μT-Kernel3.0
STM32L4 IoT-Engine 向け
実装仕様書

Version.01.00.03

2022.06.30

変更履歴

版数（日付）	内　容
1.00.03 (2022.06.30)	● 3.3 OS のメモリマップ 誤記修正 内蔵 ROM 先頭アドレス（誤)0x00000000（正)0x08000000 ● 5.2 ハードウェアの初期化および終了処理 各表の設定内容を更新 cpu_clock.c ファイルのパス変更
1.00.02 (2021.11.15)	● 1.4 関連ドキュメント バージョン番号等を更新 ● 誤記修正（正)RNG（誤)RING
1.00.01 (2021.03.31)	● 1.4 関連ドキュメント バージョン番号等を更新 ● 4. 割込みおよび例外 説明の見直 設定可能な優先度の誤記修正（正）優先度 2～14 SetCpuIntLevel および GetCpuIntLevel について、誤記修正（マスクレベル 0 指定時の動作）および INTLEVEL_EI、INTLEVEL_DI に関する説明を追加 ● 8.3 FPU 使用の際の注意点 新規追加
1.00.00 (2020.12.09)	● 初版

目次

1. 概要

1.1 目的

本書はSTM32L4 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(V3.00.05)のSTM32L4 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「T-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	STM32L4 IoT-Engine	UCテクノロジー製
搭載マイコン	STM32L486VG	STマイクロエレクトロニクス製

1.3 ターゲット名

STM32L4 IoT-Engineのターゲット名および識別名を以下とする。

分類	名称	対象
ターゲット名	_IOTE_STM32L4_	
対象システム	IOTE_STM32L4	STM32L4 IoT-Engine
対象CPU	CPU_STM32L486	STM32L486
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¥iote_stm32l4¥machine.h

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(Ver.3.00.00)	TRON フォーラム TEF020-S004-3.00.00
	μT-Kernel3.0 共通実装仕様書 (Ver.1.00.08)	TRON フォーラム TEF033-W002-211115
T-Monitor	T-Monitor 仕様書	TRON フォーラム TEF-020-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.1.00.2)	TRON フォーラム TEF033-W007-210331
搭載 マイコン	RM0351 リファレンスマニュアル	ST マイクロエレクトロニクス
	PM0214 プログラミング・マニュアル	
	STM32L486xx データシート	

