μT-Kernel 3.0
実装仕様書（EVK-XMC7200）

Rev 3.00.01

December, 2023

Technology
Ubiquitous Computing Technology Corporation

目次

1. はじめに

本品は、ユーシーテクノロジ(株)が開発したμT-Kernel 3.0 である。

本品は、トロン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 KIT_XMC72_EVK(以下、EVK-XMC7200)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 EVK-XMC7200 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(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 EVK-XMC7200 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。
<TARGET>	ターゲットボード用のディレクトリ名を表す。
<CPU>	CPU 用のディレクトリ名を表す。
<CORE>	CPU コア用のディレクトリ名を表す。

2. 概要

本書では、μT-Kernel 3.0 BSP に含まれる EVK-XMC7200 向けの実装仕様について説明する。

- 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	XMC7200D-E272K8384 (ARM Cortex-M7, 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)	KIT_XMC72_EVK (EVK-XMC7200)	Infineon Technologies AG

- 本実装の最新版は、ユーシーテクノロジ(株)の GitHub リポジトリにて公開している。
  https://github.com/UCTechnology/mtk3_bsp
- 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

2.2. ターゲット名

EVK-XMC7200 のターゲット名および識別名は以下とする。

表 2-2 EVK-XMC7200 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	_EVK_XMC7200_	EVK-XMC7200
対象システム	EVK_XMC7200	EVK-XMC7200
対象 CPU	CPU_XMC7200	EVK-XMC7200
対象 CPU アーキテクチャ	CPU_CORE_ARMV7M	ARMv7-M アーキテクチャ
	CPU_CORE_ACM7	ARM Cortex-M7 コア

ターゲット名_EVK_XMC7200_は、Makefile においてシンボルとして定義する。
他の識別名は以下のファイルで定義される。

```
include/sys/sysdepend/evk_xmc7200/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 関連ドキュメント一覧

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
	μT-Kernel 3.0 共通実装仕様書(Ver.1.00.8)	トロンフォーラム
	μT-Kernel 3.0 共通実装＆構成仕様書 (Rev 3.00.06)	ユーシーテクノロジ(株)
T-Monitor	T-Monitor 仕様書(Ver.1.00.01)	トロンフォーラム
デバイスドライバ	μT-Kernel 3.0 デバイスドライバ説明書 (Ver.1.00.06)	トロンフォーラム
	μT-Kernel 3.0 共通デバイスドライバ説明書 (Rev 3.00.05)	ユーシーテクノロジ(株)
実装仕様書	μT-Kernel 3.0 実装仕様書(EVK-XMC7200)	ユーシーテクノロジ(株)
構築手順書	μT-Kernel 3.0 構築手順書(EVK-XMC7200)	ユーシーテクノロジ(株)
ターゲットボード	取扱説明書 KIT_XMC72_EVK XMC7200 evaluation kit guide	Infineon Technologies AG
	ユーザーガイド KitProg3 user guide	
	回路図 KIT_XMC72_EVK_XMC7200-evaluation-kit_schematic-PCBDesignData	
搭載マイコン	データシート XMC7000 microcontroller	
	テクニカルリファレンスマニュアル XMC7000 MCU family architecture Technical reference manual	
	テクニカルリファレンスマニュアル XMC7200 MCU registers Technical reference manual	
	ModusToolbox ユーザーガイド ModusToolbox user guide	

トロンフォーラムが発行するドキュメントは、トロンフォーラムの Web ページ、または GitHub で公開する μT-Kernel 3.0 のソースコードに含まれている。

https://www.tron.org/ja/specifications/

https://github.com/tron-forum/mtkernel_3

ℹ ユーシーテクノロジ(株)が発行するドキュメントは、ユーシーテクノロジ(株)の GitHub で公開するμT-Kernel 3.0 のソースコードに含まれている。
https://github.com/UCTechnology/mtk3_bsp

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。

ℹ ModusToolbox の詳細については、「ModusToolbox ユーザーガイド」を参照のこと。

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、μT-Kernel 3.0 をインポートして利用する。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/                    ModusToolbox 用のワークスペース
       ├─ mtb_shared/                  ModusToolbox 用の共通モジュール
       ├─ <APP-PRJ#1>/                 アプリケーションプロジェクト#1
       :          :                         :
       └─ mtkernel_3/                  μT-Kernel 3.0 のプロジェクト
            ├─ app_program/            標準的なサンプルアプリ
            ├─ app_serial/             シリアルドライバ用のサンプルアプリ
            ├─ config/                 コンフィギュレーション
            ├─ device/                 サンプルドライバ
            ├─ include/                各種定義ファイル
            ├─ kernel/                 μT-Kernel 3.0 本体
            ├─ lib/                    μT-Kernel 3.0 用のライブラリ
            ├─ docs/                   ドキュメント
            ├─ README.md               概要説明
            ├─ ucode.png               μT-Kernel 3.0 の ucode
            ├─ evk_xmc7200_Flash.ld    EVK-XMC7200 用のリンカスクリプト
            ├─ Makefile                ModusToolbox 用の Make ファイル
            ├─ .cyignore               ビルド対象外を指定するためのファイル
            ├─ .mtbLaunchConfigs/      プログラム起動用の設定ファイル
            ├─ .settings/              Eclipse 用の設定ファイル
            ├─ .project                Eclipse 用のプロジェクト構成ファイル
            ├─ .cproject               Eclipse の CDT プラグイン用の構成ファイル
            ├─ build/                  ModusToolbox のビルド用ディレクトリ
```



```
├── bsps/          ModusToolbox の BSP 関連の構成ファイル
├── deps/          ModusToolbox のライブラリ関連構成ファイル
├── libs/          ModusToolbox のライブラリ関連構成ファイル
├── LICENSE        ModusToolbox に関するライセンスの説明
├── README_mtb.md  ModusToolbox プロジェクトの概要説明
└── .gitignore     Git のトラッキング対象外ファイルの指定
```

