ディジタル回路の記号解析に適した言語的な記述方法

菅原一孔† 林原啓二† 小西亮介†

ディジタル回路の記号解析に適した回路構造および解析内容の言語的な記述法と，それに基づく解析システムについて述べる．ここで提案する方法によると回路構造と解析内容を構造化された汎用の手続き型言語のように記述することができる．回路構造の記述に際しては，全体の回路をいくつかの部分回路の相互接続として表現し，回路全体をメインルーチン，各部分回路をサブルーチンのように記述する．その際各部分回路内の節点番号については部分回路ごとに独立して扱うことが可能であり，その管理は解析システムが自動的に行うため利用者は意識しなくてもよい．また，回路の素子値についても各部分回路ごとに独立して記号または数値として扱うことができる．回路を構成する素子として，線形回路で用いられる定数乗算器，遅延器，加算器以外に正（余）弦発生器や2つの信号の乗算器などを用意した．これにより線形回路の記号回路関数を求められるだけでなく，適応信号処理回路など非線形な回路について時間領域でシミュレーションを行う場合にも本方法は有効である．また，回路構造や解析内容の記述にあたり条件判断文や繰返し文が利用できるだけでなく，サブルーチンの再帰的な呼び出しが可能であるので，効率の良い記述が可能である．

Linguistic Descriptions Suitable for the Symbolic Circuit Analysis of Digital Circuits

KAZUNORI SUGAHARA,† KEIJI HAYASHIBARA† and RYOSUKE KONISHI†

In this paper, a linguistic description of digital circuits suitable for the symbolic circuit analysis and a processing system for it are proposed. By the proposed method, it is possible to describe a digital circuit as a linguistic form and the whole circuit is dealt as a main routine of the ordinal procedural languages and sub-circuits as sub routines. Nodes in the sub-circuits can be managed automatically by the analyzing system without users' management. The proposed descriptions are also effective for time domain simulations of adaptive signal processing circuits because the system is designed to manage multiplier coefficients of each circuit independently. Effective descriptions are also possible for the circuits which have the regular structure because the conditional branch instructions and the repeat instructions and the recursive calls of sub routines are prepared. The compiler-interpreter method is adopted for the basic construction of the proposed system, and yacc/lex is used to develop the system.

1. はじめに

ディジタル信号処理専用LSI（以下，DSP）の処理速度が向上したことや，プログラム開発支援ツールの改良により，DSPを用いたシステムの処理内容をより柔軟に記述できるようになってきた．これを背景に，たとえば適応的な信号処理回路のように，複雑な処理内容をDSPを用いたシステム上で実現しようとする回路が次々に提案されている．

このようなディジタル回路では，回路中の乗算器係数や信号を表現する語長が有限に制限されていることが原因で，実際に実現された回路の応答が理想的なものとは異なってしまうことは従来からよく知られている．これらの問題点は，リミットサイクルの発生のない回路構成の提案，丸め雑音の影響の低い回路構成の提案あるいは低係数感度な回路構成の提案などにより解決がはかられてきた．しかしこれらのほとんどは低次の回路についての考察であり，より複雑な回路構成についての解析は十分に行うことができないのが現状である．

有限語長制限による回路の応答への影響などを総合的に評価するためには回路中の乗算器係数などを記号のままで扱いながら解析を行う，いわゆる記号解析を行うことが有効である．記号解析によると回路パラメータを記号として陽の形で含まれた記号回路関数を

† 鳥取大学工学部
Faculty of Engineering, Tottori University

求めることができる．このため，

- 素子変動による回路特性の算出を，変数の標本値の組合せすべてについて行う必要がない．
- 最適化手法の適用が容易となる．
- 感度，雑音特性など様々な解析が容易となる．

などの特徴があり，各種の回路解析に応用することができる．この点は数値的に指定されたいくつかの回路パラメータの組合せについて，回路特性の分点値を求めて，これから回路の性質を調べる数値解析に比べ有効である．

このような記号解析を行う方法として，アナログ回路についてはパラメータ抽出法や，木アドミタンス積による方法あるいはシグナルフローグラフによる方法などが代表的なものとして知られており，これをディジタル回路に適応することが考えられる．

ところがこれらの方法は数値計算により記述することが前提になっており，

- 記号として取り扱うことのできる素子（記号素子）の数に制限がある．
- 記号素子の数が多くなると，素子値が数値で与えられたものに関して，計算誤差が含まれてしまう．

などの問題がある．

一方，式を記号のまま計算をすることのできる数式処理言語（Symbolic and Algebraic Manipulation Program，以下**SAMP**）の研究が進み，これまでにいくつかの言語が発表されている．このような**SAMP**を積極的に利用することにより，効率良く各種回路の記号解析を行うことが期待できる．しかし，**SAMP**により直接回路解析プログラムを記述することは，

- 回路構造の記述
- 回路中の節点の管理
- 解析内容の記述

などを考慮すると効率の良い方法とはいえない[1),2)]．

本稿では，**SAMP**を積極的に利用してディジタル回路の記号解析を行うためのシステムの構成法と，解析対象の回路構造や解析内容を入力する際の記述方法について検討を加える．まず提案する記号解析システムの全体の流れについて2章で述べた後，3章では，回路構造ならびに解析内容を効率良く記述するための方法について述べる．最後に4章で解析例を用いて本方法の有効性を確認する．

