



---

# **μC3/Standard ユーザーズガイド**

## **プロセッサ依存部 Cortex-R4 編**

---

Rev. 1. 0

2014. 10. 01

## はじめに

---

本書は $\mu$  C3/Standard の Cortex-R4 に依存する事項を説明します。「 $\mu$  C3/Standard ユーザーズガイド」とあわせてお読みください。

---

TRON は"The Real-time Operation system Nucleus"の略称です。

$\mu$  ITRON は"Micro Industrial TRON"の略称です。

$\mu$  ITRON4.0 仕様の仕様書はトロン協会のホームページ (<http://www.assoc.tron.org/>) から入手することができます。

$\mu$  C3 はイー・フォース株式会社の登録商標です。

本書で記載されている内容は予告無く変更する場合があります。

---

## 改訂記録

Rev.	発行日	改訂内容	
		ページ	内容
1.0	2014. 10. 01	—	第 1 版

## 目次

---

はじめに.....	2
目次.....	4
第1章 ファイル構成.....	6
1. 1 フォルダ構成.....	6
1. 2 カーネルライブラリ.....	7
第2章 プロセッサに依存する状態.....	8
2. 1 割込みの種類.....	8
2. 1. 1 割込みマスク.....	8
2. 2 CPU ロック状態.....	8
2. 3 ディスパッチの保留状態.....	8
2. 4 データ型.....	9
2. 5 割込みの禁止／許可.....	9
2. 6 各コンテキストの実行時のモード.....	9
第3章 カーネルのコンフィグレーション.....	10
3. 1 システムスタック.....	10
3. 1. 1 IAR Embedded Workbench 使用時のシステムスタック定義方法.....	11
3. 1. 2 ARM C/C++ Compiler 使用時のシステムスタック定義方法.....	11
3. 2 各モードのスタック.....	12
3. 2. 1 IAR Embedded Workbench 使用時の各モードのスタック定義方法.....	12
3. 2. 2 ARM C/C++ Compiler 使用時の各モードのスタック定義方法.....	12
第4章 システムコール.....	14
4. 1 CPU 例外ハンドラ.....	14
4. 1. 1 CPU 例外ハンドラの登録準備.....	14
4. 1. 2 SVC 例外の CPU 例外ハンドラ.....	15
4. 1. 3 その他の CPU 例外ハンドラ.....	15
4. 2 割込みハンドラと割込みサービスルーチン.....	16
第5章 CPU 依存ドライバ.....	17
5. 1 標準 MPU ドライバ DDR_CORTEXR_MPU. xxx.....	17
5. 2 Renesas RZ/T1 のドライバ.....	23
5. 2. 1 I/O アドレスと割込み番号の定義.....	23
5. 2. 2 割込みドライバ DDR_RZT1_VIC. c.....	23
5. 2. 3 周期タイマドライバ DDR_RZT1_CMT0. c.....	26
5. 2. 4 クロック発生器ドライバ DDR_RZT1_CPG. c.....	27
5. 2. 5 標準 COM ドライバ DDR_RZT1_SCIF. c.....	30

第6章	プロセッサに依存した注意事項 .....	32
6. 1	浮動小数点演算の使用 .....	32
6. 1. 1	FPU イネーブラの登録 .....	32

## 第 1 章 ファイル構成

---

### 1. 1 フォルダ構成

本製品のフォルダ構成は次のようになります。

uC3

└─Standard	
└─Document .....	ユーザズガイド等のドキュメント類
└─Driver	
└─inc .....	ドライバのインクルードファイ
└─src.....	ドライバのソースファイル
└─Kernel	
└─inc .....	共通インクルードファイル
└─CortexR .....	Cortex-R 依存のインクルードファイル
└─lib.....	カーネルライブラリ
└─CortexR .....	カーネルライブラ生成用プロジェクトファイル
└─src .....	共通ソースファイル
└─CortexR .....	Cortex-R 依存のソースファイル
└─Sample .....	評価ボードのサンプルプログラム

本カーネルの使用時に必要なインクルードパス<sup>1</sup>は次の例のようになります。

```
C:\uC3\Standard\Kernel\inc
C:\uC3\Standard\Kernel\inc\CortexR
C:\uC3\Standard\Driver\inc
```

