シミュレータコンパイラとシェルブレッドボード方式による LSI 設計環境の提案

平井千秋† 林 晋一†

従来の回路シミュレータ，制御系シミュレータは，解析対象系の構成情報（ネットリスト，伝達関数）と系への入力波形定義情報が混在するデータを読み込んで解析を行っていた．これに対してわれわれは，系の構成情報をコンパイルして実行形プログラムを生成するシミュレータコンパイラを提案する．実行形は，系への入力波形と設計パラメータ値を読み込んで系の出力を算出する．複数の部分系をそれぞれ実行形に変換し，プロセス間通信を行いながら全体系のシミュレーションを行うことができる．これにより，(1)設計データの管理や保守がしやすくなる．(2)UNIX 環境において，系の一部分を差し替えたり，パラメータの値を変化させながらシミュレーションを繰り返す等のバッチ制御をシェルプログラムで記述できる，(3)任意の言語で書かれたプログラムとシミュレータの結合がシェル上でできるなど，従来にない設計支援環境を実現できる．これは，UNIX シェルを計算機支援設計環境におけるブレッドボードと考えるものであり，シェルブレッドボード方式と名付ける．本論文では，シミュレータコンパイラの実現方法として，従来のシミュレータを部分計算する方法を述べ，従来シミュレータの小規模な改良により新しい設計環境を実現できることを示す．

A Proposal of LSI Design Environment Based on Simulator Compiler and Shell Breadboard

Chiaki Hirai† and Shin-ichi Hayashi†

Conventional circuit simulators analyze data that define both a circuit and input wave form to the circuit. We divide the simulation process into two steps. First, a simulator compiler compiles the data defining the circuit into an object program. The object program, then, reads input wave form and produces output wave form. A system can be divided into small object programs that communicate with each other and simulate the whole system. This simulation method gives following advantages. (1) It provides the clarity of simulation data. (2) In UNIX environment, simulations can be controled by shell programs, (3) a user program can be connected with the simulator on UNIX shell. We call this design environment a shell breadboard, for we use the UNIX shell just like a breadboard in real design environment. We also show that the simulator compiler can be obtained from a conventional simulator by using partial evaluation technique. With the simulator compiler and the shell breadboard, circuit simulation environment can be improved.

1. はじめに

回路シミュレータ[1]は，大型計算機上で開発され，使用されてきたが，近年のワークステーションの処理能力向上に伴い，設計支援環境のワークステーションへの移行が行われている．しかし，その移行のほとんどは，従来シミュレータの単純な移植であり，シミュレータの開発思想は，1970年代の SPICE 以来変わっていない．

このような中で最近ワークステーションに適応する回路シミュレータの研究が始まっている．大谷ら[2]は，UNIX☆での回路シミュレータのあり方を論じ，シミュレータの機能を可能な限り独立なコマンドで実現し，それらの仕様を統一化することを提案している．これは，UNIX におけるプログラム開発思想を踏襲したものであり，大型計算機環境にはなかった考え方である．

われわれは，この考えをさらに進め，回路の過渡解析において，回路の部分ブロックを一つのコマンドに変換するコンパイラシミュレータを提案し，実際に開発した．これは，コマンドの入出力を切り替えたり，パイプ処理によってコマンドを接続する UNIX の考え[3]を回路シミュレータに適用したもので，回路ブロックの接続を UNIX シェル上で可能とすることによ

† 日立製作所システム開発研究所第1部
Systems Development Laboratory, Hitachi, Ltd.

