μT-Kernel3.0
RX65N マイコン向け
実装仕様書

Version.01.00.01

2022.10.07

変更履歴

版数(日付)	内　容
1.00.01 (2022.10.07)	● 1.4 関連ドキュメント バージョン番号等を更新 ● 5.1 リセット処理 (2)ベクタテーブルの移動 誤記修正 (誤) USE_NOINIT (正)USE_STATIC_IVT
1.00.00 (2022.06.30)	● 初版

目次

1. 概要

1.1 目的

本書はRX65Nマイコン向けの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(V3.00.06)のうち、RX65Nマイコン向けの実装部分である。ハードウェアに依存しない共通の実装仕様は、μT-Kernel3.0実装仕様書を参照のこと。

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

1.2 対象ハードウェア

実装対象のハードウェアは以下の通りである。

分類	名称	備考
マイコン	RX600シリーズ RX65Nグループ R5F565NE	ルネサス エレクトロニクス製

1.3 ターゲット名

RX65Nマイコンのターゲット名は以下とする。

分類	名称	対象
対象CPU	CPU_RX65N	RX65Nシリーズ
対象CPUアーキテクチャ	CPU_CORE_RXV2	RXv2CPUコア

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.1.00.05)	TRON フォーラム TEF033-W007-221007
搭載マイコン	RX65N グループ、RX651 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
└ cpu                       CPU 依存部
   ├ rx65n                      RX65N マイコン依存部
   └ core                   コア依存部
      └ rxv2                    RXv2 CPU コア依存部
```

「RXv2 コア依存部」は、RXv2 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。
「RX65N 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。

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	RXv2
ROM	2048KB(内蔵フラッシュ ROM)
RAM	256KB(内蔵 RAM) + 384KB(拡張 RAM)

2.2 実行モードと保護レベル

RXv2CPU コアは、プログラムの動作モードとして、スーパーバイザモードとユーザモードの二つのプロセッサモードを持つ。
本実装では、実行モードはスーパーバイザモードのみを使用する。

OS が提供する保護レベルは、実行モードがスーパーバイザモードのみなので、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

OS の API（tk_set_reg、 tk_get_reg）を用いて実行中のタスクのコンテキストのレジスタ値を操作できる。
API で使用するマイコンのレジスタのセットは以下のように定義する。

(1) 汎用レジスタ

```
typedef struct t_regs {
        VW      r[15];          /* 汎用レジスタ R0-R15 */
} 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*/
        UW      psw;            /* ステータスレジスタ */
} T_EIT;
```

(3) 制御レジスタ T_CREGS

```
typedef struct t_cregs {
        void    *ssp;   /* System stack pointer R0 */
} 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）。

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_rx231 /power.c に空関数として記述されている。なお、本関数に適切な省電力処理を記述し、プロファイル TK_SUPPORT_LOWPOWER を TRUE に指定すれば、OS の省電力機能(tk_set_pow API)は使用可

能となる。

2.5 コプロセッサ対応

本マイコンは FPU および DSP の機能を有する。コンフィギュレーション USE_FPU および USE_DSP により、OS のコプロセッサ対応機能を有効とすることができる。

(1) FPU

コンフィギュレーション USE_FPU を TURE に指定（初期値 TRUE）すると、OS で FPU 対応の機能が有効となり、コプロセッサ番号 0 が割り当てられる。
FPU が有効の場合、タスク属性に TA_FPU または TA_COP0 が指定可能となる。

FPU は 32 ビットの 1 個の専用レジスタ（FPSW レジスタ）を持つ。本 OS 実装では、FPU が有効の場合は、割込みハンドラ実行やタスクのディスパッチの際には必ず FPU 用レジスタをスタックに退避する（「6.3 タスク・コンテキスト情報」参照)。よって、TA_FPU 属性の指定に関わらず各タスクやハンドラ内で FPU は使用が可能である。ただし、互換性や移植性の観点から、FPU を使用するタスクは TA_FPU 属性を指定することを推奨する。

(2) DSP

コンフィギュレーション USE_DSP を TURE に指定（初期値 FALSE）すると、OS で DSP 対応の機能が有効となり、コプロセッサ番号 1 が割り当てられる。
DSP が有効の場合、タスク属性に TA_COP1 が指定可能となる。

