μT-Kernel 実装仕様書

AT91(ARM7TDMI) 版

Version 1.01.02

2013年04月

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

はじめに

本書では、μT-Kernel の特定のボードへの実装の仕様を記載する。

対象とするボードは T-Engine Appliance / AT91M55800A である。
対象とするμT-Kernel ソースコードのバージョンは 1.01.01 版 である。
準拠するμT-Kernel 仕様書のバージョンは 1.01.01 版 である。

本書に記された仕様は、μT-Kernel 仕様のハードウェアに依存した実装依存部に相当する。
μT-Kernel の仕様については μT-Kernel 仕様書を参照。
また、ボードや CPU などハードウェアの仕様については該当する各仕様書を参照。

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

［目次］

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

1. CPU

1.1 ハードウェア仕様

CPU : AT91M55800A (ARM7TDMI Core)
Atmel Corporation
ROM : 4 MB(FlashROM)
RAM : 内蔵 SRAM 8 KB
外付け SRAM 2 MB

1.2 保護レベルと動作モード

保護レベルは、CPU の動作モードに以下のように対応する。

保護レベル	動作モード
3	(保護レベル 0 として扱う)
2	(保護レベル 0 として扱う)
1	(保護レベル 0 として扱う)
0	SVC:スーパーバイザモード

・どの保護レベルが指定されても保護レベル 0 として扱う。
・ユーザーモード(USR) と システムモード(SYS) の CPU 動作モードは使用しない。

1.3 Thumb 命令セットの使用

Thumb 命令セットを使用する場合は、「 ARM/Thumb Interworking 」に従ってプログラミングする必要がある。C 言語(GNU C)の場合は、コンパイル時に -mthumb-interwork -mthumb オプションを指定する。

タスクやハンドラを Thumb 命令セットを使用して書く場合、そのアドレスの最下位ビットに 1 を指定する。 ARM 命令セットの場合は 0 を指定する。これは通常リンカが自動的に行うため、プログラマは特に意識する必要はない。

カーネル内では、Thumb 命令セットは使用しない。

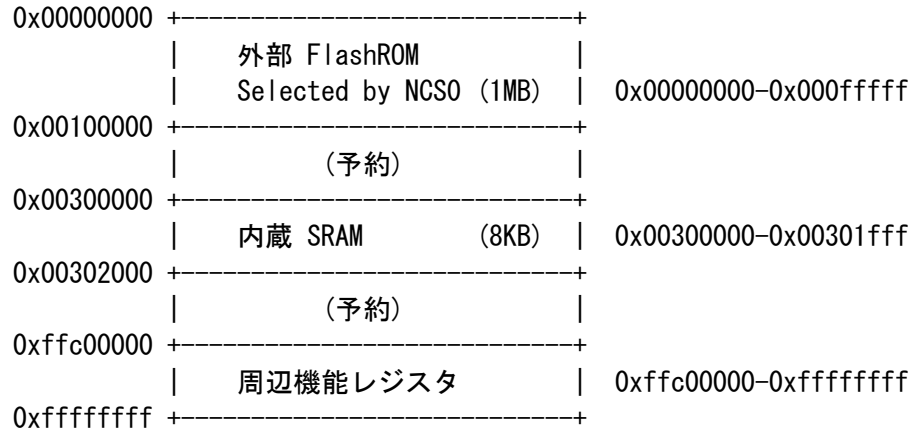
Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