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_stm32l4                 STM32L4 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
   ├ stm32l4                        STM32L4 マイコン依存部
   ├      :
   └ <CPU n>                      CPUn 依存部
     └ core                      コア依存部
       ├ armv7m                  ARMv7-M コア依存部
       ├     :
       └ <core n>                      コア n 依存部
```

「ARMv7M コア依存部」は、ARMv7M 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。

「STM32L4 マイコン依存部」は、前述のコア依存部以外の本マイコンに固有のコードである。

「STM32L4 IoT-Engine 依存部」は、前述のコア依存部およびマイコン依存部以外の STM32L4 IoT-Engin のハードウェアに固有のコードである。

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-M4F
ROM	1MB(内蔵フラッシュ ROM)
RAM	128KB(内蔵 RAM、SRAM1 96KB　SRAM2 32KB) ※本実装では OS は SRAM1 のみを使用する。

2.2 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。

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。

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_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_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 を TURE に指定すると、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　（「8.1.1 FPU の初期設定」参照）
- TA_FPU 属性のタスク　（「6.3 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（「8.1.2FPU 関連 API」参照）

3. メモリ

3.1 メモリモデル

ARM 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。

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-M4F 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-M4F 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 （上段：開始 下段：終了）	種別	サイズ (KByte)	備考
0x0800 0000 0x080F FFFF	内蔵 ROM	1024	Code 領域 内蔵フラッシュ ROM
0x1000 0000 0x1000 7FFF	SRAM2	32	CodeM 領域 内蔵 SRAM ※1
0x2000 0000 0x2001 7FFF	SRAM1	96	SRAM 領域 内蔵 SRAM
0x4000 000 0x5FFF FFFF	Peripheral		Peripheral 領域
0xE000 0000 0xFFFF FFFF	CPU 内レジスタ領域		

※1 SRAM2 領域は本実装では OS は管理、使用しない。ユーザプログラムから使用可能である。

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

本実装では、マイコンの内蔵 ROM および SRAM1 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 SRAM1 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、SRAM1 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 SRAM1 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※ （上段：開始 下段：終了）	種別	内容
0x0800 0000 -	例外ベクタ テーブル	例外や割込みのベクタテーブル リセット時のみ有効
- -	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
- -	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※ （上段：開始 下段：終了）	種別	内容
0x2000 0000 -	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
- -	プログラムデータ	C 言語の変数等が配置される領域
- -	OS 管理領域	OS 内部の動的メモリ管理の領域
- 0x2001 7FFF	例外スタック 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 の API の処理において以下のメモリが動的に確保される。

- 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

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は、動的メモリ管理は行われない（初期値は動的メモリ管理を行う）。

OS 内の動的メモリ管理に使用する OS 管理メモリ領域（システムメモリ領域）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。

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
16～97	IRQ 割込み#0 ～ #81	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¥stm32l4¥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/armv7m/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-M4F 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しているため、同様にユーザプログラムから使用は許されない。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。

