

μ T-Kernel 3.0 実装仕様書 (SBK-M4MN)

Rev 3.00.01

April, 2023



μT-Kernel 3.0

目次

1. はじめに	4
1.1. 本書について	4
1.2. 表記について	4
2. 概要	5
2.1. 目的	5
2.2. 対象ハードウェア	5
2.3. ターゲット名	5
2.4. 関連ドキュメント	6
2.5. ディレクトリ構成	7
3. 基本実装	8
3.1. 対象マイコン	8
3.2. 実行モードと保護レベル	8
3.3. CPU レジスタ	8
3.4. 低消費電力モードと省電力機能	9
3.5. コプロセッサ対応	9
4. メモリ	10
4.1. メモリモデル	10
4.2. マイコンのアドレス・マップ	10
4.3. OS のメモリマップ	11
4.4. スタック	11
4.5. μT-Kernel 管理領域	12
5. 割込みおよび例外	14
5.1. マイコンの割込みおよび例外	14
5.2. ベクタテーブル	14
5.2.1. 割込み優先度	14
5.2.2. 多重割込み対応	15
5.2.3. クリティカルセクション	15
5.3. OS 内部で使用する割込み	15
5.4. μT-Kernel/OS の割込み管理機能	16
5.4.1. 割込み番号	16
5.4.2. 割込みハンドラ属性	17
5.4.3. デフォルトハンドラ	17
5.5. μT-Kernel/SM の割込み管理機能	17
5.5.1. 割込みの優先度	18
5.5.2. CPU 割込み制御	18
5.5.3. 割込みコントローラ制御	18

5.6. OS 管理外割込み	19
5.7. その他の例外	19
6. 起動および終了処理	21
6.1. リセット処理	21
6.2. ハードウェアの初期化および終了処理	22
6.3. デバイスドライバの実行および終了	22
7. タスク	24
7.1. タスク属性	24
7.2. タスクの処理ルーチン	24
7.3. タスク・コンテキスト情報	24
8. 時間管理機能	26
8.1. システムタイマ	26
8.2. タイムイベントハンドラ	26
8.3. 物理タイマ機能	26
9. サンプルデバイスドライバ	27
9.1. シリアル通信ドライバ(UART)	27
9.1.1. 対象デバイス	27
9.1.2. UART デバイスのパラメータ	27
9.1.3. コンフィギュレーション	28
9.1.4. システムの初期化	29
9.1.5. エラー属性	29
10. その他の実装仕様	31
10.1. FPU 関連	31
10.1.1. FPU の初期設定	31
10.1.2. FPU 関連 API	31
10.1.3. FPU 使用の際の注意点	32
10.2. T-Monitor 互換ライブラリ	32
10.2.1. ライブラリ初期化	32
10.2.2. コンソール入出力	33

1. はじめに

本品はユーシーテクノロジー(株)が開発した SBK-M4MN(ES)ボード(以下、SBK-M4MN)対応の 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、イーエスピー企画製の評価ボード SBK-M4MN で動作させるための機種依存部を追加してある。

- ◆ 本書のターゲットボードである SBK-M4MN は実在しない。ターゲットボードとしては「SBK-M4KN(ES)ボード」(SBK-M4KN)を使用する必要がある。このため、本書の説明にあるターゲットボード名称は、SBK-M4MN を SBK-M4KN と読み替えること。

ただし、μT-Kernel 3.0 の対象となる CPU としては TPM4MNFYAFG 用と TPM4KNFYAFG 用を別に実装してある。ターゲットボードも SBK-M4KN と同一の構成で CPU のみが TPM4MNFYAFG に変更されているターゲットボード(SBK-M4MN)が別に存在するものと仮定して実装してある。よって、μT-Kernel のディレクトリ名やファイル名を読み替える必要はない。

1.1. 本書について

本書は SBK-M4MN 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 や RTOS と称する場合は μT-Kernel 3.0 を示し、**本実装**と称する場合は SBK-M4MN 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。
◆	注意が必要な内容の場合に記述する。
❗	補足やヒントなどの内容の場合に記述する。
<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

2.1. 目的

本書では、μT-Kernel 3.0 BSP に含まれる SBK-M4MN 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。

