

TOPPERS/JSP for Blackfin project

TJBN005

プログラムのデバッグ

TOPPERS/JSP for Blackfin プロジェクト

最終更新: 2012/Aug/27 Rev 1.0



この文書は [クリエイティブ・コモンズ 表示 3.0 非移植 ライセンス](https://creativecommons.org/licenses/by/3.0/)の下に提供されています。

1. GDB によるデバッグ

GNU ツールチェーンでビルドしたアプリケーション・ソフトウェアは GDB でデバッグすることができます。以下では GDB をつかって Blackfin 上の TOPPERS/JSP アプリケーションをデバッグする方法を説明します。

1.1. ツールバージョンなど

説明に使用するソフトウェアやハードウェアは以下のとおりです

- ホスト OS : Ubuntu 12.04 LTS 32bit (Windows 7 64bit 上の VMware Workstation 8 で検証)
- GNU Toolchain : Blackfin 2012R1 RC4
- JTAG ICE : gnICE+

ツールチェーンは、TOPPERS/JSP for Blackfin プロジェクトが供給しているスクリプトを通してインストールしていると仮定します。インストール方法については「TJBN002 ユーザーズマニュアル」を参照してください。

1.2. JTAG ICE

Blackfin のデバッグには JTAG ICE を使うことができます。JTAG ICE には

- Bluetechnix : gnICE+
- もなみソフトウェア : 刺身包丁

などがあります。以下では、gnICE+を使用する場合について説明します。

gnICE+は Blackfin Koop で紹介されている ICE であり、回路図等も公開されています。また、日本からは digi-key 経由で購入することができます。

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

この ICE は FTDI 社の USB シリアル変換 LSI を使用したもので、Linux や Windows から手軽に使うことができます。また、ARM 系 ICE と異なりターゲットからの電源供給を必要としません。

詳細は Blackfin Koop を参照してください。

1.3. GDB Proxy

gnICE+は Blackfin 専用ハードウェアであり、GDB から直接使うことができません。そこで、両者の仲立ちをする GDB Proxy が GNU Toolchain に含まれています。

GDB Proxy には、まず ICE をホスト PC に接続し、シェル・コマンドラインから以下のコマンドで起動します。

```
$ bfin-gdbproxy bfin
```

TOPPERS/JSP for Blackfin プロジェクト供与のインストーラを使用してインストールした場合には、起動は以上で終わりです。手作業でインストールした場合にはルート権限で起動する必要があるかもしれません。

正常に起動した場合には、図 1-1 のような表示が現れます。

```
takemasa@hertz: ~
takemasa@hertz:~$ bfin-gdbproxy -q bfin
Found USB cable: gnICE+
Connected to libftdi driver.
IR length: 5
Chain length: 1
Device Id: 00100010100000000010000011001011 (0x228020CB)
Manufacturer: Analog Devices, Inc. (0x0CB)
Part(0): BF592 (0x2802)
Stepping: 2
Filename: /opt/uClinux/bfin-elf/bin/./share/urjtag/analog/bf592/bf592
warning: bfin: no board selected, BF592 is detected
notice: bfin: jc: waiting on TCP port 2001
notice: bfin: jc: (you must connect GDB before using jtag console)
notice: bfin-gdbproxy: waiting on TCP port 2000
```

図 1-1: GDB Proxy

表示される情報の中には、ターゲット(この場合にはプロセッサ)の JTAG 情報や、プロキシのポート番号情報が含まれます。

最後の行に GDB プロキシが GDB による接続をポート 2000 番地で待っていることが表示されています。

1.4. GDB

GDB はツールチェーンに含まれているため、そのまま呼ぶことができます。

ターゲット DSP の製品名を指定する必要はありません。単に引数としてデバッグしたいアプリケーションのファイル名を指定します。TOPPERS/JSP の場合、アプリケーションのファイル名は常に JSP です。ファイルフォーマットは ELF です。

起動時には、待状態にある GDB Proxy と接続し、ターゲットをリセットしてアプリケーションをターゲットの SRAM にロードしなければなりません。これら一連の作業は面倒なので、.gdbinit ファイルに格納してスクリプトとして動かすことをお勧めします。

