TOPPERS/JSP for Blackfin ユーザーズ・マニュアル

Blackfin 依存部 3.3.0 対応版 20120728

はじめに

TOPPER/JSP は、TOPPERS プロジェクトがリリースしているフリーの uITRON4.0 準拠 RTOS です。

RTOS としては別段とんがっているわけではありませんが、uITRON はそこそこ関連文書が入手しやすい上に仕様が枯れているので安心して使う事が出来ます。TOPPERS プロジェクトは既に TOPPERS/JSP カーネルの次の世代の TOPPERS/ASP カーネルに軸足を移していますが、逆に言えば JSP カーネルは枯れていると言えます。

この文書は TOPPERS/JSP カーネルの Blackfin 依存部について、利用のしかたをツールのインストールから説明します。このプロジェクトの成果物の最新版は TOPPERS/JSP for Blackfin プロジェクトから入手できます。

1.1 TOPPERS/JSP for Blackfin について

TOPPERS/JSP for Blackfin は TOPPERS/JSP を Analog Devices 社の Blackfin DSP に移植したもので、TOPPERS プロジェクトによらない非公式の配布を行っています。ライセンスは TOPPERS ライセンスですので商用、非商用の区別無く自由につかえ、GPL2 コードと同時に使う事もできます。

Blackfin 依存部は 2001 年 9 月頃に最初の検討が始まり、断続的な作業を経て 2004 年 7 月に実機上で最初の動作を開始しました。最初に対応したハードウェアはアナログ・デバイセズ社の評価ボード、EZ-KIT Lite BF533 です。現在 ADSP-BF533、BBF537、BF518 で動作するシステム移植部が公開されています。

Blackfin プロセッサはマイナーな部類のプロセッサですが、アーキテクチャを TOPPERS/JSP でくるむことで利用時の障壁を著しく下げることが出来ます。また、オーディオ信号処理やビデオ信号処理では遅延がそれほど問題にならない場合もあり、RTOS の利用に向いています。Blackfin アーキテクチャが比較的素直な 32bit プロセッサということもあり、TOPPERS/JSP のオーバーヘッドは非常に小さなレベルにとどまっています。

開発環境は GCC です。Blackfin 用の GCC は Blackfin Koop によってメンテナンスされており、ツールの入手も用意です。

1.2 TOPPERS/JSP for Blackfin プロジェクトについて

プロジェクト・ページは Sourceforge によってホスティングされています。

- http://sourceforge.jp/projects/toppersjsp4bf/

このプロジェクトではリリース毎に TOPPERS/JSP for Blackfin のソースコードをパッケージ化してアップロードしています。また、最新の非リリース・コードを取得したければ、CVS から入手することも出来ます。

掲示板もあるので、質問があればそちらで問い合わせることが可能です。

1.3 ライセンス及び責任の放棄

TOPPERS/JSP for Blackfin カーネルは TOPPERS ライセンスに従います。同ライセンスについては TOPPERS プロジェクトを参照してください。基本的に商用、非商用を問わず、利用料やソースコード開示を求めないライセンスです。

TOPPERS/JSP for Blackfin プロジェクトの成果物を使った結果については、プロジェクト関係者あるいはその他の誰も、金銭的、道義的責任を負いません。またサポート義務も負いません。いかなる損害に対しても保証、謝罪をいたしません。

開発環境のインストール

1.4 解説する環境

この文書では

- Ubuntu 12.04 LTS
- Blackfin GCC Tool chain 2011R1-RC4
- Eclipse 3.7.2 Indigo
- TOPPERS/JSP 1.4.3
- TOPPERS/JSP Blackfin 依存部 3.3.0

を利用した開発環境の構築を説明します。ホストコンピュータは物理マシンの他 VMware Workstation 7.8 上の仮想マシンでも USB JTAG ICE によるターゲットのデバッグが可能であることを確認しています。なお、例に使うハードウェアは

- JTAG ICE : もなみソフトウェア Tiny JTAG for Blackfin/SHARC「刺身包丁」および gnICE+
- ターゲット : J-Person ADSP-BF533 ボード“KOBANZAME”