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。

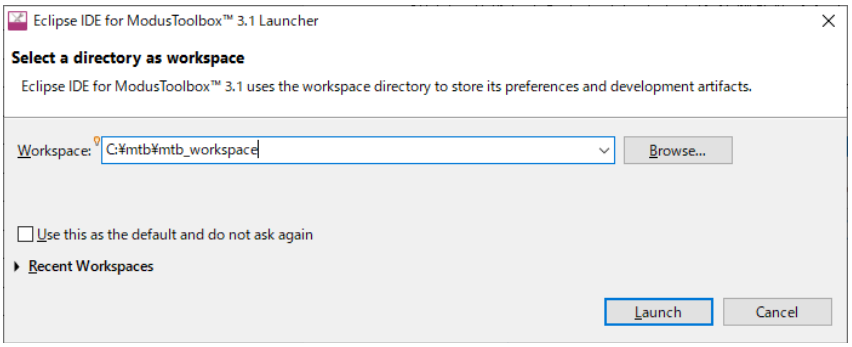


図 1　ワークスペース・ランチャーでワークスペースを指定

(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) 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。

(d) <APP-PRJ#1>/

Project Creator で生成するサンプルプロジェクトである。
下図ではデフォルトの Hello_World のまま変更していない。

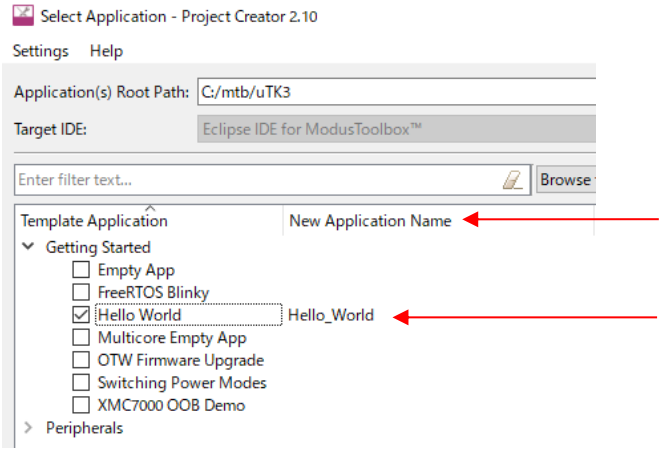


図 2　New Application Name の指定

(e) mtkernel_3

ModusToolbox にプロジェクトとしてインポートする際のルートディレクトリ

(f) kernel/、config/、device/、include/、lib/

μT-Kernel 3.0 関連のファイルが含まれるディレクトリ

(g) app_program/、app_serial/

μT-Kernel 3.0 用のサンプルアプリケーションが含まれるディレクトリ

(h) evk_xmc7200_Flash.ld

EVK-XMC7200 用のリンカスクリプト

(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。

- 用意したリンカスクリプトは Makefile において指定する。
  また、μT-Kernel 3.0 ではターゲット名を開発環境側で指定する必要がある。
  このため、Makefile に以下の設定を追加してある。

```
        :
DEFINES=_EVK_XMC7200_
        :
LINKER_SCRIPT=evk_xmc7200_Flash.ld
        :
```

(j) .cyignore

ビルド対象外のファイルやディレクトリを ModusToolbox に指示するファイル

- ModusToolbox では、新規に追加されたソースコードは ModusToolbox がビルド時に自動的に検出してビルド対象に追加する。特定のファイルやディレクトリに対してこの機能を無効にするには、.cyignore にそれらの相対パスを記載する必要がある。ModusToolbox は.cyignore に記載されているファイル(ディレクトリの場合はそれ以下の全てのファイル)をビルド対象外として無視(ignore)する。

(k) .mtbLaunchConfigs/ ～ .gitignore

Project Creator によって自動的に生成されるディレクトリやファイル

- README_mtb.md はμT-Kernel 3.0 用の README.md と重複しているので、ファイル名を変更してある。

2.5.2. 機種依存定義ディレクトリの構成

μT-Kernel 3.0 のソースコードは、以下の 2 種類のコードに大別される。

- 共通部　　機種に依存しない μT-Kernel 3.0 に共通のコード
- 機種依存部　機種に合わせて実装されたコード

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

```
└── sysdepend/                    実装依存定義
    ├── evk_xmc7200/              EVK-XMC7200 依存部
    └── cpu/                      CPU 依存部
        ├── xmc7200/              XMC7200 マイコン依存部
        └── core/                 コア依存部
            └── armv7m/           ARMv7-M コア依存部
```

「**EVK-XMC7200 依存部**」には、**コア依存部**および**マイコン依存部**以外で、**EVK-XMC7200** のハードウェアに固有となるコードがまとめられている。

「**XMC7200 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。

「**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コードがまとめられている。

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 EVK-XMC7200 の基本仕様

項目	内容
CPU コア	ARM Cortex-M7, ARM Cortex-M0+ ※本実装では、ARM Cortex-M7 のみを対象としている。
ROM	8384KB (Code Flash)
RAM	16KB(DTCM) 1MB (SRAM) ※本実装では OS は SRAM のみを使用する。

ℹ 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-M7 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 が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_LOWPOWER 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、以下に空関数として記述されている。

kernel/sysdepend/evk_xmc7200/power_save.c

本関数に適切な省電力処理を記述し、プロファイル TK_SUPPORT_LOWPOWER 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 TURE 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　「10.1.1. 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　「10.1.2. FPU 関連 API」参照

4. メモリ

4.1. メモリモデル

Cortex-M7 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-M7 と Cortex-M0+で共通である。このため、メモリ領域は Cortex-M7 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
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）
0x60000000 ～ 0x9FFFFFFF	外部 RAM 領域	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	外部デバイス領域
0xE0000000 ～ 0xE00FFFFF	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 ～ 0xFFFFFFFF	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