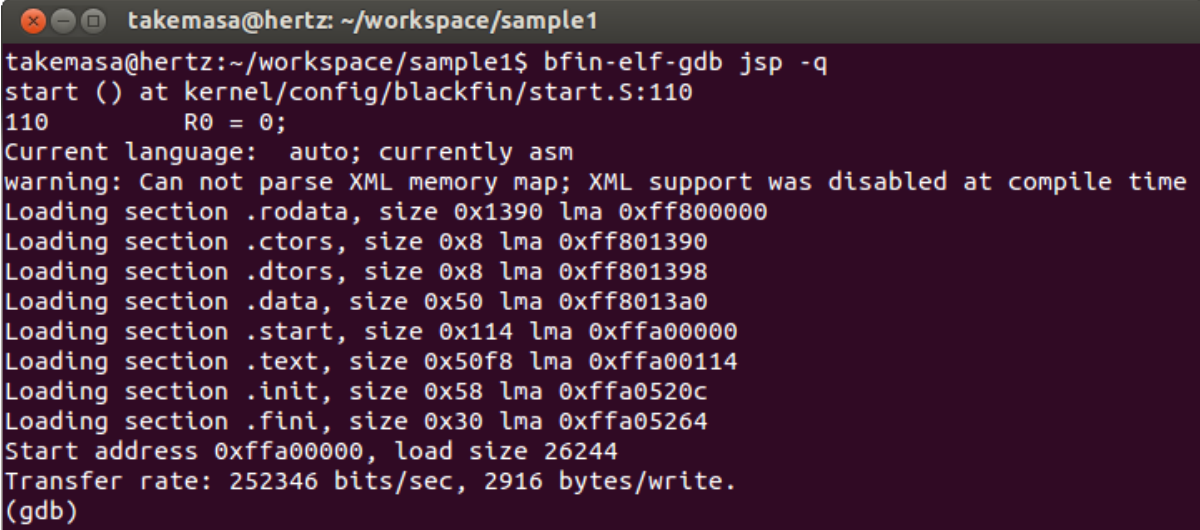
以上の動作を行うスクリプトは次のようになります。

```
target remote localhost:2000
set *( unsigned short * ) 0xFFC00100 = 0x07
set *( unsigned short * ) 0xFFC00100 = 0x00
load
```

最初の行はデバッグターゲットとして localhost (自アドレス) のポート 2000 番地を選ぶよう指示しています。このポートでは GDB Proxy が待機中ですので、結果的に GDB Proxy 経由でターゲット DSP と接続することになります。

次の行で 0xFFC0 0100 番地にある SWRST レジスタを通し、Blackfin をリセットしています。このリセットはシステムリセットであり、コアはリセットしませんがそれで構いません。続く行で 0x00 を SWRST に書き込んでいますが、これはリセット・シーケンスで要請されていることです。詳しくは Blackfin Processor Programming Reference の SWRST レジスタに関する説明を参照してください。

GDB の起動コマンド名は bfin-elf-gdb です。図 1-2 に GDB 起動時の状態を示します。



```
takemasa@hertz: ~/workspace/sample1
takemasa@hertz:~/workspace/sample1$ bfin-elf-gdb jsp -q
start () at kernel/config/blackfin/start.S:110
110      R0 = 0;
Current language:  auto; currently asm
warning: Can not parse XML memory map; XML support was disabled at compile time
Loading section .rodata, size 0x1390 lma 0xff800000
Loading section .ctors, size 0x8 lma 0xff801390
Loading section .dtors, size 0x8 lma 0xff801398
Loading section .data, size 0x50 lma 0xff8013a0
Loading section .start, size 0x114 lma 0xffa00000
Loading section .text, size 0x50f8 lma 0xffa00114
Loading section .init, size 0x58 lma 0xffa0520c
Loading section .fini, size 0x30 lma 0xffa05264
Start address 0xffa00000, load size 26244
Transfer rate: 252346 bits/sec, 2916 bytes/write.
(gdb)
```

図 1-2: GDB 起動時の表示

この状態からプログラムを実行するには C コマンドを発行してください。直ちに実行が始まります。ブレークポイントや変数の表示に関しては、GDB のマニュアルを参照してください。