です。

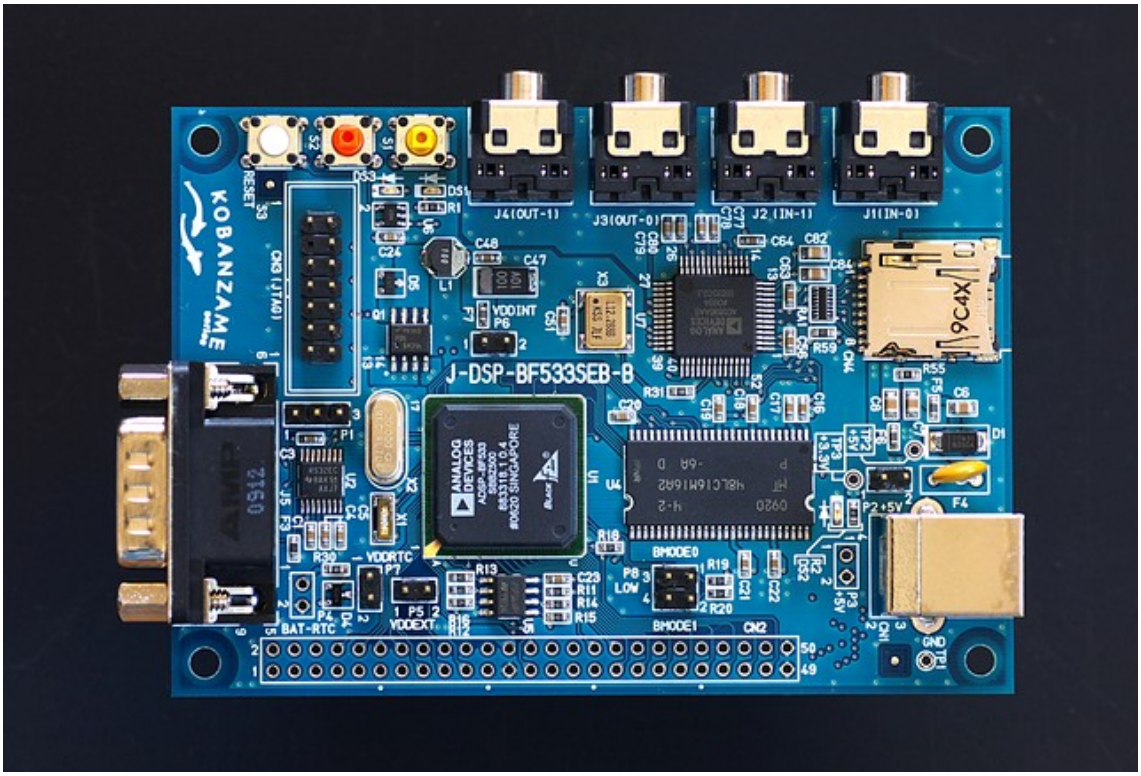

写真 1 Kobanzame 基板

1.5 Linux の取得と設定

開発環境には Linux のディストリビューションの一つ、Ubuntu 12.04LTS を使います。

Ubuntu 12.04 のインストール CD イメージは、以下のページから取得出来ます。英語版ですが、取得した Ubuntu はインストール時に指定すれば日本語環境として使用できます。

CD イメージを入手したら CD に焼きます。この CD からブートすることで実マシンへ Ubuntu をインストールする

ことができます。

- http://releases.ubuntu.com/jaunty/

日本語 Remix 版の方がいいという方は、そちらを試してみるのも面白いでしょう。

- http://www.ubuntulinux.jp/products/JA-Localized/download

実マシンに Ubuntu を入れることに躊躇するならば、VMware Player をお奨めします。VMware 上でも USB ICE を使えることを確認しています[1]。VMware Player 3.0 からは仮想マシンを作れるようになりましたので、インストールは簡単です。

- http://www.vmware.com/jp/products/player/

VMware Player 上に Ubuntu をインストールする方法は、ネットに沢山解説がありますのでそちらを参照してください。なお、VMware Player にインストールするのであれば、CD イメージを CD に焼く必要はありません。VMware Player はイメージ・ファイルを CD としてマウントすることが出来ますので、ダウンロードしたイメージから直接仮想マシンを起動出来ます。

1.6 ツールチェーンの取得と設定

Blackfin GCC コンパイラを初めとするツールチェーンは Blackfin Koop サイトのリリース・ファイルを使用します。ツールチェーンの各コマンドは Linux 上で使用します。

インストールには Ubuntu 上で TOPPERS/JSP for Blackfin プロジェクトの以下の URL のページから、Installer の release_ubuntu_1204_gcc_11r1_rev2 パッケージにある tar.gz ファイルをダウンロードし、展開して installer スクリプトを実行します。このスクリプトは Blackfin Toolchain の必要なファイルをダウンロードし、展開して/opt 以下にインストールします。

- http://sourceforge.jp/projects/toppersjsp4bf/releases/?package_id=10761

スクリプトのあるディレクトリで、次のように Shell コマンドを入力するとインストールがはじまります。途中、パスワードを求められたら自分のアカウントのパスワードを与えてください。

```
$ ./installer
```

このスクリプトはツールのインストール、パスの設定に加えて、Kermit のインストールも行います。このスクリプトはホーム・ディレクトリに.kermrc を作ります。Kermit は起動するとこのファイルを開いて設定を読み込みます。設定は以下の通りです。

- 使用ポートは /dev/ttyUSB0
- 57600baud, 8bit, 1 stop bit, non parity

使用するポートは環境によって異なるので、必要に応じて内容を修正して使用してください。

1.7 Eclipse の取得

Eclipse は、Ubuntu の標準パッケージがスクリプトによって自動的にインストールされます。インストールされるのは Eclipse 3.7.2 Indigo および CDT です。

[1] 確認したのは VMWare Player ではなく、VMware Workstation 7.0

TOPPERS/JSP for Blackfin の取得

1.8 Release ファイルを取得する

TOPPERS/JSP for Blackfin を取得する一番いい方法は、プロジェクトからリリースされているファイルをダウンロードすることです。

- http://sourceforge.jp/projects/toppersjsp4lpc/releases/

この文書の執筆時点で最新のリリースは Release 3.3.0 です。ダウンロードすると、ファイル名が blackfin.3.3.0.utf8.tar.gz となっていますので、次のコマンドで展開します。

```
$ tar xvzf blackfin.3.3.0.utf8.tar.gz
```

展開するとソース・ツリーが現れます。ソース・ツリーは後で説明するようにアプリケーション・ディレクトリにコピーして使います。

1.9 CVS から取得する

何らかの事情でリリースされている以外の版が欲しい場合には、CVS を使います。TOPPERS/JSP for Blackfin のソース・ツリーを取得するには、次のパラメータで CVS をアクセスします。

- アドレス　　cvs.sourceforge.jp
- パス(CVSROOT)　　/cvsroot/toppersjsp4bf
- ユーザー名　　anonymous
- タイプ　　pserver
- モジュール名 jsp

