設計自動化 42－3
（1988. 5. 17）

# MOSFET論理回路のスイッチレベル故障シミュレーションの一方式

岩田 忠久　　古賀 義亮
防衛大学校　電気工学教室

本報告では、従来のMOSPLUSをもとにして拡張した、MOSFET論理回路におけるスイッチレベルの故障シミュレータについて述べる。
回路を構成している素子を挿入したとき、従来は、全ての入力信号についてシミュレーションを行ったのに対し、故障情報をもとに、その故障を検出するための入力信号を生成し、シミュレーションを行うので、比較的高速に実行できる。

# A switching level fault simulation for MOSFET logic circuits

Tadahisa Iwata　　Yoshiaki Koga
Department of Electrical Engineering, National Defense Academy
1－10－20 Hashirimizu Yokosuka, 239

This paper presents a switching level fault simulator extended from MOSPLUS for MOSFET logic circuits.
When insert faults into devices of the circuits, we should simulate with all input values in usual. In this simulation, we set up test input values at the given transistor under fault insertion and then we propagate consistently the values in the rest of the circuits.

## 1　はじめに

VLSIは、多数個のトランジスタを１つのチップ上に作り上げたものであり、このトランジスタには通常MOSFETが用いられている。MOSFETは、CMOSとして使用すれば消費電力を低く抑えることができ、スイッチレベルで考えて回路を構成すればトランジスタの数を減らすことも可能である。

MOSPLUS-C は、スイッチレベルの論理シミュレータであり、MOSFET論理回路を比較的高速でシミュレートすることができる。従来のMOSPLUS は、処理系に依存する部分が多く、そのままの形では他の処理系で利用することは困難であった。

MOSPLUS-C は、PASCAL で記述されたMOSPLUS をもとに、プログラムに変更を加えることなく他の処理系でも利用することが可能な様にＣ言語で記述する。また、新たに回路を構成している素子に自動的に故障を挿入して、故障シミュレーションを行うことにする。

更に、MOSPLUS では、全ての入力信号についてシミュレートしたのに対して、MOSPLUS-C では、故障情報から故障検出入力信号を生成することにより、計算時間を大幅に短縮することができる。

本文では、今回開発しているMOSPLUS-C について記述する。

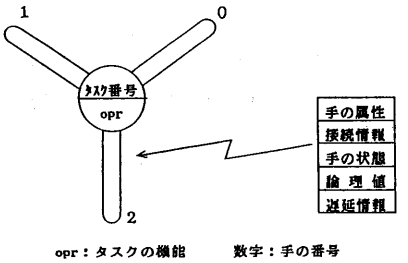
## 2　シミュレータMOSPLUS-C の概要

MOSPLUS-C は、Ｃ言語で記述されている。シミュレーションを行うには対象とする論理回路のネットワーク情報が必要である。回路のネットワーク情報は、論理回路記述言語で書かれた対象回路の記述をコンパイラPALMによりコンパイルして得られる。

### 2．1　シミュレーション・モデル

MOSPLUS-C は、トランジスタとしてMOSFETを使用している論理回路を対象とし、MOSFETをスイッチとして取扱うスイッチレベルのシミュレータである。

このシミュレータでは、論理回路の構成要素として、Ｐ、Ｎ型MOSFET、抵抗、結線、クロック・ジェネレータの５つを定義し、それらを３本の手を有するタスクというモデルで扱う。タスクと手の構造は図２．１のようになっており、タスクと手には識別のために番号が付けられている。opr はタスクの機能を表す。手の属性は素子の端子の種類を示し、接続情報は接続先のタスク番号と接続先タスクの手の番号を示す。論理値は、真（１）、偽（０）、不定（２）の３値を定義する。遅延情報は、遅延シミュレーションを行う際の素子の動作時間を与えている。




opr：タスクの機能　　数字：手の番号

図２．１　タスクとタスクの手の構造

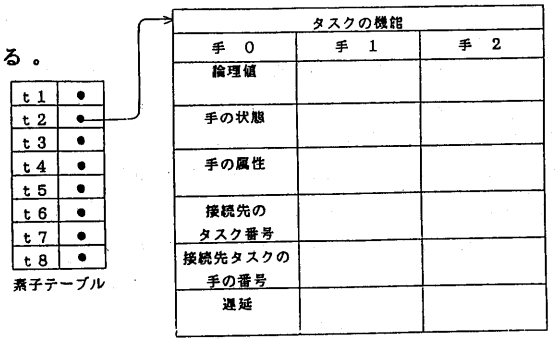



図２．２　テーブル（表）の構造

シミュレーションを実行するためネットワーク情報は、各タスクごとにテーブル（表）に格納される（図２．２参照）。素子テーブルには回路を構成する全てのタスクのテーブルを指すポインタが登録されている。

