Electrical Verification Using Rule Based Programming and Symbolic Analysis*

I. Bolsens, W. De Rammelaere, L. Claesen, H. De Man†

IMEC Lab.‡, Leuven, Belgium, Phone: +32-16-281201

Abstract

This paper describes the environment, DIALOG, providing expert analysis of MOS digital circuits to ensure electrical performance and good digital operation. The verification is founded on a formal theory that consists of a set of sufficient conditions which capture the designer's intuition of electrical correct behaviour. To bridge the gap between theory and practice, a programming environment has been created that allows for a flexible definition of good design practice by applying expert system techniques. Rule based programming and symbolic analysis form the innovative aspects of this verification environment. Rules are used to increase flexibility and to manage complexity of the algorithmic verification steps. Symbolic analysis allows to examine the context of local bugs and to report only relevant error messages. The formal statement of circuit correctness prevents the verification from being ad hoc. The goal of the system is to obtain a good compromise between efficient use of computer time and the acquisition of reliable results.

1 Introduction

1.1 Motivation

Many tools exist to aid in the design and verification of digital integrated circuits. The goal of all these tools is to aid the designer by reducing time and effort required for analysis during the design process.

Today's designers debug their design using simulation [1] [2]. The advantage of simulation is its independence of the design style. As a result simulation programs have a rather stable life. However, regardless of the level of simulation, they consume much computer time and , even worse, unexpected problems may go unnoticed. Moreover, when errors appear at the output , there is no clue about the source of the bug. As designs become more and more complex, the above makes that simulation becomes less and less reliable.

Electrical rule checkers (ERC) [3] perform a static check to ensure that circuits are not violating some structural criteria. The checks , though very useful, tend to be shallow.
All those programs share some common characteristics and suffer from the same problems :

*This research has been sponsored by the ESPRIT1058 project of the EC

†Professor at K.U.Leuven

‡Interuniversity Microelectronics Center

- Their knowledge tends to be hardcoded in the sense that only very specific patterns and cases are verified. Due to this hardcoding, detailed knowledge of the program and its data structure is required to add or change design rules.
- The ERC pinpoints all design errors without taking into account any logic constraints between signals. This results in finding a tremendous number of spurious and repeated errors, because there is absolutely no understanding of the circuit's intended behaviour.
- Interpreting the numerous and cryptic error reports is a tedious job because no context is provided with a bug and no further explanation is possible.

The use of expert system technology allows to avoid the hardcoding of design rules. Examples of such systems are the RUBICC system [5] and the CRITER system [6]. These systems perform a rule based verification of MOS circuits and are able to describe composition rules which constrain the ways that components can be interconnected. However, the increase of flexibility occurs at the cost of a decrease in efficiency, which means that only designs of a few hundreds of components can be handled. Besides, the ad hoc approach of these programs can fail in many expected but also unexpected ways, leading to unreliable fault coverage.

1.2 Key aspects

Based upon the above experiences, a prototype expert analysis system has been implemented, embedded in the DIALOG environment [4]. The basic software architecture and information flow of the system is depicted in figure 1.
The main contributions of our work are believed to be the following :

1. **Formal approach.**
   Much effort has been spent on the generalization and formalization of good electrical behaviour for digital, synchronous MOS circuits. This resulted in the formulation of **sufficient conditions** for good digital, level sensitive behaviour in terms of good **steady state** behaviour, good **memory** behaviour and good **transient** behaviour [7]. While the latter condition is handled by timing verifiers [9],[10],[11], the two other topics form the subject of the tool discussed in the remainder of this paper.
2. **Combination of rules and procedures.**
   Careful observation of many designs has shown that a rule base is too weak as a complete representation of a designer's knowledge. During the course of building the knowledge of good circuit design, it was found that particular tasks can be performed well using procedures (e.g. search for sensitive paths). Hence there is no good reason for avoiding procedures. Therefore the rule based system is extended to allow for flexible communication between algorithms and rules.
   This results in an environment that combines the application of provable correct rules with a verification strategy that exploits the information derived by the rule based analysis and as a consequence keeps computational complexity within reasonable bounds. Rules increase the flexibility of the system, allow to add additional heuristics and allow to capture the intended behaviour of the circuit.
3. **Symbolic analysis.**
   Thanks to the facility of symbolic analysis of boolean expressions, which forms an

integral part of the rule environment, only relevant error messages are generated and examining the context of a bug report is made possible.

4. **Local verification.**
   Because of efficiency, the *divide and conquer* strategy is emphasized to break down the overall problem into a sequence of loosely coupled, more easily subproblems. The partitioning is based upon a derivation of the circuit's intended signal flow and avoids the generation of large connected subnetworks.
   When a local potential circuit problem is detected, the program zooms in and allows for simulation. This introduces intelligent use of small simulators.

5. **Practical system.**
   DIALOG has been applied to several real life examples , containing a variety of design styles, such as pass-transistor logic, dynamic CMOS, static CMOS, pseudo-NMOS. Several design errors have been detected, such as the absence of refreshing logic, charge-sharing problems etc... This allows for a realistic evaluation of the tool.

The paper is organized along the following lines. In the next section, we define the basic terminology and give a survey of the verification steps. In section 3 the emphasis is on the *divide and conquer* strategy to break down the overall problem into a sequence of loosely coupled, more easily subproblems. Section 4 discusses the identification of the clocking network, memory nodes and combinational blocks. The verification of good steady state behaviour of combinational logic is discussed in section 5. In addition , heuristics to improve efficiency are presented. Results are given in section 6 and conclusions are summarized in the last section.

2 Outline of the verification strategy

2.1 Terminology.

To provide the designer with an understanding into the overall circuit behaviour and to avoid detail, abstraction must be made of the device physics level. However the circuit model must be accurate enough to model the sources of the most common error situations such as charge-sharing, ratio-errors and threshold drops.
At any level of abstraction a circuit S(N,E) is described as a set of nodes N and a set of elements E. Each element is connected via its terminals to a set of nodes. Elements, nodes and terminals can have attributes such as capacitance, resistance and relations such as adjacency, output etc...
The basic elements at the electrical abstraction level are transistors which are modeled as bidirectional, resistive switches between the drain and source terminals and controlled by their gate. The main attributes are the *width, length* and *type*.
A circuit node is an equipotential interconnection of terminals and the basic attribute is its capacitance . Transistors are represented as undirected edges between the circuit nodes connected to their drain and source terminals. This graph representation of the network is the **switch graph**. The boolean expression stating the conditions for which there is a conducting path between two nodes in the switch graph is the **switching function**.

Because the operation of synchronous circuits is based on the synchronisation of events by clock signals, the definition of clocks and gated clocks as well as memory nodes is of

prime importance.