2.2. 対象ハードウェア

実装対象のハードウェアおよび OS は以下の通りである。

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	TMPM4MNFYAFG	東芝デバイス & ストレージ株式会社
OS	μT-Kernel 3.00.06	トロンフォーラム
実機 (マイコンボード)	SBK-M4MN (ES) ボード	株式会社 イーエスピー企画

2.3. ターゲット名

SBK-M4MN のターゲット名および識別名は以下とする。

表 2-2 SBK-M4MN 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	_SBK_M4MN_	SBK-M4MN
対象システム	SBK_M4MN	SBK-M4MN
対象 CPU	CPU_M4MN	TMPM4MNFYAFG
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/sbk_m4mn/machine.h
```

2.4. 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-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。

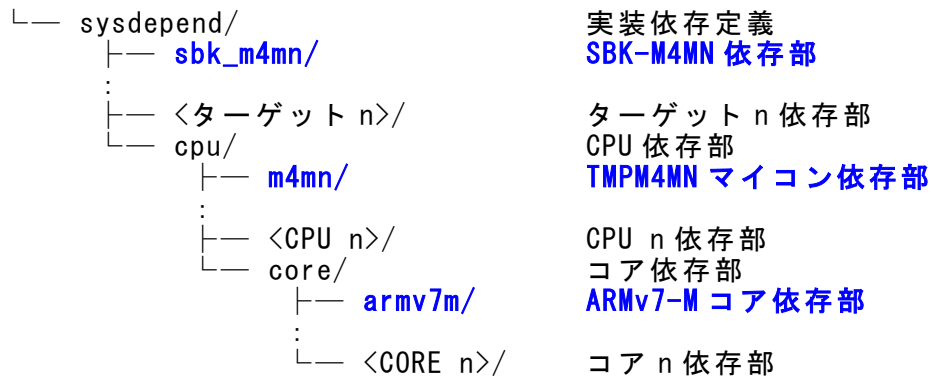
表 2-3 関連ドキュメント

分類	名称	発行
OS	μT-Kernel 3.0 仕様書 (Ver. 3.00.01)	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.1.00.10)	トロンフォーラム TEF033-W002-221018
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.1.00.06)	トロンフォーラム ユーシーテクノロジー(株)
ターゲットボード	SBK-M4MN 取り扱い説明書 SBK-M4MN 回路図	株式会社 イーエスピー企画
搭載マイコン	TXZ+ファミリー TMPM4M グループ(1) データシート	東芝デバイス & ストレージ株式会社

2.5. ディレクトリ構成

機種依存定義ディレクトリ `sysdepend/` の本実装のディレクトリ構成を以下に示す。

太字 で書かれた箇所が、本実装のディレクトリである。



「**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。

「**TMPM4MN 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。

「**SBK-M4MN 依存部**」は、コア依存部およびマイコン依存部以外の **SBK-M4MN** 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 SBK-M4MN の基本仕様

項目	内容
CPU コア	ARM Cortex-M4F
ROM	256KB (Flash)
RAM	24KB

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/sbk_m4mn/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 「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-M4 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。

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
0x00000000～0x0003FFFF	Code Flash	256KB
0x00040000～0x1FFFFFFF	Fault	
0x20000000～0x20001FFF	RAM0	8KB
0x20002000～0x20003FFF	RAM1	8KB
0x20004000～0x20005FFF	RAM2	8KB
0x20006000～0x21FFFFFF	Fault	
0x22000000～0x23FFFFFF	Bit Band Alias(RAM)	
0x24000000～0x2FFFFFFF	Fault	
0x30000000～0x30007FFF	Data Flash	32KB
0x30008000～0x3F7F7FFF	Fault	
0x3F7F8000～0x3F7F97FF	Boot ROM	
0x3F7F9800～0x40004FFF	Fault	
0x40005000～0x40005FFF	SFR	
0x40006000～0x4003DFFF	Fault	
0x4003E000～0x400FFFFF	SFR	
0x40100000～0x41FFFFFF	Fault	
0x42000000～0x43FFFFFF	Bit Band Alias(SFR)	
0x44000000～0x5DFEFFFF	Fault	
0x5DFF0000～0x5DFFFFFF	Flash(SFR)	
0x5E000000～0x5E03FFFF	CodeFlash(Mirror)	256KB
0x5E040000～0xDFFFFFFF	Fault	
0xE0000000～0xE00FFFFF	CPU Register Region	
0xE0100000～0xFFFFFFFF	Vender-Specific	

4.3. OS のメモリマップ

本実装では、Code Flash(ROM)および RAM0、RAM1、RAM2 を使用する。

OS を含む全てのプログラムのコードは CodeFlash 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 Code Flash(ROM)上にあるが、OS の初期化処理にて RAM 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、RAM 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 EWARM 統合開発環境でビルドした Code Flash(ROM)および RAM0～RAM2 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 4-2 Code Flash(ROM)のメモリマップ : 0x00000000～0x0003FFFF

アドレス	セクション名	内容
0x00000000～0x00000233	.intvec	例外ベクタテーブル RAM に再配置される。
0x00000234～(ー)	.data_init	.data 用の初期値
0x00000400～(ー)	.text	プログラムコード

表 4-3 内蔵 SRAM のメモリマップ : 0x20000000～0x20005FFF

アドレス	セクション名	内容
0x20000000～0x20000233	.data_vector	例外のベクタテーブル
0x20000400～(ー)	.data	変数領域(初期値あり)
(ー)～(ー)	.noinit	変数領域(初期値なし)
(ー)～(ー)	.bss	変数領域(0 クリア)
(ー)～(ー)	(ー)	μT-Kernel 管理領域 (OS 内部で利用する動的メモリ)
(ー)～0x2002FFFF	(ー)	初期化スタック

- ❶ 各セクションの並びや配置は以下のリンクスクリプトで指定する。

ide/iar/sbk_m4mn/config/sbk_m4mn_Flash.icf

- ❷ 初期化スタックのサイズは `CNF_EXC_STACK_SIZE` で指定する。