2. メモリマップ

2.1 全体メモリマップ

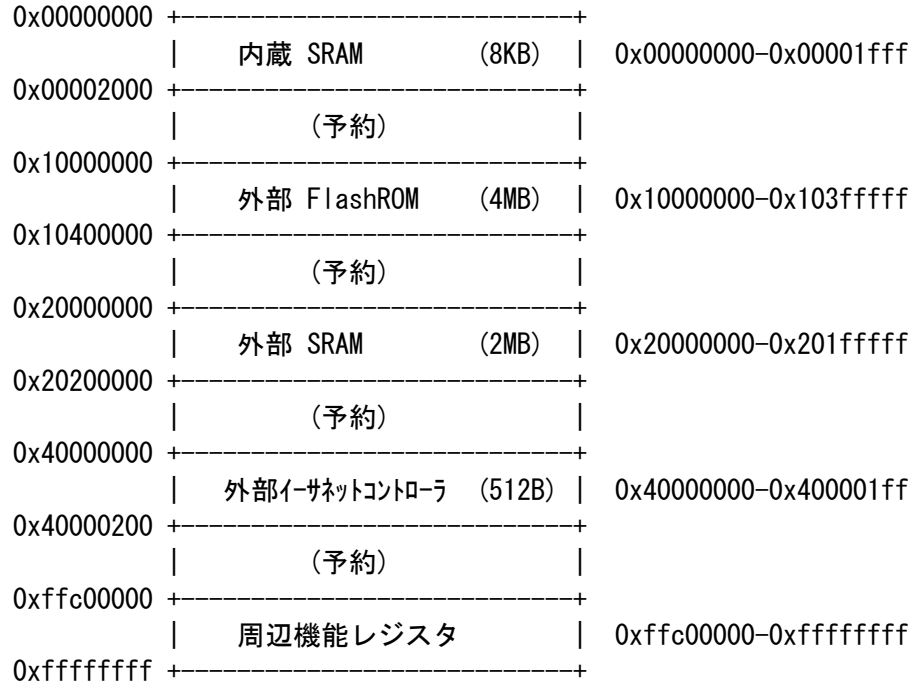
システム全体のメモリマップを以下に示す。

【 REMAP 前 】



REMAP 前は、内蔵 SRAM 、NCS0 に接続されている外部 FlashROM 、周辺機能レジスタ にのみアクセス可能。

【 REMAP 後 】

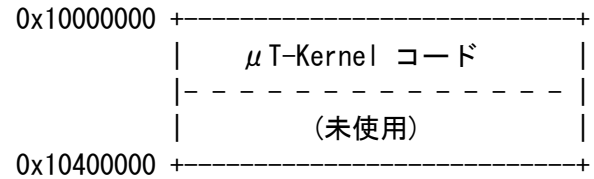


REMAP 後は、全てのメモリ空間にアクセス可能。

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

2.2 ROM 領域メモリマップ

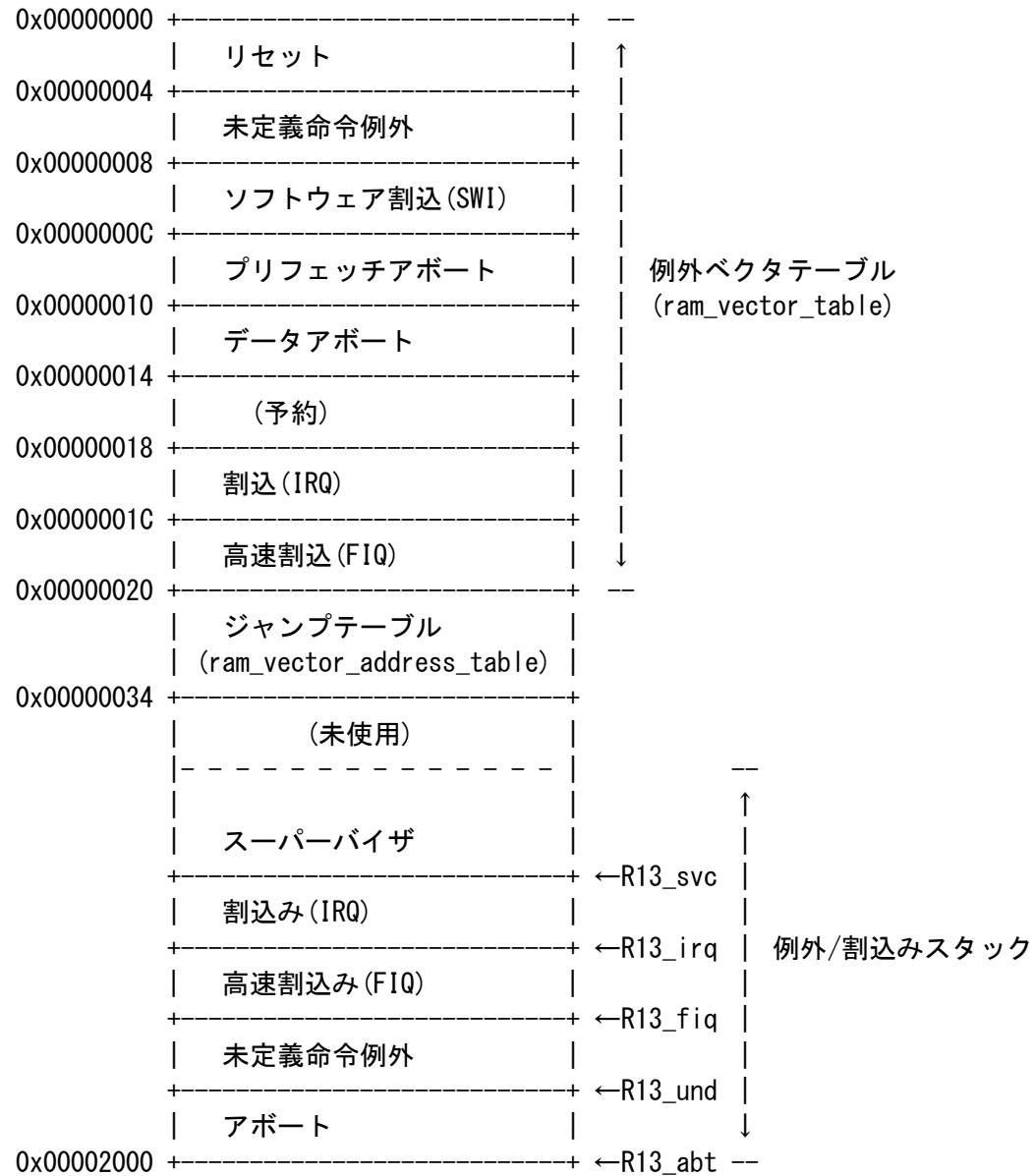
外部 FlashROM は 4MB の領域が実装されている。外部 FlashROM 領域のメモリマップを以下に示す。



外部 FlashROM の下位アドレスに μT-Kernel コード を配置する。

2.3 内蔵 SRAM 領域メモリマップ

内蔵 SRAM は 8KB の領域が実装されている。内蔵 SRAM 領域のメモリマップを以下に示す。



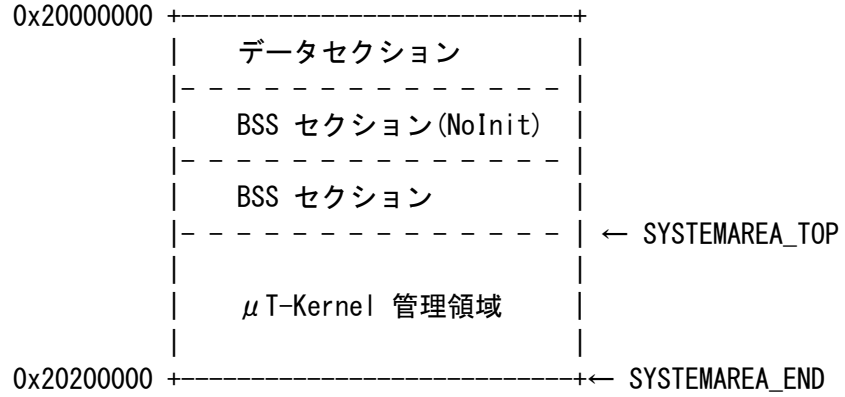
Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

内蔵 SRAM の下位アドレスに ベクタテーブル と アドレステーブル が配置される。
内蔵 SRAM の上位アドレスからスタック領域を確保する。

スタック領域の配置アドレス、および スーパーバイザモード以外の各モードのスタックサイズは、utk_config_depend.h で指定することができる。utk_config_depend.h には、各モード用スタックサイズを指定するためのマクロ定義を用意してあるが、μT-Kernel のリファレンスコードでは ユーザーモード(USR) と システムモード(SYS) を使用しない。このため、これらのモード(USR, SYS)に対応するスタックサイズ(USR_STACK_SIZE)の値は 0 に設定してある。
なお、スーパーバイザモードのスタック領域は、割込み(IRQ)スタック領域の下位側に確保され、未使用の RAM 領域を使用する。

2.4 外部 SRAM 領域メモリマップ

外部 SRAM は 2MB の領域が実装されている。外部 SRAM 領域のメモリマップを以下に示す。



NoInit： ゼロ初期化されない BSS セクション

外部 SRAM の下位アドレスから、データセクションと BSS セクションを配置する。

μT-Kernel 管理領域は、μT-Kernel のメモリ管理機能で使用する領域である。
通常は、空いているメモリ領域を全て μT-Kernel 管理領域に割り当るが、設定により変更が可能である。 μT-Kernel 管理領域は、設定ファイルの SYSTEMAREA_TOP と SYSTEMAREA_END で指定された間の領域となる。

