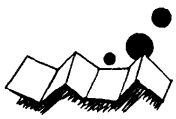
解　説



論理回路の記号シミュレーション†

斎藤隆夫††　上原貴夫††

1. まえがき

最近，知識工学の技術を応用した新しいCADシステムの研究がはじまり，注目を集めている．これは，設計のエキスパートの知識をコンピュータで扱いやすい形に表現し，設計支援のツールとして活用しようとするものである．そのためには，記号処理の技術が必要となる．すなわち，エキスパートの知識と，それに基づく判断を，コンピュータ上の記号処理の過程としてモデル化しなければならない．

論理設計のCADツールの代表としてシミュレータを取り上げてみよう．これは，設計された論理回路の動作をチェックして，設計ミスを発見する目的で作られたツールである．このシミュレータに，それを使うエキスパートの知識や判断を付け加えたCADシステムがあれば，初心者でも効率よく設計ミスを発見できるであろう．著者等は，このような構想のもとに推論機能を持つ記号シミュレータの研究開発を進めている．本文では，そこで用いられた記号処理技術，マンマシンインタフェース等について解説する．

記号シミュレーションとは，通常の数値を用いたシミュレーションに対して，変数名のままシミュレーションを行う方法のことを言う．まず，ソフトウェアの記号シミュレーションが最初に検討された[1),2)]．これに続いて，マイクロプログラム[3)~6)]やハードウェアの機能設計[7)~9)]に対しても適用された．ハードウェアの機能設計も高級言語で書かれたので，ソフトウェアの場合と同様な技術が用いられた．記号シミュレーションで特にやっかいなのは，IF文等による分岐である．変数値が未定で分岐先が決まらない場合，たいていのシステムでは使用者に質問する．すなわち，シミュレーションは使用者とシステムの会話により進行する．

これに対して，以下で述べるような論理回路の記号シミュレーションに関しては，あまり報告がない．対象が論理回路図で表現され，並列に動作するため，上記の対象とは異なる技術やマンマシンインタフェースが必要となる．

2. 記号シミュレーション

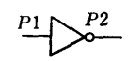
論理回路の記号シミュレーションでは，回路素子の出力は，この素子の入力に与えられた記号値からなる論理式で表わされる．**図-1** に，回路素子とこの機能を表わす表，および，対応するLISPプログラムの例を示した．例えば，ANDゲートでは，入力値に0があれば0，いずれかが1なら他方の入力値が出力値となるが，双方ともに記号 a, b の場合は論理式 $(a\&b)$ が出力値となる．(記号～，&，|は否定，論理積，論理和を表わす．図中で，(DEFUN SAND ($P1$ $P2$) 式) は，引数 $P1$ と $P2$ をもつ関数 SAND の値が式を実行した結果であることを定義している．また，(COND (式$_1$ 式$_2$) (式$_3$ 式$_4$)…) は式$_1$ が満たされれば式$_2$ の値を，式$_1$ が成立せず式$_3$ が満たされれば式$_4$ の値を返す構文である．)

この記号演算を素子の接続関係に従って実行すれば，各素子の出力値を求めることができる．例えば，**図-2**(a) の回路に信号値 a, b, 1 を入力した場合の出力は，**同図**(b)に示すごとく，論理式 $a\&((\sim a)\mid b)$ となる．しかし，このままでは信号が伝播する素子が増えるにつれ，シミュレーションの結果は複雑な論理式となる．このため，**図-3** に例示した式の簡単化規則を適用して，図-2(b)の出力を $a\&b$ とする (**同図**(c))．

このように信号値として記号を用いると，数値シミュレーションに比べて，設計者の考えに近いレベルでシミュレーションを行える．さらに，すべての設計ミスを発見しようとすれば，種々の入力を与えて回路の動作を確認しなければならないが，記号を用いればシミュレートする場合の数は少なくて済む．**図-4, 5** は，ADdress Set が1なら X の値を，BRAnch ま

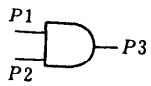
† Symbolic Simulation of Logic Circuits by Takao SAITO and Takao UEHARA (Software Laboratory, FUJITSU Laboratories Ltd.).

†† (株)富士通研究所ソフトウェア研究部第一研究室



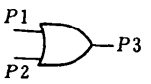
P1	P2
0	1
1	0
a	$\sim a$

```
(DEFUN SNOT (P1)
       (COND ((EQUAL P1  0) 1)
             ((EQUAL P1  1) 0)
             (T  (LIST '~  P1))))
```



P1 / P2	0	1	a
0	0	0	0
1	0	1	a
b	0	b	a & b

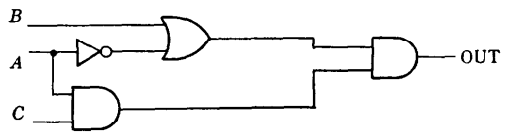
```
(DEFUN SAND (P1  P2)
       (COND ((EQUAL P1 0)   0)
             ((EQUAL P2 0)   0)
             ((EQUAL P1 1) P 2)
             ((EQUAL P2 1) P 1)
             (T  (LIST P1 '&  P2))))
```



