# リアルタイムカーネル勉強会用資料

## 開発環境＋etc

**神谷和孝**

**天城技研株式会社**

kzkamiya@amtec.co.jp

## 担当：開発環境＋etc

インラインアセンブラ
リンカスクリプト
スタートアップモジュール
関連ファイル
./config/sh3/shelf.ld　ビッグエンディアン用リンカスクリプト
./config/sh3/shlelf.ld　リトルエンディアン用リンカスクリプト
./config/sh3/start.S スタートアップモジュール

## 資料

TOPPERS/JSP jsp-1.1.1.lzh
http://www.ertl.ics.tut.ac.jp/TOPPERS/ よりダウンロード



# 1 開発環境

## 1.1 開発環境使用について

TOPPERS/JSP Kernel
Toyohashi Open Platform for Embedded Real-Time Systems/
Just Standard Profile Kernel

# 1.2 SH3 ターゲット依存部 ./jsp/doc/sh3.txt

# SH3 ターゲット依存部の概要

## ターゲットシステムと開発環境

SH3 ロセッサのターゲットシステムとしては、それぞれ SH7708, SH7709, SH7709A を搭載した以下のボードをサポートしている。

・(株)EPSON 製の CARD-E09A ボード(CPU: SH7709A)
・三菱電機マイコン機器ソフトウエア(株)製の MU200-RSH3 ボード(CPU: SH7709)
・(株)電産製の DVE-SH7700 ボード (CPU: SH7708)
・(株)日立超 LSI システムズ製の MS7709ASE01 ボード (CPU: SH7709A)

開発環境には、GCC などの GNU 開発環境を用い、オブジェクトファイルフォーマットは ELF を標準とする。

## サポートする機能の概要

SH3 依存の機能として、割込みマスクの変更・参照(chg_ixx、get_ixx)と、性能評価用システム時刻参照機能(vxget_tim)をサポートしている。割込みの禁止と許可(dis_int、ena_int)はサポートしていない。

## 他のターゲットへのポーティング

現バージョンでは、SH7708 とその拡張版の SH7709,SH7709A をサポートしている。割り込み時に割り込み要因がセットされるレジスタは、SH7708 では INVET だが、SH7709/A では追加された内蔵モジュールの割り込み要因は INVET2 に設定される。そのため、この部分をコンパイルプラグにより切り替えている。

他の SH 系列のプロセッサへのポーティングについては、SH3 と機構が似た SH4 については比較的容易にポーティング可能と思われるが、ベクタ形式の例外機能を持つ SH1,SH2 については面倒であると予想される。他のターゲットについては、SH3 が多くの内蔵モジュール持っていることもあり、内蔵モジュールを使うならば、容易に行えると思われる。



## GDB スタブ

TOPPERS/JSP カーネルは GDB スタブと共に使用することを前提にしている。そのため、sys_putc はスタブ呼び出しで実現し、割り込みハンドラはスタブ経由で呼ばれる。スタブを使用せず ROM や CF にカーネルを置き実行するためには、直接シリアルポートへの出力や、VBR の設定が必要になる。そこで、WITH_STUB というコンパイルフラグによりこれらの機能を切り替える。だたし、ROM 化の際に必要なバスコントローラの設定は記述していないため、初期化ルーチンに記述する必要がある。それぞれのターゲットで WITH_STUB のコンパイルフラグをとった場合必要な作業を下に示す。初期化ルーチンはスタブでの記述が参考になると思われる。

・MS7709ASE01
　　hardware_init_hook にバスステートコントローラ等の初期化を追加。
　　sys_initialize に SCIF の初期化を追加。

## シリアルポート

CARD-E09A, MU200-RSH3, MS7709ASE01 は、スタブによる GDB との通信と、カーネルのログ出力用に 2 つのシリアルポート使用する。全てのポートにおいて データ: 8bit,Parity:none,Stop:1bit である。

| | GDB 通信用 | カーネルログ用 |
|---|---|---|
| ・MS7709ASE01 | SCIF(CN2):115200bps | スーパーI/O 内蔵 SCI(CN3):19200bps |