プロジェクト初期からいろいろと変遷があったため、モジュールがたくさんあります。間違えずにモジュール jsp を使用してください。

Eclipse を CVS のクライアントとして使う事もできます。このクライアントは使い勝手がいいので活用をお奨めします。また、手元のソースと CVS サーバー上の各バージョンの比較機能も優れています。Eclipse による CVS アクセスについては、Eclipse 本やネット上の資料を参照してください。

Sample1 のビルドと実行

1.10 Sample1 について

Sample1 は TOPPERS/JSP の移植後のこけら落としに使われるアプリケーションで、TOPPERS/JSP のソース・ツリー中に初めから含まれています。

このアプリケーションはシリアル・ポートを通してユーザーと通信し、ユーザーのコマンドを受けてタスク状態を変更します。利用方法は TOPPERS/JSP カーネルのソース・ツリーの jsp/sample ディレクトリの sample1.c ファイルにありますのでそちらを参照してください。

1.11 ディレクトリ構成

アプリケーションのディレクトリ構成は以下のようにします。

```
sample1/jsp_current
```

jsp_current は取得したソース・ツリーの最上位階層を jsp から名前変更したものです。この名前の変更は必須です。変更する名前は何でもいいのですが、jsp のままだとビルドに失敗しますので注意してください。その上の sample1 はアプリケーションを構築するディレクトリです。アプリケーション・ディレクトリの名前は sample1 でなくてもかまいませんが、以下で“Sample1”と言う名前のアプリケーションを作るので、ここでは sample1 というディレクトリ名にしておきます。

TOPPERS/JSP のソース・ツリーはアプリケーション・ディレクトリの下に置いた方が便利ですが、CVS クライアントなど他のツールの都合で別のディレクトリに置きたいならば、そうしてもかまいません。Eclipse を CVS クライアントに使う場合には、アプリケーション・ディレクトリと jsp を並置したほうが便利です。この場合、jsp の名前を変更する必要はありません。

1.12 コンフィギュレータのビルド

最初に TOPPERS/JSP のコンフィグレータをビルドします。

コンフィグレータとは、アプリケーションが使うセマフォやタスクを、コンフィグレーション・ファイルの宣言に従って事前に生成する TOPPERS/JSP のユーティリティです。TOPPERS/JSP はソース・ツリーにコンフィグレータのソースを持っていますので、ソース・ツリーをコピーしてきたら一度だけビルドを行います。

ビルドを行うには、shell から以下のように入力します。

```
$ cd sample1/jsp_current/cfg/
$ make depend
$ make
```

以上でビルドは終わりです。ビルドが終わったらアプリケーション・ディレクトリに移動してください。

```
$ cd ../..
```

1.13 コンフィギュレーション

コンフィギュレータのビルドが終わったら、Sample1 アプリケーションのコンフィギュレーションを行います。

```
$ jsp_current/configure –C Blackfin –S kobanzame
```

configure スクリプトのパラメータ –C は CPU 依存部の名前を指定します。この名前は jsp_current/config サブディレクトリにあるディレクトリ名と同じでなければなりません。同じく、-S はシステム依存部の名前を指定します。これはシステム依存部のディレクトリ名と同じでなければなりません。

アプリケーションのファイル指定がありませんが、今はこれでかまいません。TOPPERS/JSP の configure スクリプトは、アプリケーション固有のファイルが指定されない場合には Sample1 アプリケーションをカレント・ディレクトリに生成します。

ここで生成されるファイルを見ておきましょう。

1.13.1 sample1.c, sample1.h

この二つのファイルは sample1 アプリケーションのソース・ファイルです。

今回は自動生成されましたが、ユーザーが自身のアプリケーションを開発する場合には当然アプリケーションのソースは自分で用意することになります。

1.13.2 sample1.cfg

このファイルは sample1 のコンフィグレーション・ファイルです。このファイルにはアプリケーションが使用するタスクやセマフォと言った ITRON の資源を宣言します。

Sample1 の場合、このファイルは自動的に生成されるので特にユーザーが触る必要はありません。ユーザーが独自のアプリケーションを開発する場合には自分で記述することになります。

1.13.3 Makefile

このファイルは configure スクリプトがそのパラメータから生成したファイルで、ビルドに必要なターゲット情報やパス情報などが組み込まれています。

このファイルはユーザー独自のアプリケーションであっても configure スクリプトが自動生成します。逆に言えば、configure と言う名前ですがこのスクリプトはあまりコンフィギュレータらしいことは指定ないように思えます。

1.14 ビルドと実行

configure スクリプトを実行したらビルドを行います。

```
$ make depend
$ make
```

Intel CORE i3、Windows 7 Professional 64bit 上の VMware 上に構築した Ubuntu 環境で、ビルドには 5 秒程度かかります。

ビルドが完了すると jsp という名前のファイルが生成されます。これが Blackfin の実行プログラムを格納する ELF ファイルです。このほかに、ROM 化に使用する jsp.srec ファイル等も生成されます。

1.15 gdbproxy

ビルドによって生成された jsp ファイルは GDB を使ってターゲット・ボードにロードします。JTAG ICE には USB 接続の安価なものを使用しますので、USB デバイスと GDB の間を取り持つプログラムが必要です。その仲立ちプログラムが gdbproxy です。

Blackfin 用の gdbproxy は Blackfin Koop によって移植されており、既に説明したインストーラーによってインストール済みです。このプログラムはコマンド・ラインから起動し、使用する JTAG ICE によってパラメータが異なります。以下では「刺身包丁」「gnICE+」の二つの ICE について説明します。