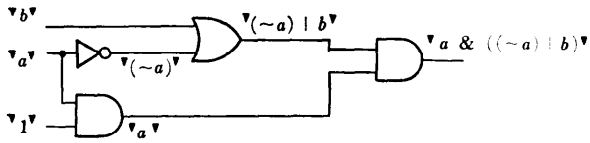
P1 / P2	0	1	a
0	0	1	a
1	1	1	1
b	b	1	a | b

```
(DEFUN SOR  (P1  P2)
       (COND ((EQUAL P1 0) P2)
             ((EQUAL P2 0) P1)
             ((EQUAL P1 1)  1)
             ((EQUAL P2 1)  1)
             (T  (LIST P1 '|  P2))))
```

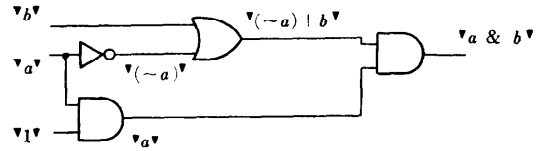
図-1 記号シミュレーションのための機能定義



(a) 回路図



(b) 記号シミュレーション



(c) 簡単化を含む記号シミュレーション

図-2 記号シミュレーション

たは BRanch Positive で NEGative でなければ Y の値をレジスタRにセットする回路の数値シミュレーション例と記号シミュレーション例である．前者では，XとRの値が同じであり設計ミスを見過ごしてしまう可能性がある．一方，後者では結果は $(x \mid y)$ であり，容易に期待値xと異なることを確認できる．また，記号値 x は信号線 X のすべての値を代表しており，数値シミュレーションにおける 2^{10} 通りの場合を1回で確かめられる．

記号シミュレーションを行う場合には，一般に，データ・パスを記号値に，制御信号を数値に設定すれば，理解しやすい結果を得ることができる．例えば，**図-6** に示すポジティブ・エッジ・トリガタイプのDフリップ・フロップのクロック入力が0からaに変化したとする．この場合，出力はクロックが変化しない場合と立ち上がった場合を統合した $(a\&d) \mid ((\sim a)$ & Qold) なる論理式が出力値となる．aの代わりに1または0とすれば，出力はdまたは以前の出力値 Qold となり，理解しやすくなる．

1) A & $A=A$
2) A & $\sim A=0$
3) A & $(\sim A \mid B)=A$ & B
4) A & $(A \mid B)=A$
5) $A \mid A=A$
6) $A \mid \sim A=1$
7) $A \mid (\sim A$ & $B)=A \mid B$
8) $A \mid (A$ & $B)=A$

図-3 式の簡単化規則の例

3. フレームとデモン

3.1 フレームによる論理回路の表現

著者等の記号シミュレータでは，フレームと呼ばれるデータ構造を用いて，対象回路を表現している．フレーム[10)~12)]は人工知能の分野で知識表現に用いられ，**図-7**(a)に示すリスト構造の枠組みで情報を表わす．例えば，**同図**(b)の記述はフレームにより**同図**(c)のように表現できる．このように，フレーム名は記述対象を表わし，スロットには記述対象の属性が与えられる．属性の値は VALUE ファセットに格納される．スロットの値として，他のフレーム名を与えることにより，フレームのネットワークを形成することが

