μT-Kernel3.0
RZ/A2M IoT-Engine 向け
実装仕様書

Version.01.00.01

2021.01.15

変更履歴

版数（日付）	内　容
1.00.01 (2021.11.15)	● 1.4 関連ドキュメント バージョン番号等を更新
1.00.00 (2021.08.27)	● 初版

目次

1. 概要

1.1 目的

本書はRZ/A2M 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RZ/A2M 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	RZ/A2M IoT-Engine	UCテクノロジー製
搭載マイコン	RZ/A2M (R7S921053VCBG)	ルネサス エレクトロニクス製

1.3 ターゲット名

RZ/A2M 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_RZA2M_	
対象システム	IOTE_ RZA2M	RZ/A2M IoT-Engine
対象CPU	CPU_ RZA2M	RZ/A2M
対象CPUアーキテクチャ	CPU_CORE_ARMV7A	ARMv7-Aアーキテクチャ
	CPU_CORE_ACA9	ARM Cortex-A9コア

識別名は以下のファイルで定義される。

include¥sys¥sysdepend¥iote_rza2m¥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.1.00.1)	TRON フォーラム TEF033-W007-201209
搭載 マイコン	RZ/A2M グループ ユーザーズマニュアル ハードウェア編	ルネサス エレクトロニクス

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_rza2m                   RZ/A2M 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
   ├ rza2m                          RZ/A2M マイコン依存部
├      :
└ <CPU n>                       CPUn 依存部
└ core                        コア依存部
       ├ armv7a                 ARMv7-A コア依存部
├     :
└ <core n>                     コア n 依存部
```

「ARMv7A コア依存部」は、ARMv7A 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。
「RZ/A2M 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。
「RZ/A2M 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 RZ/A2M 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-A9
ROM	外部
RAM	内蔵 4MB + 外部

2.2 実行モードと保護レベル

ARM Cortex-A9 コアは、プログラムの動作モードとして、特権および非特権の複数のモードを持つ。本実装では、基本的には SVC モードを使用する。メモリ保護は行わず、常にすべてのメモリにアクセス可能とする。

よって OS が提供する保護レベルは、、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）、CPSR 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      cpsr;           /* 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 */
} 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_rza2m/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

本マイコンは浮動小数演算コプロセッサ(NEON/VFP)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-A9 は MMU(Memory Management Unit：メモリ管理ユニット)は有し、32bit の仮想アドレス空間で動作する。本実装では物理アドレス空間と仮想アドレス空間のアドレスを一致させている。つまり、仮想アドレスは物理アドレスと同一である。
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 IoT-Engine のメモリマップ

以下に RZ/A2M IoT-Engine のメモリマップを記す。

アドレス （上段：開始 下段：終了）	種別	サイズ	備考
0x2000 0000 0x207F FFFF	シリアル FLASH	8M Byte	外部シリアル FLASH ROM ※1
0x4000 0000 0x407F FFFF	HyperRAM	8M Byte	外部 RAM
0x8000 0000 0x803F FFFF	RAM	4M Byte	内蔵 RAM ※2

※1 シリアル FLASH ROM はブート時のプログラム格納用
※2 内蔵 RAM は 4Mbyte のうち、先頭の 128Kbyte が保持用内蔵 RAM(BackUp RAM)として使用可能

3.3 OS のメモリマップ

本実装では、OS はマイコンの内蔵 RAM のみを使用する。
OS は外部 RAM を使用しなので、OS 管理外のメモリとしてユーザが自由に使用することができる。
外部シリアル FLASH ROM にはブート時のプログラム（ブートローダ）が格納され、ブート時に実行される。OS を含む全てのプログラムのコードも FLASH ROM に格納されており、ブートローダにより RAM 上に配置され実行される。

以下に内蔵 RAM のメモリマップを示す。表中でアドレスに「-」が記載された箇所はデータのサイズによりＣ言語の処理系にてアドレスが決定され、OS 内ではアドレスの指定は行っていない。

(1) 内蔵 RAM のメモリマップ

アドレス※ （上段：開始 下段：終了）	種別	内容
0x8000 0000 0x8001 FFFF	バックアップ RAM 領域	未使用
0x8002 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
– –	プログラムデータ	C 言語の変数等が配置される領域
– –	例外スタック 領域	例外および割込みハンドラにて使用されるスタック
– –	OS 管理領域 （システムメモリ）	OS 内部の動的メモリ管理の領域
0x8030 0000 0x803F BFFF	非キャッシュ 領域	未使用
0x803F C000 0x803F FFFF	TTB 領域	Translation Table 領域

内蔵 RAM の先頭 128KB は BackUp RAM であり OS は未使用である。
内蔵 RAM の 0x8030 0000 番地以降は非キャッシュ領域に設定される。非キャッシュ領域のうち TTB 領域以外は OS では使用しないので、OS 管理外メモリとしてユーザが自由に使用することができる。

3.4 スタック

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
　タスクの実行中に使用される。
(2) 例外スタック
　IRQ による割込みハンドラで使用される。

(3) テンポラリスタック

OS の初期化およびディスパッチ処理時に使用する。

上記の共通仕様のスタックの他に、その他の例外（FIQ、未定義命令、アボート）について、それぞれ独立にスタックが割り当てられる。これらのスタックのサイズは include¥sys¥sysdepend¥cpu¥core¥armv7a¥sysdef.h にて以下のように定義されている。この値を変更し、再構築することによりスタックのサイズを変更することができる。

