µC3/Standard ユーザーズガイド
プロセッサ依存部 Cortex-A9 編

第 15 版　イー・フォース株式会社

はじめに

本書はμC3/Standard の Cortex-A9 に依存する事項を説明します。「μC3/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。
μITRON は"Micro Industrial TRON"の略称です。
μITRON4.0 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。
μC3 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

版数	内容
第２版	VFPv3_D16 対応ライブラリに関して追加
	RealView Development Suite(RVDS) への対応に関して追加
第３版	MMU の変換テーブルに指定する属性に関して追加
	Renesas RZ/A1H への対応に関して追加
第４版	RealView Development Suite 対応を ARM C/C++ Compiler 対応に変更
	Freescale i.MX6 への対応に関して追加
第５版	Cyclone V SoC への対応に関して追加
第６版	DS-5 Altera Edition 付属 gcc への対応に関して追加
	Toshiba TZ2000 への対応に関して追加
第７版	カーネルライブラリ名の誤記を訂正
第８版	CubeGEAR 付属 gcc への対応に関して追加
	Cyclone V SoC の SCU_CLK を修正
第９版	５．６章に TZ2100 用の仕様を追加
第１０版	_memory_barrier 関数を_kernel_memory_barrier 関数に変更
第１１版	_kernel_clean_data_cache 関数、_kernel_invalid_data_cache 関数、_kernel_invalid_cache 関数の説明を修正
第１２版	ena_int, dis_int の仕様変更に伴い解説を追加しました。 (追加対象：Zynq7000, CycloneV SoC, i.MX6, RZ/A シリーズ)
第１３版	CPU ロック状態の説明で、chg_ims(1)と chg_ims(0)については、呼び出しコンテキストとして、割込みとタイムイベントハンドラからの呼び出しを禁止するとしました。
第１４版	１．１ フォルダ構成 に，フォルダ構成が異なるパケージについて説明を追加しました。 １．２ カーネルライブラリに，exeGCC 版の説明を追加しました。
第１５版	２．１ 割込み 割込みに関する章構成を変更し，割込みレベルに関する説明を追記しました。 標準 MMU ドライバに「データキャッシュのフラッシュ関数」を追記しました。

目次

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。

```
uC3Std
  ├──Document ……  ユーザーズガイド
  ├──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
  │    │    └──CortexA ……  Cortex-A 依存のインクルードファイル
  │    ├──lib……  カーネルライブラリ
  │    │    └──CortexA ……  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
  │         └──CortexA ……  Cortex-A 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は次の例のようになります。

```
C:¥uC3Std¥Kernel¥inc
C:¥uC3Std¥Kernel¥inc¥CortexA
C:¥uC3Std¥Driver¥inc
```

なお，リリースバージョンによっては，ルートフォルダ階層が増えているパッケージがあります。その際のインクルードパスは次の例のようになります。