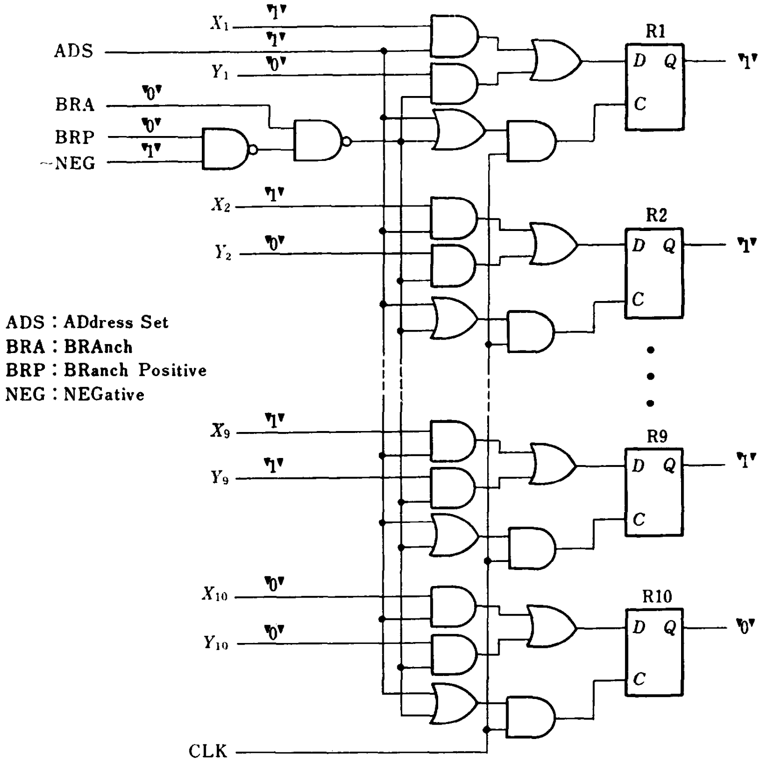


図-4 数値シミュレーション例

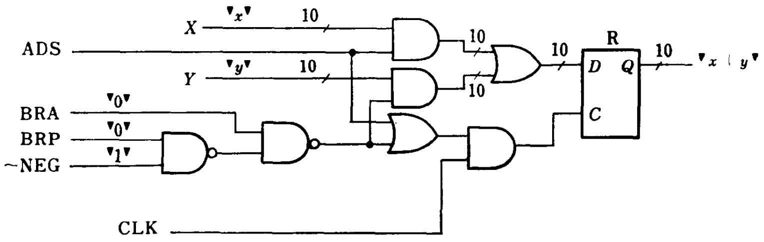
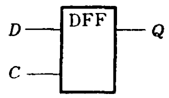


図-5 記号シミュレーション例



Q := C &~Cold & D
 | ~(C & ~Cold) & Qold
但し，Cold, Qold はCとQの前の時刻の値を表わす．

図-6 DFF の機能表現

できる．このため，階層的情報を容易に表現できる．図-8 に8ビットのシフトレジスタ (SFT 8) の階層的定義を，図-9 に対応するフレーム表現を示した．

図-8 の回路の階層は，'タイプ' と，'コンポーネント' の概念を用いて定義されている[13]．つまり，回路は1つのタイプに対応し，これを構成する素子はコンポーネントと呼ばれる．(例えば，タイプ SFT 8 のコンポーネントは SFT 8-1 と SFT 8-2 である．) コンポーネントの内部構造を再びタイプとして定義することができる．(例えば，コンポーネント SFT 8-1 の構造はコンポーネント SFT 4-1, SFT 4-2 等から構成された SFT 4 というタイプで定義されている．)

図-9 において，フレーム SFT 8 は，SFT 8 がコンポーネント SFT 8-1, SFT 8-2 と信号線 SFT 8-3 等

```
(フレーム名 (スロット  (ファセット  (値)…(値))
                       ・・・
                       (ファセット  (値)…(値)))
             ・・・
            (スロット  (ファセット  (値)…(値))
                       ・・・
                                    ))
```

(a) フレームの構造

「ANDC2は ANDゲートの一種で，クロック信号系で使用されるもので，端子 I1, I2, Yをもつ.」

(b) ANDC2の記述

```
フレーム名  スロット  ファセット  値
(ANDC2  (A-Kind-Of      (VALUE  (AND-GATE)))
        (RESTRICTION    (VALUE  ( (USED-FOR CLOCK) )))
        (PINS           (VALUE  (I1) (I2) (Y))))
```

(c) フレームによる記述

図-7 フレームの構造

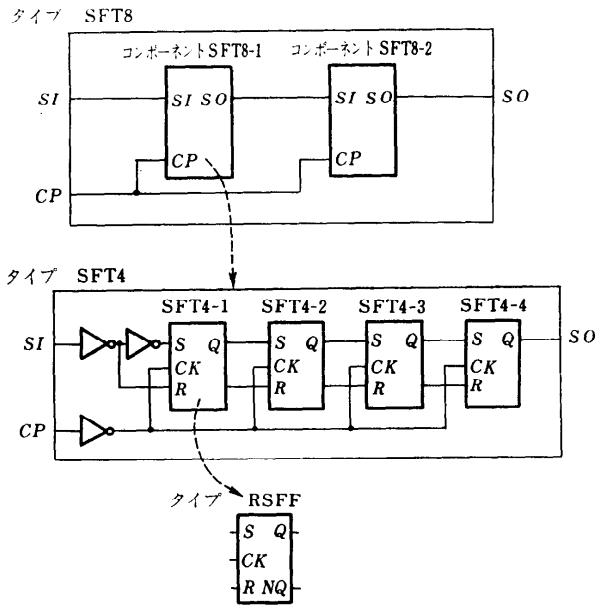


図-8 シフトレジスタの階層的な定義

```
(COMPONENT  (LOCATION     (IF-CHANGED  (MOVE-SYMBOL))
                          (IF-NEEDED   (ASK-DESIGNER)))
            (TECHNOLOGY   (DEFAULT     (TTL))))
(NET        (LINE         (IF-CHANGED  (MODIFY-LINE))))
.....
(SFT 8      (AKO          (VALUE       (TYPE)))
            (COMPONENT    (VALUE       (SFT 8-1) (SFT 8-2)))
            (NET          (VALUE       (SFT 8-3) ... )))
(SFT 8-1    (AKO          (VALUE       (COMPONENT)))
            (TYPE         (VALUE       (SFT 4)))
            (LOCATION     (VALUE       ( (100 100) )))
            (CONNECTED-NET (VALUE      (SFT 8-3))))
(SFT 8-2    (AKO          (VALUE       (COMPONENT)))
            (TYPE         (VALUE       (SFT 4))))
(SFT 8-3    (AKO          (VALUE       (NET)))
            (LINE         (VALUE       ( (140 140 200 140) )))
            (CONNECT      (VALUE       (SFT 8-1) (SFT 8-2))))
(SFT 4      (AKO          (VALUE       (TYPE)))
            (COMPONENT    (VALUE       (SFT 4-1)...))
            (NET          (VALUE       (SFT 4-8)...)))
```