2. システムの構成

提案するシステムの構成を**図1**に示す．図中に示されているとおり，本解析システムは解析コンパイラ部と解析インタプリタ部から構成されている．これらに3章で述べる回路記述方法で表現された回路記述ファイルが入力され，字句解析および構文解析が行われた後，回路解析を行うための**SAMP**プログラムが生成される．最終的な解析結果は，生成された**SAMP**プログラムを実行することにより得られる．このように本解析システムの基本構成はコンパイラ・インタプリタ方式[3)]を採用しており，解析インタプリタ部はスタック機械[3)]を実現している．

なお，解析コンパイラ部は**yacc/lex**言語[4),5)]を用いて，また解析インタプリタ部分の処理についてはC言語を用いて記述した．以下ではそれぞれの処理の様子について説明を加える．

2.1 解析コンパイラ

解析コンパイラでは入力された回路記述ファイルの字句解析ならびに構文解析を行い，回路の解析内容を示す**SAMP**プログラムを生成するための中間コードと文字列スプール領域を生成する．文字列スプール領域には字句解析によって切り出された文字列が格納される．

2.2 解析インタプリタ

解析インタプリタは，解析コンパイラから出力された回路記述部の中間コードを実行し，H_c，H_dを生成する．なお，中間コードの実行の際には，変数名などを管理するために記号管理表を作成する．記号管理表には，以下に示す意味を持つ，**name**，**type**，**val**からなる情報が格納される．

name: ローカル変数名，配列名を格納する．

type: 対応する名前が，どのような型の変数であるかを区別する識別のためのデータを格納する．

val: 対応する名前が持つ値を格納する．

続いて，H_c，H_dの行と列の入替え操作により回路中の各節点の状態が決まる順番を見い出す，いわゆる節点の順序づけ[6)]や，回路の次数の決定を行う．次に，解析処理記述部の中間コードを実行しながら，以下の処理を行う**SAMP**プログラムをファイルに出力する．

まず，時間領域での応答を求める場合，

(1) 回路記述ファイルで指定された入力節点に指定された信号を入力する．

(2) 指定された出力節点の時間応答を

$$y(n) = H_c y(n) + H_d y(n-1) + x(n) \quad (1)$$

の関係により求める．ここで$x(n)$は時刻nにおける各節点への入力信号であり，$y(n)$は各節点の状態である．ともに(総節点数)×1の大きさのベクトルである．またH_c，H_dは(総節点数)×(総節点数)の大きさの行列であり，それぞれ遅延器を除く素子と遅延器のみの係数行列を表す．

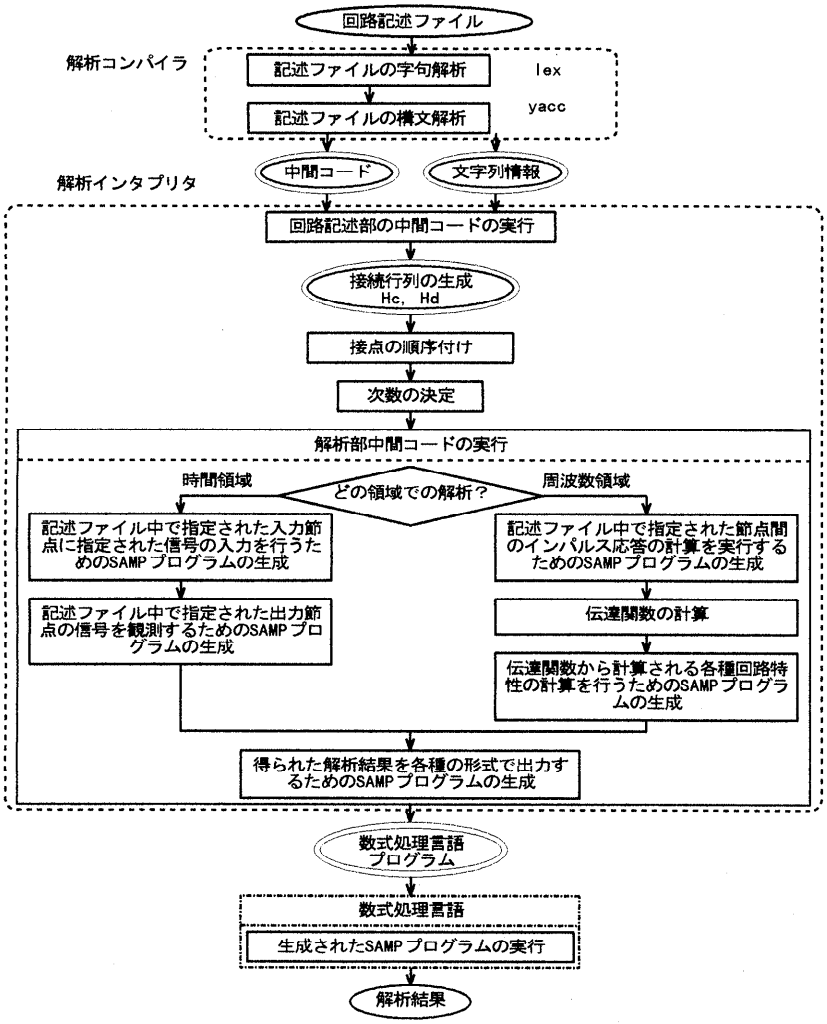


図1 システムの流れ図
Fig. 1 Flow chart of the proposed system.