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、原則として BSS 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 μ T-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。

μ T-Kernel 管理領域の範囲を指定する変数は `kn1_lowmem_top` と `kn1_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` : μ T-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) {
            kn1_lowmem_top = (UW*)INTERNAL_RAM_START;
        } else {
            kn1_lowmem_top = (UW*)SYSTEMAREA_TOP;
        }
        if ((UW)kn1_lowmem_top < (UW)__bss_end) {
            kn1_lowmem_top = (UW*)__bss_end;
        }

        if ((SYSTEMAREA_END != 0) && (INTERNAL_RAM_END > CNF_SYSTEMAREA_END)) {
            kn1_lowmem_limit = (UW*)(SYSTEMAREA_END - EXC_STACK_SIZE);
        } else {
            kn1_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	デバッグモニター	
13	予約	
14	PendSV	OS で使用
15	SysTick	OS で使用
16～140	IRQ 割込み#0～#124	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/m4mn/vector_tbl.c` に `vector_tbl` として定義される。

ベクタテーブルは OS の初期化処理時に RAM にコピーされ、以降はそちらが使用される。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。

5.2.1. 割込み優先度

Cortex-M4 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、実際には割込み優先度はハードウェアの実装に依存する。本マイコンのハード

ウェアの実装は、割込み優先度は 4bit である。よって、設定可能な割込み優先度は 16 段階(0～15)である。

以上より、外部割込みは優先度 0～15 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使用は許されない。また、優先度 1 と 15 は OS 内の割込み処理で使用するため、同様にユーザプログラムからの使用は許されない。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、2～14 の 13 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)
- ・ ハードフォールト (優先度 : -1)

5.2.2. 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。

5.2.3. 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/m4mn/sysdef.h にて以下のように定義される。

include/sys/sysdepend/cpu/m4mn/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。

5.3. 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 以外で使用するしてはならない。

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	15
強制ディスパッチ要求	14	PendSV	15

各割込み・例外の優先度は `include/sys/sysdepend/cpu/m4mn/sysdef.h` で以下のように定義される。

`include/sys/sysdepend/cpu/m4mn/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    /* SVCcall */
#define INTPRI_SYSTICK          1    /* SysTick */
#define INTPRI_PENDSV           15   /* 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(15)を使用している。

ユーザプログラムは、割込み優先度 2～14 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。

5.4. 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(IRQ0～108)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「5.7. その他の例外」を参照のこと)。

5.4.1. 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。

5.4.2. 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 `kn1_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 `kn1_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。

5.4.3. 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 互換ライブラリのコンソールに出力される(「10.2.2. コンソール入出力」参照)ので、`USE_TMONITOR` も合わせて有効に設定する必要がある。

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` 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。

5.5. 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。

5.5.1. 割込みの優先度

割込みの優先度は、2～14 を使用可能である。優先度 1 および 15 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。

5.5.2. 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 1～15 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、15 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。

5.5.3. 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) 割込み発生のカ査(**CheckInt**)

割込み発生のカ査(**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**)

