# FUNCTIONAL VERIFICATION IN AN INTERACTIVE SYMBOLIC IC DESIGN ENVIRONMENT

*Bryan Ackland*
*Neil Weste*

Bell Laboratories
Holmdel, New Jersey 07733

*ABSTRACT*

This paper describes verification techniques that have been implemented as part of an interactive symbolic IC design system. Circuit analysis programs perform node extraction and gate decomposition. They generate both transistor and gate level circuit desriptions which are used as input to a transistor level digital MOS timing simulator. The extraction programs make use of an intermediate circuit description language which captures both geometric placement and circuit connectivity. All programs are written in the *C* programming language and run under the *UNIX* operating system. An example is included to demonstrate the operation of these various techniques.

## 1. INTRODUCTION

Functional verification is an important and necessary step in the design of large scale integrated circuits. It is that part of the design cycle which eliminates most, preferably all, of the human errors introduced in the forward part of the design. It is generally a two stage process consisting firstly of automatic circuit extraction, in which electrical circuit descriptions are generated from the physical layout, and secondly of functional simulation of the derived circuit. Symbolic design techniques simplify the circuit extraction task as they introduce structural or circuit information into the layout file and remove unnecessary geometrical data. Interactive design techniques, however, place additional constraints on verification in that they demand fast response in order to avoid slowing the interactive design cycle.

This paper describes verification techniques that have been implemented on *MULGA* [1] - a *UNIX*† based interactive symbolic layout system. They consist of two programs which perform nodal circuit extraction and gate decomposition, and *EMU* - a transistor level MOS timing simulator. All programs are written in the *C* programming language and run under the *UNIX* operating system in a microcomputer based design station.

## 2. MULGA

Symbolic layout provides a means of abstracting the detailed and often laborious task of mask design. It offers the advantages of manual layout with regard to density, along with reduced design time and reduced likelihood of manual error. *MULGA* is a *UNIX* based interactive symbolic design system consisting of a suite of programs residing on a high performance color display station. Figure 1 shows the various software components of *MULGA* and