1.15.1 刺身包丁

刺身包丁は、もなみソフトウェアが開発した Blackfin 用の小型 JTAG ICE です。

- http://preview.tinyurl.com/ybffvge （tinyurl.com のプレビュー画面経由でリンク）

この ICE は OOCDLINK 互換回路となっており、ピン配置が ADI の JTAG 配列となっているため、Blackfin 基板に直接挿すことができる使いやすい ICE です。

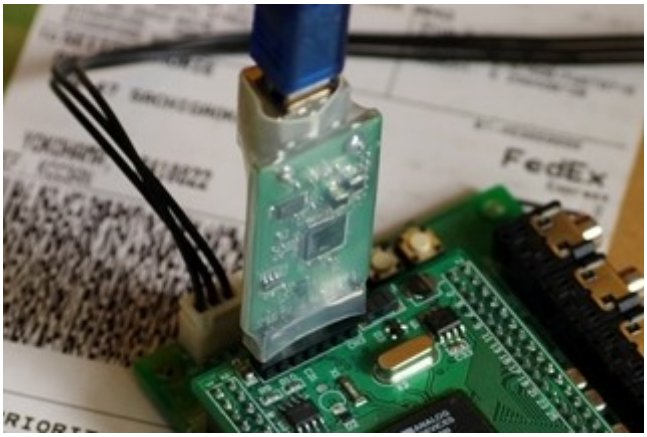

写真 2 刺身包丁

刺身包丁を使うには shell から以下のようにコマンドを投入します。

```
$ sudo /opt/uClinux/bfin-elf/bfin-gdbproxy bfin
```

接続が成功すると図2のようにターゲット情報とステータスが表示されます。GDB 向けの TCP ポートは 2000 番が開かれて接続待ちになります。2

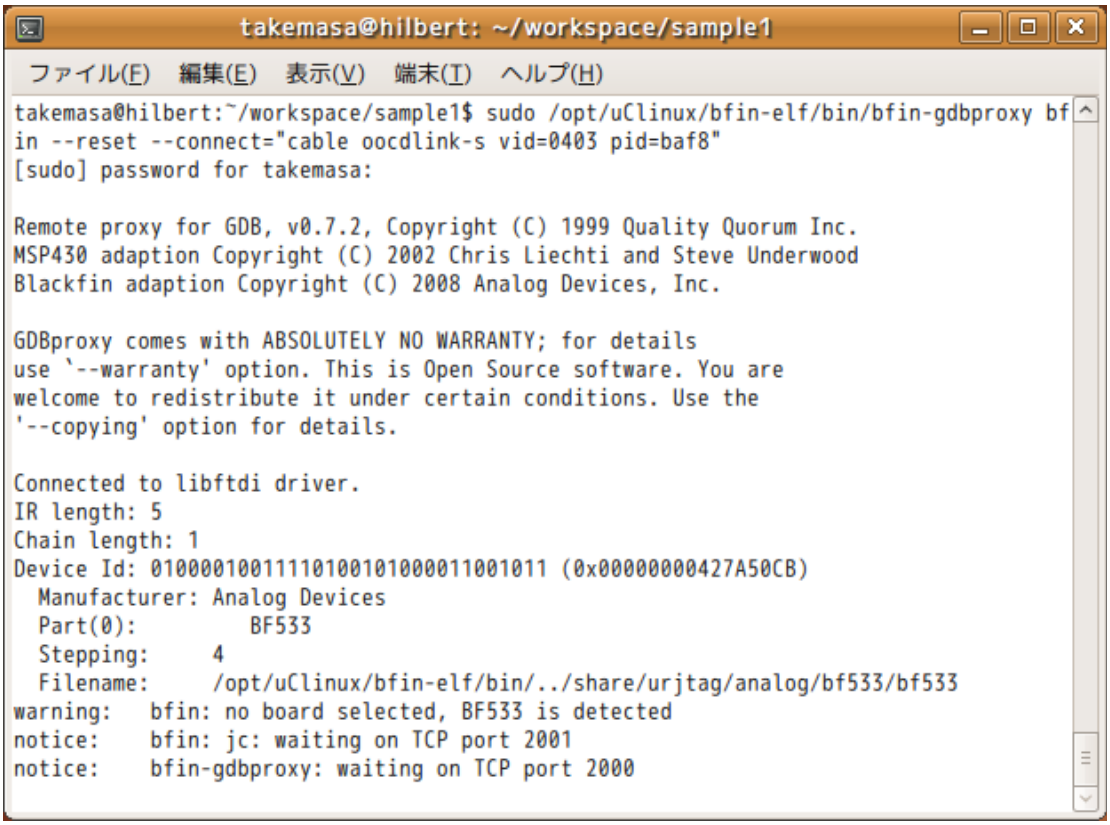

図 1 gdbproxy の接続が成功した様子

1.15.2 gnICE+

gnICE+は、Blackfin Koop で開発された USB ICE です。販売は Bluetechnix で、DigiKey などから通信販売で購入できます。

- http://docs.blackfin.uclinux.org/doku.php?id=hw:jtag:gnice-plus

刺身包丁同様に Blackfin 専用設計となっており、gdbproxy からの接続も簡単です。

刺身包丁を使うには shell から以下のようにコマンドを投入します。sudo も絶対パスも不要です。

```
$ bfin-gdbproxy bfin
```