```
#define FIQ_STACK_SIZE  (256)   /* FIQ exception stack size */
#define UND_STACK_SIZE  (256)   /* Undefined instruction exception stack size
*/
#define ABT_STACK_SIZE  (256)   /* Abort exception stack size */
```

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 以外の場合は、その値が開始アドレスとなる。ただし、その値がプログラムのデータ領域の最終アドレスより前の場合は、データ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、データ領域と重複することはない。

コンフィギュレーション CNF_SYSTEMAREA_TOP の値が 0 の場合、プログラムが使用しているデータ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。

(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 以外の場合は、その値が終了アドレスとなる。ただし、その値が非キャッシュ領域の開始アドレスより後ろの場合は、非キャッシュ領域の開始アドレスの前のアドレスが終了アドレスとなる。つまり、非キャッシュ領域と重複することはない。

コンフィギュレーション CNF_SYSTEMAREA_END の値が 0 の場合、非キャッシュ領域の開始アドレスの前のアドレスが、終了アドレスとなる。

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。

例外の種別	備考
リセット	
未定義例外	
SVC(スーパーバイザコール)	OS で使用
プリフェッチアボート	
データアボート	
IRQ 割込み	OS の割込み管理機能で管理
FIQ 割込み	

本マイコンは割込みコントローラとして、GIC-400(汎用割込みコントローラ)を搭載する。割込みコントローラは、IRQ および FIQ 割込みの管理を行う。
OS の割込み管理機能が管理対象とするのは IRQ 割込みである。よって、本実装で割込みと言った場合は、IRQ 割込みを指す。

4.2　ベクタテーブル

本マイコンでは、前述の各種例外に対応する例外ハンドラのアドレスを設定したベクタテーブルを有する。
本実装では、リセット時のベクタテーブルは、
kernel¥sysdepend¥cpu¥core¥armv7a¥vector_tbl.S に vector_table 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

割込みコントローラは、割込み優先度を 5bit(0～31)の 32 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)。
外部割込みは優先度 0～31 が割り当て可能である。ただし本実装では、システムタイマ割込みが優先度 1 を使用する。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、通常は 2～31 の 30 段階である。

4.3.2　多重割込み対応

割込みハンドラは、IRQ 割込みがマスクされた状態で実行されるので、そのままでは多

重割込みは受け付けない。
多重割込みに対応するには、割込みハンドラ内で割込みマスクレベルを適切な値に設定したのち、IRQ 割込みを有効とする。

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 CPSR レジスタの I ビットを IRQ 割込み禁止に設定することにより実現する。よって、IRQ 割込み以外の例外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・例外

本 OS の内部で使用する割込み・例外には、以下のように本マイコンの割込み・例外が割り当てられる。該当する割込み・例外は、OS 以外で使用してはならない。

種類	割り当てられる割込み・例外	割込み優先度
システムタイマ割込み	OSTMI0 (IRQ 88)	1
システムコール	SVC 6	-
ディスパッチ要求	SVC 7	-
強制ディスパッチ要求	SVC 8	-
デバッグサポートコール	SVC 9	-
拡張 SVC	SVC 10	-

SVC の割り当ては include¥sys¥sysdepend¥cpu¥core¥armv7a¥sysdef.h で以下のように記述される。