† UNIX is a trademark of Bell Laboratories

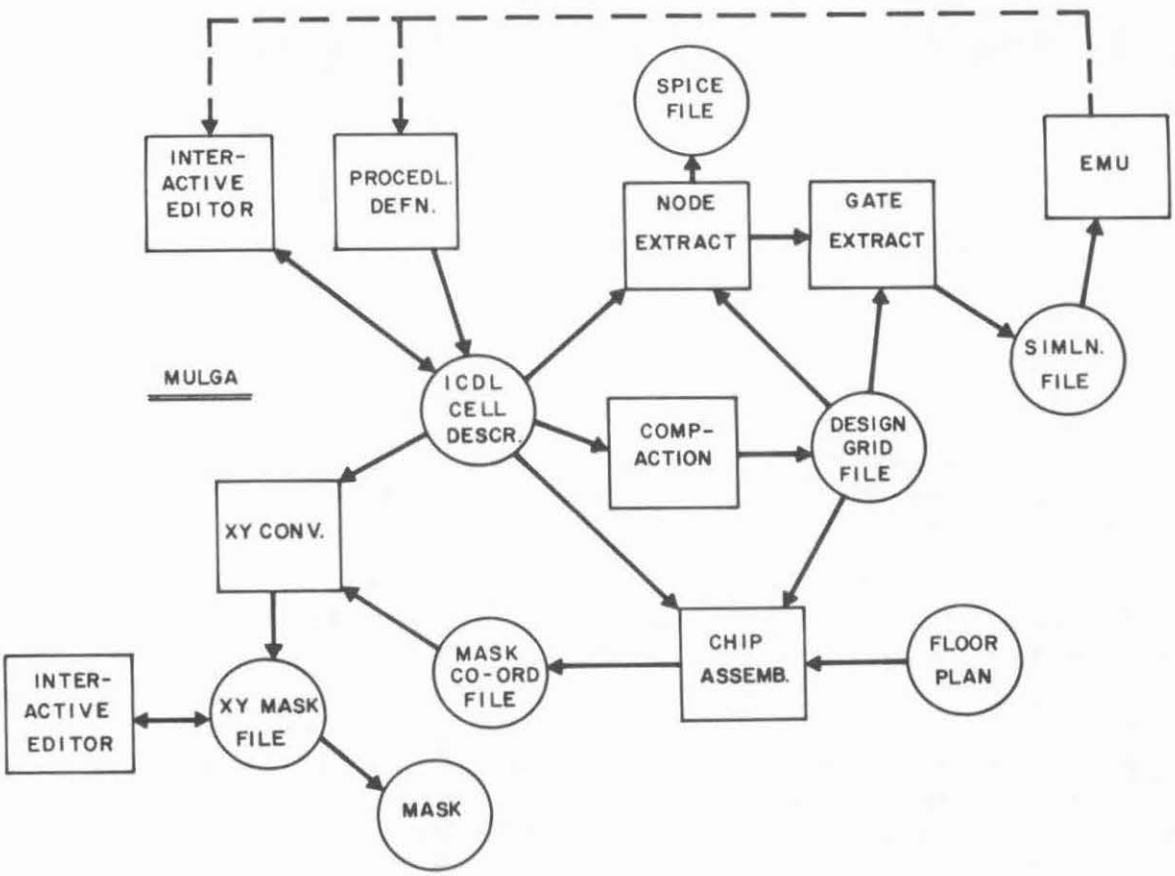


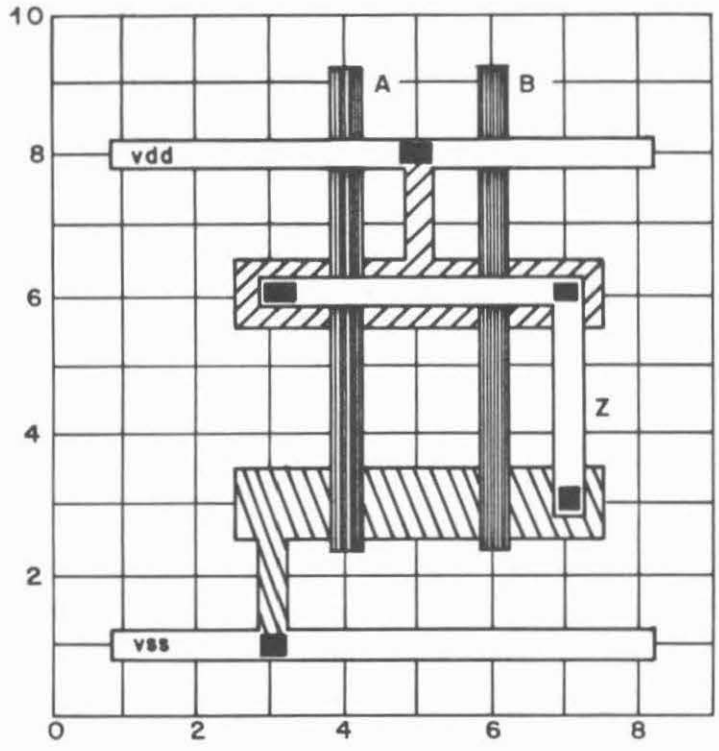

**Figure 1.** MULGA design system

the way in which they come together to effect a design.

The system is based around a symbolic Intermediate Circuit Description Language (*ICDL*) which uses a derivation of the co-ordinode notation introduced by Buchanan and Gray [2]. It combines circuit topology with geometric placement on a coarse virtual grid. In this way, the language captures designer intent with respect to the circuit, rather than a collection of abstract geometric forms.

The basic structure in *ICDL* is a cell which is a collection of elements placed on a virtual grid as shown in Figure 2. These elements may be devices, wires, contacts, pins, or other cell instances. Pins are named interconnection points that have no physical meaning in the final layout. They are a very important attribute of the language, however, and are used extensively in cell placement procedures and circuit verification. Figure 2 shows a CMOS 2-input nand gate represented graphically along side the textual *ICDL* description of the cell. Note that each line of text corresponds to an actual circuit component rather than a geometrical shape. Components are resticted to lie on grid intersection points as shown. Note, however, that this grid is only a relative placement network which defines the topology of the layout. Actual physical dimensions are determined later by a compaction process.

*ICDL* cell descriptions may be generated either via the interactive editor or else procedurally using the *C* programming language. Once the designer is satisfied with the symbolic




```
begin nand2
   pin al 1 1 vss
   pin al 1 8 vdd
   pin poly 4 9 A
   pin poly 6 9 B
   pin al 7 5 Z
   dev n or=1 4 3
   dev n or=1 6 3
   dev p or=1 4 6
   dev p or=1 6 6
   wire al 1 1  8 1
   wire al 1 8  8 8
   wire poly 4 3  4 9
   wire poly 6 3  6 9
   wire al 3 6  7 6  7 3
   wire Ndif 3 1  3 3
   wire Pdif 5 6  5 8
   con cut 7 3
   con cut 7 6
   con cut 3 1
   con cut 3 6
   con cut 5 8
end
```

**Figure 2.** ICDL description of 2-input CMOS nand gate

description, the file is compacted. The compaction program examines each symbolic grid line in the layout to determine how far it must be spaced from its neighbours in order to satisfy process design rules. Compaction information is stored in the design grid file. This file, along with the original ICDL description defines the minimum cell geometry assuming no other constraints in the design.