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	ROM (本実装では利用していない。)	64KB
0x01000000 ～ 0x0100FFFF	ROM Mirror	64KB
0x10000000 ～ 0x1082FFFF	コードフラッシュ	8384KB
0x14000000 ～ 0x1403FFFF	ワークフラッシュ (本実装では利用していない。)	32KB
0x17000000 ～ 0x17007FFF	監督用フラッシュ (本実装では利用していない。)	32KB
0x17800000 ～ 0x17807FFF	代替監督用フラッシュ (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、コードフラッシュ(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コードフラッシュ(ROM)に配置され、実行される。

例外ベクタテーブルは、リセット時はコードフラッシュ(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。

以下にコードフラッシュ(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(－)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　コードフラッシュ(ROM)のメモリマップ ：0x10000000～0x101FFFFF

アドレス	セクション名	種別	内容
0x10000000 ～ (－)	.cy_m0p_image	Cortex-M0+用 プログラム	Cortex-M0+用のプログラムイメージ
0x10080000 ～ (－)	.text	プログラム コード	プログラムコードが配置される。
(－)～(－)	.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。

アドレス	セクション名	種別	内容
(ー)～(ー)	.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
(ー)～(ー)	.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 ：0x20000000～0x280FFFFF

アドレス	セクション名	種別	サイズ	内容
0x20000000 ～ 0x20003FFF		DTCM	16KB	本実装では使用しない。
0x28000000 ～ 0x28003FFF		Cortex-M0+用	16KB	
0x28004000 ～ (ー)	.ramVectors	例外ベクタ テーブル		例外や割込みのベクタテーブル OS の初期化後に有効
(ー) ～ (ー)	.data	データ領域		プログラムの変数領域 (初期値あり)
(ー) ～ (ー)	.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。
(ー) ～ (ー)	.noinit	非初期化 変数領域		プログラムの変数領域 (初期値なし)
(ー) ～ (ー)	.bss	BSS 領域		プログラムの変数領域 (0 初期化)
(ー) ～ (ー)	.heap	ヒープ領域		ModusToolbox 用
(ー) ～ (ー)	(ー)	μT-Kernel 管理領域		OS 内部で利用する動的メモリに割り当てる領域
0x280EF000 ～ 0x280EFFFF	(ー)		4KB	初期化スタック領域 初期化処理時のみに使用する。
0x280F0000 ～ 0x280FFFFF		Cortex-M7 Core1 用	64KB	Core1 用として確保しているため、本実装では使用しない。

ℹ 各セクションの並び(やサイズ)はリンカスクリプト evk_xmc7200_Flash.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。このため、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. μT-Kernel 管理領域

OS の API の処理において以下のメモリがμT-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

μT-Kernel 管理領域は、μT-Kernel のメモリ管理機能で使用する領域であり、本実装ではヒープセクションの終端から設定ファイルの SYSTEMAREA_END で指定された間の領域となる。通常は、RAM の空いているメモリ領域が全てμT-Kernel 管理領域に割り当てられるが、設定により変更することも可能である。

μT-Kernel 管理領域の範囲を指定する変数は knl_lowmem_top と knl_lowmem_limit であり、Reset_Handler()関数において以下の様に設定されている。