1.5. MMR へのシンボル・アクセス

Blackfin アーキテクチャでは一部のコア・レジスタを覗いて、多くのコア・コントロール・レジスタや全てのペリフェラル・レジスタをメモリ空間に配置しています。これを MMR(Memory Mapped Registers)と呼びます。

MMR を採用したため、Blackfin の制御は高級言語からも行いやすくなっています。一方で、アセンブリ言語を始めとするツールはそれらのレジスタ・アドレスを知りません。当然デバuggも知りません。何らかの方法でデバuggにペリフェラル・アドレスを教える必要があります。

TOPPERS/JSP for Blackfin プロジェクトは、一部の Blackfin DSP 用に MMR のシンボル・ファイルを提供しています。これらのファイルは genmmrsyms としてプロジェクトからダウンロード可能です (git リポジトリは util-genmmrsyms)。

対応している DSP は以下のとおりです。

- ADSP-BF533
- ADSP-BF537
- ADSP-BF518
- ADSP-BF592

TOPPERS/JSP for Blackfin project

これらの DSP 用にシンボルファイルを生成するには、ソースコードを展開したディレクトリの中で、Make を実行してください。mmr_bf*.out ファイルが生成されますので、自分が使用するプロセッサの名前の.out ファイルを GDB から読み込んで使います。例えば ADSP-BF592 に関するシンボル・ファイルを読み込むには、次のコマンドを gdb 上で実行します。

```
(gdb) Add-symbol-file mmr_bf592.out 0
```

MMR の名前にはすべて接頭語”mmr”がついています。例えば、IPEND は mmrIPEND、SIC_IAR0 は、mmrSIC_IAR0 です。

MMR の名前は変数名として GDB に認識されます。ですので、以下のようにして MMR に読み書きできます。

```
(gdb) print/x mmrIMASK  
(gdb) set mmrIMASK = 0
```

2. Eclipse によるデバッグ (Ubuntu 12.04)

Eclipse の GDB 連携機能を使うと、Eclipse IDE 上で Blackfin のデバッグを行うことができます。この章では、Ubuntu 12.04 にパッケージされている Eclipse Indigo について説明します。なお、ツール類のインストーラーは TOPPERS/JSP for Blackfin プロジェクト供与のインストーラーを使っていると仮定します。

インストーラーの使い方については「TJBN-002 ユーザーズ・マニュアル」を参照してください。

2.1. GDB Hardware Debugging プラグインのインストール

TOPPERS/JSP for Blackfin が提供するインストーラーは、Ubuntu が提供しているパッケージをインストールするだけです。その後の構成はユーザーが行う必要があります。

組み込み開発環境のデバッグでは、GDB Hardware Debugging プラグインをインストールする必要がありますが、これは Ubuntu によるパッケージ管理対象外なので、ユーザーがインストールします。

GDB Hardware Debugging プラグインをインストールするには、まずメニューバーから Help → Install new software... を選んでください。Install ダイアログが現れますので、Work with:プルダウンから”Indigo Update Site”を選びます。次にフィルタ文字列として”hard”をタイプします。しばらくすると”C/C++ GDB Hardware Debugging”プラグインが現れますので、チェックしてインストールしてください(図 2-1)。