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/stm32l4/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           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。

4.5 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)は外部割込み(IRQ0～81)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「4.8 その他の例外」を参照のこと）。

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¥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出力に出力される（「8.2.2 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 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。

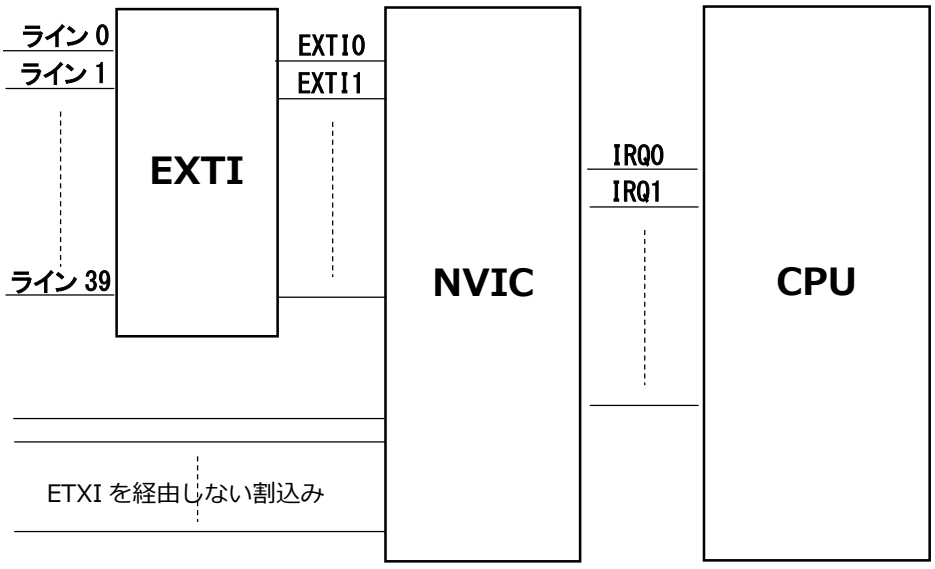
4.6　μT-Kernel/SM の割込み管理機能

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の制御を行う。

4.6.1　割込みコントローラの概要

STM32L4 マイコンには、割込みコントローラとして NVIC と EXTI(拡張割込み/イベントコントローラ)が内蔵されている。
NVIC は ARM Cortex-M に標準の割込みコントローラである。EXTI は STM32 独自の割込みコントローラであり、NVIC にカスケード接続される。
μT-Kernel/SM の割込み管理機能は、NVIC と EXTI の両割込みコントローラの制御に対応する。

NVIC、EXTI、CPU の関係の概略を以下の図に示す（詳細はマイコンのマニュアルを参照のこと）。



EXTI は入力として 40 本の割込み/イベントライン（ライン 0～39）があり、各ラインについて割込みのマスク、トリガ条件が設定可能である（トリガ条件設定の可否はラインによる）。
EXTI からの出力は NVIC に対応する割込みとして入力される。NVIC には EXTI 経由以外の割込みも入力されている。NVIC は各入力について割込みのマスク、優先度が設定可能である。

EXTI を経由する割込みを有効にするには、EXTI および NVIC の両方で割込みのマスクを解除しなくてはならない。
ただし、EXTI のライン 17、23～34、39 はリセット時にマスクが解除されている。またこれらのラインはトリガ条件の設定ができないので、基本的には EXTI を操作する必要はない。

4.6.2　割込み番号

μT-Kernel/SM の割込み管理機能では、μT-Kernel/OS の割込み管理機能の割込み番号に加えて、EXTI を制御するために EXTI の各ラインに割込み番号の 200 以降を割り当てる。たとえば、EXTI のライン 0 が割込み番号 200、ライン 1 が割込み番号 201 となる。

割込み番号 200 以降はμT-Kernel/SM の割込み管理機能でのみ使用可能である。この番号に割込みハンドラを定義することはできない（割込みハンドラを定義したい場合は、最終的に割り付けられた IRQ 割込みにハンドラを定義する）。

4.6.3　割込みの優先度

割込みの優先度は、2 から 14 が使用可能である。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。

4.6.4　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 から 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-Kernel3.0 仕様の基づき、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。

4.6.5　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 および EXTI の制御を行う。
各 API の実装を以降に記す。

①　割込みコントローラの割込み許可(EnableInt)
IRQ 割込み（割込み番号 0～81）の場合、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。
EXTI ソース（割込み番号 200～239）の場合、EXTI の割込みマスクレジスタ(EXTI_IMR)を設定し、指定された割込みを許可する。EXTI は割込み優先度の設定はできないので、指定した優先度は無視される。

②　割込みコントローラの割込み禁止(DisableInt)
IRQ 割込み（割込み番号 0～81）の場合、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。
EXTI ソース（割込み番号 200～239）の場合、EXTI の割込みマスクレジスタ(EXTI_IMR)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)
IRQ 割込み（割込み番号 0～81）の場合、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。
EXTI ソース（割込み番号 200～239）の場合、EXTI のペンディングレジスタ(EXTI_PR)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)は何も実行しないマクロとして定義される。

⑤　割込み発生の検査(CheckInt)
IRQ 割込み（割込み番号 0～81）の場合、割込みコントローラ(NVIC)の割込み保留クリアレジスタ(ICPR)を参照し、割込みの発生を調べる。
EXTI ソース（割込み番号 200～239）の場合、EXTI のペンディングレジスタ(EXTI_PR)を参照し、割込みの発生を調べる。

⑥　割込みモード設定(SetIntMode)
割込みモードは、立ち上りエッジ(IM_HI)、立ち下りエッジ(IM_LOW)、または立ち上り立下り両エッジ(IM_BOTH)が指定可能である。
指定された割込みモードに応じて、EXTI の立ち上がりトリガ選択レジスタ(EXTI_RTSR)と立ち下りトリガ選択レジスタ(EXTI_FTSR)を設定する。
指定可能な割込みは、EXTI ソース（割込み番号 200～239）のみである。IRQ 割込み（割込み番号 0～81）を指定した場合は何も実行しない。

指定可能な割込みモードは、include/tk/sysdepend/cpu/stm32l4/syslib.h に以下のように定義されている。