Definition 1 *A* **clock signal** C_i *is a low impedant signal, characterized by a periodic waveform* $V_{C_i}(t)$ *with exactly two transitions during one clock period. A circuit node* n_i *that generates a clock signal is a clock i.e.* $n_i \in Clocks(S(N,E))$.

Definition 2 *A* **clock phase** ϕ *is a set of closed time intervals* T_i*, characterized by a boolean function* F_ϕ *such that*

$$(\forall t \in T_i : F_\phi(D(V_{C_j}(t)) \ldots D(V_{C_k}(t))) = 1) \wedge$$
$$(\forall t \notin T_i : F_\phi(D(V_{C_j}(t)) \ldots D(V_{C_k}(t))) = 0)$$

whereby

$V_{n_j}(t)$ *represents the waveform of a signal* n_j.

$D(v) : R \rightarrow \{0,1,X\}$ *maps voltage* v *onto one of three states.*

A clock phase is a set of time intervals (determined by transitions of clock signals) during which it is guaranteed by the designer that specific nodes in the network (memory nodes) can be interpreted as stable boolean variables that drive logic subnetworks (combinational blocks). During this clock phase the logic states of the input memory nodes cause transitions in the combinational block and the output values must be stable before they get latched in the output memory nodes.

Definition 3 *A node* n_i *generates a* ϕ_{ij} **gated clock signal** *if*

1. n_i *is an output node of a DC-connected network [2]*
2. *and* $\forall t : Driven(n_i) = 1$. *(The predicate Driven(n_i) is 1 if there exists a conducting transistor path to an external signal.)*
3. *and* $\forall t$: *IF* $F_{\phi_i}(t) \wedge F_{\phi_j}(t) \equiv 0 \Rightarrow D(V_{n_i}(t))) \equiv 0$
4. $\forall t$: *IF* $F_{\phi_i}(t) \wedge F_{\phi_j}(t) \equiv 1 \Rightarrow D(V_{n_i}(t))) \not\equiv 0$
5. *and* $V_{n_i}(t)$ *has zero or two transitions during each clock period.*

Definition 4 *A node* n_i *generates a* ϕ_{ij} **-bar gated clock signal** *if*

1. n_i *is an output node of a DC-connected network.*
2. *and* $\forall t : Driven(n_i) = 1$.
3. *and* $\forall t$: *IF* $F_{\phi_i}(t) \wedge F_{\phi_j}(t) \equiv 0 \Rightarrow D(V_{n_i}(t))) \equiv 1$
4. $\forall t$: *IF* $F_{\phi_i}(t) \wedge F_{\phi_j}(t) \equiv 1 \Rightarrow D(V_{n_i}(t))) \not\equiv 1$
5. *and* $V_{n_i}(t)$ *has zero or two transitions during each clock period.*

Gated clocks are applied to hold signals in latches for more than one clock period. During the analysis of the circuit, under the constraints of a specific clock phase $Phase_i$, all $Phase_i$ gated clock nodes are considered to be driven to 1 and all $Phase_i - bar$ gated clock nodes are considered to be driven to 0.

It is not always obvious to make the distinction between subnetworks that perform combinational logic and subnetworks that perform a latch operation. An example are C2MOS subnetworks [15] that perform a combinatorial function of the inputs as well as a storage function. Therefore instead of recognizing subnetworks that perform logic or latch operations, we identify circuit nodes that are guaranteed to hold information during a specific time interval corresponding with a specific state of clock nodes. These nodes are called **memory nodes**.

Definition 5 *A node n_i is a ϕ_i* **memory node** *if*

- *n_i is the output of a DC-connected network.*
- *and there exists a clock phase ϕ_{i+1} that does not overlap with ϕ_i such that*
  IF $F_{\phi_{i+1}} = 1 \Rightarrow$ There exists no possible conducting path to a logic node or an external node
  or
  There only exists a possible conducting path that restores the actual value $D(V_{n_i}(t))$. (i.e. Possible-memory condition*)*
- *and $\forall t \in [0 \ldots \infty]$: IF $F_{\phi_i}(t) = 1 \Rightarrow$ There exists a* possible *conducting path to a logic node or an external node, that is no refreshing path.*

Intuitively one can state that a ϕ_i memory node *can* receive new information during phase ϕ_i and *has to* hold this information at least during ϕ_{i+1}.
The latter allows to partition the network into combinational blocks that communicate via gated clocks and memory nodes.

Definition 6 *A* **combinational block (CB)** *is a minimal subnetwork such that*

- *all terminals of the CB are external inputs or outputs of the circuit, memory nodes or gated clocks.*
- *and all external inputs, outputs, gated clocks and memory nodes of the CB are terminals of the CB.*
- *and all the compound subnetworks of the CB form a connected graph.*

Combinational blocks are intended to perform combinational logic during a specific clock phase, the **Evaluation Phase**.

2.2 Verification steps

In the following sections we will highlight some of the implementation aspects of the overall verification scenario which is depicted below :

- Partition transistor circuit into unilateral subnetworks and generate directed graph representation (i.e. data dependency graph) of the network.
- Generate pull-up and pull-down switching functions of nodes in the data dependency graph.
- Detect and analyse asynchronous feedback loops.
- Identify gated clocks and dynamic gates.
- Identify and characterize good memory nodes.
- Divide network into combinational blocks.
- For every combinational block verify its steady state behaviour by checking the well-behaving of its compound subnetworks.

3 Intelligent Partitioning

Partitioning the network into smaller subnetworks and analyzing each subnetwork separately is a well known technique to reduce complexity. Several verification programs divide a circuit into **DC-connected components** (DCN) [2], [12]. In this way, the interaction between subnetworks is less ambiguous and one can take full advantage of their unilaterality. Complicated effects due to the bidirectionality of transistors are handled in the context of these smaller subnetworks.

Although many connected components are small, the use of pass transistor logic can lead to very large subcircuits of a few thousand transistors and as a consequence makes this approach less adequate. Furthermore, working on the directed interaction graph [8] resulting from the partitioning into connected components, will introduce wrong interpretations of the circuit's behaviour. For example, figure 2 depicts the directed interaction graph of a small multiplexer circuit. The vertices represent the DCN's and directed edges are drawn from a DCN to its fanout DCN's. The graph shows non-intended feedback loops between the DCN's 3, 4 and 5 which should not exist in the logical sense because of the intended signal flow direction through the pass transistors. Flagging this invalid feedback loop can mask the reporting of other relevant error messages.

To deal with the above problems, the concept of **Intended Unilateral Blocks** (IUB) is introduced. This results in further partitioning of the DC-connected components into intended unilateral blocks, making use of **signal flow** information, and will generate a more realistic capture of the designer's intentions.