kernel/sysdepend/cpu/core/armv7m/reset_hdl.c ： μT-Kernel 管理領域の設定

```
EXPORT void Reset_Handler(void)
{
        :
#if USE_IMALLOC
        /* Set System memory area */
        if(INTERNAL_RAM_START > SYSTEMAREA_TOP) {
                knl_lowmem_top = (UW*)INTERNAL_RAM_START;
        } else {
                knl_lowmem_top = (UW*)SYSTEMAREA_TOP;
        }
#if CPU_XMC7200
        if((UW)knl_lowmem_top < (UW)&__HeapLimit) {
                knl_lowmem_top = (UW*)&__HeapLimit;
        }
```

```
#else
        if((UW)knl_lowmem_top < (UW)&__bss_end) {
                knl_lowmem_top = (UW*)&__bss_end;
        }
#endif
        if((SYSTEMAREA_END != 0) && (INTERNAL_RAM_END > CNF_SYSTEMAREA_END)) {
                knl_lowmem_limit = (UW*)(SYSTEMAREA_END - EXC_STACK_SIZE);
        } else {
                knl_lowmem_limit = (UW*)(INTERNAL_RAM_END - EXC_STACK_SIZE);
        }
#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 を設定する。

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
16～23	IRQ 割込み#0～#7 (モジュール割込み)	OS の割込み管理機能で管理
24～31	IRQ 割込み#8～#15 (ソフトウェア割込み)	

XMC7200 にはペリフェラルからの割込みが 566 本存在するが、これらの割込みは IRQ0～7 の 8 本で受け付ける構造になっている(図 3 を参照)。ペリフェラルからの割込みの数に対して NVIC の IRQ の数が少ないので、図 3 の interrupt sources(Peripherals)からの割込みを IRQ0, IRQ1, IRQ2, …, IRQ7 に対応付けて処理する必要がある。

この他に、XMC7200 では、IRQ8～15 にソフトウェア割込み専用の割込みが割り当てられており、計 575 本の割込みを有していることになる(詳細は「XMC7000 MCU family architecture Technical reference manual」を参照のこと)。

本実装では、ペリフェラルからの割込み番号 0～566 をモジュール割込み番号、567～574 をソフトウェア割込み番号としている。モジュール割込みは IRQ0～7、ソフトウェア割込みは IRQ8～15 に対応させて実装してある。

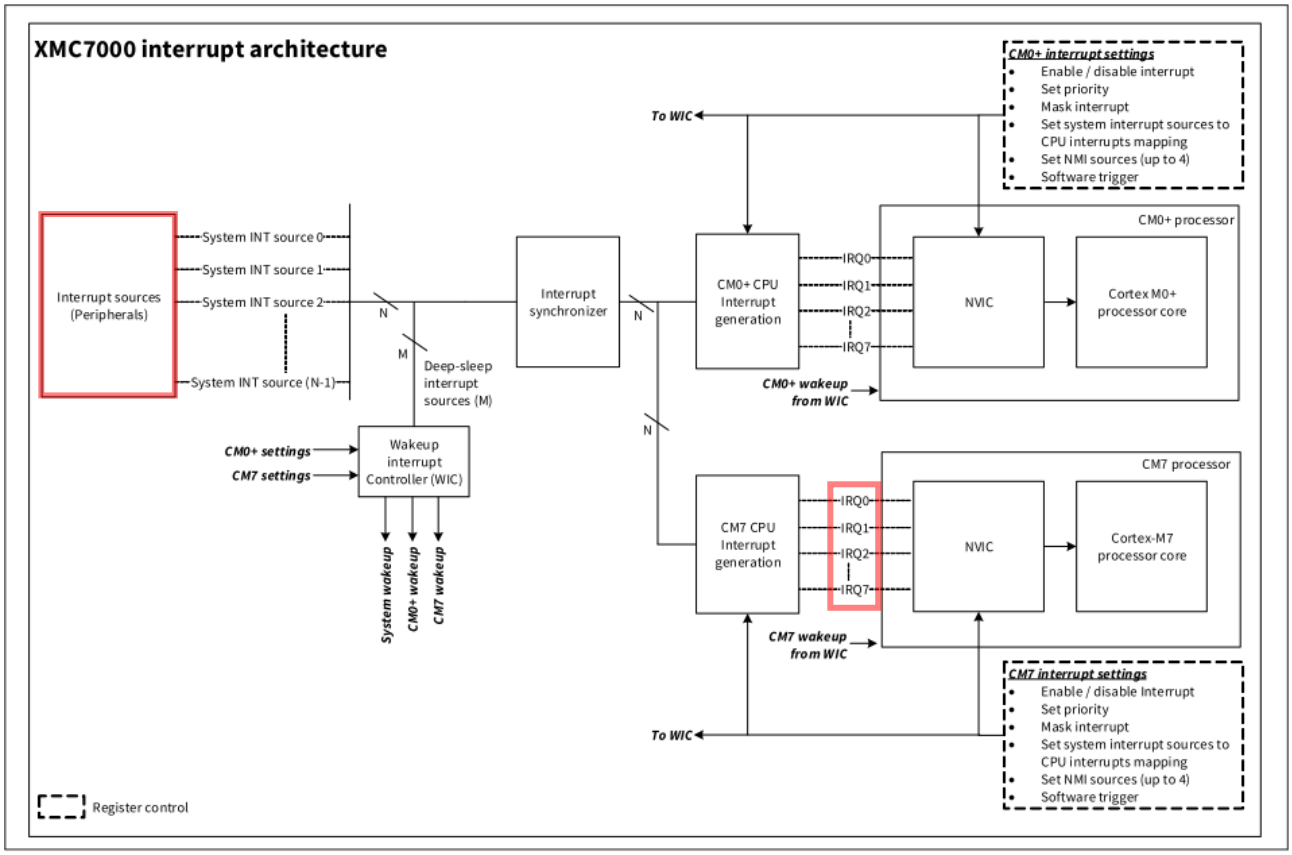


Figure 12-1. XMC7000 interrupts block diagram

図　3　XMC7000 シリーズの割込みアーキテクチャ

「XMC7000 MCU family architecture Technical reference manual」(p.170)から抜粋(説明のため赤枠を追加)

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 vector_tbl.c に vector_tbl として定義されている。

kernel/sysdepend/cpu/xmc7200/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 */
        CM7_CpuIntr0_Handler,       /* IRQ 0 (CPU User Interrupt #0) */
        CM7_CpuIntr1_Handler,       /* IRQ 1 (CPU User Interrupt #1) */
        CM7_CpuIntr2_Handler,       /* IRQ 2 (CPU User Interrupt #2) */
                :                               :
        CM7_CpuIntr7_Handler,       /* IRQ 7 (CPU User Interrupt #7) */
        CM7_CpuIntr8_Handler,       /* IRQ 8 (Internal Software Interrupt #0) */
        CM7_CpuIntr9_Handler,       /* IRQ 9 (Internal Software Interrupt #1) */
        CM7_CpuIntr10_Handler,      /* IRQ 10 (Internal Software Interrupt #2) */
                :                               :
        CM7_CpuIntr15_Handler,      /* IRQ 15 (Internal Software Interrupt #7) */
};
```

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。

RAM 上のベクタテーブルは、kernel/sysdepend/cpu/core/armv7m/interrupt.c に exchdr_tbl として定義される。

XMC7200 では、「5.1.マイコンの割込みおよび例外」で説明した通り、モジュール割込み 8 つ、ソフトウェア割込み 8 つの計 16 の割込みを有している。

566 のモジュール割込みを IRQ に割り当てる方法として、CM7_CpuIntrX_Handler(X=0～7)のハンドラにジャンプし、ハンドラ内でどのモジュール割込みが発生したかを判定している。具体的な実装を以下に示す。

kernel/sysdepend/cpu/xmc7200/ext_hdr.c

```
/*
 * The Inline handler for CPU interrupt.
 * The system interrupt mapped to CPU interrupt will be fetched and executed
 */
void CM7_CpuIntr_Handler(UINT intno)
{
        UW      system_int_idx;
        FP      inthdr;

        ENTER_TASK_INDEPENDENT;

        /* Check module interrupts with CPUSS_CM7_X_INT_STATUS of XMC7200. */
        if((*CM7_0_INT_STATUS(intno) & SYSTEM_INT_VALID) == SYSTEM_INT_VALID)
        {
                system_int_idx=(*CM7_0_INT_STATUS(intno) & SYSTEM_INT_IDX_MASK);
                inthdr = knl_inthdr_tbl[system_int_idx];
                inthdr(); // jump to system interrupt handler
        }
        ClearInt_nvic(intno);
```



```
        LEAVE_TASK_INDEPENDENT;
}
        :
void CM7_CpuIntr0_Handler(void)
{
        CM7_CpuIntr_Handler(0);
}
        :
void CM7_CpuIntr7_Handler(void)
{
        CM7_CpuIntr_Handler(7);
}
```

