

TOPPERS/JSP カーネル ユーザズマニュアル

(Blackfin ターゲット依存部)

(カーネル 1.4.3, ターゲット [3.2.03.1.1](#))

最終更新: [10-Jul-2010](#)~~[16-Aug-2009](#)~~

TOPPERS/JSP Kernel

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

Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory Toyohashi Univ.
of Technology, JAPAN

Copyright(C) 2006-2009 by Takemasa Nakamura

上記著作権者は、以下の(1)~(4)の条件か、Free Software Foundation によって公表されている GNU General Public License の Version 2 に記述されている条件を満たす場合に限り、本ソフトウェア（本ソフトウェアを改変したものを含む、以下同じ）を使用・複製・改変・再配布（以下、利用と呼ぶ）することを無償で許諾する。

1. 本ソフトウェアをソースコードの形で利用する場合には、上記の著作権表示、この利用条件および下記の無保証規定が、そのままの形でソースコード中に含まれていること。
2. 本ソフトウェアを、ライブラリ形式など、他のソフトウェア開発に使用できる形で再配布する場合には、再配布に伴うドキュメント（利用者マニュアルなど）に、上記の著作権表示、この利用条件および下記の無保証規定を掲載すること。
3. 本ソフトウェアを、機器に組み込むなど、他のソフトウェア開発に使用できない形で再配布する場合には、次のいずれかの条件を満たすこと。
 - A) 再配布に伴うドキュメント（利用者マニュアルなど）に、上記の著作権表示、この利用条件および下記の無保証規定を掲載すること。
 - B) 再配布の形態を、別に定める方法によって、TOPPERS プロジェクトに報告すること。
4. 本ソフトウェアの利用により直接的または間接的に生じるいかなる損害からも、上記著作権者および TOPPERS プロジェクトを免責すること。

本ソフトウェアは、無保証で提供されているものである。上記著作権者および TOPPERS プロジェクトは、本ソフトウェアに関して、その適用可能性も含めて、いかなる保証も行わない。また、本ソフトウェアの利用により直接的または間接的に生じたいかなる損害に関しても、その責任を負わない。

1. Blackfin ターゲット依存部の概要

1.1. ターゲットシステム

Blackfin プロセッサのターゲットシステムとしては、Analog Devices の EZ-KIT Lite BF533、EZ-KIT Lite BF537、[EZ-KIT Lite BF518](#)、[E!kit BF533](#)、[KoanzameBF533CB](#) に対応している。[中村 \(酔漢\) による RebutBF533CB](#) は、[このリリースから参考実装扱いとなった次のリリースで廃止する。代わりに Kobanzame 依存部を使用されたい。](#)

1.2. 開発環境と実行環境

開発には、GCC などの GNU 開発環境を用いた。libkernel ライブラリのビルドには Analog Devices の VisualDSP++ も利用できることを確認しているが、試験的な対応にとどめている。なお、コメントの日本語エンコーディングには UTF-8 を使用している。

1.3. サポートする機能の概要

Blackfin 依存の機能として、SIC 割込みマスクの参照 (`get_ims`) および割り込みの許可 (`ena_int`)、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。`chg_ims` および `dis_int` には対応していない。詳しくは 3.6 節「割り込みマスク制御」を参照のこと。

なお、VisualDSP++ を使用する場合、ライブラリに関してはマルチ・タスク対応にしていないので注意すること。具体的には `malloc()`/`free()` 関数と `errno` 変数がマルチ・タスク対応になっていない。特に `malloc()` に関しては [sourceforge](#) において相談されたい。

1.4. 他のターゲットへのポーティング

現バージョンでは、ADSP-BF532、ADSP-BF533 [および](#) ADSP-BF537、[ADSP-BF518のみをサポートに対応している。](#)

1.5. Blackfin 依存部のソースの取得

Blackfin 依存部のソースコードは [sourceforge](#) より取得できる。

- <http://sourceforge.jp/projects/toppers.jsp4bf/>

最新ソースコードは CVS から取得できる。ただし、複数ある CVS のモジュールのうち、Blackfin 依存部 3.x が格納されているのは “jsp” モジュールである。“blackfin-vdsp” モジュールには 1.x が、“blackfin” モジュールには 2.x が格納されている。1.x、2.x 系列はすでにメンテナンスしていないので注意すること。

なお、CVSクライアントの設定は以下の通り：

- ホストアドレス cvs.sourceforge.jp
- リポジトリ・パス /cvsroot/toppers.jsp4bf
- ユーザー名 anonymous
- 接続タイプ pserver

2. Blackfin プロセッサ依存部の機能

この節では、カーネルおよびシステムサービスの機能の中で、Blackfin 依存の部分について解説する。

2.1. データ型

signed int 型, unsigned int 型, size_t 型のサイズは、いずれも 32 ビットである。

2.2. 割り込み管理機能

デフォルトではカーネル管理外の割り込みは NMI のみである。よって、CPU ロック状態や初期化ルーチン内では、NMI 以外の割り込みはすべて禁止されている。具体的には、IMASK が 0xC01F に設定される。

任意のハードウェア割り込みをカーネル管理外割り込みにすることもできる。sys_config.h でマクロ UNMANAGED_INT を宣言すると、マクロ中で指定したビットマップに対応した割り込みがカーネル管理外割り込みとなる。この機能を使用するとわずかだが性能が劣化する。宣言についての詳細は sys_config.h のコメントを参照。

なお、ハードウェア・エラー割り込みハンドラを使うときには、sys_config.h で QUICK_HW_ERROR マクロを宣言する。このマクロを宣言しなくともハードウェア・エラー割り込みを処理することは出来る。ただし、dispatch() 関数内で割り込み待ちを行っている場合、ハードウェア・エラーが発生しても即時対応できなくなる。この点に問題がなければ、QUICK_HW_ERROR マクロを宣言する必要はない。