3.1 Signal flow direction

The application of derived signal directions in timing verifiers and electrical rule checkers is extensively discussed in [13]. Unlike [13] that attempts to derive the **real** direction of signal flow through MOS circuits, under the assumptions that the circuit is electrically and logically correct, our goal is to determine the **intended** direction of signal flow under the assumption that the circuit is only logically correct. The assumption of electrical correctness obviously can not hold in the context of an electrical verification program.

A transistor is directed if an assignment of the opposite direction would create a design that violates the following logical principles :

- Every node in the network contributes to the functionality. This implies that all nodes (except for primary inputs and outputs) should be source and sink of information. This principle leads to a set of rules that operate locally on the circuit.
- A primary input is only a source of information.
- Every node gating a transistor should be able to switch the transistor on and off i.e. every gate node in the circuit (except for primary inputs) can be driven to 1 and 0. This principle results also in local rules.
- Every node in the circuit should have a path to a primary input, in a direction opposite to the assumed signal flow and every node should have a path to a primary output in the signal flow direction. This statement introduces rules operating on the global circuit.

Assuming the obedience of a circuit towards these principles will result in the obedience of the circuit to a set of signal flow rules which operate on a partially-directed graph. To construct this graph, we introduce some definitions :

Definition 7 *A node n_i is a* **source** *(sink) of information for another node n_j if the flow of information goes from node n_i (n_j) to n_j (n_i) via* source-drain *connections, i.e. if the logic value of node n_i may determine the value of node n_j.*

Definition 8 *A node n_i is a* **1(0)-source** *for another node n_j if n_i may activate a* drain-source *path to VDD (GND).*

Figure 3 gives an outline of the applied strategy to create the partially-directed graph :

1. Identify pull-up, pull-down and transmission networks.
2. Pass transistors in the transmission networks are reduced to *super-transistors* by merging chains of serial and parallel transistors and super-transistors as illustrated in figure 3a.
3. Pull-up, pull-down networks and remaining super-transistors are modeled as shown in fig 3b.
4. A partially-directed graph is created. The vertices are the circuit nodes that form the connections between pull-up, pull-down and super-transistor subnetworks (except VDD and GND). There is a directed edge from vertex v_i to v_j if v_i is a *source* or a *1(0)-source* of v_j. There is an undirected edge between vertices v_i and v_j if they are connected to the *drain* or *source* terminals of a super-transistor.
5. Direction finding rules (see fig 3c) are fired on this graph, until the graph is maximally directed.
6. The direction assignment to an edge in the graph is expanded to the transistors, represented by the edge.

Examples of signal flow rules, operating on the partially-directed graph are illustrated by figure 3c :

1. If all vertices v_j that have an edge-connection with v_i are *source* or *1(0)-source* of v_i, except one vertex v_x which is unknown with respect to v_i then v_i is a *source* of v_x
2. If all vertices v_j that have an edge-connection with v_i are *source*, *1(0)-source* , *sink* or *1(0)-sink* of v_i except for one vertex v_x which is unknown with respect to v_i and none of the *sink or 1(0)-sink* vertices has a path in the signal flow direction to an output terminal the vertex v_i is a source of v_x.

The knowledge base contains rules that will derive the only logically meaningful signal flow directions through a transistor network without any manual intervention or additional input from the user and without any restriction on the circuit structure.
The main characteristics of the set of signal flow rules are :

- The rules are not dependent of the design style because we only assume logical correctness.
- The ordering of the rules will not influence the final result. However, it can influence the execution efficiency.

- The knowledge base guarantees that a transistor will never be labeled in the wrong direction.
- The further verification does not require a complete setting of all transistors. However, the efficiency of the verification will improve when the results of flow analysis are available.
- Local rules are complemented with global rules, to be able to handle feedbacks as depicted in figure 4.

When, because of electrical reasons, a posteriori, a violation of the derived signal flow is detected (e.g. during ratio check), a design error is reported because electrical and logical behaviour do not coincide.

3.2 Partitioning

Relying on the principle of signal flow direction, partitioning connected components into Intended Unilateral Blocks becomes possible as shown in figure 5a.

Definition 9 *Two nodes, n_i and n_j, in the same DCN, are in a different* **Intended Unilateral Block (IUB)** *if there exists a DCN-output node n_k such that there exists a* source-drain *path (in the directed transistor graph) from n_i to n_k that does not violate the derived signal flow through the transistors and there exist no such path from n_j to n_k.*

To take full advantage of the knowledge of information flow, a directed **Data Dependency Graph** is created. As illustrated in figure 5, it reflects the intended signal flow of the network and depicts as such a more realistic capture of the designer's intentions.
The vertices of the graph are the outputs of the IUB's or external inputs and the directed edges indicate a relation *has-influence-on* between circuit nodes.

4 Clocking network and memory nodes.

4.1 Gated clocks

Because of the wide variety of clocking methodologies and the vital role of clocks for the correct functioning of synchronous circuits, special attention is paid to the flexibility to analyse the clock network and to identify and characterize gated clocks. In our analysis, we start from the following information to be given by the designer :

- The primary clock signals e.g. ϕ_1 and ϕ_2.
- The *between-clocks* constraints : e.g. If $\phi_1 = 1$ then $\phi_2 = 0$
- The boolean expressions, stating the definition of the clock phases
  e.g. $phase_1 = \phi_1.\overline{\phi_2}$ and $phase_2 = \phi_2.\overline{\phi_1}$

Given the definition of gated clock in section 2, one can reformulate the obedience of a network node n_i towards some of the requirements in terms of a tautology checking problem. Node n_i is a $phase_i$ gated clock if :

1. n_i is the output of a subnetwork

```
(Defrule Gated-Clock ()
      (*If
       ((property (or clock gated-clock) node1)
        (relation has-influence-on node2 node1)
        (attribute pull-up xfpu node2)
        (attribute pull-down xfpd node2)
        (true-tautology
         (or (and (not xfpu) xfpd) (and (not xfpd) xfpu)))
        (attach xalways-zero (or xfpd xphase1 (not xclockconstr)))
        (attach xsometimes-one (or (not xphase1) (not xclockconstr) xfpd)))
        (true-tautology xalways-zero)
        (*not ((true-tautology xsometimes-one)))
       *Then
       ((assign-attribute gated-clock xphase1 node2))))
```

Rule definition of gated clock

2. $F'_{pull-down}(n_i) \equiv \overline{F_{pull-up}(n_i)}$
3. $(\ F_{phase_i} = 0 \Rightarrow F_{pull-down}(n_i) = 1) \equiv 1 \ \Leftrightarrow \ F_{phase_i} + F_{pull-down}(n_i) \equiv 1$
4. $(\ F_{phase_i} = 1 \Rightarrow F_{pull-down}(n_i) = 1) \not\equiv 1 \ \Leftrightarrow \ \overline{F_{phase_i}} + F_{pull-down}(n_i) \not\equiv 1$

$F_{pull-up}(n_i)$ and $F_{pull-down}(n_i)$ are switching functions, stating the conditions for which there is a conducting path respectively from an external 1-source or 0-source to n_i, which does not violate the direction of signal flow. Due to the information of signal flow, spurious paths are eliminated and expression sizes are kept manageable.

