

μ T-Kernel 3.0 共通実装&構成仕様書

Rev 3.00.01

May, 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. 本書について	5
2.2. 関連資料.....	5
2.3. 実装の基本方針.....	5
2.4. バージョン情報.....	6
2.5. ターゲット名	6
2.5.1. ModusToolbox でのターゲット名の指定方法.....	8
2.5.2. EWARM でのターゲット名の指定方法	8
2.6. 関連ドキュメント	9
2.6.1. トロンフォーラムが提供しているドキュメント	9
2.6.2. UCT が提供するドキュメント	10
2.7. ディレクトリ構成	11
2.7.1. 全体構成	11
2.7.2. 機種依存定義部	11
2.7.3. アプリケーション依存部.....	12
2.7.4. その他	13
3. 基本実装仕様.....	15
3.1. 対象マイコン	15
3.2. 実行モードと保護レベル	15
3.3. CPU レジスタ	15
3.4. 低消費電力モードと省電力機能	15
3.5. コプロセッサ対応	15
4. メモリ	16
4.1. メモリモデル	16
4.2. マイコンのアドレス・マップ	16
4.3. OS のメモリマップ	16
4.4. スタック	17
4.4.1. タスクスタック	17
4.4.2. 例外スタック	17
4.4.3. テンポラリスタック	17
4.5. OS 内の動的メモリ管理	17
4.6. システムメモリ管理機能	18
5. 割込みおよび例外	19

5.1.	マイコンの割込みおよび例外	19
5.2.	ベクタテーブル	19
5.3.	割込み優先度とクリティカルセクション	19
5.3.1.	割込み優先度	19
5.3.2.	多重割込み対応	19
5.3.3.	クリティカルセクション	19
5.4.	OS 内部で使用する割込み	19
5.5.	μT-Kernel/OS の割込み管理機能	21
5.6.	μT-Kernel/SM の割込み管理機能	21
5.7.	OS 管理外割込み	21
6.	起動および終了処理	22
7.	サービスプロファイル	23
7.1.	有効・無効を示すプロファイル	23
7.2.	値を持つサービスプロファイル	27
8.	タスク	29
8.1.	タスク属性	29
8.2.	タスク優先度	29
8.3.	タスクの処理ルーチン	30
8.4.	タスクのスタック	31
9.	時間管理機能	32
9.1.	システム時刻管理	32
9.1.1.	システムタイマ	32
9.1.2.	マイクロ秒の時間管理	32
9.2.	タイムイベントハンドラ	32
9.2.1.	タイムイベントハンドラ属性	32
9.2.2.	タイムイベントハンドラの実行状態	33
10.	その他の実装仕様	34
10.1.	システムコール	34
10.2.	サブシステム	34
10.3.	μT-Kernel 2.0 互換機能	34
10.4.	物理タイマ機能	34
10.5.	デバイスドライバ	35
11.	デバッグサポート機能	36
11.1.	デバッグサポート機能の有効化	36
11.2.	対応するデバッグ機能	36
12.	T-Monitor 互換 API	39
12.1.	T-Monitor 概要	39
12.2.	T-Monitor 互換 API の有効化	39
12.3.	対応 API	39

12.4.	T-Monitor のコンフィギュレーション	40
13.	コンフィギュレーション	42
13.1.	基本コンフィギュレーション	42
13.2.	機能コンフィギュレーション	47
13.2.1.	機能単位	47
13.2.2.	API 単位	48
13.3.	その他のコンフィギュレーション	51
13.3.1.	T-Monitor 関連コンフィギュレーション	51
13.3.2.	デバイスドライバ関連コンフィギュレーション	51

1. はじめに



1.1. 本書について

本書はユーシーテクノロジー株式会社(Ubiquitous Computing Technology Corporation、以下 UCT と称する)が開発した μ T-Kernel 3.0 用のボードサポートパッケージ(BSP:BoardSupportPackage)に共通する実装と構成に関する仕様書である。

本書では、UCT が公開する μ T-Kernel 3.0 BSP(BoardSupportPackage)に含まれている実装のうち、UCT が BSP を開発したターゲットボード用の μ T-Kernel 3.0 のみを対象とする。トロンフォーラムが BSP を開発したターゲットボードは対象外である。また、UCT が移植を行ったターゲットボードであっても、トロンフォーラムの管理下に移行した BSP(メインストリームに取込まれた BSP)は対象外となる。

以降、単に OS、RTOS、μ T-Kernel 3.0 と称する場合は UCT が BSP を実装したターゲットボードに対する μ T-Kernel 3.0(共通部を含む)を示すものとする。

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. 本書について

本書は、 μ T-Kernel 3.0 に共通するプロファイルやコンフィギュレーション、実装に関する仕様について記載した仕様書である。

対象は、UCT が BSP を開発したターゲットボード用の μ T-Kernel 3.0 である。トロンフォーラムが BSP を開発したターゲットボード用の μ T-Kernel 3.0 は含まない。

2.2. 関連資料

本資料に全ての内容が記載されているわけではない。本資料を参照する際には、以下の資料を合わせて参照すること。

- (a) 「 μ T-Kernel 3.0 仕様書」(トロンフォーラム)
 μ T-Kernel 3.0 のリアルタイム OS としての仕様を規定している。
- (b) ターゲットボードの実装仕様書 (UCT)
ターゲットボードに固有の実装仕様について記載されている。
- (c) ターゲットボードの構築手順書 (UCT)
ターゲットボード毎の構築手順について説明している。
- (d) 「 μ T-Kernel 3.0 共通実装仕様書」(トロンフォーラム)
 μ T-Kernel 3.0 に共通する実装仕様について記載されている。
本書や(b)に記載されていない内容については「 μ T-Kernel 3.0 共通実装仕様書」の記載に従うものとする。

- ❶ 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 のトロンフォーラムのリポジトリで公開されている。
また、UCT が発行するドキュメントは、UCT が GitHub で公開する μ T-Kernel 3.0 用 BSP のリポジトリで公開している。
資料の具体的な入手先については「2.6. 関連ドキュメント」を参照のこと。
- ❷ ターゲットボードや CPU に関する資料は、ターゲットボードメーカーや CPU メーカーが発行するドキュメントを参照すること。

2.3. 実装の基本方針

UCT による μ T-Kernel 3.0 BSP の実装に関する基本方針は以下のとおりである。

- アドレス空間は、単一の物理アドレスのみに対応し、MMU を用いた仮想アドレスやメモリ保護などには対応しない。
- OS および OS 上で実行されるプログラムは、実行モードを特権モードのみとし、静的にリンクされた一つの実行オブジェクトとする。これを前提としているため

API は関数呼び出しのみに対応する。

- プログラムは C 言語、またはアセンブリ言語で記述し、原則として他の言語処理系は使用しない。
- CPU メーカーが提供する開発環境やツール、ミドルウェア、BSP がある場合はそれらを利用して開発する場合がある。このため、ターゲットボードに依存した設定を変更する場合に専用のツールの利用が必要となる場合がある。

例)

- ・ ModusToolbox (Infineon Technologies AG)

2.4. バージョン情報

本実装のバージョン情報を以下に示す。この情報は `tk_ref_ver` で取得される。

表 2-1 バージョン情報

種別	変数	値	備考
メーカーコード	<code>maker</code>	0x0000	トロンフォーラム
識別番号	<code>prid</code>	0x0000	トロンフォーラム
仕様書バージョン番号	<code>spver</code>	0x6300	μ T-Kernel 3.0 仕様
カーネルバージョン番号	<code>prver</code>	0x0003	
製品管理番号	<code>prno[4]</code>	0x0000	全て 0

これらの値は μ T-Kernel 3.0 全体の実装に対する情報である。このため、UCT が開発した BSP においてもフォーラム版と同じ設定にしてある。

2.5. ターゲット名

OS を構築するにあたって、ターゲットボード毎に固有のターゲット名および派生する識別名を定める。これらは、機種依存部の条件コンパイルの識別子として使用される。

ターゲット名はソースコードの中には記載されない。開発ツールにおいて指定する必要がある。例えば、コンパイラのオプションや統合開発環境の設定においてターゲット名を指定する。(具体的なターゲット名の指定方法については後述)

ソースコードでは、ターゲット名から派生して他の識別名が定義される。この定義は以下のファイルに記述される。

```
include/sys/machine.h
include/sys/sysdepend/<TARGET>/machine.h
include/sys/sysdepend/cpu/<CPU>/machine.h
include/sys/sysdepend/cpu/core/<CORE>/machine.h
```

include/sys/machine.h では、開発ツールにおいて指定されたターゲット名に従ってインクルードするファイルを決定する。

他では、ターゲットボード、CPU、CPU コアに対応した他の識別子を定義する。

以下は CY8CKIT-062S2-43012 での実装例である。

```
include/sys/machine.h
#ifdef _CY8CKIT_CY8C6_
#include "sysdepend/cy8ckit_cy8c6/machine.h"
#define Csym(sym) sym
#endif
```

リスト 1 CY8CKIT-062S2-43012 用の定義ファイルのインクルードを追加

```
include/sys/sysdepend/cy8ckit_cy8c6/machine.h
#define CY8CKIT_CY8C6 1 /* Target system : CY8CKIT-062S2-43012 */
#define CPU_CY8C6 1 /* Target CPU : CY8C6 series */
#define CPU_CY8C624A 1 /* Target CPU : CY8C624A */
#define CPU_CORE_ARMV7M 1 /* Target CPU-Core type : ARMv7-M */
#define CPU_CORE_ACM4F 1 /* Target CPU-Core : ARM Cortex-M4 */

#define TARGET_DIR cy8ckit_cy8c6 /* Sysdepend-Directory name */
#define TARGET_CPU_DIR cy8c6 /* Sysdepend-CPU-Directory name */
```

リスト 2 CY8CKIT-062S2-43012 用の識別子の定義

```
include/sys/sysdepend/cpu/cy8c6/machine.h
#define TARGET_CPU_DIR cy8c6 /* Sysdepend-CPU-Directory name */
```

リスト 3 CY8C624ABZI-S2D44 用の識別子の定義

```
include/sys/sysdepend/cpu/core/armv7m/machine.h
#define ALLOW_MISALIGN 0
#define INT_BITWIDTH 32
#define BIGENDIAN 0 /* Default (Little Endian) */
```

リスト 4 ARMv7-M 用の識別子の定義

- ① ターゲットボード CY8CKIT-062S2-43012 に搭載された CPU が CY8C624ABZI-S2D44 であり、μ T-Kernel 3.0 はこの CPU の Cortex-M4 コア (ARMv7-M) に対して実装されている。

主なターゲットボードのターゲット名は以下のとおりである。

表 2-2 ターゲット名

ターゲットボード	ターゲット名	備考
CY8CKIT-062S2-43012	_CY8CKIT_CY8C6_	
SBK-M4KN	_SBK_M4KN_	

- ① 上記以外のターゲット名については、各ターゲットボードの「μ T-Kernel 3.0 実

装仕様書」を参照のこと。

2.5.1. ModusToolbox でのターゲット名の指定方法

CY8CKIT-062S2-43012 用の開発環境である ModusToolbox では、Makefile の変数 `DEFINES` にターゲット名を指定する。

Makefile

```
# Add additional defines to the build process (without a leading -D).
DEFINES=_CY8CKIT_CY8C6_
```

リスト 5 ModusToolbox でのターゲット名の指定方法

2.5.2. EWARM でのターゲット名の指定方法

EWARM では、プロジェクトのオプションでターゲット名を指定する。

- (1) メニューバー → [プロジェクト] → [オプション]
- (2) [C/C++コンパイラ] → [プリプロセッサ]タブ → [シンボル定義]
- (3) [アセンブラ] → [プリプロセッサ]タブ → [シンボル定義]