```
#define IM_EDGE   0x0000        /* Edge trigger */
#define IM_HI     0x0002        /* Interrupt at rising edge */
#define IM_LOW    0x0001        /* Interrupt at falling edge */
#define IM_BOTH   0x0003        /* Interrupt at both edge */
```

なお、IM_EDGE は互換性のために定義している。たとえば、立ち上りエッジを指定する場合に、IM_EDGE | IM_HI のように記述することができる。

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。

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/armv7m/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出力に出力される（「8.2.2 コンソール入出力」参照)。

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。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。

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ARMv7M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。詳細は「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) システムメモリ領域の確保
システムメモリ領域の確保を行う。
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、本処理は行われない（詳細は「3.5 OS内の動的メモリ管理」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の有効化
コンフィギュレーションUSE_FPUが指定されている場合、FPUの有効化を行う。

(7) 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て処理を実装する。ただし、コンフィギュレーションUSE_SDEV_DRVを有効とすることにより、基本的なデバイスデバイスの初期化を行うことができる。基本的なデバイスドライバについては、「uTK3.0デバイスドライバ説明書」を参照のこと。

コンフィギュレーションUSE_SDEV_DRVが有効(1)の場合、基本的なデバイスデバイスの初期化が行われる。無効(0)の場合は、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行われる(初期値は無効)。

本実装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。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_stm32l4/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。

knl_startup_hw関数では以下のハードウェアの初期設定が行われる。

(1) システムクロックの初期化

システムクロックの初期化は以下の関数を呼び出すことにより行われる。

ファイル：kernel/sysdepend/cpu/stm32l4/cpu_clock.c
関数名：startup_clock

本実装ではクロックは以下に設定される。

項目	内容
PLL 設定	クロックソース　PLL(HSE) メイン PLL　M = 2, N = 40, P = 7, Q = 2, R =2 PLLSAI1　N = 32, P = 7, Q = 2, R =2 PLLSAI2　N = 16, P = 7, R =2
クロック周波数	SYSCLK ：　80MHz HCLK(AHB)：　80MHz PCLK1(APB1)：80MHz PCLK2(APB2)：80MHz

クロックおよび PLL の分周設定は以下のように定義されている。

ファイル名：kernel¥sysdepend¥iote_stm32l4¥cpu_clock.c