In case of a two phase, non-overlapping clocking strategy, *Gated-Clock* depicts the production rule that will detect all $phase_1$ gated clocks.
In the example , $xphase_1$ and $xclockconstr$ are defined as constants :

$$xphase_1 = \phi_1.\overline{\phi_2}$$

$$xclockconstr = \overline{\phi_1} + \overline{\phi_2}$$

The constant $xclockconstr$ expresses the *between-clocks* constraint which states that ϕ_1 and ϕ_2 do not overlap in logic 1.

The inference mechanism will force the rule to propagate through the whole network.

4.2 Memory nodes

Given the definition of good memorisation behaviour in the previous section, one can formulate rules to identify memory nodes in terms of tautology checking in a similar way as for the detection of gated clocks. The example of a production rule that identifies $phase_2$ memory nodes is shown below.

The knowledge that some nodes have **fixed values** during a specific clock phase must be taken into account during the analysis of memory nodes. This is done by the xinput-constr variable in the production rule , which is constructed by deriving the information of

```
(Defrule  phase2-memory-nodes ()
    (*If
     ((relation DCN-output edcn node1)
      (attribute pull-up xfpu node1)
      (attribute pull-down xfpd node1)
      (attribute input-constraints xinput-constr edcn)
      (attach  xpossible-mem
        (or (not phase1)
            (and (not xfpu) (not xfpd))
            (and node1 (not xfpd))
            (and (not node1) (not xfpu)))
      (true-tautology (or (not xinput-constr) xpossible-mem))
      (*not (false-tautology
            (and xinput-constr phase2 (or xfpu xfpd)))))
     *Then
     ((assign-attribute  memory-node phase2 node1))))
```

Rule describing a good memory node.

precharged and *pre-discharged* nodes in the network. The importance of making use of such logic dependencies, implied by the circuit context, is illustrated by figure 6. It shows a bit-serial adder , using a single clock, two-phase clocking discipline [14].
The phase definitions are :

$$F_{phase_1} = \phi$$

$$F_{phase_2} = \overline{\phi}$$

For the SUM-output node, the *possible-mem* expression results in

$$\overline{\phi} + \overline{SUMI}$$

This expression is only a tautology in the given circuit context, which makes that node $SUMI$ is pre-discharged to 0 during $phase_1$ i.e

$$F_{phase_1} = 1 \Rightarrow SUMI = 0$$

or

$$\overline{\phi} + \overline{SUMI} \equiv 1$$

Therefore, one can state that :

$$xinput - constr = 1 \;\Rightarrow\; xpossible - mem = 1$$

whereby $xinput - constr = \overline{\phi} + \overline{SUMI}$

The latter confirms that SUM obeys the condition of possible memory node. A similar reasoning yields for the second tautology in the above rule.

5 Verification of steady state

5.1 Relevant errors

A combinational block has a good steady state behaviour during its evaluation phase, if the behaviour of its output signals can be characterized by a boolean function, whose value only depends on the present states of the input signals and is independent of their values in previous clock periods (i.e. during the evaluation phase no internal memory states are allowed).

The verification of a CB is reduced to local verification of the *well-behaveness* of its compound IUB's i.e. whether every IUB-output can be considered as a one-way boolean operator. The algorithm is depicted below.

The program reports only relevant errors. An error is relevant if it influences the functional behaviour of the circuit.

A combinational circuit that realizes the function f($x_i, ..., x_n$) is influenced by an error at node n_i if there exists an input combination a=($a_i, ..., a_n$) such that an error at n_i is activated i.e. $Error_{cond_{n_i}}$(a)=1 and there exists a sensitive path of this error to the output of the circuit i.e. $f_{n_i=1}$(a) $\neq f_{n_i=0}$(a) [18].

This implies two tasks to be performed as illustrated by the $Verify$ algorithm :

- When a local error occurs, i.e. when there is a local input combination such that $Error_{cond_{local}}$=1, then the consistency of this input combination with the environment must be verified.
- If there exists an input combination a_i for which a fault is activated at the internal node n_i under investigation i.e. $Error_{cond_{n_i}}(a_i) = 1$ then the program searches a sensitive path to a CB output i.e. $F_{CB}(a_i)|_{n_i=0} \neq F_{CB}(a_i)|_{n_i=1}$ where F_{CB} represents the behaviour of the CB.

Finding a sensitive path for circuits described in terms of logic gate networks, is conceptually straightforward, since the function realized by a CB is given by the composition of the functions computed by its logic gates. However our interest is in analysing MOS circuits, represented at switch-level.

An outline of the applied technique to examine local error consistency and to detect a sensitive path in a switch network which is not completely composed of complementary CMOS gates is described in in the above algorithm. A more detailed description can be found in [17].

The algorithm is extensively making use of symbolic simulation which is provided by the tautology checking program TC [16]. This allows to proceed much more quickly than exhaustive simulation, due to the heuristically efficient symbolic manipulation and simplification rules which are part of the program.

5.2 Heuristics to increase efficiency

A variety of techniques is applied to increase the performance. The main problem of efficiency is the expression sizes of the derived path expressions. We have little hope of improving the worst case, however for many cases of interest , there are methods to potentially reduce expression sizes.