図 1 と図 2 は、SBK-M4KN での設定例である。

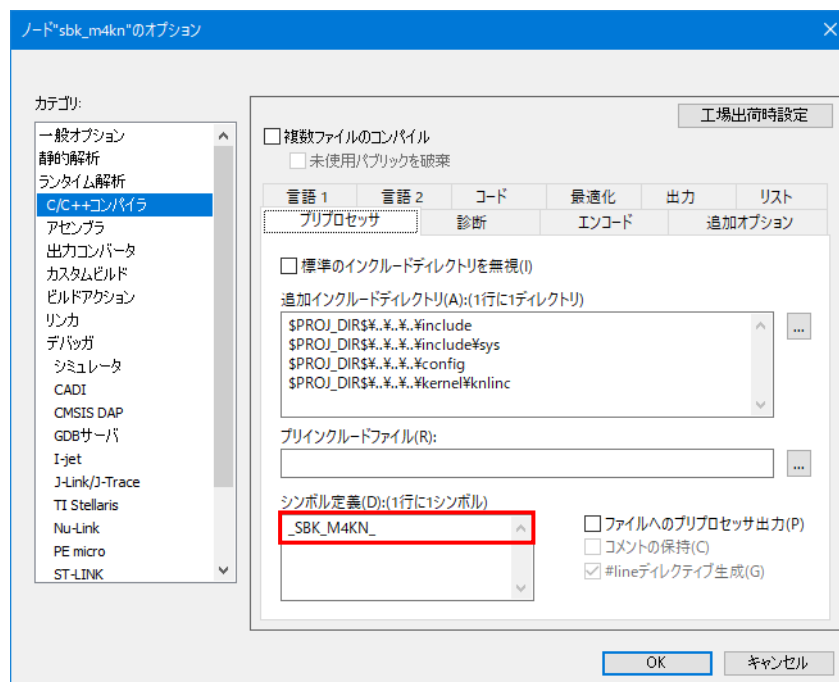


図 1 C/C++コンパイラ用にターゲット名を指定

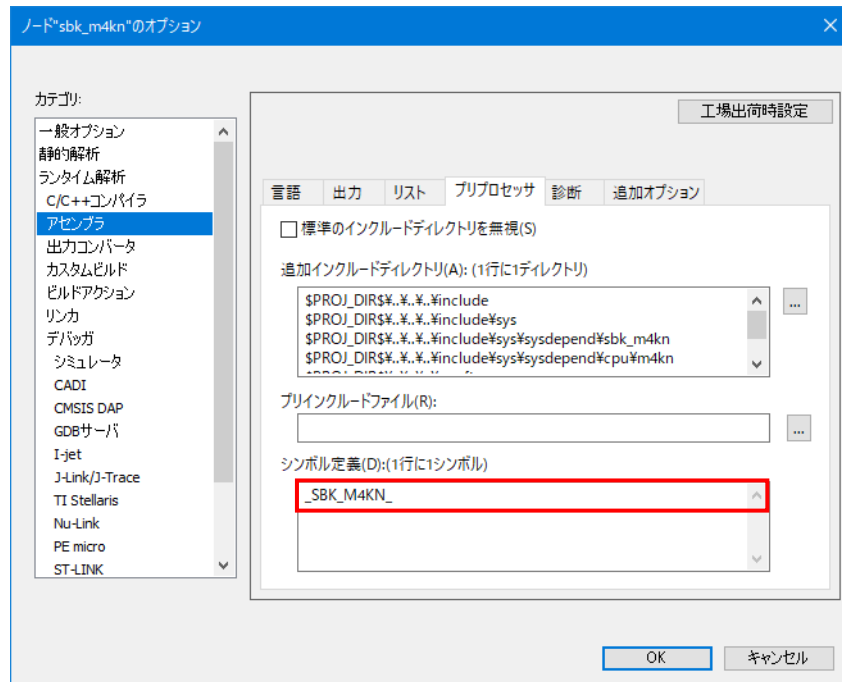


図 2 アセンブラ用にターゲット名を指定

- 💡 ターゲット名はビルドターゲット毎に設定する必要がある。
 また、C/C++コンパイラ用とアセンブラ用の2箇所を設定する必要がある。
 設定抜けがないように注意すること。

2.6. 関連ドキュメント

リアルタイム OS としての標準的な仕様は「μ T-Kernel 3.0 仕様書」に記載される。
 また、ハードウェアに依存する実装仕様は、各ターゲットボード向けの「実装仕様書」と「構築手順書」に記載される。
 以下に関連するドキュメントを記す。

2.6.1. トロンフォーラムが提供しているドキュメント

トロンフォーラムが提供するドキュメントは以下の通りである。

- μ T-Kernel 3.0 仕様書
- μ T-Kernel 3.0 共通実装仕様書
- μ T-Kernel 3.0 デバイスドライバ説明書
- T-Monitor 仕様書

- ❗ T-Monitor は μ T-Kernel 仕様には含まれていない。

T-Monitor は本来 T-Kernel の様な比較的規模が大きいシステム用のモニタシステムであり、ハードウェアの初期化などの各種機能が用意されている。

そこで、μ T-Kernel では T-Monitor の機能のうちコンソール用機能のサブセットに限定して実装してある。コンソール機能の API を T-Monitor と共通にしてあることから、この部分を便宜上 T-Monitor と呼称している。

トロントフォーラムが発行するドキュメントは、トロントフォーラムの Web ページ、またはトロントフォーラムが GitHub で公開する μ T-Kernel 3.0 のリポジトリで公開されている。

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

https://tron-forum.github.io/mtk3_spec_jp/index.html

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

https://github.com/tron-forum/mtk3_bsp

https://github.com/tron-forum/mtk3_devenv

2.6.2. UCT が提供するドキュメント

UCT が提供するドキュメントは以下の通りである。

- μ T-Kernel 3.0 共通実装 & 構成仕様書（本書）
- μ T-Kernel 3.0 共通デバイスドライバ説明書
- μ T-Kernel 3.0 実装仕様書（CY8CKIT-CY8C6）
- μ T-Kernel 3.0 構築手順書（CY8CKIT-CY8C6）
- μ T-Kernel 3.0 実装仕様書（SBK-M4KN）
- μ T-Kernel 3.0 構築手順書（SBK-M4KN）
- μ T-Kernel 3.0 実装仕様書（SBK-M4MN）
- μ T-Kernel 3.0 構築手順書（SBK-M4MN）

これらのドキュメントは、UCT が GitHub で公開する μ T-Kernel 3.0 用の BSP のリポジトリで公開している。

https://github.com/UCTechnology/mtk3_bsp

2.7. ディレクトリ構成

μ T-Kernel 3.0 のディレクトリ構成を以下に示す。

2.7.1. 全体構成

μ T-Kernel 3.0 のディレクトリの全体構成は以下のとおりである。

mtkernel_3	μ T-Kernel 3.0 ベースディレクトリ
├── config	コンフィギュレーション
├── include	インクルードファイル
│ ├── sys	システム定義
│ │ └── sysdepend	ハードウェア依存部
│ ├── tk	OS 関連定義
│ │ └── sysdepend	ハードウェア依存部
│ └── tm	T-Monitor 関連定義
├── kernel	OS ソースコード
│ ├── knlinc	OS 内共通定義
│ ├── tstdlib	OS 内共通ライブラリ
│ ├── sysinit	初期化处理
│ ├── inittask	初期タスク
│ ├── tkernel	OS 機能
│ ├── sysdepend	ハードウェア依存部
│ └── usermain	ユーザメイン処理
├── lib	ライブラリ
│ ├── libtk	μ T-Kernel ライブラリ
│ │ └── sysdepend	ハードウェア依存部
│ └── libtm	T-Monitor 互換 API
│ └── sysdepend	ハードウェア依存部
├── device	デバイスドライバ(サンプル)
│ ├── include	インクルードファイル(デバイスドライバ関連)
│ ├── common	デバイスドライバの共通部
│ │ └── drvif	SDI(Simple Device driver I/F layer)
│ ├── ser	シリアル(UART)通信デバイスドライバ
│ │ └── sysdepend	ハードウェア依存部
│ ├── adc	AD コンバータデバイスドライバ
│ │ └── sysdepend	ハードウェア依存部
│ └── i2c	I ² C 通信デバイスドライバ
│ └── sysdepend	ハードウェア依存部
├── app_program	アプリケーション(サンプル)
├── build_make	GCC 用のビルドディレクトリ
├── docs	ドキュメント
└── etc	その他

❶ **イタリック体(斜体)**のディレクトリは、正確には μ T-Kernel 3.0 のソースコードには含まれない。

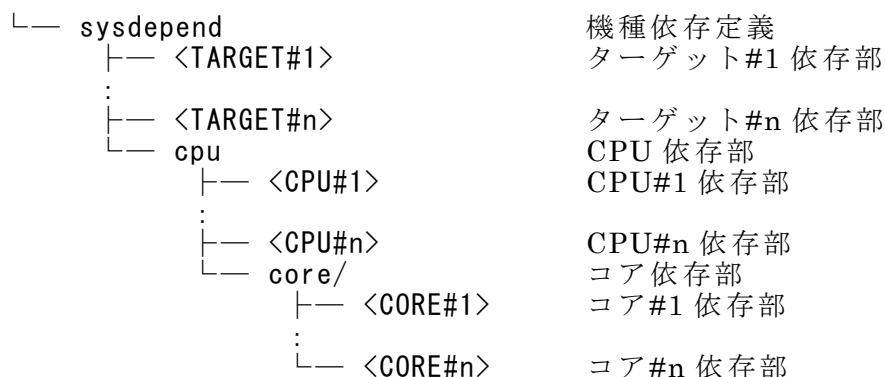
❶ GCC でビルドする際に利用するディレクトリは *build_make* に集められている。

2.7.2. 機種依存定義部

機種依存定義は各ディレクトリにある **sysdepend** に集められている。

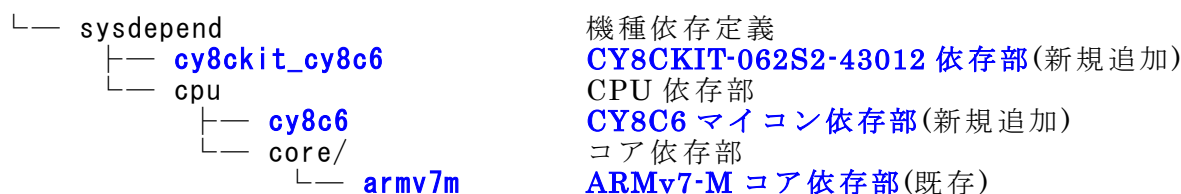
機種依存定義部は、ターゲットボードに依存して変更が必要となる部分である。通常はターゲットボード毎に用意する必要がある。ただし、CPU が既存の場合、CPU コアが既存の場合はそれぞれ既存の CPU や CPU コアを指定することで対応可能である(実装不要)。

sysdepend 以下の基本構成は以下のとおりである。



<TARGET#1>～<TARGET#n>、<CPU#1>～<CPU#n>、<CORE#1>～<CORE#n>にはそれぞれターゲットボードの識別子、CPU の識別子、コアの識別子が入る。これらの名称は「2.5. ターゲット名」で説明した `machine.h` において定義される。

例えば、CY8CKIT-062S2-43012 用の `sysdepend` は以下の構成となる。



青い太字のディレクトリが CY8CKIT-062S2-43012 用のディレクトリである。

cy8ckit_cy8c6 は新しいターゲットボードなので新規に追加した。

cy8c6 は対応実績のない CPU なので新規に追加した。

armv7m は対応実績のある CPU コアなので、既存のコードをそのまま利用した。

- ❶ 詳細については各ターゲットボードの実装仕様書も合わせて参照のこと。
- ❷ 新しいターゲットボードに対応する際に共通部で 1 箇所だけ修正が必要となる。
具体的には、`include/sys/machine.h` に新しいターゲットボード用のヘッダファイルをインクルードする起点となるコードを追加している。
実際に追加するコードについては「2.5. ターゲット名」を参照のこと。