(3) 回路記述ファイル中で指示された形式で出力する.

次に周波数領域での解析を行う場合,

(1) 回路記述ファイルで指定された入力節点にインパルスを入力する.

(2) 指定された出力節点のインパルス応答を式(1)により求める.

(3) 求めた時間応答から入力節点, 出力節点間の伝達関数を求める.

(4) 伝達関数から各種特性を求める.

(5) 回路記述ファイル中で指示された形式で出力する.

ただし記号解析を行うためには, ここで述べた計算はすべて式の形で実行されなければならない. 提案方法ではこれを **SAMP** を積極的に利用して行うこととし, 解析システムは一連の解析処理を行うための **SAMP** のプログラムを自動生成する, いわゆるプログラムジェネレータの働きをする. ここでは **SAMP** の例として比較的利用の多い Mathematica を取り上げるが, コード生成部だけを入れ替えれば他の **SAMP** にも容易に対応することができる.

2.3 生成された中間コードの例とその実行の様子

中間コードの実行の様子を説明するために, 乗算器1つからなる簡単な回路を考える. この回路を3章で提案する記述方法で表現したものを**図2** に示す.

図2を解析コンパイラにより処理すると, **表1**, **表2** と**図3** に示す中間コード, 記号管理表および文字列スプール領域が生成される. **図4** に図2の6行目に

```
 1:circuit()
 2:{
 3: setten=2;
 4: int i=1;
 5: string s[10];
 6: strcpy(s, strcat("a", itoa(i)));
 7: multi(1,2,s);
 8:}
 9:analysis()
10:{
11: index s;
12: s=transf(1,2);
13: show(s){
14:  type:expression;
15: }
16:}
```

図2 1つの乗算器からなる回路の記述
Fig. 2 Linguistic description of a circuit which contains only one multiplier.

表1 生成された中間コード
Table 1 Generated intermediate code.

line	intermediate code	subcode1	subcode2
0	OP_CIRCUIT	0	0
1	OP_SETTEN	2	0
2	OP_INT	8	0
3	OP_PSHN	1	0
4	OP_POP	8	1
5	OP_STRING	10	10
6	OP_PSHS	12	1
7	OP_PSHS	14	0
8	OP_PSHI	16	0
9	OP_STROPE	0	3
10	OP_STROPE	0	2
11	OP_STROPE	0	-1
⋮			
16	OP_END	0	0
17	OP_ANALYSIS	20	0
18	OP_INDEX	29	0
⋮			
28	OP_END	0	0

表2 生成された記号管理表（一部）
Table 2 Generated intermediate code.

行	name	type	data
1	i	Int_Var	1
2	s	String_Array	
⋮	⋮		

対応する中間コードを実行しているときのスタックの動作の様子を示す．なお，この部分は乗算器係数を文字列関数を利用して生成する部分であり，表1では6行目から11行目に対応する．

表1中，6行目では文字列スプール領域中 subcode1(=12) で指される文字列 (="s") と同じ名前の文字列が記号管理表から取り出されスタックに格納される．同様に7行目では文字列スプール領域中 subcode1(=14) で指される文字列 (="a") そのものがスタックに格納される．このように **OP_PSHS** は文字列をスタックに格納するための中間コードであり，その動作は subcode2 の値により制御される．続く8行目では文字列スプール領域中，subcode1(=16) が指す場所の変数名と同じ名前を記号管理表で検索しその値がスタックに積まれる．この後，subcode2 の値に対応して数値の文字列変換，文字列の連結あるいは文字列のコピーなどを行う中間コード **OP_STROPE** を実行し，以上の結果として対応する乗算器の係数が生成される．

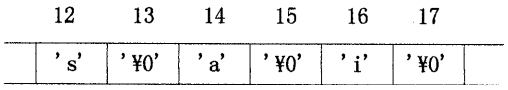


図3 文字列スプール領域
Fig. 3 Strings spool area.

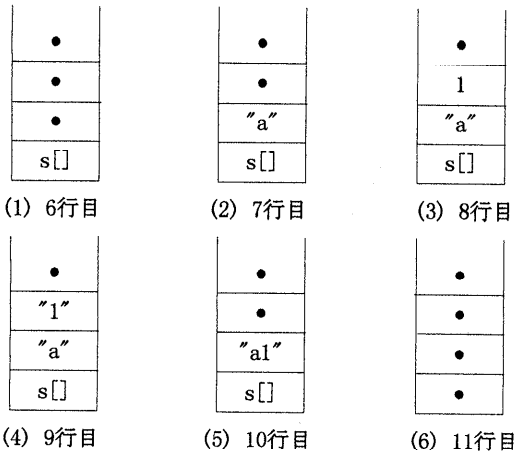


図4 スタックの動作の様子
Fig. 4 Operation of the stack area.

3. 回路構造と解析内容の記述

ディジタル回路の場合，回路構造の記述は回路中の各節点に番号をふり，各節点間の係数行列を記号の形で生成することがこれにあたる[7]．係数行列を記述する方法として，各節点間の素子に着目して，それの構造を羅列的に記述していく方法[8),9)]がある．また，信号処理専用 LSI の開発に用いられるシリコンコンパイラで採用されている SILAGE[10)]を代表とするいくつかの回路構造の記述言語も提案されている．

これらの方法は素子が接続されている節点番号と素子値などを羅列的に記述してゆくもので，簡便なため分かりやすい．しかしこの記述方法では回路内のすべ

表3 提案する記述方法の命令（抜粋）
Table 3 Command.