# 1.3 SH3 プロセッサ依存部の機能

## データ型

int 型および unsigned int 型のサイズは 32 ビットである。

## 割込み管理機能と割込みハンドラ

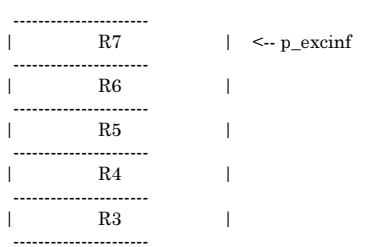
カーネル管理外の割込みとしては、 NMI がある。よって、CPU ロック状態や初期化ルーチン内では、NMI 以外の割込みはすべて禁止されている。具体的には、IPM(Interrupt Priority Mask)が 15 に設定される。しかしながら、ボード上に NMI 入力を持たないターゲットのため、GDB のスタブがホストのマシンと通信するためのシリアルポートの優先度を 15 で登録し、GDB で Ctrl-C を押すと 割り込みが入りスタブに制御が移るようにしてある。この機能を使用す

るためには CPU ロック状態では、優先度が 14 になるようにしなければならない。そのため、CPU ロックで設定される優先度をマクロで MAX_IPM で指定している。MAX_IPM は cpu_config.h の中で #define されている。スタブを使う場合は 14 にスタブを使わない場合は 15 に設定している。
DEF_INH で指定する割込みハンドラ番号(inhno)は、SH7708 では、割り込み事象レジスタ(INTEVT)に設定されるコードであり、SH7709/A では、割り込み事象レジスタ 2(INTEVT2)に設定されるコードである。データ型(INHNO)は unsigned int 型に定義されている。DEF_INH で、INTEVT,INTEVT2 コードとして有効でない値や、外部割込みに対応しない番号を指定した場合の動作は保証されない。

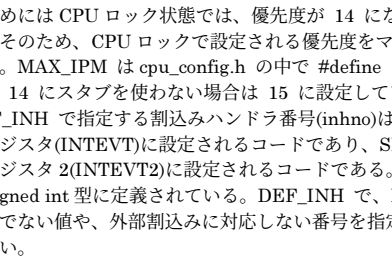
SH3 依存の機能として、SR(Status Register)中の 割り込みマスクビット(I3〜I0)の値を変更するためのサービスコール chg_ipm と、参照するためのサービスコール get_ipm をサポートしている。なお、割り込みマスクビットの値を表すデータ型 IPM は、unsigned int 型に定義されている。
これらのサービスコールは、タスクコンテキストで CPU ロック解除状態の場合にのみ呼び出すことができる。chg_ipm により IPM を 0 以外(すなわち、何らかの割込みが禁止されている状態)にした場合でも、ディスパッチは禁止され、chg_ipm により変更した IPM の値は、ディスパッチ後のタスクに引き継がれる。例えば、あるタスクで IPM を 1 に変更した後、何らかの割込みにより別のタスクに切り替わると、切り替わった後のタスクでも IPM は 1 になる。
chg_ipm をサポートするために、割込みハンドラの出入口処理などにオーバヘッドを生じている。そこで、SUPPORT_CHG_IPM というマクロにより、これらのサービスコールをサポートするかどうかを切り替えられるようにしている。
SUPPORT_CHG_IPM は、cpu_config.h の中で #define されている。

## CPU 例外管理機能と CPU 例外ハンドラ

DEF_EXC で指定する割込みハンドラ番号(excno)は、SH3 での 例外事象レジスタに設定される例外コード番号を表し、そのデータ型(EXCNO)は unsigned int 型に定義されている。DEF_EXC で、例外コード番号として有効でない値や、CPU 例外に対応しない番号を指定した場合の動作は保証されない。
CPU 例外ハンドラに渡される p_excinf は，CPU 例外発生時のコンテキストを保存したスタックへのポインタが渡される．スタックの構造を以下に示す．






