ＭＯＳ論理回路の並列論理シミュレーションについて

竹之上　典昭　　桐山　正昭　　古賀　義亮
（防衛大学校）

１．はしがき

ＭＯＳ回路が広くＬＳＩに使われるようになっており、論理設計にあたってＭＯＳ回路の形態をそのまま保存して論理設計を行うことが多くなってきている。

しかし、ＭＯＳ回路を用いた論理回路（以後ＭＯＳ論理回路という）の論理シミュレーションを行うときには、これまでのＡＮＤ・ＯＲなどの論理素子で構成された論理回路（以後単に論理回路という）に等価変換し、それを通常のシミュレータにかけてシミュレーションを行うという方法がよく用いられている。

この方法は現在まで使用してきたシミュレータをそのまま使用でき、またすでに蓄積されたデータを有効に使用できるなど利点が多い。しかし、この方法ではＭＯＳ論理回路を等価な論理回路に変換する必要がある。

ＭＯＳ論理回路を等価な論理回路に変換するツールは既に開発されつつあるが、実行に時間を要したり（１MIPSの計算機で約１秒／トランジスタ）、変換を行うために人間が介入しなければならなかったりする1)。例えば図１-AのＣＭＯＳ論理回路は直ちにＮＡＮＤに変換できるが、図１-Bに示すようなブリッジ接続されたＭＯＳ論理回路は直ちに論理回路に変換することは困難である。ＭＯＳ論理回路をそのままシミュレータにかけることができれば結線情報などを残した形でシミュレーションができこのような問題は解決する。

ＭＯＳ論理回路の論理シミュレーションについてはスイッチレベルのシミュレーションとしてBryantらがすでに提案し6)-8)、ＭＯＳＳＩＭⅡというシミュレータも開発されている6)。

ＭＯＳ論理回路のスイッチレベルのシミュレーションを行う場合、ＭＯＳ論理回路はネットワークモデルとして認識される6)。また、トランジスタはGATE端子をのぞいて他の２端子は双方向の結合になっている6)。以上のことからＭＯＳ論理回路のシミュレーションはネットワークによって並列処理ができる可能性がある6)。

論理回路の並列論理シミュレーションについてはＡＯＮ（Alternate operable network）による方法が提案されている2)。

本論文では、ＭＯＳ論理回路のスイッチレベルのシミュレーションをネットワーク状に結合されたコンピュータ（ＡＯＮを含む）を用いて並列に行う方法について提案し、さらにその動作の確認を行ったので、これについて報告する。

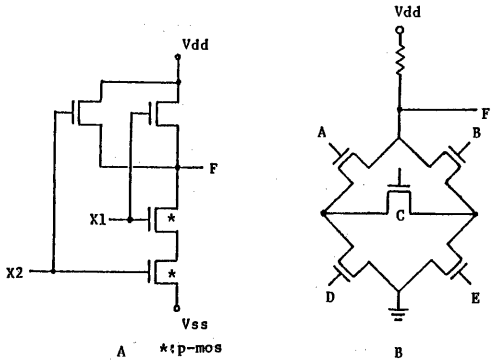

図１　ＭＯＳ論理回路例
A:CMOS NAND 回路
B:Bridge型論理回路

２．　ネットワーク型コンピュータ

本論で扱うネットワーク型コンピュータは、図２に示すように、３つの入出力ポートを持つノードコンピュータをネットワーク状に結合したものである。

ノードコンピュータはそれぞれメモリをもち、独立したコンピュータとしての機能をもっている。３つの入出力ポート

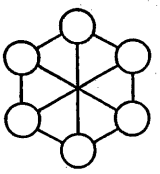

図2　ネットワーク型コンピュータ例

には専用の通信制御装置があって、ノードコンピュータ間のデータ伝送はＣＰＵが自らのメモリにデータを書き込むのと同等の速度を有するものと仮定する。この仮定は、対向の全二重回線通信方式を用いれば実現可能である。すなわち、各ノードコンピュータはすべて同等の機能をもっており、３つのポートはすべて隣のノードコンピュータに接続しているために、ノードコンピュータ間の通信をすべて全二重の対向通信にする。こうすることによって、ネットワーク型コンピュータでは、論理シミュレーションの負荷を機能によって区別することなく処理を分散して実行することができる。ネットワーク型コンピュータは、図２に示す以外にも、図３のようにノードコンピュータが多数個あるものも構成できる。