---

<sup>1</sup> パッケージを “C:\¥” にインストールした例になります。

## 1. 2 カーネルライブラリ

カーネルライブラリは ARM ステートと Thumb ステート、VFP<sup>2</sup>の有無などの組み合わせで次の種類を用意しています。

【IAR Embedded Workbench 版】

ARM/Thumb	エンディアン	VFP	ライブラリ名
ARM	Little	無	uC3cortexrl.a
Thumb	Little	無	uC3cortexrtl.a
ARM	Little	VFPv3_D16	uC3cortexrfl.a
Thumb	Little	VFPv3_D16	uC3cortexrftl.a

【ARM C/C++ Compiler 版】

ARM/Thumb	エンディアン	VFP	ライブラリ名
ARM	Little	無	uC3cortexrl.l
Thumb	Little	無	uC3cortexrtl.l
ARM	Little	VFPv3_D16	uC3cortexrfl.l
Thumb	Little	VFPv3_D16	uC3cortexrftl.l

Thumb ステートのカーネルライブラリは ARM ステートの使用を最小限にとどめたカーネルでコンパクトにできています。

<sup>2</sup> VFP についての詳細は ARM 「Cortex-R4 および Cortex-R4F テクニカルリファレンスマニュアル」を参照してください。

## 第2章 プロセッサに依存する状態

---

### 2. 1 割込みの種類

ARM コアでは IRQ と FIQ の 2 系統の割込みを持っています。μ C3/Standard は IRQ 割込みのみを管理し、FIQ 割込みは管理していません。また、FIQ 割込みは常に許可状態を前提としています。

#### 2. 1. 1 割込みマスク

割込みをマスクするために、ステータスレジスタの IRQ 割込みビットを操作しています。chg\_ims はマスクする割込みレベルを指定し、get\_ims はマスクしている割込みレベルを返します。割込みレベル 0 は IRQ 割込み許可で、割込みレベル 1 は IRQ 割込み禁止です。

### 2. 2 CPU ロック状態

μ C3/Standard の ARM 版では CPU のロック状態の関数でも、次のようにステータスレジスタの IRQ 割込みビットを操作しています。

- loc\_cpu (CPU ロック状態) : IRQ 割込み禁止
- unl\_cpu (CPU ロックの解除状態) : IRQ 割込み許可

loc\_cpu()と chg\_ims(1)、unl\_cpu()と chg\_ims(0)は同じ機能になります。また、割込みハンドラ（割込みサービスルーチンを含む）の開始直後の状態は、割込みコントローラのドライバにより異なります。特に明記していない場合は CPU ロック解除状態で開始されます。

### 2. 3 ディスパッチの保留状態

非タスクコンテキストでは常にディスパッチ保留状態です。一方、タスクコンテキストでは、以下のいずれかの条件に当てはまればディスパッチ保留状態です。

- CPU ロック状態
- ディスパッチ禁止状態



## 2. 4 データ型

ARM 依存の  $\mu$  ITRON4.0 仕様で規定しているデータ型は次のようになります。

INT	符号付き 32 ビット整数
UINT	符号無し 32 ビット整数
VP_INT	データタイプが定まらないものへのポインタまたは符号付き 32 ビット整数
FLGPTN	符号無し 32 ビット整数

## 2. 5 割込みの禁止／許可

各割込み要因に対する割込み禁止／許可を行うシステムコール (dis\_int, ena\_int) の有無は割込みコントローラのドライバにより異なります。

## 2. 6 各コンテキストの実行時のモード

カーネルは非タスクコンテキストをシステムモードで実行します。タスクコンテキストはデフォルトではシステムモードで実行しますが、タスク生成時のタスク属性に TA\_USR を指定するとユーザモードで実行します。ただし、ユーザモードで実行してもカーネルのシステム領域を保護する機能はありません。

## 第3章 カーネルのコンフィグレーション

---

### 3. 1 システムスタック

$\mu$  C3/Standard の非タスクコンテキストはすべてシステムスタックで実行されます。よって、システムスタックのサイズは一番多くスタック領域を消費するタイムイベントハンドラのスタックサイズと、割込み処理（割込みハンドラや割込みサービスルーチン）で消費するスタックサイズを加算した値が目安になります。また、多重割込みが想定される場合には、それらのスタックサイズを加算する必要があります。