```
#define RCC_CFGR_INIT            (0x00000000)
#define RCC_PLLCFGR_INIT         (0x00002810)
#define RCC_PLLSAI1CFGR_INIT     (0x00002000)
#define RCC_PLLSAI2CFGR_INIT     (0x00001000)
```

上記の定義は、RCC(リセットおよびクロック制御)のクロック設定レジスタ(RCC_CFGR)、PLL 設定レジスタ(RCC_PLLCFGR, PLLSAI1CFGR, PLLSAI2CFGR)の各分周の設定に使用される。
なお、クロックおよび PLL の初期設定を変更した場合は、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を行わなくてはならない。

(2) ペリフェラルへのクロック供給の有効化
使用するペリフェラルへのクロック供給を有効化する。
設定内容は変数 modclk_tbl に記述されている。この値を変更することにより、ペリフェラルへのクロック供給の初期設定を変更することが可能である。

初期値では以下のペリフェラルへのクロック供給が有効となっている。

● コンフィギュレーション USE_SDEV_DRV が無効(0)の場合

名称	機能	用途
SYSCFG	システム設定	
GPIO D	汎用ポート	端子設定
USART2	シリアル通信	デバッグ用シリアル入出力
TIM2,TIM3,RIM4,TIM5	汎用タイマ	物理タイマ

● コンフィギュレーション USE_SDEV_DRV が有効(1)の場合

名称	機能	用途
SYSCFG	システム設定	
GPIO A～D	汎用ポート	端子設定
USART2	シリアル通信	デバイスドライバ、他
TIM2,TIM3,RIM4,TIM5	汎用タイマ	物理タイマ

(3) 端子機能の設定

各端子の機能設定を初期化する。

各レジスタの初期値は、変数 pinfnc_tbl に記述されている。この値を変更することにより、各端子の初期設定を変更することが可能である。

初期値では以下の端子の機能が設定されている。

● コンフィギュレーション USE_SDEV_DRV が無効(0)の場合

I/O ポート名	機能名	用途
PD5	USART2_TX	デバッグ用シリアル入出力(T-Monitor)
PD6	USART2_RX	

● コンフィギュレーション USE_SDEV_DRV が有効(1)の場合

I/O ポート名	機能名	用途
PD5	USART2_TX	デバイスドライバ(シリアル通信)、
PD6	USART2_RX	デバッグ用入出力
PA0	ADC12_IN5	デバイスドライバ (A/D コンバータ)
PA1	ADC12_IN6	
PA2	ADC12_IN7	

PA4	ADC12_IN9	
PB1	ADC12_IN16	
PB8	I2C1_SCL	デバイスドライバ（I²C）
PB9	I2C1_SDA	
PB10	I2C2_SCL	
PB11	I2C2_SDA	
PC3	A/DC123_IN4	デバイスドライバ（A/D コンバータ）
PC4	A/DC12_IN13	
PD9	GPIO out	I²C イネーブル信号
PD11	GPIO out	オンボード LED
PD15	GPIO out	オンボード LED

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、以下の関数にてデバイスドライバの登録、実行、終了を行う。関数の仕様については、共通実装仕様書も参照のこと。

ファイル：kernel/sysdepend/iote_stm32l4/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 が有効(1)の場合、以下の基本的なデバイスドライバが knl_start_device で登録される。

デバイス名	機能
serb	シリアル通信 (USART2)
iica	I²C 通信 (I2C1)
adca	A/D コンバータ (ADC1)

コンフィギュレーション USE_SDEV_DRV 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処理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。

6. タスク

6.1 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_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)

6.2 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。

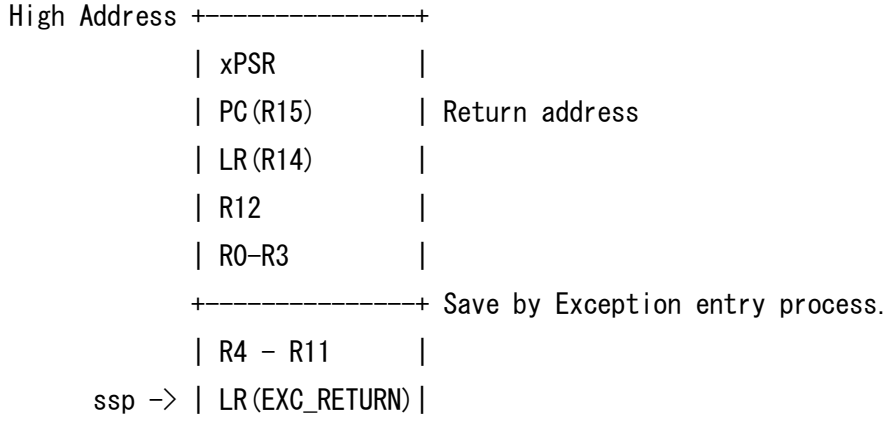
レジスタ	値	補足
PRIMASK	0	割込み許可
R0	第一引数 stacd	タスク起動コード
R1	第二引数 *exinf	タスク拡張情報
R13(MSP)	タスクスタックの先頭アドレス	

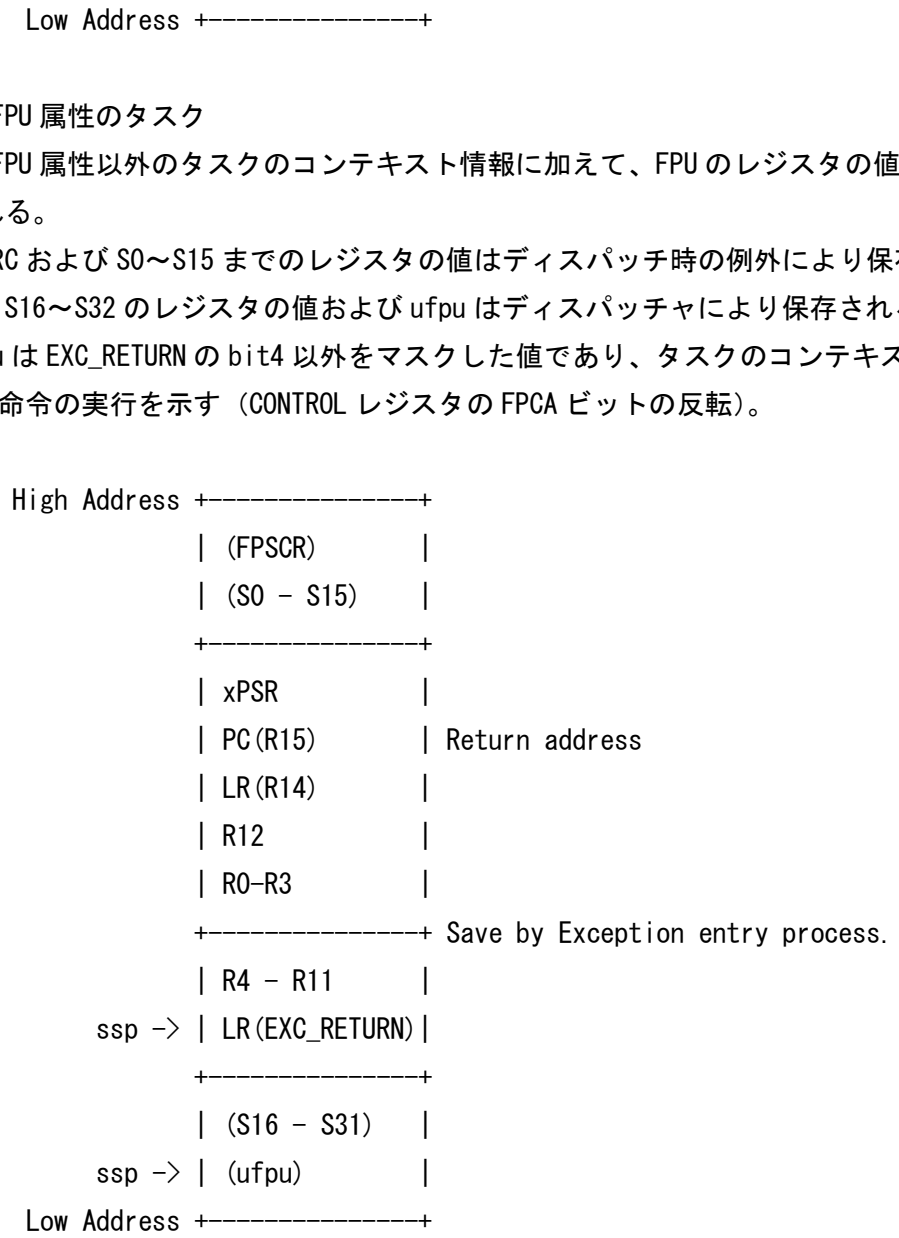
6.3 タスク・コンテキスト情報

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）。

7. 時間管理機能

7.1 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。

7.2 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。

include/sys/sysdepend/cpu/stm32l4/sysdef.h

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
```