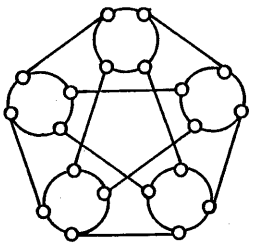

図3　ネットワーク型コンピュータ例
（ノードコンピュータ２０個）

次に、ここに示したようなネットワーク型コンピュータと、他の並列処理用プロセッサ（バス共有結合・リング結合・ゲートアレイ・木状結合）との比較を行って、ネットワーク型コンピュータの性能及び特性について考察する。

バス共有結合は、容易に並列処理を行うことができる方法であるがプロセッサ間の通信は並列に行うことができないという問題があり、プロセッサの数が増えると並列処理の効果があがらない3)。

リング結合は、特にローカルエリアネットワークによく用いられている10）がバス共有結合の場合と同様にプロセッサ間の通信に問題があり、機能ごとに仕事を分散し、プロセッサ間での通信量をできるかぎりおさえるという方法をとらざるを得ない。

ゲートアレイは、行列の積等の計算によく適合しており並列処理の効果は大きい。しかし、ゲートアレイの場合は親のプロセッサの一つの演算装置として存在するため親のプロセッサとの通信を行うために並列処理の効果がさまたげられる。

木状結合は、論理的に木構造をした問題を処理するには適しているが、物理的に並列処理できる大きさが定まっており、それ以上の大きな処理は従来の方法、つまり並列処理によらない処理がおこなわれる。また本論であつかうＭＯＳ論理回路のようにネットワーク状の構造をした問題には不利である。

ネットワーク型コンピュータは、プロセッサ間の通信はすべて対向通信のため常にプロセッサ間の通信を確保できる。プロセッサ間の通信速度を適切に設定すれば、バス結合方式のように通信量に制限を加える必要はなく仕事の負荷をノードコンピュータに効率よく分散することができる。

ネットワーク型コンピュータは、物理的にはノードコンピュータの数は決まってしまうが、分割された仕事（以後タスクという）を重複してメモリの許す限り蓄積することができる2)ため、仕事の大きさはノードコンピュータの数によって制限されることはない。

タスクの処理は、ノードコンピュータの内部ではコンカレントに処理することができるので蓄積されたタスクをすべて逐次処理する必要はない。図４は、図１-BのＭＯＳ論理回路をネットワーク型コンピュータで、並列論理シミュレーショ

ンを行った場合のタスクの処理状況をユニットタイムごとに表示したもので、高くなっている部分はタスクが処理されている様子を表している。

このようにネットワーク型コンピュータは、必要なときはいつでもプロセッサ間の通信を確保できるという特性からこれまでの並列処理プロセッサでは扱い難いネットワーク状の問題を負荷分散による並列処理によって実行できる。

3. MOS論理シミュレーションの方法

3.1 MOS論理シミュレーションにおける条件

A. トランジスタはスイッチ及び抵抗とみなし、スイッチレベルの論理シミュレートを行う。

B. ロジックの値は、真(T),偽(F),不定(U)の3値とする。

C. ネットワーク型コンピュータによる並列論理シミュレーションを行う。

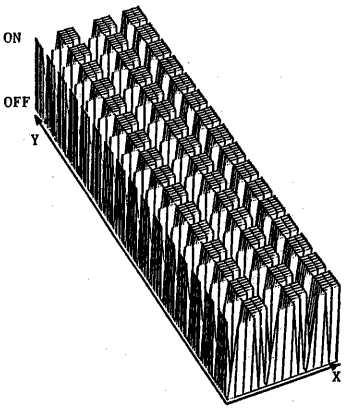

図4 Bridge型論理回路を構成するタスクのMOSPLUS上における処理状況
X:タスク番号, Y:ユニットタイム

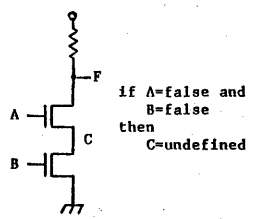

図5 NAND回路における不定(U)