```
#define SVC_SYSCALL             6       /* micro T-Kernel system call */
#define SVC_FORCE_DISPATCH      7       /* force dispatch */
#define SVC_DISPATCH            8       /* task dispatcher */
#define SVC_DEBUG_SUPPORT       9       /* debug support function */
#define SVC_EXTENDED_SVC        10      /* Extended SVC */
```

ユーザプログラムは上記の SVC 例外を使用してはならない。
ただし、本実装では SVC 例外はディスパッチ要求および強制ディスパッチ要求にのみ使用する。これ以外の SVC 割込みは、将来の拡張に備えて定義のみが存在する。

割込みの優先度は include¥sys¥sysdepend¥cpu¥rza2m¥sysdef.h で以下のように定義される。

```
#define INTPRI_SYSTICK          1       /* interrupt priority for SysTick */
```

ユーザプログラムは、割込み優先度 2～31 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は、OS の起動時とタスクの終了時に発行される。

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 IRQ 割込みのみとし、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、割込みコントローラの割込み ID 番号と同一とする。

4.5.2 割込みハンドラの呼び出し

IRQ 割込みが発生すると OS の IRQ 処理ルーチンが実行される。
OS の IRQ 処理ルーチンでは以下の処理が実行される。

(1) CPU の実行モードを SVC モードに変更する
(2) 割込みコントローラの割込みアクノリッジレジスタ(ICCIAR)より、発生している割込み ID 番号を取得する。
(3) 割込み ID 番号に基づき、割込みベクタテーブル knl_intvec_tbl に設定されている割込みハンドラを実行する。割込みハンドラが設定されていなかった場合は、デフォルトハンドラを実行する。

TA_HLNG 属性の割込みハンドラは、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) IRQ スタックの設定

スタックを例外スタック（IRQ 割込み用スタック）に変更する。

(3) FPU の無効化

FPU 使用時（コンフィギュレーション USE_FPU が有効)、割込みハンドラの FPU 制御が有効（INTHDR_DIS_FPU = 1）の場合、FPU を無効化する。割込みハンドラの実行後に FPU は元の状態に戻される。

割込みハンドラの FPU 制御が無効（INTHDR_DIS_FPU = 0）の場合は何も行わない。

(4) 割込みハンドラの実行

割込み ID 番号に基づき、テーブル knl_hll_inthdr_tbl 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、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 の割込み処理を介さずに直接ハンドラが実行される。このため 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¥armv7a¥exc_hdl.c の Default_Handler 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　CPU 割込み制御

CPU の割込み制御を行う。
各 API の実装を以降に記す。

① CPU 割込みの禁止（DI）

CPU 割込みの禁止(DI)は、CSPR レジスタの I ビットを IRQ 割込み禁止に設定する。

② CPU 割込みの許可（EI）

割込みの許可(EI)は、CSPR レジスタの I ビットの値を DI 実行前に戻す。

③ CPU 内割込みマスクレベルの設定(SetCpuIntLevel)

CPU に本機能はないため、未実装である。

④ CPU 内割込みマスクレベルの参照(GetCpuIntLevel)

CPU に本機能はないため、未実装である。

4.6.2　割込みコントローラ制御

割込みコントローラの制御を行う。
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)

割込みイネーブルセットレジスタ(GICD_ISENABLER)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(GICD_IPRIORITYR)に指定された割込み優先度を設定する。ユーザプログラムは、割込み優先度 2～31 を使用しなければならない。

② 割込みコントローラの割込み禁止(DisableInt)

割込みイネーブルクリアレジスタ(GICD_ICENABLER)を設定し、指定された割込みを禁止する。

③ 割込み発生のクリア(ClearInt)

割込み保留クリアレジスタ(GICD_ICPENDR)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)

割込み終了レジスタ(GICC_EOIR)に指定された割込みの終了を設定する。

⑤　割込み発生の検査(CheckInt)
割込み保留クリアレジスタ(GICD_ICPENDR)を参照し、割込みの発生を調べる。

⑥　割込みモード設定(SetIntMode)
割込み構成レジスタ(GICD_ICFGR)に指定された割込みモードを設定する。
指定可能な割込みモードは、include¥tk¥sysdepend¥cpu¥rza2m¥syslib.hに以下のように定義されている。