☆ UNIX オペレーティングシステムは，UNIX システムラボラトリーズ社が開発し，ライセンスしています．

り，設計環境の改善を図るものである．

また，設計パラメータをコンパイル後でも変更できるよう工夫することにより，シミュレーションの繰返し時におけるコンパイル時間の無駄をなくしている．

このパラメータ変更に関する必要性から，データ記述にオブジェクト指向データベースで用いられる識別子，継承の概念[4]を取り入れている．LSI の記述についてのオブジェクト指向モデル導入の研究[5),6)]に見られるように，モジュール間の継承を取り入れると，データの記述量を節約できる．本論文では，それに加えて，識別子の導入が設計パラメータの修正に有効であることを示す．

われわれが対象とする系は，機能ブロックと回路ブロックの混在系である．われわれは先に機能と回路の混在系のシミュレーション手法を提案した[7]．解析対象は，ブロック線図で記述され，各ブロックは，代入式による関数表現（機能ブロック）かネットリスト（回路ブロック）で記述する．

本論文におけるシミュレータの設計思想は，ブロック間の接続と設計パラメータの変更をシェル上で行えるようにすることである．これは，シェルを設計におけるブレッドボードと考えるものであり，シェルブレッドボード方式と名付けた．

以下，第2章でシミュレータの使用環境についてのわれわれの仮説を提示し，第3，4章ではシミュレータコンパイラとその実現法，第5章では実行形のパラメータ変更方法を述べる．第6章には今後の課題としてプロセス間通信によるシミュレーションを述べ，第7章でシミュレータコンパイラによって実現される新しいシミュレータ使用環境（シェルブレッドボード）について述べる．

2. シミュレータの使用環境

回路シミュレータは，1970年代に大型計算機環境を想定して開発された．この回路シミュレータは，設計が完了した回路を最終的に検査するために使われてきたといえる．この使われ方においては，使用環境を考慮する必要はあまりない．シミュレータを数値解析ソフトウェアと捉えて，高速，高精度計算を実現すればユーザの要望を満たすことができる．

われわれの研究は，大型計算機上で開発されたソフトウェアが，ワークステーション上での最適形態だろうかという疑問に端を発している．これは，シミュレータを単なる数値解析ソフトウェアと捉えて開発するのではなく，その利用環境も含めて開発しなければ，よりよい回路設計環境を得られないという認識に基づいている．

重要なのは，最近の個人用計算機の普及にともない，回路シミュレータが開発ツールとして用いられてきている[8]ことである．この場合，回路開発にソフトウェア開発の手法が用いられることになる．すなわち，データ（プログラム）を記述し，シミュレーション（インタプリタ）を実行し，データを修正（デバッグ）することを繰り返す．したがって，シミュレータを実行する計算機には，プログラミング開発環境の良いことが望まれる．

UNIX が今日浸透している理由は，大型計算機に比べてのプログラム開発環境の良さにある．われわれは，UNIX の次の二つの特徴[3]に着目した．

（1） UNIX は，多くのプログラム（ツール）を有し，それぞれが独立に働き，他のプログラムと結合して新しい機能を作り出せる．

（2） UNIX のコマンドがオプションを持ち，既定値をかえて実行できる．

そこでこの UNIX の考えを回路シミュレータに当てはめ，

（1′） 解析対象系を分割し，それぞれを独立のコマンドとして，シェル上で結合できる．

（2′） コマンドのオプションとして設計パラメータ値を実行時に変更できる．

とすると，従来の大型計算機環境では得られなかった機能を実現できるのではないかという仮説を立て，シミュレータを開発した．

分割した対象系をシェル上で結合すると，ブロック間の信号の流れは一方向となる．このような理想化シミュレーションは，設計の初期においてはむしろ必要である．

回路設計の初期段階では，回路は，複数の機能ブロックからなるブロック線図で表現される．機能ブロックは，回路設計が未完のものは，関数表現され，終了したものは素子のネットリストで表現される．この段階では，ブロックからブロックへの信号の流れは一方向と理想化して全体系の挙動を検証する．

複数の機能ブロックを統合してネットリストに詳細化していくことにより，部分回路間のインピーダンスマッチングまでを考慮した詳細シミュレーションに徐々に移行できる．すなわち，初期設計から詳細設計までの一貫支援が可能となる．