図-9 フレームによる論理回路の表現

から構成されるタイプの一種 (AKO : A Kind Of) であることを表わしている．また，フレーム SFT 8-1 は，SFT 4をタイプとするコンポーネントが，図面上の座標(100 100) に位置し，フレーム SFT 8-3 が表わす信号線に接続していることを示している．フレーム SFT 8-3 は，座標 (140 140) と (200 140) を結ぶ線分で示される信号線が，コンポーネント SFT 8-1 と SFT 8-2 に接続されていることを表現している．

フレーム COMPONENT には各コンポーネントに共通な知識が与えられ，座標が変更されたら図形移動を行う (MOVE-SYMBOL) こと，座標データが存在しないのに必要となった場合には設計者に問い合わせる (ASK-DESIGNER) こと，及び，暗黙の利用テクノロジは TTL であることが記されている．同様に，信号線の線分データが変更された場合は，これを描きなおす (MODIFY-LINE) ことが，フレーム NET に与えられている．

3.2 フレームへのアクセス

フレーム形式のデータベースへアクセスする関数は LISP で容易に作成できる．この関数として，値の取り出し，追加，変更を行う以下のものがある．

(FGET フレーム名 スロット名 ファセット名)

(FPUT フレーム名 スロット名 ファセット名 値)

(FCHANGE フレーム名 スロット名 ファセット名 値)

例えば，(FGET SFT 8 COMPONENT VALUE) により，リスト (SFT 8-1 SFT 8-2) が得られる．(FGET SFT 8-1 TECHNOLOGY VALUE) の場合は，フレーム SFT 8-1 に取り出す値がないので，AKO スロットの値が指す上位フレーム COMPONENT の TECHNOLOGY スロットが検索対象になり，暗黙値 TTL が取り出される．

このように AKO スロットで示される上位のフレームから値をうけつぐ機能は inheritance と呼ばれる．この機能により，

下位のフレームに共通するデータを，各々の下位フレームに格納する必要がなく，記憶域が少なくてすむ．また，上位フレームのデータを修正すれば，修正されたデータをすべての下位フレームがうけつぐことができるので便利である．

3.3 デモン

今度は（FGET SFT 8-2 LOCATION VALUE）を実行してみる．この場合，フレーム SFT 8-2 にも，上位フレーム COMPONENT にも LOCATION スロットの値はない．この場合，関数 FGET は，さらに，LOCATION スロットの IF-NEEDED ファセットに与えられた関数を検索し，得られた関数を実行した結果を値として返す．したがって，フレーム COMPONENT にある関数 ASK-DESIGNER により，SFT 8-2 の座標位置を設計者に入力させ，これが値となる．

このように，値が存在しない場合や値の追加・削除が行われた場合に自動的に実行される関数はデモンと呼ばれる．

このデモンの機能を利用すれば，データの変更にともなって連鎖反応的に進められる処理を容易に記述できる．例えば，図-9 において，(FCHANGE SFT 8-1 LOCATION VALUE (100 200)) により，コンポーネント SFT 8-1 の座標を (100 200) に変更する．すると，LOCATION スロットに関する IF-CHANGED デモン MOVE-SYMBOL が実行される．この関数は，座標が変更されたフレームが表わすコンポーネントをディスプレイ上に描き直し，図形移動を行う．さらに，このコンポーネントに接続された信号線（SFT 8-3）の線分データを，コンポーネントの移動に応じた位置へ変更するため（FCHANGE SFT 8-3 LINE VALUE 新しい図形データ）を行う．フレーム SFT 8-3 の図形データの変更は，同様なメカニズムにより，ディスプレイ上に書かれた信号線を描き直す関数 MODIFY-LINE の実行を引き起こす．

以上のようなデモンの機能を利用すれば，容易にシミュレーションを行える．また，デモンをフレームに追加することは容易に行えるので，これを利用したシミュレータに，種々の機能追加を行える柔軟性を与えることができる．

4. 記号シミュレータ

4.1 アルゴリズム

3 章で述べたフレームとデモンの機能を利用して，記号シミュレーションは以下の処理[14]により進められる．

1) 与えられた論理回路の階層的定義を単一レベルの定義へ展開する．展開された回路定義は，論理的接続関係ならびに信号値を記憶するためのフレームで構成されている．また，信号値を元の階層的回路定義に即して表示できるように，展開された記述と元の記述との間の対応関係も生成される．

2) 設計者が与えた，信号値の初期設定やシミュレーションの停止条件等を登録し，実行に備える．（値の設定が省略されると，素子の名前を記号値として利用する．また，停止条件を与えなくても任意の時点でシミュレーションを中断することもできる．）

