の必要最小限のハードウェアの初期化を行う。 本実装では、初期化処理に ModusToolbox を利用しているので、この関数は使用していない。処理のひな型として残してある。
<code>kn1_shutdown_hw</code>	ハードウェアの停止 周辺デバイスをすべて終了し、マイコンを終了状態とする。 本実装では、割込み禁止状態で無限ループとしている。
<code>kn1_restart_hw</code>	ハードウェアの再起動 周辺デバイスおよびマイコンの再起動を行う。 本実装ではデバイスの再起動には対応していない。 処理のひな型のみを記述している。

開発者は必要に応じて各関数の内容を変更することが可能だが、これらの関数は OS の共通部からも呼び出されるため、関数の呼び出し形式を変更することはできない。

各関数の仕様については、「μT-Kernel 3.0 共通実装仕様書」も参照のこと。

本実装ではハードウェアの初期化処理を ModusToolbox が提供する関数で行っている。このため、新しいデバイスを利用する場合に開発者がデバイスの初期化処理を個別に実装

する必要はない。新しいデバイスを利用する場合は、ModusToolbox の Device Configurator(DC)を用いてデバイスの構成(入出力端子、クロック、分周器など)を指定し、Device Configurator が生成した初期化関数を呼び出してデバイスを初期化する。

Device Configurator で指定した情報のうち、クロックなどの共通部分の初期化は「6.1. リセット処理」で説明した `cybsp_init()` で実行されるが、デバイスごとの初期化処理は個別に初期化関数を呼び出す必要がある。

例えば、本実装では T-Monitor 互換ライブラリがコンソールとして利用するシリアルポートとして、SCB5(Serial Communication Block 5)を指定している。SCB5 の構成は Device Configurator で設定しており、Device Configurator が自動生成した関数を `libtm_init()` で呼び出して SCB5 の初期化を行っている。

ModusToolbox(の Device Configurator が生成した関数)を利用した SCB5 の初期化処理の概略は以下のとおりである。

```
Reset_Handler()
├→ knl_startup_hw()
├─ セクションの初期化
├→ SystemInit()
├─ Imalloc() 用の領域を初期化
├─ 割込み関連レジスタの設定
├→ Cy_SystemInitFpuEnable()
│   ※ここまでの処理の概要については「6.1. リセット処理」を参照のこと
├→ cybsp_init()                               ModusToolbox: ペリフェラルの初期化
│   :
│   └→ init_cycfg_all()                       DC で自動生成されたコードを実行
│       └→ init_cycfg_system()                システムレベルの設定
│       └→ init_cycfg_clocks()                クロック関連の設定
│       └→ init_cycfg_routing()               AMUXBUS の設定
│       └→ init_cycfg_peripherals()           分周器の周辺デバイスへの割当など
│       └→ init_cycfg_pins()                 端子の周辺デバイスへの割当など
│       :
│       ※詳細については ModusToolbox の資料を参照のこと
├→ main()
│   :
│   └→ libtm_init()                          T-Monitor 互換ライブラリの初期化
│       └→ tm_com_init()                     通信ポートの初期化
│           ※SCB5 に割り当てた分周器を設定してボーレートを変更する。
│           └→ Cy_SysClk_PeriphDisableDivider() 分周器を一時停止
│           └→ Cy_SysClk_PeriphSetDivider()      分周器の設定
│           └→ Cy_SysClk_PeriphEnableDivider()   分周器を再開
│           :
│           ※SCB5 を初期化する。
│           └→ Cy_SCB_UART_Init()               SCB5 のパラメータなどを設定
│           └→ Cy_SCB_UART_Enable()            SCB5 を実行状態に設定
│           :
│           :
```

Device Configurator では SCB5 のボーレートも設定している。このため、`tm_com_init()` で分周器を設定する必要はないが、`tm_com.c` で定義しているマクロ `UART_BAUD` に従ってボーレートを変更できるようにするために分周器の設定処理を追加してある。

なお、`tm_com_init()` で分周器を設定しない場合は、Device Configurator で指定したボーレートで動作する(本実装では 115200bps となる)。