接続が成功すると図1のようにターゲット情報とステータスが表示されます。GDB 向けの TCP ポートは 2000 番が開かれて接続待ちになります。

JTAG ポートの pin 5 について

ADI の JTAG ポートは、pin 5 の扱いに混乱があります。そのため、上のとおりに作業しても正しく動作しない場合があります。ADI の DSP ターゲット・ボードは、設計時期や準拠する設計指針に応じて pin 5 の役割が次のように異なります。

1. GND
2. インターフェース電圧検出用端子
3. JTAG ポッド用電源

すでに 1 の使い方はほとんどされていません。Blackfin のボードでこの方法を採用しているのは、非常に古い設計のものだけと思われます。

2 の方法が ADI の正式の設計で、JTAG インターフェースの設計指針である EE-68 にはこの方法を採用するよう指示されています。

http://www.analog.com/static/imported-files/application_notes/ee-68_rev10.pdf

3 の方法は、Blackfin Koop がボードを作ったときに作用しました。初期の uClinux for Blackfin プロジェクトでは、利用できる ICE が限られていたため、ARM 系のプリンタポート ICE などを使わざるを得ませんでした。そのため、ICE のインターフェース・バッファ用電源がターゲット側に必須であり、苦肉の策として電圧検出ピンからそのまま電源を出していました。

この文書の執筆時点では、Blackfin 専用の ICE がいくつか出ており、それらはインターフェース用の電源をターゲットから供給する必要はありません。そのため、3 の方法は今後は廃れていくと思われます。

さて、これらの事情から、ボードによっては使えない ICE が出てきます。JTAG を変換ケーブルで Blackfin ターゲットに接続する場合、ターゲットから IO 電源を供給しなければ動作しません。そのため、EZ-KIT など、EE-68 に正しく準拠したボードでは何らかの形で外部から pin 5 に電源を与えてやらない限り動作しません[1]。

刺身包丁と gnICE+に関しては、この文書の執筆時点で手元にあるものについては、外部インターフェース電源を必要としません。したがって、上のいずれの設計でも動作します。

今後基板を設計する際には上記の 2 の方法をとることをお奨めします。その上でどうしても 3 の方法に未練があるならば、ピン 5 と電源を接続する抵抗をジャンパーで短絡できるようにしておくといいでしょう。

1.16 gdb によるロードと実行

GDB Proxy が正常に起動して待機メッセージが表示されたら、GDB を起動してターゲットと接続できます。

GDB のコマンド名は bfin-elf-gdb です。パラメータとして ELF 形式の実行プログラムを与えてください。TOPPERS/JSP の場合、ファイル名は jsp になります。

GDB を起動したら、target コマンドでターゲットに接続します。

```
(gdb) target remote localhost:2000
```

“remote”は、target コマンドの引数で、ネットワーク越しにターゲットにアクセスすることを意味します。そのターゲットはネットワーク・アドレスとポート番号を”:”（コロン）でつなげた形式で指定します。上の例では自 PC（localhost）の 2000 番ポートを使って接続することを指示しています。図 2 にあるように 2000 番ポートは gdbproxy が接続を待っているポートです。

接続が完了したらターゲットに実行プログラムをロードします。そして必要に応じて、_kernel_enable_boot_for_debug 変数に 1 をセットし、実行します。_kernel_enable_boot_for_debug 変数に 1 を書き込むのは、bfin_gdbproxy のリセット能力の不備を補うためです。bfin_gdbproxy はターゲットのリセット

[1] EZ-KIT BF537 は例外的に 3.3V が pin 5 に生で出ている。

能力が不完全なのか、単にターゲットにプログラムをロードして実行すると失敗することがあります。そこで、TOPPERS/JSP for Blackfin は、boot_for_gdb()という関数を使い、_kernel_enable_boot_for_debug 変数が真であるときに限り初期化中にシステムリセットをかけて動作を強制的に安定化しています。この変数は通常は 0 で初期化されますので、GDB を使わないときにはシステム・リセットはおきません。

kernelenable_boot_for_gdb の設定はオプションです。通常は適切なブートモードを選んでおけば、monitor reset コマンドの実行によって正しくリセットされるようです。ブートモードは ADSP-BF531/2/3 ではノーブートモードを、それ以外のプロセッサは SPI Slave か UART SLAVE モードを選びます。

実行時の一連のシーケンスは面倒ですので、以下の内容のファイル”.gdbinit”を作って作業用ディレクトリに入れておくことをお奨めします。このファイルは GDB を起動するたびに実行されます。

```
target remote localhost:2000
monitor reset
load
set _kernel_enable_boot_for_gdb = 1
```

GDB の詳しい操作方法については説明をしません。他の資料を参照してください。図3に実行までの様子を図示します。このスクリーンショットは上の.gdbinit を使ったときのものです。

```
takemasa@hilbert:~/workspace/sample1$ bfin-elf-gdb jsp
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=bfin-elf"...
0xef000000 in ?? ()
Loading section .rodata, size 0x11e8 lma 0xff800000
Loading section .ctors, size 0x8 lma 0xff8011e8
Loading section .dtors, size 0x8 lma 0xff8011f0
Loading section .data, size 0x50 lma 0xff8011f8
Loading section .start, size 0x114 lma 0xffa00000
Loading section .text, size 0x50fc lma 0xffa00114
Loading section .init, size 0x3c lma 0xffa05210
Loading section .fini, size 0x28 lma 0xffa0524c
Start address 0xffa00000, load size 25788
Transfer rate: 110205 bits/sec, 2865 bytes/write.
Current language:  auto; currently asm
(gdb)
```

図 2 GDB をターゲットに接続した様子

1.17 Kermit による動作確認

「ツールチェーンの取得と設定」で解説したスクリプトは、Kermit の設定とインストールも行います。Kermit を使って sample1 の動作確認を行うことができます。