B.の条件として3値を扱う理由は、図5の回路において入力端子A, Bに偽(F)が入力された場合、Cでしめされる線路はその値が不定(U)となることによる。したがって、MOS論理回路のシミュレーションにおいては最低限3値が必要である。

以上の条件のもとで論理シミュレーションを行う方法について以下に説明する。

3.2 MOS論理回路の展開アルゴリズム

与えられたMOS論理回路の論理シミュレーションをネットワーク型コンピュータを用いて行うための入力方法について述べる。

MOS論理回路をネットワーク型コンピュータに分散して入力し、効率のよいシミュレーションを行うためには、MOS論理回路を小さな部品に分割したタスクにおいて、隣接したタスクはネットワーク型コンピュータの隣接したノードコンピュータに入力すべきである。そのため木構造を用いてMOS論理回路を構成するタスクをネットワーク型コンピュータに展開する。しかし、どうしても木構造とならない部分は、補木枝を用いて展開する方法を用いる。

以下にそのアルゴリズムを示す。

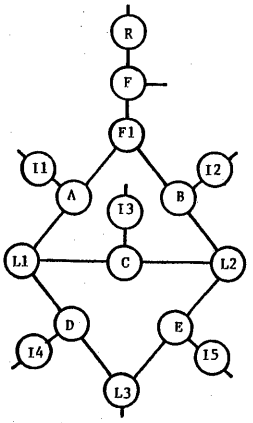

図6 タスクネットワーク
(Bridge型論理回路)

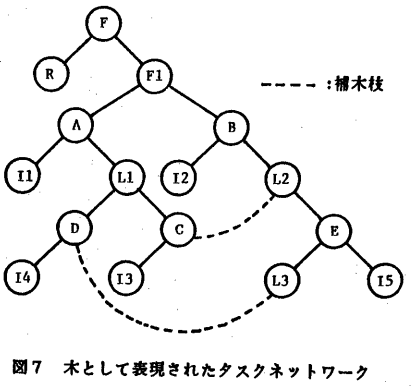

図7 木として表現されたタスクネットワーク

（展開アルゴリズム）

A. ＭＯＳ論理回路をタスク（ＭＯＳトランジスタ・結線・抵抗等の各機能）に分解し、タスクをネットワーク状に結合したもの（以後タスクネットワークという）を構成する。

B. タスクネットワーク上の１つのタスクを根とする２分木をタスクネットワーク上に構成する。

C. タスクネットワーク上で２分木とならない線路は補木枝として登録する。

D. ２分木となったタスクをネットワーク型コンピュータ上の１つのノードコンピュータから隣接するノードコンピュータへと２分木を展開する。

E. 補木枝となった線路は、ネットワーク型コンピュータ内の最短経路で接続する（この際、中継を行なうノードコンピュータには、新たにPASSというダミーのタスクを設ける）。

図１-BのＭＯＳ論理回路（以後Bridge型論理回路という）を例として展開アルゴリズムの説明を行う。

この回路は、トランジスタ５個と１つの抵抗で構成されており、展開アルゴリズムに従ってタスクネットワークをつくると図６のようになる。これを木構造に展開すると図７のように与えられる。さらにこの木を図２のネットワーク型コンピュータ上に展開すると、図８のように各タスクを分散することができる。

このアルゴリズムには、ＭＯＳ論理回路を２分木に変換して、ネットワーク型コンピュータに展開する方式を用いている。このため、ネットワーク型コンピュータの利点の１つである隣接するノードコンピュータ間の通信は常に確保されるという条件をうまく利用できる。なぜなら、図８に示すようにタスク間でデータの交換を行なわねばならないものは、すべて隣り合ったノードコンピュータとして存在するからである。

以上示した方法により、タスクネットワークの実行の負荷を有効に分散することができる。しかし、この方法を用いても、補木枝の部分を結ぶダミーのタスクの付加、及びノードコンピュータの数が十分に多くないときには、タスクを折り返して展開・蓄積するなどのための処理速度の低下はさけられない。

