第2部 実践編

Appendix

Insightでデバッグできるようにする

ARM付録基板用GDBスタブの作成

山際 伸一

1 デバッガとは何か

● 表示機器がない組み込み機器でのデバッグ

組み込み用にプログラムを書いて，実機で検査をしたものの，期待どおりに動作しないことがあります．バグを修正しようとしても，組み込み機器ではでディスプレイなどの表示機器が装備されていないことが多く，かりに装備されていたとしてもLEDくらいのものです．このような目隠し状態でプログラムの不具合を直すのはたいへんな作業となってしまいます．

そこで，デバッガというツールがあります．デバッガというと，デスクトップ用Windowsなどではソフトウェアだけで実装されたデバッガがありますが，組み込み向けには「ソフトウェア＋通信機器などのハードウェア」といった，ハードウェアを組み合わせたデバッガが用意されています（第2章参照）．

● ホストからターゲットをデバッグする

組み込み向けデバッガはターゲットとなる組み込み機器に何かしらの通信ケーブルを介して接続され，プロセッサからの応答を表示します．ここで考えるデバッガではシリアル・ケーブルを使うので，**図1**のような構成を取ります．

ホスト・マシンとは，デバッガの動作するパソコンなどのマシンのことで，組み込み機器とは独立した環境です．

デバッガはプログラムに発生する「イベント」をきっかけにして，プログラマにそのイベントが発生したことを通知します．イベントとは，

- プロセッサに外部割り込みが発生した
- プロセッサが異常な状態に陥った
- あるレジスタの値が期待したものになった

などが該当します．

● 三つのイベント

これらのイベントをデバッガに通知させるために，ブレークポイント，ウォッチポイント，キャッチポイントというイベントをデバッガに設定し，事象を待つことになります．これら三つの機能は以下のように働きます．

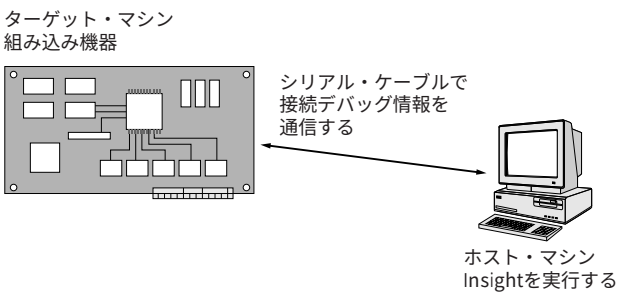

図1 デバッグにおける組み込み機器とホスト・マシンの関係

(1) ブレークポイント

実行中のプログラムを，ある番地で停止させたい場合に使います．プログラム・カウンタがある番地を指したときに停止します．

(2) ウォッチポイント

レジスタ，メモリなどのデータがプログラマの期待する値に変化したときにプログラムが停止し，そのときの状態をデバッガに返すのがウォッチポイントです．この機能は条件式をデバッガに与えることで実現しています．たとえば，変数aの挙動を見たい場合，「`a!=10`」とすると，aが10以外になったときにプログラムが停止し，デバッガがプログラムのその時点での状態を表示します．

(3) キャッチポイント

C++をサポートするデバッガではこのキャッチポイントを使って，`catch`文でのイベントの発生時にプログラムを停止することができます．本特集ではCを前提としているので，説明はこの程度にとどめます．

● デバッガの機能を実現するための実装

これらの機能を実現するデバッガの実装としては，ハードウェアで支援するデバッガとモニタ・プログラムが支援するデバッガの2種類あります．この章で扱うデバッガは，後者のモニタ・プログラム方式を用います．

デバッガが動作するよりも先に，モニタ・プログラムと呼ばれるデバッガとのみ通信が可能な小さなプログラムを組み込み機器に書き込んでおき，デバッガをそのプログラムと通信することで，その後にダウンロードされるアプリケーション・プログラムをデバッグします．

この方式の場合，事前に専用のモニタ・プログラムが動いているため，プロセッサの起動直後に起こる不具合などが発見できないというデメリットがありますが，すべてのプロセッサで可能な方式であるため，広く受け入れられています．

2 Insightでのデバッグのしくみ

● InsightとGDBリモート・シリアル・プロトコル

InsightはデバッガであるGDBのGUIフロントエンドです．Insightは接続された組み込み機器で実行中のプログラムと通信して，状況の報告とInsightからの実行手順制御を行うことができます．前述のブレークポイント，ウォッチポイント，キャッチポイントを設定する機能がありますが，それらの設定の際には実行中のプログラムと通信して，設定したイベント発生時にプログラムが再度，Insight側に状況報告をするようなしくみが必要になります．

このしくみはGDBの開発者たちが標準化し，文書化されています．組み込み機器で実行中のプログラムが含まなければいけない通信プロトコルと，作業手順が規定されており，通信プロトコルは「GDBリモート・シリアル・プロトコル」と呼ばれています．

● スタブがプロトコルを解釈して GDB と通信する

組み込み機器で動作する，GDB リモート・シリアル・プロトコルを解釈するソフトウェアのことをスタブ（stub）と呼びます．スタブを作成しておけば，どのような C 言語プログラムでも Insight でのデバッグが可能になります．

● スタブのしくみ

スタブは例外ハンドラとして実装されます．つまり，すべてのデバッグ用のイベントは例外として発生させます．たとえば，ブレークポイントは，停止させたい番地に未定義命令を書き込み，その番地を実行しようとしたプロセッサが未定義命令例外を発生し，その例外ハンドラであるスタブを実行して，Insight と通信を行い，情報交換をするというしくみになっています．