部分ブロック数の多い初期段階のシミュレーションでは，部分ブロックを関数表現することによりシミュレーションを高速化する[7]ため，シェルでの結合に伴うデータ転送のオーバヘッドを相殺できる．実際，UNIX ワークステーションで90秒を要するシミュレーションを二つの実行形に分割してパイプ接続した場合，通信オーバヘッドによる時間増は，1％以下であった．

3. シミュレータコンパイラ

3.1 シミュレータコンパイラ

従来のシミュレータ（S）は，系の定義情報（D）と入出力波形定義情報（W_i）を入力として，系の出力（W_o）を解析していた．

$$W_o = S(D, W_i).$$

これに対し，シミュレータコンパイラ（M）は，解析する系の定義情報をコンパイルして実行形（E）を生成する．この実行形は，任意の入力波形を読み込み，この入力波形に対する系の応答を出力する（**図1**）．

$$E = M(D),$$
$$W_o = E(W_i).$$

シミュレータコンパイラを導入すると，以下に示すように UNIX シェル環境の機能を利用することができる．

3.2 シェル上での部分系の接続

系が二つの部分系（系1，系2）の直列接続からなる場合を考える．系1，系2をそれぞれ実行形（実行形1，実行形2）にコンパイルする．実行形1の出力を実行形2の入力として解析を行えば，全体系を解析できる．UNIX シェル上では，パイプ機能により，あるコマンドの標準出力を別のコマンドの標準入力とすることができる．そこで，実行形の入出力を標準入出力とし，パイプ機能を利用すると，シェル上で部分系を結合することができる．

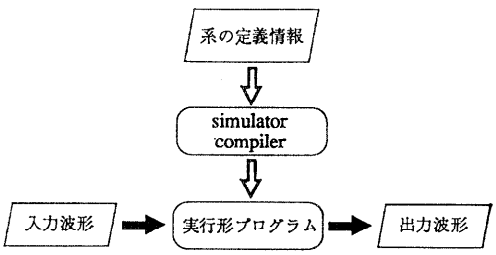

図1 シミュレータコンパイラ
Fig. 1 Simulator compiler.

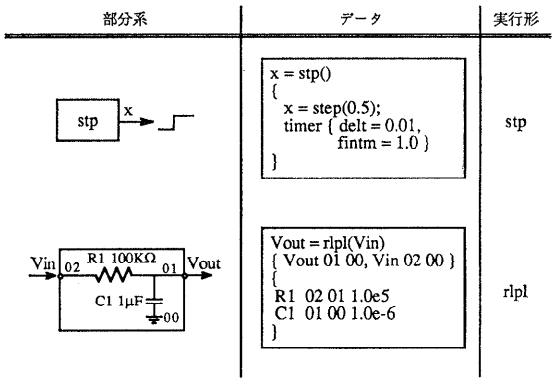

図2 部分系
Fig. 2 Subsystems.

具体例を示す．**図2**に示す二つの部分系をコンパイルしてそれぞれ，実行形 stp，実行形 rlpl を得る．stp は，ステップ関数を出力するブロックのシミュレータであり，$t=0.5$ で立上り，$t=0.0$ から，$t=1.0$ (fintm) まで時間刻み 0.01 (delt) で波形を出力する．ここで，step は，記述言語での組み込み関数であるとする．rlpl は，入力波形に対する一次遅れを出力する回路ブロックである．

ブロックをコンパイルして得られる実行形の入出力データ仕様を次のように定義する．データ記述の第1行（ブロックヘッダ）が，

$$y_1, \cdots, y_n = \mathrm{blk}(x_1, \cdots, x_m),$$
$$(n>0, m \geqq 0)$$

のとき，このブロックの実行形への入力を，

$m=0$ のとき，

入力なし，

$m>0$ のとき，

t $x_1(t)$ $\cdots$ $x_m(t)$
$t+\Delta t$ $x_1(t+\Delta t)$ $\cdots$ $x_m(t+\Delta t)$
$\cdots$

とする．

出力を，

t $y1(t)$ $\cdots$ $yn(t)$
$t+\Delta t$ $y1(t+\Delta t)$ $\cdots$ $yn(t+\Delta t)$
$\cdots$