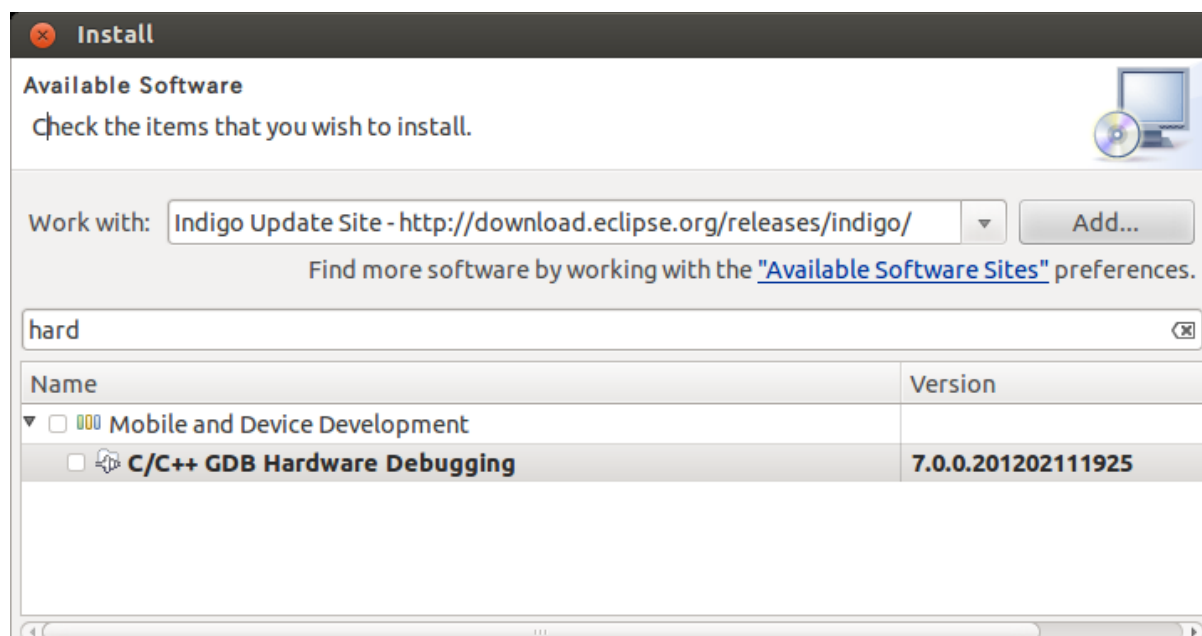


図 2-1: GDB Hardware Debugging プラグインのインストール

2.2. デバッグ・セッションの設定

デバッグをはじめするには、デバッグセッションの設定をしなければなりません。この設定で、ターゲットの情報や、デバッガの種類、初期動作などを指定します。

プロジェクトを選択し、メニューバーから Program → Debug Configurations... を選択します。ダイアログが現れたら、“GDB Hardware Debugging”を選択し、新しい設定を作ってください。図 2-2 の例では“sample1 Default”が、新しく作った設定です。このとき、“C/C++ Application:”欄が JSP になっていることを確認します。この欄はターゲットにダウンロードする ELF ファイルを指定します。

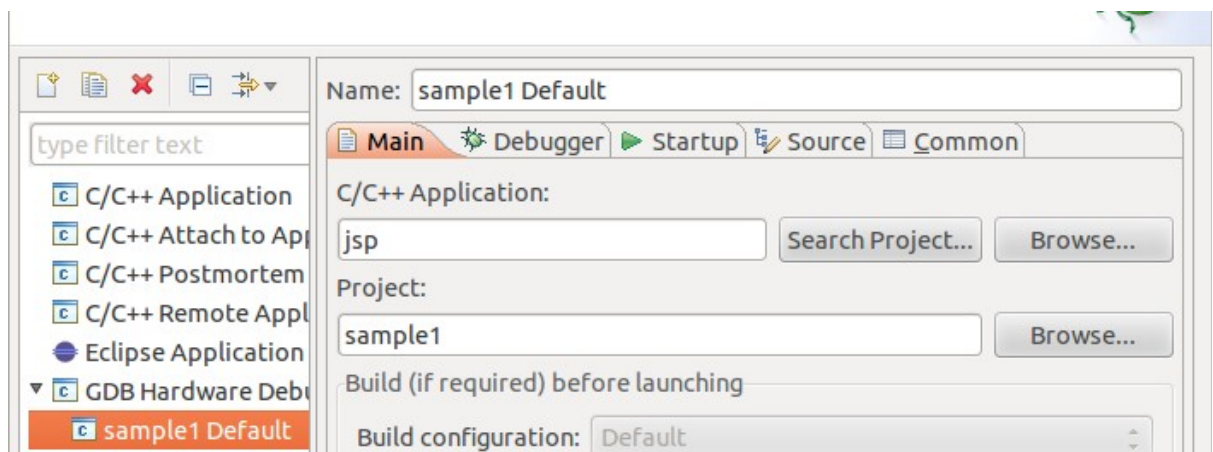


図 2-2: デバッグ・アプリケーションの指定

次に Debugger タブをクリックします(図 2-3)。

このタブでは、使用するデバッガと、その接続先を指定します。使用するデバッガは bfin-elf-gdb です。デフォルトでは gdb となっていますが、これは x86 の GDB ですので気をつけてください。ターゲットは localhost のポート 2000 を指定します。