カーネル管理外割り込みを利用する場合、その割り込み用のスタックもユーザーが用意し、イベントのエントリで設定しなければならない。また、カーネル管理外割り込みの最中であっても、CPU 例外は発生する。CPU 例外はこの場合カーネル管理外割り込みのスタックをそのまま使用するので必要量は慎重に計算しなければならない。

IMASK レジスタの、カーネル管理外割り込みに対応するビットは、kernel_start ルーチンの開始時点で 0 になっている。このビットのオン/オフはアプリケーション・プログラムが行うこと。

デフォルトの例外ハンドラは、スプリアス割り込みハンドラであり、回復不能な割り込みが発生したと考えてコンソールに割り込み発生箇所やレジスタ情報を出力する。コンソール上に“?”が表示されたら、“!”と入力することでダンプ情報を取得できる。

2.3. 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、単に無視される。例外ハンドラはただひとつだけが登録可能で、seqstat の excause による処理の振り分けは登録した例外ハンドラ内部で行う。

デフォルトの例外ハンドラは、スプリアス割込みハンドラであり、回復不能な割込みが発生したと考えてコンソールに割込み発生箇所やレジスタ情報を出力する。コンソール上に“?”が表示されたら、“!”と入力することでダンプ情報を取得できる。

2.4. スタートアップモジュール

Blackfin 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。

2.4.1. 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 LC0 と LC1 をクリアしてハードウェア・ループを無効化した後、L0-L3 を 0 にする。次にスタックをイベントスタックのトップに設定する。このスタックは dispatch() がタスク・スタックを設定するまで非タスクコンテキスト用のスタックとして使われる。プロセッサの割り込みベクトルに割り込みディスパッチャのエントリを設定する。最後にリセットモードから IVG15 モードに遷移する。

イベントスタックは LDF が定義した領域をリンカーが処理して埋め込む。

なお、GCC を使用した場合には、必要に応じてソフトウェア・リセットを行う関数 boot_for_gdb(BOOL) を追加している。これは、2006 年時点で、gdbproxy がアプリケーションのロード時にプロセッサをリセットしないためである。

この関数は 引数大域変数 enable boot for gdb が真かつプロセッサがコールド・ブートした場合に限り、リポートを行う。デフォルトでは引数変数は常に偽であるため、必要に応じて真にして使う。

2.4.2. 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。

2.5. 性能評価用システム時刻参照機能

Blackfin 依存部では、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。性能評価用システム時刻の精度は 1 コア・サイクルである。この機能はコアのサイクルカウンタを直接読み出すため正確である。なお、SYSUTIM 型は UD 型 (64 ビットの符号無し整数型) に定義している。

2.6. 予約資源

いくつかの資源はカーネル内部で使用しているため、ユーザーアプリケーションが使用することは禁止されている。

- プロセッサ・コア・イベント番号 15 および 14

3. EZ-KIT Lite BF533 システム依存部の機能

3.1. PLL の設定

PLL の設定は「PLL_CTL レジスタが初期値で、かつ SDRAM コントローラがディセーブルの場合」に限り行う。これは、SDRAM のようなクロックに依存する外部デバイスの動作を保護するためである。

イニシャライザやローダーは SDRAM コントローラと PLL をあらかじめ初期化する場合がある。PLL の設定をバイパスするのはこのような事態に備えるためである。

VisualDSP++ のデバッグ環境では、この機能が邪魔になる場合がある。そのときには、`sys_config.h` で `FORCE_PLL_INITIALIZE` マクロを宣言すると良い。

3.2. システムクロックドライバ

システムクロックドライバが `isig_tim` を呼び出す周期は、`sys_defs.h` 中の `TIC_NUME` と `TIC_DENO` で定義されている（標準は 1 ミリ秒周期）。システムクロックのタイマーは汎用タイマー 2 を使用する。汎用タイマー 2 を使用する場合に必要な `SIC_IMASK` の変更は自動的におこなうのでユーザーが気にする必要はない。

`sys_config.h` のなかで `USE_TIC_CORE` マクロを定義するとコアタイマーを使う。ただし、この機能はシステム依存部の移植時の作業用に残しているだけであり、推奨しない。アプリケーション用としては無サポートである。

3.3. シリアルインタフェースドライバ

シリアル・インターフェース・ドライバはシステム依存部の `UART.*` にて実装している。それぞれの割り込みは `INTNO_UART_TX` および `INTNO_UART_RX` である。シリアルポートをオープンすると、これらの割り込みは自動的に有効になる。

3.4. 割り込みハンドラ

`DEF_INH` で指定する割り込みハンドラ番号 (`inhno`) は、ADSP-BF533 のシステム割り込み番号を表し、そのデータ型 (`INHNO`) は `unsigned int` 型に定義されている。`DEF_INH` に指定できる値は、`sys_config.h` に `INHNO_XXXX` として定義してある。なお、システム割り込み番号とは、`SIC_IMASK` におけるビット番号である。

`INHNO_RAISE` は `RAISE` 命令がおこなうソフトウェア割り込みのためのハンドラ番号である。`RAISE` 命令に与えるオペランドは割り込みの順位番号を与えるが、順位がどうであれ

INHNO_RAISEに登録されたユーザーのハンドラが呼び出される。RAISE 命令による割り込みを他から区別する手段がないため、RAISE を使う場合は他のデバイスからの割り込み要因が使用されていない順位を利用しなければならない。ただし、割り込み要因は実際に発生していなければ割り当てられていてもかまわない。

割り込みハンドラがシステム割り込み番号ごとに呼び出されるため、アプリケーション・プログラムが割り込みハンドラ内部で実際に割り込みを起こしたデバイスを同定する必要はない。ただし、割り込みのクリアはユーザーの割り込みハンドラの責任で行う。