- ① UART\_BAUD によるボーレートの設定については「9.2.2. コンソール入出力」を参照のこと。

### 6.3. デバイスドライバの実行および終了

OS の起動および終了に際して、以下の関数にてデバイスドライバの登録、実行、終了処理を行う。関数の仕様については、共通実装仕様書も参照のこと。

表 6-2 ファイル : `kernel/sysdepend/cy8ckit_cy8c6/devinit.c`

関数名	内容
<code>knl_init_device</code>	デバイスの初期化 デバイスドライバの登録に先立ち、必要なハードウェアの初期化を行う。本関数の実行時は、OS の初期化中のため、原則 OS の機能は利用できない。 本実装では、初期化処理に <code>ModusToolbox</code> を利用しているので、この関数は使用していない。処理のひな型として残してある。
<code>knl_start_device</code>	デバイスの実行 デバイスドライバの登録、実行を行う。本関数は、初期タスクのコンテキストで実行され、OS の機能を利用できる。
<code>knl_finish_device</code>	デバイスの終了 デバイスドライバを終了する。本関数は、初期タスクのコンテキストで実行され、OS の機能を利用できる。

コンフィギュレーション `USE_SDEV_DRV` と個別マクロ定義 `DEVICE_USE_*` が有効(1)の場合、以下のサンプルデバイスドライバが `knl_start_device()` で登録される。

表 6-3 サンプルドライバ

デバイス名	機能	個別マクロ定義
<code>serb</code>	シリアル通信 (SCB1, SCB2)	<code>DEVICE_USE_SER</code>
<code>iica</code>	I2C 通信	<code>DEVICE_USE_IIC</code>
<code>adca</code>	A/D コンバータ	<code>DEVICE_USE_ADC</code>

- ① 本実装ではシリアル通信にのみ対応している。I2C、ADC には対応していない。

`USE_SDEV_DRV` または `DEVICE_USE_*` が無効(0)の場合、デバイスドライバの登録は行われない(初期値は無効(0))。

ユーザが使用するデバイスドライバを変更する場合は、必要に応じて上記の関数の内容を変更する必要がある。ただし、これらの関数は OS の共通部からも呼ばれるため、関数の呼び出し形式を変更してはならない。

## 7. タスク

### 7.1. タスク属性

タスク属性のハードウェア依存仕様を以下に示す。

表 7-1 タスク属性

属性	可否	説明
TA_COP0	○	FPU(TA_FPU と同じ)
TA_COP1	×	対応無し
TA_COP2	×	対応無し
TA_COP3	×	対応無し
TA_FPU	○	FPU(TA_COP0 と同じ)

### 7.2. タスクの処理ルーチン

タスクの処理ルーチンの実行開始時の各レジスタの状態は以下である。これ以外のレジスタの値は不定である。

表 7-2 タスク処理ルーチン実行開始時のレジスタ地

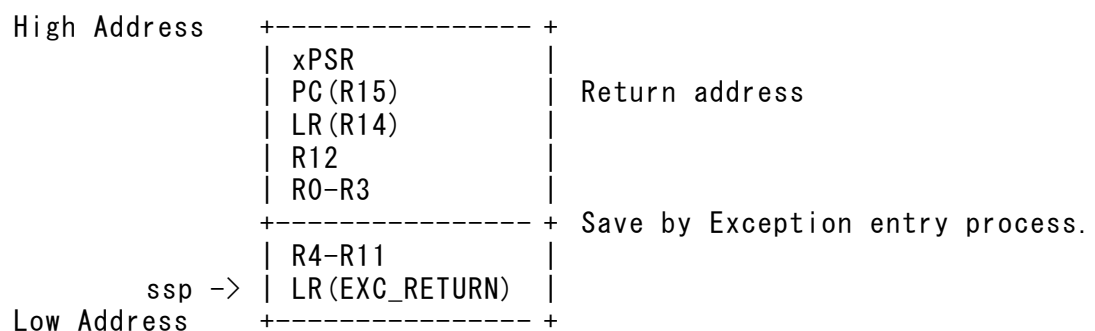
レジスタ	値	補足
PRIMASK	0	割込み許可
R0	第一引数 <code>stacd</code>	タスク起動コード
R1	第二引数 <code>*exinf</code>	タスク拡張情報
R13(MSP)	タスクスタックの先頭アドレス	

### 7.3. タスク・コンテキスト情報

スタック上に保存されるタスクのコンテキスト情報を以下に示す。

#### (1) TA\_FPU 属性以外のタスク

xPSR から R0 までのレジスタの値はディスパッチ時の例外により保存される。R4 から LR(EXC\_RETURN 値)のレジスタの値はディスパッチャにより保存される。