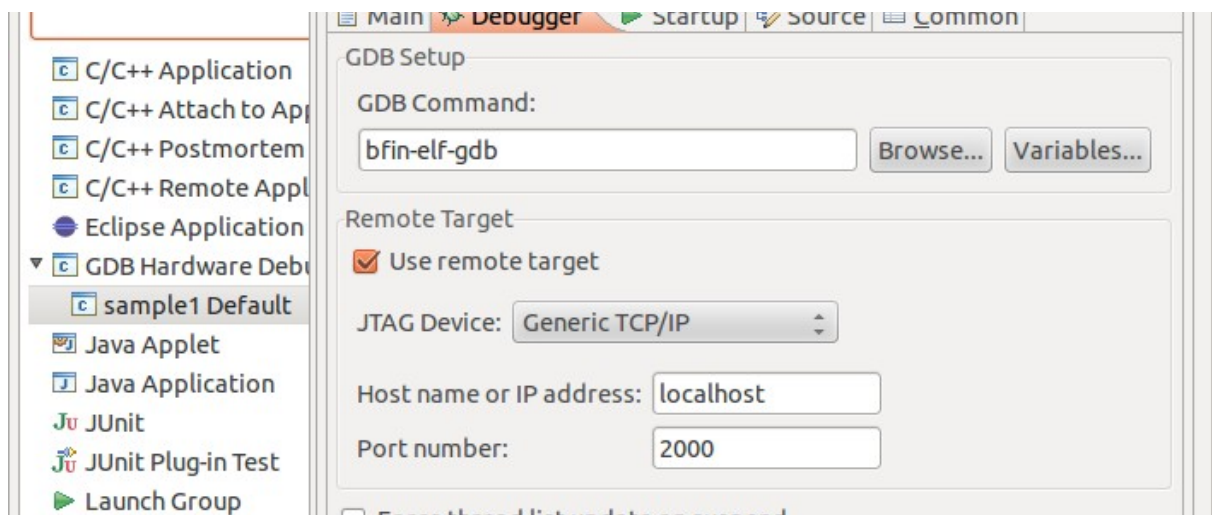
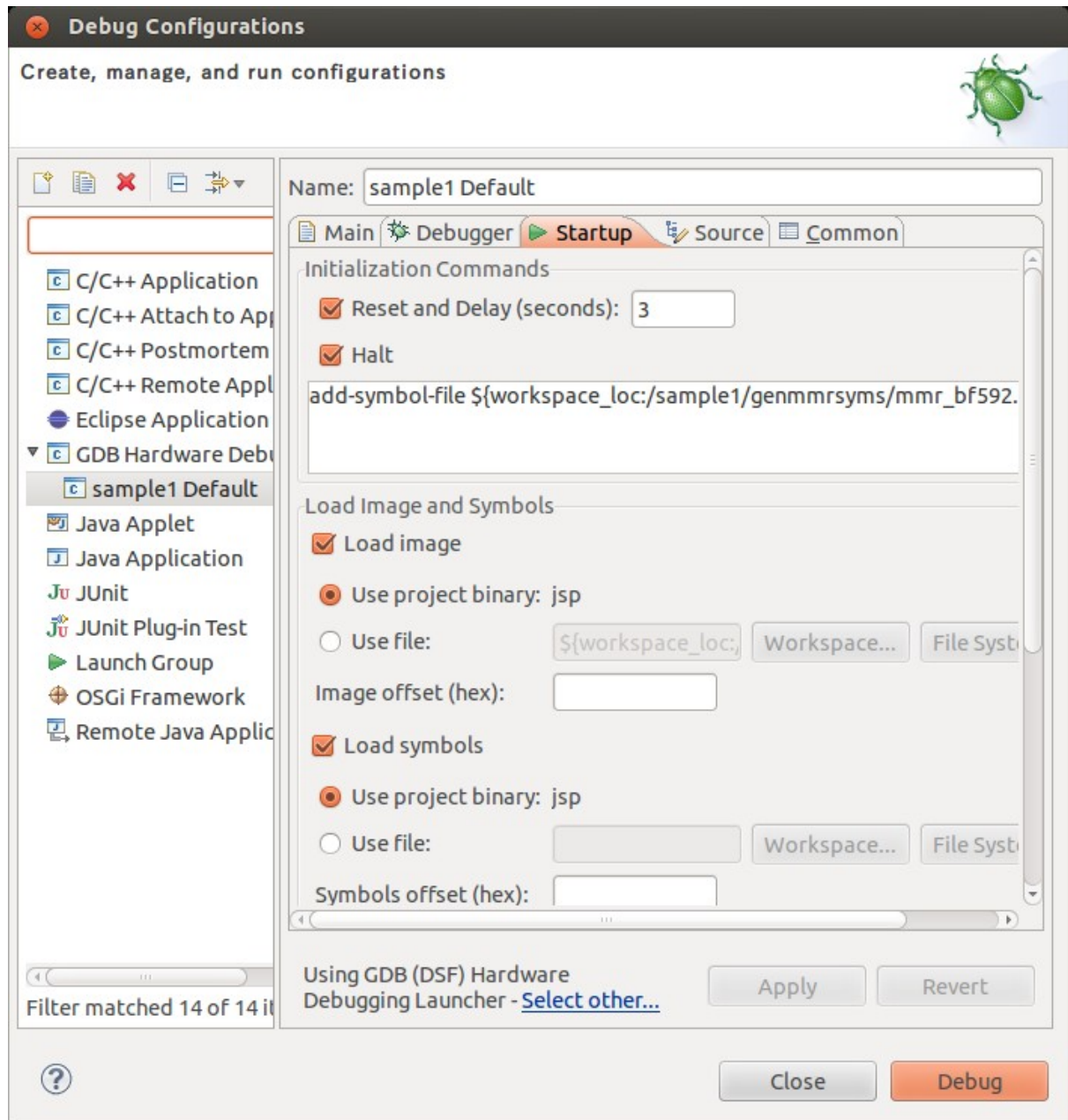