3.5. SIC_IARx の変更

何らかの理由で内蔵デバイスからの割り込み要求順位を変更したいときには SIC_IARx を変更しなければならない。変更はアプリケーション初期化コードで行い、静的サービスコール ATT_INI を使ってシステムに登録する。変更用の初期化コードは次のようになる。

```
void init_iar(VP_INT vp_int)
{
    *pSIC_IAR0 = ...;
    *pSIC_IAR1 = ...;
    ...
    make_priority_mask();
}
```

make_priority_mask は SIC_IARx から TOPPERS/JSP が内部で使用する参照ビットマップを作り出す。この初期化コードは、コンフィグレーションファイルの中で他の初期化コードより先に宣言されなければならない。

3.6. 割り込みマスク制御

ADSP-BF533 依存の機能として、SIC_IMASK 中のマスク値を参照するためのサービスコール get_ims をサポートしている。なお、マスクの値を表すデータ型 IMS は、unsigned int 型に定義されている。

このサービスコールは、タスク・非タスクコンテキストのいずれでも、また CPU ロック状態、ロック解除状態のいずれでも呼び出すことができる。

chg_ims() および ena_int() は実装していない。これは Blackfin プロセッサのアーキテクチャでは、アプリケーション実行中に SIC_IMASK を変えることが危険であることによる。

Blackfin コアは IMASK による割り込み禁止 (CPU ロック状態に相当) であっても ILAT レジスタに割り込み要求を記録する。したがって、CPU ロック状態に入ってから SIC_IMASK の何らかの

ビットをクリアする前までに割り込みが生じる可能性がある。この問題のため、chg_ims および dis_int()を安全に実装することはできない。

SIC_IMASK の設定は割り込みが有効になる前に設定すべきであり、その後変更をするとしても特定割り込みの許可だけに制限しなければならない。

Blackfin 依存の割り込みマスクの変更・参照のためのサービスコールの仕様は次の通り。

3.6.1. get_ims

割り込みマスクの参照

【C 言語 API】

```
ER ercd = get_ims(IMS *p_ims);
```

【パラータ】

なし

【リターンパラメータ】

ER ercd	エラーコード
IMS ims	現在の SIC_IMASK の値

【エラーコード】

E_CTX	コンテキストエラー
-------	-----------

【機能】

現在の SIC_IMASK の値を読み出し、ims に返す。

このサービスコールは常に 0 を返す。

【C 言語 API】

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

【パラメータ】

INTNO intno 許可するシステム割り込みの番号

【リターンパラメータ】

ER ercd エラーコード

【エラーコード】

E_PAR パラメータエラー (intno が不正)

【機能】

SIC_IMASK を intno ビットを 1 にする。対応する割り込みが許可される。
パラメータとして渡すことのできる値は sys_ config.h に INTNO_XXXX と
して宣言してある。

許されない番号を渡すと E_PAR を返す。

3.7. gdb の為のレジスタ宣言

Blackfin アーキテクチャでは、ペリフェラル・レジスタをシステム MMR、コア MMR としてメモリに配置している。これらのレジスタは VisualDSP++ デバッグ環境ではウィンドウを通して読むことができるが、gdb にはその機能はない。そのため、gcc 環境でビルドする際には mmrXXXX という変数を用意してある。ここで XXXX はシステム MMR 名あるいはコア MMR 名である。たとえば、mmrSIC_IMASK、mmrIMASK といった変数がある。これらの変数はレジスタと同じアドレスにあり、レジスタと同じ長さを持つので、デバッグ時に便利である。

4. その他のシステム依存部の機能

EZ-KIT Lite BF537、BF533CB、Rebun システム依存部は、以下に記述する部分をのぞいて EZ-KIT Lite BF533 システム依存部と同等の機能を実装している。詳細については、EZ-KIT Lite BF533 システム依存部の章を参照のこと。

4.1. システム依存部間の差異

各システム依存部の細かい差異については、表 1 を参照のこと。

表 1 システム依存部間の差異

依存部	ezkit bf533	kobanzame	ekit bf533	ezkitbf 537	ezkit bf518	ezkit bf518 00
プロセッサ	ADSP-BF533	ADSP-BF533	ADSP-BF533	ADSP-BF537	ADSP-BF518	ADSP-BF518 rev 0.0
CLKIN	27MHz	20MHz	12MHz	25MHz	25MHz	25MHz
CCLK	594MHz	500MHz	600MHz	600MHz	400MHz	400MHz
SCLK	118.8MHz	125MHz	120MHz	120MHz	80MHz	80MHz
MEM INST	64kB(*1)	64kB(*1)	64kB(*1)	48kB(*1)	32kB(*1)	32kB(*1)
MEM DATA A	32kB	32kB	32kB	32kB	32kB	32kB
MEM DATA B	16kB(*1)	16kB(*1)	16kB(*1)	16kB(*1)	16kB(*1)	16kB(*1)
MEM SCRATCH	4kB	4kB	4kB	4kB	4kB	4kB
コンソール入出力	UART	UART	UART	UART0	UART0	UART0
システムタイマー	GP Timer 2	GP Timer 2	GP Timer 2	GP Timer 7	GP Timer 7	GP Timer 7

(*1) 16kB はキャッシュ用に予約

(*2) [ADSP-BF518 rev 0.0](#) は、[命令 SRAM](#) の配置に深刻なバグがある。システム依存部には必ず [ezkit bf518 00](#) を使うこと。