論理値の計算には、イベント・ドリブンを用いている。イベントの発生を判断するためにタスクの手の状態として次の３つ状態を定義する。

（ａ）　active　　（０）：論理値が変化し、活性化した状態。

（ｂ）　waiting　　（１）：論理値が到着するのを待っている状態。

（ｃ）　sleeping　（２）：論理値の処理が終わり、静的な状態。

（数字）：シミュレータ内で各状態に割当てられている数字。

NMOSGATE、PMOSGATEを除くタスクでは、activeな手が１つ以上ある時イベントが発生する。NMOSGATE、PMOSGATEでは、activeな手が１つ以上あり、かつ属性がgate（又はingate、datagate）である手がwaiting でない時イベントが発生する。waiting の状態はMOSFETのゲートに相当する手のみについて定義され、この状態が直接MOSFET（スイッチ）のON、OFF を制御している。

## ２．２内部構造

MOSPLUS-C は、initialize、task_setup、simulationの３つの関数から構成されており、タイム・ホイールとしてwheel_struct、回路素子の情報を格納するテーブルとしてtaskstack 、及び遅延シミュレーション用のスタックとしてzero_list、line_avail を持っている。

initializeは、タイム・ホイールを作り、遅延スタックを初期化する。タイム・ホイールwheel_structは、同一時刻における情報を並列に処理するための時間制御を行い、現時点と１単位先の時間の２単位時間を管理する。遅延シミュレーションを行う場合には、普通回路素子の最大遅延程度の大きさのタイム・ホイールが必要であるが、MOSPLUS-C は遅延スタックを２つ持つことによりタイム・ホイールの大きさを２単位時間にとどめてシミュレーションの高速化を図っている。

task_setupは、ネットワーク情報の読み込み、出力ファイルの作成などシミュレーションの動作環境を設定し、また、関数set_feild によりネットワーク情報を各タスクごとテーブルに格納する。関数makeffについては３節で述べる。ネットワーク情報には、入力信号系列のデータファイル名、シミュレーション結果出力ファイル名、各タスクの情報が記述されている。タスク情報は、タスク番号、タスクの機能、０番から２番の３本の手の情報（手の属性、接続先のタスク番号、接続先タスクの手の番号）を示す。手の情報は手の属性により分類されてテーブルの所定の位置へ格納される。

simulation では、タイム・ホイール、calculation（計算部）、network_man（通信制御部）、get_active_task（情報更新部）が主要な役割を果たしている。

calculationの機能はタスクの３本の手の情報からイベントの発生を判断し送信すべきタスクの情報（論理値、手の状態、送信先のタスク）を決定し、それをnetwork_man に渡すことである。calculation での処理を次に示す。

（ａ）NMOSGATE

| gate | drain | source | gateの状態 |
|---|---|---|---|
| 0 | 2 | 2 | waiting |
| 1 | 値の受け渡し | | sleeping |
| 2 | 動作しない | | waiting |

（ｂ）PMOSGATE

| gate | drain | source | gateの状態 |
|---|---|---|---|
| 0 | 値の受け渡し | | sleeping |
| 1 | 2 | 2 | waiting |
| 2 | 動作しない | | waiting |