最初に、ここで説明する Kermit には一つ制限があることに注意してください。伝統的にシリアル通信ポートの名前は固定であったため、Kermit の設定ファイルも決まった名前のシリアル・デバイスを参照します。ところが、最近主流の USB シリアル・アダプタの場合、いろいろな条件でデバイスの名前が変わります。

名前を決める重要な要素は、シリアル・デバイスとして何番目にシステムに認識されたかと言うことです。先のインストール・スクリプトでは、シリアルポートの名前は”/dev/ttyUSB0”に固定して使っています。USB シリアル変換アダプタを複数使う場合には、シリアル・ポートとして使うものを最初にシステムに挿すようにしてください。

Shell から

```
$ kermit
```

と、入力することで kermit を起動できます。起動直後はユーザーコマンドを待っていますので”c”コマンドで端末モードに移行してください。端末モードから戻るには、エスケープ・コマンド”CTRL-¥ C”を入力します。

図 4 にシリアル通信の様子を示します。

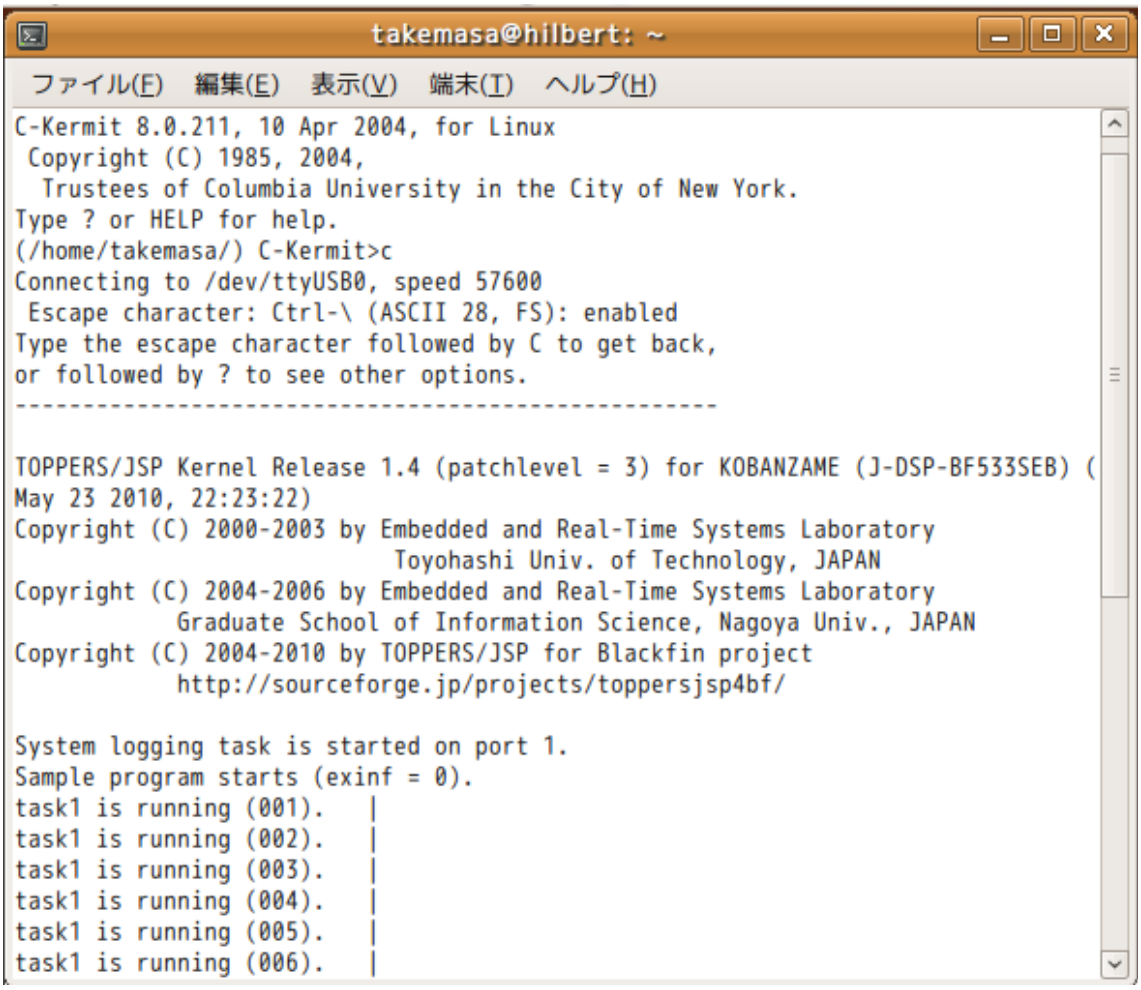

図 3 kermit でシリアル通信を行っているところ

独自アプリケーションを作る

TOPPERS/JSP for Blackfin のアプリケーション開発は、通常の TOPPERS/JSP アプリケーション開発と全く同じで、なにも特殊なところはありません。

この章は当面空白のままです。TOPPERS/JSP アプリケーションの作り方については、他の文献、たとえば Interface 誌 2010 年 5 月号の特集記事 7 章を参照してください。

1.18 Configure スクリプトの動作

1.19 LED 点滅アプリケーション

1.19.1 blink.c

1.19.2 blink.h

1.19.3 blink.cfg

1.19.4 コンフィギュレーションとビルド

U-BOOT

DAS U-BOOT は、組み込みマイクロプロセッサ用のブートローダーで、ボード上の ROM に格納して使うことが出来ます。このプログラムは Blackfin Koop の手で Blackfin DSP に移植されており、これを使うことで Blackfin を使った独自ボードから uClinux や TOPPERS/JSP を起動することが出来ます。