2.5 スタック

μT-Kernel では以下の 2 種類のスタックがある。

(1) システムスタック

割込みハンドラ以外で使用するスタックで、タスク毎に 1 本ずつ存在する。
μT-Kernel には保護レベルの概念が無いため、T-Kernel のようにユーザスタックとシステムスタックの使い分けはない。

(2) 例外/割込みスタック

割込みハンドラで使用するスタックで、スーパーバイザーモード(SVC)を除くすべての例外モードのスタックを例外/割込みスタックとする。それぞれのモード毎に独立したスタック領域が割り当てられる。

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

割込みスタックはシステムで共有されるため、使用中にタスクスイッチが起きてはいけない。

本実装では、アボート/未定義命令例外/高速割込み(FIQ)のスタック領域は使用しない。

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

3. 割込みおよび例外

3.1 割込み定義番号

tk_def_int で定義する割込み定義番号(dintno)のうち、0～31 は割込みコントローラ(AIC)が管理する割込み要因に対応し、32～111 番はソフトウェア割込み(SWI)に対応する。この為、SWI 0 ～ SWI 31 は使用できない。

割込みコントローラの割込み要因の一覧を以下に示す。

```
/*
 * Interrupt Controller Sources
 */
#define FIQ        0    /* Fast interrupt */
#define SWIRQ      1    /* Software interrupt */
#define US0IRQ     2    /* USART Channel 0 interrupt */
#define US1IRQ     3    /* USART Channel 1 interrupt */
#define US2IRQ     4    /* USART Channel 2 interrupt */
#define SPIRQ      5    /* SPI interrupt */
#define TC0IRQ     6    /* Timer Channel 0 interrupt */
#define TC1IRQ     7    /* Timer Channel 1 interrupt */
#define TC2IRQ     8    /* Timer Channel 2 interrupt */
#define TC3IRQ     9    /* Timer Channel 3 interrupt */
#define TC4IRQ    10    /* Timer Channel 4 interrupt */
#define TC5IRQ    11    /* Timer Channel 5 interrupt */
#define WDIRQ     12    /* Watchdog interrupt */
#define PIOAIRQ   13    /* Parallel I/O Controller A interrupt */
#define PIOBIRQ   14    /* Parallel I/O Controller B interrupt */
#define AD0IRQ    15    /* Analog-to-digital Converter Channel 0 interrupt */
#define AD1IRQ    16    /* Analog-to-digital Converter Channel 1 interrupt */
#define DA0IRQ    17    /* Digital-to-analog Converter Channel 0 interrupt */
#define DA1IRQ    18    /* Digital-to-analog Converter Channel 1 interrupt */
#define RTCIRQ    19    /* Real-time Clock interrupt */
#define APMCIRQ   20    /* Advanced Power Management Controller interrupt */
                        /*  21  Reserved */
                        /*  22  Reserved */
#define SCLKIRQ   23    /* Slow Clock Interrupt */
#define IRQ5      24    /* External interrupt 5 */
#define IRQ4      25    /* External interrupt 4 */
#define IRQ3      26    /* External interrupt 3 */
#define IRQ2      27    /* External interrupt 2 */
#define IRQ1      28    /* External interrupt 1 */
#define IRQ0      29    /* External interrupt 0 */
#define COMMRX    30    /* RX Debug Communication Channel interrupt */
#define COMMTX    31    /* TX Debug Communication Channel interrupt */
```

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

3.2　ソフトウェア割込みの割当て

ソフトウェア割込みは 36 ～ 39 を使用する。
各ソフトウェア割込みは以下のように使用される。

SWI 36　　μT-Kernel システムコール・拡張 SVC
SWI 37　　tk_ret_int()システムコール
SWI 38　　タスクディスパッチャ
SWI 39　　デバッガサポート機能

3.3　例外・割込みハンドラ

例外が発生した場合、例外ベクタテーブルを参照して各々のハンドラへジャンプするための分岐処理ルーチンが起動される。この時、ip(R12)レジスタにはベクタテーブルアドレスが設定される。このベクタテーブルアドレスによって、発生した例外のベクタ番号を知ることができる。

(ip - knl_intvec) / 4 = ベクタ番号

例外・割込みハンドラへは、割込み禁止状態のままでジャンプする。また、bx 命令によりハンドラへジャンプするので、ベクタテーブルへ設定するハンドラアドレスの最下位ビットが 1 の場合は Thumb モードへ切り替わる。そうでなければ ARM モードのままでジャンプする。

分岐ルーチンでは分岐処理のために、いくつかのレジスタを使用する。これらのレジスタは、下記に示すようにスタックへ保存される。下記に示す以外のレジスタについては、分岐ルーチンでは保存しないので、各ハンドラで必要に応じて保存する必要がある。またハンドラから戻るときには、分岐ルーチンで保存したレジスタを含めて復帰する必要がある。

ハンドラ呼び出し時のスタック

```
        +---------------+
ssp ->  | R3            |
        | SPSR          |
        | R12 = ip      |
        | R14 = lr      |
        +---------------+
```

ハンドラ呼び出し時のレジスタ

ip = ベクタテーブルアドレス
lr, R3 = 不定

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

3.4　例外・割込みハンドラのエントリルーチン

例外/割込み処理ルーチンのコードの例を以下に示す。

```
/*
 * プログラムステータスレジスタ(PSR)
*/
#define PSR_N           0x80000000      /* 条件フラグ    負 */
#define PSR_Z           0x40000000      /*               ゼロ */
#define PSR_C           0x20000000      /*               キャリー */
#define PSR_V           0x10000000      /*               オーバーフロー */

#define PSR_I           0x00000080      /* 割込み(IRQ)禁止 */
#define PSR_F           0x00000040      /* 高速割込み(FIQ)禁止 */
#define PSR_T           0x00000020      /* Thumb モード */

#define PSR_M(n)        ( n )           /* プロセッサモード 0～31 */
#define PSR_USR         PSR_M(16)       /* ユーザーモード */
#define PSR_FIQ         PSR_M(17)       /* 高速割込み(FIQ)モード */
#define PSR_IRQ         PSR_M(18)       /* 割込み(IRQ)モード */
#define PSR_SVC         PSR_M(19)       /* スーパーバイザモード */
#define PSR_ABT         PSR_M(23)       /* アボートモード */
#define PSR_UND         PSR_M(27)       /* 未定義命令モード */
#define PSR_SYS         PSR_M(31)       /* システムモード */

#define N_INTVEC        112             /* ベクターテーブルサイズ */

/*
 * μT-Kernel 用ソフトウェア割込み番号
 */

#define SWI_SVC         36      /* μT-Kernel システムコール・拡張 SVC */
#define SWI_RETINT      37      /* tk_ret_int() システムコール */
#define SWI_DISPATCH    38      /* タスクディスパッチャ */
#define SWI_DEBUG       39      /* デバッガサポート機能 */

/* ------------------------------------------------------------------------ */
/*
 * 例外分岐処理
 */
        .section .vector,"ax"
        .code 32
        .align 0
        .global __reset
__reset:
        b       start               /* 00 : リセット */
        .global undef_vector
undef_vector:
        b       undef_vector        /* 04 : 未定義命令例外 */
        .global swi_vector
swi_vector:
        b       swi_handler         /* 08 : ソフトウェア割込(SWI) */
```

