1M-3

ハードウェア動作記述からのソフトウェア論理シミュレータの生成

井上勝博　三原幸博
(株式会社東芝　システム・ソフトウェア技術研究所)

1．はじめに

マイクロコンピュータ（以下マイコン）組み込み製品の増加により、組み込み型マイコンのデバッグ／テスト支援環境の強化が望まれている。

従来、組み込み型マイコンのソフトウェアのテストは、実機、エミュレータ等を用いて行ってきた。このような環境では、開発の対象となるハードウェア（以下ターゲット）が必要であり、そのようなハードウェア環境をターゲットが変わる毎に作成する必要があることや、環境が大規模になりやすい等の問題がある。そこで、我々は、ＩＭＡＰシステムの一テスト手法としてソフトウェアシミュレータによる論理デバッグ／テスト[1]を提案し、ハードウェアから切り離した状態でのソフトウェアのデバッグ／テストの強化を図った。ソフトウェアシミュレータは、アセンブラまたは、コンパイラから出力されたオブジェクトコードを読み込み、シミュレート実行する。マイコンの持つメモリ・レジスタ等のリソースをコンピュータ内部に仮想的に実現し、これらの変化状況を動的に把握できる機能を持つ。シミュレータには、ハードウェアの影響を受けない、ターゲットに対して柔軟に対応できる、エディタ、コンパイラ等と統合することによりコンパクトな開発環境が実現できる等の利点がある。

ここでは、このような特徴を持ったシミュレータを各種のターゲットに対して柔軟に生成する方式について述べる。

2．内部構成

シミュレータの汎用化のため、システム全体の構成を明確にする。シミュレータには、ターゲットとなるハードウェアが持つリソース（レジスタ、メモリ等）を仮想的に実現、オブジェクトコードをロード、仮想的に実現したレジスタ、メモリを参照することによる命令のシミュレーション、リソースの参照、設定、ブレークポイントの設定、シミュレーション過程のトレース、等の機能がある。これらの機能を持つシミュレータを異なるターゲットに対応できるようにする場合、通常、ターゲットの動作をシミュレーションできるように作りかえるが、設計によっては、上記機能のほとんどの部分を作り替えなければならないことになる。ここでは、オブジェクト指向の考え方を用いて、ターゲットに依存する部分と依存しない部分を明確にし、モジュール分割することにより、プログラムの再利用を図る。シミュレータの内部構成を図１に示す。

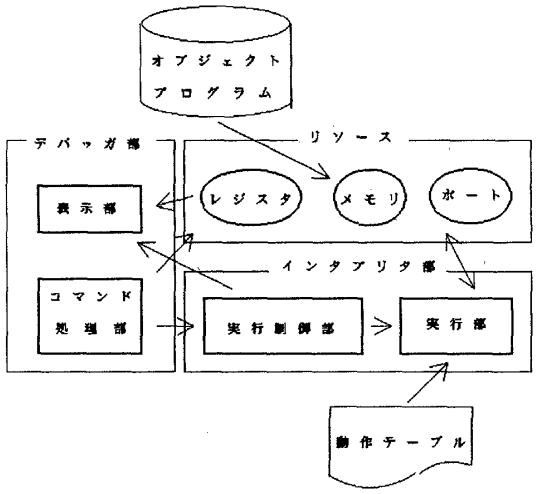

図１．シミュレータ内部構成

シミュレータの構成をプログラムのシミュレーションを行うインタプリタ部とデバッグ用の機能を与えるデバッガ部、および、レジスタ、メモリ等のリソースを仮想的に実現する部分に分ける。デバッガ部は、ユーザからのコマンドを受け付けるコマンド処理部と、その結果の表示を受け持つ表示部よりなる。この部分をターゲットから切り離すことにより、ターゲットによらない共通したユーザ・インタフェースを与える。しかし、表示フォーマットに関しては、ターゲットに依存するためフォーマットの指定機能を持つ。

リソースの実現部とインタプリタ部は、ターゲットに依存する部分である。リソースは、レジスタ、メモリ、ポート等のクラスに分け、個々の実体は、オブジェクトとして実現する。個々のリソースの持つ特性はオブジェクトの中に入れておく。インタプリタ部は、これらのオブジェクト間のデータの流れを操作するもので、仮想的に実現されたメモリ上にロードされたプログラムと、レ

Generation of Software-Simulator from hardware behavior description
Katsuhiro INOUE, Yukihiro MIHARA
TOSHIBA Corporation

