# μT-Kernel3.0
# TX03 M367 IoT-Engine向け
# 実装仕様書

Version.01.00.03

2020.07.13

Copyright © 2019-2020 TRON Forum. All rights reserved.

変更履歴

| 版数（日付） | 内　容 |
|---|---|
| 1.00.03<br>(2020.07.13) | ● 4.4 OS 内部で使用する割込み<br>ディスパッチ要求と強制ディスパッチ要求を同一の割込み（PendSV）を使用するように変更<br>● 4.6 4.6 μT-Kernel/SM の割込み管理機能 ユーザが使用できる割込み範囲を変更（2～5 → 2～6）<br>● 6.3 物理タイマ機能　新規 |
| 1.00.02<br>(2020.05.29) | ● 1.4 関連ドキュメント バージョン番号等を更新<br>● 4.4 OS 内部で使用する割込み　説明分の見直し |
| 1.00.01<br>(2020.03.13) | ● 1.4 関連ドキュメント　更新<br>● 3.3 OS のメモリマップ<br>コンフィグレーション USE_STATIC_IVT の説明を追加<br>● 4.5 μT-Kenrel/OS の割込み管理機能<br>「割込みハンドラの優先度」の説明を本来の「4.6 μT-Kernel/SM の割込み管理機能」に移動。また「割込みハンドラの優先度」を「割込みの優先度」に修正。<br>● 5.2 ハードウェアの初期化および終了処理<br>5.3 デバイスドライバの実行および終了<br>ハードウェアの周辺デバイスと、OS が管理するデバイスドライバが、デバイスという同一の名称で一緒に説明されていたので、節を分けるように修正。また、後者をデバイスと呼び前者はハードウェアと呼ぶこととした。<br>● その他<br>全角、半角の文字統一など文章の体裁の修正 |
| 1.00.00<br>(2019.12.11) | ● 初版 |

Copyright © 2019-2020 TRON Forum. All rights reserved.

# 目次



Copyright © 2019-2020 TRON Forum. All rights reserved.

Copyright © 2019-2020 TRON Forum. All rights reserved.

# 1. 概要

## 1.1 目的

本書は TX03 M367 IoT-Engine 向けのμT-Kernel3.0 の実装仕様を記載した実装仕様書である。
対象は、TRON フォーラムから公開されているμT-Kernel 3.0(V3.00.01)のうち、TX03 M367 IoT-Engine 向けの実装部分である。ハードウェアに依存しない共通の実装仕様は、μT-Kernel3.0 実装仕様書を参照のこと。

以降、単に OS と称する場合はμT-Kenrel3.0 を示し、本実装と称する場合、前述のソースコードの実装を示す。

## 1.2 対象ハードウェア

実装対象のハードウェアは以下の通りである。

| 分類 | 名称 | 備考 |
|---|---|---|
| 実機 | TX03 M367 IoT-Engine | UC テクノロジー製 |
| 搭載マイコン | TX03 シリーズ M360 グループ<br>TMPM367FDFG | 東芝デバイス＆ストレージ製 |

## 1.3 ターゲット名

TX03 M367 IoT-Engine のターゲット名は以下とする。

| 分類 | 名称 | 対象 |
|---|---|---|
| ターゲット名 | _IOTE_M367_ | |
| 対象システム | IOTE_M367 | TX03 M367 IoT-Engine |
| 対象 CPU | CPU_TMPM369FDFG | TMPM367FDFG |
| 対象 CPU アーキテクチャ | CPU_CORE_ACM3 | ARM Cortex-M3 コア |

Copyright © 2019-2020 TRON Forum. All rights reserved.

## 1.4 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-Kernel3.0 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。

以下の関連するドキュメントを記す。

| 分類 | 名称 | 発行 |
|---|---|---|
| OS | μT-Kernel 3.0 仕様書<br>(Ver.3.00.00) | TRON フォーラム<br>TEF020-S004-3.00.00 |
| | μT-Kernel3.0 共通実装仕様書<br>(Ver.1.00.03) | TRON フォーラム<br>TEF033-W002-200713 |
| T-Monitor | T-Monitor 仕様書 | TRON フォーラム<br>TEF-020-S002-01.00.01 |
| 実機 | UCT TX03 M367 IoT-Engine<br>取扱説明書 | UC テクノロジー |
| 搭載マイコン | TMPM367FDFG データーシート | 東芝デバイス＆ストレージ |
| | Cortex™-M3 テクニカル<br>リファレンス マニュアル | ARM |

Copyright © 2019-2020 TRON Forum. All rights reserved.