VERIFY (IUB_i)

If i $>$ total number of unilateral blocks in CB

DO return CB-output expressions

ELSE DO

$\forall k_o \in$ outputs(IUB_i)

DO

Let $Error_{cond_{local}} = \overline{\overline{F_{phase}} + \overline{F_{pu_{local}}}.F_{pd_{local}} + F_{pu_{local}}.\overline{F_{pd_{local}}}}$

If $\overline{Error_{cond_{local}}} \equiv 1$

DO

$k_o \Leftarrow F_{pu_{global}}$

ELSE DO

push k_o on list $error - nodes$

If $error - nodes = \emptyset$

DO return verify(IUB_{i+1})

ELSE DO

Let $Error_{cond_{global}} = \overline{\overline{F_{phase}} + \overline{F_{pu_{global}}}.F_{pd_{global}} + F_{pu_{global}}.\overline{F_{pd_{global}}}}$

If $\overline{Error_{cond_{global}}} \equiv 1$

DO

$k_o \Leftarrow F_{pu_{global}}$

remove k_o from list $error - nodes$

If $error - nodes = \emptyset$

DO return verify$(IUB_{i+1}))$

ELSE DO

$\forall$ input-combinations A_i of $error - nodes$

DO

Let $F_{CB}(A_i)=$ verify(IUB_{i+1})

$\forall$ input-combinations $A_{i,j}$ of error-nodes

DO

If $\overline{Error_{cond_{global}}} + F_{CB}(A_j) \not\equiv \overline{Error_{cond_{global}}} + F_{CB}(A_i)$

DO

There is a sensitive path to output

$\forall$ local input combinations for which $E_{cond_{local}} = 1$

DO investigate IUB_i

Else No sensitive path

return behaviour of fanout network of IUB_i

i.e. $(and(A_i \Rightarrow F_{CB}(A_i))...(A_n \Rightarrow F_{CB}(A_n))$

Algorithm to identify local errors, verify their consistency and find a sensitive path to the CB-outputs in a switch network.

Perhaps the most obvious one is exploiting knowledge of reconvergent paths to avoid global expansion of the resulting symbolic expressions.

Definition 10 *Two paths in the data dependency graph are* **reconvergent** *if they start at a common node (i.e. the* **fanout** *node) and terminate at another common node (i.e. the* **reconvergent** *node).*

Several rules are applied that reduce the need for global expansion of logic expressions :

- If the data dependency graph does not contain any reconvergent paths , then a local error condition is always consistent with the environment and a local fault will always propagate to the output, because there are no logic interdependencies between the inputs of a IUB.
- The consistency of a local error with the environment should only be verified if the IUB-output is a reconvergent node because, for any non-reconvergent node, there is no logic relation between its input nodes, and as a consequence any local input pattern is consistent with the environment.
- During verification of the consistency of local error condition at a reconvergent node, the backward propagation of boolean expressions can stop at the input nodes of the minimal subgraph, feeding the node, whose inputs are logically independent. The latter subgraph is defined in [19] as a **supergate**.
- The existence of a sensitive path of a local error to an output has to be investigated only for those nodes that are on a path between a fanout node and his reconvergent node.

5.3 Error reports

It is important to state that errors which occur due to transient effects, delays in signal transitions will not be detected by the above verification procedure.

Next we give an overview of the possible error configurations that are detected and reported during the execution of the above algorithm.
A node n_i is part of the list *high-impedant* if there exists an input combination such that $F_{pull-up}(n_i) = 0$ and $F_{pull-down}(n_i) = 0$.
A node n_i is part of the list *short-circuit* if there exists an input combination such that $F_{pull-up}(n_i) = 1$ and $F_{pull-down}(n_i) = 1$.
For any local input combination that causes a relevant error, the following analysis of the IUB-outputs takes place :

IF node $\in$ list high-impedant **DO**
 IF number of logic nodes in conducting part > 1 **DO**
  $ERROR - MESSAGE$: More than one logic node
  writes info to a high impedant output.
 ELSE IF number of logic nodes in conducting part $= 0$
 DO
  $ERROR - MESSAGE$: No information is given to node,
  internal memory state detected.

ELSE IF capacitance(logic-node) >
3 x $\sum$ capacitance(conducting-part - logicnode)
DO
IF logic-node $\neq$ node **DO**
$WARNING - MESSAGE$: info passed
via charge-sharing from logic-node to output
ELSE DO
$ERROR - MESSAGE$: invalid value given to output
via charge-sharing.

The nodes that are flagged because there exists a short circuit between VDD and GND are further analysed :

IF nr-logicnodes in conducting part >=1 **DO**
Simulate conducting part with given input pattern
IF D(V(node)) = X **DO**
$ERROR - MESSAGE$: bad W/L ratios.
IF D(V(node)) = 1 **DO**
$WARNING - MESSAGE$: possible pseudo nmos
or restoring logic.
IF D(V(node)) = 0 **DO**
$WARNING - MESSAGE$: possible restoring logic.
ELSE DO
$ERROR - MESSAGE$: probably pmos transistor in
nmos path or vice versa.

Results

The ideas presented in this paper are currently being tested on a variety of circuits to demonstrate their soundness and feasibility. Several examples of 100...500 transistors [20] designed in a static or dynamic CMOS design style could be verified within a few minutes on a Symbolics 3670 as illustrated in table 1. The largest example up to now is a 24 bit comparator execution unit of 3900 transistors [21]. The design style is 3u Cmos, the circuit consists of a mixture of pass transistor logic and static gates. It includes pass transistor XOR's , pseudo-nmos decoders, multiplexers and 'pass-gate' register files with static refresh. A two-phase non-overlapping clocking strategy is used. The complete verification takes about 400 min elapsed time (Symbolics 3670 machine).
These benchmarks resulted in the detection of several errors which were not reported during other verification steps : wrong connection of clocks (in the ALU), wrong connection of VDD and GND in the shifter which led to a bad functionality (for some input patterns), detected by the tautology checker [16].

These results prove the feasibility of the above approach, however it is not a good measure for the final system performance. A variety of drastic improvements can be made in its performance by adding known heuristics. Therefore we firmly believe that the presented approach is practical.

circuit	# of trans.	time
4-bit ALU	292	260 sec
4-bit Divider	220	90 sec
8-bit Logarithmic Shifter (up-down)	240	600 sec
8-bit Comparator with scan-path	1560	36 min
8-bit ALU Execution Unit	2414	90 min
24-bit Comparator	3880	400 min

Table 1: Run time results

Conclusions

In this paper an outline has been given of a new approach to electrical verification. An environment has been discussed that examines well defined aspects of correct electrical behaviour i.e good steady state and good memorisation behaviour.
The well-considered use of rule based programming is a key aspect of the proposed strategy. It allows to get insight into the behaviour of a flattened circuit description and to exploit this knowledge to manage complexity. Besides, the system does not place any restrictions on the network structure. The combination of rules and algorithms, based upon a set of sufficient conditions for good digital behaviour, guarantees a flexible, reliable and practical verification tool.
To cope with the problem of large subnetworks the concept of Intended Unilateral Blocks is introduced. To report only relevant errors, the use of symbolic analysis in switch networks is discussed.

Further work will be directed towards increasing efficiency, generating the digital behaviour of an approved circuit and integrate this with functional verification. All basic capabilities to compile switch level circuits are available in the program, as a consequence, a compiled code simulator is one of the obvious extensions. The creation of a rule base for high level timing verification will also be the subject of future research.

References

[1] A.Vladimirescu, K. Zhang, R.Newton, D.Pederson and A.Sangiovanni-Vincentelli, SPICE Version 2G User's Guide, University of California, Berkeley, August, 1981.