U-BOOT は ROM に格納して使うことが出来ます。プロセッサがリセットされると、最初に ROM 上の U-BOOT が起動します。U-BOOT は最小限のハードウェアを設定したあと、自分自身を RAM 上にコピーし、必要なペリフェラルを初期化します。

U-BOOT が初期化した UART や ETHERNET ペリフェラルを通して、ホストマイコンからプログラムのファイルをボードのメモリにダウンロード出来ます。ダウンロードされたファイルは実行することも可能ですし、そのまま Flash メモリに書き込むこともできます。

U-BOOT は JSSF2 のようなファイル・システムや、CF のようなリムーバブル記憶デバイスにも対応しており、使い方によってはかなり複雑なことも出来ます。

この章では、U-BOOT を使って Kobanzame に TOPPERS/JSP アプリケーションを書き込み、自動起動させる方法を説明します。

1.20 U-BOOT 上でのアプリケーションの実行

Kobanzame に電源を入れると、U-Boot が立ち上がります。黙ってみていると自動実行してしまいますので、スクリーン上でカウントダウンが始まったらキーボードの適当なキーを押してカウントダウンを止めます。

```
U-Boot 1.1.6 (ADI-2008R1.5) (Dec 22 2008 - 22:57:37)

CPU:   ADSP bf533-0.4 (Detected Rev: 0.4)
Board: BF533CB
       No Support: http://blackfin.s36.coreserver.jp/
Clock: VCO: 500 MHz, Core: 500 MHz, System: 125 MHz
RAM:   32 MB
Flash: 176 erase regions found, only 4 used
 0.1 kB
In:    serial
Out:   serial
Err:   serial
Hit any key to stop autoboot:  0
bfin>
```

U-BOOT がコマンド待ちになったら、早速何か試してみましょう。まずは、バイナリ・ファイルのダウンロードを試します。U-BOOT はシリアル回線越しにバイナリ・ファイルを受け取るためのプロトコルをいくつか用意しています。ここでは Kermit 転送を使ってみます。

U-BOOT に Y-MODEM プロトコルでバイナリ・ファイルを送るには、loadb コマンドを実行します。

```
bfin> loadb
## Ready for binary (kermit) download to 0x01000000 at 57600 bps...
```

loadb は、ファイルをメモリ空間に展開します。デフォルトでは loadaddr 環境変数で指定したアドレスに置きますが、直接指定することも可能です。

"C"の連打を確認したら、端末ソフト側でもデータの送信にかかります。Kermit では、コマンド・ラインからデータ送信を行います。端末モードからコマンド・ラインに切り替えるために Kermit に大して CTRL-¥ C をタイプします。Kermit のデータ送信コマンドは send です。なお、送信前に robust コマンドを実行しておいてください。

```
 (Back at hilbert)
-----------------------------------------------------
(/home/takemasa/) C-Kermit>robust
(/home/takemasa/) C-Kermit>send jsp
(/home/takemasa/) C-Kermit>c
Connecting to /dev/ttyUSB0, speed 57600
 Escape character: Ctrl-\ (ASCII 28, FS): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
-----------------------------------------------------
## Total Size      = 0x000425b1 = 271793 Bytes
## Start Addr      = 0x01000000
```

転送が終わったらKermitのcコマンドで、再び端末からU-Bootのコンソールに入ります。そうして、bootefl コマンドを実行してください。コマンドの引数は実行開始番地であり、この場合${loadaddr}とすれば、バイナリがロードされたアドレスになります。

```
 bootelf $(loadaddr)


 Loading .start @ 0xffa00000 (276 bytes)
 Loading .text @ 0xffa00114 (20592 bytes)
 Loading .init @ 0xffa05184 (60 bytes)
 Loading .fini @ 0xffa051c0 (40 bytes)
 Loading .rodata @ 0xff800000 (4916 bytes)
 Loading .ctors @ 0xff801334 (8 bytes)
 Loading .dtors @ 0xff80133c (8 bytes)
 Loading .data @ 0xff801344 (80 bytes)
 Clearing .bss @ 0xff801394 (7320 bytes)
 ## Starting application at 0xffa00000 ...

 TOPPERS/JSP Kernel Release 1.4 (patchlevel = 3) for KOBANZAME (J-DSP-
BF533SEB) (Jun 20 2010, 13:45:28)
 Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory
                            Toyohashi Univ. of Technology, JAPAN
 Copyright (C) 2004-2006 by Embedded and Real-Time Systems Laboratory
                Graduate School of Information Science, Nagoya Univ.,
JAPAN
 Copyright (C) 2004-2010 by TOPPERS/JSP for Blackfin project
             http://sourceforge.jp/projects/toppersjsp4bf/

 System logging task is started on port 1.
 Sample program starts (exinf = 0).
 task1 is running (001).   |
 task1 is running (002).   |
 task1 is running (003).   |
```

bootelfコマンドは、メモリ上のオブジェクトをその場で実行するわけではない点に注意してください。メモリ上にあるオブジェクトは ELF 形式のデータであると考えられ、bootelfコマンドはそのデータを解析して、指定されたアドレスに指定されたオブジェクトを次々と配置します。そうして、すべての配置が終わったら、ELF 形式のデータが指定する実行開始番地へとジャンプします。