3) シミュレーション実行コマンドにより，次の処理が繰り返される．

- 初期設定コマンドやシミュレーション実行で生じた，素子の値の変更要求（イベント）から，現在のシミュレーション時刻に関するものが取り出される．これを基に，FCHANGE 関数を用いて，要求された素子に関するフレーム内の信号値を新しい値に書き換える．
- 現シミュレーション時刻でのクロック信号の値も同様に変更される．
- 現時刻に関するイベントがなくなり，停止条件が満たされていなければ，シミュレーション時刻を更新する．

上述の処理の流れには，素子の出力値を求める演算が含まれていない．実は，この演算はデモンにより行われる．**図-10** に，シミュレータがアクセスするフレームの例を示した．フレーム AND に存在するデモン ADD EVENT は入力端子の値の変化により起動され，入力値を SAND した値を出力端子Yにセットする要求を登録する．（図中の関数 SAND は図-1 に掲げてある．また，遅延モデルとしては，各素子に代表的な遅延時間を与え，入力変化がこの時間だけずれて表われるとする標準遅延方式を用いている．）一方，出力端子Yに関するデモンは，出力値が変化した場合，これを接続先素子の端子の値を変更する．したがって，シミュレータがフレーム AND1 のスロット $I1$ の値を 1 に変更すると，AND 1 の出力を a に変更する要求が出される．シミュレータは，この要求を検出すると AND 1 のスロットYに値 a をセットし，出力端子用デモンがこの値を接続先 AND 2 の端子 $I1$ を変更する．これは再び入力端子用デモンを起動する

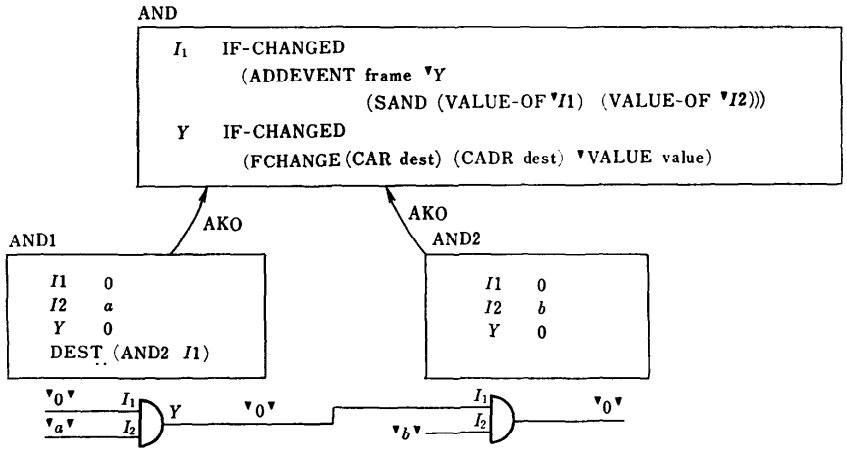


図-10 フレームとデモンによるシミュレーション

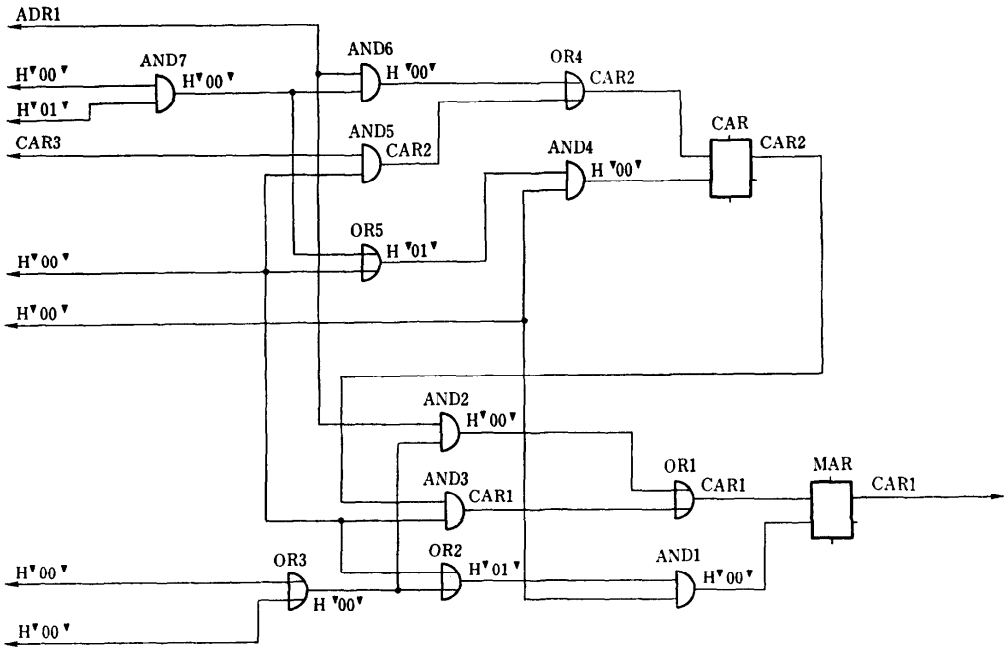


図-11 記号シミュレーション表示例

ことになる．以上のように，IF-CHANGED デモンを利用すれば，変化した信号のみをシミュレートしてゆくセレクティブ・トレース方式を容易に実現できる．

また，デバッグ等に役立つ，信号値の履歴作成も，プログラム (FPUT frame HISTORY VALUE 端子・値・時刻のリスト) をデモンとして追加するだけで実現できる．

