ネットワーク型コンピュータによるMOS論理シミュレーションの一方法について†

竹之上 典昭†† 古賀 義亮†††

LSIの発達とともに論理回路の集積化が進み，そのため論理シミュレーションがますます重要になっている．そのなかで論理回路の設計をMOS回路の形態で行うことが多くなってきているにもかかわらず，論理シミュレーションにあたっては，MOS論理回路をAND-OR等の論理素子に変換して従来のゲートレベルのシミュレータをそのまま用いることが多い．一方，MOS論理回路のトランジスタをスイッチとみなしたスイッチレベルのシミュレーションがBryantによって提案されている．本論文においても，MOS論理回路をスイッチで結合されたネットワークとしてとらえ，論理シミュレーションの並列処理による高速化が可能なネットワーク状に結合されたコンピュータを用いてMOS論理回路のシミュレーションを行う方法について新たな提案を行う．また，アルゴリズム検証のため，大型コンピュータ上にシミュレータMOSPLUSを作成した．その結果，ここで提案する論理シミュレーションの方法が有用であることを明らかにする．

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(以後たんに論理回路という)に等価変換し[9]，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すで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]．たとえば図1AのCMOS論理回路はただちにNANDに変換できるが，図1Bに示すようなブリッジ接続されたMOS論理回路はただ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]，MOSSIM II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[2],[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．

2. ネットワーク型コンピュータについて

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，三つの入出力ポートをも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を3ポートとしたのは任意のポート数をもつネットワークを表現することのできる最小ポート数のネットワークであることによる．したがっていかなる与えられたネットワーク型の論理回

† Switch Level Logic Simulation for MOS Circuit by Network-computer by NORIAKI TAKENOUE (Department of Ground Defense Science, National Defense Academy) and YOSHIAKI KOGA (Department of Electrical Engineering, National Defense Academy).

†† 防衛大学校陸上防衛学教室
††† 防衛大学校電気工学教室

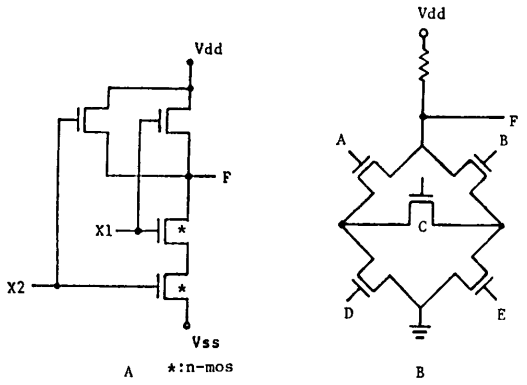

図 1 MOS 論理回路例
A: CMOS NAND回路, B: Bridge 型論理回路
Fig. 1 Two samples of MOS (metal oxide semiconductor) circuits.

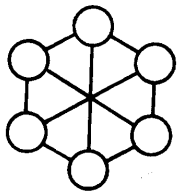

図 2 ネットワーク型コンピュータ例
Fig. 2 An example of Network-computer.

路も3ポートネットワークの中に埋め込んで表現できる.

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. 三つの入出力ポート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対向の全二重回線通信方式を用いる.

ハードウェア専用のシミュレータは, これまでのところ論理演算を実行する部分と結線の部分を別々にわけて行う方法がよく用いられている. したがって結線の部分には, 交換機 (スイッチング・ネットワーク) を必要としている[5]. ここで提案する方式は, 結線部分を含めてネットワーク型コンピュータにより MOS 論理回路のハードウェアのシミュレーションを行うものである. とくに, 結線関係をシミュレートする交換機に相当するものを用いないでよいことが特徴である. 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.

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・プロセッサアレイ・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および特性について考察する.

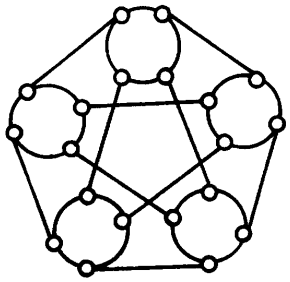

図 3 ネットワーク型コンピュータ例
(ノードコンピュータ 20 個)
Fig. 3 Another example of Network-computer.

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, とく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えない.