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

```
        .global prefetch_vector
prefetch_vector:
        b       prefetch_vector     /* 0C : プリフェッチアボート */
        .global data_abort_vector
data_abort_vector:
        b       data_abort_vector  /* 10 : データアボート */
        .global reserved_vector
reserved_vector:
        b       reserved_vector    /* 14 : (予約) */
        .global irq_vector
irq_vector:
        ldr pc, [pc, #-0xf20]      /* 18 : 割込(IRQ: AIC_IVR) */
        .global fiq_vector
fiq_vector:
        ldr pc, [pc, #-0xf20]      /* 1C : 高速割込(FIQ: AIC_FVR) */

/*
 * ソフトウェア割込み(SWI)
 */

swi_handler:
        str    lr, [sp, #-4]!
        str    ip, [sp, #-4]!
        mrs    ip, spsr
        str    ip, [sp, #-4]!

        ldr    ip, [lr, #-4]           /* load SWI No. */
        bic    ip, ip, #(0xff << 24)

        ldr    lr, =Csym(knl_intvec)   /* exception vector table */
        add    ip, lr, ip, LSL #2      /* lr := lr + ip*4 = vecaddr */
        ldr    lr, [ip]
        bx     lr

/*
 * 割込み(IRQ)
 */

        .global knl_irq_handler
knl_irq_handler:
        sub    lr, lr, #4
        stmfd  sp!, {lr}              /* sp-> lr_xxx */

#if USE_PROTECT_MODE
        ldr    lr, =AIC_BASE
        str    lr, [lr, #AIC_IVR]
#else
        ldr    lr, =AIC_BASE
        ldr    lr, [lr, #AIC_IVR]
#endif

        stmfd  sp!, {ip}             /* sp-> ip, lr_xxx */
        mrs    ip, spsr
```

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

```
        stmfd   sp!, {ip}               /* sp-> spsr_xxx, ip, lr_xxx */
        stmfd   sp!, {r3}               /* sp-> r3, spsr_xxx, ip, lr_xxx */

        ldr     lr, =(AIC_BASE | AIC_ISR)
        ldr     lr, [lr]                /* lr := IRQ No. */
        ldr     ip, =Csym(knl_intvec) /* exception vector table */
        add     ip, ip, lr, LSL #2      /* ip := &vector[IRQ No.] */
        ldr     r3, [ip]                /* r3 := vector[IRQ No.]  */
        mov     lr, pc
        bx      r3

/* ------------------------------------------------------------------------ */
/*
 * 例外復帰処理 (asm_depend.h)
 */
.macro EXC_RETURN
        .arm
        ldmfd   sp!, {ip}
        msr     spsr_fsxc, ip
        ldmfd   sp!, {ip, pc}^
.endm

/* ------------------------------------------------------------------------ */

/*
 * tk_ret_int() によるハンドラからの戻り
 */
.macro TK_RET_INT_FIQ mode
        .arm
        mov     r3, lr                          // r3 = lr_svc
        msr     cpsr_c, #PSR_I|PSR_F|¥mode      // 元の例外モードへ戻る
        swp     r3, r3, [sp]                    // lr_svc を保存して r3 復帰
        swi     SWI_RETINT
.endm

/* ------------------------------------------------------------------------ */
```

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

4. 初期化および起動処理

4.1 μT-Kernel の起動手順

システムがリセットされるとμT-Kernel が起動する。
μT-Kernel が起動してから、main 関数が呼ばれるまでのμT-Kernel の起動手順を以下に示す。

icrt0.S
(1) スーパーバイザモードに切り換え [start:]
(2) FlashROM 設定 (WaitState:5) [flashrom_init:]
(3) 水晶振動子 設定 (動作クロック数:16MHz) [crystal_init:]
(4) 内蔵 SRAM の先頭に例外ベクタを設定する [setup_ram_vectors:]
(5) FlashROM 設定 (WaitState 数:5) [setup_ram_vectors:]
(6) 外部 SRAM 設定 (WaitState 数:3) [setup_ram_vectors:]
(7) 外部イーサネットコントローラ設定[setup_ram_vectors:]
(8) REMAP 実行[setup_ram_vectors:]
(9) 割込みコントローラのモード設定 [after_remap_start:]
(10)スタックポインタの設定(内蔵 SRAM の最上位ビットから) [init_stacks:]
　　R13_abt : アボートモード
　　R13_und : 未定義命令例外モード
　　R13_fiq : 高速割込み(FIQ)モード
　　R13_irq : 割込み(IRQ)モード
　　R13_svc : スーパーバイザモード
(11)TC0,TC1,TC2 の有効化 [tm_init:]
(12)シリアル 設定(115200 bps, 8bit, non-parity, 1 stop bit) [tm_init:]
(13)データセクションの初期値設定(ROM->RAM)
(14)BSS セクションをゼロクリア
(15)μT-Kernel 管理領域の再計算
(16)main 関数(sysinit_main.c) 呼出し [kernel_start:]

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

4.2　ユーザー初期化プログラム

ユーザー初期化プログラムは、ユーザー定義のシステム起動処理/終了処理を実行するためのルーチンである。ユーザー初期化プログラムは、初期タスクから次の形式で呼び出される。

```
INT userinit( INT flag )

flag    = 0             起動時呼出し
        = -1            終了時呼出し

戻り値  1               usermain()を起動
        それ以外        システム終了
```