とする．

実行形の入出力を標準入出力にすれば，rlpl にステップ関数が入力される場合の解析は，UNIX のパイプ処理を用いて，

stp|rlpl

として実行できる．

具体的には，stp が，波形，

```
0.0  0.0
0.01 0.0
…
0.5  1.0
…
1.0  1.0
```

を出力し，rlpl では，新しい時刻における入力値を得る度に，その時刻でのブロックの出力を解析し出力する．

われわれのシミュレータでは，時間刻みが定義されないブロック（例では，rlpl）の数値積分の時間刻みとして，入力時系列データの時間刻みを採用している．このため，rlpl は，入力データから時間刻み 0.01 を逆算し，これを用いて数値積分を行い，出力刻み 0.01 で出力を行っている．

数値積分に可変時間刻みを採用し，入力の時間刻みよりも小さな時間刻みで解析し出力する場合は，入力時系列データを補完して対応する入力値を算出する必要がある．

他の波形生成器（sin（サイン波），pls（パルス））や，信号処理器（例えば，amp）を用意して

```
sin|rlpl
pls|rlpl
stp|rlpl|amp
```

とすれば，他の波形に対する rlpl の応答を調べることができる．このように，部分系を個別にコンパイルしてこれらをシェル上で組み合せることが可能となる．

3.3 シミュレータコンパイラの利点

（1） データの管理と保守

先の例に見るようにシミュレーションの終端時間定義，時間刻み定義は，波形を生成するブロックの属性として定義される．すなわちこれらは，波形生成器が生成する波形の持続時間と出力刻みと解釈される．これにより，解析対象を記述するファイル（例では，rlpl）から，解析対象とは本来無関係な情報（入力波形定義，解析条件定義）を分離できる．この分離は，次のようなデータの管理や保守における利点を生む．

（a） 入力波形，解析条件を変える際に解析対象記述ファイルを変更する必要がない．

（b） 測定データや他の系の解析結果等のファイル化されたデータを入力とする場合も，データの書式をあわせておけば，UNIX の入力切り替え機能（<）により，

```
rlpl<file
```

として実行できる．

（c） 波形生成器，信号処理器をライブラリ化してシェル上で再利用できる．

（2） シェルプログラムによるバッチ処理の記述

従来のシミュレータは，数値解析アルゴリズムとシミュレーションの制御アルゴリズム（シミュレーションの繰返し，パラメータの最適化など）の双方から成り立っている．このため，シミュレータソフトは大規模化し，保守が困難になる一因となっている．

シミュレータコンパイラの考え方では，シミュレーションをコンパイル過程とシミュレーション実行過程に分割する．この結果，数値解析アルゴリズムはコンパイラが有し，シミュレーションの制御機構は，実行形を制御する別プログラムとして開発することができる．

例えば，シェルプログラムを用いると，系の一部を変更しながらシミュレーションを繰り返す処理のバッチ化が可能となる．最も単純な例は，**図 3** に示すようにあらかじめ用意した入力波形に対する応答解析のバッチ化である．この例は，波形生成器 stp，pls，sin に対する rlpl の応答を解析し，それぞれの結果を，stp.out，pls.out，sin. out のファイルに格納することを指示するプログラムを示す．

C言語を用い，実行形を子プロセスとしてより高度な制御を行うプログラムを開発することも可能である．

このように，数値解析アルゴリズムと制御機構を別プログラムとすることにより，プログラムの保守性が向上すると共に，制御機構の独自開発が可能になる．

（3） 他のプログラムとの接続

シェル上での接続は，入出力仕様の一致が要請されるのみなので，ユーザが汎用プログラム言語で開発したプログラムの実行形をシェル上で接続することができ，入力波形の定義，出力波形の処理等に利用することができる．

```
#!/bin/sh
for  i  in stp  pls  sin
do
        $i | rlpl  >  $i.out
done
```

図 3 シェルプログラム（1）
Fig. 3 Shell program (1).