[2] R.Bryant : "A Switch Level Model and Simulation for MOS digital Systems", IEEE Transactions on Computers, no2, pp 160-177, 1984.

[3] ERC-user's guide, Silvar-Lisco, Santa Clara, California, August 1986.

[4] H.J De Man, I. Bolsens, E. vanden Meersch and J. van Cleynebreugel : "Dialog : an Expert Debugging System for MOS VLSI Design". IEEE Transactions on CAD of Integrated Circuits and Systems, vol CAD-4, no 3, pp 303, July 1985.

[5] C. Lob, "RUBICC: A Rule-Based Expert System for VLSI Integrated Circuit Critique",

Memorandum No. UCB/ERL M84/80, University of California, Berkeley, September 1984.

[6] E. van Kelly, "The CRITTER system" Proc. 21th DAC, 1984

[7] I.Bolsens, W. De Rammelaere, C. Van Overloop, L. Claesen, H. De Man "A Formal Approach Towards Electrical Verification Of Synchronous MOS circuits" , Proceedings ISCAS-88, pp 2113, Helsinki, June 1988.

[8] A. Ruehli, A. Sangiovanni-Vincentelli, G. Rabbat , "Time Analysis of Large Scale Circuits Containing One-way Macromodels", Proc. ISCC, pp 766-770, Houston, 1980.

[9] J.K. Oosterhout, "A Switch-level Timing Verifier for Digital MOSVLSI", IEEE Trans. on Computer-Aided Design, CAD-4, no 3,pp336-349, July 1985.

[10] N.Jouppi : "Timing Analysis for nMOS VLSI", Proceedings of the 20th Design Automation Conference, July 1985.

[11] J.Benkoski, E. Vanden Meersch, H. De Man :" Efficient Algorithms for Solving the False Path Problem in Timing Verification" Digest of Technical Papers ICCAD-87, pp 24-27, Santa-Clara, California, November 1987.

[12] D. Dumlugol, H.J. De Man, P. Stevens, G. Schrooten : "Local Relaxation Algorithms for Event-Driven Simulation of MOS networks Including Assignable Delay Modelling", IEEE Transactions on CAD of Integrated Circuits and Systems, vol CAD-2, no 3, July 1983.

[13] N.P. Jouppi : "Derivation of Signal Flow Direction in MOS VLSI" IEEE Transactions on CAD, Vol CAD-6, No 3, May 1987.

[14] I. Karlsson : " True Single Phase Clock Dynamic CMOS Circuit Technique" , Proceedings ISCAS-88, pp 475, Helsinki, June 1988.

[15] N. Weste, K. Eshraghian : " Principles of CMOS VLSI Design " Addison-Wesley Publishing, 1984.

[16] P. Lammens : "The Function TC, a Tutorial", Esprit1058, Technical Report, Period 2, Leuven, December 1986.

[17] I.Bolsens, W.De Rammelaere,H.De Man : "Expert Analysis of Synchronous Digital MOS Circuits Using Rule Based Programming and Symbolic Analysis", accepted for publication in the Proc. of ESPRIT technical week, North Holland, Brussels, november 1988.

[18] M. Breuer, A. Friedman : "Diagnosis and Reliable Design of Digital Systems", Computer Science Press, Inc., California, 1977.

[19] G. Seth, L.Pan and V.Agrawal :"PREDICT-Probabilistic Estimation of Digital Circuit Testability", Proc. FTCS-15, pp. 220-225, Ann-Arbor, Mich., June 1985.

[20] E. Blokken : "Benchmarks for Verification", Esprit1058, Technical Report Period 5, Leuven, June 1988.

[21] F.Catthoor, H. De Man: "Customized Architectural Methodologies for High-speed Image and Video Processing" , Proc. Int. Conf on Acoustics, Speech and Signal Processing, pp 1985-1988, New York, April 1988.