（ｃ）RESISTOR

２本の手のうちどちらか一方の手の論理値が不定（２）の場合のみもう一方の手の論理値をかえす。

（ｄ）CONNECTION

最も強い論理値を全ての手に送る。

（ｅ）CLOCKGEN

１単位時間ごとに論理値０と１を交互に送る。

network_man は、遅延スタックによりデータを渡す時刻を管理しており、データをget_active_task に渡す。スタックzero_list は遅延０のタスクの情報の受け渡しに使用され、実際には時間の制御はしない。スタックline_availは、遅延１以上のタスクの情報の受け渡しに使用される。このスタックは情報を受け取ってから遅延時間が経過するまで情報をスタックにためておき、この間時間が１単位進むごとに遅延時間を１単位減らし、遅延時間が０となった時に情報を渡すという制御を行う。

get_active_task は、タスク情報を受信して論理値とイベントを更新し、この情報をタイム・ホイールに登録する。

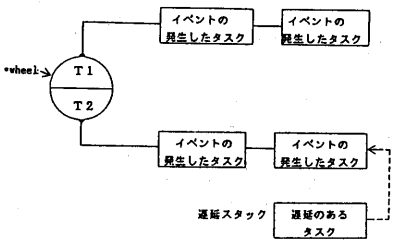



図２．３　時間管理のモデル

MOSPLUS-C の時間管理のモデル化を図２．３に示す。タスクの並列処理のための時間管理と遅延制御は独立して行われる。ポインタwheel が現時点Ｔ１を指し、Ｔ２が１単位先の時間を指しているとする。Ｔ１には現時点でイベントが発生しているタスクのイベントリストが登録されている。Ｔ１のタスクを処理して新たにイベントが発生したタスクはＴ２に登録される。Ｔ１のイベントリストが全て処理されるとwheel はＴ２を指し、今度はＴ２が現時点、Ｔ１が１単位先の時間を担当し、後は同様の処理が繰り返される。遅延のあるものは遅延の処理を終えた後イベントリストに登録される。

## ３　故障シミュレーション

MOSPLUS-C では、回路素子に自動的に故障を挿入して故障シミュレーションを行う。シミュレーションを行うには、前述の回路のネットワーク情報（以下、正常回路のネットワーク情報と呼ぶ。）と故障素子のネットワーク情報が必要となる。故障素子のネットワーク情報は、正常回路のネットワーク情報をもとに関数makeffにより作成される。makeffはtask_setupに組込まれており、st_value、st_open、line_short の３つの関数から構成されている。この３つの関数は、各々縮退故障、断線故障、短絡故障のネットワーク情報を作成する。以下、故障素子のネットワーク情報の作成の仕方、故障シミュレーションの方法について述べる。

## ３．１　故障状態のモデル化

本シミュレータでは、０、１の縮退、短絡、断線の３つの故障状態を取り扱う。２入力AND回路を例にとりこれらのタスクモデルを考える。縮退故障とは、１つの線路又は手の値が０又は１に固定してしまうものであり、これは縮退故障を起こした線路又は手を切断し、その線路又は手に接続しているタスクの手を縮退した値に固定することで表現する。短絡故障は、短絡部分に新たに２つのタスク（タスクの機能はCONNECTION）を追加して、それらを結ぶことで表現する。但し、これらのタスクは零遅延にする。

断線故障は、タスクの未接続で表現する。

### ３．１．１　縮退故障

正常な２入力AND回路モデル化を図３．１に示す。この回路において、タスク２とタスク５の間の線路に１の縮退故障を挿入すると図３．２のようなモデルとなる。故障が挿入された手の属性が”drain”から”value”へ、接続情報が”５　１”から”１　＊”へ各々変更されている。

関数st_valueでは、上述の故障挿入を１つのパターンとして全てのMOSFETに関して故障パターンを作成する。この関数では、正常回路のネットワーク情報から故障素子のネットワーク情報を作成している。そのアルゴリズムを次に示す（図３．１（ｂ）参照）。

１の縮退の場合について説明する。

（１）　正常回路のネットワーク情報のMOSFETのタスク情報を記述している行、即ちopr の値が１(NMOSGATE)又は２（PMOSGATE）である最初の行を読み込む。