## 1.5 ソースコード構成

機種依存定義 sysdeped ディレクトリ下の本実装のディレクトリ構成を以下に示す。太文字で書かれた箇所が、本実装のディレクトリである。

```
― sysdepend                    実装依存定義
├ iote_m367                    TX03 M367 Iot-Engine 依存部
├       :
├ <ターゲット n>                      ターゲット n 依存部
└ cpu                          CPU 依存部
   ├ tx03_m367                       TX03 M367 マイコン依存部
├      :
└ <CPUn>                       CPUn 依存部
└ core                        コア依存部
       ├ acm3                        ARM Cortex-M3 コア依存部
├     :
└ <coren>                      コア２依存部
```

Copyright © 2019-2020 TRON Forum. All rights reserved.

# 2. 基本実装仕様

## 2.1 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。

| 項目 | 内容 |
|---|---|
| CPU コア | ARM Cortex-M3 |
| ROM | 512KB(内蔵フラッシュ ROM) |
| RAM | 128KB(内蔵 RAM) |

## 2.2 実行モードと保護レベル

ARM Cortex-M3 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_RING0～TK_MEM_RING3 はすべて 0 が返される。

Copyright © 2019-2020 TRON Forum. All rights reserved.

## 2.3 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する。

(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

Copyright © 2019-2020 TRON Forum. All rights reserved.

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。

## 2.4 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、/kernel/sysdepend/iote_m367/power.c に空関数として記述されている。なお、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 を TRUE に指定すれば、OS の省電力機能(tk_set_pow API)は使用可能となる。

## 2.5 コプロセッサ対応

本マイコンにはコプロセッサは存在しない。よって、プロファイル TK_SUPPORT_COPn はすべて FALSE である。
コプロセッサに関する API(tk_set_cpr, tk_get_cpr)は提供されない。

Copyright © 2019-2020 TRON Forum. All rights reserved.

# 3. メモリ

## 3.1 メモリモデル

ARM Cortex-M3 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。

## 3.2 マイコンのアドレス・マップ

マイコンのアドレス・マップは、基本的に ARM Cortex-M3 のデフォルト・アドレス・マップに従う。
以下にマイコンのアドレス・マップを記す。表の備考欄の Code、SRAM、Peripheral の各領域は、Cortex-M3 のアドレス・マップの対応する領域を示す（詳細はマイコンの仕様書を参照のこと）。

| アドレス<br>（上段：開始<br>下段：終了） | 種別 | サイズ<br>(KByte) | 備考 |
|---|---|---|---|
| 0x0000 0000<br>0x0007 FFFF | 内蔵 ROM | 512 | Code 領域<br>内蔵フラッシュ ROM |
| 0x2000 0000<br>0x2000 07FF | バックアップ RAM | 2 | SRAM 領域※1 |
| 0x2000 0800<br>0x2001 FFFF | メイン RAM | 126 | SRAM 領域 |
| 0x4000 000<br>0x41FF FFFF | SFR(Special function Register) | | Peripheral 領域 |
| 0x6000 0000<br>0x63FF FFFF | 外部バス領域 | | |
| 0xE000 0000<br>0xE00F FFFF | CPU 内レジスタ領域 | | |

※1 低消費電力モード時もデータ保持するメモリ領域

本実装では、ビットバンドによるメモリアクセスは行っていない。ユーザのプログラムからは可能である。

## 3.3 OS のメモリマップ

Copyright © 2019-2020 TRON Forum. All rights reserved.