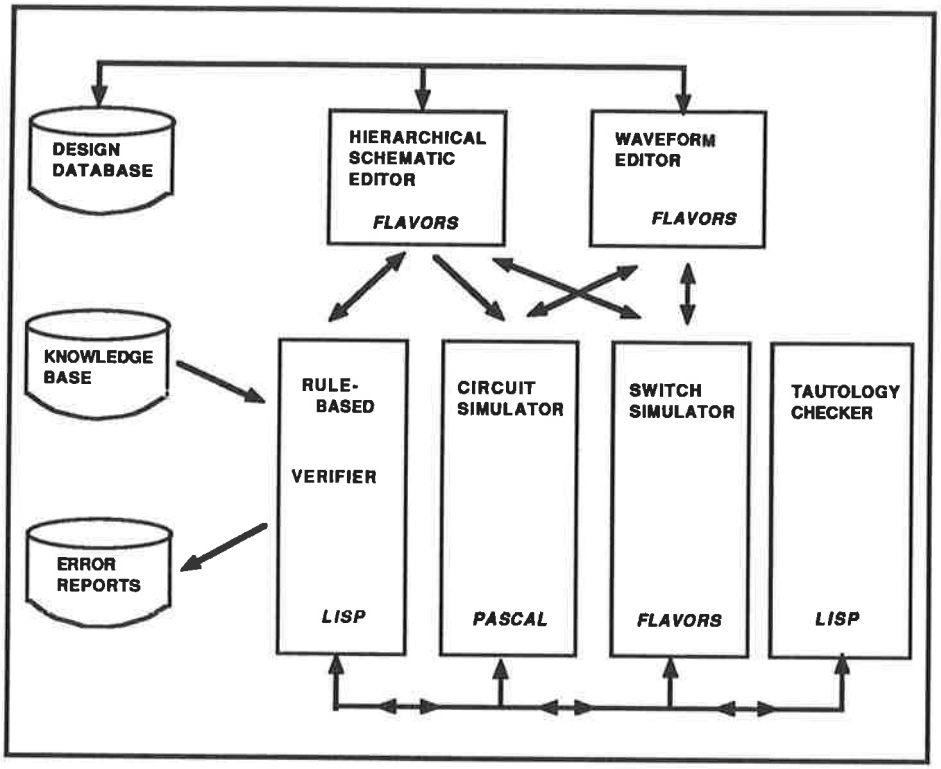


Figure 1 : DIALOG system architecture and information flow

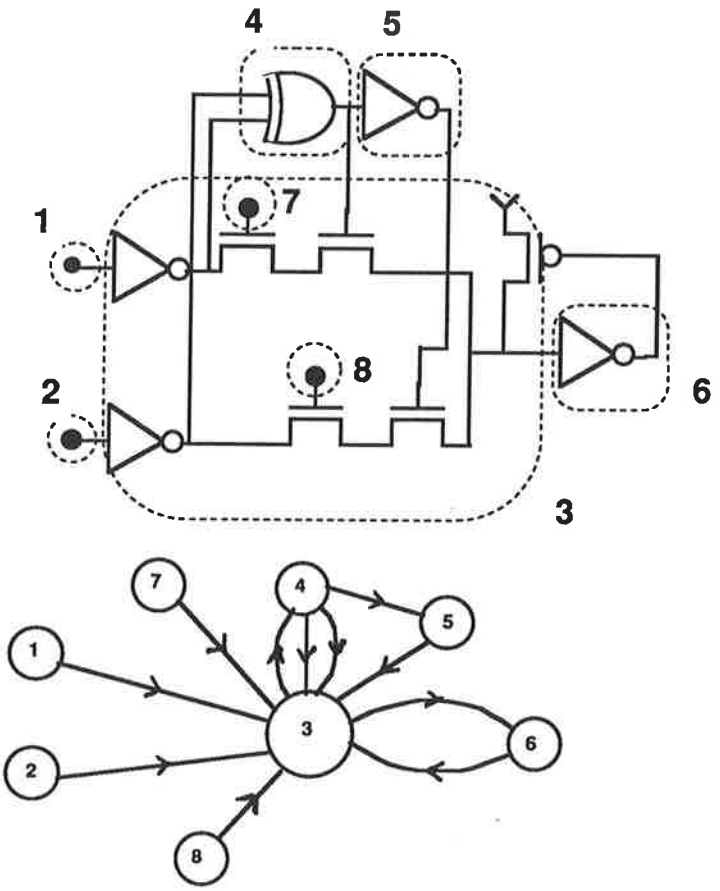


Figure 2 : Data Interaction Graph

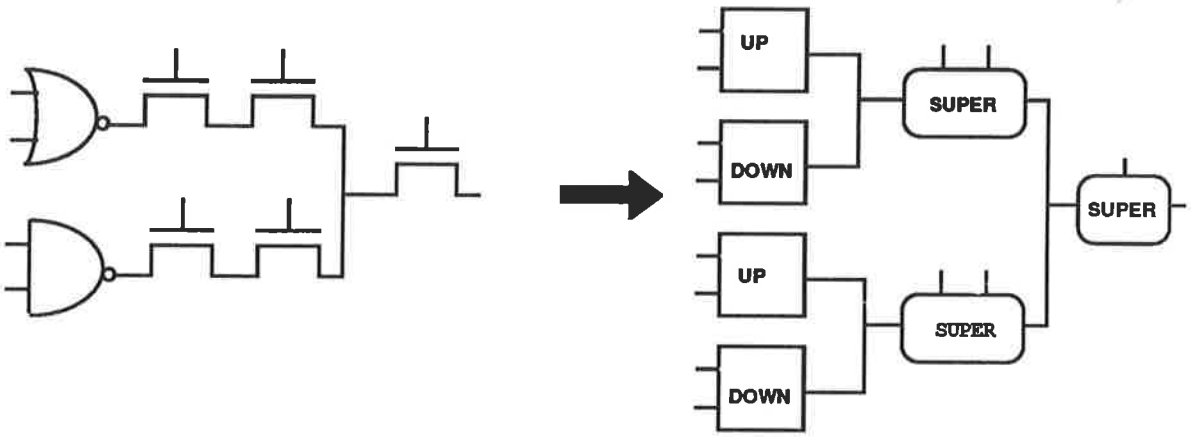


Figure 3a. Reduce transistor network

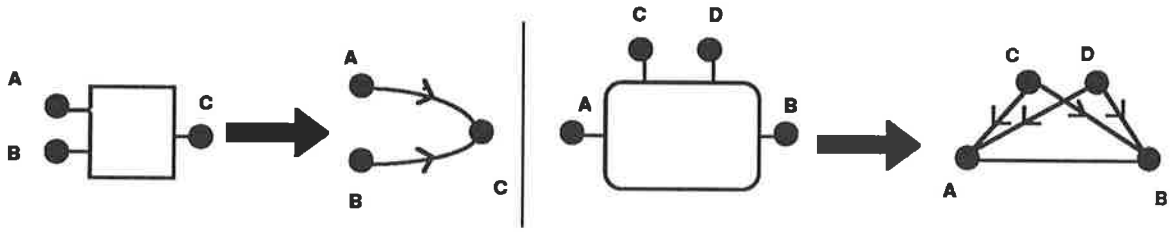


Figure 3b. Model into semi-directed graph