```
C:¥uC3¥Standard¥Kernel¥inc
C:¥uC3¥Standard¥Kernel¥inc¥CortexA
C:¥uC3¥Standard¥Driver¥inc
```

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 の有無などの組み合わせで次の種類を用意しています。

【Xilinx ISE Design suite 版】

ARM/Thumb	エンディアン	VFP	ライブラリ名
ARM	Little	無	libuC3cortexal.a
Thumb	Little	無	libuC3cortexatl.a
ARM	Little	VFPv3 or NEON	libuC3cortexafl.a
Thumb	Little	VFPv3 or NEON	libuC3cortexaftl.a

【IAR Embedded Workbench 版】

ARM/Thumb	エンディアン	VFP	ライブラリ名
ARM	Little	無	uC3cortexal.a
Thumb	Little	無	uC3cortexatl.a
ARM	Little	VFPv3_D16	uC3cortexahl.a
Thumb	Little	VFPv3_D16	uC3cortexahtl.a
ARM	Little	VFPv3 or NEON	uC3cortexafl.a
Thumb	Little	VFPv3 or NEON	uC3cortexaftl.a

【ARM C/C++ Compiler 版】

ARM/Thumb	エンディアン	VFP	ライブラリ名
ARM	Little	無	uC3cortexal.l
Thumb	Little	無	uC3cortexatl.l
ARM	Little	VFPv3_D16	uC3cortexahl.l
Thumb	Little	VFPv3_D16	uC3cortexahtl.l
ARM	Little	VFPv3 or NEON	uC3cortexafl.l
Thumb	Little	VFPv3 or NEON	uC3cortexaftl.l

【DS-5 Altera Edition 付属 gcc 版】

ARM/Thumb	エンディアン	VFP	ライブラリ名
ARM	Little	無	libuC3cortexal_ds5-gcc.a
Thumb	Little	無	libuC3cortexatl_ds5-gcc.a
ARM	Little	VFPv3 or NEON	libuC3cortexafl_ds5-gcc.a
Thumb	Little	VFPv3 or NEON	libuC3cortexaftl_ds5-gcc.a

【CubeGEAR 付属 gcc 版】

ARM/Thumb	エンディアン	VFP	ライブラリ名
ARM	Little	無	libuC3cortexal_cg.a
Thumb	Little	無	libuC3cortexatl_cg.a
ARM	Little	VFPv3 or NEON	libuC3cortexafl_cg.a
Thumb	Little	VFPv3 or NEON	libuC3cortexaftl_cg.a

【exeGCC 版】

ARM/Thumb	エンディアン	VFP	ライブラリ名
ARM	Little	無	libuC3cortexal.a
Thumb	Little	無	libuC3cortexatl.a
ARM	Little	VFPv3 or NEON	libuC3cortexafl.a
Thumb	Little	VFPv3 or NEON	libuC3cortexaftl.a

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。

第２章　プロセッサに依存する状態

２．１　割込み

２．１．１　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。μC3/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。

２．１．２　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 はマスク値を指定し、get_ims は現在のマスク値を返します。割込みマスク値 0 は IRQ 割込み許可で、割込みマスク値 1 は IRQ 割込み禁止です。

２．１．３　割込みレベル

割込みレベルは cre_isr/acre_isr や def_inh のパラメータ imask で指定します。指定できる割込みレベルの範囲は CPU の割込みドライバに依存しますが，通常は

（高優先度）0x00，0x08，0x10，0x18，…，0xF8（低優先度）

と 8 刻みで 32 レベルが指定できます。(※最後の 0xF8 の割込みレベルは GIC のマスクレジスタの値と等しい値になるため割込みは発生しません。)

同じ割込み番号に対して複数の割込みサービスルーチンを登録した場合，最初に登録した割込みサービスルーチンのパラメータ imask が割込みレベルとして使用されます。

２．２　CPU ロック状態

μC3/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。

なお、chg_ims(1)と chg_ims(0)については、呼び出しコンテキストとして、割込みとタイ

ムイベントハンドラからの呼び出しを禁止します。

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μITRON4.0 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。

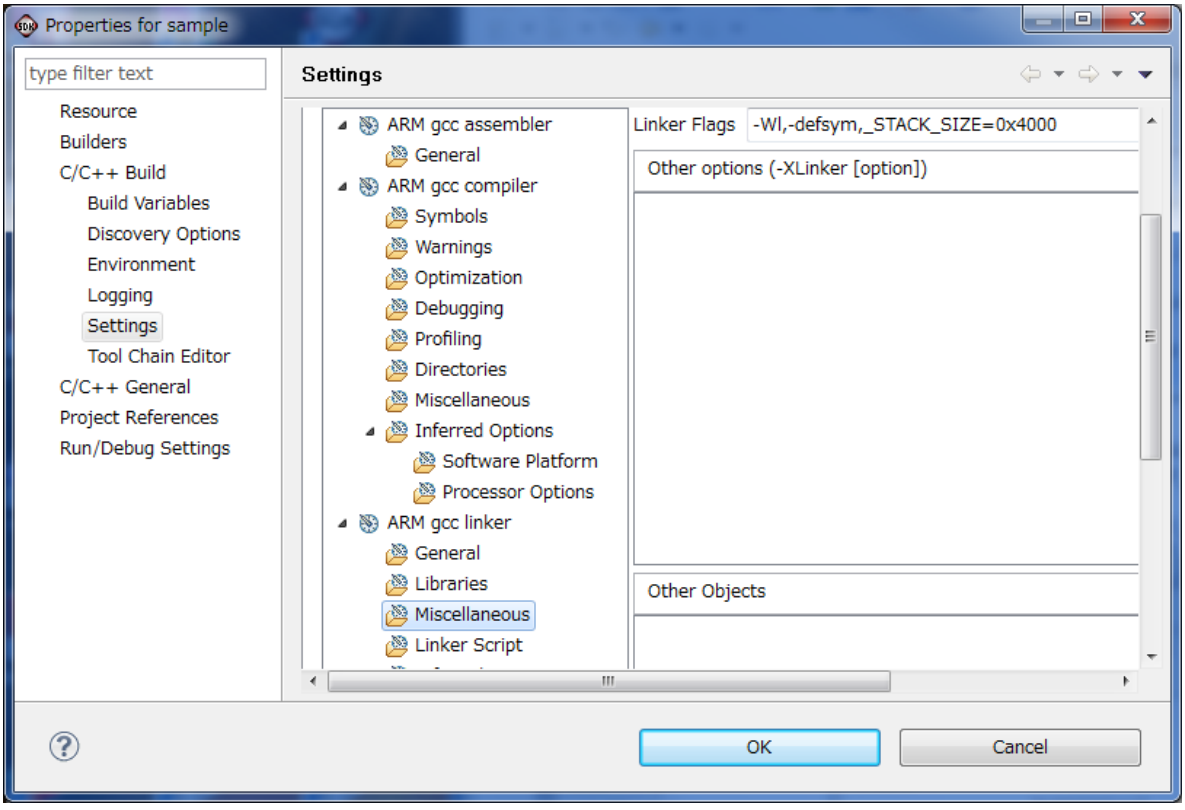
第３章　カーネルのコンフィグレーション

３．１　システムスタック

μC3/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　Xilinx ISE Design suite 使用時のシステムスタック定義方法

SDK の場合はリンカコマンドファイル（.ld）でセクション定義と共に領域の確保を行います。領域の確保はシンボルの_STACK_SIZE に設定し、デフォルトでは 0x2000（KB）を確保しています。変更は Properties の下記画面の”Linker Flags”で設定します。



３．１．２　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。

３．１．３　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、スキャッタファイル(.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。

３．１．４　DS-5 Altera Edition 付属 gcc 使用時のシステムスタック定義方法

DS-5 Altera Edition 付属の gcc（arm-none-eabi-gcc）を使用する場合は、リンカスクリプトファイル（.ld）でセクション定義と共に領域の確保を行います。領域の確保はシンボルの_STACK_SIZE に設定し、デフォルトでは 0x2000（KB）を確保しています。変更はリンカスクリプトファイルをエディタ等で修正して下さい。

３．１．５　CubeGEAR 付属 gcc 使用時のシステムスタック定義方法

CubeGEAR 付属の gcc（arm-none-eabi-gcc）を使用する場合は、リンカスクリプトファイル（.ld）でセクション定義と共に領域の確保を行います。領域の確保はシンボルの_STACK_SIZE に設定し、デフォルトでは 0x2000（KB）を確保しています。変更はリンカスクリプトファイルをエディタ等で修正して下さい。

３．２　各モードのスタック

μC3/Standard ARM 版では、ユーザ(システム)モード以外に割込み(IRQ)モードとスーパバイザ(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。割込み(IRQ)モードのスタックサイズは単一の割込み発生で 64 バイトを使用します。割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　Xilinx ISE Design suite 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リンカコマンドファイルで領域の確保を行います。デフォルトでは IRQ 割込みモードに 8KB（0x2000）を、その他のモードに 1024B（0x400）を確保しています。以下が各スタックサイズのシンボルです。

- _SUPERVISOR_STACK_SIZE　：スーパバイザモードのスタックサイズ
- _IRQ_STACK_SIZE　：割込み(IRQ)モードのスタックサイズ
- _FIQ_STACK_SIZE　：高速割込み(FIQ)モードのスタックサイズ
- _UNDEFINED_STACK_SIZE　：未定義(UND)モードのスタックサイズ
- _ABORT_STACK_SIZE　：アボート(ABT)モードのスタックサイズ

３．２．２　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：スーパバイザモードのスタックサイズ
- IRQ_STACK：割込み(IRQ)モードのスタックサイズ
- FIQ_STACK：高速割込み(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

３．２．３　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、スキャッタ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、スーパバイザモードは"SVC_STACK"、割込み(IRQ)モードは"IRQ_STACK"、高速割込み(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。