また，CPU 例外発生時の PC の値はインクリメントされる．そのため，CPU 例外ハンドラから復帰を行うと，CPU 例外発生した命令の次の命令から実行を再開する．

## レジスタバンク

例外ハンドラの入口を除き基本的にレジスタバンク 1 を使用している。レジスタバンク 1 は r7 のみ例外/割り込みのネスト回数のカウント用に使用している。非タスクコンテキストとタスクコンテキストの判別はこのレジスタにより行っている。なお、スタブはレジスタバンク 0 の r0 を使用するため、このレジスタに書き込んだデータは書き変る恐れがある。

## MACH と MACL

MACH と MACL については，gcc に-mhitachi オプションをつけると，関数で MACH と MACL を使用する場合，スタックに保存してから使用し，関数を抜けると元に戻すため，割込みでは保存していない．また，自らディスパッチャを呼び出し，ディスパッチする場合は，関数呼び出しになるため，この呼び出しにまたがって，これらのレジスタを使うことはないため保存していない．そのためアセンブラのコードを使用する場合もこのルールに従う必要がある．なお，割込みからのディスパッチ(受動的ディスパッチ)では保存している．

## GBR

gcc は GBR を使用しないため，割込みハンドラの入口では保存していない．割込みハンドラ内で GBR を使う場合はアプリケーション側で GBR の待避/保存を行う必要がある．また，上記の MACH と MACL と同じ理由により，自らディスパッチャを呼び出す場合にも保存していない．なお，割込みからのディスパッチ(受動的ディスパッチ)では保存している．

## 1.4 メモリマップ

・MS7709ASE01
依存部では、コード領域を 0x0c001000 ～ 0x0c0fffff 約 1MB、データ領域を 0x0c100000 ～ の約 3MB，非タスクコンテキスト用のスタック領域を ～ 0x0c3fffff に確保している。0x0c000000 ～ 0x0c000fff は、ROM モニタまたはスタブのワークエリアとなっており、使用することができない.

## 1.5 ディレクトリ・ファイル構成

SH3 ターゲット依存部の各ファイルの概要は次の通り.

```
config/sh3/
        Makefile.config     Makefile の SH3 依存定義
        cpu_defs.h              プロセッサ依存部のアプリケーション用定義
        cpu_config.h        プロセッサ依存部の構成定義
        cpu_config.c        プロセッサ依存部の関数
        cpu_support.S       プロセッサ依存部のサブルーチン
        cpu_context.h       コンテキスト操作
        makeoffset.c        offset.h 生成サポートプログラム
        cpu_insn.h              低レベルのプロセッサ操作ルーチン
        start.S                 スタートアップモジュール
        shelf.ld        ビッグエンディアン用リンカスクリプト
        shlelf.ld         リトルエンディアン用リンカスクリプト
        hw_timer.h        タイマ操作ルーチン
    sh3.h               SH3 の定義

    config/sh3/ms7709ase01
        Makefile.config     Makefile の MS7709ASE01 依存定義
        hw_serial.h             シリアルインタフェースデバイス操作ルーチン
        ms7709ase01.h       MS7709ASE01 ハードウェア定義
        sys_config.c        システム依存部の関数
        sys_config.h        システム依存部の構成定義
        sys_defs.h              システム依存部のアプリケーション用定義
        sys_support.S       システム依存部のサブルーチン
```