7.3 物理タイマ機能

7.3.1 使用するハードウェアタイマ

マイコン内蔵の汎用タイマ（TIM2～TIM5）を使用して 4 個の物理タイマが実装されている。TIM2 および TIM5 は 32 ビットタイマ、TIM3 および TIM4 は 16 ビットタイマである。
汎用タイマには以下のよう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	TIM2
2	TIM3
3	TIM4
4	TIM5

汎用タイマ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）。

7.3.2 タイマの設定

物理タイマのクロック設定は、include/sys/sysdepend/cpu/stm32l4¥sysdef.h に以下のように定義される。

```
#define TIM2PSC_PSC_INIT        0
#define TIM3PSC_PSC_INIT        0
#define TIM4PSC_PSC_INIT        0
#define TIM5PSC_PSC_INIT        0
```

この値は、汎用タイマの TIMx_PSC レジスタにプリスケーラ値として設定される。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。
本実装の初期値では、カウンタクロック周波数にタイマクロックを未分周で使用する。また、タイマクロックは PCLK1（80MHz）と同じ周波数である。

7.3.3　タイマ割込み

物理タイマはその内部処理において、各タイマの割込みを使用する。

物理タイマ番号	対応する割込み	割込み番号
1	TIM2 グローバル割込み	28
2	TIM3 グローバル割込み	29
3	TIM4 グローバル割込み	30
4	TIM5 グローバル割込み	50

各割込みの優先度は include/sys/sysdepend/cpu/stm32l4/sysdef.h に以下のように定義される。

```
#define INTPRI_TIM2     5       // 物理タイマ 1
#define INTPRI_TIM3     5       // 物理タイマ 2
#define INTPRI_TIM4     5       // 物理タイマ 3
#define INTPRI_TIM5     5       // 物理タイマ 4
```

割込み優先度は必要に応じて変更可能である。

8. その他の実装仕様

8.1 FPU 関連

8.1.1 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。

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。

8.1.2 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。

```
typedef struct t_copregs {
        VW      s[32];  /* FPU General purpose register S0-S31 */
        UW      fpscr;  /* Floating-point Status and Control Register */
} T_COPREGS;
```

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）。

8.1.3　FPU 使用の際の注意点

ユーザのプログラム中において 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 属性以外のタスクは、ディスパッチの際に FPU レジスタの値を退避しないため、FPU レジスタの内容が破壊される可能性がある。
  なお、TA_FPU 属性の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は退避する 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「6.3 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 属性以外のタスクにおける FPU の使用が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、すべてのタスクを TA_FPU 属性とすることである。

8.2　T-Monitor 互換ライブラリ

8.2.1　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。

8.2.2　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仕様を以下に示す。

項目	内容
デバイス	内蔵 UART (USART2)
ボーレート	115200bps
データ形式	data 8bit, stop 1bit, no parity

以上