2.7.3. アプリケーション依存部

`app_program` は、μ T-Kernel 3.0 で動作するアプリケーションに応じて変更が必要な部分である。通常は応用製品毎に用意する。サンプルとして `app_main.c` を実装してあるので、必要に応じて拡張する。

μ T-Kernel 3.0 でのアプリケーションの開発手順などに関しては、各ターゲットボード用の「μ T-Kernel 3.0 構築手順書」を参照のこと。

- 💡 アプリケーション依存部のディレクトリ名はリポジトリによって異なる。
具体的には以下の名称になっているので注意すること。

<code>mtkernel_3</code>	<code>app_sample/</code>
<code>mtk3_bsp</code>	<code>app_program/</code>

2.7.4. その他

(1) config

config には、μ T-Kernel 3.0 用の設定ファイルが格納されている。

- config.h μ T-Kernel 3.0 の構成に関する設定
- config_func.h μ T-Kernel 3.0 の機能に関する設定
- config_device.h デバイスドライバサンプルの対応(実装)状況
- config_tm.h T-Monitor に関する設定

詳細については「13. コンフィギュレーション」を参照のこと。

(2) include

include には、μ T-Kernel 3.0 用のヘッダファイルが格納されている。

include/sys には、μ T-Kernel 3.0 の本体が利用するヘッダファイルがある。

include/tk には、μ T-Kernel 3.0 の API を利用するためにアプリケーションでインクルードする必要のあるヘッダファイルがある。

include/tm には、T-Monitor の API を利用するためにアプリケーションでインクルードする必要のあるヘッダファイルがある。

具体的にはアプリケーションのソースコードに以下の様に記述して、μ T-Kernel 3.0 用のヘッダファイルをインクルードする。

app_sample/app_main.c での実装例

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

リスト 6 アプリケーションでのヘッダファイルのインクルード

他のヘッダファイルについては、上記のインクルードファイルから読み込まれる。

もしくは、ドライバサンプルなどがインクルードする構成になっている。

- ❶ デバッグサポート機能(μ T-Kernel/DS)を利用する場合は、<tk/dbgspt.h>も追加でインクルードする必要がある。

(3) kernel

kernel には、μ T-Kernel 3.0 本体が格納されている。

knlibc, tstdlib, sysinit, inittask, tkernel には μ T-Kernel 3.0 の共通部のコードが含まれている。μ T-Kernel 3.0 の機能は API を介して利用することになるので、これらのディレクトリに含まれるコードをユーザが直接利用することはない。

kernel/sysdepend は、ハードウェア依存部でありターゲットボード、CPU、CPU コアに応じて修正が必要となる。

kernel/usermain は、初期タスク(μ T-Kernel 3.0 で最初に起動されるタスク)から呼び出される usermain 関数が定義されている。このファイルでは Weak シンボルとして定義しているので、ビルド時に実際にリンクされるのは app_program/app_main.c に実装され

ている `usermain` 関数の方となる。

(4) lib

lib には、μ T-Kernel 3.0 の周辺機能などが含まれる。

lib/libtk には、μ T-Kernel 3.0 のユーティリティ機能などが含まれる。

lib/libtm には、T-Monitor 互換のコンソール入出力関数が含まれる。

具体的には以下の関数のサブセットを利用可能である。

```
INT  tm_getchar( INT wait );
INT  tm_putchar( INT c );
INT  tm_getline( UB *buff );
INT  tm_putstring( const UB *buff );
```

また、追加機能として以下を利用可能である。

```
INT  tm_printf( const UB *format, ... );
INT  tm_sprintf( UB *str, const UB *format, ... );
```

(5) build_make

build_make には、GCC でビルドする際に利用するディレクトリが含まれる。

3. 基本実装仕様

基本実装仕様に関する共通の説明については、トロンフォーラムの「 μ T-Kernel 3.0 共通実装仕様書」の「2. 基本実装仕様」を参照のこと。

ターゲットボード毎の実装については各ターゲットボードの「 μ T-Kernel 3.0 実装仕様書」を参照のこと。

3.1. 対象マイコン

本実装は、対象マイコンとして以下を想定している。

- 32 ビットまたは 16 ビットマイコン
- 単一の物理アドレス空間(MMU は無し、または使用しない)

3.2. 実行モードと保護レベル

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し、非特権(ユーザ)モードは使用しない。よって、プログラムを実行するモードは基本的に同一である。具体的なマイコンの実行モードは実装するマイコンに依存する。

マイコンの実行モードは特権モードのみを使用するので、OS が提供する保護レベルはすべて保護レベル 0 と同等となる。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、メモリ保護は保護レベル 0 を指定されたものと同じとなる。このため、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

OS が扱うレジスタは実装するマイコンに依存する。

特別なレジスタとして、`taskmode` レジスタを定義する。`taskmode` レジスタは、メモリへのアクセス権限(保護レベル)を保持する仮想的なレジスタである。マイコンの物理的なレジスタを割り当てるのではなく、OS 内の仮想的なレジスタとして実装する場合もある。

なお、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、`taskmode` レジスタの値は常に 0 となる。

3.4. 低消費電力モードと省電力機能

省電力機能は実装するマイコンに依存する。

3.5. コプロセッサ対応

コプロセッサ対応は実装するマイコンに依存する。

4. メモリ

メモリに関する共通の説明については、トロンフォーラムの「μ T-Kernel 3.0 共通実装仕様書」の「3. メモリ」を参照のこと。

ターゲットボード毎の実装については各ターゲットボードの「μ T-Kernel 3.0 実装仕様書」を参照のこと。

4.1. メモリモデル

本実装では、単一の物理アドレス空間を前提とする。アドレス空間上にはプログラムから使用可能な ROM および RAM が存在する。

プログラムは、静的にリンクされた一つの実行オブジェクトを前提とする。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

マイコンのアドレス・マップは、実装するマイコンに依存する。

4.3. OS のメモリマップ

本実装では、プログラムコードは ROM に置くことを想定している。

以下に本実装における一般的な ROM および RAM の内容(用途)を示す。

表 4-1 ROM の用途

種別	内容
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。OS の初期化後は RAM に配置される。
プログラムコード	プログラムコードが配置される領域
定数データ	定数データなどが配置される領域

表 4-2 RAM の用途

種別	内容
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	変数等が配置される領域
μ T-Kernel 管理領域 (OS 管理メモリ領域)	OS 内部の動的メモリ管理の領域
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する。

◆ μ T-Kernel 管理領域は、アドレス的に連続した単一の領域に割り当てられることを前提に実装されている。このため、アドレスが連続していたとしても、μ T-Kernel 管理領域に対して異なる領域にまたがったメモリ領域を割り当てることはできない。

異なる領域にまたがってメモリ領域を割り当てると、領域の境界をまたぐメモリが確保された場合に、そのメモリに対して正常なアクセスができないことがある。これは CPU のアーキテクチャによる制限事項である。

少なくとも Cortex-M3/M4/M7 では正常に動作しないことが、これらの "Technical Reference Manual" の "Unaligned accesses that cross regions" に記載されている。

4.4. スタック

本実装では以下の種類のスタックを使用する。

4.4.1. タスクスタック

タスクが使用するスタックであり、タスク毎に存在する。各タスクの生成時に指定され、OS が管理するシステムメモリ領域から動的に確保される。ただし、ユーザが確保した領域を使用することも可能である。

本実装では、すべてのプログラムは特権モードで実行しているので、システムスタックとユーザスタックの分離は行わない。一つのタスクに一つのタスクスタックが存在する。プロファイル TK_HAS_SYSSTACK は FALSE である。

- ◆ 割込みもタスクスタックを使用する。このため、タスクスタックのサイズには、タスク実行中に発生する割込み用のスタックサイズを加算しておく必要がある。

4.4.2. 例外スタック

例外処理および OS 初期化処理の実行中に使用するスタックである。

コンフィギュレーション情報 (EXC_STACK_SIZE) にてサイズが指定され、メモリマップ上に静的に領域が確保される。

- ◆ 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する割込み用のスタックサイズを加算しておく必要がある。

4.4.3. テンポラリスタック

OS の内部処理 (タスクディスパッチ処理など) で使用されるスタックである。

コンフィギュレーション情報 (TMP_STACK_SIZE) にてサイズの指定が指定され、メモリマップ上に静的に領域が確保される。

4.5. OS 内の動的メモリ管理

本実装では、OS 内で使用するメモリの動的な管理を行うことができる。

OS は必要に応じて、μ T-Kernel 管理領域 (OS 管理メモリ領域) からメモリを確保し、使用後はメモリを返却する。具体的には、以下のメモリ領域が動的管理の対象となる。

- タスクのスタック領域
- 固定長メモリプール領域

- 可変長メモリプール領域
- メッセージバッファ領域
- システムメモリ

OS 内の動的メモリ管理は、コンフィギュレーションの `USE_IMALLOC` を 1 に設定することにより有効となる(初期値は有効)。

`USE_IMALLOC` を 0(無効)に設定した場合、OS 内の動的メモリ管理の機能は使用できなくなる。この場合、上記のうち、カーネルオブジェクトの生成の際には、`TA_USERBUF` 属性(ユーザ指定のメモリ領域を使用)を指定しなくてはならない。

プログラムのコードやデータが割り当てられていない RAM の領域が、μ T-Kernel 管理領域に割り当てられる。標準の設定では、すべての RAM の空き領域を μ T-Kernel 管理領域としている。

μ T-Kernel 管理領域は、コンフィギュレーションの `CNF_SYSTEMAREA_TOP` と `CNF_SYSTEMAREA_END` により、先頭アドレスと最終アドレスを指定することにより、調整が可能である。また、これらの定義に 0 を指定することにより、デフォルトの設定とすることができる。

μ T-Kernel 管理領域のアドレスを調整することにより、OS 管理外のメモリ領域をつくることができる。OS 管理外のメモリ領域は、OS からは使用されないの、ユーザプログラムで自由に使用することができる。

- ① アドレスが連続していないメモリ領域についても原則として OS 管理外のメモリ領域となる。これらについてもユーザプログラムで自由に使用することができる。

4.6. システムメモリ管理機能

本実装はシステムメモリ管理機能をサポートする。μ T-Kernel/SM のメモリ割り当てライブラリ関数(`Kmalloc`/`Kcalloc`/`Krealloc`/`Kfree`)が提供される。標準の設定では、プロファイル `TK_SUPPORT_MEMLIB` は `TRUE` となる。

ただし、システムメモリ管理機能は、OS 内の動的メモリ管理を使用し実現しているため、動的メモリ管理を無効(`USE_IMALLOC` を 0)とした場合は使用不可能となる。この時、プロファイルの `TK_SUPPORT_MEMLIB` は自動的に `FALSE` となる。

- ① 本実装ではメモリ保護には対応していない。このため、システムメモリもすべての保護レベルからアクセス可能である。


5. 割込みおよび例外

割込みおよび例外に関する共通の説明については、トロンフォーラムの「μ T-Kernel 3.0 共通実装仕様書」の「4. 割込みおよび例外」を参照のこと。

ターゲットボード毎の実装については各ターゲットボードの「μ T-Kernel 3.0 実装仕様書」を参照のこと。

5.1. マイコンの割込みおよび例外

マイコンには各種の割込みおよび例外が存在する。

 OS の仕様上は割込み、例外をまとめて割込みと称している。

5.2. ベクタテーブル

対象マイコンの仕様に依存する。

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

対象マイコンの仕様に依存する。

5.3.2. 多重割込み対応

対象マイコンの仕様に依存する。

5.3.3. クリティカルセクション

OS 内部処理において不可分に実行しなければならない箇所をクリティカルセクションと呼ぶ。クリティカルセクションでは原則割込みは禁止である。一般にクリティカルセクション中は、割込みのマスクレベルを最高に設定することにより実現される。