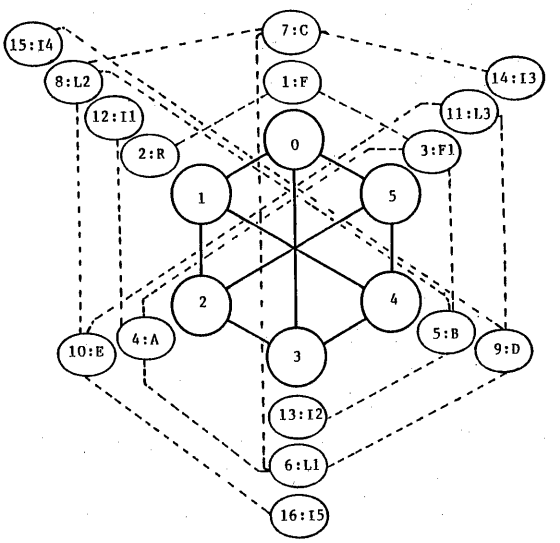

図８　ネットワーク型コンピュータ上に展開されたタスクネットワーク

３．３　タスクの構成及び機能

ネットワーク型コンピュータに蓄積されるタスクの構成は、図９のように表現できる。タスクには、３つのポートとタスクの機能を示すOPR さらにタスクの値を記憶するVALUE がある。タスクはOPRによって３つのポートのデータをどのように処理するか定まる。またポートにはそれぞれのデータを記憶するDATA，そのデータが有意であることを示すTOKEN，ポートのタスク上での意味を示すFUNC，さらにポートに接続した隣のタスクを指し示すポインタのLINKがある。

この論理シミュレーションでは以下の５種のタスクが必要である。

A. ３つのポートの接続機能：CONNECTION
線が結合しているだけであるので、表１の真理値表のようにタスクの値を決定。

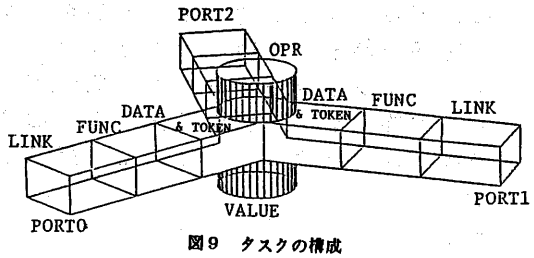

図9　タスクの構成

B.２つのポートの接続機能：PASS
　補木枝の接続に使用される他は
　CONNECTIONと同じ。
C.N-TYPEのＭＯＳトランジスタの機能：
　NMOSGATE
　GATEの値によって動作を決定。
　(GATE-VALUE)
　(T):表１の真理値表により値を決定。
　(F):タスクの値を不定(U) に決定。
　(U):ＤＡＴＡの更新をせず現状維持。
D.P-TYPEのＭＯＳトランジスタの機能：
　PMOSGATE
　NMOSGATEの真と偽が反対動作になる。
E.抵抗の機能：RESISTOR
　２端子の抵抗であるが、タスクとしては値をもち、その値は表２による。

これらのタスクの機能とポートの意味の関係を表わすと、表３のようになる。

表1　接続関係の真理値表

	F	T	U
F	F	F	F
T	F	T	T
U	F	T	U

表2　抵抗の真理値表

	F	T	U
F	F	現	F
T	現	T	T
U	F	T	U

現：現状維持

表3　タスクとポートの機能

TASK \ FUNC	JOINT	INPORT	OUTPORT	SOURCE	GATE	DRAIN	NONE
CONNECTION	O	O	O				
PASS	O	O	O				O
NMOSGATE		O	O	O	O	O	
PMOSGATE		O	O	O	O	O	
RESISTOR	O	O	O				O

３．４　シミュレーションの実行

ネットワーク型コンピュータ上に展開されたタスクネットワークは、NMOSGATEとPMOSGATEのGATE,VALUE, INPORT,OUTPORTを除いて、各ポートは論理的には、２重結合によってつながっている。つまり、論理的にデータの流れる方向を規定できないタスクネットワークにおいて、どのようにシミュレーションデータを流していくかということが問題である。また、各タスクはノードコンピュータ上に分散しているためそれを統括して管理することはできない。

これらの問題を解決するため、本論では各タスクのデータの更新を以下の方法によって行い、シミュレーションの実行を管理している。ここで各ノードコンピュータは各個に、自ら割当てられたタスクの処理を行なう自律分散方式を用いる10)。