4.2. ADSP-BF531/532/533 を使ったシステムの割り込み宣言

```
#define INHNO_PLL 0
#define INHNO_DMA_ERROR 1
#define INHNO_PPI_ERROR 2
#define INHNO_SPORT0_ERROR 3
#define INHNO_SPORT2_ERROR 4
#define INHNO_SPI_ERROR 5
#define INHNO_UART_ERROR 6
#define INHNO_RTC 7
#define INHNO_PPI 8
#define INHNO_SPORT0_RX 9
```

```

#define INHNO_SPORT0_TX          10
#define INHNO_SPORT1_RX          11
#define INHNO_SPORT1_TX          12
#define INHNO_SPI                 13
#define INHNO_UART_RX            14
#define INHNO_UART_TX            15
#define INHNO_GP_TIMER0          16
#define INHNO_GP_TIMER1          17
#define INHNO_GP_TIMER2          18
#define INHNO_PFA                 19
#define INHNO_PFB                 20
#define INHNO_MEMORY_DMA0        21
#define INHNO_MEMORY_DMA1        22
#define INHNO_WDG                 23

// SIC_ISR にない特殊な割り込み
#define INHNO_HW_ERROR            24
#define INHNO_CORE_TIMER         25
#define INHNO_RAISE              26

```

4.3. ADSP-BF534/536/537 を使ったシステムの割り込み宣言

```

#define INHNO_PLL                 0
#define INHNO_DMA_ERROR          1
#define INHNO_PERIPHERAL_ERROR   2
#define INHNO_RTC                 3
#define INHNO_PPI                 4
#define INHNO_SPORT0_RX          5
#define INHNO_SPORT0_TX          6
#define INHNO_SPORT1_RX          7
#define INHNO_SPORT1_TX          8
#define INHNO_TWI                 9

```

```
#define INHNO_SPI          10
#define INHNO_UART0_RX    11
#define INHNO_UART0_TX    12
#define INHNO_UART1_RX    13
#define INHNO_UART1_TX    14
#define INHNO_CAN_RX      15
#define INHNO_CAN_TX      16
#define INHNO_MAC_RX      17
#define INHNO_PORT_H_A    17
#define INHNO_MAC_TX      18
#define INHNO_PORT_H_B    18
#define INHNO_GP_TIMER0   19
#define INHNO_GP_TIMER1   20
#define INHNO_GP_TIMER2   21
#define INHNO_GP_TIMER3   22
#define INHNO_GP_TIMER4   23
#define INHNO_GP_TIMER5   24
#define INHNO_GP_TIMER6   25
#define INHNO_GP_TIMER7   26
#define INHNO_PORT_FG_A   27
#define INHNO_PORT_G_B    28
#define INHNO_MEMORY_DMA0 29
#define INHNO_MEMORY_DMA1 30
#define INHNO_WDG         31
#define INHNO_PORT_F_B    31

// SIC_ISR にない特殊な割り込み
#define INHNO_HW_ERROR     32
#define INHNO_CORE_TIMER  33
#define INHNO_RAISE        34
```

4.4. ADSP-BF512/514/516/518 を使ったシステムの割り込み宣言

```
/* Peripheral Masks For SIC ISR0, SIC IWR0, SIC IMASK0 */  
  
#define INHNO_PLL_WAKEUP          0  
  
#define INHNO_DMA_ERR0           1  
  
#define INHNO_DMAR0              2  
  
#define INHNO_DMAR1              3  
  
#define INHNO_DMAR0_ERR          4  
  
#define INHNO_DMAR1_ERR          5  
  
#define INHNO_PPI_ERR            6  
  
#define INHNO_MAC_ERR            7  
  
#define INHNO_SPORT0_ERR         8  
  
#define INHNO_SPORT1_ERR         9  
  
#define INHNO_PTP_ERR            10  
  
/* reserved */  
  
#define INHNO_UART0_ERR          12  
  
#define INHNO_UART1_ERR          13  
  
#define INHNO_RTC                 14  
  
#define INHNO_PPI                 15  
  
#define INHNO_SPORT0_RX           16  
  
#define INHNO_SPORT0_TX           17  
  
#define INHNO_SPORT1_RX           18  
  
#define INHNO_SPORT1_TX           19  
  
#define INHNO_TWI                 20  
  
#define INHNO_SPI                 21  
  
#define INHNO_UART0_RX           22  
  
#define INHNO_UART0_TX           23  
  
#define INHNO_UART1_RX           24  
  
#define INHNO_UART1_TX           25  
  
#define INHNO_OTP                 26  
  
#define INHNO_CNT                 27  
  
#define INHNO_ETHERNET_RX        28  
  
#define INHNO_PFA_PORTH          29
```

```
#define INHNO_ETHERNET_TX    30

#define INHNO_PFB_PORTH      31

/* Peripheral Masks For SIC ISR1, SIC IWR1, SIC IMASK1 */

#define INHNO_TIMER0         32
#define INHNO_TIMER1         33
#define INHNO_TIMER2         34
#define INHNO_TIMER3         35
#define INHNO_TIMER4         36
#define INHNO_TIMER5         37
#define INHNO_TIMER6         38
#define INHNO_TIMER7         39

#define INHNO_PFA_PORTG      40
#define INHNO_PFB_PORTG      41
#define INHNO_MDMA0_DST      42
#define INHNO_MDMA0_SRC      42
#define INHNO_MDMA1_DST      43
#define INHNO_MDMA1_SRC      43
#define INHNO_WDOG           44
#define INHNO_PFA_PORTF      45
#define INHNO_PFB_PORTF      46

#define INHNO_SPI0_ERR       47
#define INHNO_SPI1_ERR       48

/* reserved */
/* reserved */

#define INHNO_RSI_INT0        51
#define INHNO_RSI_INT1        52

#define INHNO_PWM_TRIPINT     53
#define INHNO_PWM_SYNCINT     54
#define INHNO_PTP_STATINT     55
```