5.4. OS 内部で使用する割込み

OS 内部の処理において使用する割込みの種類を以下に示す。

表 5-1 OS 内部の処理で使用する割込み

種類	説明
SVC 割込み (スーパーバイザコール)	システムコールまたは拡張 SVC の呼び出しに使用するソフトウェア割込み
システムタイマ割込み	システムタイマによるハードウェア割込み
ディスパッチ要求	ディスパッチを要求する割込み
強制ディスパッチ要求	強制ディスパッチを要求する割込み

SVC 割込みは、システムコールや拡張 SVC を発行する際に使用されるソフトウェア割込みである。ただし、本実装ではシステムコールは関数呼び出しのみであり、SVC には対応しない。また、拡張 SVC の機能も無い。よって SVC 割込みは使用しない。

システムタイマ割込みは、システムタイマにより周期的に発生するハードウェア割込みである。これにより OS の時間管理機能が実行される。

ディスパッチ要求は、タスクのディスパッチを発生させるために使用される割込みである。OS の API 処理にて、タスクのディスパッチが必要となった場合(実行タスクが変更になった場合)に発行される。

強制ディスパッチ要求は、OS の処理にて強制ディスパッチを発生させるために使用される割込みである。強制ディスパッチは、ディスパッチ元のタスクが存在しない場合に行われる特別なディスパッチである。具体的には OS の起動時とタスクの終了時のみに行われる。

上記の各種割込みには、各ハードウェアへの実装に応じて、具体的な割込みや例外が割り当てられる。実装によっては使用されない割込みもある。また、ハードウェア的に例外番号の割り当てがない CPU もある。

表 5-2 例) Corex-M3/M4 での例外番号と例外の種別

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	
13	(予約)	
14	PendSV	OS で使用
15	SysTick	OS で使用
16	IRQ 割込み#0	OS の割込み管理機能で管理
17	IRQ 割込み#1	
18	IRQ 割込み#2	
...		
255	IRQ 割込み#239	

- ① 実際に使用可能な例外、IRQ 割込みの個数、IRQ 割込みの各デバイスへの割り当てなどは CPU によって異なる。

詳細は各ターゲットボードの「μ T-Kernel 3.0 実装仕様書」や各 CPU のデータシートを参照のこと。

5.5. μ T-Kernel/OS の割込み管理機能

各ハードウェア向けの実装仕様書を参照のこと。

5.6. μ T-Kernel/SM の割込み管理機能

各ハードウェア向けの実装仕様書を参照のこと。

5.7. OS 管理外割込み

OS が管理する割込みよりも優先度の高い割込みを OS 管理外割込みと呼ぶ。

OS 管理外割込みはクリティカルセクションにおいても受け付けられる。つまり、管理外割込みの処理は、OS 自体の動作よりも優先して実行される。このため、OS 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。

OS 管理外割込みは、非常に高い応答性を要求される割込みなどに使用される。具体的な実装は各ハードウェア向けの実装仕様書を参照のこと。

- ❗ UCT が提供しているターゲットハードウェアに対する実装では、OS 管理外割込みは標準では対応していない。

6. 起動および終了処理

UCT が提供するターゲットボードの実装では、基本的に各 CPU 用の開発環境(や構築ツール)が提供する初期化ルーチンを利用して起動処理を実装する場合がある。これにより CPU のコンフィギュレーションやデバイスの組み込みに開発環境の機能が利用可能となり、開発作業の負荷を軽減することができる。

初期化処理はターゲットボードや開発環境によって異なるので、詳細については各ターゲットボードの「μ T-Kernel 3.0 実装仕様書」を参照のこと。

また、μ T-Kernel 3.0 の標準的な起動処理および終了処理に関しては、トロンフォーラムの「μ T-Kernel 3.0 共通実装仕様書」の「5. 起動および終了処理」を参照のこと。

トロンフォーラムの仕様書では以下について説明している。

5.1 システム起動処理

5.2 初期タスク

5.3 システム終了処理

5.4 ハードウェアの初期化および終了処理

5.5 デバイスドライバの実行および終了

7. サービスプロファイル

本実装でのサービスプロファイルの定義状況を以下に示す。

- ❶ 各サービスプロファイルの意味や使い方については「μ T-Kernel 3.0 仕様書」の「3.4 サービスプロファイル」を参照のこと。
- ❶ 各プロファイルの設定値は、トロンフォーラムが配布している μ T-Kernel 3.0 リファレンスコードでの設定値と同じである。
- ⚠ 以下の一覧にある設定値と実装に齟齬がある場合は実装が優先される。
一覧は実装状況を確認するための資料であり、実装を保証するものではない。

7.1. 有効・無効を示すプロファイル

ある機能が有効か無効かを示すサービスプロファイルであり、TRUE、FALSE のいずれかに定義された以下のマクロによってそれぞれ規定されている。

- ⚠ サービスプロファイルは μ T-Kernel 3.0 の実装状況を示すものである。
サービスプロファイルの設定値が FALSE に設定されているマクロ定義において設定値を TRUE に変更したとしても、当該機能が利用可能になるとは限らない。
- ❶ 一部機能については当該機能が有効になる場合がある。この様な機能については個別に説明を追加してある。

プロファイル用マクロを定義するファイルは以下のいずれかとなる。

- COMMON include/sys/profile.h
- TARGET include/sys/sysdepend/<TARGET>/profile.h
- CPU include/sys/sysdepend/cpu/<CPU>/profile.h
- CORE include/sys/sysdepend/cpu/core/<CORE>/profile.h

ただし、対象によっては<TARGET>、<CPU>、<CORE>の中で更に設定値が異なる場合がある。このような場合は「定義ファイル」の欄に対象を区別するための文字列を記載する。現在実装されている<CORE>は ARMv7-M, ARMv7-A, RXv2 である。<TARGET>と<CPU>については個別に提示する。

- ❶ 現在 UCT が提供しているターゲットボードの実装は ARMv7-M のみである。
ARMv7-A, RXv2 に関してはトロンフォーラムが提供しているので参考情報として記載している。

表 7-1 デバイスドライバ向け機能

マクロ定義	設定値	定義ファイル	対象機能
TK_SUPPORT_TASKEVENT	FALSE	COMMON	タスクイベント関連機能
TK_SUPPORT_DISWAI	FALSE	COMMON	タスクの待ち禁止状態
TK_SUPPORT_IOPORT	TRUE	CPU	I/O ポートアクセス機能
TK_SUPPORT_MICROWAIT	TRUE	CORE	微小待ち機能

- ❶ TK_SUPPORT_TSKEVENT と TK_SUPPORT_DISWAI は、μ T-Kernel 3.0 リファレンスコードとして対応していない(FALSE)。このため、「定義ファイル」に COMMON と記載してある。(以下同様)
- ❷ TK_SUPPORT_IOPORT は現在実装されている全ての<CPU>において同じ設定値(TRUE)である。このため、「定義ファイル」に CPU と記載してある。(以下同様)
- ❸ TK_SUPPORT_MICROWAIT は現在実装されている全ての<CORE>において同じ設定値(TRUE)である。このため、「定義ファイル」に CORE と記載してある。(以下同様)

表 7-2 省電力機能

マクロ定義	設定値	定義ファイル	対象機能
TK_SUPPORT_LOWPOWER	FALSE	TARGET	省電力機能

表 7-3 動的/静的メモリ管理機能

マクロ定義	設定値	定義ファイル	対象機能
TK_SUPPORT_USERBUF	TRUE	COMMON	ユーザバッファ指定
TK_SUPPORT_AUTOBUF	TRUE	COMMON	自動バッファ割当て
TK_SUPPORT_MEMLIB	TRUE	COMMON	メモリ割当てライブラリ USE_IMALLOC の設定値に依存

- ❶ TK_SUPPORT_MEMLIB は USE_IMALLOC の定義に依存する。
デフォルトでは USE_IMALLOC は(1)と定義されているので、TK_SUPPORT_MEMLIB も TRUE となる。

表 7-4 タスク例外処理機能

マクロ定義	設定値	定義ファイル	対象機能
TK_SUPPORT_TASKEXCEPTION	FALSE	COMMON	タスク例外処理機能

表 7-5 サブシステム関連機能

マクロ定義	設定値	定義ファイル	対象機能
TK_SUPPORT_SUBSYSTEM	FALSE	COMMON	サブシステム管理機能
TK_SUPPORT_SSYEVENT	FALSE	COMMON	サブシステムのイベント処理

表 7-6 システム構成情報取得機能

マクロ定義	設定値	定義ファイル	対象機能
TK_SUPPORT_SYSCONF	FALSE	COMMON	システム構成情報管理機能

表 7-7 64 ビット対応、16 ビット対応

マクロ定義	設定値	定義ファイル	対象機能
TK_HAS_DOUBLEWORD	TRUE	COMMON	64 ビットデータ型 (D, UD, VD)
TK_SUPPORT_USEC	FALSE	COMMON	マイクロ秒
TK_SUPPORT_LARGEDEV	FALSE	COMMON	大容量デバイス (64 ビット)
TK_SUPPORT_SERCD	FALSE	COMMON	サブエラーコード

表 7-8 割込み関連機能

マクロ定義	設定値	定義ファイル	対象機能
TK_SUPPORT_INTCTRL	TRUE	CORE	割込みコントローラ制御機能
TK_HAS_ENAINTLEVEL	TRUE	CORE	割込み優先度の指定が可能
TK_SUPPORT_CPUINTLEVEL	FALSE TRUE TRUE	ARMv7-A ARMv7-M RXv2	CPU 内割込みマスクレベル
TK_SUPPORT_CTRLINTLEVEL	FALSE	CORE	割込みコントローラ内割込みマスクレベル
TK_SUPPORT_INTMODE	TRUE	CORE	割込みモード設定機能

- ① TK_SUPPORT_CPUINTLEVEL は対象とする<CORE>によって設定値が異なる。
このため、対象とする<CORE>毎に設定値を記載してある。(以下同様)

表 7-9 キャッシュ制御機能

マクロ定義	設定値	定義ファイル	対象機能
TK_SUPPORT_CACHECTRL	FALSE	CORE	メモリキャッシュ制御機能
TK_SUPPORT_SETCACHEMODE	FALSE	CORE	キャッシュモード設定機能
TK_SUPPORT_WBCACHE	FALSE	CORE	ライトバックキャッシュ
TK_SUPPORT_WTCACHE	FALSE	CORE	ライトスルーキャッシュ

表 7-10 FPU(COP)サポート

マクロ定義	設定値	定義ファイル	対象機能
TK_SUPPORT_FPU	FALSE	ARMv7-A	FPU 機能
	TRUE	ARMv7-M	CPU_CORE_ACM4F の定義がある場合
	FALSE	ARMv7-M	CPU_CORE_ACM4F の定義がない場合
	TRUE	ARMv8-M	CPU_CORE_ACM85 の定義がある場合
	FALSE	ARMv8-M	CPU_CORE_ACM85 の定義がない場合
	TRUE	RXv2	USE_FPU が TRUE の場合
	FALSE	RXv2	USE_FPU が FALSE の場合

マクロ定義	設定値	定義ファイル	対象機能
TK_SUPPORT_COP0	FALSE TRUE FALSE TRUE FALSE TRUE FALSE	ARMv7-A ARMv7-M ARMv7-M ARMv8-M ARMv8-M RXv2 RXv2	番号 0 のコプロセッサ利用機能 CPU_CORE_ACM4F の定義がある場合 CPU_CORE_ACM4F の定義がない場合 CPU_CORE_ACM85 の定義がある場合 CPU_CORE_ACM85 の定義がない場合 USE_FPU が TRUE の場合 USE_FPU が FALSE の場合
TK_SUPPORT_COP1	FALSE FALSE TRUE FALSE	ARMv7-A ARMv7-M RXv2 RXv2	番号 1 のコプロセッサ利用機能 USE_DSP が TRUE の場合 USE_DSP が FALSE の場合
TK_SUPPORT_COP2	FALSE	CORE	番号 2 のコプロセッサ利用機能
TK_SUPPORT_COP3	FALSE	CORE	番号 3 のコプロセッサ利用機能