（２）　最初に、番号が０の手に対し故障を挿入する。番号が０の手の情報のみ”value １ ＊”に変更して縮退故障素子のネットワーク情報ファイルSTACK_VALUE に出力し、今故障を挿入したタスクのタスク番号と手の番号をスタックhandstack に格納する。この時、故障を挿入した手に別のタスクが接続している場合には、そのタスク番号と手の番号も格納する。

handstack は故障の挿入が重複するのを防止するために設定されており、故障を挿入したタスクの番号と手の番号は全てこれに格納する。

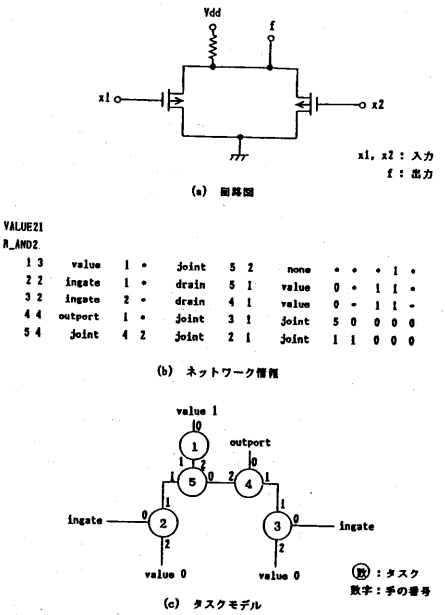



図３．１　正常な２入力AND回路のモデル

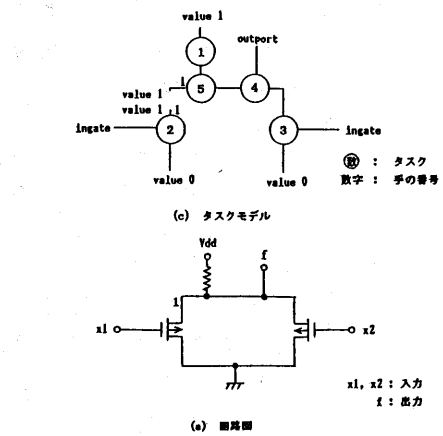



図３．２　２入力AND回路の縮退故障モデル

（３）　故障を挿入した手に別のタスクが接続している場合には、そのタスクの故障を挿入した手の情報も ”value 1 ＊” に変更する。

（４）　再び正常回路のネットワーク情報よりopr が１又は２の行を検索し故障を挿入する。このとき故障を挿入しようとした手をhandstack の内容と比較して未だ故障が挿入されていないことをチェックする。

（５）　全てのMOSFETの手に対して故障の挿入が行われるまで（１）から（４）が繰り返される。

st_valueでは、０の縮退の挿入が終了後、１の縮退の挿入が行われる。

### ３．１．２　短絡故障

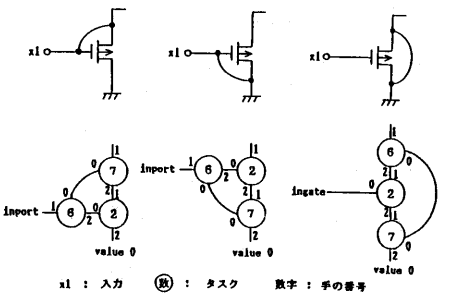
図３．１の回路において、タスク２の素子に短絡故障を挿入すると図３．３のようになり、１つのMOSFETについて３通りのモデル（comb1～comb3）が考えられる。

この場合のアルゴリズムは次の通りである。

（１）　縮退故障の場合と同様にしてopr が１又は２である最初のタスクを捜しその行を読み込む。故障タスクのタスク番号はtaskbuf に格納する。

taskbuf は故障の挿入が重複するのを防止するために設定する。

（２）　comb1 ～comb3 の故障を挿入する。故障タスクの手に別のタスクが接続する場合には、そのタスクにtask 1 又はtask 2の接続する手の情報を渡し、task 1、task 2にもそのタスクの手の接続情報を渡す。このとき、故障タスクの手で task 1 、task 2 に結合した手の属性は、gate、drain、sourceに変更される。属性がgateに変更される手に関しては、変更前の属性がgate以外であった場合、その手の情報がtask 1、task 2に渡される時点で、ingateはinport、data-gateはvalue 、outgate はoutport にそれぞれ変更される。作成された情報は短絡故障素子のネットワーク情報ファイルSHORT に出力される。




```
pattern1
 5 4   joint   4 2    joint   7 1    joint   1 1  0 0 0
 2 2    gate   6 2    drain   7 2    value   0 ＊ 1 1 ＊
 6 4   joint   7 0   inport   1 ＊   joint   2 0  0 0 0
 7 4   joint   6 0    joint   5 1    joint   2 1  0 0 0
pattern2
 2 2    gate   6 2    drain   5 1   source   7 1  1 1 ＊
 6 4   joint   7 0   inport   1 ＊   joint   2 0  0 0 0
 7 4   joint   6 0    joint   2 2    value   0 ＊ 0 0 0
pattern3
 5 4   joint   4 2    joint   6 1    joint   1 1  0 0 0
 2 2  ingate   1 ＊   drain   6 2   source   7 1  1 1 ＊
 6 4   joint   7 0    joint   5 1    joint   2 1  0 0 0
 7 4   joint   6 0    joint   2 2    value   0 ＊ 0 0 0
```