// SIC_ISRにない特殊な割り込み

#define INHNO_HW_ERROR 56

#define INHNO_CORE_TIMER 57

#define INHNO_RAISE 58

5. 開発環境の構築

開発環境は Analog Devices の VisualDSP++ および GNU のツールチェーンを使用した

5.1. 開発環境のバージョン

動作確認したツールのバージョンは以下の通りである。VisualDSP++については、ライブラリのビルドのみ確認した。動作については確認していない。

- GCC 4.1.2
- VisualDSP++ : 5.0 Update 5

5.2. offset.h

VisualDSP++を開発に使う場合、あらかじめ jsp/config/blackfin/offset.h を生成しなければならない。生成には、jsp/utills/blackfin-vdsp サブディレクトリ内部のプロジェクトを用いること。VisualDSP++のシミュレータで動かすことでoffset.hを生成する。

5.3. サンプル・アプリケーション

TOPPERS プロジェクトより配布されている TOPPERS/JSP のコンフィグレータが生成する sample1 は、そのままではビルドできない。これは、sample1.h が blackfin 依存部を認識できず、タスク・スタックとして 8kB を割り当てるためである。これは ADSP-BF532、ADSP-BF533 および BF537 で使うには大きすぎる。

この対策として、以下の宣言を jsp/sample/sample1.h の CPU 依存宣言部に挿入している。そのため、CVS の jsp モジュールから取得したソースツリーを使う場合、問題無く sample1 のビルドができる。

```
#elif defined(BLACKFIN)
#define CPUEXC1 0 /* CPU 例外ハンドラ番号 */
#define RAISE_CPU_EXCEPTION asm(" excpt 0;") /* ソフトウェア割込み発生 */
#define STACK_SIZE 1024 /* タスクのスタックサイズ */
#define TASK_PORTID 1 /* 文字入力するシリアルポート ID */
```

また、継続的な動作チェックはしていないが、VisualDSP++用のプロジェクトを jsp/tools/blackfin-vdsp/ 以下に置いている。

6. 独自ボードへの移植

TOPPERS/JSP for Blackfinは Analog Devices 社の評価基板である EZ-KIT Lite BF533 上で開発された。そのため、一部設定は同ボードに依存している。それらの点は `sys_config.h` のマクロを書き換えることで変更できる。また、それ以外にもユーザーが独自のボードにあわせこむための機能が用意されている。

以下は ADSP-BF533 に関する説明だが、ADSP-BF537、[ADSP-BF518](#) に関しても同様である。

6.1. クロック設定の変更

ユーザーが任意の周波数で Blackfin を動かすことができるよう、`sys_config.h` にクロック設定用のマクロが用意してある。これらの既定値は以下のとおり。この既定値は EZ-KIT BF533 上で 600MHz 動作を行うためのものである。

```
#define CSELVAL 1
#define SSELVAL 5
#define MSELVAL 22
#define CLKIN 27000000
```

CSELVAL, SSELVAL はそれぞれ PLL_DIV レジスタの CSEL フィールド、SSEL フィールドの設定制御用の値である。CSELVAL は 1, 2, 4, 8 の中のいずれかの値を指定する。これはコア用の分周比であって PLL_DIV の CSEL フィールドに書き込む値ではないことに注意する。SSELVAL は SSEL フィールドに書き込む値そのものである。

MSELVAL は PLL_CTL レジスタの MSEL フィールドに書き込む値を指定する。これらの設定を元に `sys_config.c` の `sys_initialize()` が PLL の設定を行う。

CLKIN は ADSP-BF533 の CLKIN 端子に入力する周波数を Hz 単位で宣言する。EZ-KIT BF533 の場合 27MHz 入力なので既定値は 27000000 になっている。

以上の 4 つのマクロを元に、TIC (システム) タイマーの設定値と微小時間待機関数の設定も自動的に行われる。したがってこれらのマクロさえ正しく設定すれば TIC (システム) タイマーの設定を気にする必要はない。

6.2. システムタイマーの変更

ユーザーのシステムに応じてシステムタイマーは ADSP-BF533 の GP タイマー 2 である。ただし、システム依存部の移植の際に一時的にコアタイマーを使うことも出来る。

コアタイマーを使う場合には `sys_config.h` の中で `USE_TIC_CORE` マクロを定義する。

```
#define USE_TIC_CORE
```

デフォルトではこのマクロはコメント・アウトされている。すなわち、デフォルトでは GP タイマー 2 を使っている。コアタイマーを使う場合には、dispatch() の割り込み待ち部分で idle 命令ではなく nop 命令をつかうようにプログラムされている。

[この機能は移植時の利便性のために残してある。](#) 実アプリケーションでコアタイマーを使うことは推奨しない。

6.3. SIC_IARx の変更

SIC_IARx の値を変えると、割り込み源と優先順位の対応付けが変わる。TOPPERS/JSP for Blackfin はこの対応を表として持っているため、SIC_IARx を書き換えたら必ずこの表をアップデートしなければならない。表のアップデートは make_priority_mask() 関数を呼ぶことで完了する。

SIC_IARx の変更は複雑な問題を起こしうるため、原則としてシステム起動時に一度だけ行うべきである。コンフィグレーションファイルの中で ATT_INI によってアタッチされる初期化コードの中で変更するのがもっともよい方法である。この場合、他の初期化コードより先にアタッチしなければならない。

SIC_IARx の変更に関してはサンプルプログラムを用意しているので参照されたい。

6.4. 実行時初期化の選択

実行時初期化機能は ノーブート・モード 時に変数を初期化するための VisualDSP++ 埋め込み機能である。この機能を使うには、sys_config.h にて USE_RUNTIME_INIT マクロを宣言する

```
#define USE_RUNTIME_INIT
```