```
#define IM_LEVEL        0x00    /* high level detection */
#define IM_EDGE         0x01    /* Rising edge detection */
```

設定可能な割込みは、外部端子割込み（割込み番号 480～511）である。

⑦　割込みコントローラの割込みマスクレベル設定(SetCtrlIntLevel)
割込み優先度マスクレジスタ(GICC_PMR)に指定された割込みマスクレベルを設定する。

⑧　割込みコントローラの割込みマスクレベル取得(GetCtrlIntLevel)
割込み優先度マスクレジスタ(GICC_PMR)から設定されている割込みマスクレベルを取得する。。

4.7 OS 管理外割込み

OS のクリティカルセクション中に割込み可能な割込みおよび例外は、OS 管理外割込みとなる。
OS 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、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。
本実装ではクリティカルセクションは IRQ 割込みが無効に設定される。よって、未定義例外、プリフェッチアボート、データアボート、FIQ 割込みが、OS 管理外割込みとなる。

4.8 その他の例外

例外のエントリルーチンは kernel¥sysdepend¥cpu¥core¥armv7a¥exc_entry.S に記述される。
本実装では、IRQ 割込み以外の例外は OS の管理外である。ただし、FPU 使用時（コンフィギュレーション USE_FPU が有効）は、FPU 命令実行による未定義例外は OS の FPU 処理が実行される。
IRQ 割込み以外の例外は、各例外のエントリルーチンから暫定的な例外ハンドラを呼び出す。暫定的な例外ハンドラは、kernel¥sysdepend¥cpu¥core¥armv7a¥exc_hdl.c の以下の関数として実装されている。

例外の種別	関数名
未定義例外	UndefinedInst_Handler
プリフェッチアボート	PrefetchAbort_Handler
データアボート	DataAbort_Handler
FIQ 割込み	FIQ_Handler

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 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。

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）。

暫定的な例外ハンドラは、発生した例外の CPU モードで実行される。スタックは、例外毎に割り当てられた独立のスタックが使用される。

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¥armv7a¥reset_hdl.SのReset_Handlerおよび
kernel¥sysdepend¥cpu¥core¥armv7a¥reset_main.cのreset_main関数に記述される。

リセット処理の手順を以下に示す。

(1) ハードウェア初期化（Reset_Handler）
リセット後の基本的なハードウェア（キャッシュ、MMU、FPU等）の初期化を行った後、reset_main関数を実行する。

(2) 周辺デバイス初期化（knl_startup_hw）
周辺デバイスの初期化を行う。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。

(3) 変数領域(data, bss)の初期化（reset_main）
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（reset_main）
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) L1キャッシュの初期化（L1CacheInit）
L1キャッシュを初期化する。

(6) 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_rza2m¥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¥rza2m¥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	設定	周波数
CPU クロック(I) 周波数の分周率	×1/2 倍	528MHz
内部バスクロック(B) 周波数の分周率	×1/8 倍	132MHz
周辺クロック 1(P1)周波数の分周率	×1/16 倍	66MHz

上記設定は、クロックパルス発振器の周波数制御レジスタ FRQCR を設定することにより行われる。FRQCR の設定値は以下で定義されている。

ファイル名：include¥sys¥sysdepend¥cpu¥rza2m¥sysdef.h

```
#define CPG_FRQCR_INIVAL        0x1012
```

なお、クロックの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 stbcr_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。

名称	機能	用途
SCIF4	シリアル通信	デバッグ用シリアル入出力
OSTM0, OSTM1, OSTM2	OS タイマ	システムタイマ、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 pmode_tbl および pfunc_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。

I/O ポート名	機能名	用途
P90	TxD4	デバッグ用シリアル入出力(T-Monitor)
P91	RxD4	

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_rza2m¥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
sere	シリアル通信 (SCIFA4)

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。

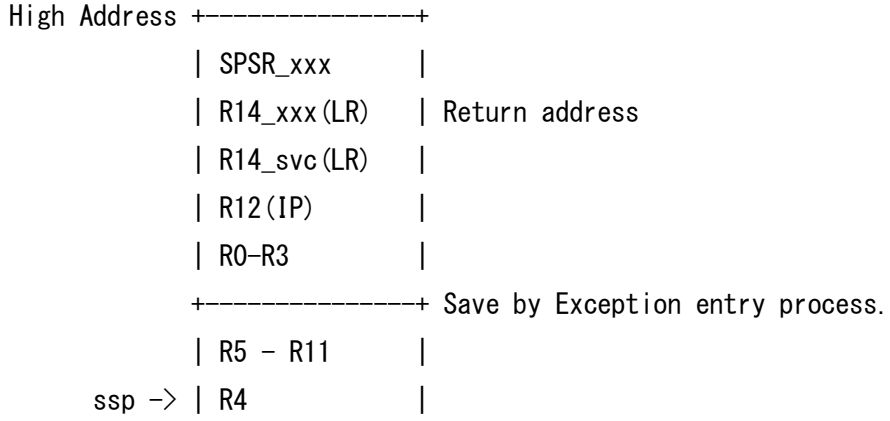
レジスタ	値	補足
PRIMASK	0	割込み許可
R0	第一引数 stacd	タスク起動コード
R1	第二引数 *exinf	タスク拡張情報
R13(SP)	タスクスタックの先頭アドレス	

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

SPSR 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 R11 のレジスタの値はディスパッチャにより保存される。