(b) ネットワーク情報

図３．３　２入力AND回路の短絡故障モデル

（３）　１つのタスクに対し３パターンの挿入が終了すると、taskbuf より大きなタスク番号のMOSFETに対して同様の挿入が行われる。

（４）　全ての素子に対して挿入が行われるまで、（２）から（３）が繰り返される。

### ３．１．３　断線故障

断線故障は、図３．４のようなモデルとして考え、図３．１（ｂ）と図３．４（ｂ）とを比較してわかるように、故障の挿入された手の情報を ” none ＊　＊” に変更する。

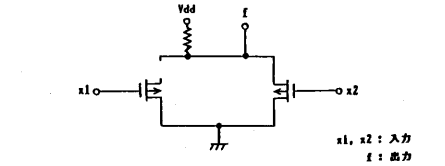
関数st_open では上述の故障挿入を１つのパターンとして、全ての素子について故障パターンを作成する。アルゴリズムは、縮退故障の場合と殆ど同じであり、

故障挿入の対象となるopr に４（CONNECTION ON）が加わる。作成された情報は断線故障素子のネットワーク情報ファイルSTACK_OPENに出力される。

## ３．２　シミュレーションの方法

故障シミュレーションの手順を図３．５に示す。シミュレーションを実行するには、正常回路のネットワーク情報ファイル、故障素子のネットワーク情報ファイルが必要である。正常回路のネットワーク情報は、task_setup中のset_field により、故障素子のネットワーク情報はsimulation中のset_field によりシミュレータ内に読み込まれる。正常回路のネットワーク情報と同様に故障素子のネットワーク情報についても情報テーブルとしてtask_boxが用意されており、故障素子の情報は、この中に格納される。故障素子のネットワーク情報ファイルには多数の故障パターンが記述されている。シミュレートする際は、故障素子のネットワーク情報に基づいて、後述する故障検出入力生成アルゴリズムにより、故障を検出するための入力信号を求めると共に、正常回路の情報を書き換え、この書き換えた情報に対して計算処理を行うのだが、故障パターンの１つについて回路の記述を全て書き換えるとすれば、故障の仮定されていない素子については同じ記述を再び書くことになり無駄である。この無駄を省くために、図３．６のような方法をとる。

計算前に、正常回路の故障が挿入されている素子の情報テーブルと故障素子の情報テーブルを差し換え、正常回路の情報テーブルは保存しておく。計算するときはtaskstack に故障素子の情報が含まれている。計算後、保存しておいた正常回路の情報テーブルをtaskstack に復帰させ、故障素子の情報テーブルは捨てる。この手順を全ての故障パターンについて計算が終るまで繰り返す。




```
pattern2
 2 2  ingate   1 .     none    . .     value   0 . 1 1 .
 5 4  joint    4 2     none    . .     joint   1 1 0 0 0
```