命令の種類	命令	処理内容
素子記述	multi(t1,t2,str){fnum}	入力節点がt1，出力節点がt2，係数が文字列strである乗算器を表す．乗算器係数値が指定されている場合は実数fnumにその値を書き込む．計数値が未定の場合省略する．
	delay(t1,t2)	入力節点がt1，出力節点がt2である遅延器を表す．
	adder(t1,t2,t3)	入力節点がt1，t2，出力節点がt3である加算器を表す．
	sub(t1,t2,t3)	入力節点がt1，t2，出力節点がt3である減算器を表す．
	sin(t1,t2), cos(t1,t2)	入力節点がt1，出力節点がt2である正弦あるいは余弦信号発生器を表す．
	inverse(t1,t2)	出力節点がt2に入力節点がt1の逆数を出力する素子を表す．
	ccmulti(t1,t2,t3,str,a/r){fnum}	入力節点がt1，出力節点がt2，係数が文字列strである可変乗算器を表す．節点t3と乗算器係数strの関係付けを行い，a/rによりadjust，replaceを行う．
	sep(t1,t2)	節点を分離して入力節点をt1，出力節点をt2にする．
	sigmulti(t1,t2,t3)	入力節点がt1，t2，出力節点がt3の信号同士の乗算器を表す．
解析内容	timesim(time1,time2){timesim_option}	条件timesim_optionを考慮に入れ，時刻time1からtime2について時間領域でのシミュレーションを実行する．
	transf(t1,t2)	節点t1からt2の伝達関数を求める．
	amp_res(id),pha_res(id)	index変数idから周波数特性（振幅特性あるいは位相特性）を求める．
	amplitude(id)	index変数idから振幅を求める．
	dif(id,str)	index変数idの結果を文字列strで偏微分する．
	substitution(id,str)	index変数idに文字列strをセットする．
	show(id){show_option}	index変数idをshow_optionに基づいて表示する．
文字列操作	strcpy(destination,str)	文字型配列destinationに文字列strをコピーする．
	strcat(str1,str2)	文字列str1とstr2を連結する．
	itoa(num or var)	数値numまたは変数varの値を文字列に変換する．
制御文	while(condition){stmt}	条件文conditionが真の間命令文stmtを繰り返し実行する．
	for(stmt1;condition;stmt2){stmt}	まず命令文stmt1を実行し，条件文conditionが真の間命令文stmtを繰り返し実行する．繰返しの際の初期化として命令stmt2を実行する．
	if(condition){stmt1} else {stmt2}	条件文conditionが真のときは命令文stmt1を実行し，偽のときは命令文stmt2を実行する．else以下は省略できる．
宣言文	int var or var[num]	整数型変数varまたは配列var[num]を宣言する．
	str var[num]	文字型配列var[num]を宣言する．
	float var or var[num]	浮動小数点型変数varまたは配列var[num]を宣言する．
	term t or t[num]	節点tまたはt[num]を宣言する．宣言された節点はシステムが管理する．
	index id	解析結果を入れる変数idを宣言する．
代入文	var = stmt	四則演算文stmtの計算結果を変数varに代入する．
四則演算	+,−,*,/	一般の演算と同じ．しかし，index変数が混在の演算はMathematicaプログラムに出力される．
比較演算	⟩,⟨,==,!=,⟩=,⟨=	一般の演算と同じ．
コメント	/* ..... */	'/*'と'*/'で囲まれた部分をコメントにする．
その他	setten = num	主回路中の総節点数numを大域変数settenに設定する．
	return (num or var)	数値numまたは宣言文で宣言したあらゆる変数varを呼び出し関数に返す．

ての節点番号や素子値などについて利用者が把握しておく必要があり，その管理に多大な手間が必要となる．また，各種ディジタル回路が持つさまざまな特性を調べるために行う記号解析を，効率的に記述しようとする本提案方法と目的が異なり，記号回路パラメータの管理ができないなど，そのまま適用することはできない．

本章ではこれらの問題点を解決するために，

- 手続き型プログラミング言語のように「繰返し」，「条件判断」などの制御構造および「関数・手続き」や「再帰的呼び出し」の考え方を導入することにより，効率の良い記述を行えるようにする．
- 回路全体や解析内容を複数の部分回路や処理の集まりで記述し，これらの相互の呼び出しにより階層的に表現できるようにする．
- 回路構造や解析内容の記述を柔軟に行えるようにするために，後述する**term**型変数や**index**型変数を新たに導入する．

ことなどを行った．そして入力データを単なる接続情報の羅列ではなく，回路構造や解析内容を記述する一種の言語として取り扱う方法を提案する．提案する記述方法で利用できる命令の抜粋したものを**表3**にまとめる．記述の際の文法は，表記方法が比較的簡便でありかつ最近特に利用の多い言語であるC言語に準ずる

ものになるよう工夫した．回路構造と解析内容はそれぞれ **circuit()** と **analysis()** と呼ばれる特別な名前の関数から記述が始まる．この2つの関数は記述ファイル中に必ず存在しなければならない．**circuit()** 関数（以下，主回路）は回路全体を表し，各種の回路素子や部分回路の相互接続で表現される．また，**analysis()** 関数は解析処理全体を指し，いくつかのより細かな解析処理の集まりとして記述される．