DPS は 72 ビットの 2 個の専用レジスタ(ACC0, ACC1)を持つ。本 OS 実装では、DSP が有効の場合は、タスクのディスパッチの際には必ず DSP 用レジスタをスタックに退避する（「6.3 タスク・コンテキスト情報」参照)。よって、TA_COP1 属性の指定に関わらず各タスクで DSP は使用が可能である。ただし、互換性や移植性の観点から、DSP を使用するタスクは TA_COP1 属性を指定することを推奨する。
なお、割込みハンドラ実行の際には DSP 用レジスタの退避は行われない。よって、各ハンドラでは DSP 命令を使用してはいけない。もし、DSP 命令を使用したい場合はハンドラの実行時にレジスタの退避と復帰の処理を行わなければならない。

FPU や DSP が有効の場合、コプロセッサレジスタ操作 API が使用可能となる（「8.1.1 コプロセッサ関連 API」参照)。

3. メモリ

3.1 メモリモデル

RXv2 コアは 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、シングルチップモードまたは内蔵 ROM 有効拡張モードとする。
以下にマイコンのアドレス・マップを記す。なお、リザーブ領域は記載しない（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
0x0000 0000 0x0003 FFFF	メイン RAM	256	
0x0008 0000 0x000F FFFF	周辺 I/O レジスタ		
0x0010 0000 0x0010 7FFF	内蔵 ROM	32	データフラッシュメモリ
0x007F C000 0x007F FFFF	周辺 I/O レジスタ		
0x0080 0000 0x0085 FFFF	拡張 RAM	384	
0xFFE0 0000 0xFFFF FFFF	内蔵 ROM	2048	コードフラッシュメモリ

3.3 OS のメモリマップ

本実装では、マイコンの内蔵 ROM(コードフラッシュメモリ)およびメイン RAM を使用する。他の領域のメモリは使用しないのでユーザが使用することは可能である。
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。
リセットベクタおよび例外ベクタテーブルは内蔵 ROM 上に配置される。

割込みベクタテーブルおよび割込みの高級言語ハンドラテーブルは、リセット時は内蔵ROM上にあるが、OSの初期化処理にてRAM上に再配置される。ただし、コンフィグレーションUSE_STATIC_IVTを有効にすることにより、RAM上への再配置を禁止することができる（初期値は無効）。再配置を禁止した場合、APIによる割込みハンドラの登録が不可となる。

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

(1) 内蔵ROMのメモリマップ

アドレス※ （上段：開始 下段：終了）	種別	内容
0xFFE0 0000 -	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
- -	割込みベクタ テーブル	割込みのベクタテーブル リセット時のみ有効
- -	割込み高級言語 ハンドラテーブル	割込みの高級言語ハンドラの登録テーブル リセット時のみ有効
- -	定数データ	C言語の定数データなどが配置される領域
0xFFFF FF80 -	例外ベクタ テーブル	例外のベクタテーブル リセット時のみ有効
0xFFFF FFFC 0xFFFF FFFF	リセットベクタ	

(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
0x0000 0000 -	割込みベクタ テーブル	割込みのベクタテーブル OSの初期化後に有効
- -	割込み高級言語 ハンドラテーブル	割込みの高級言語ハンドラの登録テーブル 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
- 0x0003 FFFF	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

3.4 スタック

マイコンには、ISP(割込みスタックポインタ)と USP(ユーザスタックポインタ)の二種類のスタックポインタが存在する。ただし、本実装では ISP のみを使用し、USP 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

3.5 OS 内の動的メモリ管理

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

(1) OS 管理メモリ領域の開始アドレス

コンフィギュレーション CNF_SYSTEMAREA_TOP の値が 0 の場合、RAM 上のプログラムのデータ領域(BSS 領域)の最終アドレスの次のアドレスが、開始アドレスとなる。

値が 0 以外の場合は、その値が開始アドレスとなる。ただし、その値が RAM 上のプログラムのデータ領域(BSS 領域)の最終アドレス以下の場合は、BSS 領域の最終アドレスの次のアドレスが開始アドレスとなる。つまり、BSS 領域と重複することはない。

(2) OS 管理メモリ領域の終了アドレス

コンフィギュレーション CNF_SYSTEMAREA_END の値が 0 の場合、RAM 上の例外スタックの開始アドレスの前のアドレスが、終了アドレスとなる。

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

4. 割込みおよび例外

4.1 マイコンの割込みおよび例外

RXv2 コアには以下の例外が存在する。OS の割込み管理機能が管理するのは、割込みと無条件トラップのみである。

種別	優先順位	備考
リセット	1	OS で使用
ノンマスカブル割込み(NMI)	2	
割込み	3	255 要因（ベクター番号 0～255） OS の割込み管理機能で管理
アクセス例外（命令）	4	
未定義命令例外	5	
特権命令例外		
無条件トラップ	6	OS の割込み管理機能で管理
アクセス例外（オペランド）	7	
浮動小数点例外	8	