ジスタを参照しながらプログラムのシミュレーションを行う。これらの操作手順は、動作テーブルを参照する。動作テーブルは、後述する動作定義より生成される。このようにすることにより、リソースの一般的特性や、動作テーブルの駆動ルーチン等は、汎用に使用できる。ターゲットに依存し、汎用的に利用できない部分については、ハードウェア動作記述より生成する。

3．生成方式

シミュレータの生成方式を図２に示す。本方式では、移植性と、動作効率を考慮して、動作定義ファイルをトランスレータがＣ言語のソースファイルに変換し、それをコンパイルする方式を取っている。

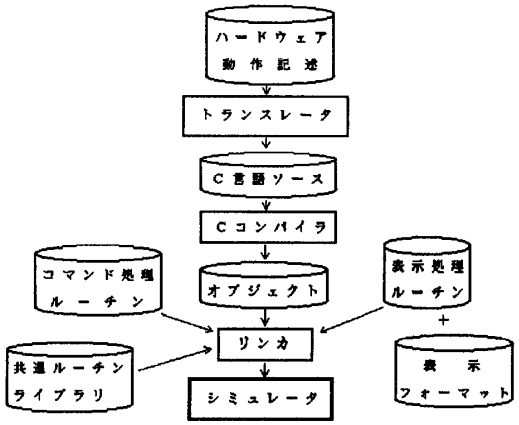

図２．シミュレータ生成方式

4．ハードウェア動作記述

シミュレータの核となるインタプリタ部および、リソースは、ハードウェア動作記述より生成される。動作記述中には、リソースのオブジェクトとしての実現形式と、そのオブジェクト間の操作を動作記述として記述する。動作記述の概要を以下に示す。この言語は、既存プログラミング言語の記述形式の利用と柔軟性に重点をおいて設計した。

(1) リソース定義

メモリ、レジスタ等のリソースのビット幅、アドレス幅等を定義する。

例：
```
memory RAM(0,255,4);
 /* 1ﾜｰﾄﾞ4ﾋﾞｯﾄで、256ﾜｰﾄﾞのﾒﾓﾘである */
assign(0,255,RW);
 /* 上記ﾒﾓﾘをﾘｰﾄﾞ/ﾗｲﾄとして割り付ける */
register PC(12);
 /* PC は12ﾋﾞｯﾄのレジスタである */
```

大規模なアドレス空間を持つハードウェアへの対応を考慮して、メモリの割付はユーザ定義可能な形式を取っている。

(2) 命令動作定義

命令動作定義として、各命令コード、動作、ニーモニックの関係を記述する。

例：
```
{
00111111:xxxxxxxx,{ RAM[x]=AC; SF=1; cycle(2);
                  },      "ST     A,x";
........ }
```

命令コードに対応する動作は、リソース定義で定義した名前を用いて、Ｃ言語の記述形式で記述する。

なお、ここでの実行タイミングは命令サイクルを基本にしており、各命令の命令サイクルも指定する。

(3) 割り込み処理定義

割り込み発生条件とそれに対応する処理を次のように記述する。

例：
```
{ INTL5&EIF,      { interrupt(002,INTL5); }
  INTL4&EIF&EIR3,{ interrupt(003,INTL4); }
  ........ }
```

このように、発生条件とアクションを対応付けて定義する。ここに定義してあるように、アクション記述の内部は関数記述が可能であり、それらの関数は、ユーザ関数定義部で自由に記述することができる。

(4) イベント定義

割り込み処理定義と同様の記述により、イベントに対するアクションを記述する。イベントとしては、リソースからの読みだし、書き込み等がある。

(5) 実行制御定義

命令フェッチ、実行、割り込み処理等の順序を手続き記述の形式で記述する。

その他、付加機能については、関数記述形式で自由に付加することが可能である。

6．まとめ

ハードウェア動作記述からソフトウェアシミュレータを自動的に生成する方式について述べた。本方式では、シミュレータの内部構成を明確にし、モジュール化、再利用を図ると共に、動作記述言語を設計することにより各種ハードウェアに対し柔軟に対応する方式を提案した。これにより、シミュレータ開発期間を短縮でき、ターゲットの変更に迅速な対応が可能である。

また、この動作記述より生成されるモジュールは、プログラムのシミュレートエンジンとなるもので、これにプログラムの自動実行、検証等の機能を付加することにより有効な論理テストツールになると考えられる。

参考文献：

[1] 井上、貫井、三原、平尾：マイコンソフト開発における論理テスト環境、情処第35回全国大会3Y-8(1987)