4.2 マンマシンインタフェース

シミュレータは設計された論理回路の動作を確認するための支援ツールであるため，容易に動作を把握できるマンマシンインタフェースをシミュレータが備えていることが望まれる．

図-11 は，入力した論理回路とその状態を対応付け

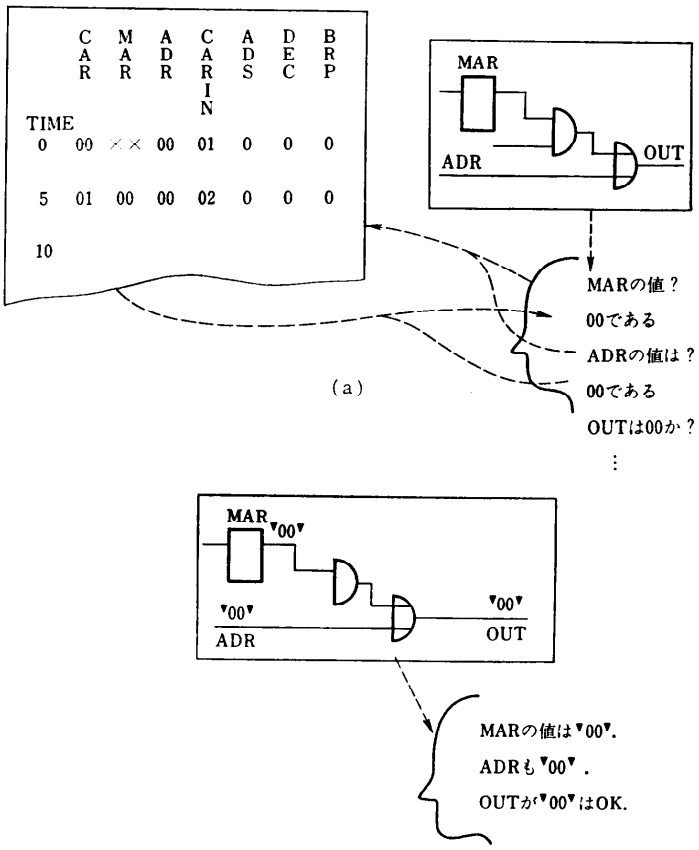


図-12 シミュレーション結果の表示

て表示した例である．普通，設計者はシミュレーション結果のリストと図面内の素子を結びつけつつ動作の確認を行う．ここで紹介した表示方式は，このような確認過程におけるシミュレーション結果と図面の対応付けをシミュレータに行わせるものである（図-12）．

図-11 に示される値は，値が変更されるごとに，書き換えられてゆく．この機能もデモンにより実現されている．また，値を表示する素子とシミュレーション時刻をもとに，タイム・チャートを出力する機能もデモンとして容易に追加できる．

4.3 監視機能

仕様や設計の過程で生じた論理回路の制約条件をフレーム・データベースに格納し，シミュレーションの際に，これをチェックできれば，より効果的に設計の確認を行える．例えば，図-13 の回路に，クロック入力 CP が立ち上がる時はデータの選択信号 A, B のいずれかは 0 でなければならない制約があるとする．この条件，$\uparrow CP$ ⊐～$(A \& B)$，をデモンとしてクロック入力端子のフレームに与えれば，クロック信号が変化するつど，信号 A, B をチェックすることができる．この種のデモンにより，条件違反のフラグが立てられると，シミュレータは処理を中断する．したがって，設計者は次章に述べるデバッグ支援機能を用いて違反の原因を探したり，値を強制的に修正しさらにシミュレーションを続行することができる．

5. 推論機能

設計者が，シミュレーション出力から，あるいは先に述べた監視機能により誤りに気づいたとき，この原因を捜す支援機能がシミュレータに備わっていれば効率的にデバッグを進められる．ここでは，この試みについて紹介する[15]．

5.1 バックトレース

これは，誤った値を得るに至った原因を表示する機能である．この機能は，素子間の接続情報，および素子の変化した値と変化した時刻からなる履歴情報を用いて，時間的に遡ることにより実現されている．また，不必要な探索を避けるため，素子固有の知識を利用しており，例えば，“AND ゲート出力が 0 であり，入力端子で値 0