- ① TK_SUPPORT_FPU、TK_SUPPORT_COP0、TK_SUPPORT_COP1 は対象とする<CORE>によって設定値が異なる。
- ① USE_FPU と USE_DSP の定義については「13. コンフィギュレーション」を参照のこと。

表 7-11 その他の機能

マクロ定義	設定値	定義ファイル	対象機能
TK_SUPPORT_ASM	FALSE	CORE	アセンブリ言語による処理ルーチン
TK_SUPPORT_REGOPS	TRUE	CORE	タスクレジスタ取得・設定機能
TK_ALLOW_MISALIGN	FALSE	CORE	メモリのミスアラインアクセス可能 ALLOW_MISALIGN の設定値に依存
TK_BIGENDIAN	FALSE	CORE	ビッグエンディアン BIGENDIAN の設定値に依存
TK_TRAP_SVC	FALSE	COMMON	システムコールの呼び出しに CPU のトラップ命令を利用
TK_HAS_SYSSTACK	FALSE	COMMON	タスクが独立したシステムスタックを持つ
TK_SUPPORT_PTIMER	FALSE	TARGET	物理タイマ機能
TK_SUPPORT_UTC	TRUE	COMMON	UNIX 表現のシステム時刻
TK_SUPPORT_TRONTIME	TRUE	COMMON	TRON 表現のシステム時刻

- ① TK_ALLOW_MISALIGN は ALLOW_MISALIGN の設定値に依存する。
ALLOW_MISALIGN の設定値は実装されている全ての<CORE>において 0 と定義されているので、TK_ALLOW_MISALIGN は FALSE となる。
- ① TK_BIGENDIAN は BIGENDIAN の設定値に依存する。
BIGENDIAN の設定値は実装されている全ての<CORE>において 0 と定義されているので、TK_BIGENDIAN は FALSE となる。
- ① TK_SUPPORT_PTIMER はターゲットボードの実装に依存する。

UCT が BSP を提供するターゲットボード用の実装では **FALSE** となる。

他のターゲットボードでの実装状況については各ターゲットボードの実装仕様書、またはソースコードを参照のこと。

表 7-12 デバッグサポート関連機能

マクロ定義	設定値	定義ファイル	対象機能
TK_SUPPORT_DSNAME	FALSE	COMMON	DS オブジェクト名称 USE_OBJECT_NAME の設定値に依存
TK_SUPPORT_DBGSP	FALSE	COMMON	μ T-Kernel/DS USE_DBGSP の設定値に依存

- ① TK_SUPPORT_DSNAME は USE_OBJECT_NAME の設定値に依存する。

USE_OBJECT_NAME はデフォルトでは (0) と定義されており、TK_SUPPORT_DSNAME は FALSE となる。ただし、DS オブジェクト名称の機能自体は実装されているので、USE_OBJECT_NAME を (1) と定義すると、TK_SUPPORT_DSNAME は TRUE となる。

- ① TK_SUPPORT_DBGSP は USE_DBGSP の設定値に依存する。

USE_DBGSP はデフォルトでは (0) と定義されており、TK_SUPPORT_DBGSP は FALSE となる。ただし、μ T-Kernel/DS の機能自体は一部を除いて実装されているので、USE_DBGSP を (1) と定義すると、TK_SUPPORT_DBGSP は TRUE となる。

- ① μ T-Kernel/DS の対応状況については「11. デバッグサポート機能」を参照のこと。

7.2. 値を持つサービスプロファイル

上限値やバージョン番号などを示すサービスプロファイルは、その値を示すマクロとして規定される。

仕様により設定可能な範囲が決められているプロファイルもある。

表 7-13 μ T-Kernel 仕様書のバージョン番号

マクロ定義	設定値	定義ファイル	対象機能
TK_SPECVER_MAGIC	6	COMMON	μ T-Kernel の MAGIC ナンバー
TK_SPECVER_MAJOR	3	COMMON	仕様書のメジャーバージョン番号
TK_SPECVER_MINOR	0	COMMON	仕様書のマイナーバージョン番号
TK_SPECVER	0x0300	COMMON	μ T-Kernel のバージョン番号

表 7-14 カーネル構成情報

マクロ定義	設定値	定義ファイル	対象機能
TK_MAX_TSKPRI	32	COMMON	最大タスク優先度 MAX_TSKPRI に依存
TK_WAKEUP_MAXCNT	+2147483647L	COMMON	タスクの起床要求の最大キューイング数
TK_SEMAPHORE_MAXCNT	+2147483647L	COMMON	セマフォ資源数の最大値 (maxsem) の上限値

マクロ定義	設定値	定義ファイル	対象機能
TK_SUSPEND_MAXCNT	+2147483647L	COMMON	タスクの強制待ち要求の最大ネスト数
TK_MAX_PTIMER	0	CPU	最大物理タイマ番号

① TK_MAX_TSKPRI は MAX_TSKPRI の設定値に依存し、MAX_TSKPRI は CNF_MAX_TSKPRI の設定値に依存する。CNF_MAX_TSKPRI はデフォルトでは 32 と定義されており、TK_MAX_TSKPRI も 32 となる。なお、TK_MAX_TSKPRI の最小値は 16 と仕様で決められている。

① TK_MAX_PTIMER はターゲットボードの実装に依存する。

UCT が提供するターゲットボード用の実装では全て 0(未対応)となる。

他のターゲットボードでの実装状況については各ターゲットボードの実装仕様書、またはソースコードを参照のこと。

表 7-15 リング保護情報

マクロ定義	設定値	定義ファイル	対象機能
TK_MEM_RNG0	0	CORE	TA_RNG0 の実際のメモリ保護レベル
TK_MEM_RNG1	0	CORE	TA_RNG1 の実際のメモリ保護レベル
TK_MEM_RNG2	0	CORE	TA_RNG2 の実際のメモリ保護レベル
TK_MEM_RNG3	0	CORE	TA_RNG3 の実際のメモリ保護レベル

① UCT が提供するターゲットボードの実装では特権モードと非特権(ユーザ)モードを区別せず、全て特権モードで動作する。このため、TK_MEM_RNG0、TK_MEM_RNG1、TK_MEM_RNG2、TK_MEM_RNG3 の定義値は全て 0 となる。

8. タスク

タスクに関する共通の説明については、トロンフォーラムの「μ T-Kernel 3.0 共通実装仕様書」の「6. タスク」を参照のこと。

ターゲットボード毎の実装については各ターゲットボードの「μ T-Kernel 3.0 実装仕様書」を参照のこと。

8.1. タスク属性

本実装における各タスク属性の設定の可否を以下に記す。

設定不可のタスク属性を指定した場合は、エラーE_RSATR となる。

表 8-1 タスク属性

マクロ定義	可否	説明
TA_HLNG	○	本実装では高級言語(C言語)のみ対応
TA_ASM	×	
TA_SSTKSZ	×	本実装では非対応(独立したシステムスタックとユーザスタックを持たない)
TA_USERSTACK	×	
TA_USERBUF	○	
TA_DSNAME	△	コンフィギュレーションで使用可否を設定する
TA_RNG0	○	実行モードは特権モードのみのため、いずれを指定してもメモリ保護はレベル 0(RNG0)と同等に扱われる
TA_RNG1	○	
TA_RNG2	○	
TA_RNG3	○	
TA_COPn	—	マイコンの仕様に依存する。 各ハードウェアの実装仕様書を参照のこと
TA_FPU	—	

- ① タスク属性で TA_ASM をサポートしていないので、サービスプロファイル TK_SUPPORT_ASM は FALSE と定義してある。
- ① TA_HLNG と TA_ASM は、仕様上以下のように定義(実装)されており、LSB の 1 ビットを利用した背反の関係になっている。

```
#define TA_ASM    0x00000000    /* アセンブリ言語によるプログラム */
#define TA_HLNG   0x00000001    /* 高級言語によるプログラム */
```

上記の通り TA_ASM の値は 0 であるので、タスク属性に指定してもエラーにはならないので注意が必要である。

8.2. タスク優先度

設定可能なタスク優先度 pri は以下となる。

$$1 \leq \text{pri} \leq \text{TK_MAX_TSKPRI} \quad (\text{タスク最大優先度})$$

タスク最大優先度 `TK_MAX_TSKPRI` は、コンフィギュレーション `CNF_MAX_TSKPRI` で設定される 16 以上の値である。

- ❶ `CNF_MAX_TSKPRI` が 16 以上であることは μT-Kernel 3.0 仕様で定義されている。
本実装では、`CNF_MAX_TSKPRI` に 16 未満の値を設定するとビルドエラーが発生する様になっている。

システムコンフィギュレーションで定義する `CNF_MAX_TSKPRI` からサービスプロファイル `TK_MAX_TSKPRI` までの定義の流れは以下のリストのとおりである。

```
config/config.h
```

```
#define CNF_MAX_TSKPRI          32          /* Task Max priority */
```

リスト 7 システムコンフィギュレーション

```
include/sys/knldef.h
```

```
#define MAX_TSKPRI (CNF_MAX_TSKPRI) /* Maximum priority number = lowest priority */
```

リスト 8 システム定義

```
include/sys/profile.h
```

```
#define TK_MAX_TSKPRI (MAX_TSKPRI) /* Maximum task priority */
```

リスト 9 サービスプロファイル

8.3. タスクの処理ルーチン

タスクの処理ルーチンは、以下の形式の C 言語の関数である。

```
void task( INT stacd, void* exinf )
```

```
{
```

```
    /* 処理 */
```

```
    tk_ext_tsk(); または tk_exd_tsk();          /* タスクの終了 */
```

```
}
```

リスト 10 タスクの記述形式

タスクの終了には、必ず `tk_ext_tsk()` または `tk_exd_tsk()` いずれかの API を発行すること。これらの API を発行することなく、タスクの処理関数が終了した場合の動作は保証しない。

- ❶ `tk_ext_tsk()` や `tk_exd_tsk()` はその中でタスクを終了するための処理を実行しており、原則としてタスクのコンテキストには戻ってこない。
ただし、本実装では例外的に内部でコンテキストエラーが発生した場合のみ `return` するようになっている。

8.4. タスクのスタック

本実装では、すべてのタスクは特権モードで実行されるので、保護レベル 0 とみなし、タスクのスタックはタスク毎に一つとする。ユーザスタックとシステムスタックは独立には実装されない(プロファイル `TK_HAS_SYSSTACK` は `FALSE` となる)。

スタックのサイズは、タスク生成時にユーザが指定する。

9. 時間管理機能

時間管理機能に関する共通の説明については、トロンフォーラムの「μT-Kernel 3.0 共通実装仕様書」の「7. 時間管理機能」を参照のこと。

ターゲットボード毎の実装については各ターゲットボードの「μT-Kernel 3.0 実装仕様書」を参照のこと。

9.1. システム時刻管理

9.1.1. システムタイマ

本実装ではシステム時刻管理のために、マイコン内蔵のタイマの一つをシステムタイマとして使用する。

UCT が提供するターゲットボードのうち、Cortex-M 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している。

システムタイマのティック時間は、コンフィグレーション `CNF_TIMER_PERIOD` で指定する。単位は 1 ミリ秒であり、標準の設定値は 10(ミリ秒)である。

`config/config.h`

```
#define CNF_TIMER_PERIOD      10      /* System timer period */
```

リスト 11 システムタイマのティック時間の設定

`CNF_TIMER_PERIOD` に設定可能な範囲は以下であり、これを外れるとビルドエラーが発生する。

$$\text{MIN_TIMER_PERIOD} \leq \text{CNF_TIMER_PERIOD} \leq \text{MAX_TIMER_PERIOD}$$