### 3. 1. 1 IAR Embedded Workbench 使用時のシステムスタック定義方法

EWARM の場合は EW を起動し、"Options"を開き、"Linker"のタブの"Config"にある "Linker configuration file"の"Edit"ボタンをクリックし、"Linker configuration file editor"を開きます。ここで"Stack/Heap Sizes"のタブの"CSTACK"の値がシステムスタックのサイズになります。

### 3. 1. 2 ARM C/C++ Compiler 使用時のシステムスタック定義方法

ARMC の場合では、スキヤッタファイル(.scat)の中で STACK セクションの定義と共に、領域の確保を行います。

### 3. 2 各モードのスタック

μ C3/Standard ARM 版では、ユーザ(システム)モード以外に割り込み(IRQ)モードとスーパーバイザ(SVC)モードを使用するため、スタック領域の確保が必要です。その他のモードもデバッグのために、スタック領域を確保することをお勧めします。

スーパーバイザモードのスタックサイズは最少で 24 バイトを、さらに、ユーザ定義の例外ハンドラを用いる場合はそこで使用するスタックサイズを加算したサイズにします。割り込み(IRQ)モードのスタックサイズは単一の割り込み発生で 64 バイトを使用します。割り込みがネストする場合は“ネスト数×64 バイト”を加算したサイズにします。

#### 3. 2. 1 IAR Embedded Workbench 使用時の各モードのスタック定義方法

システムスタックの定義方法と同様に"Linker configuration file editor"を開き、"Stack/Heap Sizes"のタブの次の数値を設定します。

- ・ SVC\_STACK : スーパーバイザモードのスタックサイズ
- ・ IRQ\_STACK : 割り込み(IRQ)モードのスタックサイズ
- ・ FIQ\_STACK : 高速割り込み(FIQ)モードのスタックサイズ
- ・ UND\_STACK : 未定義(UND)モードのスタックサイズ
- ・ ABT\_STACK : アボート(ABT)モードのスタックサイズ

#### 3. 2. 2 ARM C/C++ Compiler 使用時の各モードのスタック定義方法

システムスタックの定義方法と同様に、スキュットファイルの中で領域の確保を行います。セクション名に制限はありませんが、サンプルのスタートアップファイルでは、スーパーバイザモードは"SVC\_STACK"、割り込み(IRQ)モードは"IRQ\_STACK"、高速割り込み(FIQ)モードは"FIQ\_STACK"、未定義(UND)モードは"UND\_STACK"、アボート(ABT)モードは"ABT\_STACK"のセクション名を使っています。

例)

```
LR 0x00100000 0x40000000
{
    RAM_EXEC 0x00100000 0x01000000
    {
        prst.o (.intvec, +FIRST); Initialization code
        * (+R0) ;
    }
    RAM_DATA 0x00300000
    {
        * (+RW) ;
    }
    RAM_BSS +0
    {
        * (+ZI) ;
    }
}
```

```

VINFTBL +0x0
{
    * (VINFTBL)                ;
}
VECTTBL +0x0
{
    * (VECTTBL)                ;
}
SYS_DATA 0x00400000
{
    * (STKMEM)                 ;
    * (MPLMEM)                 ;
    * (SYSMEM)                 ;
}

FIQ_STACK 0x009F8000 EMPTY 0x00000100
{
}
UND_STACK +0x0      EMPTY 0x00000100
{
}
ABT_STACK +0x0      EMPTY 0x00000100
{
}
SVC_STACK +0x0      EMPTY 0x00000100
{
}
IRQ_STACK +0x0      EMPTY 0x00000400
{
}
STACK +0x0          EMPTY 0x00003700
{
}
SYS_TBL +0x0
{
    * (SYS)                   ;
}
}

```

## 第4章 システムコール

### 4. 1 CPU 例外ハンドラ

サービスコールの `def_exc` を使用して、SVC 例外、未定義命令例外、プリフェッチアポート例外、データアポート例外を CPU 例外ハンドラとして定義することができます。ただし、システムコールの呼び出しができないなど  $\mu$ ITRON4.0 仕様の CPU 例外ハンドラの機能は満たしていません。

#### 4. 1. 1 CPU 例外ハンドラの登録準備

例外をハンドリングするために、例外ベクタは次のように決められた書式で記述します。SVC 例外と IRQ 例外は、必ず `_kernel_svchdr` と `_kernel_inthdr` に分岐させます。FIQ 例外はカーネル管理外の割込みとして任意の関数を指定できます。