（データ更新アルゴリズム）

A.　タスクはトークンの立っているポートのデータのみをOPR に従って処理し、タスクの値とする。
B.　タスクの値をトークンの立っていないポートおよびタスクの値と異なるデータをもつポートに対してトークン付きのデータとして送出する。
C.　送出するデータには、ポートのLINKに入っているデータを付加して送出する。

以上の動作を行うタスクは１つ以上のトークンが立っているもとする。この処理により、データの流れを制御することができる。また、A.B.C.の一連の処理を各タスクに対してノードコンピュータが行う時間をユニットタイムとする。

３．５．　停止問題

３．４．のデータ更新アルゴリズムに従ってシミュレーションを行う場合、タスクが複数個のノードコンピュータに分散して存在するため、どの時点でシミュレーションを終了すればよいかということが問題となる。言換えると一つの入力データに対し、出力データを得る時期をどのように決定するかという問題である。

この問題の解決法の一つとして、総てのタスクの値が定常状態になった時にシミュレーションを終了することが先ず考えられる。この方法であればタスクの値が次々と変化して一定にならない状態、つまり発振状態にならないタスクネットワーク（実際に発振状態となるような論理回路があれば設計上のミス）であれば、組合せ回路でも、順序回路でもシミュレーションできる。しかし、この方法では総てのタスクが定常状態になったということを各ノードコンピュータがどのようにして判断するかという問題がある。

この判断を簡単に行うためには、ネットワーク型コンピュータに図１０に示す処理の終了時期を判定するAND と同じような機能を持つハードウェアを付加しなければならない。

このような付加ハードウェアなしで、停止条件を知るためには二つの方法が考えられる。

A. 各タスクの停止条件がすべて満たされたノードコンピュータがその情報を互いに伝達し合う一斉射撃の問題のような手法を用いる方法。

B. 事前に停止条件が満たされる時期を指定しておく方法

A.の場合付加ハードウェをつけたのと同等の効果を得ることができる。しかし、この方法はある一つの入力データに対してシミュレーションが終るごとに一斉停止のためのデータを送ってシミュレーションを停止させなければならない。またその時、どのノードコンピュータからそのデータを送り始めるかということも問題となる。

B.の場合もし事前に停止時期がわかるならば、それを各ノードコンピュータに与えておいて、停止問題を簡単に解決できる。そこで本論においては、B.の方法を用いることにする。

組合せ回路のみを扱うことにすれば、シミュレーションの停止を行う時期は回路の入力端子から出力端子へ情報が伝搬する時間が経過した後とすればよい。

タスクネットワークおいて各入力端子から出力端子まで情報が伝搬する時間は、入力端子から出力端子までの最大経路長に等しい。

次に最大経路長を出力端子から求めるアルゴリズムを以下に示す。

（最大経路長検査アルゴルリズム）

タスクネットワークの連結表は与えられているものとする。

A.２端子結合のタスクを省略し、ネットワークを縮小させた連結表を作る。
（この際、距離の重み付けを行う）

B.ブレッズ・ファースト・サーチ法を応用してネットワークの深さを測るための木をつくる。

C.木をなぞり最長の深さ(Depth) を測定する。

この深さをタスクネットワークの停止時期として用いる変数をUtc(Unit time counter)とすれば

$$Utc=Depth+1$$

で与えられる。ここで１が加算されているのは、木の深さを測定する際、根を深さ０として測定するため、シミュレーションにおいては根となるタスクを処理する時間を加算する必要があるためである。

このUtc を用いることにより停止問題を解決することができる。

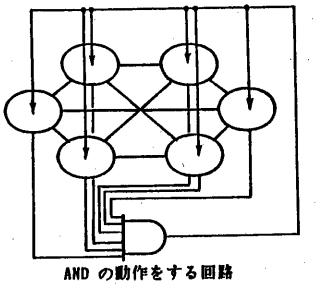

図１０ 停止問題解決のためハードウェアを付加されたネットワーク型コンピュータ

４．　シミュレーション例

本論で提案するアルゴリズムの検証のためHITAC M-200H(S) 上で図２のネットワーク型コンピュータを仮想的につくり、その上でＭＯＳ論理回路の並列論理シミュレーションを行うシミュレータＭＯＳＰＬＵＳ(MOS Parallel Logic Universal Simulator)を作成した。ＭＯＳＰＬＵＳは、図２のネットワーク型コンピュータだけではなく、ネットワーク情報を変更することによってどのようなネットワーク型コンピュータをもシミュレーションすることができる。