ソフトウェア割込みは CM7_CpuIntrX_Handler(X=8～15)のハンドラにジャンプし、どのソフトウェア割込みが発生したかを判定している。具体的な実装を以下に示す。

kernel/sysdepend/cpu/xmc7200/ext_hdr.c

```
/*
 * The handler for Software interrupt.
 * The system interrupt mapped to CPU interrupt will be fetched and executed
 */
void CM7_SoftIntr_Handler (UINT intno)
{
        UW      system_int_idx;
        FP      inthdr;

        ENTER_TASK_INDEPENDENT;

        /* The interrupt number corresponding to software interrupt number 567-
574 is IRQ8-15. */
        system_int_idx = (N_INTVEC - (CORE_EXT_INTVEC + CORE_SOFT_INTVEC) + intno);
        inthdr = knl_inthdr_tbl[system_int_idx];
        inthdr(); // jump to system interrupt handler

        ClearInt_nvic(intno);

        LEAVE_TASK_INDEPENDENT;
}
        :
void CM7_CpuIntr8_Handler(void)
{
        CM7_SoftIntr_Handler(8);
}
        :
void CM7_CpuIntr15_Handler(void)
{
        CM7_SoftIntr_Handler(15);
}
```


なお、XMC7200 では、設定可能な割込み優先レベルは 8 レベルとなっており、使用可能な割込み優先度は 2～6 である(「5.3.1. 割込み優先度」を参照)。このため、本実装では、IRQ0～7 をそれぞれ以下の優先度で予め設定しておく実装にした。結果として、使用可能な IRQ 番号は 0～4 のみとなり、それ以外の IRQ 番号(5～7)は未使用(使用不可)となる。

表 5-2 IRQ 割込みの優先度の割り当て

IRQ 割込み番号	優先度
0	2
1	3
2	4
3	5
4	6
5～7	未使用

ソフトウェア割込みに関しては、同様の制限事項はない。

5.3. 割込み優先度とクリティカルセクション

5.3.1. 割込み優先度

Cortex-M7 は、割込み優先度を 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、割込み優先度が 3bit になっているので、設定可能な割込み優先度は 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. 多重割込み対応

本マイコンの割込みコントローラ NVIC(Nested Vector Interrupt Controller)は、ハードウェアの機能として、多重割込みに対応している。割込みハンドラの実行中に、より優先度の高い割込みが発生した場合は、実行中の割込みハンドラに割り込んで優先度の高い割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/xmc7200/sysdef.h にて以下のように定義される。

include/sys/sysdepend/cpu/xmc7200/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 5-3 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 sysdef.h において以下のように定義される。

include/sys/sysdepend/cpu/xmc7200/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 */
#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-Kenrel/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)はモジュール割込みとソフトウェア割込み(0～574)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)。

「5.1. マイコンの割込みおよび例外」で説明した通り、XMC7200 の場合は、モジュール割込み 567 種類、ソフトウェア割込み 8 種類の計 575 種類存在するが、NVIC が受付可能な割込みは 16 種類である。

その内、モジュール割込みに割り当てられる割込みは IRQ0～7、ソフトウェア割込みに割り当てられる割込みは IRQ8～15 である。ただし、「5.3.1. 割込み優先度」で説明した通り、使用可能な割込み優先度は 2～6 である。

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、モジュール割込み番号とソフトウェア割込み番号とする。

5.5.2. 割込みハンドラ属性

Cortex-M では、ハードウェアの仕様として、C 言語の関数による割込みハンドラの記述が可能となっている。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

割込みハンドラが未登録の状態で、割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(「10.2.2. 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 の割込み管理機能

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

5.6.1. 割込みの優先度

割込みの優先度は、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 割込み制御

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

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

(3) CPU 内割込みマスクレベルの設定(SetCpuIntLevel)
CPU 内割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の制御を行う。モジュール割込み番号(0～566)の割込みは、IRQ0～4(IRQ5～7 は未使用)に割り当てられ、ソフトウェア割込み番号(567～574)の割込みは、IRQ8～15 に割り当てられている。
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)
モジュール割込み番号(0～566)を指定した場合と、ソフトウェア割込み番号(567～574)を指定した場合とで処理が異なる。
モジュール割込み番号(0～566)を指定した場合は、XMC7200 の割込みコントローラ(CPUSS_CM7_0_SYSTEM_INT_CTL)を指定し、指定された割込みを許可する。割込み優先度によって、IRQ0～IRQ4(IRQ5～7 は未使用)のいずれかの割込みを許可する。また、「5.6.1. 割込みの優先度」で説明した通り、割込みの優先度は 2～6 のいずれかを指定する(1 と 7 は使用不可)。

例1) 割込み番号 0、割込み優先度 2 を指定した場合は、
割込み番号 0 の割込みが IRQ0 で発生する。

例2) 割込み番号 1、割込み優先度 6 を指定した場合は、
割込み番号 1 の割込みが IRQ4 で発生する。

ソフトウェア割込み番号(567～574)を指定した場合は、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)
モジュール割込み番号(0～566)を指定した場合は、XMC7200 の割込みコントローラ(CPUSS_CM7_0_SYSTEM_INT_CTL)を設定し、指定された割込みを禁止する。
ソフトウェア割込み番号(567～574)を指定した場合は、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) 割込み発生のクリア(ClearInt)
ソフトウェア割込み番号(567～574)を指定した場合は、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。
モジュール割込み番号(0～566)は、該当するレジスタがないため、何も実行しない。