本実装では、マイコンの内蔵 ROM およびメイン RAM を使用する。
OS を含む全てのプログラムのコードは内蔵 ROM に配置され、実行される。
例外ベクタテーブルは、リセット時は内蔵 ROM 上にあるが、OS の初期化処理にて RAM 上に転送される。ただし、コンフィグレーション USE_STATIC_IVT を有効にすることにより、RAM 上への再配置を禁止することができる（初期値は無効）。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。

以下に内蔵 ROM およびメイン RAM 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、OS 内ではアドレスの指定は行っていない。

(1) 内蔵 ROM のメモリマップ

| アドレス※<br>（上段：開始<br>下段：終了） | 種別 | 内容 |
|---|---|---|
| 0x0000 0000<br>- | 例外ベクタ<br>テーブル | 例外や割込みのベクタテーブル<br>リセット時のみ有効 |
| -<br>- | プログラムコード | C 言語のプログラムコードが配置される領域 |
| -<br>- | 定数データ | C 言語の定数データなどが配置される領域 |

(2) メイン RAM のメモリマップ

| アドレス※<br>（上段：開始<br>下段：終了） | 種別 | 内容 |
|---|---|---|
| 0x2000 0800<br>- | 例外ベクタ<br>テーブル | 例外や割込みのベクタテーブル<br>OS の初期化後に有効 |
| -<br>- | プログラムデータ | C 言語の変数等が配置される領域 |
| -<br>- | OS 管理領域 | OS 内部の動的メモリ管理の領域 |
| -<br>0x2001 FFFF | 例外スタック<br>領域 | 初期化スタックとして、初期化処理時のみに使用 |

## 3.4 スタック

マイコンには、MSP(Main Stack Pointer)と PSP(Process Stack Pointer)の二種類のス

Copyright © 2019-2020 TRON Forum. All rights reserved.

タックが存在する。ただし、本実装では MSP のみを使用し、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。

## 3.5 OS 内の動的メモリ管理

OS 内の動的メモリ管理に使用する OS 管理メモリ領域は、以下のように定められる。

(1) OS 管理メモリ領域の開始アドレス
コンフィギュレーション CNF_SYSTEMAREA_TOP の値が 0 の場合、RAM 上のプログラムのデータ領域(BSS 領域)の最終アドレスの次のアドレスが、開始アドレスとなる。
値が 0 以外の場合は、その値が開始アドレスとなる。ただし、その値が RAM 上のプログラムのデータ領域(BSS 領域)の最終アドレス以下の場合は、BSS 領域の最終アドレスの次のアドレスが開始アドレスとなる。つまり、BSS 領域と重複することはない。

(2) OS 管理メモリ領域の終了アドレス
コンフィギュレーション CNF_SYSTEMAREA_END の値が 0 の場合、RAM 上の例外スタックの開始アドレスの前のアドレスが、終了アドレスとなる。

(3) 値が 0 以外の場合は、その値が終了アドレスとなる。ただし、その値が例外スタックの開始アドレス以上の場合は、例外スタックの開始アドレスの前のアドレスが開始アドレスとなる。つまり、例外スタックの領域と重複することはない。

Copyright © 2019-2020 TRON Forum. All rights reserved.

# 4. 割込みおよび例外

## 4.1 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。

| 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 | ハードフォルト | |
| 4 | メモリ管理 | |
| 5 | バスフォールト | |
| 6 | 用法フォールト | |
| 7～10 | 予約 | |
| 11 | SVC(スーパーバイザコール) | OS で使用 |
| 12 | デバッグモニタ | |
| 13 | 予約 | |
| 14 | PendSV | OS で使用 |
| 15 | SysTick | OS で使用 |
| 16～255 | 外部割込み#0 ~ #239 | OS の割込み管理機能で管理 |

## 4.2 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/core/acm3/vector_tbl.c に vector_tbl として定義される。

ただし、OS の初期化処理において、ベクタテーブルは RAM 上にコピーされ、以降そちらが使用される。RAM 上のベクタテーブルは、kernel/sysdepend/cpu/core/acm3/interrupt.c に exchdr_tbl として定義される。

## 4.3 割込み優先度とクリティカルセクション

### 4.3.1 割込み優先度

ARM Cortex-M3 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。

Copyright © 2019-2020 TRON Forum. All rights reserved.