ＭＯＳＰＬＵＳは、PASCALによって記述されており、HITAC のTSS モードで使用できるようにしてある。また出力結果をタイムチャートにしたり、タスクの処理状況を３次元表示するためのツールもある。

以下にＭＯＳＰＬＵＳを用いたシミュレーション例をしめす。

４．１　Bridge型論理回路

図１-Bの回路のシミュレーション例について説明する。

図１-BのBridge型論理回路は、これを直ちに等価な論理回路におきかえることは困難である。しかし、このような回路もＭＯＳＰＬＵＳでは、直接シミュレーションすることができる。さらにこの回路の場合、C のトランジスタにおいては、情報は双方向のいづれにも流れることができるようになっている。

このことによりBridge型論理回路の論

```
**  BRIDGE SOURCE LIST FOR MOSPLUS  **
NODE
 0, 1
 1, 2
 2, 4
 3, 6
 4, 5
 5, 3
TASK
 16
  1,#,  JOINT, 3,OUTPORT, 1,  JOINT, 2, 7
  2,?,  JOINT, 1,   NONE,-1,  VALUE, 1, 8
  3,#,  JOINT, 5,  JOINT, 4,  JOINT, 1,11
  4,%,   GATE,12,  DRAIN, 3, SOURCE, 6,10
  5,%,   GATE,13, SOURCE, 8,  DRAIN, 3, 9
  6,#,  JOINT, 4,  JOINT, 7,  JOINT, 9,13
  7,%,   GATE,14,  DRAIN, 6, SOURCE, 8,-1
  8,#,  JOINT, 7,  JOINT, 5,  JOINT,10,12
  9,%,  DRAIN, 6,   GATE,15, SOURCE,11,-1
 10,%,  DRAIN, 8, SOURCE,11,   GATE,16,-1
 11,#,  JOINT, 9,  JOINT,10,  VALUE, 0,14
 12,&,   NONE,-1, INPORT, 1,  JOINT, 4,15
 13,&,   NONE,-1, INPORT, 2,  JOINT, 5,16
 14,&,   NONE,-1, INPORT, 3,  JOINT, 7,-1
 15,&,   NONE,-1,  JOINT, 9, INPORT, 4,-1
 16,&,  JOINT,10,   NONE,-1, INPORT, 5,-1
```