未定義命令例外、プリフェッチアポート例外、データアポート例外の CPU 例外ハンドラを、システムコールの `def_exc` で登録しない場合は、任意の関数を定義しても構いません。

`_PRST`

```
ldr    pc, _ResetHandler
ldr    pc, _UndHandler
dr     pc, _SwiHandler
ldr    pc, _PreHandler
ldr    pc, _AbtHandler
nop
ldr    pc, _IrqHandler
ldr    pc, _FiqHandler
```

`LTORG`

`DATA`

<code>_ResetHandler</code>	DCD	<code>Reset_Handler</code>
<code>_UndHandler</code>	DCD	<code>_kernel_undhdr</code>
<code>_SwiHandler</code>	DCD	<code>_kernel_svchdr</code>
<code>_PreHandler</code>	DCD	<code>_kernel_prahdr</code>
<code>_AbtHandler</code>	DCD	<code>_kernel_dtahdr</code>
	DCD	0
<code>_IrqHandler</code>	DCD	<code>_kernel_inthdr</code>
<code>_FiqHandler</code>	DCD	<code>FiqHandler</code>

#### 4. 1. 2 SVC 例外の CPU 例外ハンドラ

SVC 例外の CPU 例外ハンドラはサービスコールの `def_exc` で第一引数の CPU 例外ハンドラ番号に “EXC\_SVC” を指定して定義します。“SVC *xx*”命令のイミディエート値は、0 から 9 までを  $\mu$ C3/Standard が予約し、10 以上をユーザに開放しています。SVC 例外の CPU 例外ハンドラは次の形式で記述します。引数の `svcno` は“SVC *xx*”命令のイミディエート値です。`reg` は割込み発生時のレジスタ値で、`reg[0]=R0`, `reg[1]=R1`, `reg[2]=R2`, `reg[3]=R3`, `reg[4]=R12`, `reg[5]=PC` が格納されて、レジスタ値の変更は例外復帰時の状態に反映されます。

```
void svc_exchdr(UW svcno, UW *reg)
{
    SVC 例外の CPU 例外ハンドラ本体
}
```

#### 4. 1. 3 その他の CPU 例外ハンドラ

`def_exc` で定義可能なその他の CPU 例外ハンドラの番号は次のようになります。

- EXC\_UND : 未定義命令例外
- EXC\_PRA : プリフェッチアボート例外
- EXC\_DTA : データアボート例外

例外ハンドラは次の形式で記述します。引数の `reg` は例外割込み発生時のレジスタ値で、`reg[0]=R0`, `reg[1]=R1`, `reg[2]=R2`, `reg[3]=R3`, `reg[4]=R12`, `reg[5]=PC` が格納されて、レジスタ値の変更は例外復帰時の状態に反映されます。`psr` は例外発生時の `cpsr` の値です。

```
void und_exchdr(UW *reg, UW psr)
{
    CPU 例外ハンドラ本体
}
```

## 4. 2 割込みハンドラと割込みサービスルーチン

割込みコントローラのドライバに依存するため、各 CPU の割込みコントローラのドライバを参照してください。



## 第5章 CPU 依存ドライバ

### 5. 1 標準 MPU ドライバ DDR\_CORTEXR\_MPU. xxx

---

<code>_ddr_cortexr_mpu_init</code>	MPU の初期化
------------------------------------	----------

---

#### 【書式】

```
void _ddr_cortexr_mpu_init(void * mpu_cfgtbl)
```

---

#### 【解説】

MPU の初期化を行います。OS を起動する前、かつメモリの設定後に本関数を発行して MPU を初期化してください。本関数はアセンブラ言語で記述されています。本関数を発行するために、領域のサイズ／ベースアドレス／アクセス許可／属性を記述したテーブル `mpu_cfgtbl` が必要です。このテーブルの実装例は次のようになります。

領域サイズは次の値から選択します。

REGION_32B	32B
REGION_64B	64B
REGION_128B	128B
REGION_256B	256B
REGION_512B	512B
REGION_1K	32B
REGION_2K	2KB
REGION_4K	4KB
REGION_8K	8KB
REGION_16K	16KB
REGION_32K	32KB
REGION_64K	64KB
REGION_128K	128KB
REGION_256K	256KB
REGION_512K	512KB
REGION_1M	1MB
REGION_2M	2MB
REGION_4M	4MB
REGION_8M	8MB
REGION_16M	16MB