※優先順位は数字が小さいほど高い。

4.2 ベクタテーブル

本マイコンは、例外ベクタテーブルと割込みベクタテーブルを有する（各テーブルの具体的な仕様はマイコンのユーザマニュアルを参照のこと）。
また、OS 内の管理用に HLL 割込みハンドラテーブルが存在する。

(1) 例外ベクタテーブル

例外ベクタテーブルは、リセット、割込み、無条件トラップ以外の例外ハンドラのアドレスが設定される。

本実装では、例外ベクタテーブルは、kernel/sysdepend/cpu/core/rxv2/vector_tbl.c に exvect_tbl として定義される。

例外ベクタテーブルは、OS の機能により動的に変更されることはない。

(2) 割込みベクタテーブル

割込みベクタテーブルは、割込みおよび無条件トラップの割込みハンドラのアドレスが設定される。

本実装では、リセット時の割込みベクタテーブルは、
kernel/sysdepend/cpu/rx65n/intvect_tbl.c に knl_int_vect_rom として定義される。

ただし、OS の初期化処理において、割込みベクタテーブルは RAM 上にコピーされ、以降そちらが使用される。RAM 上の割込みベクタテーブルの領域は、kernel/sysdepend/cpu/core/rxv2/interrupt.c に knl_int_vect_ram として定義される。

(3) HLL 割込みハンドラテーブル

TA_HLANG 属性の割込みハンドラ（高級言語（C 言語）で記述された割込みハンドラ）は、OS 内の高級言語対応ルーチンを経由して呼び出される。TA_HLANG 属性の割込みハンドラの実行アドレスは、HLL 割込みハンドラテーブルに登録される。

HLL 割込みハンドラテーブルは、kernel/sysdepend/cpu/rx65n/hllint_tbl.c に knl_hll_inthdr_rom として定義される。

ただし、OS の初期化処理において、HLL 割込みハンドラテーブルは RAM 上にコピーされ、以降そちらが使用される。RAM 上の HLL 割込みベクタテーブルは、kernel/sysdepend/cpu/core/rxv2/interrupt.c に knl_hll_inthdr_ram 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

本マイコンの割込みコントローラ(ICUb)は、割込み優先度を 1～15 の 15 段階に設定できる（優先度の数字の大きい方が優先度は高い）。

OS の API により割込みは優先度 1～15 が割り当て可能である。ただし、最高優先度である 15 は OS の割込み処理を阻害するため、ユーザプログラムからの使用可能は許されない。よって、ユーザプログラムから使用可能な外部割込みの優先度は、1～14 の 14 段階である。

4.3.2 多重割込み対応

OS の割込みの高級言語対応ルーチンは多重割込みに対応している。

高級言語対応ルーチンから TA_HLANG 属性の割込みハンドラを実行する際に、その割込みより優先度の高い割込みは受付可能となる。よって、割込みハンドラ実行中により優先度の高い割込みが発生した場合は、割込みハンドラの実行が優先度の高い割込みハンドラに割り込まれる。

もし、割込みハンドラの実行中に他の割込みをマスクしたい場合は OS の API を使用し、割込みのマスクレベルを変更する必要がある。

なお、TA_ASM 属性の割込みハンドラは、OS の高級言語対応ルーチンを介さないため、j ハンドラののプログラム中で必要に応じて割込みコントローラの制御を行う必要がある。

4.3.3　クリティカルセクション

OS の API 実行中などのクリティカルセクションでは、すべての優先度の割込みはマスクされる。ただし、割込み以外の例外はマスクされない。
本実装では、クリティカルセクションは CPU の PSW レジスタに最高割込み優先度 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¥rx231¥sysdef.h にて以下のように定義される。

```
#define INTPRI_MAX_INT_PRI      15
```

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
SVC 割込み	2～4	無条件トラップ	-
システムタイマ割込み	28	CMI0	15
ディスパッチ要求	1	無条件トラップ	-
強制ディスパッチ要求	1	無条件トラップ	-