ここで注意しなければならないことが一つあります．Insight では例外ハンドラを利用していることから，例外ハンドラ自体のデバッグができないという問題があります．また，プログラムのダウンロードから実行まで Insight で可能ですが，その代償として，スタブを含んだモニタ・プログラムの動作環境における制限が付きまといます．すなわち，例外ハンドラまでを含んだプログラムを Insight でデバッグ可能にするためには，スタブも含んだプログラムを作成し，Insight を使ってダウンロードしなければなりません．さらに，ブレークポイントをサポートするために後述する未定義命令例外が使えないといった制限があります．

それでは，スタブはどのようになっているかを見てみましょう．

● GDB スタブの内容

スタブのほとんどの作業としては Insight から受け取った指示を理解し，その指示に従った情報や制御を行うことです．この Insight からの指示，または，Insight への応答は GDB リモート・シリアル・プロトコルで規定されています．まずは，この通信プロトコルから理解します．

● GDB リモート・シリアル・プロトコルの基本

リモート・シリアル・プロトコルは，GDB ドキュメントに規定されています．

```
http://sources.redhat. com/gdb/download/
                        onlinedocs/gdb_33.html#SEC655
```

GDB リモート・シリアル・プロトコルは，ターゲットとホストの間でやり取りされる通信データのルールです．今回の例では，（Insight から指示のあった）イベント発生時に組み込み機器側で発行され，ホスト・マシン上の Insight と通信します．

厳密には Insight ではなく，そのコア・プログラムである GDB がこのプロトコルで組み込み機器で実行されているスタブと交信します．

筆者の印象としては，プロトコルの歴史が古いわりに，雑に定義されていると感じます．これは後で説明しますが，GDB が実際のところプロトコルの一部分しか使っていないということに起因しているものと思われます．

したがって，複雑なもの（たとえば後述のクエリ）は「ああ，こういうのもあるんだ」くらいで頭の片隅にとめておくだけでよいと思います．

GDB リモート・シリアル・プロトコルを理解するうえで，パケット形式とパケットを形成するコマンドが重要になるので，それぞれ説明していきます．

● パケット形式

GDB は，一定のデータの塊をターゲット・システムとの間で交換し，スタブに要求を伝え，また，スタブから返答をもらいます．この単位となるデータ形式を「パケット」と呼びます．すべての通信はこのパケットを元にして行われます．

パケットは，

```
$パケット・データ#チェック・サム
```

という文字列から形成されます．パケット・データはリクエスト・コマンド，または返答をなす一つ以上の文字列で，チェック・サムは $ を含み，# までの ASCII 文字を，16 進数に変換した値の単純和をとった際の下位 8 ビットを 2 桁の 16 進で表します．

ここで，データの流れる方向に従い，パケットに名前を付けようと思います．GDB からターゲット・システムに送られるパケットを「リクエスト・パケット」，その反対方向に送られるパケットを「返答パケット」と呼ぶことにします．

リクエスト・パケットも返答パケットも上記のパケット形式で交換されます．

パケットは送信されても受け手側で本当に受け取られたかどうかはわかりません．つまり，シリアル通信の途中で文字化けなどが起こるとうまく伝わらず，正しくリクエストと返答が行われません．このようなエラー発生の際には，GDB もスタブも再送を要求する必要があります．GDB とスタブの間では，パケットが送信されると，受け手側は送信者に対し，正しく受信できた（つまり，チェック・サムが正しい）場合，「＋」を，再送を要求する場合には「-」を送信し，受信状況を知らせます．

ターゲット・システムで動作するスタブは，GDB からのリクエスト・パケットに含まれるコマンドを理解し，その要求された操作を行います．スタブは，後述する‘ g ’, ‘ G ’, ‘ m ’, ‘ M ’, ‘ c ’, ‘ s ’を最低限含む必要があります（前述したように，これ以外のコマンドはオプションで，GDB に特別な操作をしない限り使われず，いまいち整理されていないように感じる）．

サポートされていないコマンドをスタブが受け取った場合には，

```
$#00
```

を GDB に返します．これはエンプティ・レスポンスと呼ばれ，コマンドのないパケットとなります．

● パケットを形成するコマンド

パケットはリクエストと返答があります．コマンドにもリクエスト用と返答用があります．リクエスト用コマンド群は GDB からスタブへと送られるパケットにしか用いられません．一方，返答用コマンドはスタブから GDB に送られるパケットにしか用いられません．

リクエスト・パケットに用いられるコマンドを**表 1** に示します．コマンドは「1 文字のコマンド文字＋追従データ」という形で送信されます．GDB にはコマンドのモードが二つあります．一つはノーマル・モードでこのモードではターゲット・システムの再起動コマンドを含みません．再起動‘ R ’を含めるようにするには拡張モード設定コマンド‘ ! ’を使う必要があります．

本章で構築するスタブでは拡張モードを使っています．前節でもコンソール・ウィンドウに extended-remote と設定したのはこのためです．

返答パケットに用いられるコマンドを**表 2** に示します．返答パケットは，リクエスト・コマンドのうち，‘ C ’, ‘ c ’, ‘ S ’, ‘ s ’, ‘ ? ’のリクエスト・コマンドに対し，送られます．

● GDB リモート・シリアル・プロトコルでのデータ交換例

それでは，実際の通信例を見てみます．