プロセッサアレイは, 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, プロセッサ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行われる. 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つね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

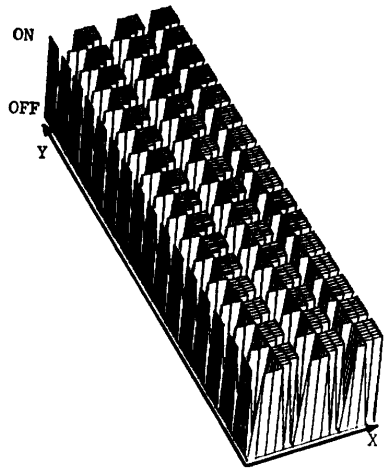

図 4 Bridge 型論理回路を構成するタスクの MOSPLUS 上における処理状況
X: タスク番号, *Y*: ユニットタイム
Fig. 4 Executing task state for Bridge type logical circuit on MOSPLUS.

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.

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および抵抗とみなし, 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.

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

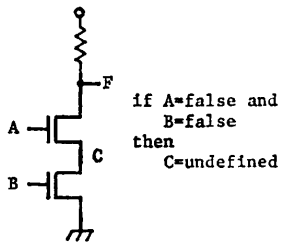

図 5 NAND 回路における不定 (U)
Fig. 5 Undefined value (U) in NAND circuit.

めされる線路はその値が不定(U)となることによる.

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, MOS 論理回路を小さな部品に分割したタスクとして, 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.

(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一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一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 個と一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[2),4)]. このため, ネットワーク型コンピ

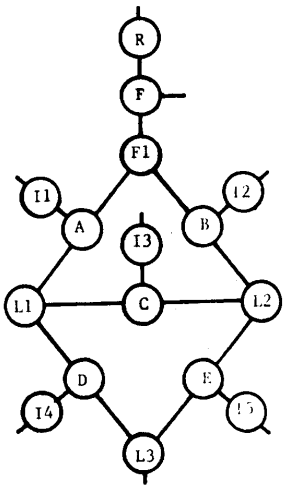

図 6 タスクネットワーク
(Bridge 型論理回路)
Fig. 6 Task-network (Bridge type logical circuits).

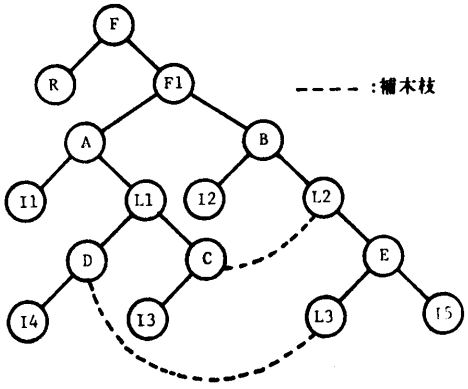

図 7 木として表現されたタスクネットワーク
Fig. 7 Task-network represented by tree structure.

ュータの利点の一つである隣接するノードコンピュータ間の通信はつね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，およ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．

3.3 タスクの構成および機能

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．

この図9のモデル図は，中心部にタスクで実行する

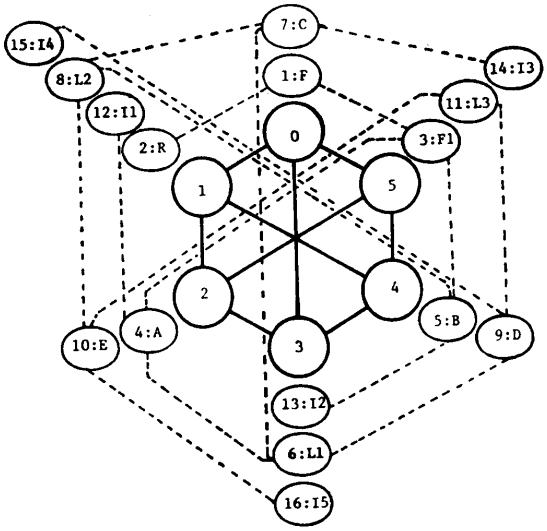

図 8 ネットワーク型コンピュータ上に展開されたタスクネットワーク
Fig. 8 Spreaded task-network in Network-computer.

