【実践で使えるプログラミング編】

第6章 複雑なプログラムにも対処できる GNUフリー・ソフトウェア GDBを使ったデバッグ手法

山際 伸一

本章で使用しているプログラムは http://www.cqpub.co.jp/interface/download/ からダウンロードできる.

前章で解説したモニタは，複雑な処理を行うプログラム開発には機能不足である．ここでは，モニタよりもさらに高機能なデバッガを取り上げ，使い方やそのしくみ，スタブの作成などについて解説する．（編集部）

1．デバッガとは何か

非常に複雑な動作を行うプログラムを組み込み機器で実行しているときに発見してしまったバグには，前章で作成したようなモニタの機能だけではどのように対処してよいかわからず，どこが原因となっているのかを発見するのが困難な場合があります．そこで，組み込み開発環境にはデバッガという，モニタよりもさらに高機能なソフトウェアがあります．

デバッガの意図するところは，実行中のプログラムの全容を解明するということです．つまり，モニタのように，どこを実行しているのかわからない状態でのスナップ・ショットを取るのではなく，プログラマの意図した時点でのプログラムの実行状態をすべて解明しようというものです．

デバッガは，モニタ・プログラムと同様に，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して，マイコン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**のような構成をとります．

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．たとえば，マイコンに外部割込みが発生したときや，マイコンが異常な状態に陥ったとき，また，あるレジスタの値が期待したものになったときがそのイベントに当たります．

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．これら三つのイベントは以下のように働きます．

● Breakpoint ── ブレークポイント

実行中のプログラムを意図的にある番地で停止させたい場合に使います．

データ，つまりある番地のメモリへアクセスが出た場合や，レジスタの値がある値になったときと，命令，つまりプログラム・カウンタがある番地を指したという2種類のブレークポイントがあります．データでのブレークポイントはマイコンがレジスタのデータをハードウェアで監視したり，メモリ・データの場合であれば，メモリ・アクセスのデータを監視しなくてはいけないので，マイコンがデータ・ブレークポイント専用のハードウェアを内蔵している必要があります．

● Watchpoint ── 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．これもブレークポイントと同様に，データに関するマイコン内部の専用回路がない限り，正確なイベント発生の把握ができません．

● Catchpoint ── キャッチポイント

C++言語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い，`catch`文でのイベントの発生時にプログラムを停止す

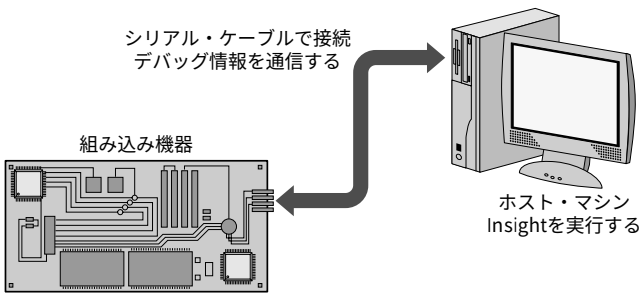

図1 デバッグにおける組み込み機器とホスト・マシンの関係

KEYWORD ──デバッガ，ブレークポイント，ウォッチポイント，キャッチポイント，Insight，GDB，スタブ，モニタ・プログラム

ることができます．本特集ではC言語での場合を前提としているので説明はこの程度に留めます．

これらの機能を実現するデバッガの実装は2種類あります．

▶ ハードウェアで支援するデバッガ

JTAGなどのデバッグ専用の通信線をマイコンが内蔵しているものがこれにあたります．マイコンのベンダがサポートするものの多くはこの方式によるもので，マイコンが内蔵するデバッグ機能を暗にアクティブにし，プログラムの解析を行います．専用のデバッグ機能が起動されるので，非常に細かなイベントまで把握することが可能です．しかし，一般的に，このデバッグ機能の仕様は公開されていない場合が多く，マイコンのベンダが提供する開発ツールを使うしか方法がありません．

▶ モニタ・プログラムが支援するデバッガ

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させることで，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，マイコン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，すべてのマイコンで可能な方式であるため，広く受け入れられています．

本章で扱うデバッガでもこのモニタ・プログラム方式を用います．

2. Insightでデバッグしてみる

● Insightの起動

GNUフリー・ソフトウェアの中には，GCC，bintutilsと組み合わせた，「GDB」と呼ばれるデバッガが提供されています．GDBはコマンドライン・ベースのデバッガでしたが，RedHat社により，GUIをもつ非常に使いやすい殻が構築され，Insightとして世に送り出されました．

GDBはもともと，GDBを実行しているマシン上で走るプログラムをデバッグするために開発されましたが，クロス開発環境にも対応できるように変更が加えられ，最近では組み込み機器開発にも用いられることが多くなりました．InsightはGUI部分のみをかぶせてあるので，GDBの機能はそのまま保持されています．

さて，Insightは第4章のGNUフリー・ソフトウェアで開発環境を構築した際に，実はすでにインストールされています．

```
/usr/local/sh-tools/
```

にInsightが構築されています．まずは，起動してみましょう．以下のコマンドを実行してください．

```
$ /usr/local/sh-tools/bin/sh-elf-insight.exe
```

すると，**図2**に示すようなウィンドウが表示されます．

● デバッガを体験してみよう

▶Insightと付録基板のセットアップ

本誌のWebページから，ここで用いるパッケージである`gdb-stub.tar.gz`をダウンロードして，Cygwinのホーム・ディレクトリに展開してください．

```
$ tar zxvf gdb-stub.tar.gz
```

`gdb-stub`フォルダの下には`monitor`フォルダと`download_sample`フォルダがあります．

`monitor`フォルダには，Insightと通信してデバッグを進めるための通信プロトコルをもつソフトウェア`monitor.mot`があります．そのファイルをFDTを使って付録基板に書き込んでください．

`download_sample`フォルダにはデバッグ対象となるC言語プログラムのサンプルが入っています．このサンプルは**リスト1**に示すように，付録基板に付いているLEDを20回点滅させて，SCI0からメッセージを出力した後に，入力を待つようになっています．SCI0の設定は，9600bps，8ビット・データ，パリティなし，ストップ・ビットが1ビットになっています．

ダウンロードに用いているシリアル・ポートをInsightで使うので，`monitor.mot`のダウンロード後はFDTを閉じておきます．パワーONリセットを付録基板にかけて，Cygwinコンソールを開き，`download_sample`ディレクトリに移動します．

```
$ cd ~/gdb-stub/download_sample
```

以下のコマンドでInsightを起動してください．

```
$ /usr/local/sh-tools/bin/
        sh-elf-insight.exe download_sample
```

そうすると，**図3**のようなウィンドウが起動します．残念ながら日本語がサポートされていないため，文字化けしてしまいま

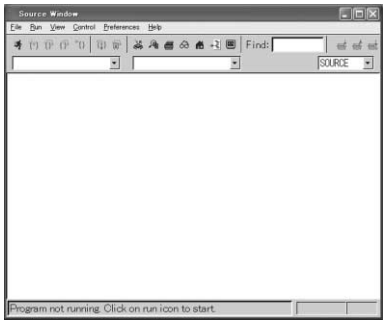

図2
Insightの起動画面

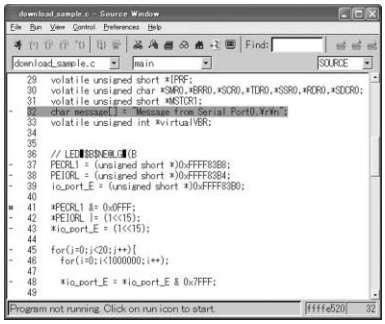

図3
起動したウィンドウ