PowerPC を用いたハードウェアシステム用の GDB サーバ

6 Z − 4

河内谷 清久仁　　森山 孝男

日本アイ・ビー・エム（株） 東京基礎研究所

1 はじめに

PowerPC [1] を用いたハードウェアのデバッグを行うツールとして Engineering Support Processor（ESP）と呼ばれる装置が存在する．ESP は，JTAG（IEEE 1149.1）ポートへのインタフェースとバッファを備えたハードウェアと，それを制御するための AIX 用のソフトウェアから構成されている．これにより，ターゲットシステム上の PowerPC のレジスタやメモリへのアクセス，プログラムのダウンロードと起動，停止，ブレークポイントの設定などが行える．ユーザインタフェースとしては，X-Windows 上の GUI ベースのものが利用できるほか，EZsockets と呼ばれるプログラミングインタフェースも用意されており，これによりネットワークごしにプログラムから制御を行うことも可能である．

ESP により，ターゲットシステム上のプログラムをマシン語レベルでデバッグすることはできるが，C などの高級言語で書かれたプログラムをソースコードレベルでデバッグすることはできない．そのような目的には，GNU デバッガ（GDB）[2] のようなシンボリックデバッガが必要となる．GDB ではすでにターゲット CPU として PowerPC がサポートされており，PowerPC 用プログラムのシンボリックデバッグが行える．しかし，GDB 自身を安定していないターゲットシステム上で動かすのは困難である．このような，ハードウェア設計時のデバッグのために，GDB にはリモートデバッグ機能が用意されている．これにより，シリアルラインもしくはネットワークごしにターゲットシステム上のプログラムのデバッグを行うということが可能となる．この際のプロトコルとしては，GDB 独自の GDB リモートプロトコル（GDB remote serial protocol）[1]が使用される．

我々は今回，このリモートデバッグ機能と ESP の EZsockets インタフェースとを橋渡しするプログラム（GDB サーバ/ESP）を作成した [3]．これにより，GDB は安定したホストシステム上で動作させ，ESP ごしにターゲットシステムのシンボリックデバッグを行うことが可能となる．本稿ではこのデバッグ環境について，実装上の考慮点を中心に概要を述べる．

2 GDB サーバ/ESP の実装

図 1 は，GDB サーバ/ESP を用いたデバッグ環境の構成を示したものである．ホスト GDB は，PowerPC 用にビルドされた通常の GDB である．ただし，ネットワークごしのリモートデバッグを可能にするため，構成ファイルの変更を行っている．GDB リモートプロトコルによるホスト GDB からのデバッグ指令は，GDB サーバにより ESP に対するコマンド群に変換される．変換された命令は，EZsockets インタフェースを通じて ESP に送られ処理される．なお，GDB サーバからの命令処理と並行して通常の GUI による ESP の操作も可能で，これにより GDB がサポートしていないスペシャルレジスタの操作などを行うこともできる．

GDB サーバ/ESP の実装は，GDB パッケージに含まれる GDB サーバ・スケルトンに，ESP にアクセスする下位部分をリンクする形で行われている．この下位部分（`low-esp.c`）が今回の開発の中心部分である．ここで用意すべき機能の一覧を

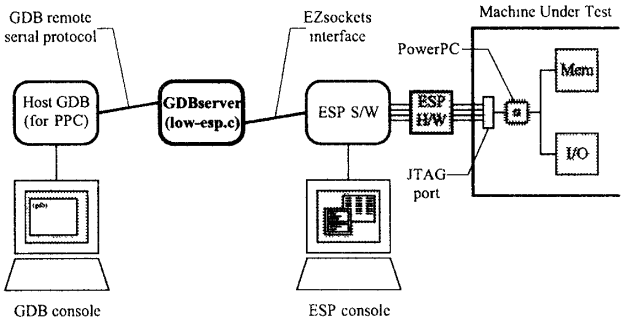

図 1: GDB サーバ/ESP を用いたデバッグ環境

表 1 に示す．これらの機能を ESP ごしに実現する関数を用意することで，ホスト GDB からのシンボリックデバッグが可能となる．以下の節では，実装時に注意が必要となった部分を中心に述べる．