図１１　Bridge型論理回路をＭＯＳＰＬＵＳでシミュレーションするための記述

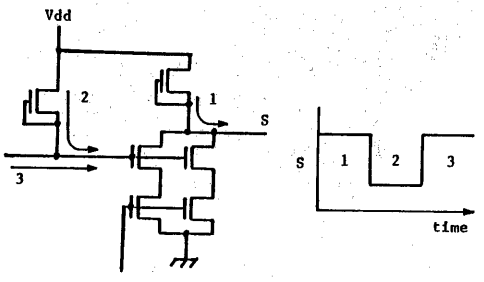

図１４　回路内の遅延の様子

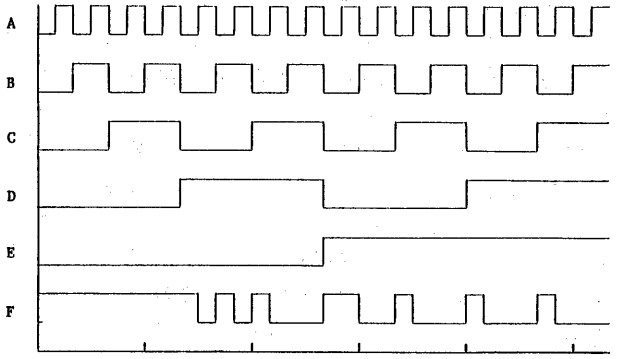

図１２　Bridge型論理回路のＭＯＳＰＬＵＳによるシミュレーション結果

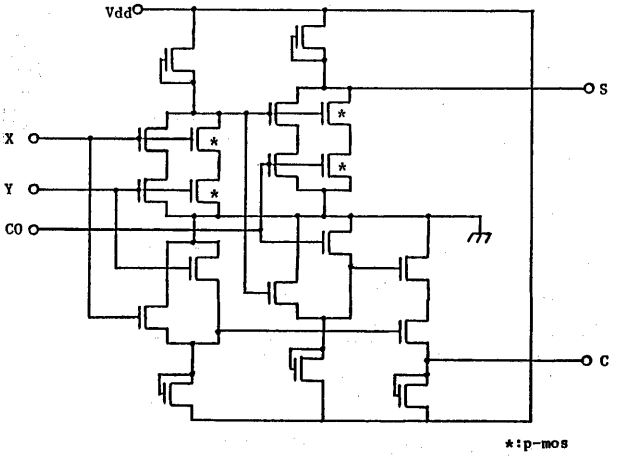

図１３　全加算器の回路例(IFADDER)

理シミュレーションを行う場合には、シミュレータ本体であるネットワーク型コンピュータの物理構造、Bridge型論理回路の論理構造、さらにその中を流れる情報フロー構造という３つの構造を制御しなければならない。しかし、これら３つの構造は３章で述べた方法によって、すべてコントロールすることができる。

実際にネットワーク型コンピュータで並列論理シミュレーションを行う場合には、ＭＯＳ論理回路を入力するための回路記述言語やコンパイラなどが将来は必要となる。ここではシミュレータが正しく動作可能であることを主目的としたため、ＭＯＳＰＬＵＳ用として仮の記述形式によって入力している。Bridge型論理回路をＭＯＳＰＬＵＳ用に記述すると図１１のようになる。これは、展開アルゴリズムに従ってネットワーク型コンピュータ上にタスクを展開した状態（図８）を記述したものである。

図１１においてNODEからTASKまでの数字の左列がノードコンピュータの番号、右列がそれに配分された先頭のタスクの番号を表している。TASK以降には展開されるタスクの総数と各タスクの情報が与えられている。タスクの情報は、左からタスク番号、OPR 、ポート０から２までのFUNC,DATA,LINKなどの情報、最後は次に接続するタスクの番号となっている。ここで－１は、未結合を示している。

ＭＯＳＰＬＵＳによって、Bridge型論理回路の入力すべてのパターンをシミュレーションした結果が図１２である。この図からBridge型論理回路論理関数を逆に生成すると、

$$F = \overline{(D+E)}+\overline{(A+B)}+B*\overline{(A+C+E)}+E*\overline{(B+C+D)}$$

で表されることがわかる。

４．２　全加算器

図１３は、NMOS,PMOS,抵抗を使用した全加算器の回路(TFADDER) である。この回路の場合、入力端子(X,Y,CO)から入力されたデータによって正しく処理された情報が到達するまえに、Ｖdd等の情報が先に到達し、それ以後の回路が一時的に誤情報を発成するという事象が現われることがある。その例を図１４にしめす。これは実際の回路中おいても生じる現象であり、ＭＯＳＰＬＵＳではその様子をもシミュレーションすることができる。図１５が出力端子S のユニットタイムごとの値を図示したものである。このことからディレイシミュレーションへの応用