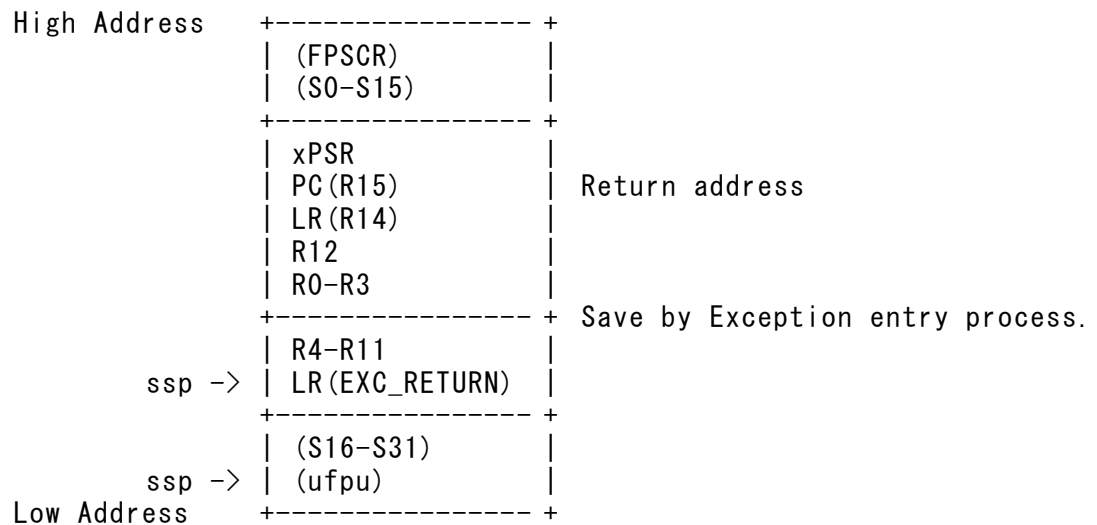


## (2) TA\_FPU 属性のタスク

TA\_FPU 属性以外のタスクのコンテキスト情報に加えて、FPU のレジスタの値が保存される。

FPSRC および S0～S15 までのレジスタの値はディスパッチ時の例外により保存される。S16～S32 のレジスタの値および ufpu はディスパッチャにより保存される。

ufpu は EXC\_RETURN の bit4 以外をマスクした値であり、タスクのコンテキストでの FPU 命令の実行を示す(CONTROL レジスタの FPCA ビットの反転)。



## 8. 時間管理機能

### 8.1. システムタイマ

本実装では、マイコン内蔵の `SysTick` タイマをシステムタイマとして使用する。  
システムタイマのティック時間は、1 ミリ秒から 50 ミリ秒の間で設定できる。  
ティック時間の標準の設定値は 10 ミリ秒である。

### 8.2. タイムイベントハンドラ

タイムイベントハンドラの実行中の割込みマスクレベルは、タイムイベントハンドラ割込みレベル `TIMER_INTLEVEL` に設定される。`TIMER_INTLEVEL` は、以下のファイルで定義される。

本実装では初期値は以下のように 0(すべての割込みを許可) が設定されている。

```
include/sys/sysdepend/cpu/cy8c6/sysdef.h : タイムイベントハンドラの割込レベル
/*
 * Time-event handler interrupt level
 */
#define TIMER_INTLEVEL          0
```

### 8.3. 物理タイマ機能

本実装では、この機能に対応していない。

## 9. その他の実装仕様

### 9.1. FPU 関連

#### 9.1.1. FPU の初期設定

OS 起動時にコンフィギュレーション `USE_FPU` が指定されている場合、FPU を有効化するとともに、`Lazystack` を有効にする。

具体的には以下のように FPU のレジスタが設定される。

表 9-1 FPU レジスタの設定値

レジスタ	設定値	意味
CPACR	0x00F00000	CP10 および CP11 のアクセス許可
FPCCR	0xC0000000	ASPEN=1 自動状態保持を有効 LSPEN=1 レイジーな自動状態保持を有効

`Lazystack` により、タスクにて FPU を使用中に例外が発生した場合、FPU レジスタ (S0～S15 および FPSCR) の退避領域がスタックに確保される。その後、例外処理中に FPU が使用されたときに実際のレジスタの値が保存される。

本実装では、タスクのディスパッチの際の FPU レジスタの保存に本機能を使用している。`Lazystack` を無効にした場合の動作は保証しない。

#### 9.1.2. FPU 関連 API

コンフィギュレーション `USE_FPU` が指定されている場合、FPU 関連の API (`tk_set_cop`、`tk_get_cop`) が使用可能となる。本 API を用いて実行中のタスクのコンテキストの FPU レジスタ値を操作できる。