(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) 割込み発生の検査(CheckInt)
ソフトウェア割込み番号(567～574)を指定した場合は、割込み発生の検査(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。
モジュール割込み番号(0～566)は、該当するレジスタがないため、何も実行しない。

(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 5-4 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(予約)	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。

> XMC7200 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(未使用)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。

リセット処理は ARMv7-M に固有の処理であるため kernel/sysdepend/cpu/core/armv7m/reset_hdl.c の Reset_Handler 関数として実装される。

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。
knl_startup_hw に関しては「6.2.ハードウェアの初期化および終了処理」を参照のこと。

(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 に移動し、割込みベクタテーブルを OS 経由で変更可能にする。
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) 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。

(5) 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。

(6) FPU の初期化
コンフィギュレーション USE_FPU が指定されている場合、FPU 初期化処理を実行する。

(7) OS 初期化処理の実行
OS の初期化処理 main()関数を実行する。
main()関数の中で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/evk_xmc7200/hw_setting.c

関数名	内容
knl_startup_hw	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。
knl_shutdown_hw	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。
knl_restart_hw	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。

knl_startup_hw 関数では以下のハードウェアの初期設定を行う。

(1) 割込みコントローラ(NVIC)の初期化
割り込み番号 0～4 の割込みイネーブルセットレジスタ(ISER)を設定し、割込みを許可する。割り込み番号 5～7 は未使用のため、常時禁止とする。

(2) クロック(PLL、FLL)の初期化
クロックは以下の関数によって初期化する。

ファイル　kernel/sysdepend/cpu/xmc7200/cpu_clock.c
関数　　startup_clock

表 6-2 クロックの設定値

項目	内容
PLL400M_0 逓倍値設定	入力クロック 16MHz 出力クロック 340MHz
PLL400M_1 逓倍値設定	出力クロック 196MHz
PLL2 逓倍値設定	出力クロック 144MHz
PLL3 逓倍値設定	出力クロック 100MHz
FLL 逓倍値設定	出力クロック 50MHz

- クロックの値はプログラム中で使用可能なように定義してある。
- クロックの初期設定は原則として変更してはならない。もし変更した場合は、OS のクロック制御やデバイスドライバの制御にも影響がある場合があるため、合わせてプログラムの変更が必要となる。

(3) I/O 端子の機能設定

入出力ポートの各レジスタを初期化し、各 I/O 端子の機能設定を行う。
初期値では以下の端子の機能が設定されている。

表 6-3 初期設定端子

I/O ポート	機能	用途
P13_0	SCB3_RX	デバッグ用シリアル入出力(T-Monitor) シリアルドライバ入出力
P13_1	SCB3_TX	
P13_2	SCB3_RTS	
P13_3	SCB3_CTS	
P21_2	ECO_IN	外部水晶発振器
P21_3	ECO_OUT	

SE_SDEV_DRV を 1 に設定した場合、SCB3 の TX / RX はデバッグ用シリアルとシリアル通信デバイスドライバの両方の機能で利用されることになる。
ただし、両機能での排他制御は実装していない。このため、シリアルドライバでの出力中に tm_printf()で出力するなど、同時に動作させた場合の挙動は不定となる。
シリアルドライバを中心にアプリケーションを構築する場合は、デバッグ用シリアル機能を停止するか、または、それぞれを別の UART ポート(I/O 端子)に割り当てるなどの対応が必要となる。

- XMC7200 ではシリアル通信ポート(UART)を SCB(Serial Communication Block)と呼称している。このため、本資料では SCB として説明している。

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 6-4 ファイル：kernel/sysdepend/evk_xmc7200/devinit.c

関数名	内容
knl_init_device	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。
knl_start_device	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。
knl_finish_device	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 DEVCNF_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 6-5 サンプルドライバ

デバイス名	機能	個別マクロ定義
ser	シリアル通信 (SCB)	DEVCNF_USE_SER
iic	I2C 通信	DEVCNF_USE_IIC
adc	A/D コンバータ	DEVCNF_USE_ADC

ℹ 本実装ではシリアル通信にのみ対応している。
I2C 通信、A/D コンバータには対応していない。

USE_SDEV_DRV または DEVCNF_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

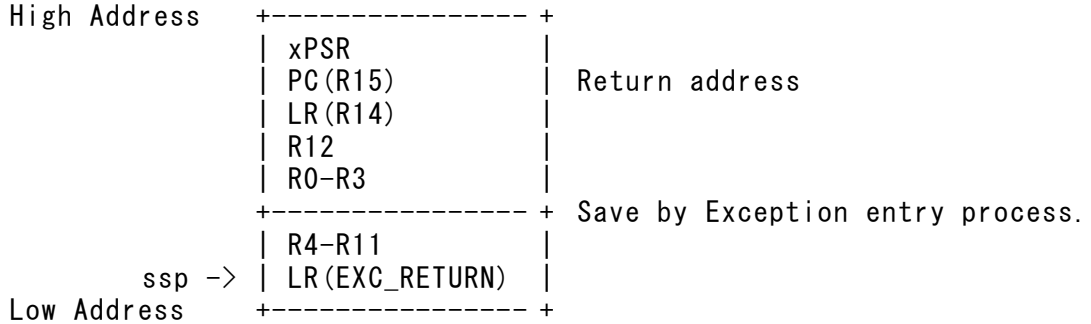
レジスタ	値	補足
PRIMASK	0	割込み許可
R0	第一引数 stacd	タスク起動コード
R1	第二引数 *exinf	タスク拡張情報
R13(MSP)	タスクスタックの先頭アドレス	

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_RETUN 値)のレジスタの値はディスパッチャにより保存される。