REGION_32M	32MB
REGION_64M	64MB
REGION_128M	128MB
REGION_256M	256MB
REGION_512M	512MB
REGION_1G	1GB
REGION_2G	2GB
REGION_4G	4GB

アクセス許可は次の値から選択します。

AP_NA	アクセス禁止
AP_RW	読み書き可
AP_RO	書き込み禁止
AP_RWNA	読み書き可、ユーザモードからはアクセス禁止
AP_RWRO	読み書き可、ユーザモードからは書き込み禁止
AP_RONA	書き込み禁止、ユーザモードからはアクセス禁止

属性の定義は次の値から選択します。

ATR_STRG	ストロングリオーダ
ATR_SDEV	共有デバイス
ATR_WTNW	ライトスルー、書き込み割り当てなし
ATR_WBNW	ライトバック、書き込み割り当てなし
ATR_NONC	キャッシュ不可
ATR_WBAW	ライトバック、書き込み割り当てあり
ATR_NDEV	非共有デバイス
ATR_SELA	詳細属性あり

ATR\_SELA を選択した場合には、L1 キャッシュと L2 キャッシュの属性を個別に設定する詳細属性を指定することができます。

L1 キャッシュの詳細属性は次の値から選択します。

ATR_INONC	L1 キャッシュ不可
ATR_IWBAW	L1 キャッシュライトバック、書き込み割り当てあり
ATR_IWTNW	L1 キャッシュライトスルー、書き込み割り当てなし
ATR_IWBNW	L1 キャッシュライトバック、書き込み割り当てなし

L2 キャッシュの詳細属性は次の値から選択します。

ATR_ONONC	L2 キャッシュ不可
ATR_OWBAW	L2 キャッシュライトバック、書き込み割り当てあり
ATR_OWTNW	L2 キャッシュライトスルー、書き込み割り当てなし
ATR_OWBNW	L2 キャッシュライトバック、書き込み割り当てなし

これらの初期化内容は、次の例のように 4 つのパラメータを一組にして複数組を定義し、領域サイズが 0 のパラメータを終端とします。

### 【ARM C/C++ Compiler 版】

```
INCLUDE DDR_CORTEXR_MPU.h
```

```
mpu_cfgtbl1
```

```

;           領域サイズ,      ベースアドレス,      アクセス許可,      属性
DCD REGION_512K, 0x24000000, AP_RO, ATR_WTNW
DCD REGION_512K, 0x22000000, AP_RW, ATR_WBNW
DCD 0x00000000, 0x00000000, 0, 0

```

---

---

## **\_kernel\_synch\_cache**

## **データ同期バリア**

---

---

### **【書式】**

```
void _kernel_synch_cache(void)
```

---

### **【解説】**

書き込みバッファにデータが存在する場合はそのデータを書き戻します。

---

---

## **\_memory\_barrier**

## **データメモリバリア**

---

---

### **【書式】**

```
void _memory_barrier(void)
```

---

### **【解説】**

この前後の命令によるメモリアクセスの順序を保証します。

---

---

## **\_kernel\_clean\_data\_cache**

## **データキャッシュのクリーニング**

---

---

### **【書式】**

```
void _kernel_clean_data_cache(void *addr, SIZE size)
```

---

### **【パラメータ】**

void *	addr	クリーニングする領域の先頭番地
SIZE	size	クリーニングする領域のバイト数

---

### **【解説】**

ddr の番地から size のバイト数のデータキャッシュを外部メモリに書き戻して無効化します。当該領域のデータキャッシュが存在しない場合はメモリやキャッシュに影響を与えません。また、同じキャッシュラインにクリーニングする領域以外の他の領域が存在する場合は、その領域もクリーニングされます。

---



---

**\_kernel\_invalid\_data\_cache      データキャッシュの無効化**


---



---

**【書式】**

```
void _kernel_invalid_data_cache(void *addr, SIZE size)
```

---

**【パラメータ】**

void*	addr	無効化する領域の先頭番地
SIZE	size	無効化する領域のバイト数

---

**【解説】**

addr の番地から size のバイト数のデータキャッシュを無効化します。当該領域のデータキャッシュが存在しない場合はメモリやキャッシュに影響を与えません。また、同じキャッシュラインに無効化する領域以外の他の領域が存在する場合は、その領域も無効化されます。