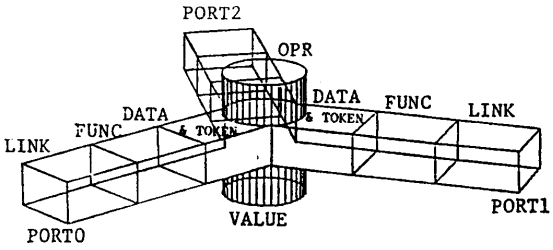

図 9 タスクの構成
Fig. 9 A task structure on a Node-computer.

仕事の内容を示す OPR とその実行結果を入れる VALUE 部がある．OPR はトランジスタ，抵抗，接続等をシミュレートするために，それぞれ対応した仕事の内容を定める．これは後に示すように5種類（表3）ある．VALUE は OPR によって実行された値を入れるためのものである．図9の OPR と VALUE のまわりの3方向の長方形のはりは，入出力ポートに関する情報を入れるためのものをモデル化して示したものである．FUNC は各ポートの状態，すなわち，接続，入力，出力等7種類の機能（表3）を表している．DATA はそのポートに存在するデータを示し，TOKEN はそのデータが有意であるかどうかを示すためのものである．LINK は，そのポートに接続されている隣のタスクを示す．これは隣のノードの中にタスクが多数存在することがあるので，特定のタスクを示すために必要である．このように表現されたタスクの内容をさらに詳しく次に説明する．

A．三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．

B．二つのポートの接続機能：PASS

補木枝の接続に使用されるほかは 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すと，**表3**のようになる．

表1 接続関係の真理値表
Table 1 Truth table for wiring operation.

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

表2 抵抗の真理値表
Table 2 Truth table for RESISTOR.

	F	T	U
F	F	現	F
T	現	T	T
U	F	T	U

現：現状維持

表3は，図9に示したようにモデル化したとき中心部の OPR と各3方向のはりの FUNC との関係を示したものである．例えば，OPR が NMOSGATE のとき，FUNC は○印のついた INPORT, OUTPORT, SOURCE, DRAIN の組合せで与えられる．MOS 回路の場合は GATE だけが入力になるとは限らず，SOURCE, DRAIN も入力として使われることがありうるからである．

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総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，自ら割り当てられたタスクの処理を行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一つ以上のトークンが立っているものとする．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

表3 タスクとポートの機能
Table 3 Relations task operations and port functions.

OPR \ FUNC	JOINT 接続	INPORT 入力	OUTPORT 出力	SOURCE	GATE	DRAIN	NONE 非接続
CONNECTION	○	○	○				
PASS	○	○	○				○
NMOSGATE		○	○	○	○	○	
PMOSGATE		○	○	○	○	○	
RESISTOR	○	○	○				○

をユニットタイムとする.

3.5 停止問題

3.4 節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．言い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，すべてのタスクの値が定常状態になったときにシミュレーションを終了することがまず考えられる．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，この方法ではすべ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も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ア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わ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．またそのとき，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.

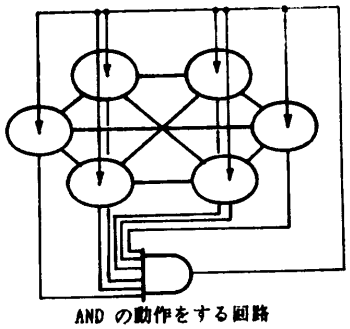

図 10 停止問題解決のためハードウェアを付加されたネットワーク型コンピュータ
Fig. 10 Additional hardware make decision when Network-computer stops.

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に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.

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-200 H(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，そのうえで 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) を開発した．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1Bの回路のシミュレーション例について説明する.