```
Low Address +---------------+
```

(2) TA_FPU 属性のタスク

TA_FPU 属性のタスクは、タスクスタック領域の先頭に FPU コンテキスト領域を確保し、FPU のレジスタの値を保存する。FPU レジスタの保存は、ディスパッチ時に FPU を使用しているタスクが変更になる際に行われる。

FPU コンテキスト領域は以下の構造体として定義される。

```
typedef struct {
        UW      rsv;            // リザーブ
        UW      fpscr;          // fpscr レジスタ
        UD      d[32];          // FPU レジスタ D0～D31
} FPUContext;
```

その他のレジスタについては、TA_FPU 属性以外のタスクと同様である。

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 OS タイマ(OSTM0)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

本実装ではタイムイベントハンドラの実行中はすべての割込みを許可される。

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 OS タイマ（OSTM1～OSTM2）を使用して 2 個の物理タイマが実装されている。OS タイマは 32 ビットタイマである。
OS 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	OSTM1
2	OSTM2

OS 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 OS タイマ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（OS タイマ）のカウントクロックは P1 クロックである。本実装では周波数 33MHz となる。

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	OSTMI1 割込み	89
2	OSTMI2 割込み	90

各割込みの優先度は include¥sys¥sysdepend¥cpu¥rza2m¥sysdef.h に以下のように定義される。

```
#define INTPRI_OSTM1    5
#define INTPRI_OSTM2    5
```

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。具体的には、アドバンスド SIMD 拡張機能と VFP 拡張機能が稼働状態となる。

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 {
        UD      d[32];  /* FPU General purpose register d0-d31 */
        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　タスクからの FPU の使用

TA_FPU 属性のタスクは FPU の使用が可能である。非 TA_FPU 属性のタスクから FPU を使用しようとした場合は未定義命令例外が発生する。
タスクの実行開始時は、FPU は常に無効である。タスクのコンテキストで FPU が最初に実行されると、未定義命令例外の処理が実行され、タスクが TA_FPU 属性であれば FPU が有効化される。
非 TA_FPU 属性のタスクへのディスパッチで FPU は無効化される。また、TA_FPU 属性のタスクであっても、他のタスクが FPU を使用していた場合は、いったん FPU は無効化される。ディスパッチ後にそのタスクの中で FPU 命令が実行された時、未定義命令例外の処理にて再び FPU が有効化される。その際、FPU レジスタの保存と復帰が行われる。

8.1.4　割込みハンドラからの FPU 使用

割込みハンドラの起動時に FPU レジスタの退避は行われない。よって、原則として割込みハンドラ内の FPU の使用は禁止である。
もし割込みハンドラ内で FPU を使用する場合は、FPU の有効化および FPU レジスタの退避を割込みハンドラにて行わなければならない。また、割込みハンドラから復帰の際には FPU の状態および FPU レジスタの内容を基の状態に復帰させなくてはならない。

割込みハンドラ内での意図しない FPU の使用を検出するため、割込みハンドラの高級言語ルーチンにて FPU の無効化を行うことができる。本処理は include¥sys¥sysdepend¥cpu¥core¥armv7a¥sysdef.h の INTHDR_DOS_FPU の定義で制御できる。
INTHDR_DOS_FPU=1 の場合、割込みハンドラの呼び出し時に FPU は無効化される。
INTHDR_DOS_FPU=0 の場合は OS は何も行わない。割込みハンドラ内で FPU を使用する場合は、INTHDR_DOS_FPU=0 とする必要がある。標準の定義では INTHDR_DOS_FPU=1 である。

(3) タイムイベントハンドラ
タイムイベントハンドラにおけるFPUの使用は、割込みハンドラに準ずる。

(4) その他の例外ハンドラ
その他の例外ハンドラの実行時には、OSはFPUに関する操作を行わない。必要に応じて、FPUの無効化やレジスタの退避などを行うこと。

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	内蔵シリアルI/F (SCIFA4)
ボーレート	115200bps
データ形式	data 8bit, stop 1bit, no parity

以上