FPU のレジスタは以下のように定義される。

`include/tk/sysdepend/cpu/core/armv7m/cpudef.h` : FPU 用レジスタの定義

---

```
#if NUM_COPROCESSOR > 0
/*
 * Co-processor register
 */
typedef struct t_copregs {
    VW    s[32];    /* FPU General purpose register S0-S31 */
    UW    fpscr;    /* Floating-point Status and Control Register */
} T_COPREGS;
#endif /* NUM_COPROCESSOR */
```

---

API によって操作されるのは、実際にはスタック上に退避されたレジスタの値である。

よって、実行中のタスクに操作することはできない (OS 仕様上、自タスクへの操作、またはタスク独立部からの API 呼出しは禁止されている)。

### 9.1.3. FPU 使用の際の注意点

ユーザのプログラムにおいて FPU を使用する場合(プログラムコード中に FPU 命令が含まれる場合)は以下の点に注意する必要がある。

- タスク中にて FPU を使用する場合は、タスク属性に TA\_FPU 属性を指定しなければならない。TA\_FPU 属性のタスクは、ディスパッチの際に、スタックに FPU レジスタの値を退避する。  
TA\_FPU 属性が指定されていないタスクでは、ディスパッチの際に FPU レジスタの値を退避しない。このため、TA\_FPU 属性が指定されていないタスクに FPU 命令が含まれていた場合は FPU レジスタの内容が破壊される可能性がある。  
なお、TA\_FPU 属性が指定されたタスクは他の属性のタスクより、ディスパッチ時のスタック使用量が退避する FPU レジスタの分だけ増加する。  
(タスクのスタックについては「7.3. タスク・コンテキスト情報」を参照のこと)
- 割込み発生時に FPU が使用されている場合は、マイコンの機能として FPU レジスタ(S0～S15 および FPSCR)がスタック上に退避される。退避していない FPU レジスタ(S16～S31)が、割込みハンドラ中において使用される場合は、プログラム中で対応しなければならない。  
なお、割込み発生時のスタック使用量は退避する FPU レジスタの分だけ増加する。
- 本実装では OS の API は関数呼び出しである。よって API はタスクのコンテキスト(スタック)で実行されるため、特に FPU の使用を考慮する必要はない。なお、API の呼び出しが例外など他の実装の場合には FPU 使用に際し考慮する必要がある場合もありうる。

以上より、FPU を使用したプログラムを作成する際は、必ず TA\_FPU 属性が指定されていないタスクにおける FPU の使用を禁止しなければならない(タスクのコード中に FPU 命令があってはならない)。

ただし、GCC などの C 言語処理系において、プログラムの部分的に FPU を使用、未使用を指定することは難しい。もっとも簡単な実現方法は、すべてのタスクにおいて TA\_FPU 属性を指定することである。

## 9.2. T-Monitor 互換ライブラリ

### 9.2.1. ライブラリ初期化

T-Monitor 互換ライブラリを使用するにあたって、ライブラリの初期化関数 `libtm_init()` を実行する必要がある。コンフィグレーション `USE_TMONITOR` が有効(1)の場合、初期化関数 `libtm_init()` は OS の起動処理関数 `main()` の中で実行される。

kernel/sysinit/sysinit.c : main() 関数内での libtm\_init() 呼び出し部分

```

/*
 * Start micro T-Kernel
 *   Initialize sequence before micro T-Kernel start.
 *   Perform preparation necessary to start micro T-Kernel.
 */
EXPORT INT main( void )
{
    ER        ercd;

    DISABLE_INTERRUPT;

#ifdef USE_TMONITOR
    /* Initialize T-Monitor Compatible Library */
    libtm_init();
#endif
    :

```

### 9.2.2. 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通信パラメータは以下の通りである。

表 9-2 コンソールの通信パラメータ

項目	内容
デバイス	内蔵 UART (SCB5)
通信速度	115,200 bps
データ形式	データ長           8bit ストップビット   1bit パリティビット   なし

CY8CKIT-062S2-43012 では、ターゲットマイコン(CY8C624ABZI-S2D44A0)の SCB5 (Serial Communication Block 5)を USB 経由で PC と接続することができる。そこで、本実装では SCB5 を T-Monitor のコンソールとして利用している。

開発時にはターゲットボードの KitProg3 USB connector(J6)を PC と接続する(下図)。



図 4 コンソール用 UART(SCB5)の接続先(J6)