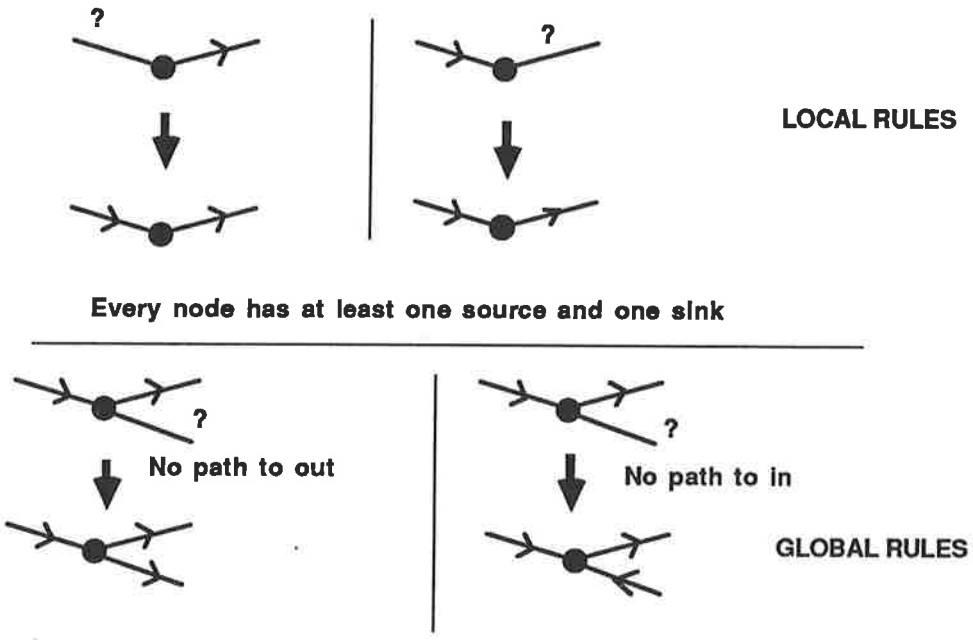


Figure 3c. Derive signal flow direction

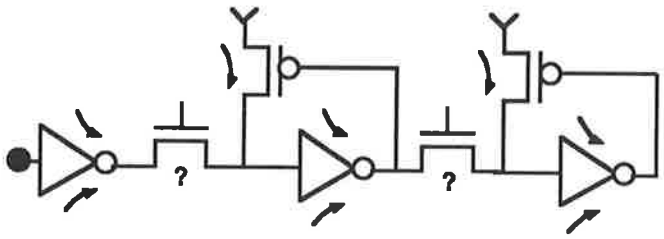


Figure 4 : Global Rules

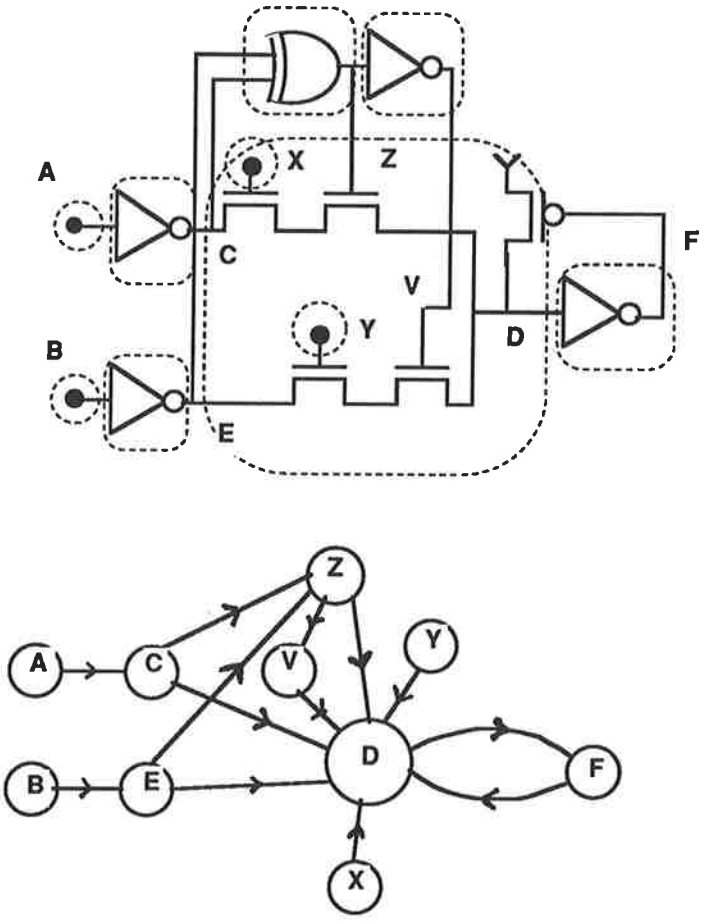
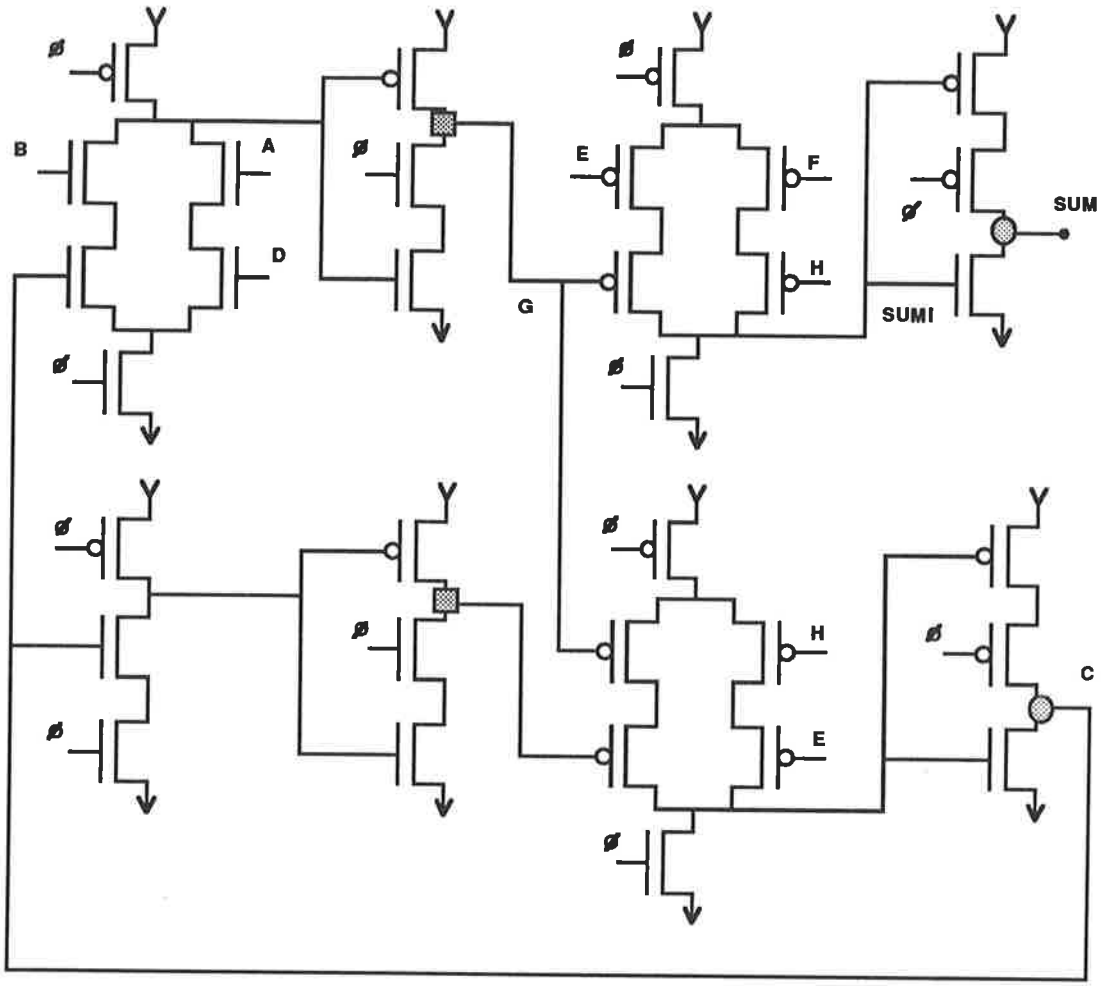
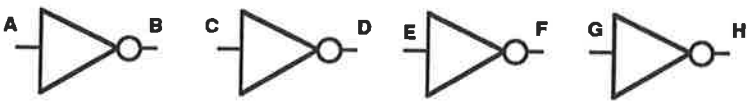


Figure 5 : Data-Dependency Graph





: PHASE-1 MEMORY NODE

: PHASE-2 MEMORY NODE

Figure 6 : A serial full adder