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。また、ディスパッチ要求および強制ディスパッチ要求には割込みを使用しない。システムタイマ割込み以外の割込みは、将来の拡張に備えて定義のみが存在する。

各割込み・例外の割り当ては/include/sys/sysdepend/cpu/rx65n/sysdef.h で以下のように定義される。

```
#define INTNO_SYS_DISPATCH      1       /* Dispatch (reserved)*/
#define INTNO_SYS_SVC           2       /* System call (reserved) */
#define INTNO_SYS_RET_INT       3       /* System call:tk_ret_int (reserved) */
#define INTNO_SYS_DGSPT         4       /* Debugger support (reserved) */
#define INTNO_SYS_TICK          28      /* Systten timer tick */
```

各割込みの優先度は/include/sys/sysdepend/cpu/rx65n/sysdef.h で以下のように定義さ

れる。

```
#define INTLEVEL_SYS_TICK       15
```

ユーザプログラムは、割込み優先度 1～14 を使用しなければならない。

4.5 μT-Kenrel/OS の割込み管理機能

本実装では、割込み管理機能はマイコンの割込みおよび無条件トラップを管理対象とし、割込みハンドラの管理を行う。その他の例外については対応しない。

4.5.1 割込み番号

割込み番号 0～255 は、マイコンの割込み番号（ベクタ番号）と同一とする。具体的な番号はマイコンのマニュアルを参照のこと。
割込み番号 256～447 は、グループ割込みの各要因に割り当てられる。グループ割込みについては「4.5.4 グループ割込み」を参照。

割込み番号	対応する割込み
0 ～ 255	ベクタ番号 0 ～ ベクタ番号 255
256 ～ 287	グループ割込み GROUPBE0 の要因 0～31
288 ～ 319	グループ割込み GROUPBL0 の要因 0～31
320 ～ 351	グループ割込み GROUPBL1 の要因 0～31
352 ～ 383	グループ割込み GROUPBL2 の要因 0～31
384 ～ 415	グループ割込み GROUPAL0 の要因 0～31
416 ～ 447	グループ割込み GROUPAL1 の要因 0～31

4.5.2 割込みハンドラ属性

割込みハンドラは、TA_HLNG 属性と TA_ASM 属性のいずれかが登録時にしてできる。

TA_HLNG 属性の割込みハンドラは、割込みの発生後、OS の高級言語対応ルーチンを経由して呼び出される。高級言語対応ルーチンは kernel/sysdepend/cpu/rx231/hllint_ent.S に記述されている。
高級言語対応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) 多重割込み対応

処理開始時にシステム変数 knl_int_nest をインクリメントし、終了時にデクリメントする。本変数は割込みのネスト数を記録する。

(3) スタックの切り替え

使用するスタックを例外スタックに切り替える。

(4) 割込みハンドラの実行

テーブル knl_hll_inthdr に登録されている発生した割込みに該当する割込みハンドラを実行する。また、その際に多重割込みを有効とする。

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。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 の高級言語対応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¥rxv2¥exc_hdr.c に Default_Handler 関数として定義されている。
デフォルトハンドラは、プロファイル USE_EXCEPTION_DBG_MSG を有効にすることにより、デバッグ用の情報を出力する（初期設定は有効である）。

必要に応じてユーザがデフォルトハンドラを記述することにより、未定義割込み発生時の処理を行うことができる。デフォルトハンドラは weak 宣言がされているので、ユーザが同名の関数を作成しリンクすることにより、上書きすることができる。

4.5.4 グループ割込み

グループ割込み（GROUPBE0、GROUPBL0、GROUPBL1、GROUPBL2、GROUPAL0、GROUPAL1）は、一つの割込みに対して、最大 32 本の割込み要求が割り当てられている。
本実装では、グループ割込みの各要因について、割込み番号 256 以降を割当て、それぞれに割り込みハンドラを登録可能とする。
各グループ割込みには、OS が用意したグループ割込みハンドラが登録されており、そのハンドラの処理で発生したグループ割込みの要因を調べ、対応する割込みハンドラを実行する。よって、グループ割込みの各要因も通常の割込みと同様に割込みハンドラが登録できる。

グループ割込みの各要因の割込みハンドラは、TA_HLNG 属性のみが指定可能である。
なお、グループ割込み自体の割込みハンドラに対して、ユーザ定義の割込みハンドラを登録することも可能である。この場合は、割込み番号 256 以降に登録した割り込みハンドラは無効となる。