```
LR 0x00100000 0x40000000
{
    RAM_EXEC 0x00100000 0x01000000
    {
        prst_zynq7000.o (.intvec, +FIRST); Initialization code
        * (+RO)                          ;
    }
    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)                          ;
    }
}
```

３．２．４　DS-5 Altera Edition 付属 gcc 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、リンカスクリプトファイルで領域の確保を行います。デフォルトでは IRQ 割込みモードに 8KB(0x2000)を、その他のモードに 1024B(0x400)を確保しています。以下が各スタックサイズのシンボルです。

- _SUPERVISOR_STACK_SIZE　：スーパバイザモードのスタックサイズ
- _MONITOR_STACK_SIZE　：モニタモードのスタックサイズ
- _IRQ_STACK_SIZE　：割込み(IRQ)モードのスタックサイズ
- _FIQ_STACK_SIZE　：高速割込み(FIQ)モードのスタックサイズ
- _UNDEFINED_STACK_SIZE　：未定義(UND)モードのスタックサイズ
- _ABORT_STACK_SIZE　：アボート(ABT)モードのスタックサイズ

３．２．５　CubeGEAR 付属 gcc 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、リンカスクリプトファイルで領域の確保を行います。デフォルトでは IRQ 割込みモードに 8KB(0x2000)を、その他のモードに 1024B(0x400)を確保しています。以下が各スタックサイズのシンボルです。

- _SUPERVISOR_STACK_SIZE　：スーパバイザモードのスタックサイズ
- _MONITOR_STACK_SIZE　：モニタモードのスタックサイズ
- _IRQ_STACK_SIZE　：割込み(IRQ)モードのスタックサイズ
- _FIQ_STACK_SIZE　：高速割込み(FIQ)モードのスタックサイズ
- _UNDEFINED_STACK_SIZE　：未定義(UND)モードのスタックサイズ
- _ABORT_STACK_SIZE　：アボート(ABT)モードのスタックサイズ

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、プリフェッチアボート例外、データアボート例外を 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μITRON4.0 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、プリフェッチアボート例外、データアボート例外の 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
        b       Reset_Handler
        ldr     pc, _UndHandler
        dr      pc, _SwiHandler
        ldr     pc, _PreHandler
        ldr     pc, _AbtHandler
        nop
        ldr     pc, _IrqHandler
        ldr     pc, _FiqHandler

        LTORG
        DATA
                        DCD     0
_UndHandler             DCD     _kernel_undhdr
_SwiHandler             DCD     _kernel_svchdr
_PreHandler             DCD     _kernel_prahdr
_AbtHandler             DCD     _kernel_dtahdr
                        DCD     0
_IrqHandler             DCD     _kernel_inthdr
_FiqHandler             DCD     FiqHandler
```