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。

5.6. 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。

5.7. 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。

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 (`kn1_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。

`kn1_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_NOINIT` 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ず。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 (0) が指定されている。

(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) μT-Kernel 管理領域の確保

μT-Kernel 管理領域(システムメモリ領域)の確保を行う。

ただし、`USE_IMALLOC` に (0) が指定された場合、本処理は実行されない(詳細は「4.5. μT-Kernel 管理領域」を参照)。

(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

`USE_FPU` に (1) が設定されている場合は FPU を有効に設定する。

(7) OS 初期化処理(`main`)の実行

`main()` 関数を呼び出して OS の初期化処理を実行する。

原則として `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。

μ T-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/sbk_m4mn/hw_setting.c`

関数名	内容
<code>knl_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。
<code>knl_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、割込み禁止状態での無限ループとしている。
<code>knl_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実装してある。

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してはならない。

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

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/sbk_m4mn/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、OS の機能は利用できない。
<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るので、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るので、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 `kn1_start_device()` 関数で登録される。

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
sera	シリアル通信 (UART0)	DEVCNF_USE_SER
iica	I2C 通信	DEVCNF_USE_IIC
adca	A/D コンバータ	DEVCNF_USE_ADC

※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

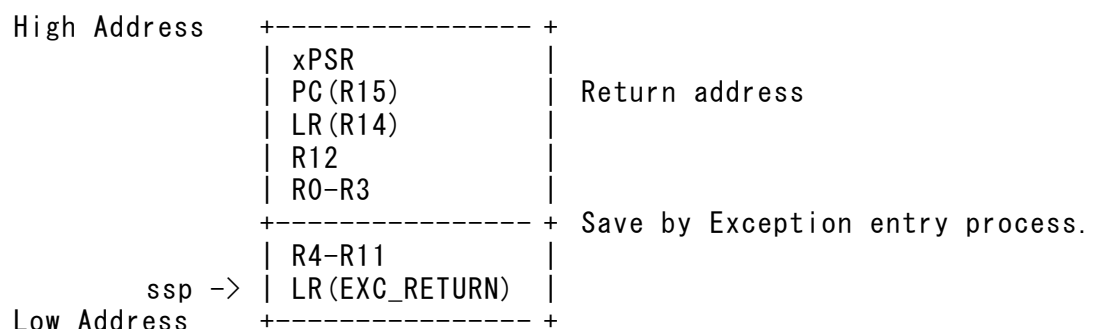
レジスタ	値	補足
PRIMASK	0	割込み許可
R0	第一引数 <code>stacd</code>	タスク起動コード
R1	第二引数 <code>*exinf</code>	タスク拡張情報
R13(MSP)	タスクスタックの先頭アドレス	

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。

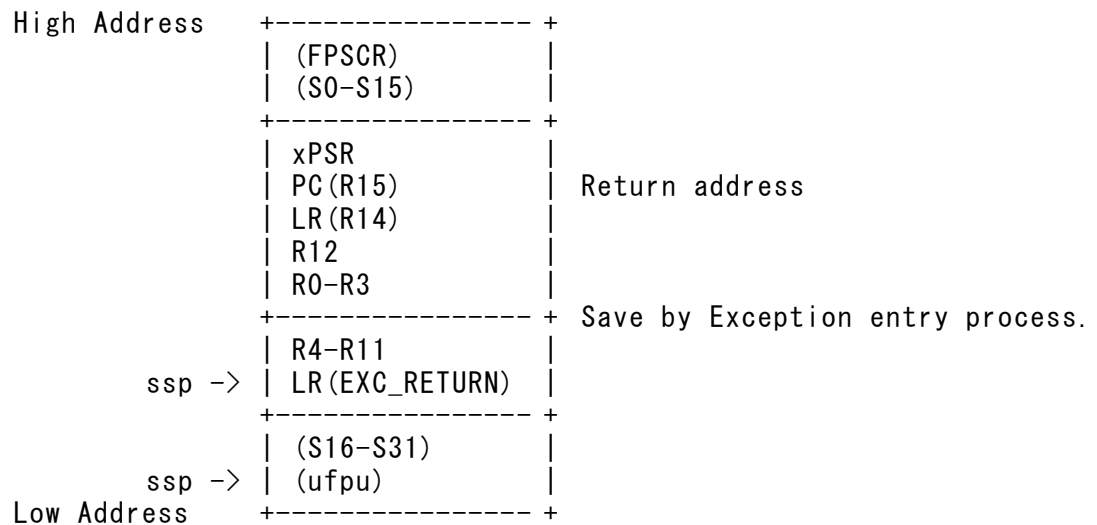


(2) TA_FPU 属性のタスク

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ケベルは、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/m4mn/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ー(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チャネル(UART)を使用する。

以下にデバイスドライバとUARTの対応を記す。

表 9-1 対象ハードウェア

デバイス名	ユニット番号	対象デバイス	備考
"sera"	0	UART0	T-Monitor 共用、USB2 で接続
"serb"	1	UART1	非対応
"serc"	2	UART2	非対応
"serd"	3	UART3	非対応

9.1.2. UART デバイスのパラメータ

- 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

対応していない通信速度が指定された場合は、デバイスオープン時にエラー(E_PAR)となる。(通信速度の設定時にはエラーにならない。)

- 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)	CTS フロー制御は常時無効
	有効	DEV_SER_MODE_CTSEN	設定不可
RTS フロー 制御	無効	(指定なし)	RTS フロー制御は常時無効
	有効	DEV_SER_MODE_RTSEN	設定不可
ストップ ビット長	1	DEV_SER_MODE_1STOP	
	2	DEV_SER_MODE_2STOP	
パリティ ビット	奇数	DEV_SER_MODE_PODD	
	偶数	DEV_SER_MODE_P EVEN	
	なし	DEV_SER_MODE_PNON	
データ長	7	DEV_SER_MODE_7	
	8	DEV_SER_MODE_8	
	9	なし	指定不能

❗ SBK-M4MN では UART0 の CTS と RTS の信号線が接続されていない。

このため、CTS と RTS によるハードウェアフロー制御は無効である。

通信モードとして DEV_SER_MODE_CTSEN、DEV_SER_MODE_RTSEN を指定しても内部的に無効に設定される(エラーにはならない)。

❗ データ長=9 ビットは、DEV_SER_MODE_9 が定義されていないので指定できない。

- FIFO の設定

本デバイスの受信バッファと送信バッファは共に「シフトレジスタ」と「FIFO」で構成されている。FIFO は、受信用、送信用ともに 8 段(固定)である。

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/m4mn/ser_cnf_sysdep.h` : シリアル通信ドライバの定義