3.1 回路構造の記述について

利用者が回路中すべての節点の番号を管理して，これに基づき回路の構造の記述を行うことは単純ではあるが，不用意な記述の誤りが発生しやすかったり，あるいは少しの回路構造の変更でもはじめから記述し直さなければならなくなるなど効率の良い方法であるとはいえない．回路全体をいくつかの部分回路で表現し，その内部の節点については解析システムが自動的に節点番号をふり，それに関する情報の管理を自動的に行うシステムが望まれる．このような記述方法は単に表現が簡単になるだけでなく，回路構造の記述の不用意な誤りを防ぐことにも役立つものと思われる．

回路の節点番号などをシステムが自動的に管理する場合，まず問題になることに，部分回路内の節点や素子名を利用者がどのように表現するかということがある．部分回路からさらに自分自身あるいは他の部分回路が呼び出される場合でも柔軟に対応する必要がある．この点に関して提案方法では，部分回路中の節点を表現するために，**term** 型変数を導入した．**term** 型変数が宣言されると，解析システム内部では節点の総数を表す変数の値を “1” だけ増加させ，同時に2.2節で述べた記号管理表に対応する節点の記憶場所を確保する．

部分回路内の節点番号や素子名は，それが含まれる部分回路名と節点名を使って記述する．たとえば，主回路中の部分回路1から呼び出された部分回路2に含まれる節点 **t1** は

```
circuit/部分回路1/部分回路2/t1
```

のように記述することにする．ここで **t1** は，**term** 型の変数として部分回路2中で宣言されたものである．

3.2 解析内容の記述について

回路を数値解析する場合は，数値で表される回路関数の分点値のみを解析結果として求めればよい．この点，記号解析の場合は，記号回路関数に含まれる各素子値について，それを記号として扱うのか数値として扱うのか指定する必要がある．またこれらの素子値を，どのようにして記号あるいは数値として記述するか．また，どのような解析処理を行うのかなど，膨大な数のパラメータの組合せが存在する．このために効率の良い解析内容の記述方法が必要となる．

ここでは解析内容の柔軟な記述を目的として，新たに **index** 型の変数を導入する．これには **SAMP** の実行時に得られる結果を格納し，その後の解析に利用するものである．この型の変数には解析結果として得られた数値や数式だけでなく，回路特性を表現するグラフなども格納される．また，数値や数式が格納されている場合，この変数による演算を記述することも可能であるため，たとえば伝達関数から感度特性を求める場合などでは大変柔軟な記述ができる．これについては，4章で例題を使って詳しく説明する．

4. 記 述 例

ディジタル信号処理の分野で頻繁に用いられる **LMS** 適応信号処理回路を記述例として取り上げ，これの様々な特性の解析を提案する記述法で表現し，提案方法の有効性を確認する．なお，この回路そのものの評価はここでの目的ではなく，それの回路構造ならびに解析内容の記述方法を確認することが目的であるため簡単な回路を取り上げた．

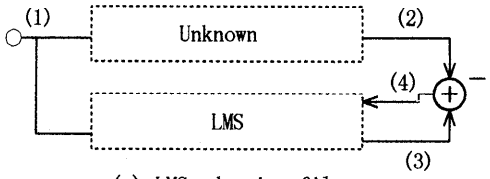
図5 に例で取り上げた回路を示す．この回路は2次のトランスバーサル型フィルタで表現された未知のシステム **Unknown** を，**LMS** 適応アルゴリズムを実現する回路 **LMS** により同定しようとしているところである．図5では，本方法の特徴の1つである階層的な回路の記述方法の有効性を示すために，システム全体を **Unknown** 部分回路と **LMS** 部分回路の相互接続として表し，さらにそれぞれはいくつかの部分回路により表現している．

4.1 回路構造の記述例

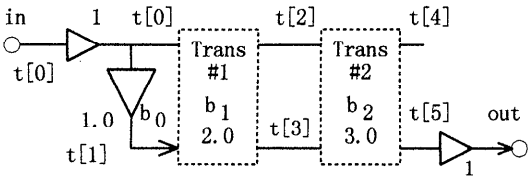
図5を提案方法で記述すると**図6** のようになる．この記述によると主回路中に節点は4つ存在し，利用者はこの4つの節点番号のみ管理すればよい．各部分回路内の節点はそれぞれ11，24，31，44，51行目の **term** 文により宣言され，その番号はシステムにより自動的に管理される．

図6中，3行目の **setten** 命令で主回路中の総節点が設定され，それに基づき5行目から7行目で各部分回路が接続されている．4行目の **float** 型の配列 **fnum** は **Unknown** 部分回路中の乗算器係数であり，ここではこの値を **LMS** 部分回路により同定しようとしている．

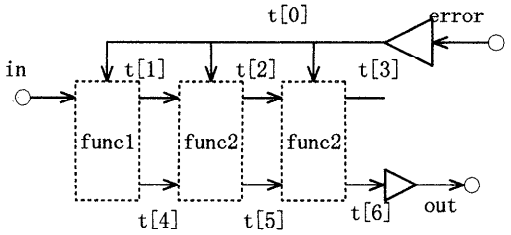
9行目からは各部分回路の記述であり，引数は通常のC言語と同様，ユーザが自由に宣言できるものである．まず **Unknown** 部分回路中，内部の節点は11



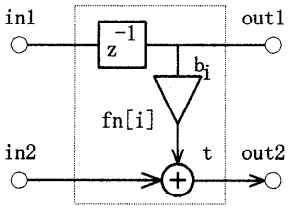
(a) LMS adaptive filter.