---



---

**\_kernel\_invalid\_inst\_cache      命令キャッシュの無効化**


---



---

**【書式】**

```
void _kernel_invalid_inst_cache(void *addr, SIZE size)
```

---

**【パラメータ】**

void *	addr	無効化する領域の先頭番地
SIZE	size	無効化する領域のバイト数

---

**【解説】**

addr の番地から size のバイト数の命令キャッシュを無効化します。当該領域の命令キャッシュが存在しない場合はメモリやキャッシュに影響を与えません。また、同じキャッシュラインに無効化する領域以外の他の領域が存在する場合は、その領域も無効化されます。

---



---

**\_kernel\_invalid\_cache      すべてのキャッシュの無効化**


---



---

**【書式】**

```
void _kernel_invalid_cache(void)
```

---

**【解説】**

一次／二次／データキャッシュ／命令キャッシュのすべてのキャッシュを無効化(初期化)

します。

## 5. 2 Renesas RZ/T1 のドライバ

### 5. 2. 1 I/O アドレスと割込み番号の定義

RZ/T1 の I/O アドレスと割込み番号の定義は RZT1\_UC3.h に実装しています。このファイルはドライバのインクルードファイルのフォルダに収録しています。

### 5. 2. 2 割込みドライバ DDR\_RZT1\_VIC.c

割込み要因ごとに独立した割込みハンドラや割込みサービスルーチンの生成が可能です。生成時に割込み要因の割込み番号を指定します。設定された割込み優先度に従って多重割込みの受け付けが可能です。使用できる割り込み優先度は 0（最高）～15（最低）です。割込みハンドラや割込みサービスルーチンの開始直後の CPU ロックは解除状態になっています。

---

---

`_ddr_rzt1_vic_init`

割込みコントローラの初期化

---

---

#### 【書式】

```
void _ddr_rzt1_vic_init(void)
```

---

#### 【解説】

割込みコントローラを初期化します。本関数はカーネルの起動前に発行してください。

#### 【プログラム例】

```
_ddr_rzt1_vic_init();
:
start_uC3(&csys, initpr);
```

**dis\_int****割込みの禁止****【書式】**

```
ER ercd = dis_int(INTNO intno)
```

**【パラメータ】**

INTNO	intno	割込み番号

**【戻り値】**

ER	ercd	正常終了 (E_OK) またはエラーコード

**【エラーコード】**

E_PAR	パラメータエラー

**【解説】**

intno の割り込み番号の割込みを禁止します。このシステムコールはタスクコンテキストだけでなく、非タスクコンテキストからも呼び出すことができます。

**ena\_int****割込みの許可****【書式】**

```
ER ercd = ena_int(INTNO intno)
```

**【パラメータ】**

INTNO	intno	割込み番号

**【戻り値】**

ER	ercd	正常終了 (E_OK) またはエラーコード

**【エラーコード】**

E_PAR	パラメータエラー

**【解説】**

intno の割り込み番号の割込みを許可します。このシステムコールはタスクコンテキストだけでなく、非タスクコンテキストからも呼び出すことができます。本関数を発行する前に、割込みハンドラを定義するか、または、割込みサービスルーチンを生成する必要があります。



---



---

**set\_type\_int**                      **割込み検出タイプの指定**


---



---

**【書式】**

```
ER ercd = set_type_int(INTNO intno, int edge_mode)
```

---

**【パラメータ】**

INTNO	intno	割込み番号
int	edge_mode	検出タイプ IRQ_TYPE_LEVEL (0 : レベル検出) IRQ_TYPE_EDGE (1 : エッジ検出)

---

**【戻り値】**

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

---

**【エラーコード】**

E_PAR	パラメータエラー
-------	----------

---

**【解説】**

割込み番号 `intno` の割込み検出タイプを指定します。このシステムコールはタスクコンテキストだけでなく、非タスクコンテキストからも呼び出すことができます。本関数は `intno` の割込みが禁止状態で発行してください。