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μC3/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 MMU ドライバ DDR_CortexA_MMU.xxx

_ddr_cortexa_mmu_init　　MMU の初期化

【書式】

void　_ddr_cortexa_mmu_init(void * mmu_ttb, void * mmu_cfgtbl)

【解説】

MMU 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 MMU 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、物理アドレスから仮想アドレスへの変換／アクセス許可／属性を記述した変換テーブルの mmu_cfgtbl と、変換テーブルのアドレスの mmu_ttb 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。

変換テーブルの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

変換テーブルの属性の定義は次の値から選択します。

ATR_STRG	ストロングオーダ
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	L2 キャッシュを含めた詳細属性あり

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 5 つのパラメータを一組にして複数組を定義し、すべてが 0 のパラメータを終端とします。また、領域の定義が重複した場合には、後に定義した内容が優先されます。

【Xilinx Design Tools 版】

```
  #include "DDR_CortexA_MMU.sh"

      .align   2

  mmu_cfgtbl
  @              バイト数,       物理アドレス,      仮想アドレス,    アクセス許可,       属性
      .long    0x00100000,   0xC0000000,   0x00000000,     AP_RO,    ATR_WTNW
      .long    0x00100000,   0xC0100000,   0xC0100000,     AP_RW,    ATR_WBNW
               :
      .long    0x00000000,   0x00000000,   0x00000000,       0,            0

          .section .mmu_tbl
          .align   14
          .globl   mmu_space
mmu_space:  .space   0x00100000
```

【IAR Embedded Workbench 版】

```
#include "DDR_CortexA_MMU.inc"

        ALIGNROM 2
        DATA

mmu_cfgtbl:
;                バイト数,        物理アドレス,  仮想アドレス,  アクセス許可,  属性
        DCD   0x08000000, 0x00000000, 0x00000000,   AP_RO,   ATR_WTNW
        DCD   0x00A00000, 0x20000000, 0x20000000,   AP_RW,   ATR_WBNW
        DCD   0x00000000, 0x00000000, 0x00000000,        0,        0

mmu_ttb                 DCD       SFB(TLB_RAMSEC)
```

【ARM C/C++ Compiler 版】

```
        INCLUDE DDR_CortexA_MMU.h

mmu_cfgtbl1
;                 バイト数,      物理アドレス,  仮想アドレス,  アクセス許可,  属性
        DCD    0x01000000, 0x10000000, 0x00000000,    AP_RO,   ATR_WTNW
        DCD    0x10000000, 0x20000000, 0x20000000,    AP_RW,   ATR_WBNW
        DCD    0x00000000, 0x00000000, 0x00000000,    0,       0

mmu_ttb                 DCD       0x2FF00000
```

_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。

_kernel_memory_barrier　　データメモリバリア

【書式】

void　_kernel_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。

（※古いバージョンや異なる CPU 依存部では _memory_barrier, _kernel_data_memory_barrier 関数名で定義されている場合があります。）

_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】

addr の番地から size のバイト数のデータを L1 データキャッシュから低いメモリ階層（L2 キャッシュ・外部メモリ等）に書き戻します。（※Cortex-A9 では L2 キャッシュのクリーニングは別途おこなう必要があります。）当該領域のデータが L1 データキャッシュ上に存在しない場合は影響を与えません。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、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 L1 データキャッシュを無効化します。(※Cortex-A9 では L2 キャッシュの無効化は別途おこなう必要があります。)当該領域のデータが L1 キャッシュ上に存在しない場合は影響を与えません。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_flush_data_cache　　データキャッシュのフラッシュ

【書式】

void　_kernel_flush_data_cache(void *addr, SIZE size)

【パラメータ】

void *	addr	フラッシュする領域の先頭番地
SIZE	size	フラッシュ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外部メモリに書き戻して無効化します。(※Cortex-A9 では L2 キャッシュの無効化は別途おこなう必要があります。) 当該領域のデータが L1 キャッシュ上に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 L1 命令キャッシュを無効化します。（※Cortex-A9 では L2 キャッシュの無効化は別途おこなう必要があります。）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_tlb　　TLB の無効化

【書式】

```
void _kernel_invalid_tlb(void)
```

【解説】

プロセッサ内にフェッチしている TLB を無効化します。

_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】

L1 キャッシュ（命令キャッシュ・データキャッシュ）を無効化（初期化）します。

５．２　Zynq7000 のドライバ

５．２．１　I/O アドレスと割込み番号の定義

Zynq7000 の I/O アドレスと割込み番号の定義は zynq7000.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_ZYNQGIC.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 16 刻みで 0（最高）～240（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_gic_init	割込みコントローラの初期化