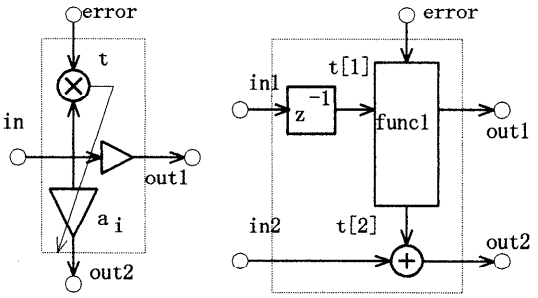
(b) Unknown sub-circuit.



(c) LMS sub-circuit.



(d) i-th trans sub-circuit.



(e) func1 sub circuit.

(f) func2 sub circuit.

図5 解析例（LMS 適応信号処理回路）
Fig. 5 Example.

```
 1:circuit()
 2:{
 3: setten=4;
 4: float fnum[3]={1.0, 2.0, 3.0};
 5: Unknown(1,2,fnum);
 6: LMS(1,3,4);
 7: sub(2,3,4);
 8:}
 9:Unknown(term in,term out
          ,float fnum1[])
10:{
11: term t[6];
12: string s[10];
13: int i;
14: sep(in,t[0]);
15: sep(t[5],out);
16: multi(t[0],t[1],"b0"){fnum1[0]};
17: for(i=1; i<=2; i++){
18:  strcat(strcpy(s, "b"), itoa(i));
19:  Trans(t[2*i-2],t[2*i-1],t[2*i]
          ,t[2*i+1],s,fnum1[i]);
20: }
21:}
22:Trans(term in1,term in2,term out1
       ,term out2,string s1[],float f)
23:{
24: term t;
25: delay(in1,out1);
26: multi(out1,t,s1){f};
27: adder(in2,t,out2);
28:}
29:LMS(term in,term out,term error)
30:{
31: term t[7];
32: string s[10] = "a0";
33: int n;
34: sep(t[6],out);
35: func1(in,t[0],t[1],t[4],s);
36: for(n=1; n<=2; n++){
37:  strcpy(s,strcat("a",itoa(n)));
38:  func2(t[n],t[n+3],t[0],t[n+1]
          ,t[n+4],s);
39: }
40: multi(error,t[0],"myu"){0.01};
41:}
42:func1(term in,term error,term out1
        ,term out2,string s1[])
43:{
44: term t;
45: sep(in,out1);
46: sigmulti(in,error,t);
47: ccmulti(in,out2,t,s1,a){0.1};
48:}
49:func2(term in1,term in2,term error
     ,term out1,term out2,string s1[])
50:{
51: term t1,t2;
52: delay(in1,t1);
53: func1(t1,error,out1,t2,s1);
54: adder(in2,t2,out2);
55:}
```

図6 提案方法による図1の記述（回路記述部分）
Fig. 6 Linguistic description of Fig.1 by using proposed method.

行目で **term** 型の配列 **t** で表されている．図5に示したようにこの **Unknown** 部分回路は2つの **Trans** 部分回路からなり，17行目から20行目でその様子が繰返し文により記述されている．18行目で各 **Trans** 部分回路中の乗算器係数が文字列処理関数により生成され，その結果が19行目の **Trans** 部分回路を呼び出す際の引数として受け渡されていることにご注意いただきたい．また，17行目の **for** 文中の繰返し回数を決定する値 "2" が回路の次数に対応している．同様

に **Trans** 部分回路，**LMS** 部分回路，**func1** 部分回路，**func2** 部分回路がそれぞれ 22，29，42，49 行目から記述されている．

4.2 解析内容の記述例

解析内容の記述例を図 **7**，**8** に示す．この記述例についてどのような解析がされているのか，記述例にそって説明を加える．

59 行目　**Unknown** 部分回路の入力節点 (1) と出力節点 (2) 間の伝達関数を求め，結果を **index** 変数として宣言された **t1** に代入している．この操作により **t1** には，式の形で求められた伝達関数が格納される．

```
56:analysis()
57:{
58: index t1,t2;
59: t1 = transf(1,2);
60: tf_result1(t1);
61: tf_result2(t1);
62:
63: sensitiv(t1);
64:
65: amplitude_res(t1);
66:
67: t2 = lms_ts1();
68: lms_result1(t2);
69: lms_result2(t2);
70:
71: t2 = lms_ts2();
72: lms_result1(t2);
73:}
74:tf_result1(index t)
75:{
76: show(t){
77:  type : expression;
78:  symbol : "b0","b1","b2";
79: }
80:}
81:tf_result2(index t)
82:{
83: show(t){
84:  type : expression;
85:  symbol : "b1";
86:  numerical : "b0","b2";
87: }
88:}
89:sensitiv(index t)
90:{
91: index s1, s2, s3, s4;
92: s1 = amplitude(t);
93: s2 = dif(s1, "b2");
94: substitution(s3, "b2");
95: s4 = s2*(s3/s1);
96: sensi_result(s4);
97:}
98:sensi_result(index t)
99:{
100: show(t){
101:  type : expression;
102: };
103:}
```

図 7　提案方法による図 1 の記述（解析内容続く）

Fig. 7　Linguistic description of Fig.1 by using proposed method.