設定可能な範囲は CPU に依存するが、UCT が提供するターゲットボードの実装では以下の設定になっている。

`include/sys/sysdepend/cpu/<CPU>/sysdef.h`

```
/* Settable interval range (millisecond) */
```

```
#define MIN_TIMER_PERIOD      1
```

```
#define MAX_TIMER_PERIOD      50
```

リスト 12 システムタイマのティック時間の設定可能な範囲の指定

9.1.2. マイクロ秒の時間管理

本実装ではマイクロ秒の時間管理には対応しない。

このため、サービスプロファイル `TK_SUPPORT_USEC` は `FALSE` である。

9.2. タイムイベントハンドラ

9.2.1. タイムイベントハンドラ属性

タイムイベントハンドラ(周期ハンドラ、アラームハンドラ)は C 言語で記述されることを前提とした実装になっている。このため、タイムイベントハンドラのハンドラ属性に

TA_ASM 属性を指定することはできない。TA_HLNG 属性のみ指定可能である。

- ❶ タイムイベントハンドラ属性で TA_ASM をサポートしていないので、サービスプロファイル TK_SUPPORT_ASM は FALSE と定義してある。
- ❷ TA_HLNG と TA_ASM は、仕様上以下のように定義(実装)されており、LSB の 1 ビットを利用した背反の関係になっている。

```
#define TA_ASM    0x00000000    /* アセンブリ言語によるプログラム */  
#define TA_HLNG  0x00000001    /* 高級言語によるプログラム */
```

上記の通り TA_ASM の値は 0 であるので、ハンドラ属性に指定してもエラーにはならないので注意が必要である。

9.2.2. タイムイベントハンドラの実行状態

タイムイベントハンドラは、OS のシステムタイマの割込み処理から実行される。このため、タイムイベントハンドラも非タスク部のコンテキストで実行されることになる。

ただし、タイムイベントハンドラの実行中は、原則として、タスクと同様にすべての割込みを受け付けるように実装されている。

ただし、TA_STA 属性の周期ハンドラで周期起動位相(cycphs)が 0 の場合は、API(tk_cre_cyc)の処理内で、最初の周期ハンドラの実行が行われる。この場合は、OS のクリティカルセクション内でハンドラが実行されるため、タイムイベントハンドラの実行中も割込みは禁止となる。

10. その他の実装仕様

その他の実装仕様に関する共通の説明については、トロンフォーラムの「μ T-Kernel 3.0 共通実装仕様書」の「8. その他の実装仕様」を参照のこと。

ターゲットボード毎の実装については各ターゲットボードの「μ T-Kernel 3.0 実装仕様書」を参照のこと。

10.1. システムコール

本実装では、システムコールの呼び出し形式は、C 言語の関数呼び出しのみとする。ソフトウェア例外（SVC 例外）による呼出しには対応しない。

このため、プロファイル TK_TRAP_SVC は FALSE である。

10.2. サブシステム

本実装では、サブシステムはサポートされない。

このため、プロファイル TK_SUPPORT_SUBSYSTEM および TK_SUPPORT_SSYEVENT は FALSE である。

10.3. μ T-Kernel 2.0 互換機能

本実装では、μ T-Kernel 3.0 仕様からは除外されたランデブ機能を、μ T-Kernel 2.0 互換機能としてソースコードに残されている。

💡 今後、実装から削除される可能性があるので使用は推奨されていない。

μ T-Kernel 2.0 互換機能は、コンフィギュレーションの USE_LEGACY_API を 1 に設定すると有効となる。コンフィギュレーションの初期設定は 0(無効)である。

USE_LEGACY_API を 1 に設定した場合は、合わせて CNF_MAX_PORID を 1 以上に設定すること。CNF_MAX_PORID の初期設定は 0 であり、そのままビルドするとシステムの初期化時にエラー終了する。

❗ コンフィギュレーションの設定に関しては「13. コンフィギュレーション」を参照のこと。

10.4. 物理タイマ機能

UCT が提供するターゲットハードウェアの実装では物理タイマ機能は非対応である。

このため、コンフィギュレーションの設定 USE_PTMR は 0(無効)に固定とする。

❗ 物理タイマ機能に対応したターゲットハードウェアでは、コンフィギュレーションの USE_PTMR を 1 に設定すると有効となる。

10.5. デバイスドライバ

μ T-Kernel 3.0 はデバイス管理機能を有するが、デバイスドライバは OS のソースコード自体には含めない。

本実装では、基本的なデバイスドライバをサンプルコードとして提供する。詳細は「μ T-Kernel 3.0 デバイスドライバ説明書」を参照のこと。

11. デバッグサポート機能

デバッグサポート機能に関する共通の説明については、トロンフォーラムの「μT-Kernel 3.0 共通実装仕様書」の「9. デバッグサポート機能」を参照のこと。

ターゲットボード毎の実装については各ターゲットボードの「μT-Kernel 3.0 実装仕様書」を参照のこと。

11.1. デバッグサポート機能の有効化

本実装では、コンフィギュレーション `USE_DBGSP` を有効(1)に設定し、OS を再構築することにより、μT-Kernel/DS のデバッグサポート機能が有効となり、デバッグ用 API が使用可能となる。コンフィギュレーションの初期設定は無効(0)である。

また、DS オブジェクト名を使用するにはコンフィギュレーション `USE_OBJECT_NAME` を有効(1)に設定する。コンフィギュレーションの初期設定は無効(0)である。

config/config.h

```
/* Debugger support function
```

```
 * 1: Valid 0: Invalid
```

```
 */
```

```
#define USE_DBGSP (0) /* Use mT-Kernel/DS */
```

```
#define USE_OBJECT_NAME (0) /* Use DS object name */
```

リスト 13 デバッグサポート機能の設定値(初期設定)

11.2. 対応するデバッグ機能

本実装では以下の機能に対応していない。

- マイクロ秒 TK_SUPPORT_USEC は FALSE
- タスク例外処理機能 TK_SUPPORT_TASKEXCEPTION は FALSE
- 実行トレース機能 TK_TRAP_SVC は FALSE

このため、μT-Kernel/DS のデバッグサポート機能についても上記に関連する API は対応していない。

本実装における μT-Kernel/DS の API の対応状況(使用可否)は下表のとおりである。

表 11-1 μT-Kernel/DS の API 対応状況

API 名	対応	備考
td_lst_tsk	○	
td_lst_sem	○	
td_lst_flg	○	
td_lst_mbx	○	
td_lst_mtx	○	
td_lst_mbf	○	
td_lst_mpf	○	

API 名	対応	備考
td_lst_mpl	○	
td_lst_cyc	○	
td_lst_alm	○	
td_lst_por	○	μT-Kernel 2.0 互換機能
td_lst_ssy	○	
td_rdy_que	○	
td_sem_que	○	
td_flg_que	○	
td_mbx_que	○	
td_mtx_que	○	
td_smbf_que	○	
td_rmbf_que	○	
td_mpf_que	○	
td_mpl_que	○	
td_cal_que	○	μT-Kernel 2.0 互換機能
td_acp_que	○	μT-Kernel 2.0 互換機能
td_ref_tsk	○	
td_ref_tex	×	タスク例外処理機能は非対応
td_ref_sem	○	
td_ref_flg	○	
td_ref_mbx	○	
td_ref_mtx	○	
td_ref_mbf	○	
td_ref_mpf	○	
td_ref_mpl	○	
td_ref_cyc	○	
td_ref_cyc_u	×	マイクロ秒は非対応
td_ref_alm	○	
td_ref_alm_u	×	マイクロ秒は非対応
td_ref_por	○	μT-Kernel 2.0 互換機能
td_ref_sys	○	
td_ref_ssy	○	
td_get_reg	○	
td_set_reg	○	
td_get_utc	○	
td_get_utc_u	×	マイクロ秒は非対応
td_get_tim	○	

API 名	対応	備考
td_get_tim_u	×	マイクロ秒は非対応
td_get_otm	○	
td_get_otm_u	×	マイクロ秒は非対応
td_ref_dsname	○	
td_set_dsname	○	
td_hok_svc	×	実行トレース機能は非対応
td_hok_dsp	×	実行トレース機能は非対応
td_hok_int	×	実行トレース機能は非対応

- ① μ T-Kernel 2.0 互換機能用の API は、コンフィギュレーションで `USE_LEGACY_API` を有効(1)に設定した場合に使用可能となる。

12. T-Monitor 互換 API

T-Monitor 互換 API に関する共通の説明については、トロンフォーラムの「μ T-Kernel 3.0 共通実装仕様書」の「10. T-Monitor 互換ライブラリ」と「T-Monitor 仕様書」を参照のこと。

ターゲットボード毎の実装については各ターゲットボードの「μ T-Kernel 3.0 実装仕様書」を参照のこと。

- ① トロンフォーラムの資料では「T-Monitor 互換ライブラリ」と説明されているが、UCT が提供するターゲットハードウェア向けの機能としては直接リンクする方式で T-Monitor 互換の API を提供している。このため、本書では「T-Monitor 互換 API」と称している。ただし、利用しているソースコードは同じものである。

12.1. T-Monitor 概要

T-Monitor は、T-Kernel 1.0 の標準開発環境であった T-Engine のモニタプログラムである。μ T-Kernel はターゲットハードウェアを T-Engine に限定していないので、T-Engine のモニタプログラムを前提としない実装になっている。

また、T-Monitor は μ T-Kernel の仕様には含まれていない。

本実装では、T-Monitor の一部の機能を主にデバッグ用途として、T-Monitor 互換 API として提供する。

12.2. T-Monitor 互換 API の有効化

T-Monitor 互換 API は、コンフィグレーションで `USE_TMONITOR` を有効(1)に設定すると使用可能となる。OS の起動処理の中で T-Monitor 互換 API に対する初期化が行われる。

また、`USE_SYSTEM_MESSAGE` を有効(1)に設定すると、OS の起動メッセージなどが、T-Monitor のコンソールに出力される。

コンフィグレーション `USE_EXCEPTION_DBG_MSG` を有効(1)に設定すると、暫定的な例外ハンドラのデバッグ出力が、T-Monitor のコンソールに出力される。

- ① 上記のコンフィグレーション(`USE_~`)の初期値はすべて有効(1)である。
- ① 入出力の対象となる通信ポートはターゲットハードウェアの実装に依存する。

12.3. 対応 API

本実装では、T-Monitor 互換の API 機能として表 12-1 の API を提供する。

表 12-1 T-Monitor 互換 API

API 名	備考
<code>tm_getchar</code>	コンソールから 1 文字入力
<code>tm_putchar</code>	コンソールへの 1 文字出力

API 名	備考
tm_getline	コンソールから 1 行入力
tm_putstring	コンソールへの文字列出力
tm_printf	コンソールへの書式付文字列出力
tm_sprintf	文字列変数への書式付文字列出力

- ❗ 各 API はサブセットであり、全ての機能には対応していない。
- ❗ tm_printf と tm_sprintf は T-Monitor 仕様には存在しない。
μ T-Kernel において独自に追加した API である。
- ❗ tm_printf と tm_sprintf の書式は printf 関数のサブセットになっている。
全ての書式には対応していない。

12.4. T-Monitor のコンフィギュレーション

T-Monitor 互換 API の機能は、コンフィギュレーションファイル(config/comfig_tm.h)にて設定される。設定値を変更し OS を再構築することにより、T-Monitor 互換機能の設定を変更することができる。

以下にコンフィグレーションの一覧を示す。値は初期値であり、変更可能である。

(1) 通信ポート設定

T-Monitor 互換 API で通信ポートを使用するか否かを指定する。

表に示した初期設定[1, 0]であればシリアル通信デバイスが通信ポートとして利用され、[0, 1]とすれば入出力は行われない。

表 12-2 通信ポート設定

マクロ定義	値	説明
TM_COM_SERIAL_DEV	1	シリアル通信デバイスを通信ポートに使用する。 1 : 有効、0 : 無効
TM_COM_NO_DEV	0	通信ポート無し(入出力は行われない) 1 : 有効、0 : 無効