2.1 対象プログラムの生成

対象プログラムの生成は，`create_inferior()` で行われる．その大まかな流れは以下の通りである．

1. ESP のロード（`load`）コマンドにより対象プログラムをダウンロードする．このコマンドでは，XCOFF ヘッダ中に記述されたアドレス情報に従い各セクションを配置する．
2. 対象プログラムの XCOFF ヘッダを解析し，エントリポイントと TOC（Table of Contents）アドレスを得る．
3. GPR1（スタックポインタ），GPR2（TOC レジスタ）に値を設定する[2]．
4. MSR に `0x00000040` を設定する．これによりステップ 7 でのリセット時の飛び先が `0xfff00100` になる．
5. エントリアドレスへのブランチ命令（`ba entry`）を生成し，システムリセットのベクタアドレス `0xfff00100` に書き込む．
6. ESP のハードウェアブレークポイントをエントリアドレスに設定する．
7. ESP のターゲットリセット（`iplrun`）コマンドにより，PowerPC のソフトリセット処理を行う．
8. ステップ 6 で設定したブレークポイントにより，対象プログラムはエントリポイントで停止する．

以上の手順により，対象プログラムをダウンロードし，最初の命令で停止させている．なお，対象プログラムに対する引数の受渡しはサポートしていない．

2.2 対象プログラムの再開

対象プログラムの再開は，`myresume()` で行われる．ここでの処理は，以下の通りである．

A GDB Server for PowerPC-Based Hardware
Kiyokuni KAWACHIYA and Takao MORIYAMA
IBM Research, Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato, Kanagawa 242, Japan
E-Mail: <kawatiya@trl.ibm.co.jp>

[1] このプロトコルの概要は GDB のソースファイル `remote.c` 中のコメントに記述されている．

[2] スタックポインタの初期値はターゲットシステムに依存するが，デフォルトでは，`0xfffffff4` を設定している．これは，PowerPC の標準的なリンケージコンベンション [4] ではスタックポインタから上位 3 ワード分のアドレスは呼び出された側の関数で使用される可能性があるためである．

表 1: GDB サーバ下位部分（low-esp.c）が提供する関数

Name	Description
create_inferior()	対象プログラムの生成
kill_inferior()	対象プログラムの終了
mythread_alive()	対象プログラムの生存確認
interrupt_inferior()	対象プログラムの強制停止
mywait()	対象プログラムの停止待ち合わせ
myresume()	対象プログラムの再開
registers[]	停止時のレジスタ保存用エリア
fetch_inferior_registers()	レジスタの読み出し
store_inferior_registers()	レジスタの書き込み
read_inferior_memory()	メモリの読み出し
write_inferior_memory()	メモリの書き込み

1. ESP のハードウェアブレークポイントをプログラム例外のベクタアドレス 0xfff00700 に設定する.
2. ESP の実行再開（run 0）コマンドにより，実行を再開する.

GDB では，実行されるコード中にトラップ命令を挿入することでブレークポイントを実現している．トラップが起こると，PowerPC の処理はプログラム例外のベクタアドレス 0xfff00700 へ移る．ステップ 1 で設定したハードウェアブレークポイントにより，ここで対象プログラムを停止させることができる.

2.3 対象プログラムの停止待ち合わせ

対象プログラムが停止するまで，GDB サーバは mywait() 内でブロックする．対象プログラムを停止させるには 2 つの方法がある．一つは前述のトラップ命令によるもの，もう一つはホスト GDB からの割り込み（通常は Ctrl-C で可能）によるものである．割り込みがあった場合，GDB サーバには非同期 I/O シグナルにより通知され，シグナル処理コンテクスト中で interrupt_inferior() が呼び出される．この関数では，割り込みがあったというフラグを立てるだけで，実際の停止処理は mywait() 内でこのフラグをチェックして行っている.

mywait() での処理の流れは，以下の通りである.