60 行目　求められた伝達関数を，すべての乗算器係数を記号として扱った場合の結果を出力するために，74 行目から 80 行目で記述された関数 **tf_result1** を実行している．

61 行目　同じ伝達関数を b_1 だけ記号として扱い，b_0，b_2 は回路記述部分で指定された数値を使って表現した結果を出力するために，81 行目から 88 行目で記述された関数 **tf_result2** を実行している．

63 行目　さらに **t1** を使って，$\frac{b_2}{|H|}\frac{\partial |H|}{\partial b_2}$ で与えられる振幅特性の b_2 に関する係数感度を求めるために，89 行目から 97 行目で記述された **sensitiv** 関数を実行している．結果は **sensitiv** 関数内で **sensi_result** 関数が呼ばれすべての変数を記号として扱い出力している．

以上 3 つの解析結果はそれぞれ，図 **9** の 1–4，5–8，

```
104:amplitude_res(index t)
105:{
106: index t1;
107: t = amp_res(t);
108: show(t1){
109:  axislabel : "w", "|H|";
110: }
111:}
112:lms_ts1()
113:{
114: index t1;
115: t1 = timesim(0,700){
116:  signalin(1,"Random[]");
117:  probe(4);
118: }
119: return(t1);
120:}
121:lms_ts2()
122:{
123: index t1;
124: t1 = timesim(0,700){
125:  signalin(1,"Random[]");
126:  round_fix(8,5);
127:  probe(4);
128: }
129: return(t1);
130:}
131:lms_result1(index t)
132:{
133: show(t){
134:  h_axis    : 0, 700;
135:  v_axis    : 0.0, 5.0;
136:  axislabel : "t", "|node4|";
137: }
138:}
139:lms_result2(index t)
140:{
141: show(t){
142:  h_axis    : 500, 600;
143:  v_axis    : 0.0, 1.0;
144:  axislabel : "t", "|node4|";
145: }
146:}
```

図 8　提案方法による図 1 の記述（解析内容続き）

Fig. 8　Linguistic description of Fig.1 by using proposed method.

```
1:       b2    b1
2:{b0 + -- + --}
3:        2    z
4:       z

5:       3.    b1
6:{1. + -- + --}
7:        2    z
8:       z

9:    b2 (b2 + b1 Cos[w]
                    + b0 Cos[2 w])
10:{---------------------------------}
11:   2      2      2
12: b0  + b1  + b2  + 2 b1 (b0 + b2)
             Cos[w] + 2 b0 b2 Cos[2 w]
```

図9 解析結果
Fig. 9 Results.

9–12行に示しているように，正しく求められている．

65行目 **t1** の振幅特性を図示するために104行目から111行目で記述された **amptitude_res** 関数を実行している．ただし，**amptitude_res** 関数中，**amp_res** 命令は表3にもあるとおり，すべての乗算器係数に回路記述部分で指定された数値を代入して求めた振幅特性をグラフとして求めるものである．なお注意が必要なことは107行目の **index** 型の変数 **t** はグラフが格納されるが，実際に出力しているのは108行目から110行目の **show** 命令である点である．ここでは軸の名前や出力範囲を自由に設定しながら望む結果を出力している．

67行目 節点(1)に乱数を入力した際の節点(4)の応答を，時刻0から時刻700まで調べるために，112行目から120行目で記述された **lms_ts1** 関数を実行している．結果は **t2** に格納され，68，69行目で出力されている．

68行目 回路のおおまかな応答を調べるために，**lms_result1** 関数により，**t2** として求めた時間応答のうち0～700単位時間について出力している．

69行目 収束後の応答について詳しく調べるために，**t2** として求めた時間応答のうち，500～600単位時間について応答を拡大して出力する，**lms_result2** を実行している．なお，**lms_result2** 関数では67行目で求めた時間応答結果の表現方法を変更しているのであって，改めて時間応答を求める作業をしているわけではない．

71，72行目 有限語長での応答を調べ出力している．そのために，121行目から130行目で記述された **lms_ts2** 関数を実行している．**lms_ts2** 関数中，126行目で係数語長を設定している．結果の表示については先と同じ **lms_result1** 関数を利用している．

以上4つの解析結果は図 **10**～**13** に示すよう，正しく求められている．

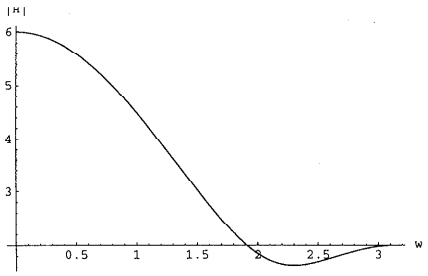


図10 解析例2（振幅特性）
Fig. 10 Example2.

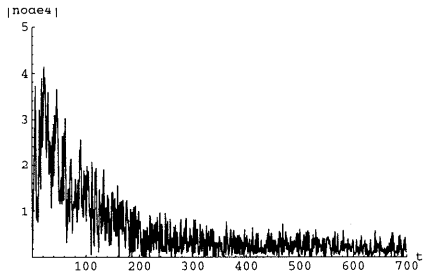


図11 解析例3（語長制限なし）
Fig. 11 Example3.