一般にブート・モードを使用するならこの機能は不要である。

6.5. 起動メッセージ

TARGET_NAME マクロは起動メッセージとして使う文字列である。

6.6. 外部ペリフェラル・レジスタの保護

外部バスに接続されたペリフェラルのレジスタが読み出しによって値が変わる破壊性のものである場合は、これらのレジスタからの読み出しを割り込みから保護しなければならない。これは、sys_defs.h において BOARD_DESTRUCTIVE_READ マクロを変更することで達成できる。なお、保護されるのはシステム・インターフェース・レイヤーを使用する場合だけである。

BOARD_DESTRUCTIVE_READ マクロは渡された引数 iop が外部バスの破壊性読み出しレジスタのアドレスと一致するときに論理値真を返すマクロである。

デフォルトでは、すべての外部アドレスに対して真を返す。

6.7. C++への対応

[Blackfin 依存部 3.2 より、C++への限定的な対応を開始した。](#)

[現時点では標準ライブラリには対応していない。new 演算子には対応しているが delete 演算子は何もしない。](#)

VisualDSP++環境でC++言語を利用するためには、あらかじめそのためのデータ領域を確保した上でその領域を初期化しなければならない。これを行うには、sys_config.hにてINIT_C_PLUS_PLUS マクロを宣言する。

```
#define INIT_C_PLUS_PLUS
```

C言語しか使わないのであればこの機能は不要である。

6.8. ハードウェア・エラーへの対応

Blackfinのハードウェア・エラー割り込みは特殊な構造になっており、コア・ウェイクアップ信号を生成しない。そのため、IDLE 命令で安全かつ即座に検出できない。

TOPPERS/JSP for BlackfinではIDLE 命令による安全な検出を行っている。この検出方法ではIDLE 命令の実行中にハードウェア・エラーが発生しても、他のデバイスによる割り込み要求が発生するまでハードウェア・エラーを検知できない。他のデバイスによる割り込み要求が受理されると、IDLE 命令を終了して、受理されたデバイスではなくハードウェア・エラー・イベントが実行される。

安全かつ即座に検出したい場合には以下のマクロを sys_config.h で宣言する。

```
#define QUICK_HW_ERROR
```

なお、このマクロを宣言するとIDLE 命令を使わないため、割り込み待ち状態での消費電力が増大する。

6.9. 管理外割り込みへの対応

汎用割り込みイベントを管理外割り込みとして予約することが出来る。この場合、sys_config.hでUSE_HW_ERROR マクロを宣言する。

```
#define UNMANAGED_INT 0x0020
```

割り込み順位はビットマップとして指定する。上の例では、ビット5が1なので、IVHWが管理外割り込みとなる。

管理外割り込みハンドラは、割り込みスタックの用意、資源の退避、復帰、RTI 命令の実行をすべて自分自身で行わなければならない。また、ハンドラの登録もプログラマが自分自身でイベント・ベクトルに書き込まねばならない。

カーネル管理外割り込みを利用する場合、その割り込み用のスタックもユーザーが用意し、イベントのエントリで設定しなければならない。また、カーネル管理外割り込みの最中であっても、CPU 例外は発生する。CPU 例外はの場合カーネル管理外割り込みのスタックをそのまま使用するので必要量は慎重に計算しなければならない。

IMASK レジスタの、カーネル管理外割り込みに対応するビットは、kernel_start ルーチンの開始時点で0になっている。このビットのオン/オフはアプリケーション・プログラムが行うこと。

6.10. システム略称

TOPPERS/JSP の jsp/doc/config.txt の指針に基づき、sys_def.h の「システム略称」マクロを変更すること。

6.11. PLL の強制初期化

sys_config.h で FORCE_PLL_INITIALIZE マクロを宣言すると、PLL および SDRAM コントローラの初期状態にかかわらず、PLL を初期化する。

これは、VisualDSP++のようにデバッガが SDRAM を初期化するような環境で利用する時に便利である。マクロを宣言しない場合には、PLL および SDRAM が未初期化時に限って PLL を初期化する。

7. 他プロセッサへの移植

Blackfin コアを使った他のプロセッサへの移植はチップ依存部を書き換えることによって行う。ADSP-BF533 チップ依存部は `jsp/config/blackfin/_common_bf533` にまとめてある。CPU 依存部は Blackfin コアにのみ依存するように設計しているが、問題がある場合はこの文書の「[連絡先](#)」に書いてある sourceforge まで連絡していただきたい。

7.1. システムタイマー制御

システムタイマーは `hw_timer.h` を書き換えて移植する。コアタイマーを使う場合にはこの部分の変更はないと思われるが、GP タイマーを使う場合は変更が必要になる。

7.2. シリアルポート制御

シリアルポートのアドレス宣言は `chip_config.h` 中で行っている。この部分を書き換えるだけでシリアルポートの移植は終わりである。

ただし、カーネルが割り込みを許可する前のコピーライト表示や、スプリアス割り込み中の文字列表示機能は、TOPPERS/JSP のシリアルポート機能を使わない。このため、`sys_config.c` に `は割り込みを使わないUART 通信用のコードが含まれている。sys_putc() および sys_initialize() を参照のこと。`

7.3. PLL 制御

PLL の制御は `chip_config.c` で行っている。場合によっては `sys_config.h` の設定用マクロも変更が必要になるかもしれない。変更を受けうるのは `MSELVAL`, `CSELVAL`, `SSELVAL`, `CLKIN` である。一例として ADSP-BF535 の場合 `CSEL` が不要になる。