図 2-3: Debugger タブ

次に Startup タブをクリックします(図 2-4)。

TOPPERS/JSP for Blackfin project

このタブでは、デバッグとの接続直後に行う処理を指定します。このタブに関しては、幾分手探りのな面もありますが、図を参考にして設定してみてください。カスタム的に与えているコマンドは、MMR アドレスを読み込むためのものですが、後で説明するようにこれはありません。



2.3. Eclipse でペリフェラル・レジスタを読む

すでに説明したように、genmmrsyms プロジェクトが生成する実行ファイルのシンボルを読み込むと、Blackfin の MMR を変数としてデバッガから読み書きできます。この方法は Eclipse でも通用します。一方、Blackfin Koop は GDB ではなく Eclipse そのものにペリフェラル・レジスタへのアクセス方法を追加するプラグインを公開しています。このプラグインを使えば、genmmrsyms は不要です。

この節ではそのプラグインを利用する方法を説明します。

Blackfin 対応プラグインは、GUI からインストールできます。まず Eclipse のメニューバーから Help->Install New Software... を選びます。ダイアログが現れますので、Work with フィールドに <http://blackfin.uclinux.org/eclipse/> を入力してください。しばらくすると下のリストにインストール可能なフィーチャーが現れますので、Memory Mapped Registers View Feature を選んでインストールします(図 2-4)。