(b) ネットワーク情報

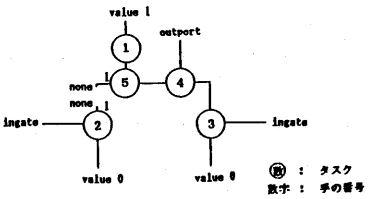



図３．４　２入力AND回路の断線故障モデル

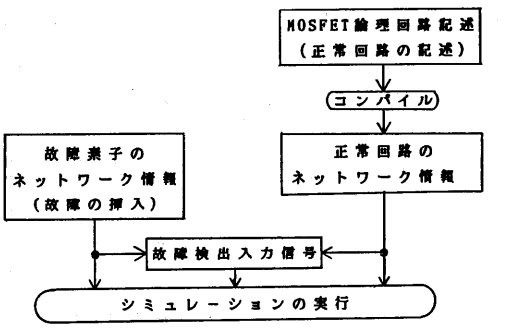



図３．５　シミュレーションの手順

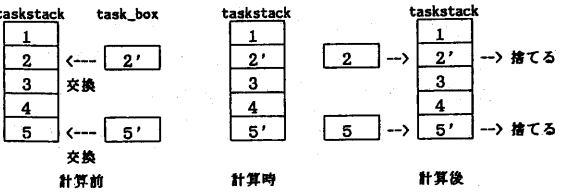



図３．６　ネットワーク情報の差し換え

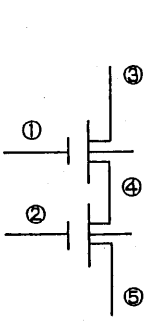
## ３．３　故障検出入力の生成アルゴリズム

従来のMOSPLUS では、全ての入力信号について、シミュレーションを行っていたが、それでは計算に無駄が生じる。そこでMOSPLUS-C では、挿入した故障のみをヒットする入力信号を生成し、計算時間を短縮する。

P、NMOSFETの直、並列接続の回路図及び入、出力は図３．７、３．８の様になる。また、図３．９、３．10は、Ｄアルゴリズムを適用したP、NMOSFETの基本キューブである。ここで、図３．７、３．８の回路に対して基本キューブを適用すると、その結果は図３．７(ｂ)、図３．７(ｂ)を十分に満足する（但し、Ｘ，Ｙ，Ｚは任意）。したがって、MOSFETにＤアルゴリズムを適用することにより入力信号を求める。

## ４　まとめ

MOSPLUS-C は、MOSFET論理回路を比較的高速で処理することができるので、LSI やVLSIの設計支援には有効である。LSI 、VLSIの普及や需要の増大に伴い、このようなシミュレータの必要性は一層高まるであろう。また、このシミュレータでは縮退、短絡、断線の３つの故障を自動的に挿入してシミュレーションすることができる。現在、論理回路の故障をトランジスタレベルで挿入し、テスト生成を行うシミュレータの開発を行っている。




(PMOS)

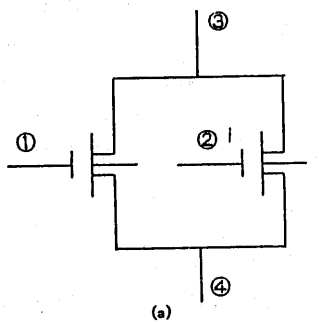
| ① | ② | ③ | ④ | ⑤ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | X | X | Y |
| 1 | 0 | X | Y | Y |
| 1 | 1 | X | Y | Z |

(NMOS)

| ① | ② | ③ | ④ | ⑤ |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | X | X | Y |
| 1 | 0 | X | X | Y |
| 0 | 0 | X | Y | Z |

(a)　(b)

図３．７　MOSFET直列接続回路




(a)

(PMOS)

| ① | ② | ③ | ④ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | X | Y |

(NMOS)

| ① | ② | ③ | ④ |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 0 | 0 | X | Y |

(b)

図３．８　MOSFET並列接続回路

(PMOS)

| D | G | S |
|---|---|---|
| 1 | 0 | 1 |
| 0 | X | 0 |
| X | 1 | 0 |

| S | G | D |
|---|---|---|
| 1 | 0 | 1 |
| 0 | X | 0 |
| X | 1 | 0 |

| S | D | G |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 1 | 0 |

図３．９　基本キューブ(PMOS)

(NMOS)

| D | G | S |
|---|---|---|
| 1 | 1 | 1 |
| 0 | X | 0 |
| X | 0 | 0 |

| S | G | D |
|---|---|---|
| 1 | 1 | 1 |
| 0 | X | 0 |
| X | 0 | 0 |

| S | D | G |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |

図３．10　基本キューブ(NMOS)


参考文献

1)　竹之上、古賀：MOS論理回路のスイッチレベルシミュレータの開発、情報処理学会、設計自動化研究資料、DA31-1 (1986)

2)　竹之上、川端、古賀：MOS論理回路のフォルトシミュレーションの一方法、情報処理学会、第34回全国大会、3f-6 (1987)

3)　樹下、藤原：ディジタル回路の故障診断（上）（工学図書）