4. シミュレータコンパイラの実現法

4.1 部分計算による実現

以下にシミュレータコンパイラの一実現方法を述べる．すでに回路シミュレータがある場合，これを部分計算[9]することにより，シミュレータコンパイラと等価な機能を実現することができる．部分計算とは，プログラムの入力の一部が既知である場合に，既知入力に対して可能な計算部分を実行しておくプログラム変換法である．

従来シミュレータ（S）を，系の構成情報（D）と入力波形（W_i）から，出力波形（W_o）を計算するプログラムであると捉えると，

$$W_o = S(D, W_i) \quad (1)$$

と書ける．いまシミュレータコンパイラを M とすると，

$$W_o = M(D)(W_i) \quad (2)$$

である．

部分計算プログラム α をシミュレータ S に作用させ，既知の構成情報 D について部分計算すれば，

$$W_o = S_D(W_i) \quad (3)$$

ただし，

$$S_D = \alpha(S, D) \quad (4)$$

を得る．$\alpha(S, D)$ は，シミュレータ S に系の構成情報 D を入力して，計算可能な部分の計算を進めたプログラムである．

式（2）と式（3）を比べると，

$$M(D) = S_D$$

であることから，従来のシミュレータを系の構成情報について部分計算すれば，シミュレータコンパイラで系の構成情報をコンパイルしたプログラムと同等のプログラムを得ることができる．

4.2 プログラミングの実際

以上の議論にしたがって，シミュレータコンパイラの具体的な構成法を述べる．まず，従来回路シミュレータの入力部を系の構成情報読み込みルーチンと入力波形読み込みルーチンに分離する（**図4**）．

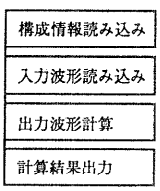

図4 従来回路シミュレータ
Fig. 4 A conventional circuit simulator.

ここで，構成情報読み込み部分は，原理的には次のサブルーチンで実行できる．

```
read_system(x)
char *x;
{
  set_data(x);
}
```

サブルーチン set_data は，入力文字列をファイルから読み込んでポインタ x の指す記憶領域に格納する機能を持つサブルーチンである．例えば，系の構成情報を表す入力文字列が，“abcdef” であった場合，x の指す領域に “abcdef” が格納される．

ところで，文字列（例えば，“abcdef”）を読み込んで，次に示すサブルーチンを生成するプログラムを書くことが可能である．

```
read_system(x)
char *x;
{
  strcpy(x, "abcdef");
}
```

このサブルーチン（系定義サブルーチン）は，設定された文字列（系の構成情報）をポインタ x の指す領域に入力する機能を持つ．

図4の構成情報読み込みルーチンを生成した系定義サブルーチンで置き換える．この新しいプログラムは，図4のシミュレータにネットリストを入力して部分計算したものである．実際，このプログラムを実行させると，任意の入力波形を読み込み，系定義サブルーチンで定義される系の出力波形を計算し，解を出力する．これが求める実行形である．

このように，系定義サブルーチンを生成するプログラムを用意しておき，与えられた系の構成情報より系定義サブルーチンを生成し，これを従来シミュレータの後段（入力波形読み込みルーチン以降）とリンクすれば，求める実行形を得る．

5. 実行形でのパラメータ変更

5.1 実行形オプションによるパラメータ変更

シミュレータコンパイラを導入すると，設計パラメータの変更の度に再コンパイルが必要となり，シミュレーション効率を落とす懸念がある．そこで，次に述べるパラメータ文と実行形オプションを導入して，コンパイル後にパラメータ値が変更できるようにする．