ただし、本マイコンのハードウェアの実装は、割込み優先度は 3bit である。よって、実際に設定可能な割込み優先度は 8 段階(0～7)である。
以上より、外部割込みは優先度 0～7 が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仕様は許されない。また、優先度 1、6、7 は OS 内の割込み処理で使用しているため、同様にユーザプログラムから使用可能は許されない。よって、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、2～5 の 4 段階である。

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)
- ハードフォルト(優先度 -1)

### 4.3.2　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 Intrrupt Controler）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。

### 4.3.3　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、/sys/sysdef.h にて以下のように定義される。

```
#define INTPRI_MAX_EXTINT_PRI   1
```

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、ハードフォルトはクリティカルセクション中でもマスクされない。

## 4.4 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。

Copyright © 2019-2020 TRON Forum. All rights reserved.

| 種類 | 割込み番号 | 割り当てられる割込み・例外 | 優先度 |
|---|---|---|---|
| SVC 割込み | 11 | SVC<br>(スーパーバイザコール) | 0 |
| システムタイマ割込み | 15 | SysTick | 1 |
| ディスパッチ要求 | 14 | PendSV | 7 |

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定義のみが存在する。

SVC の例外番号は/include/sys/sysdepend/cpu/core/acm3/sysdef.h で以下のように定義される。

```
#define SVC_SYSCALL             0x00    /* System call */
#define SVC_EXTENDED_SVC        0x10    /* Extended SVC */
#define SVC_DEBUG_SUPPORT       0xFF    /* Debugger support function */
```

各割込み・例外の優先度は/include/sys/sysdepend/cpu¥tx03_m367/sysdef.h で以下のように定義される。

```
#define INTPRI_SVC              0       /* SVCall */
#define INTPRI_SYSTICK          1       /* SysTick */
#define INTPRI_PENDSV           7       /* PendSV */
```

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。

## 4.5 μT-Kenrel/OS の割込み管理機能

本実装では、割込み管理機能はマイコンの外部割込み(IRQ0～127)を対象とし、割込みハンドラの管理を行う。その他の例外については対応しない。

Copyright © 2019-2020 TRON Forum. All rights reserved.

### 4.5.1　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 は割込み番号 0 となる。

### 4.5.2　割込みハンドラ属性

ARM 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 0 以上の値のとき、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、前述の OS 割込み処理ルーチン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。

### 4.5.3　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¥acm3¥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、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 宣言がされているので、ユーザが同名の関数を作成しリンクすることにより、上書きすることができる。

Copyright © 2019-2020 TRON Forum. All rights reserved.

## 4.6 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。
各 API の実装について以降に記す。

### 4.6.1 割込みの優先度

割込みの優先度は、2 から 6 が使用可能である。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。また、優先度 0 は OS でマスクできないため使用してはならない。

### 4.6.2 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。
割込みマスクレベルは 1 から 7 の値が指定可能である。指定したマスクレベル以下の優先度の割込みはマスクされる。また、割込みマスクレベルに 0 が指定された場合は、すべての割り込みはマスクされない。

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する。

### 4.6.3 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。NVIC は ARM Cortex-M3 の機能である。
また、スタンバイ解除要因となる割込みは、マイコン内蔵のクロックジェネレータの制

Copyright © 2019-2020 TRON Forum. All rights reserved.

御も行う。これはTX03 M367マイコンの機能である（スタンバイ解除要因となる割込みについてはマイコンの仕様書を参照）。

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。割込み優先度は、２から５の値が使用可能である。

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 割込み発生のクリア(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。
また、スタンバイ解除要因となる割込みについては、クロックジェネレータの設定を行い、保持されている割込み要因をクリアする。

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)は何も実行しないマクロとして定義される。

⑤ 割込み発生の検査(CheckInt)
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。

⑥ 割込みモード設定(SetIntMode)
クロックジェネレータに割込みのモードを設定する。指定可能な割込みは、スタンバイ解除要因となる割込みのみである。
指定可能な割込みモードは、/sys/syslib.hに以下のように定義されている。

```
#define IM_LEVEL        0x0002          /* Level trigger */
#define IM_EDGE         0x0000          /* Edge trigger */
#define IM_HI           0x0000          /* H level/Interrupt at rising edge */
#define IM_LOW          0x0001          /* L level/Interrupt at falling edge
```

Copyright © 2019-2020 TRON Forum. All rights reserved.