```
/* Interrupt priority */
#define DEVCNF_UART0_INTPRI    6
#define DEVCNF_UART1_INTPRI    6
#define DEVCNF_UART2_INTPRI    6
#define DEVCNF_UART3_INTPRI    6
```

```

/* Debug option
 *      Specify the device used by T-Monitor.
 *      -1 : T-Monitor does not use serial devices
 */
#if USE_TMONITOR
#define DEVCNF_SER_DBGUN      0          // Used by T-Monitor
#else
#define DEVCNF_SER_DBGUN      -1         // T-Monitor not executed
#endif

```

リスト 1 SBK-M4MN 用のシリアル通信ドライバの定義

- DEVCNF_UARTx_INTPRI (x=0~3)


UARTx(x=0~3)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 UART 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 UART の使用を可能とする。

 シリアル通信の設定は T-Monitor と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実行されるドライバの設定が有効となる。

9.1.4. システムの初期化

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 UART0 を使用するためには UART 用の初期化が必要となる。

本デバイスドライバでは、初期タスクから呼び出されるサンプルデバイスドライバの初期化処理において、UART0 用の端子の初期化処理を実行している。

具体的には、本実装では初期化時に以下の処理を実行している。

- (1) UART0 の送信割込、受信割込、エラー割込の割込み優先順位を設定
- (2) UART0 用のシステムクロックの設定
- (3) UART0 用の GPIO の設定
- (4) UART0 のパラメータを設定

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/m4mn/ser_mode_sysdep.h : 通信エラーの定義