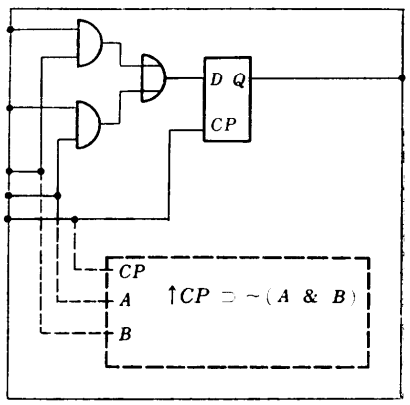
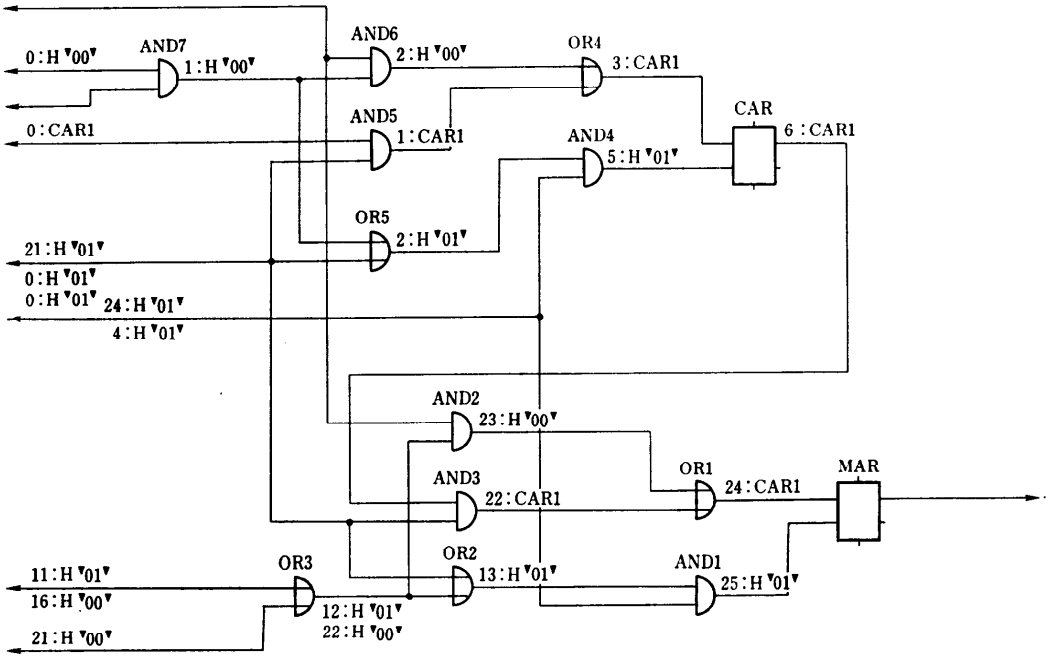


図-13 制約条件のチェック




```
TIME=26
*Stopped at 26
->REFRESH;
->WHY MAR.Q CAR1 26;
->
```

図-14 出力値を得るに至った原因

をもつものがあれば，その端子の値が0となった原因を求めよ”，“マルチプレクサの出力が0で，ストローブ信号が1なら，ストローブ信号について探索を進めよ” などがある．

バックトレースした例を**図-14** に示す．図において，“Dフリップ・フロップ MAR の出力が CAR 1 であるのは，時刻 24 で入力が CAR 1 となり，時刻 25 でクロックが立ち上がったことによる”等が示されている．この図から，例えば，フリップ・フロップ CAR の出力は時刻6から時刻 22 まで，まったく変化してないことがわかる．