1. 対象プログラムが停止するまで以下のループを実行する.
   a. interrupt_inferior() が呼ばれ割り込みフラグが立てられている場合，ESP の実行停止（stop）コマンドにより対象プログラムを停止させる.
   b. ESP のステータスチェック（wait -v）コマンドにより，対象プログラムの停止チェックを行う．停止していた場合はループを抜けステップ 2 に進む.
   c. 1 秒間スリープした後，ステップ 1a に戻る.
2. IAR（プログラムカウンタ）レジスタをチェックし，トラップハンドラ内（0xfff00700）で停止している場合は以下の処理でトラップハンドラを抜けるまで処理を進める.
   a. ESP のハードウェアブレークポイントをトラップを起こしたアドレス（SRR0 にセットされている）に設定する.
   b. 0xfff00700 に rfi 命令を書き込み，ESP の実行再開（run）コマンドで対象プログラムの処理を再開する.
   c. ステップ 2a で設定したブレークポイントにより，対象プログラムはトラップを起こしたアドレスで停止する.
3. プロセッサのキャッシュが ON になっている場合には，キャッシュのフラッシュ処理を行う．この部分の詳細は次節で述べる.
4. 停止の原因（SIGTRAP もしくは SIGINT）をセットして戻る.

ステップ 2 でトラップハンドラ内からの脱出処理を行っているのは，停止時のレジスタなどの状態としてチップの状態を直接使えるようにするためである．これにより，ESP から直接デバッグ操作を行う場合の混乱も回避できる.

2.4 停止時のキャッシュフラッシュ

ESP からのメモリアクセスは，CPU 内のデータキャッシュ及び命令キャッシュをバイパスして行われる．そのため，ホスト GDB からのメモリ読み書き要求を正しく処理するには，あらかじめキャッシュのフラッシュ（書き戻しと無効化）を行っておかなければならない．ESP にはキャッシュをフラッシュする機能がないため，キャッシュが ON になっている場合には，キャッシュフラッシュのためのプログラムを書き込み，PowerPC 自身に実行させるという処理を行っている．この処理は mywait() 内で，対象プログラムの停止直後に毎回行われる（前節のステップ 3）.

このプログラムでは，ターゲットシステムに搭載されている物理メモリ全体について順次データキャッシュのフラッシュ（dcbf 命令）と命令キャッシュの無効化（icbi 命令）[3]を行う．プログラムの書き込み先としては，トラップハンドラ用のエリアである 0xfff00710 からの十数バイトを利用している.

2.5 メモリアドレスの変換

ESP からのメモリアクセスは，物理アドレスで行われる．そのため，対象プログラムがアドレス変換を利用している場合，同様の変換を GDB サーバ内でソフト的に行う必要がある．この処理は，read_inferior_memory() と write_inferior_memory() で必要となる．アドレス変換の内容はターゲットシステム及び対象プログラムに依存するが，今回作成したデバッグ環境を必要とするターゲットシステムの多くでは，アドレス変換を用いないか，使用するとしても I/O 空間のマッピングなどごく一部の固定的な変換だけであると考えられる．そのため今回の実装では，このような固定的な変換だけをあらかじめ登録しておくことで対処している.

メモリアクセスの際に BAT レジスタやページテーブルの情報を読み出し，GDB サーバ内でアドレス変換をシミュレートすることで本格的に対応することも不可能ではないが，処理の負荷が大きくなるので採用していない.

3 まとめ

本稿では，PowerPC を用いたハードウェアシステムでシンボリックデバッグを可能とする環境について述べた．このデバッグ環境は，JTAG インタフェースごしにチップにアクセスする ESP と呼ばれる装置とホストとなる GDB を仲介する，GDB サーバと呼ばれるプログラムにより実現されている．GDB リモートプロトコルによるホスト GDB からのデバッグ指令は，GDB サーバにより ESP のコマンド群に変換し処理される．これにより，デバッグ環境は安定したホストシステム上で動作させ，ESP ごしにターゲットシステムのシンボリックデバッグを行うことが可能となる.

参考文献

[1] IBM: *The PowerPC Architecture*, Morgan Kaufmann Publishers, Inc. (1994).

[2] R. M. Stallman and Cygnus Support: *Debugging with GDB version 4.16*, Free Software Foundation (1994).

[3] K. Kawachiya and T. Moriyama: "A Symbolic Debugger for PowerPC-Based Hardware, Using the Engineering Support Processor (ESP)," IBM Research Report, RT0212, IBM (1997).

[4] IBM: *AIX Version 3.2 Assembler Language Reference*, Chapter 5, IBM Manual SC23-2197-02.

[5] IBM Microelectronics and Motorola: *PowerPC 604 RISC Microprocessor User's Manual.*

[3] PowerPC 604 [5] の場合，HID0 レジスタの Bit 20 で，命令キャッシュの内容の無効化の制御が行えるので，こちらを利用している.