コネクタ J6 を USB ケーブルで PC に接続すると KitProg3 が USB-UART ブリッジとして動作し、PC に仮想 COM ポートが作成される (COM ポートの番号は環境によって異なるので適宜確認すること。Windows であればデバイスマネージャーで確認できる)。

PC 側では作成された COM ポートを指定して通信ソフトを起動し、上記のパラメータ (通信速度、データ形式) を指定することで μT-Kernel のコンソールとして利用できる。

各通信パラメータは、ModusToolbox の Device Configurator でパラメータを調整することができる (下図)。

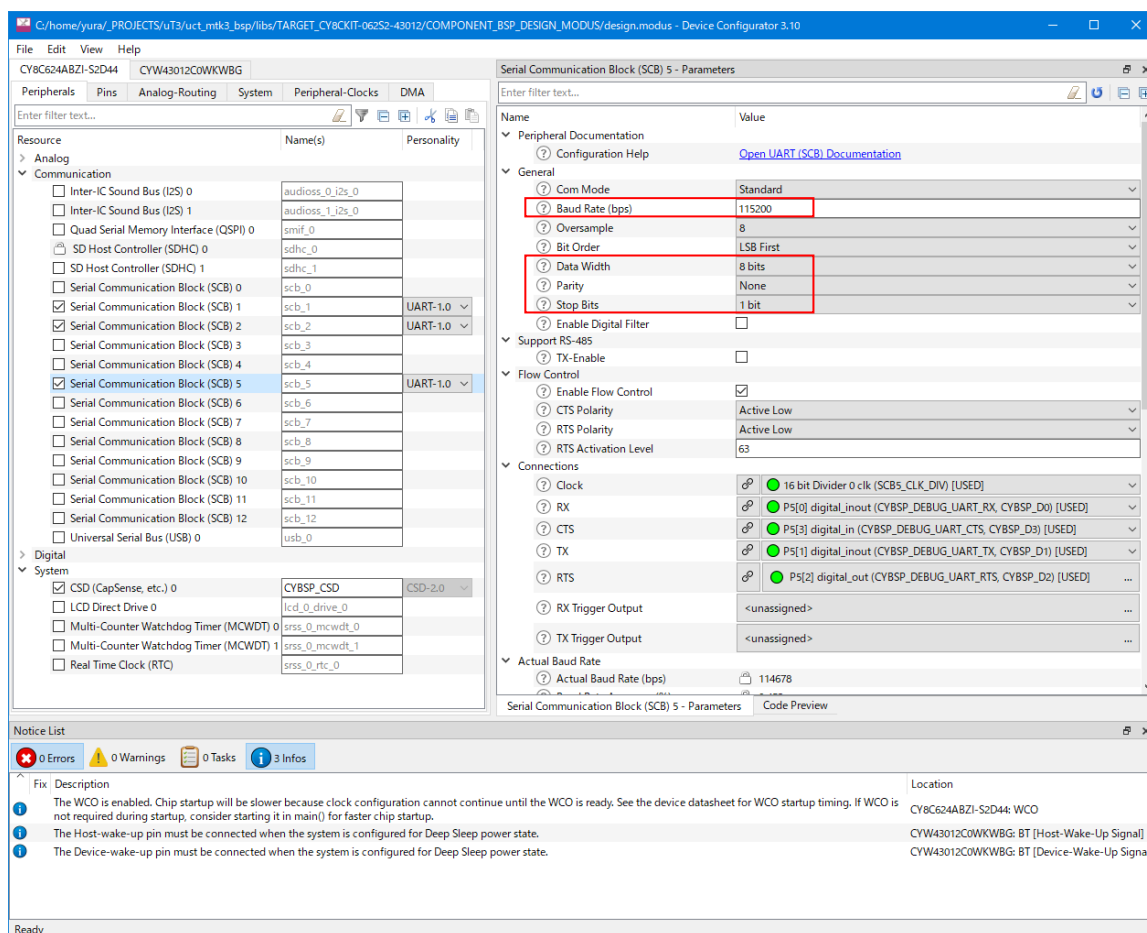


図 5 SCB5 の通信パラメータの設定

通信速度のみ `tm_com.c` のマクロ `UART_BAUD` の定義で指定可能な実装にしてある。

`lib/libtm/sysdepend/cy8ckit_cy8c6/tm_com.c` : 通信速度の指定

```
/* Communication speed */
#define UART_BAUD          (115200)          /* 115200 bps */
```

以上