```
/* Communication Error */
#define DEV_SER_ERR_OE    (0x00000008)    /* Overrun Error  UART_ERR_OVRERR_ERR */
#define DEV_SER_ERR_BE    (0x00000001)    /* Break Error    UART_ERR_BERR_ERR */
#define DEV_SER_ERR_PE    (0x00000004)    /* Parity Error   UART_ERR_PERR_ERR */
#define DEV_SER_ERR_FE    (0x00000002)    /* Framing Error  UART_ERR_FERR_ERR */
```

リスト 2 SBK-M4MN 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。

uct_mtk3_bsp/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>
#include <sysdepend.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、`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 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

`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。

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;

#ifdef 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 (UART0)
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

SBK-M4MN では、ターゲットマイコン(TMPM4MNFYAFG)の UART0 を USB2 経由で PC にシリアルポートとして接続することができる。そこで、本実装では UART0 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 USB1 を CMSIS-DAP として、USB2 を UART0 として PC と接続する(図 1 参照)。

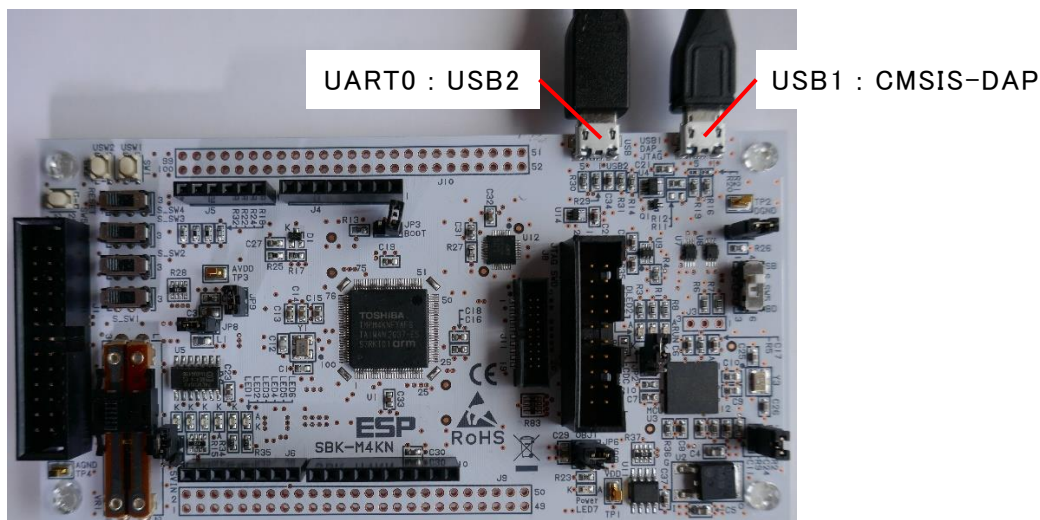


図 1 UART と CMSIS-DAP を USB で接続

PC 側では SBK-M4MN に対応する COM ポートを指定して通信ソフトを起動し、表 10-2 のパラメータ(通信速度、データ形式)を指定することで μ T-Kernel のコンソールとして利用できる。

通信速度やパラメータは `tm_com.c` のマクロの定義で指定可能な実装にしてある。

`lib/libtm/sysdepend/sbk_m4mn/tm_com.c` : 通信速度、パラメータの指定

```

/*-----*/
/*      Initialize UART for T-Monitor (CONSOLE)      */
/*-----*/
#define UART_BAUD      (115200)
#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

```

本実装で対応している通信速度は以下のとおりである。

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

また、設定可能なパラメータは以下のファイルでマクロ定義されている。

`device/ser/sysdepend/m4mn/ser_sysdep.h`

本実装ではサンプルのシリアル通信ドライバも同じ `UART0` を利用している。

このため、`DEVICE_USE_SER` が有効(1)の場合は、後から設定されるシリアルドライバの通信速度とパラメータがパラメータが有効となる。

サンプルのシリアル通信ドライバに関しては以下を参照のこと。

- ・「μ T-Kernel 3.0 共通デバイスドライバ説明書」
- ・「9.1. シリアル通信ドライバ(UART)」

μ T-Kernel 3.0

実装仕様書 (SBK-M4MN)

Rev 3.00.01 (April, 2023)

ユーシーテクノロジー株式会社

141-0031 東京都品川区西五反田 2-12-3 第一誠実ビル 9F

©2023 Ubiquitous Computing Technology Corporation All Rights Reserved.