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(ICUb)の制御を行う。
各 API の実装について以降に記す。

4.6.1 割込みの優先度

割込みの優先度は 1～15 であり、数字が大きいほど優先度が高い。最高優先度 15 は OS の動作を阻害するため、原則として指定してはならない。よって、ユーザが使用できる優先度は 1～14 である。

4.6.2 CPU 割込み制御

CPU 割込み制御は、マイコンの BASEPRI レジスタのみを制御して実現する。PRIMASK および FAULTMASK レジスタは使用しない。

① CPU 割込みの禁止（DI）

CPU 割込みの禁止(DI)は、PSW レジスタに最高割込み優先度 INTPRI_MAX_INT_PRI を設定し、すべて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、PSW 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、PSW レジスタのプロセッサ割込み優先レベル(IPL)を設定する。
割込みマスクレベルは 1 から 15 の値が指定可能である。指定したマスクレベル以下の優先度の割込みはマスクされる。また、割込みマスクレベルに INTLEVEL_EI を指定された場合は、すべての割り込みは許可され、INTLEVEL_DI を指定した場合はすべての割込みは禁止される。

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、PSW レジスタのプロセッサ割込み

優先レベル(IPL)を参照する。

4.6.3　割込みコントローラ制御

マイコン内蔵の割込みコントローラ（ICUb）の制御を行う。

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(ICUb)の割り込み要求許可レジスタ(IER)を設定し、指定された割込みを許可する。

同時に割り込み要因プライオリティレジスタ(IPR)に指定された割込み優先度を設定する。割込み優先度は 1 から 14 の値が使用可能である。

グループ割込み要因に割り当てられた割込み（割込み番号 256 以降）が指定された場合は、グループ割り込み要求許可レジスタを設定し、指定された割込み要因を有効とする。割込み優先度の指定は無視される。なお、元となるグループ割込み自体の許可は行わないので、グループ割込み要因に割り当てられた割込みを有効にする場合は、まずグループ割込み自体を本 API で有効に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)

割込みコントローラ(ICUb)の割り込み要求許可レジスタ(IER)を設定し、指定された割込みを禁止する。

グループ割込み要因に割り当てられた割込み（割込み番号 256 以降）は、グループ割り込み要求許可レジスタを設定し、指定された割込み要因を無効とする。なお、元となるグループ割込み自体は無効にしないので、必要であればグループ割込みに対して本 API を実行する必要がある。

③　割込み発生のクリア(ClearInt)

割込みコントローラ(ICUb)の割り込み要求レジスタ(IR)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。

ただし、クリア可能な割込みはエッジ検出のみであり、また割込み要求先が DTC または DMAC の場合はクリアしてはならない。

グループ割込み要因に割り当てられた割込み（割込み番号 256 以降）は、グループ割り込み要求クリアレジスタを設定し、指定された割込み要因をクリアとする。指定可能なグループ割込み要因はエッジ検出のみである。なお、元となるグループ割込み自体のクリアは行わないので、必要であればグループ割込みに対して本 API を実行する必要がある。

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る。

⑤　割込み発生の検査(CheckInt)
割込み発生の検査(CheckInt)は、割込みコントローラ(ICUb)の割り込み要求レジスタ(IR)を参照することにより実現する。
グループ割込み要因に割り当てられた割込み（割込み番号 256 以降）は、さらにグループ割込み要求レジスタの値を調べる。

⑥　割込みモード設定(SetIntMode)
割込みコントローラ(ICUb)の IRQ コントロールレジスタ(IRQCR)を指定された割込みモードに設定する。
指定可能な割込みモードは、include¥tk¥sysdepend¥cpu¥rx231¥syslib.h に以下のように定義される。

```
#define IM_LEVEL        0x0004          /* Level trigger */
#define IM_EDGE         0x0000          /* Edge trigger */

#define IM_HI           0x0000          /* H level/Interrupt at rising edge */
#define IM_LOW          0x0001          /* L level/Interrupt at falling edge */
#define IM_BOTH         0x0003          /* L level/Interrupt at both edge */
```

ただし、割込みの種別により指定可能なモードが定められているので、マイコンの仕様書を参照のこと。
グループ割込み要因に割り当てられた割込み（割込み番号 256 以降）には使用できない。

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(ICUb)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(ICUb)に本機能はないため、未実装である。

4.6.4　選択型割込みの対応
選択型割込みに対応して選択型割込み設定 API setPERI を提供します。本 API はμT-Kernel 3.0 仕様には含まれません。