基本的な考えは，ブロックの記述データ中に，パラメータ文，

parameter {p1=0.0, p2=0.1}

を記述すると，実行時にオプションでパラメータ値が変更できるようにするものである．

すなわち，実行形を exm として，

exm −p1 1.0

exm −p1 1.0 −p2 2.1

とすると，p1, p2 の値を変更して解析が行われる．

5.2 継承と識別子の導入

ここで，パラメータが異なるだけの部分系の接続によって定義される系を考える（例を**図5**に示す）．このような系の記述を容易にし，かつ実行形のオプションによるパラメータ値の変更を可能にするために，部分系の定義に関し，継承と識別子を導入する．

部分系の定義

$y_1, \cdots, y_n = \mathrm{blk}(x_1, \cdots, x_m)$

{

parameter $\{p_1 = c_1, \cdots, p_k = c_k\}$

…

}

がすでになされているとき，

$y_1, \cdots, y_n = \mathrm{id} : \mathrm{blk}(x_1, \cdots, x_m)[p_j = c'_j, \cdots]$

なる記述によって新しい部分系の定義を可能にする．

この記述の意味を，新しいブロックの名前を id（ブロック識別子）とし，系の構成要素とパラメータ値を元の部分系（blk）から継承し，[ ] の中に記述されたパラメータのみに関しては，パラメータ値を再定義し直すものと定める．すなわち，ブロック blk とブロック id で異なるのは，ブロックの名前と再定義されたパラメータだけである．これは，オブジェクト指向データベース[4]における識別子と継承の考えを取り入れたものである．

この定義により，図5の例は，**図6**のデータで記述できる．まず，rlpl により部分系を定義し，rlpl2 でこれを接続する．ブロック識別子 id1, id2 により，二つのブロックの実体が異なることを表し，id2 で示

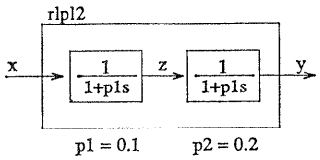

図 5 パラメータの異なる部分系
Fig. 5 Subsystems with different parameters.

```
y = rlpl (x)
{
  parameter { p = 0.1 }
  y = realpl(0.0,p,x);
}
y = rlpl2(x)
{
  z = id1 : rlpl (x);
  y = id2 : rlpl (z) [ p = 0.2 ];
}
```

図 6 データ記述
Fig. 6 Simulation data.

される rlpl のパラメータ p の値は，0.2 と再定義する．ただし，rlpl 中の realpl は，一次遅れ要素を表す組み込み関数としている．

二つのブロックの実体が異なることを明示できることにより，次のようにして，部分系内のパラメータを実行時に変更できる．

図6の系をコンパイルした実行形名を rlpl2 とする．実行形のオプションとして指定するパラメータ名を

ブロック名．パラメータ名

と規約すれば，先の例題では，

rlpl2 −id1.p 0.2 −id2.p 0.3

として，部分系中のパラメータを一意に指定できる．逆に識別子がないと，実行形オプションにおいて二つの部分系を区別することができず，それぞれのパラメータに異なる値を設定することができない．

部分系が入れ子になっている場合には，ピリオドを重ねて入れ子を表すことにする．例えば，ブロック id1 の中に，パラメータ p′ の定義されるブロック id11 がある場合，

id1. id11. p′

としてパラメータを指定できる．以上の表記法により，部分系のパラメータを含め，実行時でのパラメータ変更が可能となる．

実行形のオプションとしてパラメータが変えられることにより，シェルプログラムによるシミュレーション制御が可能となる．シェルプログラムによる制御例を**図7**に示す．

```
#!/bin/sh
for i in 0.1 0.2 0.3
do
        stp | rlpl -p $i > $i.out
done
```

図 7 シェルプログラム（2）
Fig. 7 Shell program (2).