7.4. `INTNO_xxxx` と `INHNO_xxxx`

`INTNO_xxxx` と `INHNO_xxxx` は `chip_config.h` に定義してある。これらは全面的に再宣言が必要である。なお、システムタイマー割り込みハンドラ番号である `INHNO_TIMER` には `USE_TIC_CORE` の定義の有無に応じて適切な値を割り当てる。具体的には `USE_TIC_CORE` があるときにはコアタイマー割り込みの割り込みハンドラ番号を、ないときにはシステムタイマーとして使う GP タイマーの割り込みハンドラ番号を割り当てる。

ハードウェア割り込み、コアタイマー割り込み、ソフトウェア割り込み用のハンドラ番号は、デバイス用の割り込みハンドラ番号より大きなものを割り当てる。

7.5. DEVICE_INTERRUPT_COUNT マクロ

マクロ定数 `DEVICE_INTERRUPT_COUNT` に割り込み要因の数を定義すること。これは `SIC_IMASK` のビット数である。

7.6. `priority_mask[]` と `make_priority_mask()`

`priority_mask[]` のサイズは `DEVICE_INTERRUPT_COUNT` マクロの値+3 にする。

割り込み要因の数が 32 を超える場合、`priority_mask` を大幅に変更しなければならない。同様に `make_priority_mask()` も変更が必要になる。

割り込み要因の数が 32 を超えない場合でも、`priority_mask[]` の初期値はプロセッサに合わせて変更しなければならない。

7.7. DESTRUCTIVE_READ マクロ

`DESTRUCTIVE_READ(iop)` マクロはシステム・インターフェース・レイヤーの I/O アクセス関数で使われる。このマクロは、渡された I/O アドレス `iop` のレジスタ内容が読み出しによって破壊されるなら真、読み出しによっては破壊されないならば偽を返す。一般には受信レジスタや UART のステータスレジスタが「破壊読み出し」にあたる。

`DESTRUCTIVE_READ` マクロは `sys_defs.h` で宣言する。このマクロは論理式の形をとり、引数として与えられた `iop` を保護が必要なレジスタと比較すればそれでよい。なお、プロセッサ外部のペリフェラルの保護のために、ADSP-BF533 の実装では `BOARD_DESTRUCTIVE_READ` を `sys_defs.h` で宣言している。プロセッサ内蔵ペリフェラルについては考慮しなくて良い(ADSP-BF535 を除く)。

このマクロをどうすればよいかわからないならば、単に

```
#define DESTRUCTIVE_READ(iop) 1
```

としておけばよい。ただしすべての外部 I/O アクセスは遅いものになる。

7.8. [chip_debug.c](#)

[2010 年前半時点で、`bf_in_gdbproxy` のリセット機能に問題がある。そのため、GDB によるデバッグを行う場合には最初に実行コード内で `Blackfin` をシステム・リセットしなければならない。そのため `chip_debug.c` の `boot_for_gdb\(\)` 関数を `start.S` から呼んでいる。この関数は大域変数](#)

[`enable_boot_for_debug` が真の場合のみ再ブートを行う。再ブート前に同変数を偽に変えるため、再ブートは 1 回だけ実行される。](#)

[再ブートの方法はプロセッサによって異なりうるので、注意が必要である。](#)

7.9. chip_dump.c

スプリアス割り込みやスプリアス例外が発生すると、chip_dump.c内部の関数が呼ばれる。これらはプロセッサによって異なりうるので移植に注意する。

8. その他

8.1. ディレクトリ・ファイル構成

Blackfin ターゲット依存部の各ファイルの概要は次の通り.

config/blackfin/

<code>cpu_defs.h</code>	プロセッサ依存部のアプリケーション用定義
<code>cpu_config.h</code>	プロセッサ依存部の構成定義
<code>cpu_config.c</code>	プロセッサ依存部の関数
<code>cpu_support.S</code>	プロセッサ依存部のサブルーチン
<code>cpu_context.h</code>	コンテキスト操作
<code>cpu_support.S</code>	プロセッサ依存部のサブルーチン
<code>cpu_malloc.c</code>	C++用の割り込み保護された <code>malloc</code> 関数
<code>cpu_rename.h</code>	カーネルの内部識別名のリネーム
<code>cpu_unrename.h</code>	カーネルの内部識別名のリネーム解除
<code>tool_defs.h</code>	開発環境依存部のアプリケーション用定義 (VisualDSP++ 開発環境用)
<code>tool_config.h</code>	開発環境依存部の構成定義 (VisualDSP++開発環境用)
<code>start.S</code>	スタートアップモジュール

[config/blackfin/_common_bf518](#)

chip_debugboot.c	gdbproxy のリセット問題対策用
chip_dump.c	未定義割り込み用のダンプ関数
chip_defs.h	チップ依存部のアプリケーション用定義
chip_defs.c	チップ依存部のアプリケーション用関数
chip_config.h	チップ依存部の構成定義
chip_config.c	チップ依存部の関数
sys_rename.h	カーネルの内部識別名のリネーム
sys_unrename.h	カーネルの内部識別名のリネーム解除
hw_timer.h	タイマ操作ルーチン
hw_serial.h	PDIC インターフェース
bf518elf.ld	ADSP-BF518 用 ld スクリプト
bf518_00elf.ld	ADSP-BF518 rev 0.0 用 ld スクリプト

[config/blackfin/_common_bf533](#)

chip_debugboot.c	gdbproxy のリセット問題対策用
chip_dump.c	未定義割り込み用のダンプ関数
chip_defs.h	チップ依存部のアプリケーション用定義
chip_defs.c	チップ依存部のアプリケーション用関数
chip_config.h	チップ依存部の構成定義
chip_config.c	チップ依存部の関数
sys_rename.h	カーネルの内部識別名のリネーム
sys_unrename.h	カーネルの内部識別名のリネーム解除
hw_timer.h	タイマ操作ルーチン
hw_serial.h	PDIC インターフェース
bf533elf.ld	ADSP-BF533 用 ld スクリプト
bf532elf.ld	ADSP-BF532 用 ld スクリプト