1.21 アプリケーションの ROM 化

U-BOOT はブートローダーとして使用します。ですので、SRDRAM 上の ELF データを実行できるだけでは意味がありません。アプリケーション・プログラムを ROM に置き、U-BOOT 起動後に自動的にその ROM 上のプログラムを実行して、はじめて実用と言えます。

以下では Kobanzame 上の SPI FLASH に焼いたアプリケーション・プログラムを自動実行する方法を説明します。

1.21.1 ROM への書き込み

最初に SPI フラッシュにアプリケーション・プログラムを焼く必要があります。

まずは eeprom info コマンドで SPI FLASH の情報を見てみましょう。

```
bfin> eeprom info
SPI Device: m25p16 0x20 (ST) 0x20 0x15
Parameters: num sectors = 32, sector size = 65536, write size = 256
Flash Size: 16 mbit (2 mbyte)
Status: 0x00
bfin>
```

ちゃんと認識されています。

これによると、SPI FLASH は 2MB の容量があります。SPI FLASH のアドレス 0 から始まる領域には U-BOOT が入っていますので、アプリケーションを格納できるのは U-BOOT が入っていない部分だけです。おおざっぱに言って後半 1MB なら使って大丈夫でしょう。アドレスで言うと 0x100000 以降です。

なお、SPI FLASH はメモリ・マップされていませんので、ここで言うアドレスとは Blackfin のアドレス空間とは無関係です。

さて、上の節で行った操作をもう一度行い、SDRAM 上にデータをダウンロードしましょう。ダウンロード結果は次のようになります。

```
## Total Size      = 0x000425b1 = 271793 Bytes
## Start Addr      = 0x01000000
```

このデータは TOPPERS/JSP アプリの ELF データであり、そのまま SPI FLASH に書き込む必要があります。

書き込みは eeprom write コマンドを使います。このコマンドは、Blackfin メモリ空間上のスタート番地、SPI FLASH 内部のオフセット、書き込みデータの長さを引数として受け取ります。今回はそれぞれ${loadaddr}、0x100000、0x425b1 となります。

```
bfin> eeprom write ${loadaddr} 0x100000 0x425b1

EEPROM   @0x0   write:   addr   01000000      off   100000      count
271793 ... ..........done
bfin>
```

これで ROM への書き込みは終わりです。SPI Flash の 0x100000 から始まる領域にアプリケーションが書き込まれました。

1.21.2 ROM 上のアプリの実行

SPI FLASH へ書き込んだデータの実行は、ROM からのデータの取り出しと、その実行の 2 段階です。ROM からのデータの取り出しには eeprom read コマンドを使用します。このコマンドは、Blackfin メモリ空間上のスタート番地、SPI FLASH 内部のオフセット、読み込みデータの長さを引数として受け取ります。以下の例では読み込みデータの長さを 0x100000(1MB)に決め打ちしています。

```
bfin> eeprom read ${loadaddr} 0x100000 0x100000

EEPROM @0x0 read: addr 01000000 off 100000 count
1048576 ... ................done
bfin> bootelf ${loadaddr}
Loading .start @ 0xffa00000 (276 bytes)
Loading .text @ 0xffa00114 (20592 bytes)
Loading .init @ 0xffa05184 (60 bytes)
Loading .fini @ 0xffa051c0 (40 bytes)
Loading .rodata @ 0xff800000 (4916 bytes)
```

読み込み後に bootelf コマンドを実行すると、SPI FLASH に書き込んだアプリケーションを実行することができます。

1.21.3 起動時実行

SPI FLASH に焼き込んだプログラムをロードする方法がこれでわかりました。あとは、以上のシーケンスをリセット時に自動実行すれば完了です。

自動実行の指定は bootcmd 環境変数で行います。以下にその方法を示します。bootcmd 環境変数にはリセット時に自動実行するコマンド列を";"で区切って書き込んでおき、seveenv コマンドで SPI FLASH に環境変数を書き込みます。

```
bfin> setenv bootcmd 'eeprom read ${loadaddr} 0x100000 0x100000;
bootelf ${loadaddr}'
bfin> saveenv
Saving Environment to EEPROM...
..done
bfin>
```

これでリセット時に TOPPERS/JSP アプリケーションを ROM から起動できます。なお、bootcmd 環境変数への書き込みは一度だけ行えば結構です。

1.21.4 ELF ファイルの軽量化

上の節ではビルドした JSP アプリケーションをそのまま Kobanzame に転送しました。それでもちゃんと動くのですが、たかがサンプルアプリケーションにファイルサイズが 270KB とは、あまりにも大きすぎます。そこで、ファイルを軽量化します。

ELF ファイルにはデバッグに使うシンボル情報のほか、いくつかの補助情報が入っています。これらの情報は単純に実行する場合には無視してもかまいません。そこで、以下のように OBJCOPY コマンドを使ってファイルサイズを小さくできます。

```
$ ls -g -G jsp
-rwxr-xr-x 1 271793 2010-06-20 14:30 jsp
$ bfin-elf-objcopy --strip-all -R .comment jsp
$ ls -g -G jsp
-rwxr-xr-x 1 33768 2010-06-20 14:30 jsp
```

OBJCOPYを使った結果、ファイルサイズは34KBまで縮小しました。

1.22 ターゲットへ U-BOOT を焼く

この項、未完了。

おわりに

1.23 履歴

2012年7月8日 Blackfin 依存部 3.3.0 対応版 20120708

2012 年 6 月 18 日 Blackfin 依存部 3.3.0 対応版

2010 年 6 月 20 日 Blackfin 依存部 3.2.0 対応版。