【書式】

```
void    _ddr_gic_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。また、初期化内容を次の例のようにファイルの DDR_ZINQGIC_cfg.h に記述します。

【設定例】

```
#define IPL_MASK     0xf0       /* priority mask for ICCICR register */
```

【プログラム例】

```
_ddr_gic_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。マルチコア構成の場合、呼び出したコアに割込みが割当てられている場合、割り当てを解放します。呼び出したコアに割当てられていない場合は E_PAR を返します。

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。マルチコア構成の場合、呼び出したコアに割込みを割り当てます。すでに他のコアに割り当てられている場合は E_PAR を返します。

５．２．３　周期タイマドライバ DDR_ZYNQPTIMER.c

_ddr_private_timer_init　　周期タイマの初期化

【書式】

void　_ddr_private_timer_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 PTIMER（Private Timer）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_ZYNQPTIMER_cfg.h に記述します。

【設定例】

```
#define PS_CLK      1       /* PS CLK の入力周波数（Hz) */
#define IPL_PTIM    0xe0    /*  周期タイマ割込み優先度  */
```

５．２．４　標準 COM ドライバ DDR_ZYNQUART.c

_ddr_zynquart_init　　UART ポートの初期化

【書式】

ER　_ddr_zynquart_init(ID devid, volatile struct t_uart *uart_port)

【パラメータ】

ID	devid	デバイスの ID 番号
volatile struct t_uart *	uart_port	UART のデバイスアドレス

【解説】

UART 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用します。

【プログラム例】

UART のチャネル 1 を ID_UART1 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 "zynq7000.h"
#include "DDR_COM.h"

extern ER   _ddr_zynquart_init (ID, volatile struct t_uart *);
            :
            _ddr_zynquart_init (ID_UART1, &REG_UART1);
```

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_ZYNQUART_cfg.h に記述します。チャネルごとに 7 つのパラメータがあります。複数のチャネルの設定を記述することができます。

```
#define   UART_n          /* チャネル n 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_1          14      /* レシーブ FIFO データ数トリガ（1..63） */
#define   IPL_UARTn       0xa0    /* 割込みレベル */
```

５．３　Renesas RZ/A1H のドライバ

５．３．１　I/O アドレスと割込み番号の定義

RZ/A1H の I/O アドレスと割込み番号の定義は rza1h_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。

５．３．２　割込みドライバ DDR_RZA1H_GIC.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 8 刻みで 0（最高）～248（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_rza1h_gic_init	割込みコントローラの初期化

【書式】

```
void　_ddr_rza1h_gic_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_gic_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。マルチコア構成の場合、呼び出したコアに割込みが割当てられている場合、割り当てを解放します。（※RZ/A1 は 1 コアですが MPCore のため同じ動作を行います。）

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。マルチコア構成の場合、呼び出したコアに割込みを割り当てます。（※RZ/A1 は 1 コアですが MPCore のためこの動作を行います。）

５．３．３　周期タイマドライバ DDR_RZA1H_OSTM.c

_ddr_rza1h_ostm_init　　周期タイマの初期化

【書式】

```
void    _ddr_rza1h_ostm_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 OS Timer 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_RZA1H_OSTM_cfg.h に記述します。

【設定例】

```
#define CH_OSTM     0       /* OS Timer のチャネル番号 (0～1) */
#define IPL_OSTM    16      /*  周期タイマ割込み優先度  */
```

５．３．４　クロックパルス発生器ドライバ DDR_RZA1H_CPG.c

クロックパルス発生器に入力されている発振器の周波数とクロックモードを次の例のようにファイルの DDR_RZA1H_CPG_cfg.h に記述します。

【設定例】

```
#define MAINCLK         13333333        /* クロックソースの周波数 */
#define CLK_MODE        0               /* クロックモード(0～1) */
```

_ddr_rza1h_cpg_getclk　　クロックパルス発生器の出力周波数の取得

【書式】

void　_ddr_rza1h_cpm_getclk(T_CPGFRQ *frq)

【パラメータ】

T_CPGFRQ *　frq		各種周波数情報を入れたパケットへのポインタ
frq の内容（T_CPGFRQ 型）		
UW	I	CPU クロック（Iφ）
UW	G	画像処理クロック（Gφ）
UW	B	内部バスクロック（Bφ）
UW	P1	周辺クロック１（P1φ）
UW	P0	周辺クロック０（P0φ）

【解説】

クロックパルス発生器により生成されている各種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_RZA1H_CPG.h"

    T_CPGFRQ frq;

    _ddr_rza1h_cpg_getclk(&frq);
```

_ddr_rza1h_cpg_stabclk　　クロックパルス発生器の周波数安定待ち

【書式】

void　_ddr_rza1h_cpm_stabgetclk(void)

【解説】

周波数変更後の安定化まで待ちます。

_ddr_rza1h_cpg_chg_clk　　クロックパルス発生器の出力周波数の変更

【書式】

void　_ddr_rza1h_cpm_chg_clk(UINT cpuclk, UINT gpuclk)