システム起動時に flag = 0 で呼出され、システム終了時に flag = -1 で呼出される。終了時の呼出では戻り値は無視される。処理の概略は次のようになる。

```
fin = userinit(0);
if ( fin > 0 ) {
        usermain();
}
userinit(-1);
```

ユーザー初期化プログラムは初期タスクのコンテキストで実行される。タスク優先度は(CFN_MAX_PRI-2)である。

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

5. μT-Kernel 実装定義

5.1 システム状態判定

(1) タスク独立部（割込みハンドラ、タイムイベントハンドラ）

μT-Kernel 内にソフトウェア的なフラグを設けて判定する。

knl_taskindp	= 0	タスク部
knl_taskindp	> 0	タスク独立部

(2) 準タスク部（拡張 SVC ハンドラ）

μT-Kernel 内にソフトウェア的なフラグを設けて判定する。

TCB の sysmode	= 0	タスク部
TCB の sysmode	> 0	準タスク部

5.2 μT-Kernel で使用する例外・割込み

SWI 36	μT-Kernel システムコール・拡張 SVC
SWI 37	tk_ret_int() システムコール
SWI 38	タスクディスパッチャ
SWI 39	デバッガサポート機能
TC0IRQ	プログラマブルタイマー#0(TC0)

5.3 システムコール／拡張 SVC のインターフェース

呼出し側は、C 言語の関数の呼出形式で、インターフェースライブラリを呼出す方法と直接呼出す方法を選択できる（カーネルの構築時にオプションとして選択）。

アセンブラから呼び出す場合も、C 言語と同様に関数形式によりインターフェースライブラリを経由して呼び出すこととするが、下記のインターフェースライブラリ相当のことを行い、直接 SWI 命令で呼び出してもよい。その場合も、C 言語の規則にしたがってレジスタの保存を行う必要がある。

インターフェースライブラリの基本的な処理は以下のようになる。

・ R12 レジスタに機能コードを設定して SWI_SVC (SWI 36) により呼び出す。
機能コードが負の値ならシステムコール、0 または正の値なら拡張 SVC となる。
ただし、デバッガサポート機能のサービスコールは SWI_DEBUG (SWI 39) を使用する。

・ システムコールおよび拡張 SVC は、SWI 命令により以下のように呼び出される。

```
stmfd   sp!, {lr}
swi     36
ldmfd   sp!, {lr}
```

SWI 命令を使用できるのは、原則としてスーパーバイザモード(SVC)でのみである。

他のモードからシステムコールを呼び出す場合は、SVC モードに切り替えてから呼び出すか、R14_svc を保存してから呼び出す(システムコールから戻ったあとに R14_svc を元の値に復帰する)必要がある。

尚、USR モードおよび SYS モードには切り替えてはいけない。

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

・ レジスタの保存規則は C 言語レジスタ保存規則に従い以下の通りとなる。

```
R0～R3, R12=ip  テンポラリレジスタ
R4～R10         パーマネントレジスタ
R11 = fp        フレームポインタ
R13 = sp        スタックポインタ
R14 = lr        リンクレジスタ(関数戻りアドレス)
R15 = pc        プログラムカウンタ

引数    R0～R3
戻り値  R0
```

テンポラリレジスタが関数の呼び出しによって破壊される。それ以外のレジスタは保存される。

(1) システムコールのインターフェース

システムコールは、第4引数まではレジスタに設定し、第5引数以降はスタックに積んでSWI_SVC (SWI 36)により呼び出す。レジスタは以下のように使用される。

```
R12=ip  機能コード (<0)
R0      第 1 引数
R1      第 2 引数
R2      第 3 引数
R3      第 4 引数
R4      第 5 引数以降が格納されたスタックへのポインタ

R0      戻り値
```

システムコールのインターフェースの実装例を以下に示す。

```
ER tk_xxx_yyy(p1, p2, p3, p4, p5)
```

引数は 0～5 個の整数またはポインタで、C 言語の関数の引数渡しと同じ形式。

```
// r0 = p1
// r1 = p2
// r2 = p3
// r3 = p4
//         +---------------+
//   sp -> | p5            |
//         +---------------+
Csym(tk_xxx_yyy):
        stmfd   sp!, {r4}       // r4 保存
        add     r4, sp, #4      // r4 = スタック上のパラメータの位置
        stmfd   sp!, {lr}       // lr 保存
        ldr     ip, =機能コード
#if USE_TRAP
        swi     SWI_SVC
#else
        bl      Csym(knl_call_entry)
#endif
```

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

```
        ldmfd   sp!, {lr}       // lr 復帰
        ldmfd   sp!, {r4}       // r4 復帰
        bx      lr
```

(2) 拡張 SVC のインターフェースライブラリ

引数は全てパケット化し、パケットの先頭アドレスを R0 レジスタに設定して SWI_SVC (SWI 36)により呼び出す。パケットは通常スタックに作成するが、他の場所でもかまわない。引数はパケット化するため、数や型に制限はない。レジスタは以下のように使用する。

```
R12=ip  機能コード（≧0）
R0      引数パケット

R0      戻り値
```

拡張 SVC のインターフェースの実装例を以下に示す。

```
INT zxxx_yyy( .... )
```

引数は呼出側でパケット化し、パケットの先頭アドレスを R0 レジスタに設定する。

```
Csym(zxxx_yyy):
        stmfd   sp!, {r0-r3}    // レジスタ上の引数をスタックに積みパケット化
        mov     r0, sp          // R0 = 引数パケットのアドレス
        stmfd   sp!, {lr}       // lr 保存
        ldr     ip, =機能コード
        swi     SWI_SVC
        ldmfd   sp!, {lr}       // lr 復帰
        add     sp, sp, #4*4    // スタックに積んだ引数を捨てる
        bx      lr
```

(3) デバッガサポート機能 システムコールのインターフェース

デバッガサポート機能 システムコールは、基本的には他の μT-Kernel のサービスコールと同じだが、SWI_DEBUG(SWI 39) を使用する。

システムコールのインターフェースの実装例を以下に示す。

```
ER td_xxx_yyy(p1, p2, p3, p4)

//      r0 =    p1
//      r1 =    p2
//      r2 =    p3
//      r3 =    p4
Csym(td_xxx_yyy):
        stmfd   sp!, {lr}    // lr 保存
        ldr     ip, =機能コード
#if USE_TRAP
        swi     SWI_DEBUG
#else
        bl      Csym(knl_call_dbgspt)
#endif
        ldmfd   sp!, {lr}    // lr 復帰
```

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

```
        bx      lr
#endif
```