*/

ただし、割込みの種別により指定可能なモードが定められているので、マイコンの仕様書を参照のこと。

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。

## 4.7 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、ハードフォルト、メモリ管理の例外がこれに該当する。

Copyright © 2019-2020 TRON Forum. All rights reserved.

## 4.8 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¥acm3¥exc_hdr.c の以下の関数として実装されている。

| 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 |

暫定的の例外ハンドラは、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。暫定的の例外ハンドラは weak 宣言がされているので、ユーザが同名の関数を作成し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。
優先度が 2～6 の場合は、ASM 属性の割込みハンドラと同等である。

Copyright © 2019-2020 TRON Forum. All rights reserved.

# 5. 起動および終了処理

## 5.1 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 ARM Cortex-M3 に固有の処理であるため kernel/sysdepend/cpu/core/acm3/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_device)
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_device は「5.2 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、変更可能とする。
ただし、コンフィギュレーションで USE_NOINIT が指定された場合が、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出来なくなる (初期値では USE_NOINIT は指定なし)。

(3) 変数領域(data, bss)の初期化
C 言語のグローバル変数領域の初期化を行い、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。

(4) 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。

(5) OS 初期化処理(main)の実行
リセット処理を終了する OS の初期化処理(main)を実行し、リセット処理は終了する。

## 5.2 ハードウェアの初期化および終了処理

OS の起動および終了に際して、マイコンおよび周辺デバイスの初期化、終了処理を行う。
マイコンの周辺デバイスの初期化および終了処理は、ユーザのアプリケーションに応じ

Copyright © 2019-2020 TRON Forum. All rights reserved.

て処理を実装する必要がある。
OS の標準のソースコード内では、OS が使用する周辺デバイスのみの必要最低限の初期化および終了処理が、以下の関数で実装されている。ユーザは必要に応じて各関数の内容を変更してよい。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。関数の仕様については、共通実装仕様書も参照のこと。

ファイル：kernel/sysdepend/iote_m367/hw_setting.c

| 関数名 | 内容 |
|---|---|
| knl_startup_hw | ハードウェアのリセット<br>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。<br>本実装では以下の処理を実行している。<br>・ウォッチドックの初期化(停止)<br>・クロック(PLL)初期化 (enable_pll)<br>・I/O 端子の機能設定 |
| knl_shutdown_hw | ハードウェアの停止<br>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。<br>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<br>周辺デバイスおよびマイコンの再起動を行う。<br>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。 |

## 5.3 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、デバイスドライバの登録、実行、終了を行う。
本実装には、デバイスドライバは含まれていないため、実際の処理は実装されていない。
以下の関数に必要な処理を記述する必要がある。関数の仕様については、共通実装仕様書も参照のこと。

ファイル：kernel/sysdepend/iote_m367/devinit.c

| 関数名 | 内容 |
|---|---|
| knl_init_device | デバイスの初期化<br>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 |

Copyright © 2019-2020 TRON Forum. All rights reserved.

| | |
|---|---|
| | 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。<br>本実装では処理は未実装である。 |
| knl_start_device | デバイスの実行<br>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。<br>本実装では処理は未実装である。<br>本実装では処理は未実装である。 |
| knl_finish_device | デバイスの終了<br>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。<br>本実装では処理は未実装である。 |

Copyright © 2019-2020 TRON Forum. All rights reserved.

# 6. その他の実装仕様

## 6.1 タスク

### 6.1.1 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。

| 属性 | 可否 | 説明 |
|---|---|---|
| TA_COPn | × | 本マイコンは FPU を持たない。 |
| TA_FPU | × | |

### 6.1.2 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。

| 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 | |

## 6.2 時間管理機能

### 6.2.1 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。

### 6.2.2 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、
include¥sys¥sysdepend¥cpu¥tx03_m367¥sysdef.h で定義さる。
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。

```
#define TIMER_INTLEVEL  0     //すべての割込みを許可
```

Copyright © 2019-2020 TRON Forum. All rights reserved.