形式	ER SetPERI(UINT intno, UINT fctno)
引数	UINT intno　　選択型割込みの割込み番号 UINT fctno　　設定する割込み要因番号
戻り値	エラーコード
機能	intno で指定した選択型割込みに、fctno で指定した割込み要因番号を設定する。

4.7 OS 管理外割込み

最高外部割込み優先度 INTPRI_MAX_INT_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、管理外割込みのハンドラ中で 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_INT_PRI は最高優先度 15 と定義されている。よって、管理外割込みは存在しない。ただし、割込みおよび無条件トラップ以外の例外は、管理外割込みと同じ扱いとなる。

INTPRI_MAX_INT_PRI を変更することにより管理外割込みをつくることはできる。ただし、管理外割込みハンドラでは原則として OS 機能は使用できないこと、および、OS の実行よりも優先して実行されることを十分考慮する必要がある、

4.8 その他の例外

割込みおよび無条件トラップ以外の例外は、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¥rxv2¥exc_hdr.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
ノンマスカブル割込み(NMI)	NMI_Handler
アクセス例外	AccessInst_Handler
未定義命令例外	UndefinedInst_Handler
特権命令例外	SuperVisorInst_Handler
浮動小数点例外	FloatingPoint_Handler

暫定的の例外ハンドラは、プロファイル USE_EXCEPTION_DBG_MSG を有効にすることにより、デバッグ用の情報を出力する（初期設定は有効である）。

必要に応じてユーザが例外ハンドラを記述することにより、各例外に応じた処理を行うことができる。暫定的の例外ハンドラは weak 宣言がされているので、ユーザが同名の関数を作成しリンクすることにより、上書きすることができる。

各例外ハンドラは、例外発生時に OS を介さず直接実行される。これらの例外ハンドラは OS 管理外例外割込みと同じ扱いとなる。

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。
リセット処理のエントリは kernel/sysdepend/cpu/core/rxv2/reset_hdl.S の Reset_Handler である。Reset_Handler では、マイコンの最小限の初期化を実行したのち、kernel/sysdepend/cpu/core/rxv2/reset_main.c の reset_main 関数を実行する。

reset_main 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。
knl_startup_hw 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) ベクタテーブルの移動
割込みベクタテーブルおよび HLL 割込みハンドラテーブルを ROM から RAM に移動し、変更可能とする。
ただし、コンフィギュレーションで USE_STATIC_IVT が指定された場合が、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_STATIC_IVT 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) メモリの OS 管理領域の決定
コンフィギュレーションに従いメモリの空き領域から、OS 管理領域を決定する。

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

5.2 ハードウェアの初期化および終了処理

OS の起動および終了に際して、マイコンおよび周辺デバイスの初期化、終了処理を行う。以下の関数を対象とするハードウェアおよびアプリケーションに応じて実装する。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。

関数名	内容
knl_startup_hw	ハードウェアの初期化（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。
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。

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	○	DSP
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
PSW	0x00010000	割込み許可
R1	第一引数 stacd	タスク起動コード
R2	第二引数 *exinf	タスク拡張情報
SP(R0)	タスクスタックの先頭アドレス	
fpsw	0x00000100	FPU 使用時のみ設定