【パラメータ】

UINT	cpuclk	CPU クロックの周波数
UINT	gpuclk	画像処理クロックの周波数

【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 Iφと画像処理クロック Gφを変更します。

PLL_1_1_DIV	PLL 出力周波数
PLL_2_3_DIV	PLL 出力周波数の 2/3
PLL_1_3_DIV	PLL 出力周波数の 1/3
PLL_IGNORE	周波数を変更しない

【プログラム例】

CPU クロックだけを PLL 出力周波数の 1/3 に変更し、割込み待ち命令を実行する場合は、以下のコードになります。

```
#include "DDR_RZA1H_CPG.h"

    _ddr_rza1h_cpg_chg_clk(PLL_1_3_DIV, PLL_IGNORE);
    _ddr_rza1h_cpg_stabclk();
    __WFI();
```

５．３．５　標準 COM ドライバ DDR_RZA1H_SCIF.c

_ddr_rza1h_scif_init　　SCIF ポートの初期化

【書式】

ER　_ddr_rza1h_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用します。

【プログラム例】

SCIF のチャネル 1 を ID_SCIF1 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 "rza1h_uC3.h"
#include "DDR_COM.h"

extern ER _ddr_rza1h_scif_init(ID devid, volatile struct st_scif *scif_port);
        :
        _ddr_rza1h_scif_init (ID_SCIF1, &SCIF1);
```

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_RZA1H_SCIF_cfg.h に記述します。チャネルごとに 9 つのパラメータがあります。複数のチャネルの設定を記述することができます。

```
#define   UART_n          /* チャネルｎ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_1      8       /* 受信 FIFO データ数トリガ (1,4,8,1)  */
#define   TTRG_1      4       /* 送信 FIFO データ数トリガ (0,2,4,8)  */
```

```
#define   RTRG_1          14   /* RTS 出力アクティブ（1,4,6,8,10,12,14,15） */
#define   IPL_UARTn       24   /* 割込みレベル */
```

５．４　Freescale i.MX 6Solo/6DuaLite のドライバ

５．４．１　I/O アドレスと割込み番号の定義

i.MX 6Solo/6DualLite の I/O アドレスと割込み番号の定義は imx6sdl.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。

５．４．２　割込みドライバ DDR_FS_MX6GIC.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 16 刻みで 0（最高）～240（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_gic_init	割込みコントローラの初期化

【書式】

```
void    _ddr_gic_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。また、初期化内容を次の例のようにファイルの DDR_FS_MX6GIC_cfg.h に記述します。

【設定例】

```
#define IPL_MASK     0xF0       /* priority mask for ICCICR register */
```

【プログラム例】

```
_ddr_gic_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。マルチコア構成の場合、呼び出したコアに割込みが割当てられている場合、割り当てを解放します。呼び出したコアに割当てられていない場合は E_PAR を返します。

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。マルチコア構成の場合、呼び出したコアに割込みを割り当てます。すでに他のコアに割り当てられている場合は E_PAR を返します。

５．４．３　周期タイマドライバ DDR_FS_MX6PTIMER.c

_ddr_private_timer_init　　周期タイマの初期化

【書式】

void　_ddr_private_timer_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 PTIMER（Private Timer）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_FS_MX6PTIMER_cfg.h に記述します。

【設定例】

```
#define SCU_CLK     792000000UL         /* SCU CLK の入力周波数 (Hz) */
#define IPL_PTIM    0xe0                /*  周期タイマ割込み優先度  */
```

５．４．４　標準 COM ドライバ DDR_FS_MX6UART.c

_ddr_fs_mx6uart_init　　UART ポートの初期化

【書式】

ER　_ddr_fs_mx6uart_init(ID devid, volatile struct t_uart *uart_port)

【パラメータ】

ID	devid	デバイスの ID 番号
volatile struct t_uart *	uart_port	UART のデバイスアドレス

【解説】

UART 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用します。

【プログラム例】

UART のチャネル 1 を ID_UART1 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 "mx6sdl.h"
#include "DDR_COM.h"

extern ER   _ddr_mx6uart_init (ID, volatile struct t_uart *);
            :
            _ddr_mx6uart_init (ID_UART1, &REG_UART1);
```

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_FS_MX6UART_cfg.h に記述します。チャネルごとに 10 のパラメータがあります。複数のチャネルの設定を記述することができます。

```
#define   UART_n        /* チャネル n 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   RXTL_n        24       /* レシーブ FIFO データ数トリガ 0～32 */
#define   TXTL_n        8        /* トランスミット FIFO データ数トリガ 2～32 */
```

```
#define   RTSTL_n       24          /* RTS 出力アクティブトリガ 0～32 */
#define   IPL_UARTn     0xA0        /* 割込みレベル */
#define   DCEDTE_n      0           /* DCE(0)/DTE(1)の選択 */
```

５．５　Cyclone V SoC のドライバ

５．５．１　I/O アドレスと割込み番号の定義

Cyclone V SoC の I/O アドレスと割込み番号の定義は CycloneV_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。