**【プログラム例】**

IRQ 端子割り込み 5 の検出タイプをエッジ設定し、割込みを有効にするには次のように記述します。

```
extern ER set_type_int(INTNO intno, int edge_mode);
:
set_type_int(INT_IRQ5, IRQ_TYPE_EDGE);
ena_int(INT_IRQ5);
```

### 5. 2. 3 周期タイマドライバ DDR\_RZT1\_CMT0.c

`_ddr_rzt1_cmt0_init`

周期タイマの初期化

#### 【書式】

```
void _ddr_rzt1_cmt0_init(UINT tick)
```

#### 【パラメータ】

UINT	tick	チック時間
------	------	-------

#### 【解説】

カーネルで使用する周期タイマ用にコンペアマッチタイマを初期化します。本タイマの初期化内容を次の例のようにファイルの `DDR_RZT1_CMT0_cfg.h` に記述します。

#### 【設定例】

```
#define CH_TIM      0      /* CMT のチャンネル番号 (0~2)      */
#define IPL_TIMER  15      /* 周期タイマ割込み優先度          */
#define CLK_SRC     0      /* CMT のクロックソース(0~3)      */
                        /* 0:  PCLKD/8                    */
                        /* 1:  PCLKD/32                   */
                        /* 2:  PCLKD/128                  */
                        /* 3:  PCLKD/512                  */
```

#### 5. 2. 4 クロック発生器ドライバ DDR\_RZT1\_CPG.c

クロック発生器に入力されている発振器の周波数を次の例のようにファイルの DDR\_RZT1\_CPG\_cfg.h に記述します。

##### 【設定例】

```
#define MAIN_CLK          25000000u    /* 入力クロック周波数 */
#define AUDIO_CLK         11289600u    /* オーディオ用クロック周波数 */
```

---

---

## **\_ddr\_rzt1\_cpg\_get\_sericlkl**

## **高速シリアルクロック (SERICLK) の取得**

---

---

### **【書式】**

UW **\_ddr\_rzt1\_cpg\_get\_sericlkl(void)**

---

### **【解説】**

クロック発生器により生成されている高速シリアルクロック (SERICLK) の周波数を取  
得することができます。取り出す周波数の単位は Hz です。

### **【プログラム例】**

```
#include "DDR_RZT1_CPG.h"
```

```
UW frq;  
frq = _ddr_rzt1_cpg_get_sericlkl();
```

---

---

## **\_ddr\_rzt1\_cpg\_get\_ckio**

## **外部バスクロック (CKIO) の取得**

---

---

### **【書式】**

UW **\_ddr\_rzt1\_cpg\_get\_ckio(void)**

---

### **【解説】**

クロック発生器により生成されている外部バスクロック (CKIO) の周波数を取  
得することができます。取り出す周波数の単位は Hz です。

### **【プログラム例】**

```
#include "DDR_RZT1_CPG.h"
```

```
UW frq;  
frq = _ddr_rzt1_cpg_get_ckio();
```

---

---

**\_ddr\_rzt1\_cpg\_chg\_cpucclk****CPU クロック周波数の変更**

---

---

**【書式】**

ER \_ddr\_rzt1\_cpg\_chg\_cpucclk(UINT cpucsel)

**【パラメータ】**

UINT	cpucsel	CPU 動作周波数選択
		CPG_CPUCKSEL_150_MHZ (0:150MHz)
		CPG_CPUCKSEL_300_MHZ (1:300MHz)
		CPG_CPUCKSEL_450_MHZ (2:450MHz)
		CPG_CPUCKSEL_600_MHZ (3:600MHz)

---

**【エラーコード】**

E\_PAR                  パラメータエラー（使用できない周波数、あるいは組み合わせ）

**【解説】**

クロックパルス発生器により生成されている CPU クロックを変更します。

**【プログラム例】**

CPU クロックを 600MHz に変更する場合は、以下のコードになります。

```
#include "DDR_RZT1_CPG.h"
```

```
_ddr_rzt1_cpg_chg_cpucclk(CPG_CPUCKSEL_600_MHZ);
```

## 5. 2. 5 標準 COM ドライバ DDR\_RZT1\_SCIF.c

---

---

**\_ddr\_rzt1\_scif\_init****SCIF ポートの初期化**

---

---

**【書式】**


---

```
ER _ddr_rzt1_scif_init(ID devid, volatile struct st_scif *scif_port)
```