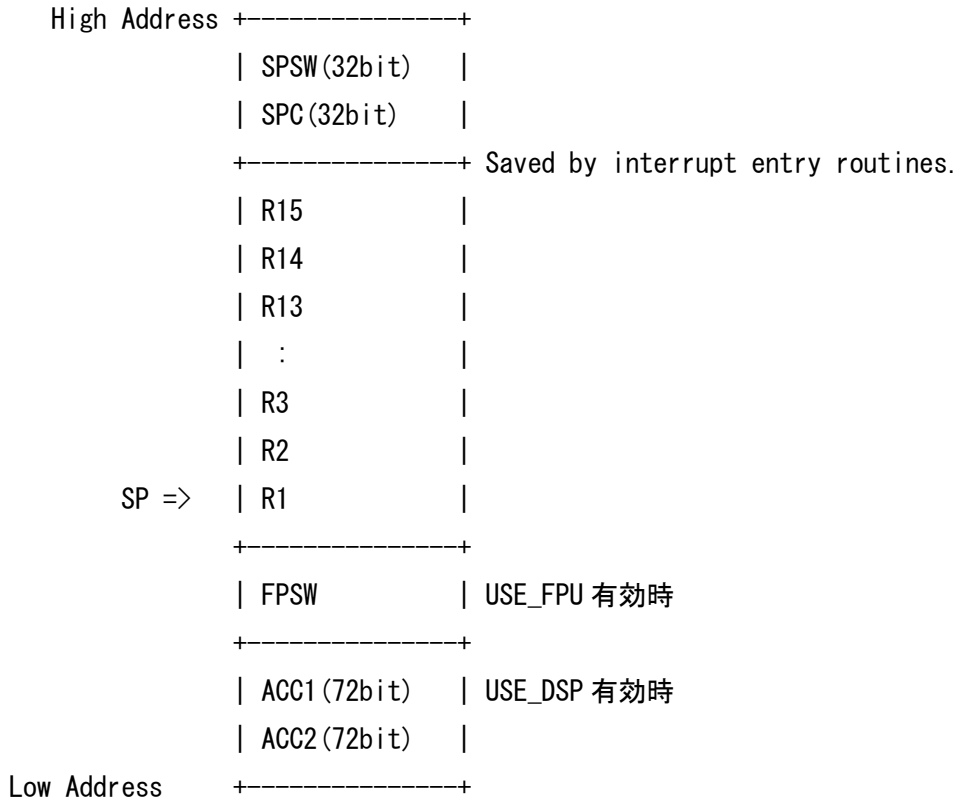
6.3 タスク・コンテキスト情報

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。

SPSW および SPC 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。R1 から R15 のレジスタの値はディスパッチャにより保存される。

コンフィギュレーション USE_FPU が有効の場合、FPSW レジスタの値がディスパッチャにより保存される。

コンフィギュレーション USE_DSP が有効の場合、ACC1 および ACC2 レジスタの値がディスパッチャにより保存される。



7. 時間管理機能

7.1 システムタイマ

本実装では、マイコン内蔵のコンペアマッチタイマ・ユニット 0(CMT0)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/rx65n/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 8 ビットタイマ（TMR）を 16 ビットカウントモードで使用して 2 個の物理タイマが実装されている。
8 ビットタイマは 4 チェンネルが存在する。そのうち TMR0 と TMR1、TMR2 と TMR3 の組み合わせで 16 ビットカウントモードとし、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	TMR0、TMR1
2	TMR2、TMR3

MTR 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 MTR 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¥rx65n¥sysdef.h に以下のように定義される。

```
#define TMR01_CLOCK             (0x08)          // Count PCLK
#define TMR23_CLOCK             (0x08)          // Count PCLK
```

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

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 TMRB のコンペアマッチ割込み(CMIA)およびオーバーフロー割込み(OVI)を使用する。これらの割込みは選択型割込みなので、INTB128 から INTB207（割込み番号 128～207）の間のいずれかの割込みに割り付けることができる。
選択型割込みの割り付けは include¥sys¥sysdepend¥cpu¥rx65n¥sysdef.h に以下のように定義されている。

```
#define INTNO_PTMR1             128     // INTB128
#define INTNO_PTMR2             129     // INTB129
```

割り当てる選択型割込みは必要に応じて変更可能である。

各割込みの優先度は include¥sys¥sysdepend¥cpu¥rx65n¥sysdef.h に以下のように定義される。

```
#define INTPRI_PTMR1            5       // 物理タイマ 1
#define INTPRI_PTMR2            5       // 物理タイマ 2
```

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

8.1 コプロセッサ関連

8.1.1 コプロセッサ関連 API

プロファイル USE_FPU および USE_DSP により、コプロセッサが有効な場合、OS の API (tk_set_cop、 tk_get_cop) を用いて実行中のタスクのコンテキストのコプロセッサ用レジスタ値を操作できる。

コプロセッサ用のレジスタは以下のように定義される。

```
typedef struct t_cop0reg {
        VW      fpsw;           /* Floating Point Status register */
} T_COP0REG;

typedef struct t_cop1reg {
        UW      acc0lo;
        UW      acc0hi;
        VW      acc0gu;         /* Accumulator 0 */
        UW      acc1lo;
        UW      acc1hi;
        VW      acc1gu;         /* Accumulator 1 */
} T_COP1REG;

typedef struct t_copregs {
        T_COP0REG       cop0;   /* COP0: FPU */
        T_COP1REG       cop1;   /* COP1: DSP */
} T_COPREGS;
```

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）。

以上