(2) tm_printf 関連設定

tm_printf および tm_sprintf に関する設定を指定する。

表 12-3 tm_printf 関連設定

マクロ定義	値	説明
USE_TM_PRINTF	1	tm_printf および tm_sprintf を使用する。 1 : 使用する、0 : 使用しない
TM_OUTBUF_SZ	0	API 実行時にスタック上に確保する出力バッファのサイズ(単位: バイト) 0 を指定した場合はバッファは確保されず、出力はコンソールに順次出力されていくことになる。

- ◆ **TM_OUTBUF_SZ** を 1 以上にした場合は、**tm_printf** を発行したコンテキストのスタックに **TM_OUTBUF_SZ** バイトの出力バッファが確保される。すなわち、**tm_printf** を発行するタスクなどではスタックサイズを余分に確保しなければならない。
また、**TM_OUTBUF_SZ** にはバッファとして十分な量を指定しなければならない。

13. コンフィギュレーション

コンフィギュレーションに関する共通の説明については、トロンフォーラムの「μ T-Kernel 3.0 共通実装仕様書」の「11. コンフィギュレーション」を参照のこと。

ターゲットボード毎の実装については各ターゲットボードの「μ T-Kernel 3.0 実装仕様書」を参照のこと。

13.1. 基本コンフィギュレーション

OS の変更可能な設定値は、コンフィギュレーションファイル(`config/config.h`)にて設定される。設定値を変更して OS を再構築することにより、OS の設定を変更することができる。

表 13-1 以下にコンフィギュレーションの一覧を示す。表の「設定値」はデフォルトの設定であり、変更可能である。ただし、オブジェクトの数などはその値に応じてメモリなどの資源が確保されるので、ユーザのシステムに応じた使用可能な資源から適切な値とする必要がある。

表 13-1 カーネル基本設定

名称	設定値	説明
CNF_SYSTEMAREA_TOP	0	システムメモリ領域の開始アドレス 0：システムのデフォルト値を使用
CNF_SYSTEMAREA_END	0	システムメモリ領域の終了アドレス 0：システムのデフォルト値を使用
CNF_MAX_TSKPRI	32	タスク優先度の最大値(≧16)
CNF_TIMER_PERIOD	10	システムタイマの割込み周期(単位：ミリ秒)

❶ CNF_SYSTEMAREA_TOP と CNF_SYSTEMAREA_END には、μ T-Kernel 管理領域(μ T-Kernel のメモリ管理機能により動的に管理される RAM 領域)の最下位アドレスと最上位アドレスを指定する。ただし、0 を指定した場合はシステムのデフォルト値(`INTERNAL_RAM_START` と `INTERNAL_RAM_END`)が使用される。

❷ CY8CKIT-CY8C6 では、μ T-Kernel 管理領域には固定の領域を割り当てている。このため、CNF_SYSTEMAREA_TOP、CNF_SYSTEMAREA_END、INTERNAL_RAM_START、INTERNAL_RAM_END は使用していない。

詳細については「μ T-Kernel 3.0 実装仕様書(CY8CKIT-CY8C6)」を参照のこと。

❸ CNF_TIMER_PERIOD は MIN_TIMER_PERIOD 以上、MAX_TIMER_PERIOD 以下である必要がある。指定可能な範囲から外れた場合はビルド時にエラーが発生する。

MIN_TIMER_PERIOD と MAX_TIMER_PERIOD は CPU 毎に `sysdef.h` で定義されているが、いずれも最小値が 1、最大値が 50 となっている。

```
include/sys/sysdepend/cpu/<CPU>/sysdef.h
```

表 13-2 カーネルオブジェクト設定

名称	設定値	説明
CNF_MAX_TSKID	32	最大タスク数
CNF_MAX_SEMID	16	最大セマフォ数
CNF_MAX_FLGID	16	最大イベントフラグ数
CNF_MAX_MBXID	8	最大メールボックス数
CNF_MAX_MTXID	4	最大ミューテックス数
CNF_MAX_MBFID	8	最大メッセージバッファ数
CNF_MAX_MPLID	4	最大可変長メモリプール数
CNF_MAX_MPFID	8	最大固定長メモリプール数
CNF_MAX_CYCID	4	最大周期ハンドラ数
CNF_MAX_ALMID	8	最大アラームハンドラ数

- ❗ 各マクロの設定値は 1 以上とする必要がある。
0 以下を設定するとシステム起動時にエラーとなる。
- ❗ 特定のカーネルオブジェクトをシステムから除外したい(使用しない)場合は USE_XXX に 0 を指定する。
USE_XXX については「13.2. 機能コンフィギュレーション」を参照のこと。
- 💡 カーネルオブジェクトの数はカーネルやデバイスドライバなどが使用する数も考慮して設定する必要がある。具体的には初期タスクを起動するだけのサンプルアプリを実行した状態であっても、タスクが 1 個、セマフォが 2 個、イベントフラグが 1 個生成(使用)されている。

表 13-3 デバイス情報

名称	設定値	説明
CNF_MAX_REGDEV	8	デバイスの最大登録数
CNF_MAX_OPNDEV	16	デバイスの最大同時オープン数
CNF_MAX_REQDEV	16	デバイスの最大要求数
CNF_DEVT_MBF SZ0	-1	デバイスイベント用メッセージバッファサイズ -1 : 使用しない
CNF_DEVT_MBF SZ1	-1	デバイスイベント用メッセージの最大長

- ❗ CNF_MAX_REGDEV は、tk_def_dev() で登録可能な最大デバイス数、つまり物理デバイスの最大数である。
- ❗ CNF_MAX_OPNDEV は、tk_opn_dev() でオープン可能な最大数、つまりデバイスディスクリプタの最大数である。
- ❗ CNF_MAX_REQDEV は、tk_rea_dev()、tk_wri_dev()、tk_srea_dev()、tk_swri_dev() で要求可能な最大数、つまりリクエスト ID の最大数である。
- ❗ CNF_DEVT_MBF SZ0 は、デバイスイベント用メッセージバッファのバッファサイズ(bufsz)をバイト数で指定する。CNF_DEVT_MBF SZ0 に -1 を指定すると、デバイスイ

ベント用のメッセージバッファは生成されない。

- ① CNF_DEVT_MBFSZ1 は、デバイスイベント用メッセージバッファのメッセージの最大長(maxmsz)をバイト数で指定する。CNF_DEVT_MBFSZ0 が -1 の場合は、CNF_DEVT_MBFSZ1 は無視される。

表 13-4 バージョン情報

名称	設定値	説明
CNF_VER_MAKER	0x0000	カーネルのメーカコード
CNF_VER_PRID	0x0000	カーネルの識別番号
CNF_VER_PRVER	3	カーネルのバージョン番号
CNF_VER_PRN01	0	製品管理情報
CNF_VER_PRN02	0	
CNF_VER_PRN03	0	
CNF_VER_PRN04	0	

- ① CNF_VER_MAKER と CNF_VER_PRID はカーネルの開発元を表す。
0x0000 はトロンプォーラムを表している。
UCT のメーカコードは 0x011D だが、機種依存部の追加だけなの変更していない。

表 13-5 OS 内部設定

名称	設定値	説明
USE_LEGACY_API	0	μ T-Kernel 2.0 互換 API 0：使用しない、1：使用する
CNF_MAX_PORID	0	最大ランデブポート数
CNF_EXC_STACK_SIZE	2048	初期化スタック(例外スタック)のサイズ
CNF_TMP_STACK_SIZE	256	テンポラリスタックのサイズ
USE_NOINIT	0	BSS 領域(初期値をもたない静的変数領域)の初期化の有無の指定 0：ゼロクリアする、1：初期化しない
USE_IMALLOC	1	OS 内部の動的メモリ管理の指定 0：使用しない、1：使用する
USE_SHUTDOWN	1	OS 終了処理の指定 0：使用しない、1：使用する
USE_STATIC_IVT	0	静的割込みベクタテーブルの指定 0：使用しない、1：使用する

- ① USE_LEGACY_API と CNF_MAX_PORID の設定については、「10.3. μ T-Kernel 2.0 互換機能」を参照のこと。
- ① CNF_EXC_STACK_SIZE は CY8CKIT-CY8C6 では使用していない。
初期化スタック領域を調整する場合はリンカスクリプト(cy8c6xxa_cm4_dual.ld)で調整する必要がある。
- ① スタックについては「4. メモリ」を参照のこと。
- ① USE_IMALLOC を 0 に設定した場合は、タスク、メッセージバッファ、固定長／可

変長メモリプールのオブジェクト生成時に `TA_USERBUF` 属性を指定し、アプリケーションがバッファを指定する必要がある。

- ① `USE_SHUTDOWN` を 0 に設定した場合は、OS 終了処理(デバイスの終了処理など)が削除されるのでコードサイズを小さくすることができる。
- ① `USE_STATIC_IVT` を 1 に設定した場合は、割込みベクタテーブル(Interrupt Vector Table)を ROM に配置する。ベクタテーブルを ROM に配置した場合、割込みハンドラを動的に定義できなくなるので、`tk_def_int()`は未サポートとなる。

◆ `CY8CKIT-CY8C6` では `USE_NOINIT` は 0 固定とする。(1 は設定不可)

`CY8CKIT-CY8C6` ではメモリを初期化した後で `ModusToolbox` の初期化ルーチン `SystemInit()` を呼び出している。このルーチンは `BSS` セクションが 0 クリアされていることを前提に実装されているため、`BSS` セクションが 0 クリアされていない場合の動作は不定となる。

表 13-6 API のパラメータチェック

名称	設定値	説明
<code>CHK_NOSPT</code>	1	API パラメータの指定(<code>E_NOSPT</code>) 0: チェック無、1: チェック有
<code>CHK_RSATR</code>	1	API パラメータの指定(<code>E_RSATR</code>) 0: チェック無、1: チェック有
<code>CHK_PAR</code>	1	API パラメータの指定(<code>E_PAR</code>) 0: チェック無、1: チェック有
<code>CHK_ID</code>	1	API パラメータの指定(<code>E_ID</code>) 0: チェック無、1: チェック有
<code>CHK_OACV</code>	1	API パラメータの指定(<code>E_OACV</code>) 0: チェック無、1: チェック有
<code>CHK_CTX</code>	1	API パラメータの指定(<code>E_CTX</code>) 0: チェック無、1: チェック有
<code>CHK_CTX1</code>	1	API パラメータの指定(<code>E_CTX</code>) ディスパッチ禁止中 0: チェック無、1: チェック有
<code>CHK_CTX2</code>	1	API パラメータの指定(<code>E_CTX</code>) タスク独立部実行中 0: チェック無、1: チェック有
<code>CHK_SELF</code>	1	API パラメータの指定(<code>E_OBJ</code>) 自タスクを指定 0: チェック無、1: チェック有
<code>CHK_TKERNEL_CONST</code>	1	<code>CONST</code> 指定のチェック 0: チェック無、1: チェック有

表 13-7 ユーザ定義初期化プログラム

名称	設定値	説明
<code>USE_USERINIT</code>	0	ユーザ定義初期化プログラムの有無 0: 無、1: 有
<code>RI_USERINIT</code>	0	ユーザ定義初期化プログラムの開始アドレス

- ① USE_USERINIT が 0 の場合、RI_USERINIT は無視される。

ユーザ定義初期化プログラムに関しては「5. 割込みおよび例外」を参照のこと。

表 13-8 デバッグサポート機能

名称	設定値	説明
USE_DBGSP	0	デバッグサポート機能の指定 0: 無、1: 有
USE_OBJECT_NAME	0	オブジェクト名機能の指定 0: 無、1: 有
OBJECT_NAME_LENGTH	8	オブジェクト名の最大長
USE_TMONITOR	1	T-Monitor 互換 API の使用 0: 使用しない、1: 使用する
USE_SYSTEM_MESSAGE	1	OS からのメッセージ(T-Monitor 出力) 0: 無、1: 有
USE_EXCEPTION_DBG_MSG	1	例外ハンドラメッセージ(T-Monitor 出力) 0: 無、1: 有