---

**【パラメータ】**

ID	devid	デバイスの ID 番号
volatile struct st_scif *	scif_port	SCIF のデバイスアドレス

---

**【解説】**

SCIF を初期化します。初期化後は標準 COM ポートドライバが使用できるようになります。devid で指定したデバイスの ID 番号は標準 COM ポートドライバで使

**【プログラム例】**

SCIF のチャンネル 2 を DID\_SCIF2 のデバイス ID 番号で初期化する場合は次の例のようになります。

```
#include "RZT1_UC3.h"
#include "DDR_COM.h"

extern ER _ddr_rzt1_scif_init(ID devid, volatile struct st_scif *scif_port);
:
_ddr_rzt1_scif_init(DID_SCIF2, &SCIF2);
```

本 COM ポートの初期化内容を次の例のようにファイルの DDR\_RZT1\_SCIF\_cfg.h に記述します。チャンネルごとに 9 つのパラメータがあります。複数のチャンネルの設定を記述することができます。

```
#define SCIF_n          /* チャンネル n3を使用する */
#define TXBUF_SZn      1024 /* 送信バッファサイズ(0 以上) */
#define RXBUF_SZn      1024 /* 受信バッファサイズ(1 以上) */
#define XOFF_SZn       768  /* XOFF 送出受信バッファデータ数トリガ */
#define XON_SZn        128  /* XON 送出受信バッファデータ数トリガ */
#define RTRG_n         8    /* 受信 FIFO データ数トリガ (1,4,8,14) */
```

---

<sup>3</sup> n には 0~4 までの数値が入ります。

```
#define TTRG_n      4      /* 送信 FIFO データ数トリガ (0,2,4,8) */
#define RTRG_n      14     /* RTS 出力アクティブ (1,4,6,8,10,12,14,15) */
#define IPL_SCIFn    15     /* 割込みレベル(0~15) */
```

## 第6章 プロセッサに依存した注意事項

### 6. 1 浮動小数点演算の使用

デフォルトでは各コンテキストはFPU抑止状態で起床されます。よって、コンテキストで浮動小数点演算を行いたい場合は、コンテキストの生成時にTA\_FPU属性を指定してください。TA\_FPU属性の指定が無いコンテキストで浮動小数点レジスタを操作した場合は未定義命令例外が発生します。TA\_FPU属性を指定できるシステムコールはcre\_tsk／cre\_cyc／cre\_alm／cre\_isr／def\_inh／def\_ovrです。なお、初期化ハンドラには予めTA\_FPU属性が付与されています。

#### 6. 1. 1 FPU イネーブラの登録

TA\_FPU 属性の指定が無いコンテキストで浮動小数点レジスタを操作し、未定義命令例外が発生した場合に、システムダウンを防ぐため強制的に FPU 許可状態にする FPU イネーブラ (\_kernel\_enavfp) を使用することができます。使用方法は、はじめに「4. 1. 1 CPU 例外ハンドラの登録準備」のように未定義命令例外に CPU 例外ハンドラを登録できるようベクターテーブルを記述します。次に CPU 例外ハンドラを下記の例にしたがって記述します。FPU イネーブラが E\_OK 以外を返した場合は、既に FPU 許可状態か、浮動小数点レジスタの操作以外です。この場合は適切な処理を記述してください。

```
void undefined_instruction_handler(UW *reg, UW psr)
{
    ER ercd;
    ercd = _kernel_enavfp(reg, psr);
    if (ercd != E_OK) {
        int_abort();
    }
}

const T_DEXC dexc_vfp = {TA_NULL, (FP)undefined_instruction_handler};
```

そして、最後に初期化ルーチン内で CPU 例外ハンドラを登録します。

```
def_exc(EXC_UDF, (T_DEXC *)&dexc_vfp);
```



---

**μC3/Standard ユーザーズガイド プロセッサ依存部 Cortex-R4 編**

---

2014年 10月          第 1 版

イー・フォース株式会社    <http://www.eforce.co.jp/>  
〒103-0006 東京都中央区日本橋富沢町 5 - 4 ゲンベエビル 7 F  
TEL 03-5614-6918    FAX 03-5614-6919  
お問い合わせ [info@eforce.co.jp](mailto:info@eforce.co.jp)  
Copyright (C) 2009-2014 eForce Co.,Ltd. All Rights Reserved.

---