# 2 スタートアップモジュール、リンカスクリプト

## 2.1 リンカスクリプトについて

リンカスクリプト (Linker Scripts)
ld コマンド・ランゲージ (ld command language) はステートメントの集まりで、いくつかは、ある種のオプションをセットする単純なキーワードとなる。また、いくつかは入力ファイルを選択して、グループ化するためや、出力ファイルの名称付けに使用される。
ld コマンド・ランゲージの最も基本的なコマンドは SECTIONS コマンドで、出力ファイルの詳細なレイアウトを指定する。
今回のターゲットシステムのリンカスクリプトには、
ビッグエンディアン用リンカスクリプト./config/sh3/shelf.ld と
リトルエンディアン用リンカスクリプト./config/sh3/shlelf.ld
があり、 それらの差分は、OUTPUT_FORMAT のみとなっている。

```
<! OUTPUT_FORMAT("elf32-shl","elf32-shl","elf32-shl")
 !> OUTPUT_FORMAT("elf32-sh","elf32-sh","elf32-sh")
```

## 2.2 スタートアップモジュールについて

SH3 依存のスタートアップモジュール(start.S)では、次の初期化処理を行う。
(A) プロセッサモードの初期化とスタックポインタの初期化
最初に、すべてのキャッシュを無効化した後、キャッシュをライトスルーモードに設定し、有効にする。また、プロセッサのモードを、特権モード、レジスタバンク 1 に設定し、割り込みマスクを MAX_IPM に設定する。
次に、スタックポインタ(r15)を STACKTOP に設定する。ここで割込みスタックポインタに設定されたスタック領域は、カーネル起動後は非タスクコンテキスト用のスタック領域として使われる。STACKTOP は、sys_config.h 部で定義することを想定している。

(B) hardware_init_hook の呼出し
hardware_init_hook が 0 でない場合には、hardware_init_hook を呼び出す。
hardware_init_hook は、カーネルが起動される前に行う必要があるターゲット依存の初期化を行うために用意している。hardware_init_hook がどこでも定義されていない場合、リンカでこのシンボルを 0 に定義する(リンカスクリプト内に記述あり)。

(C) bss セクションと data セクションの初期化
bss セクションをゼロクリアする。また、data セクションを初期化する。

(D) software_init_hook の呼出し
software_init_hook が 0 でない場合には、software_init_hook を呼び出す。software_init_hook は、カーネルが起動される前に行う必要があるソフトウェア環境(具体的には、ライブラリ)依存の初期化を行うために用意している。software_init_hook がどこでも定義されていない場合、リンカでこのシンボルを 0 に定義する(リンカスクリプト内に記述あり)。

(E) カーネルの起動
kernel_start へ分岐し、カーネルを起動する。kernel_start からリターンしてくることは想定していない。

## 2.3 ビッグエンディアン用リンカスクリプト./config/sh3/shelf.ld

```
/*
 *   @(#) $Id: shelf.ld,v 1.1 2000/11/14 16:29:53 honda Exp $
 */
OUTPUT_FORMAT("elf32-sh","elf32-sh","elf32-sh")
OUTPUT_ARCH(sh)
PROVIDE(_hardware_init_hook = 0);
PROVIDE(_software_init_hook = 0);

SECTIONS
{
      .text :
      {
            __text = . ;
            *(.text)

      }
      _etext = .        ;
      PROVIDE (etext = .)     ;
      .rodata : { *(.rodata) }
      . = ALIGN(4);
      __idata_start = . ;
      .data     :   AT(__idata_start)
      {
            __data_start = . ;
            *(.data)
      }
      __idata_end = __idata_start + SIZEOF(.data);
      _edata    =    . ;
      PROVIDE (edata = .);
      . = ALIGN(4);
      __bss_start = .               ;
      .bss            :
      {

         *(.bss)
```



```
         *(COMMON)
      }
      _end = .                      ;
      PROVIDE (end = .)       ;
      .comment 0 : { *(.comment) }
      .debug                  0 : { *(.debug) }
      .line                   0 : { *(.line) }
      .debug_srcinfo    0 : { *(.debug_srcinfo) }
      .debug_sfnames    0 : { *(.debug_sfnames) }
      .debug_aranges    0 : { *(.debug_aranges) }
      .debug_pubnames 0 : { *(.debug_pubnames) }
}
```