５．５．２　割込みドライバ DDR_CYCLONEV_GIC.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 8 刻みで 0（最高）～248（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_gic_init	割込みコントローラの初期化

【書式】

```
void    _ddr_gic_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。また、初期化内容を次の例のようにファイルの DDR_CYCLONEV_GIC_cfg.h に記述します。

【設定例】

```
#define IPL_MASK     0xF0       /* priority mask for ICCICR register */
```

【プログラム例】

```
_ddr_gic_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。マルチコア構成の場合、呼び出したコアに割込みが割当てられている場合、割り当てを解放します。呼び出したコアに割当てられていない場合は E_PAR を返します。

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。マルチコア構成の場合、呼び出したコアに割込みを割り当てます。すでに他のコアに割り当てられている場合は E_PAR を返します。

５．５．３　周期タイマドライバ DDR_CYCLONEV_PTIMER.c

_ddr_private_timer_init　　周期タイマの初期化

【書式】

void　_ddr_private_timer_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 PTIMER（Private Timer）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_CYCLONEV_PTIMER_cfg.h に記述します。

【設定例】

```
#define SCU_CLK     200000000UL         /* SCU CLK の入力周波数 (Hz) */
#define IPL_PTIM    0xe0                /*  周期タイマ割込み優先度  */
```

５．５．４　標準 COM ドライバ DDR_CYCLONEV_UART.c

_ddr_cyclonev_uart_init　　UART ポートの初期化

【書式】

ER　_ddr_cyclonev_uart_init(ID devid, volatile struct t_uart *uart_port)

【パラメータ】

ID	devid	デバイスの ID 番号
volatile struct t_uart *	uart_port	UART のデバイスアドレス

【解説】

UART 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用します。

【プログラム例】

UART のチャネル 0 を ID_UART0 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 "CycloneV_uC3.h"
#include "DDR_COM.h"

extern ER   _ddr_cyclonev_uart_init(ID, volatile struct t_uart *);
        :
        _ddr_cyclonev_uart_init(ID_UART0, &REG_UART0);
```

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_CYCLONEV_UART_cfg.h に記述します。チャネルごとに 9 のパラメータがあります。複数のチャネルの設定を記述することができます。

```
#define   PCLK_UART0   100000000UL        /* 入力クロック */
#define   UART_n       /* チャネル n を使用する */
#define   TXBUF_SZn    256          /* 送信バッファサイズ */
#define   RXBUF_SZn    256          /* 受信バッファサイズ */
#define   XOFF_SZn     192          /* XOFF 送出受信バッファデータ数トリガ */
#define   XON_SZn      64           /* XON 送出受信バッファデータ数トリガ */
#define   RTRG_n       14           /* 受信 FIFO データ数トリガ */
```

```
#define   TTRG_n        2       /* 送信 FIFO データ数トリガ */
#define   IPL_UART_n    224     /* プリエンプト割込みレベル */
#define   FIFO_n        1       /* FIFO イネーブル */
```

５．６　Toshiba TZ2000/TZ2100 のドライバ

５．６．１　I/O アドレスと割込み番号の定義

TZ2000/TZ2100 の I/O アドレスと割込み番号の定義は tz2000_uC3.h/ tz2100_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。

５．６．２　割込みドライバ DDR_TZ2000_GIC.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 16 刻みで 0（最高）～240（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。

本ドライバは TZ2000 と TZ2100 の両方で使用できます。ただし、本ドライバを TZ2100 で使用する場合は ID の 0～31 の割り込み、つまり Cortex-A9 MPCore のグローバルタイマやプライベートタイマなどの割り込みは使用できない制限があります。

_ddr_gic_init	割込みコントローラの初期化

【書式】

```
void   _ddr_gic_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。また、ファイルの DDR_CA_GIC_cfg.h に設定例のように初期化内容を記述してください。

【プログラム例】

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

【設定例】

TZ2100 で使用する場合はファイルの tz2100_uC3.h をインクルードしてください。TZ2000 では tz2000_uC3.h のインクルードは不要です。

```
#include "uC3mcext.h"
#include "tz2100_uC3.h"      /* TZ2100 の場合のみ追加 */

#define IPL_MASK      0xF0      /* Priority mask */
```

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。

５．６．３　周期タイマドライバ DDR_TZ2000_PTIMER.c

本ドライバは TZ2000 用です。

_ddr_private_timer_init	周期タイマの初期化

【書式】

void　_ddr_private_timer_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 PTIMER（Private Timer）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_CA_PTIMER_cfg.h に記述します。

【設定例】

```
#define IPL_PTIM    224       /* 周期タイマ割込みの優先度 */
```

５．６．４　周期タイマドライバ DDR_TZ2100_TMR.c

本ドライバは TZ2100 用です。割り込みドライバの仕様により TZ2100 では Cortex-A9 MPCore のプライベートタイマを使用することはできません。よって、TZ2100 では周辺のタイマである TMR0 を使用します。

なお、デバッガにマイクロ秒のシステム時刻を提供する関数の_kernel_micro_systim は使用できません。互換性のため関数を実装していますが正しい時刻を返しません。この制限はタイマの TMR0 がカウンタの値を読み出せない仕様であるために適用されます。なお、この制限は OS の動作には影響はありません。デバッガの動作のみに対する制限になります。