## 6.3 物理タイマ機能

### 6.3.1 使用するハードウェアタイマ

マイコン内蔵の 16 ビットタイマ（TMRB）を 16 ビットインターバルタイマモードで使用して 8 個の物理タイマが実装されている。
16 ビットタイマは TMRB0～TMRB7 の 8 チェンネルが存在し、順番に物理タイマ番号が 1 から割り当てられる。

| 物理タイマ番号 | 対応するタイマ |
|---|---|
| 1 | TMRB0 |
| 2 | TMRB1 |
| 3 | TMRB2 |
| 4 | TMRB3 |
| 5 | TMRB4 |
| 6 | TMRB5 |
| 7 | TMRB6 |
| 8 | TMRB7 |

MTRB は物理タイマ以外の用途にも使用可能である。その場合は物理タイマ API を呼び出さなければよい（API StartPhysicalTimer にて MTRB は物理タイマに初期化される）。

### 6.3.2 タイマの設定

物理タイマのクロック設定は、include¥sys¥sysdepend¥cpu¥tx03_m367¥sysdef.h に以下のように定義される。

```
#define TB0MOD_TBCLK    (0x00000001)    // Source clock = T1
#define TB1MOD_TBCLK    (0x00000001)    // Source clock = T1
#define TB2MOD_TBCLK    (0x00000001)    // Source clock = T1
#define TB3MOD_TBCLK    (0x00000001)    // Source clock = T1
#define TB4MOD_TBCLK    (0x00000001)    // Source clock = T1
#define TB5MOD_TBCLK    (0x00000001)    // Source clock = T1
#define TB6MOD_TBCLK    (0x00000001)    // Source clock = T1
#define TB7MOD_TBCLK    (0x00000001)    // Source clock = T1
```

この値は、TMRB の TBcMOD レジスタの TBCLK ビットに設定される。上記の設定を変更することにより、各物理タイマのクロックを変更できる。

Copyright © 2019-2020 TRON Forum. All rights reserved.

### 6.3.3　タイマ割込み

物理タイマはその内部処理において、各 TMRB のコンペアー致/オーバーフロー割込み(INTTB)を使用する。

| 物理タイマ番号 | 対応する割込み | 割込み番号 |
|---|---|---|
| 1 | INTTB0 | 75 |
| 2 | INTTB1 | 78 |
| 3 | INTTB2 | 81 |
| 4 | INTTB3 | 84 |
| 5 | INTTB4 | 87 |
| 6 | INTTB5 | 90 |
| 7 | INTTB6 | 93 |
| 8 | INTTB7 | 96 |

各割込みの優先度は include¥sys¥sysdepend¥cpu¥tx03_m367¥sysdef.h に以下のように定義される。

```
#define INTPRI_TMRB0    5       // 物理タイマ 1
#define INTPRI_TMRB1    5       // 物理タイマ 2
#define INTPRI_TMRB2    5       // 物理タイマ 3
#define INTPRI_TMRB3    5       // 物理タイマ 4
#define INTPRI_TMRB4    5       // 物理タイマ 5
#define INTPRI_TMRB5    5       // 物理タイマ 6
#define INTPRI_TMRB6    5       // 物理タイマ 7
#define INTPRI_TMRB7    5       // 物理タイマ 8
```

割込み優先度は必要に応じて変更可能である。

## 6.4 T-Monitor 互換ライブラリ

### 6.4.1　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。本初期化関数は OS の起動処理関数(main)の中で実行される。

### 6.4.2　コンソール入出力

API によるコンソール入出力の仕様を以下に示す。

Copyright © 2019-2020 TRON Forum. All rights reserved.

| 項目 | 内容 |
|---|---|
| デバイス | 内蔵 UART Channel 5 |
| ボーレート | 115200bps |
| データ形式 | data 8bit, stop 1bit, no parity |

以上

Copyright © 2019-2020 TRON Forum. All rights reserved.