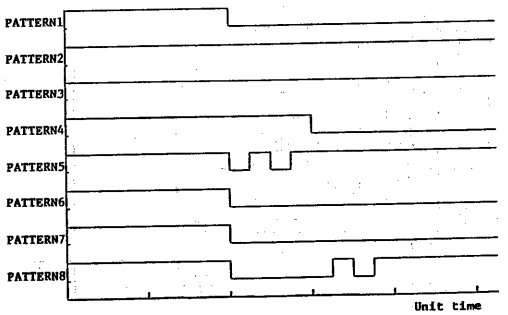

図１５　TFADDER の出力端子S のユニットタム毎の値の変化

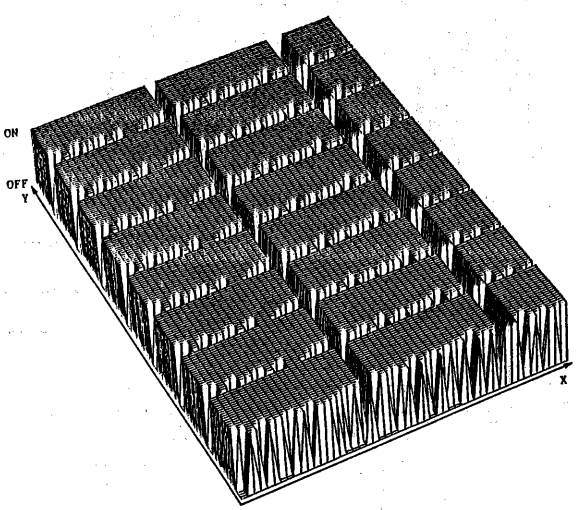

図１６　TFADDER 回路を構成するタスクのＭＯＳＰＬＵＳ上における処理状況
X:タスク番号，Y:ユニットタイム

も多少の変更で簡単に行えることがわかる。これは、ＭＯＳＰＬＵＳでは１つの入力データから結果を出力するまでの経緯を観察するできるからである。図１５からトークン付きのデータを情報として用いるデータ更新アルゴリズムによって正しく情報が処理されていることがわかる。

図１６はTFADDER のユニットタイムごとのタスク処理状況を表わしたものであり、コンカレントな処理が行われている様子がよくわかる。

５．　まとめ

本論文においては、ＭＯＳ論理回路をネットワーク型コンピュータによって並列に論理シミュレーションする一方法について新たな提案を行った。

また、ＭＯＳＰＬＵＳを作成し、本方式によってＭＯＳ論理回路のシミュレーションがおこなえることを明らかにした。

ネットワーク型コンピュータは、その特性上からＭＯＳ論理回路を負荷分散によって並列に処理することができる。そのため、従来よく行われる機能分散による並列処理に比べ、ハードウェアの構造、あるいはＭＯＳ論理回路の論理構造によって人手で行っていた変換の作業を省くことができる。

また、本方式によれば、タスクの関数機能を変更することにより、ネットワーク状に構成されたコンピュータ上において、ネットワーク状の構造をもつ仕事を処理する問題にも応用が可能である。

今後の課題としては、ネットワーク型コンピュータの実現とあいまって、ＭＯＳ論理回路を入力するための言語およびそのコンパイラの開発がある。

参考　文献

1) 羽山，渡里，京井：「 MOSマスク解析における論理検証」，情報処理，DA19-6(1983)

2) 竹之上，古賀：「コンピュータネットワークによる並列論理シミュレーションの一考察」，情報処理，Vol.25, No.3, pp.394-403(1984)

3) 高橋義造：「並列処理のためのプロセッサ結合方式」，情報処理，Vol.23, No.3, pp.201-209(1982)

4) D. E. Knuth:THE ART OF COMPUTER PROGRAMMING （基本算法／情報構造），（米田訳），Vol. 2, サイエンス社

5) Gregory F. Pfister:THE YORKTOWN SIMULATION ENGINE:INTRODUCTION, 19th Design Automation Conference, paper 7.1, pp.51-54 (1982)

6) RANDAL E.BRAYANT:A Switch-Level Model and Simulator for MOS Digital Systems, IEEE TRANSACTIONS ON COMPOMPUTERS, VOL.C-33, NO.2, pp.160-177, FEBRUARY 1984

7) M. R. Lighter, G. D. Hachtel: IMPLICATION ALGORITHMS FOR MOS SWITCH LEVEL FUNCTIONAL MACRO-MODELING IMPLICATION AND TESTING, 19th Design Automation Conference, Paper 38. 3, pp.691-698 (1982)

8) V. Ramachandran : An Improved Switch-Level Simulator for MOS Circuits, 20th DAC, Paper 21.3, pp.293-299(1983)

9) P. Kozak,A. K. Bose,A. Gupta: DESIGN AIDS FOR THE SIMULATION OF BIPOLAR GATE ARRAYS, 20th DAC, Paper 21.2, pp.286-292(1983)

10) Hirokazu Ihara and Kinji Mori: HIGHLY RELIABLE LOOP COMPUTER NETWORK SYSTEM BASED ON AUTO-NOMOUS DECENTRALIZATION CONCEPT, 12th Annal International Symposium Fault Tolerant Computing, pp.187-194(1982)