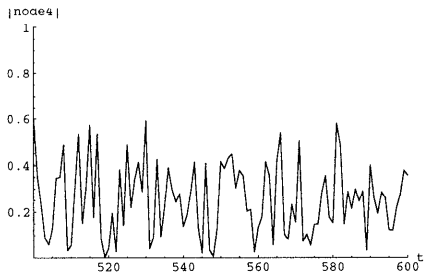


図12 解析例4（図11の500–600間の拡大表示）
Fig. 12 Example4.

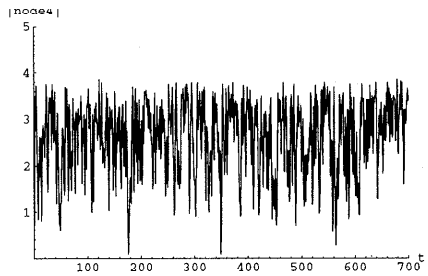


図13 解析例5（語長8 bit，小数点以下5 bit）
Fig. 13 Example5.

このように提案する記述法では，一度求めた結果をもとに新たな特性を式の形で記述して求めることができたり，解析結果を式やグラフなど，様々な形式で柔軟に出力することができる特徴を持つ．

5. おわりに

本稿ではディジタル回路を記号解析するためのシステムの構成方法と，回路構造や解析処理内容を言語的に記述する方法について述べた．

回路構造の記述にあたっては，従来の手続き型言語におけるメインルーチンとサブルーチンのように，1つの主回路といくつかの部分回路の相互接続として記述することができる．その際各部分回路の節点番号やその中に含まれる乗算器の係数はシステムが管理するため，利用者は回路ごとに独立して指定することができる．

また，本方法では繰返し命令や条件判断命令を用意しているだけでなく部分回路の再帰呼び出しも可能であるので，規則的な回路構造を持つ回路の場合大変効率よく記述することができる．本解析法の有効性を確認するために，ディジタル信号処理の際に頻繁に利用される回路を例にとり，解析例を示した．

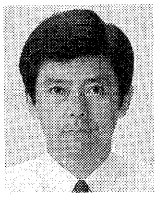
今後は論理回路や分布定数回路の解析に，ここで提案した方法を応用することを検討してゆく予定である．

謝辞 貴重なご意見をいただきました査読者の方々に感謝いたします．また本研究の一部は平成9年度文部省科学研究費助成金（基盤研究(C)）による．

参考文献

1) 菅原一孔，柿本秀樹，堂野正弘，落山謙二：数式処理プログラムによる線形回路の記号解析，情報処理学会論文誌，Vol.30, No.6, pp.779–785 (1989).
2) 松本康宏，菅原一孔，小西亮介：線形回路の言語的表現とその記号解析への応用，情報処理学会論文誌，Vol.35, No.11, pp.2403–2413 (1994).
3) 佐々政幸：プログラミング言語処理系，岩波書店 (1989).
4) 五月女健治：*yacc/lex*，啓学出版 (1992).
5) Levine, J.R., Mason, T. and Brown, D.: lex & yacc プログラミング，アスキー出版局 (1994).
6) 菅原一孔，長谷川隆之，高沢智志：ディジタル回路解析のための計算順序の決定法，信学論（A），Vol.J67-A, No.2, pp.149–150 (1984).
7) Crochiere, R. and Oppenheim, A.: Analysis of Linear Digital Networks, *Proc. IEEE*, Vol.63, pp.581–595 (1975).
8) 杉野暢彦，年清昭彦，渡部英二，西原明法：ディジタル信号処理回路の計算順序決定とそのシグナルプロセッサ用コンパイラへの応用，信学論（A），Vol.J71-A, No.2, pp.327–335 (1988).
9) Tuinenga, P.W.: *SPICE—A Guide to Circuit Simulation & Analysis Using Pspice*, Prenticc Hall (1988).
10) Hilfinger, P.: A high-level language and silicon compiler for digital signal processing, *Proc. IEEE Custom Integrated Circuits Conference Portland OR*, pp.213–216, Addison-Wesley (1985).

(平成 9 年 6 月 27 日受付)
(平成 10 年 1 月 16 日採録)



菅原 一孔（正会員）

昭和31年生．昭和56年東京工業大学大学院理工学研究科電子物理工学専攻修士課程修了．同年神戸市立工業高等専門学校電気工学科講師．同助教授，同校電子工学科助教授を経て平成6年鳥取大学工学部電気電子工学科助教授．電気，電子回路理論，数式処理言語の応用技術，画像処理，ディジタル信号処理などに関する教育，研究に従事．工学博士．IEEE，電子情報通信学会各会員．

林原 啓二

昭和49年生．平成8年鳥取大学工学部電気電子工学科卒業．現在同大学院博士前期課程に在籍．回路網の解析手法への数式処理言語の応用に関する研究に従事．

小西 亮介

昭和21年生．昭和43年神戸大学工学部計測工学科卒業．昭和45年同大学院修士課程修了．同年鳥取大学工学部電子工学科助手．現在同教授．センサシステムの開発，画像手法を取りいれた音声認識およびDSPの計測工学への応用技術に関する研究に従事．工学博士．応用物理学会，計測自動制御学会，電気学会各会員．