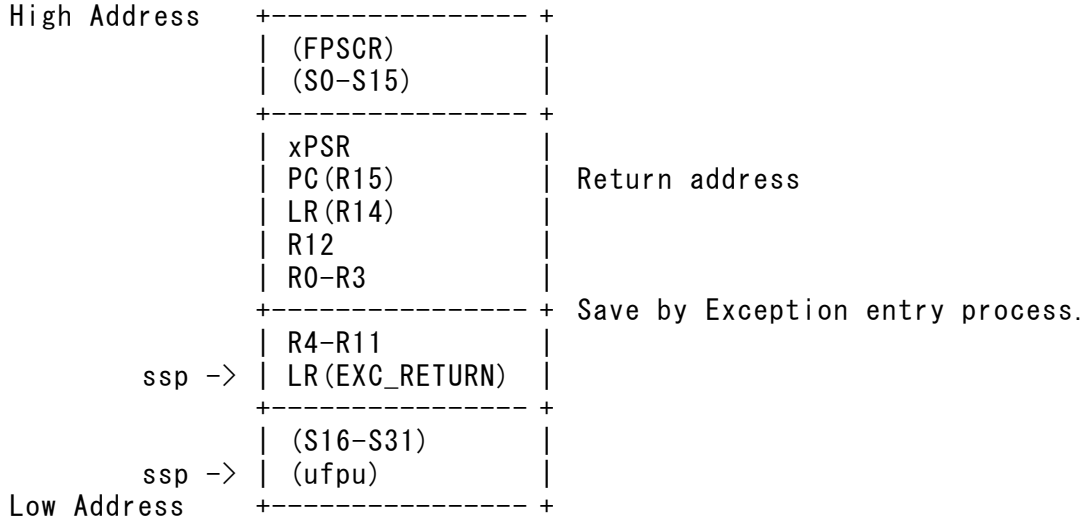


(2) TA_FPU 属性のタスク

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/xmc7200/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. サンプルデバイスドライバ

本実装では以下のデバイスに対するサンプルデバイスドライバを提供する。

・シリアル通信ドライバ　　(UART)

本章では、デバイスドライバの実装依存部分(固有仕様)について説明する。
なお、デバイスドライバに共通の仕様については以下の資料を参照のこと

(a)「μT-Kernel 3.0 共通デバイスドライバ説明書」　ユーシーテクノロジ(株)
(b)「μT-Kernel 3.0 デバイスドライバ説明書」　　トロンフォーラム

(a)では、**UCT** が開発した BSP に含まれるサンプルデバイスドライバに関する共通の仕様について説明している。
(b)では、**トロンフォーラム**が開発した BSP に含まれるサンプルデバイスドライバに関する共通の仕様について説明されている。

9.1. シリアル通信ドライバ(UART)

9.1.1. 対象デバイス

マイコン内蔵の非同期シリアル通信ポート(SCB)の一部を使用する。
以下にデバイスドライバと SCB の対応を記す。

表 9-1　対象ハードウェア

デバイス名	ユニット	デバイス	IO ピン	コネクタ	備考
"sera"	0	SCB0			非対応
"serb"	1	SCB1			非対応
"serc"	2	SCB2			非対応
"serd"	3	SCB3	P13[0:3]	ARD_P13[0:3]	T-Monitor 共用
"sere"	4	SCB4			非対応
:	:	:			:
"serk"	10	SCB10			非対応

◈ コネクタ ARD_P13[0:3]は IO ピン P13[0:3]とは接続されていない。
このため、コネクタを利用する場合は予め接続(0Ω抵抗を実装)する必要がある。

9.1.2. UART デバイスのパラメータ

UART(SCB)の通信速度などのパラメータの設定は以下の通りである。

(1) 通信速度

本実装では以下の通信速度(bps)に対応している。

115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200

通信速度の設定時に非対応の通信速度が指定された場合もエラーにはならない。
デフォルト値である 115200 bps が設定される。

(2) 通信パラメータ

本デバイスが対応する主な通信パラメータは以下の通りである。

表 9-2 通信パラメータの設定可否

通信パラメータ	設定	設定用マクロ	備考
CTS フロー制御	無効	(指定なし)	
	有効	DEV_SER_MODE_CTSEN	
RTS フロー制御	無効	(指定なし)	
	有効	DEV_SER_MODE_RTSEN	
ストップビット長	1	DEV_SER_MODE_1STOP	
パリティビット	なし	DEV_SER_MODE_PNON	
データ長	8	DEV_SER_MODE_8	

⚠ SCB3 の USB-UART ブリッジを利用して PC に接続する場合、KitProg3 の制限からデータ長／パリティビット／ストップビットはデフォルトの設定(8N1)から変更できない(変更しても正常に動作しない)。詳細については、"KitProg3 user guide"の"2.2.5.1 USB-UART Bridge Feature"を参照のこと。

(3) FIFO の設定

本デバイスの受信と送信は FIFO 経由で行われる。FIFO は、受信用、送信用共に 128 段(固定)である。
送信 FIFO、受信 FIFO のトリガレベルにはデフォルト値(63)を設定している。