5.4 割込みハンドラ

割込みハンドラのハードウェアに依存した実装定義を以下に示す。

・ 割込みハンドラ定義情報 : T_DINT

```
typedef struct t_dint {
        ATR     intatr;         /* 割込みハンドラ属性 */
        FP      inthdr;         /* 割込みハンドラアドレス */
} T_DINT;
```

・ 割込み定義番号 : dintno

dintno は 0～111 の範囲で指定可能。

割込みハンドラは、属性（TA_HLNG、TA_ASM）により実装が異なる。

(1) TA_HLNG 属性の割込みハンドラ

割込みハンドラ属性が TA_HLNG の場合、例外/割込みベクターテーブルにはμT-Kernel 内の高級言語対応ルーチンのアドレスが設定され、高級言語対応ルーチンから設定された割込みハンドラが呼び出される。

例外・割込み処理の開始時のプロセッサモードは、発生した例外・割込みの種類に応じて、以下のいずれかとなる。

SVC:スーパーバイザーモード
ABT:アボートモード
UND:未定義命令例外モード
IRQ:割込みモード
FIQ:高速割込みモード

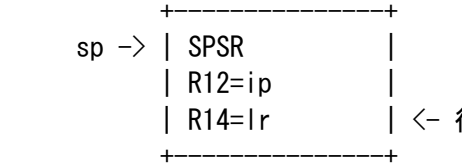
しかし、TA_HLNG 指定の場合、高級言語対応ルーチンによって SVC モードへ自動的に変更される。したがって、どのタイプの例外・割込みでも、ユーザーの割込みハンドラに入った時点では SVC モードとなっている。

割込みハンドラの定義は以下となる。

```
void inthdr( UINT dintno, VP sp )
```

dintno　発生した例外/割込みベクタ番号
デフォルトハンドラの場合、デフォルトハンドラのベクタ番号ではなく、
発生した例外/割込みのベクタ番号となる。

sp　スタックに保存された以下の情報へのポインタ



復帰アドレス

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

```
FIQ の時        割込みからの復帰アドレス
IRQ の時        割込みからの復帰アドレス
SVC の時        SWI 命令の次の命令への復帰アドレス
ABT の時        アボートした命令への復帰アドレス
UND の時        未定義命令の次の命令への復帰アドレス
```

割込みハンドラに入ったときの CPU の状態は以下のようになる。

```
FIQ の時
CPSR.F = 1      高速割込み禁止
CPSR.I = 1      割込み禁止
CPSR.M = 19     SVC:スーパーバイザーモード

FIQ 以外の時
CPSR.F = ?      割込み・例外発生時の状態のまま
CPSR.I = 1      割込み禁止
CPSR.M = 19     SVC:スーパーバイザーモード
```

多重割込みは禁止されている。CPSR.I または F を 0 とすることで、多重割込みを許可することはできるが、その場合割込みコントローラの設定を適切に変更する必要がある。割込みコントローラの設定を変更した場合は、割込みハンドラから戻る前に割込みコントローラの設定を元に戻す必要がある。

μT-Kernel 内の高級言語対応ルーチンでは割込みコントローラに対する処理は何も行われない。割込みのクリア等は、割込みハンドラが処理しなければならない。

割込みハンドラからの復帰は、関数からの return で行う。

(2) TA_ASM 属性の割込みハンドラ

割込みハンドラ属性が TA_ASM の場合、例外/割込みベクターテーブルへ直接割込みハンドラのアドレスを設定する。

高級言語対応ルーチンを経由しないため、タスク独立部判定のためのフラグが更新されない。そのため、タスク独立部として判定されないので、以下の注意が必要となる。

割込みを許可(CPSR.I=0 かつ F=0)すると、タスクディスパッチが発生する可能性がある。SWI 命令による例外の場合を除き、タスクディスパッチが起きるとその後の動作が異常になる。したがって、ハンドラ内で割込みを許可する場合には、必要に応じてタスク独立部判定フラグを設定しなければならない。

タスク独立部判定フラグの設定は、システム共有情報内の knl_taskindp フラグを操作することで行う。この操作は、割込み禁止状態(CPSR.I=1 かつ F=1)で行わなければならない。また、割込みハンドラを終了する前にフラグを必ず戻さなければならない。

```
knl_taskindp++;         /* タスク独立部に入る */
knl_taskindp--;         /* タスク独立部を出る */
```

多重割込みの場合などもあるため、knl_taskindp は必ずインクリメント／デクリメントによって設定する必要がある。knl_taskindp = 0 のような設定を行ってはいけない。

割込みハンドラからの復帰は、tk_ret_int() システムコールを使用するか、EXC_RETURN マクロを使用する。マクロを使用したときは、遅延ディスパッチは起こらない。なお、マクロを使用する代わりに、独自にマクロ相当の処理を行うこともできる。

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

tk_ret_int() システムコールは、他のシステムコールとは異なる専用の SWI 37 により呼び出す。他のシステムコールのような関数形式の呼び出しはできない。

```
SWI  37       // tk_ret_int() 呼び出し（戻らない）
```

tk_ret_int() を呼び出すマクロ TK_RET_INT_FIQ が用意されている。

tk_ret_int() を呼び出す時点で、スタックを以下のような状態にしておく必要がある。また、スタックに格納されているレジスタ以外(R0～R11)は、すべて復帰しておく必要がある。

```
          +---------------+
sp -> | R14_svc       |
      | SPSR          |
      | R12=ip        |
      | R14=lr        | <- 復帰アドレス
      +---------------+
```

sp は、発生した例外・割込みのモードのスタックポインタ(R13)である。例外モードは、発生した例外・割込みのモード(ハンドラに入ったときのモード)に戻しておく必要がある。

スタック上の R14_svc には、ハンドラに入ったときの SVC モードの R14 レジスタの値を保存する。

割込みハンドラ以外から tk_ret_int() を呼び出した場合の動作は保証されない。

5.5　タイムイベントハンドラ

ハンドラ属性に TA_ASM 属性を指定した場合も、TA_HLNG 属性の場合と同様に高級言語対応ルーチンを経由して呼び出される。したがって、TA_ASM 属性の場合もハンドラへわたされるパラメータ(exinf)は C 言語の規則にしたがって R0 レジスタに渡される。また、C 言語の規則に従ってレジスタを保存しなければならない。