## 2.4 スタートアップモジュール ./config/sh3/start.S

```
#define _MACRO_ONLY
#include "sys_config.h"
#include "cpu_config.h"

#define CCR                 0xffffffec        /* CCR のアドレス */
#define CCR_DISABLE     0x00000008        /* キャッシュの無効化 */

/*
 *    SH3 用スタートアップモジュール
 *    カーネルはアプリケーションとリンクして使用する
 *    だけなので JSP カーネルのみ使用する。
 */
            .global    _start
            .align 2
_start:
            /*
             *   キャッシュの初期化
             */
            mov.l _ccr_addr,r1
            mov.l _ccr_disable,r2
            mov.l r2,@r1
            mov.l _ccr_mode,r2
            mov.l r2,@r1

            /*
             *   SR を初期化する。
             *   MD=1, RB=0, BL=0, I3～I0=e f は stub のみ
             */
            mov.l _init_sr,r0
            ldc     r0,sr
            mov     #0x01,r0
            ldc     r0,r7_bank
```

```
        /*
         * タスク独立のスタックを STACKTOP に初期化する
         * STACKTOP は、sys_config.h で定義
         */
        mov.l   _stack_k, r15

        /*
         *   _hardware_init_hook を呼び出す。(0 でない場合)
         *
         *   ハードウェア依存に必要な初期化処理がある場合は,
         *   hardware_init_hook という関数を用意すればよい.
         *   具体的には ROM 化の際、RAM を使用可能にするための
         *   バスコントローラの初期化等を行う。
         *   sys_support.S 内で hardware_init_hook を定義してい
         *   る.
         */
start_0:
        mov.l    _hardware_init_k, r0
        tst      r0,r0
        bt       start_1
        jsr      @r0
        or       r0,r0

        /*
         * bss セクションをクリア
         */
start_1:
        mov.l    _bss_start_k,r0
        mov.l    _end_k,r1
        cmp/eq   r0,r1
        bt       start_3
        mov      #0,r2
start_2:
        mov.l    r2,@r0
        add      #4,r0
        cmp/hi   r0,r1
        bt       start_2

        /*
         *   data セクションを初期化する（ROM 化対応）.
         *
         *   __idata_start から __idata_end までを, __data_start 以降に
         *   コピーする.
         */
start_3:
        mov.l    __idata_start_k,r1
        mov.l    __idata_end_k,r2
        cmp/eq   r1,r2
        bt       start_5
        mov.l    __data_start_k,r0
start_4:
        mov.l    @r1+,r4
        mov.l    r4,@r0
```



```
        cmp/hi   r1,r2
        add      #4,r0
        bt       start_4

        /*
         *   software_init_hook を呼び出す（0 でない場合）.
         *
         *   ソフトウェア環境（特にライブラリ）に依存して必要な初期化処
         *   理がある場合は, software_init_hook という関数を用意すれば
         *   よい.
         */
start_5:
        mov.l   _software_init_hook_k, r0
        tst     r0,r0
        bt      start_6
        jsr     @r0
        or      r0,r0

start_6:
        /*
         *   カーネルを起動する
         */
        ! call the mainline
        mov.l   _kernel_start_k,r0
        jsr     @r0
        or      r0,r0

        .align 4
_ccr_addr:
        .long CCR
_ccr_disable:
        .long CCR_DISABLE
_ccr_mode:
        .long CCR_MODE
_init_sr:
        .long   0x40000000 + MAX_IPM << 4
_stack_k:
        .long    STACKTOP
_bss_start_k:
        .long    __bss_start
_end_k:
        .long    _end
__idata_start_k:
        .long    __idata_start
__idata_end_k:
        .long    __idata_end
__data_start_k:
        .long    __data_start
_kernel_start_k:
        .long    _kernel_start
_hardware_init_k:
        .long    _hardware_init_hook
_software_init_hook_k:
        .long    _software_init_hook
```