ℹ 受信 FIFO 内のデータ数が受信 FIFO のトリガレベルより少ない場合、RTS 出力信号(uart_rts_out)がアクティブになる。

9.1.3. コンフィギュレーション

コンフィギュレーション定義ファイル ser_cnf_sysdep.h において、以下のハードウェアに依存するデバイス固有コンフィギュレーションを定義する。

device/ser/sysdepend/xmc7200/ser_cnf_sysdep.h ： シリアル通信ドライバの定義

```
/* Interrupt priority */
#define DEVCNF_UART_INTPRI    (3)
#define DEVCNF_UART0_INTPRI   (DEVCNF_UART_INTPRI)
#define DEVCNF_UART1_INTPRI   (DEVCNF_UART_INTPRI)
#define DEVCNF_UART2_INTPRI   (DEVCNF_UART_INTPRI)
#define DEVCNF_UART3_INTPRI   (DEVCNF_UART_INTPRI)
#define DEVCNF_UART4_INTPRI   (DEVCNF_UART_INTPRI)
#define DEVCNF_UART5_INTPRI   (DEVCNF_UART_INTPRI)
#define DEVCNF_UART6_INTPRI   (DEVCNF_UART_INTPRI)
#define DEVCNF_UART7_INTPRI   (DEVCNF_UART_INTPRI)
#define DEVCNF_UART8_INTPRI   (DEVCNF_UART_INTPRI)
#define DEVCNF_UART9_INTPRI   (DEVCNF_UART_INTPRI)
#define DEVCNF_UART10_INTPRI  (DEVCNF_UART_INTPRI)

/* Debug option
 *       Specify the device used by T-Monitor.
 *         -1 : T-Monitor does not use serial devices
 */
#if USE_TMONITOR
#define DEVCNF_SER_DBGUN      3           // SCB3 is used by T-Monitor
#else
#define DEVCNF_SER_DBGUN      -1          // T-Monitor not executed
#endif
```

リスト 1　EVK-XMC7200 用のシリアル通信ドライバの定義

- DEVCNF_UARTx_INTPRI (x=0～10)
  SCBx(x=0～10)の割込み優先順位を指定する。

- DEVCNF_SER_DBGUN
  T-Monitor 互換 API がコンソールとして使用する SCB のユニット番号を指定する。
  これにより、デバイスドライバと T-Monitor 互換 API の競合を防止する。
  具体的にはデバイスのオープン時以外でもコンソールの使用を可能とする。

  ℹ T-Monitor 互換 API がコンソールとして使用する SCB のシリアル通信の設定は、T-Monitor とシリアル通信ドライバで同一である必要がある。設定が異なる場合は、後から初期化が実行されるシリアル通信ドライバの設定が有効となる。

9.1.4. システムの初期化

SCB(UART)が使用するマイコンの端子には複数の機能が割り当てられている。このため、シリアル通信ドライバで SCB を使用するためには SCB 用の初期化が必要となる。

本デバイスドライバでは、端子の初期化は行わないので、マイコンの初期化時など、本デバイスを使用する前に使用する端子の初期化を行う必要がある。

> 本実装では SCB3 はデバッグ用シリアル通信にも使用するため、SCB3 の端子はシリアル通信用に初期化してある。他の UART ポートを使用する場合は、マイコンの初期化処理などにおいて、対象とする UART ポート用の端子の設定が必要となる。

9.1.5. エラー属性

通信中に発生したエラーは属性データ TDN_SER_COMERR に記録される

通信中にエラーが検出されるつど、属性データ TDN_SER_COMERR の該当するビットがセットされる。よって、記録されているエラーは同時に発生したとは限らない。

tk_srea_dev により属性データ TDN_SER_COMERR を読み出すと、値はクリアされる。

属性データ TDN_SER_COMERR のエラービットは、ser_mode_sysdep.h において以下のように定義されている。

device/ser/sysdepend/xmc7200/ser_mode_sysdep.h ： 通信エラーの定義

```
/* Communication Error : SCBx_INTR_RX*/
#define DEV_SER_ERR_PE   (0x00000200)    /* Parity Error   */
#define DEV_SER_ERR_FE   (0x00000100)    /* Framing Error  */
#define DEV_SER_ERR_OE   (0x00000020)    /* Overrun Error  */
```

リスト 2　EVK-XMC7200 用の通信エラーの定義

上記の定義は、device/include/device.h をインクルードすることにより、アプリケーションプログラムからも使用可能である。

app_serial/app_main.c ：device.h のインクルード例

```
#include <tk/tkernel.h>
#include <tm/tmonitor.h>
#include <device.h>
```

リスト 3　device.h をインクルードする際の実装例

10. その他の実装仕様

10.1. FPU 関連

10.1.1. 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、Lazystacking 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 10-1 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

Lazystacking 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。Lazystacking を無効にした場合の動作は保証しない。

10.1.2. 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_cpr、tk_get_cpr)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、cpudef.h において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)。

10.1.3. 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。

10.2. T-Monitor 互換ライブラリ

10.2.1. 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;

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

10.2.2. 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 10-2 コンソールの通信パラメータ

項目	内容
デバイス	内蔵 UART (SCB3)
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

EVK-XMC7200 では、ターゲットマイコン(XMC7200D-E272K8384)の SCB3 (Serial Communication Block 3)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 SCB3 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(J7)を PC と接続する(下図)。

図 4 コンソール用 UART(SCB3)の接続先(J7)

コネクタ J7 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。

- SCB3 の USB-UART ブリッジを利用して PC に接続する場合、KitProg3 の制限からデータ長／パリティビット／ストップビットはデフォルトの設定(8N1)から変更できない(変更しても正常に動作しない)。詳細については、"KitProg3 user guide"の"2.2.5.1 USB-UART Bridge Feature"を参照のこと。

tm_com.c には各種通信パラメータが定義してあるが、実際に指定(変更)可能なパラメータは通信速度 UART_BAUD のみである。UART_BAUD に指定可能な通信速度については、「9.1.2. UART デバイスのパラメータ」を参照のこと。

lib/libtm/sysdepend/evk_xmc7200/tm_com.c : **通信パラメータの指定**

```
#define UART_WORD_LEN       UART_WORD_LEN_8
#define UART_PARITY         UART_PARITY_NONE
#define UART_STOP_BIT       UART_STOP_BIT_1
#define UART_HW_FLOW        UART_HW_FLOW_DISABLE
#define UART_BAUD           (115200)
```


μT-Kernel 3.0

実装仕様書 (EVK-XMC7200)

Rev 3.00.01 (December, 2023)

ユーシーテクノロジ株式会社
141-0031 東京都品川区西五反田 2-12-3 第一誠実ビル 9F