6. プロセス間通信によるシミュレーション

現在，上記に述べた機能を実現し，シミュレーションを行っている．しかし，UNIX の標準シェルのパイプ機能を使用した場合，直列に接続された部分系はシェル上で結合できるが，より複雑に結合された系（例を**図8**に示す）を扱うことはできない．

このような系を実現するためには，シェルの拡張が必要になる．本章では，プロセス間通信による全体シミュレーションの基本的方法を述べる．

シェルを拡張することにより，パイプをループにしたり，二つのプロセスの出力を1行にマージすることが可能である[10]．すなわち，部分系をすべて独立のプロセスとして同時に実行させ，たがいにプロセス間通信を行いながら全体系の解析を行うことが可能である．

ループにおけるプロセスの計算順序が問題となるが，従来，連続系シミュレータでは，系を前進形積分[11]で離散化してブロックの計算を順序化し，時間刻みごとにこの順序にしたがって計算を実行することが行われている[12]．このとき，ループ中に $t=0$ での初期値を持つ要素（積分作用素や時間遅れ要素）が現れることを解析対象の条件としておき，代数的ループが生じるのを防いでいる．

そこで，ループを持つ系をプロセス間通信で解析する場合にも，積分法として前進形積分を採用し，プロセスの実行を順序化すれば，時間刻みごとにこの順序にしたがってプロセスを実行させることができる．

実行順序の制御は，プロセスが必要な入力データを得るまでは処理を開始しないという機能によって実現できる．例えば，図8の例において，Eが 1/S（積分作用素）であるとする．$t=0$ において，A，B，C，Dは，それぞれのプロセスへの入力が揃うまでは計算を開始しない．Eには，$t=0$ での出力値（初期値）が与えられているため，入力値がなくても出力を行うことができる．Eの出力によりAの入力データが得られたことになり，Aの計算が行われる．順次，B，C，Dの計算が実行される．Eは，Cの出力を読み込んで前進形積分を実行し，$t=\Delta t$ における出力を計算する．$t=\Delta t$ でのAの値が計算でき，B，D，Cの計算が実行できる．以下これを繰り返し，終端時間までのシミュレーションを実行できる．

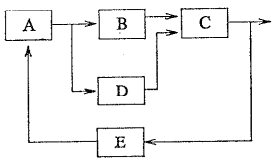

図 8 複雑な系
Fig. 8 A complicated system.

7. シェルブレッドボード

以上のように，部分系のコンパイル，実行形のパラメータオプション，識別子，継承，プロセス間通信を取り入れると，部分系を用意してこれを組み上げながら全体系を構成するという作業が，データファイル上とシェル上の双方で可能になることになる．

例として**図9**に比較をあげる．部分系 rlpl によって構成される系 rlpl 2 は，ファイル上で接続を定義してコンパイルすることが可能である一方，部分系をコンパイルしてシェル上で接続することも可能である．

シェル上での接続は，再コンパイルの必要がなく，シミュレーションの試行錯誤的再実行が容易である．とくに，シェルプログラムあるいはCプログラムによって系の構成要素やパラメータを変化させながらシミュレーションを繰り返すバッチ処理を記述できる．さらに，シミュレーション結果に依存させて系の構成要素を差し替えたり，パラメータを変化させるという従来のシミュレーション環境では困難であった機能を実現できる．

シェル上の結合がこのように柔軟である半面，データは一過性であり，かつ，実行速度においては，全体系をコンパイルしたものに劣る．そこで，シェル上で試行錯誤し，設計完了データをファイルに保存する利用形態が考えられる．これは，シェルを実回路設計におけるブレッドボードに模擬するものであり，シェルブレッドボードと名付けている．

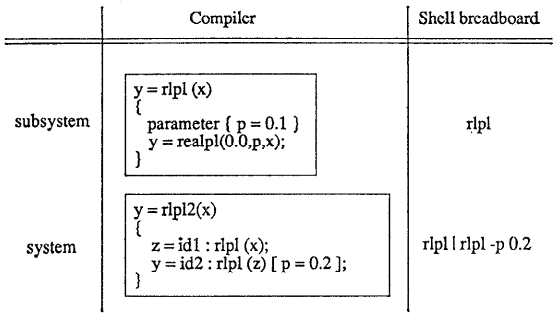

図 9 コンパイラとシェルブレッドボードの記述形式
Fig. 9 Data format for the compiler and the shell breadboard.