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

5.6　タスクの実装依存定義

タスクのハードウェアに依存した実装定義を以下に示す。

(1) タスク生成情報　T_CTSK

独自に追加した情報はなし。

```
typedef struct t_ctsk {
        VP      exinf;          /* 拡張情報 */
        ATR     tskatr;         /* タスク属性 */
        FP      task;           /* タスク起動アドレス */
        PRI     itskpri;        /* タスク起動時優先度 */
        W       stksz;          /* スタックサイズ(バイト) */
        UB      dsname[8];      /* DS オブジェクト名称 */
        VP      bufptr;         /* ユーザーバッファポインタ */
} T_CTSK;
```

(2) タスク属性

実装独自属性はなし。

```
tskatr := (TA_ASM // TA_HLNG)
        | [TA_USERBUF] | [TA_DSNAME]
        | (TA_RNG0 // TA_RNG1 // TA_RNG2 // TA_RNG3)
```

(3) タスクの形式

タスクは次の形式で、TA_HLNG, TA_ASM のどちらを指定しても違いはない。

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

タスク起動時のレジスタの状態は下記のようになる。

```
CPSR.F = 0    高速割込許可
CPSR.I = 0    割込許可
CPSR.T = 0    ARM モード タスク起動アドレスの最下位ビットが 0 の場合
         1    Thumb モード タスク起動アドレスの最下位ビットが 1 の場合
CPSR.M = 19   SVC:スーパーバイザモード （TA_RNG0～3 の指定に影響されない）

R0 = stacd    タスク起動パラメータ
R1 = exinf    タスク拡張情報
R13(sp)       スタックポインタ
```

その他のレジスタは不定である。
タスクの終了は、tk_ext_tsk() または tk_exd_tsk() を用いなければならない。単に return してもタスクの終了とはならない。return した場合の動作は保証されない。

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

5.7　タスクレジスタの設定/参照

```
ER tk_set_reg( ID tskid, T_REGS *pk_regs, T_EIT *pk_eit, T_CREGS *pk_cregs )
ER tk_get_reg( ID tskid, T_REGS *pk_regs, T_EIT *pk_eit, T_CREGS *pk_cregs )
```

タスクレジスタの取得/設定（tk_get_reg/tk_set_reg）の対象となるレジスタは以下のように定義される。

(1) 汎用レジスタ T_REGS

```
typedef struct t_regs {
        VW r[13];       /* 汎用レジスタ R0～R12 */
        VP lr;          /* リンクレジスタ R14 */
} T_REGS;
```

DORMANT 状態のタスクに対してレジスタの設定を行ったとき、R0, R1 は tk_sta_tsk() によってタスク起動パラメータ／拡張情報が設定されるため、tk_set_reg() で設定した値は捨てられることになる。

(2) 例外時に保存されるレジスタ T_EIT

```
typedef struct t_eit {
        VP pc;          /* プログラムカウンタ R15 */
        UW cpsr;        /* プログラムステータスレジスタ */
        UW taskmode;    /* タスクモードフラグ */
} T_EIT;
```

CPSR は、フラグフィールド(ビット 31～24)以外は変更できない。他のフィールド(ビット 23～0)への設定は無視される。
taskmode は、システム共有情報にあるタスクモードフラグと同じ。メモリのアクセス権情報を保持するレジスタとして扱われる。

(3) 制御レジスタ

```
typedef struct t_cregs {
        VP ssp;         /* システムスタックポインタ R13_svc */
        VP usp;         /* ユーザースタックポインタ R13_usr */
} T_CREGS;
```

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

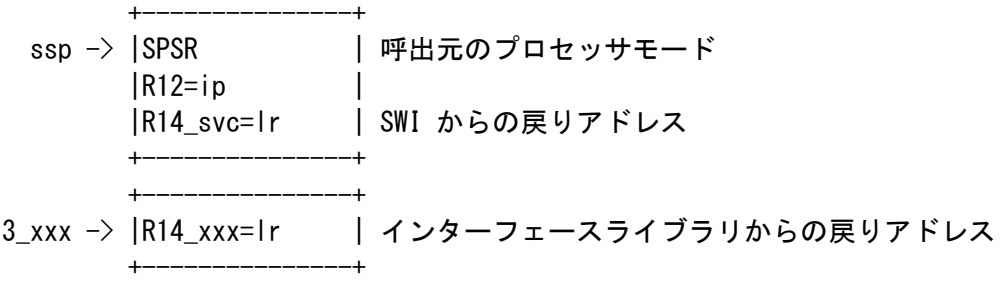
5.8　システムコール・拡張 SVC 呼び出し元情報

システムコール/拡張 SVC フックルーチンの実装依存定義を以下に示す。

フックルーチン定義情報　TD_CALINF

```
typedef struct td_calinf {
        VP      ssp;    /* システムスタックポインタ */
        VP      r11;    /* 呼出時のフレームポインタ */
} TD_CALINF;
```

フックルーチンに入ったときのシステムスタックは以下の状態になっている。



システムコール・拡張 SVC からの戻り番地は R14_svc となる。しかし、通常はインターフェースライブラリを利用して呼び出すため、R14_svc はインターフェースライブラリの番地を指している。

インターフェースライブラリからの戻り番地は R14_xxx となる。R13_xxx は呼出元のプロセッサモードにしたがったスタックポインタであり、呼出元のプロセッサモードは SPSR で知ることができる。呼出元のプロセッサモードが SVC の場合、ssp + 12 が R13_xxx に相当する。ただし、標準のインターフェースライブラリを使用していない場合は、この限りではない。

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

5.9　割込みコントローラ制御

割込みコントローラ制御は、ハードウェアへの依存が強いため、μT-Kernel 仕様書では規定されていないが、以下のとおり実装する。

(1) 割込み定義番号

割込み定義番号のうち 0～31 は、割込みコントローラ(AIC)の割込み要因と同じ番号を使用する。

(2) 割込みモード設定

```
void SetIntMode( INTVEC intvec, UINT mode )
```

intvec に対応した 割込コントローラの AIC_SMRn レジスタに mode で指定したモードに設定する。

```
mode := (IM_LEVEL // IM_EDGE) | (IM_HI // IM_LOW)

#define IM_LEVEL        0  /* level trigger */
#define IM_EDGE         1  /* edge  trigger */
#define IM_HI           2  /* high level trigger/positive edge trigger */
#define IM_LOW          0  /* low  level trigger/negative edge trigger */
```