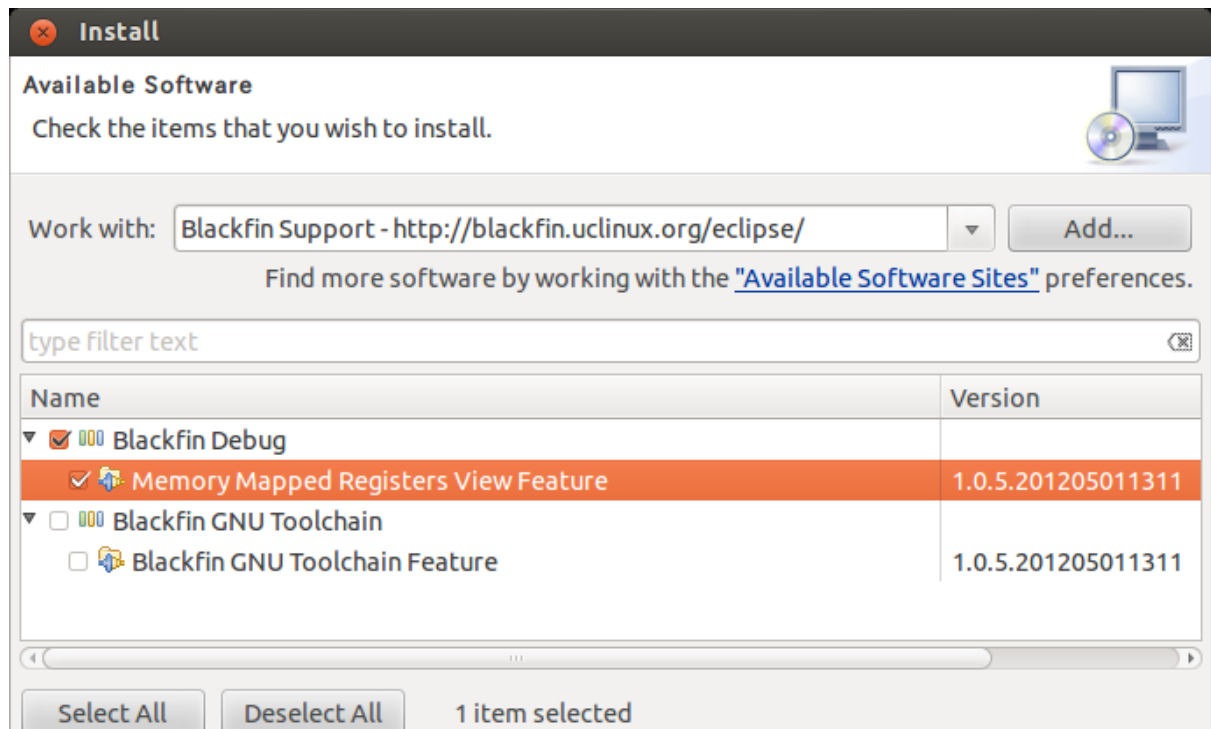


図 2-4: Eclipse への Blackfin サポートのインストール

インストールが終了し、Eclipse を再立ち上げしたら、デバッグと接続してください。接続が完了したらメインメニューの Window → Show View → Others... を選び、ダイアログで Memory と打ち込んで検索します(図 2-5)。Memory Mapped Register が現れますので、選択して表示してください。

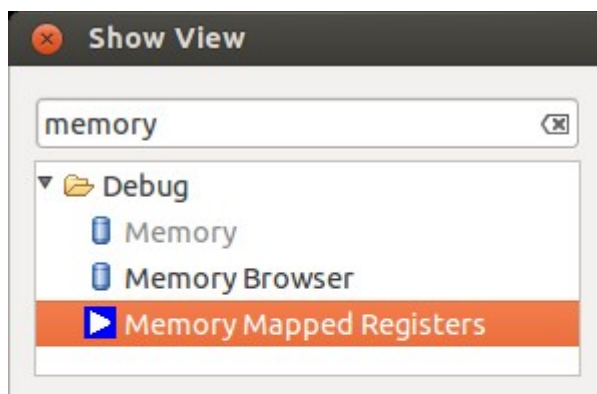


図 2-5: *Memory Browser* の表示

ウインドウ内でのレジスタ表示は、右クリックしてメニューでおこなってください。

3. TIPS

以下、簡単な TIPS を列挙します。

- 実行を開始しても、kermit ターミナルに文字が現れないときには、USB ICE と USB シリアルポートの接続順を疑ってください。インストール直後の状態では、USB シリアルポートが/dev/ttyUSB0 でなければ、正常動作しません。
- ターゲットにプログラムをロードする前、あるいは直後には必ずターゲットのリセットをおこなってください。

4. 文献・履歴など

4.1. 参考文献

- Blackfin Processor Programming Reference
- <http://docs.blackfin.uclinux.org/doku.php?id=toolchain:eclipse:install>

4.2. 履歴

- 2012/Aug/27 : Rev 1.0