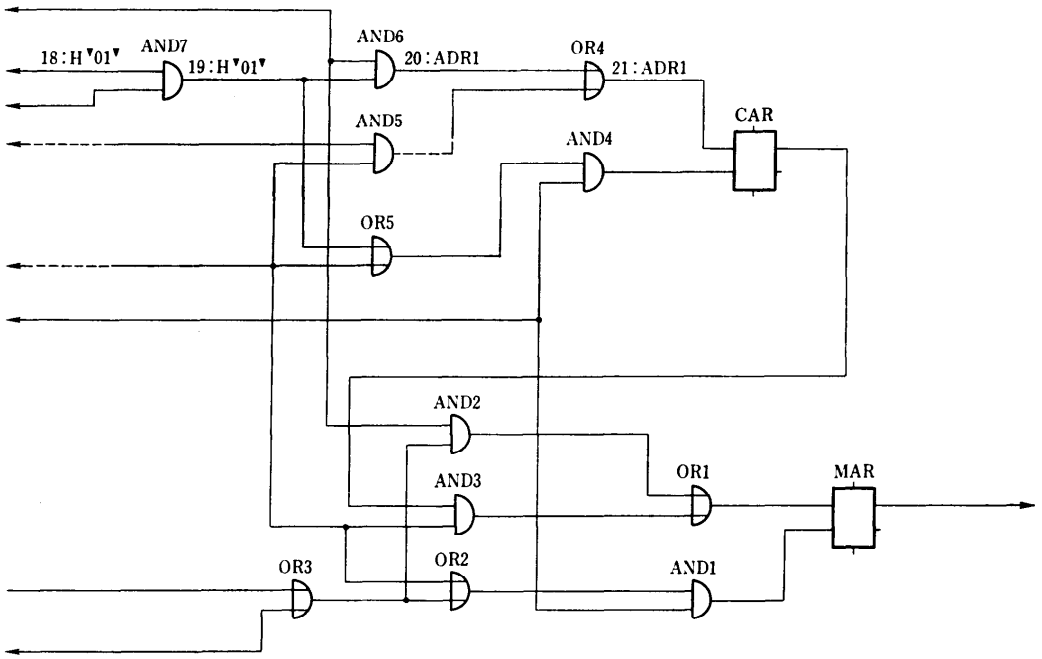
5.2 WHY-NOT 機能

これは，シミュレーションの結果が期待した値と異なる原因を推論する機能である．もし，素子の出力が正しくなければ，素子に間違いがあるか，入力が正しくないのが原因である．ここで，素子は正しいと仮定し，所望の値を得るための入力値を推論する．シミュレーションで得た値と推論された値が同じであれば，この入力に対してはさらに推論を進める必要はない．異なる入力に対して，同様に配線は正しいと仮定して，信号を送出する素子が満たすべき出力値が決まる．これを外部入力端子に至るまで繰り返せば，設計者の与えた値の誤りがあるか，推論で正しいと仮定した素子，結線に誤りがある可能性があることが分る．

例えば**図-15** において設計者はなぜ OR ゲート OR 4 の出力が時刻21で値 ADR 1 でないか質問する．まず OR ゲートが正しいとし，OR ゲートの知識から入力が ADR 1 と 0 であるべきことが推論される．シミュレーションで得た値と異なるのは ADR 1 であるので，次に AND 6 に対してなぜ出力が ADR 1 でないかが調べられ，AND 6 の入力が ADR 1 と 1 であるべきことが推定される．図-15 は，このようにして得られた，設計ミスを含む可能性のある信号経路を，推定された値とともに表示したものである．

6. むすび

著者等が研究開発している論理回路の記号シミュレータについて，そこに用いられている記号処理やマンマシンインタフェースを中心に紹介した．LISP，特にフレームの利用により，FORTRAN 等の通常の




```
TIME=26
->WHY MAR.Q CAR1 26;
->WHY-NOT OR4.Y ADR1 21;
ANOTHER CASE?Y
ANOTHER CASE?
```

図-15 OR 出力値が ADR 1 でない原因

言語に比べて容易に，設計の制約条件などの知識を組み入れ，これを活用する等の柔軟性をもつシステムを構築することができる．

これまでに，年々大規模化・複雑化するディジタル・システムの設計を支援するために，シミュレータの高速化がはかられ，シミュレーション専用ハードウェアの研究開発も行われてきている．ここで紹介した記号シミュレータは，従来の数値シミュレータより低速である．しかし，得られるデバッグ情報の量やデバッグの効率まで含めて考えたとき，より良い結果を得ることを目標としている．今後，このような視点からの研究が推進されることを望むものである．

参 考 文 献

1) Boyer, R. S., Elspas, B. and Levitt, K. N.: SELECT-A Formal System for Testing and Debugging Programs by Symbolic Execution, Proc. International Conference on Reliable Software, pp. 234-244 (1975).
2) 玉井，福永：記号実行システム，情報処理，Vol. 23, No. 1, pp. 18-28 (1982).
3) Darringer, J. A.: The Application of Program Verification Techniques to Hardware Verification, Proc. 16th DA Conf., pp. 375-381 (1979).
4) Cory, W. E. and van Cleemput, W. M.: Developments in Verification of Design Correctness, Proc. 17th DA Conf., pp. 156-164 (1980).
5) Carter, W. C., Joyner, W. H. and Brand, D.: Symbolic Simulation for Correct Machine Design, Proc. 16th DA Conf., pp. 280-286 (1979).
6) Abbott, C.: A Symbolic Simulator for Micro Program Development, IEEE Trans. on Comput., Vol. C-32, No. 8, pp. 770-774 (1983).
7) Evangelisti, C. J., Goertzel, G. and Ofek, H.: Design with LCD: Language for Computer Design, Proc. 14th DA Conf., pp. 369-376 (1977).
8) Cory, W. E.: Symbolic Simulation for Functional Verification with ADLIB and SDL, Proc. 18th DA Conf., pp. 82-89 (1981).

9) Saito, T., Uehara, T. and Kawato, N.: A CAD System for Logic Design based on Frames and Demons, Proc. 18th DA Conf., pp. 451-456 (1981).
10) Winston, P. H.: Artificial Intelligence, Addison Wesley Publishing Company, Inc. (1977).
11) Roberts, R. B. and Goldstein, I. P.: The FRL Manual, MIT Memo 409 (1977).
12) Winston, P. H. and Horn, B. K. P.: LISP, Addison Wesley Publishing Company, Inc. (1981).
13) van Cleemput, W. M.: An Hierarchical Language for Structured Discription of Digital Systems, 14th DA Conf., pp. 377-385 (June 1977).
14) 斎藤, 川戸, 上原：フレームとデモンによるCAD システム (FIDDLE), 情報処理学会電子装置設計技術研究会資料 9-1 (1981).
15) 斎藤, 川戸, 上原：推論機能をもつ記号シミュレータ, 情報処理学会第27回全国大会論文集, pp. 1439-1440 (1983).
16) Davis, R. and Shrobe, H.: Representing Structure and Behavior of Digital Hardware, IEEE Computer, Vol. 16, No. 10, pp. 75-82 (1983).

(昭和59年6月27日受付)