(3) 割込許可

```
void EnableInt( INTVEC intvec )
```

intvec で指定した割込を許可する。

(4) 割込禁止

```
void DisableInt( INTVEC intvec )
```

intvec で指定した割込を禁止する。

(5) 割込要求のクリア

```
void ClearInt( INTVEC intvec )
```

intvec の割込要求をクリアする。
エッジトリガーの場合のみ有効。
エッジトリガーの場合、割込ハンドラで割込をクリアする必要がある。

(6) 割込要求の有無の確認

```
BOOL CheckInt( INTVEC intvec )
```

intvec の割込要求があるか調べる。
割込要求があれば TRUE (0 以外) を返す。

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

6. システムコンフィグレーションデータ

utk_config_depend.h では、μT-Kernel のシステム構成情報や各種資源数、各種制限値などの設定が記述する。なお、各項目の指定可能範囲の最大値は、論理的な最大値であり、実際にはメモリの使用量により制限を受ける。

6.1 utk_config_depend.h の設定値

```
/*
 *      utk_config_depend.h (at91)
 *      System Configuration Definition
 */

/* RAMINFO */
#define SYSTEMAREA_TOP          0x20000000
#define SYSTEMAREA_END          0x20200000
```

※ μT-Kernel のメモリ管理機能により動的に管理される領域
外部 SRAM の最下位アドレスと最上位アドレスを指定

```
/* User definition */
#define RI_USERAREA_TOP         0x20100000
```

※ 本設定は使用しない

```
#define RI_USERINIT             NULL
```

※ ユーザー初期化/完了プログラム

```
/* SYSCONF */
#define CFN_TIMER_PERIOD        10
```

※ システムタイマの割込み周期(ミリ秒)。各種の時間指定の最小分解能(精度)となる。

```
#define CFN_MAX_TSKID           32
#define CFN_MAX_SEMID           16
#define CFN_MAX_FLGID           16
#define CFN_MAX_MBXID           8
#define CFN_MAX_MTXID           2
#define CFN_MAX_MBFID           8
#define CFN_MAX_PORID           4
#define CFN_MAX_MPLID           2
#define CFN_MAX_MPFID           8
#define CFN_MAX_CYCID           4
#define CFN_MAX_ALMID           8
#define CFN_MAX_SSYID           4
```

※ μT-Kernel の各オブジェクトの最大数。
カーネルが使用するオブジェクトの数も考慮して指定する必要がある。

```
#define CFN_MAX_REGDEV          (8)
```

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

※ tk_def_dev() で登録可能な最大デバイス数。物理デバイスの最大数となる。

```
#define CFN_MAX_OPNDEV          (16)
```

※ tk_opn_dev() でオープン可能な最大数。デバイスディスクリプタの最大数となる。

```
#define CFN_MAX_REQDEV          (16)
```

※ tk_rea_dev()、tk_wri_dev()、tk_srea_dev()、tk_swri_dev() で要求可能な最大数。
リクエスト ID の最大数となる。

```
#define CFN_VER_MAKER           0
#define CFN_VER_PRID            0
#define CFN_VER_SPVER           0x6101
#define CFN_VER_PRVER           0x0101
#define CFN_VER_PRNO1           0
#define CFN_VER_PRNO2           0
#define CFN_VER_PRNO3           0
#define CFN_VER_PRNO4           0
```

※ バージョン情報（tk_ref_ver）

```
#define CFN_REALMEMEND          ((VP)0x20200000)
```

※ μT-Kernel 管理領域で利用する RAM の最上位アドレス

```
/*
 * Stack size for each mode
 */
#define IRQ_STACK_SIZE          2048
#define FIQ_STACK_SIZE          16      /* 4 words */
#define ABT_STACK_SIZE          4       /* 1 word */
#define UND_STACK_SIZE          4       /* 1 word */
#define USR_STACK_SIZE          0       /* not used */
```

※ 各例外モード用のスタックサイズ(バイト数)
※ USR_STACK_SIZE では、ユーザモードとシステムモードに共通のスタックのサイズを指定する。

```
#define EXCEPTION_STACK_TOP     INTERNAL_RAM_END
```

※ スタック領域の初期位置
これらの例外モード用スタックは、通常はハンドラの入り口と出口においてのみ使用するため、
大きな容量は必要としない

```
/*
 * Use zero-clear bss section
 */
#define USE_NOINIT              (1)
```

※ 1 :初期値を持たない静的変数(BSS 配置)のうち、初期化が必要のない変数 は
カーネル初期化処理内で ゼロクリアしない。
ゼロクリアの処理が削減される為、カーネル起動時間が短縮される。
0 :全ての 初期値を持たない静的変数(BSS 配置) をゼロクリアする。

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

```
/*
 * Use dynamic memory allocation
 */
#define USE_IMALLOC             (1)
```

※ 1 : カーネル内部の 動的メモリ割当て機能を使用する。
0 : カーネル内部の 動的メモリ割当て機能を使用しない。
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、
TA_USERBUF を指定して、アプリケーションがバッファを指定しなければならない。

```
/*
 * Use program trace function (in debugger support)
 */
#define USE_HOOK_TRACE          (0)
```

※ 1 : デバッガサポート機能のフック機構を使用する。
但し、USE_DBGSPT が 0 になっている場合は、フック機構は使えない。
0 : デバッガサポート機能のフック機構を使用しない。

```
/*
 * Use clean-up sequence
 */
#define USE_CLEANUP             (1)
```

※ 1 : アプリケーション終了後に、カーネルのクリーンアップ処理を行う。
0 : アプリケーション終了後に、カーネルのクリーンアップ処理を行わない。
usermain 関数から戻らないシステムは、本フラグをオフにすることで ROM 消費量が減る。

```
/*
 * Use high level programming language support routine
 */
#define USE_HLL_INTHDR          (1)
```

※ 1 : 割込みで高級言語対応ルーチンを使用する。
0 : 割込みで高級言語対応ルーチンを使用しない。
ジャンプテーブルなどが外れ、ROM/RAM 消費量が減る。

```
/* ------------------------------------------------------------------------*/
```

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.

6.2 sysinfo_depend.h の設定値

```
#define N_INTVEC                112
```

※ 割込み定義番号の上限値

6.3 makerules

以下の引数を指定して make を行うことによってモード選択を行う。

・mode （ コンパイルモード ）

```
指定なし : リリースモード
debug    : デバックモード
           例: $ make mode=debug
```

・trap（トラップモード ）

```
指定なし  : トラップ未使用
on        : システムコール、ディスパッチ時にトラップ実行
            例: $ make mode=debug trap=on
```

Copyright © 2007-2013 by T-Engine Forum. All rights reserved.