[config/blackfin/_common_bf537](#)

chip_debugboot.c	gdbproxy のリセット問題対策用
chip_dump.c	未定義割り込み用のダンプ関数
chip_defs.h	チップ依存部のアプリケーション用定義
chip_defs.c	チップ依存部のアプリケーション用関数
chip_config.h	チップ依存部の構成定義

chip_config.c	チップ依存部の関数
sys_rename.h	カーネルの内部識別名のリネーム
sys_unrename.h	カーネルの内部識別名のリネーム解除
hw_timer.h	タイマ操作ルーチン
hw_serial.h	PDIC インターフェース
bf537elf.ld	ADSP-BF537 用 ld スクリプト

[config/blackfin/ezkit_bf518](#)

sys_defs.h	システム依存部のアプリケーション用定義
sys_config.h	システム依存部の構成定義
Makefile.config	システム依存部の依存性宣言

[config/blackfin/ezkit_bf518_00](#)

sys_defs.h	システム依存部のアプリケーション用定義
sys_config.h	システム依存部の構成定義
Makefile.config	システム依存部の依存性宣言

config/blackfin/ezkit_bf533

sys_defs.h	システム依存部のアプリケーション用定義
sys_config.h	システム依存部の構成定義
Makefile.config	システム依存部の依存性宣言

config/blackfin/ezkit_bf537

sys_defs.h	システム依存部のアプリケーション用定義
sys_config.h	システム依存部の構成定義
Makefile.config	システム依存部の依存性宣言

config/blackfin/[kobanzamebf533eb](#)

sys_defs.h	システム依存部のアプリケーション用定義
sys_config.h	システム依存部の構成定義
Makefile.config	システム依存部の依存性宣言

[config/blackfin/rebun](#)

sys_defs.h	システム依存部のアプリケーション用定義
sys_config.h	システム依存部の構成定義
Makefile.config	システム依存部の依存性宣言

pdic/simple_sio

uart.h	UART 宣言
--------	---------

8.2. 連絡先

TOPPERS/JSP for Blackfinへの要望やバグ報告は
<http://sourceforge.jp/projects/toppers.jsp4bf/> まで連絡されたい。なお、TOPPERS/JSP 自身の使い方等は、情報共有のためにも TOPPERS プロジェクト(<http://www.toppers.jp/>)のMLで行うことをお勧めする。

9. 文書履歴

9.1. 2004/09/18

リリース 1.0.1 に適合するよう、「独自ボードへの移植」の章を `sys_config.h` を元にしたものに変更

`USE_RUNTIME_INIT` のつづりを訂正。

9.2. 2004/10/30

`DESTRUCTIVE_READ` に関する記述を追加。

9.3. 2005/02/05

`DESTRUCTIVE_READ` に関する記述を、`sys_defs.h` を使用する形に変更。`sys_defs.h` に関する記述を追加。

予約資源に関する記述を追加。

管理外割り込みに関する記述を追加。

9.4. 2005/03/22

C++対応機能に関する記述を 5.7 節として追加。

9.5. 2005/04/01

管理外割り込みと、ハードウェア・エラー割り込みへの対応を追加。

9.6. 2006/02/05

シリアル・インターフェース・ドライバに関する記述を追加。管理外割込みに関する補足を追加。

9.7. 2006/08/06

Blackfin 依存部リリース 2.x に対応するために記述を変更。GCC に関する記述を追加。シリアル・インターフェース・ドライバに関する記述を修正。Sourceforge に関する記述を追加。著作権者に関する記述を追加。

9.8. 2006/12/31

Blackfin 依存部リリース 2.0.1 に対応するために記述を変更。ADSP-BF537 に関する記述を追加。offset.h に関する記述を追加。カーネル管理外割り込みの初期状態が禁止であることを明記。

9.9. 2007/6/11

Blackfin 依存部のリリース番号を 2.0.2 に変更。TOPPERS/JSP のリリース番号を 1.4.3 に変更。TOPPERS/JSP 1.4.2 と 1.4.3 の差は僅少であり、Blackfin 依存部はなんら変更を行わずに両者に対応できる。

5.3 節に Sample1 のビルドに関する注意を追加。

9.10. 2008/12/12

Blackfin 依存部のリリース番号を 2.2.0 に変更。小規模の書き間違いのほか、文書内容に幾分の補足。また、システム依存部変更に対応した。

9.11. 2009/1/25

Blackfin 依存部のリリース番号を 2.3.0 に変更。6.8 節「ハードウェア・エラーへの対応」の記述を全面的に変更。6.10 節「システム略称」を追加。4.1 節、4.2 節にプロセッサごとの INHNO_XXX 宣言を掲載。

9.12. 2009.2.3

Blackfin 依存部のリリース番号を 3.0.0 に変更。システム依存部の構造変更にあわせて全面的に改定。

9.13. 2009.5.30

Blackfin 依存部のリリース番号を 3.0.1 に変更。dis_int(), chg_ims() を廃止したことを明記。コアタイマーを非推奨とすることを明記。

9.14. 2009.7.26

Blackfin 依存部のリリース番号を 3.1.0 に変更。この版では、カーネルソースコード中から VisualDSP++ 由来のヘッダファイルを取り除いたほか、E!Kit-BF533 への対応を追加している。また、gdb でデバッグするための mmrXXXXX レジスタ宣言を行った。

9.15. 2009.8.16

Blackfin 依存部のリリース番号を 3.1.1 に変更。この版では、gdb に対応したリセットコードをデフォルトで無効にする変更を行った。

9.16. [2010.7](#)

[Blackfin 依存部のリリース番号を 3.2.0 に変更。この版では BF518 依存部を追加し、C++ への暫定的な対応を開始した。](#)