8. むすび

UNIX 環境に適する回路シミュレータを考察し，シミュレータコンパイラとシェルブレッドボード方式を提案し，シミュレータコンパイラを開発した．シミュレータコンパイラは，系の構成情報をコンパイルして，系の挙動をシミュレートする実行形プログラムを生成する．実行形プログラムは，系への入力波形と設計パラメータ値を読み込んで系の出力を算出する．実行形は，シェル上で接続，制御できる．

これにより

(1) 設計データの管理や保守がしやすくなる．

(2) UNIX 環境において，系の一部分を差し替えたり，パラメータの値を変化させながらシミュレーションを繰り返す等のバッチ制御をシェルプログラムで記述できる．

(3) 任意の言語で書かれたプログラムとシミュレータの結合ができる．

という従来のシミュレーション環境にはない機能を実現できる．

謝辞 機能/回路混在系シミュレータに関し，ご助言をいただいた日立製作所システム開発研究所の渡辺俊典博士（現電気通信大）に感謝いたします．また，回路設計およびシミュレーション法に関しご助言ご討論いただいた日立製作所中央研究所永田穣博士，増田弘生博士，日立マイコンシステムの池松龍一氏，研究機会を与えていただいた当所明石吉三博士に感謝いたします．

参考文献

1) Calahan, D. A.: Computer-Aided Network Design, McGraw-Hill, Inc. (1972). (鈴木登紀男(監訳)：コンピュータによる電子回路設計，日刊工業新聞社 (1974).)
2) 大谷ほか：新世代回路シミュレーション技法の開発のための一提案，電子情報通信学会論文誌 A, Vol. J74-A, No. 8, pp. 1197-1207 (1991).
3) Kernighan, B. W. et al.: The UNIX Programming Environment, Prentice-Hall, Inc. (1984). (UNIX プログラミング環境，アスキー出版 (1985).)
4) Atkinson, M. et al.: The Object-Oriented Database System Manifesto, *Proc. of the First International Conf. on Deductive and Object-Oriented Databases* (*DOOD 89*), pp. 40-57 (1989).
5) Wolf, W.: An Object-Oriented Procedural Database for VLSI Chip Planning, *23rd DA Conference*, 42.1, pp. 744-751 (1986).
6) 森山ほか：アナログ回路設計自動化のための統合環境，電気学会研究会資料電子回路研究会，ETC-92-8 pp. 1-10 (1992).
7) 平井ほか：後退形/前進形両積分法を併用した機能/回路混在系の新シミュレーション手法，情報処理学会論文誌，Vol. 32, No. 8, pp. 1014-1021 (1991).
8) 岡村廸夫：SPICE によるシミュレータ新活用法，CQ 出版社 (1991).
9) 二村：計算過程の部分評価—コンパイラ・コンパイラの一方法—，電子通信学会論文誌，Vol. 54-C, No. 8, pp. 721-728 (1971).
10) Rochkind, M. J.: Advanced UNIX Programming, Prentice-Hall, Inc. (1985). (UNIX システムコールプログラミング，アスキー出版社(1987).)
11) 赤坂 隆：数値計算，コロナ社 (1967).
12) Korn, G. A. and Wait, J. V.: *Digital Continuous-System Simulation*, Prentice-Hall Inc. (1978).

(平成4年7月21日受付)
(平成5年7月 8 日採録)

平井 千秋（正会員）

1961 年生．1985 年東京大学工学部精密機械工学科卒業．1987 年同大学院修士課程修了．同年日立製作所システム開発研究所入社．LSI 設計 CAD，制御システム CAD の研究に従事．日本応用数理学会会員．

林 晉一（正会員）

1946 年生．1969 年東京工業大学理工学部電気工学科卒業．1971 年同大学院修士課程修了．同年日立製作所入社．以来，中央研究所を経て，現在，システム開発研究所にて，知識処理による LSI 設計 CAD 研究に従事．電気学会，電子情報通信学会，計測自動制御学会，IEEE-CAS 各会員．