図1Bの Bridge 型論理回路は, これをただ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, 情報は双方向のいず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三つの構造を制御しなければならない. しかし, これら三つの構造は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.

```
* **  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
Fig. 11 An example of Bridge type logical circuit description for MOSPLUS.

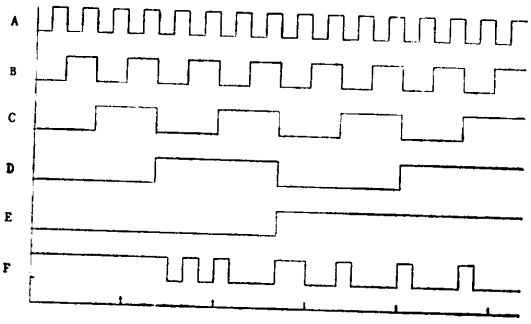

図 12 Bridge 型論理回路の MOSPLUS によるシミュレーション結果
Fig. 12 Results of Bridge type logical circuits simulated by MOSPLUS.

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, V dd 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も多少の変更で簡単に行えることがわかる. これは, MOSPLUS

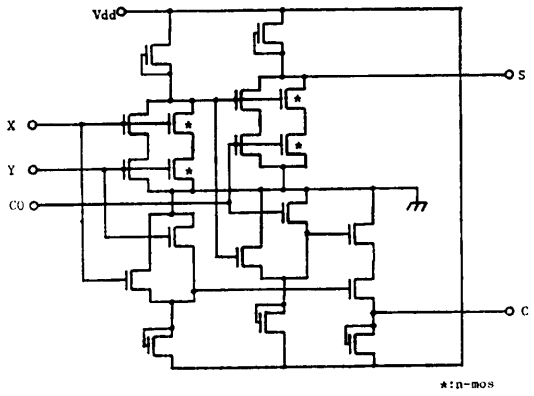

図 13 全加算器の回路例 (TFADDER)
Fig. 13 An example circuit of Full-adder (TFADDER).

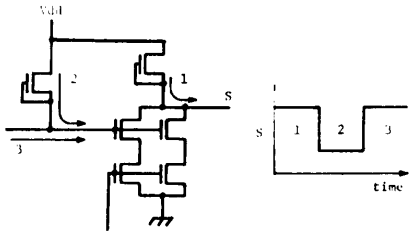

図 14 回路内の遅延の様子
Fig. 14 Circuit delay.

では一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ことができるからである．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．

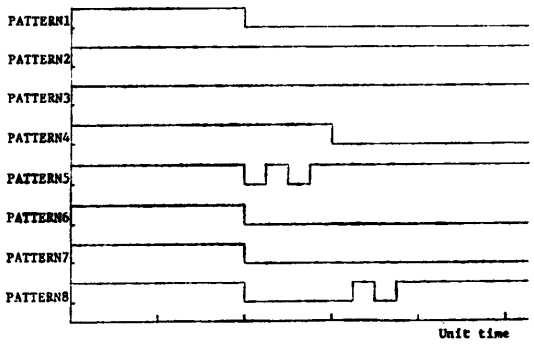

図 15 TFADDER の出力端子 S のユニットタイムごとの値の変化
Fig. 15 Output values of S-port.

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
Fig. 16 Executing task states for TFADDER on MOSPLUS.

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行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，設計自動化研究会資料，

DA 19-6 (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，情報処理学会論文誌，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) Knuth, D. E.: *The Art of Computer Programming* (米田，筧訳：基本算法／情報構造，Vol. 2，サイエンス社 (1978)).
5) Pfister, G. F.: The Yorktown Simulation Engine: Introduction, 19 th Design Automation Conference, Paper 7. 1, pp. 51-54 (1982).
6) Bryant, R. E.: A Switch-Level Model and Simulator for MOS Digital Systems, *IEEE Trans. Comput.*, Vol. C-33, No. 2, pp. 160-177 (1984).
7) Lighter, M. R. and Hachtel, G. D.: Implication Algorithms for MOS Switch Level Functional Macro Modeling Implication and Testing, 19 th Design Automation Conference, Paper 38. 3, pp. 691-698 (1982).
8) Ramachandran, V.: An Improved Switch-Level Simulator for MOS Circuits, 20 th DAC, Paper 21. 3, pp. 293-299 (1983).
9) Kozak, P., Bose, A. K. and Gupta, A.: Design Aids for the Simulation of Bipolar Gate Arrays, 20 th DAC, Paper 21. 2, pp. 286-292 (1983).
10) Ihara, H. and Mori, K.: Highly Reliable Loop Computer Network System Based on Autonomous Decentralization Concept, 12 th Annal International Symposium Fault Tolerant Computing, pp. 187-194 (1982).

(昭和59年 5 月 31 日受付)
(昭和59年 10 月 18 日採録)