- ① OBJECT_NAME_LENGTH は USE_OBJECT_NAME が 0 の場合は利用されない。
 ① OBJECT_NAME_LENGTH は DS オブジェクト名称用の領域のサイズを表す。

表 13-9 コプロセッサ制御

名称	設定値	説明
USE_FPU	0	FPU 対応 0: 無、1: 有
USE_DSP	0	DSP 対応 0: 無、1: 有

- ① 対象の CPU に FPU や DSP があり、かつ機種依存部の実装が対応している場合のみ 1 を指定可能である。
 ① 対象の CPU に FPU や DSP がある(1)かない(0)かは以下のマクロで定義される。
 CPU_HAS_FPU
 CPU_HAS_DSP
 これらは、include/sys/sysdepend/cpu/<CPU>/sysdef.h で定義されている。
 ① 各ターゲットボードでの対応状況は下記のとおりである。
 詳細については各ターゲットボードの実装仕様書を参照のこと。

表 13-10 各ターゲットボードでの FPU と DSP の有無と対応状況

ターゲットボード	CPU	CPU_HAS_FPU	USE_FPU	CPU_HAS_DSP	USE_DSP
CY8CKIT-CY8C6	Cortex-M4F	1	0 or 1	0	0
SBK-M4KN	Cortex-M4F	1	0 or 1	0	0

表 13-11 物理タイマ機能

名称	設定値	説明
USE_PTMR	0	物理タイマ機能対応 0：無、1：有

- ❶ 対象の CPU が物理タイマを有し(CPU_HAS_PTMR が 1)、かつソフトウェア(機種依存部)の実装が対応している場合のみ 1 を指定可能である。
- ❶ UCT が提供するターゲットハードウェアでは物理タイマ機能は非対応である。
- ❶ CPU としてのタイマ機能の有無は include/sys/sysdepend/cpu/<CPU>/sysdef.h の CPU_HAS_PTMR で指定する。一般には有り(1)となる。

表 13-12 デバイスドライバ

名称	設定値	説明
USE_SDEV_DRV	1	デバイスドライバの使用 0：無、1：有

- ❶ デバイスドライバについては「μ T-Kernel 3.0 デバイスドライバ説明書」を参照のこと。
- ❶ 機種依存部の実装が対応している場合のみ 1 を指定可能である。
- ❶ UCT が提供するターゲットハードウェアではシリアル(UART)ドライバのみ対応している。他のデバイスドライバには対応していない。
詳細については各ターゲットボードの実装仕様書を参照のこと。

13.2. 機能コンフィギュレーション

OS の取り外しが可能な機能は、機能コンフィギュレーションとして有効・無効の指定ができる。無効とした機能はプログラムコードも生成されない。本機能は、使用しない機能を取り外すことにより、OS のプログラムコードのサイズを小さくすることが主な目的である。

機能コンフィギュレーションは、機能単位と API 単位で有効・無効の指定ができる。機能単位で無効化した場合、関連する API はすべて使用できなくなる。API 単位の指定は、その機能単位が有効な場合に指定可能である。

機能コンフィギュレーションは、すべての機能に対して指定できるわけではない。OS として必須の機能や、ある機能単位で必須の API などは無効にすることができないので、機能コンフィギュレーションの対象から除外してある。

機能コンフィギュレーションは、以下のファイルに記述する。

config/comfig_func.h : 機能コンフィギュレーションファイル

13.2.1. 機能単位

機能単位のコンフィギュレーションでは、それぞれのマクロに有効(1)、無効(0)を設定する。初期値ではすべての機能は有効(1)に設定されている。

以下にコンフィギュレーションの一覧を示す。

表 13-13 機能単位のコンフィギュレーション

名称	機能単位	備考
USE_SEMAPHORE	セマフォ	
USE_MUTEX	ミューテックス	
USE_EVENTFLAG	イベントフラグ	
USE_MAILBOX	メールボックス	
USE_MESSAGEBUFFER	ミューテックス	
USE_RENDEZVOUS	ランデブ	μ T-Kernel 2.0 互換機能
USE_MEMORYPOOL	可変長メモリプール	
USE_FIX_MEMORYPOOL	固定長メモリプール	
USE_TIMEMANAGEMENT	時間管理	
USE_CYCLICHANDLER	周期ハンドラ	
USE_ALARMHANDLER	アラームハンドラ	
USE_DEVICE	デバイス管理	メッセージバッファとセマフォを使用
USE_FAST_LOCK	高速ロック	セマフォを利用
USE_MULTI_LOCK	高速マルチロック	イベントフラグを利用

❗ ランデブは μ T-Kernel 3.0 の仕様からは削除されているため、USE_RENDEZVOUS が有効(1)でも機能単位で削除された状態になっている。

ランデブを使用する場合は USE_LEGACY を有効(1)に設定し、かつ CNF_MAX_PORID を 1 以上に設定する必要がある。

💡 各機能は内部的に他の機能を利用している場合がある。このため、設定によってはビルドできなくなる場合があるので注意すること。

13.2.2. API 単位

API 単位の指定は、以下の形式で記述される。

```
#define USE_FUNC_XX_YYY_ZZZ
```

XX_YYY_ZZZ は対応する API 名を大文字にした文字列である。

たとえば、tk_del_tsk は以下のように定義される。

```
#define USE_FUNC_TK_DEL_TSK
```

定義が存在する場合に対応する API が有効となる。定義がない場合に対応する API が無効となる。

表 13-14 に、API 単位でのコンフィギュレーションに使用する全てのマクロを示す。

表 13-14 にない API は無効化ができない。

表 13-14 API 単位のコンフィギュレーション

名称	設定値	備考
タスク管理	USE_FUNC_TK_DEL_TSK	
	USE_FUNC_TK_EXT_TSK	
	USE_FUNC_TK_EXD_TSK	
	USE_FUNC_TK_TER_TSK	
	USE_FUNC_TK_CHG_PRI	
	USE_FUNC_TK_REL_WAI	
	USE_FUNC_TK_GET_REG	
	USE_FUNC_TK_SET_REG	
	USE_FUNC_TK_GET_CPR	
	USE_FUNC_TK_SET_CPR	
	USE_FUNC_TK_REF_TSK	
	USE_FUNC_TK_SUS_TSK	
	USE_FUNC_TK_RSM_TSK	
	USE_FUNC_TK_FRSM_TSK	
	USE_FUNC_TK_SLP_TSK	
	USE_FUNC_TK_WUP_TSK	
	USE_FUNC_TK_CAN_WUP	
	USE_FUNC_TK_DLY_TSK	
	USE_FUNC_TD_LST_TSK	
	USE_FUNC_TD_REF_TSK	
	USE_FUNC_TD_INF_TSK	
	USE_FUNC_TD_GET_REG	
	USE_FUNC_TD_SET_REG	
セマフォ管理	USE_FUNC_TK_DEL_SEM	
	USE_FUNC_TK_REF_SEM	
	USE_FUNC_TD_LST_SEM	
	USE_FUNC_TD_REF_SEM	
	USE_FUNC_TD_SEM_QUE	
ミューテックス管理	USE_FUNC_TK_DEL_MTX	
	USE_FUNC_TK_REF_MTX	
	USE_FUNC_TD_LST_MTX	
	USE_FUNC_TD_REF_MTX	
	USE_FUNC_TD_MTX_QUE	
イベントフラグ管理	USE_FUNC_TK_DEL_FLG	
	USE_FUNC_TK_REF_FLG	
	USE_FUNC_TD_LST_FLG	

名称	設定値	備考
	USE_FUNC_TD_REF_FLG	
	USE_FUNC_TD_FLG_QUE	
メールボックス管理	USE_FUNC_TK_DEL_MBX	
	USE_FUNC_TK_REF_MBX	
	USE_FUNC_TD_LST_MBX	
	USE_FUNC_TD_REF_MBX	
	USE_FUNC_TD_MBX_QUE	
メッセージバッファ管理	USE_FUNC_TK_DEL_MBF	
	USE_FUNC_TK_REF_MBF	
	USE_FUNC_TD_LST_MBF	
	USE_FUNC_TD_REF_MBF	
	USE_FUNC_TD_SMBF_QUE	
	USE_FUNC_TD_RMBF_QUE	
ランデブ管理 (μ T-Kernel 2.0 互換機能)	USE_FUNC_TK_DEL_POR	
	USE_FUNC_TK_FWD_POR	
	USE_FUNC_TK_REF_POR	
	USE_FUNC_TD_LST_POR	
	USE_FUNC_TD_REF_POR	
	USE_FUNC_TD_CAL_QUE	
	USE_FUNC_TD_ACP_QUE	
可変長メモリプール管理	USE_FUNC_TK_DEL_MPL	
	USE_FUNC_TK_REF_MPL	
	USE_FUNC_TD_LST_MPL	
	USE_FUNC_TD_REF_MPL	
	USE_FUNC_TD_MPL_QUE	
固定長メモリプール管理	USE_FUNC_TK_DEL_MPF	
	USE_FUNC_TK_REF_MPF	
	USE_FUNC_TD_LST_MPF	
	USE_FUNC_TD_REF_MPF	
	USE_FUNC_TD_MPF_QUE	
時間管理	USE_FUNC_TK_SET_UTC	
	USE_FUNC_TK_GET_UTC	
	USE_FUNC_TK_SET_TIM	
	USE_FUNC_TK_GET_TIM	
	USE_FUNC_TK_GET_OTM	
	USE_FUNC_TD_GET_TIM	
	USE_FUNC_TD_GET_OTM	

名称	設定値	備考
周期ハンドラ管理	USE_FUNC_TK_DEL_CYC	
	USE_FUNC_TK_STA_CYC	
	USE_FUNC_TK_STP_CYC	
	USE_FUNC_TK_REF_CYC	
	USE_FUNC_TD_LST_CYC	
	USE_FUNC_TD_REF_CYC	
アラームハンドラ管理	USE_FUNC_TK_DEL_ALM	
	USE_FUNC_TK_STP_ALM	
	USE_FUNC_TK_REF_ALM	
	USE_FUNC_TD_LST_ALM	
	USE_FUNC_TD_REF_ALM	
システム情報管理	USE_FUNC_TK_ROT_RDQ	
	USE_FUNC_TK_GET_TID	
	USE_FUNC_TK_DIS_DSP	
	USE_FUNC_TK_ENA_DSP	
	USE_FUNC_TK_REF_SYS	
	USE_FUNC_TK_REF_VER	
	USE_FUNC_TD_REF_SYS	
	USE_FUNC_TD_RDY_QUE	

① ランデブは μ T-Kernel 3.0 仕様から削除されているため、機能単位で削除された状態になっている。

ランデブを使用する場合は `USE_LEGACY` を有効(1)に設定し、かつ `CNF_MAX_PORID` を 1 以上に設定する必要がある。

13.3. その他のコンフィギュレーション

以下の機能に対するコンフィギュレーションが可能である。

13.3.1. T-Monitor 関連コンフィギュレーション

T-Monitor 互換 API の各種設定は、`config/config_tm.h` において行う。

詳細については「12.4. T-Monitor のコンフィギュレーション」を参照のこと

13.3.2. デバイスドライバ関連コンフィギュレーション

デバイスドライバの各種設定は、`config/config_device.h` において行う。

詳細については「μ T-Kernel 3.0 デバイスドライバ説明書」の「2.2 コンフィギュレーション」を参照のこと。

μ T-Kernel 3.0

共通実装&構成仕様書

Rev 3.00.01 (May, 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.