_ddr_private_timer_init　　周期タイマの初期化

【書式】

void　_ddr_private_timer_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周辺の TMR0 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_TZ2100_TMR_cfg.h に記述します。

【設定例】

```
#include "tz2100_uC3.h"

#define CFG_TIM_CH      0           /* TMR0 のチャネル指定 (0 または 1) */
#define CFG_TIM_CLK     24000000    /*  タイマのクロックソース(Hz) */
#define CFG_TIM_IPL     128         /*  割り込み優先度  */
```

５．６．５　電源管理ユニットドライバ DDR_TZ2000_PMU.c

本ドライバは TZ2000 用です。電源管理ユニット（Power Management Unit）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_TZ2000_PMU_cfg.h に記述します。

【設定例】

```
#define MAINCLK             25000000        /* クロックソースの周波数 */
```

_ddr_tz2000_pmu_getclk　　出力周波数の取得

【書式】

void　_ddr_tz2000_pmu_getclk(T_PMUFRQ *frq)

【パラメータ】

T_PMUFRQ *　frq		各種周波数情報を入れたパケットへのポインタ
frq の内容（T_PMUFRQ 型）		
UW	ATB	CoreSight の ATB に供給されるクロック
UW	DAPB	CoreSight の APB に供給されるクロック
UW	CORE	Cortex-A9 コアに供給されるクロック
UW	PERIPH	Cortex-A9 MPCore の周辺に供給されるクロック

【解説】

分周器により各モジュールへ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_TZ2000_PMU_CPG.h"

    T_PMUFRQ frq;

    _ddr_tz2000_pmu_getclk(&frq);
```

_ddr_tz2000_pmu_chg_clk　　分周器による出力周波数の変更

【書式】

void　_ddr_tz2000_pmu_chg_clk(UINT clk_sel)

【パラメータ】

UINT	clk_sel	周波数組み合わせのケース番号

【エラーコード】

E_PAR	パラメータエラー（不正なケース番号）

【解説】

ケース番号と分周比の関係は以下のようになり、これらを動的に変更することができます。

ケース番号	CORE	PERIPH	ATB	DAPB
CLOCK_CASE_1	1/1	1/16	1/8	1/8
CLOCK_CASE_2	1/2	1/16	1/16	1/16
CLOCK_CASE_3	1/4	1/16	1/16	1/16
CLOCK_CASE_4	1/8	1/16	1/16	1/16

【プログラム例】

コアクロックを 1/4 に変更する CLOCK_CASE_3 の場合は、以下のコードになります。

```
#include "DDR_TZ2000_PMU.h"

    _ddr_tz2000_pmu_chg_clk(CLOCK_CASE_3);
```

５．６．６　標準 COM ドライバ DDR_TZ2000_UART.c

本ドライバは TZ2000 と TZ2100 の両方で使用できます。ただし、TZ2100 ではクロックソースを指定するマクロの CLK_UART_n をファイルの DDR_TZ2000_UART_cfg.h に記述してください。

_ddr_tz2000_uart_init　　UART ポートの初期化

【書式】

ER　_ddr_tz2000_uart_init(ID devid, volatile struct t_uart *uart_port)

【パラメータ】

ID	devid	デバイスの ID 番号
volatile struct t_uart *	uart_port	UART のデバイスアドレス

【解説】

UART 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用します。

【プログラム例】

UART のチャネル 1 を ID_UART1 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 "tz2000_uC3.h"
#include "DDR_COM.h"

extern ER _ddr_tz2000_uart_init(ID devid, volatile struct t_uart *uart_port);
        :
        _ddr_tz2000_uart_init (ID_UART1, &REG_UART1);
```

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_TZ2000_UART_cfg.h に記述します。チャネルごとに 9 つ(TZ2100 では 10)のパラメータがあります。複数のチャネルの設定を記述することができます。

```
#define   UART_n          /* チャネル n 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           14       /*  受信 FIFO データ数トリガ（1,4,8,1）  */
#define   TTRG_n           2        /*  送信 FIFO データ数トリガ（0,2,4,8）  */
#define   FIFO_n           1        /* FIFO イネーブル（1)／ディセーブル(0)  */
#define   IPL_UARTn        224      /*  割込みレベル */
#define   CLK_UART_n       24000000 /*  クロックソース(Hz) [TZ2100 のみ] */
```

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-A9 編

2013年　1月	第１版
2013年　4月	第２版
2013年　7月	第３版
2013年　8月	第４版
2013年　10月	第５版
2013年　10月	第６版
2014年　1月	第７版
2014年　6月	第８版
2014年　11月	第９版
2015年　8月	第10版
2016年　3月	第11版
2016年　3月	第12版
2017年　3月	第13版
2017年　3月	第14版
2017年　8月18日	第15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３Ｆ
TEL 03-5614-6918　FAX 03-5614-6919
お問い合わせ info@eforce.co.jp