A chip assembler program takes the design grid file along with a specified chip floor plan and generates a mask coordinate file describing the actual physical location of the symbolic grid lines in the final layout. The chip assembler frequently needs to expand previously compacted cells in order to maintain global connectivity. The final step in the forward design path is the conversion and placement of cells into *XYMASK* data files. *XYMASK* is the geometric mask definition language used by the Bell System. A second interactive editor provides a means whereby the designer can view and evaluate his final design.

The verification phase consists of two circuit extraction programs and *EMU* - a transistor level MOS timing simulator. The first program performs node extraction and produces, in addition to the node list, a transistor level description of the circuit suitable as input to a circuit simulator such as *SPICE*. The second performs gate decomposition producing the higher level circuit description required by *EMU*. The following sections describe the operation of these three programs.

## 3. NODE EXTRACTION

The complexity of the circuit extraction process is heavily influenced by the nature of the layout definition language. One of the advantages of *ICDL* is that it carries implicit circuit connectivity information along with the physical topology. As shown in Figure 3, devices have designated connection points which are related by simple geometric rules to the center of the device. Wires serve to connect devices and external connections via interlayer contacts. Electrical connectivity is established when two elements exist on the same layer at the same virtual grid position. The pin construct aids the designer in naming specific nodes and connection points. This implicit connectivity is used, in conjunction with a simple algorithm, to arrive at a transistor node table description of the cell.

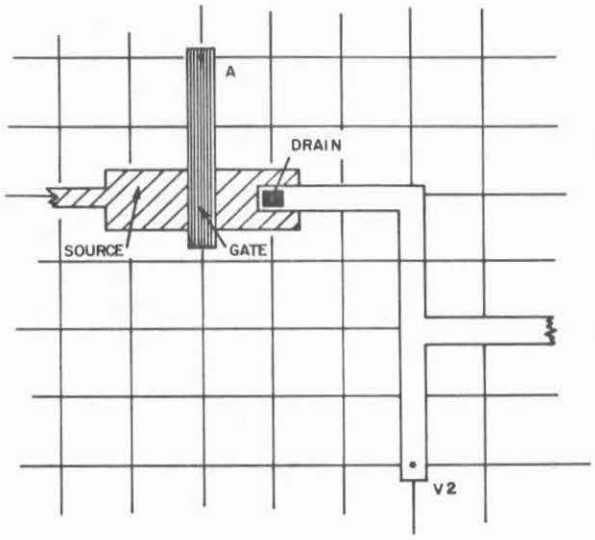



**Figure 3.** Implicit connectivity of ICDL components

The algorithm begins by first reading in a complete description of the *ICDL* cell. Following this, each pin, contact and transistor connection is assigned a different node number. Figure 4 shows a hypothetical net of components labelled in this manner. If there were no wires in the circuit, all such initial node numbers would be unique. Wires serve to connect components and reduce the overall number of unique nodes in the circuit. Accordingly, each wire is examined in turn to determine which nodes are redundant. A list is made of all nodes belonging to that wire. If the wire crosses another wire of the same type, connectivity is established by adding one node from the new wire to the old wire node list.

These wire node lists are used to eliminate redundant node numbers and generate a node net list description of the circuit. Pin names are used, wherever possible, to identify named nodes. Un-named nodes are given an internally generated name. Parasitic capacitance values for each node are calculated using the topology contained in the *ICDL* description along with absolute grid dimensions obtained from the mask coordinate file and specified process parameters. At this stage, sufficient information has been gathered to produce a transistor level circuit description. A simple filter converts this data into a *SPICE* simulation file. Figure 5 shows the simulation file generated from the 2-input nand gate described in Figure 2.

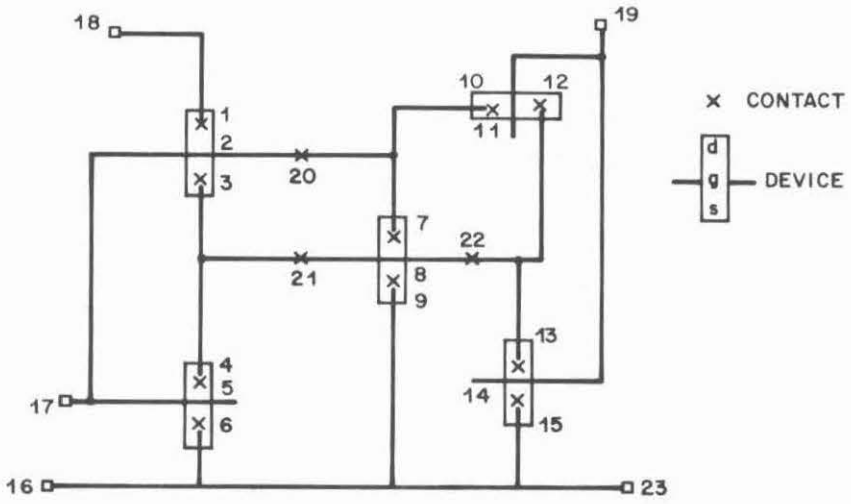


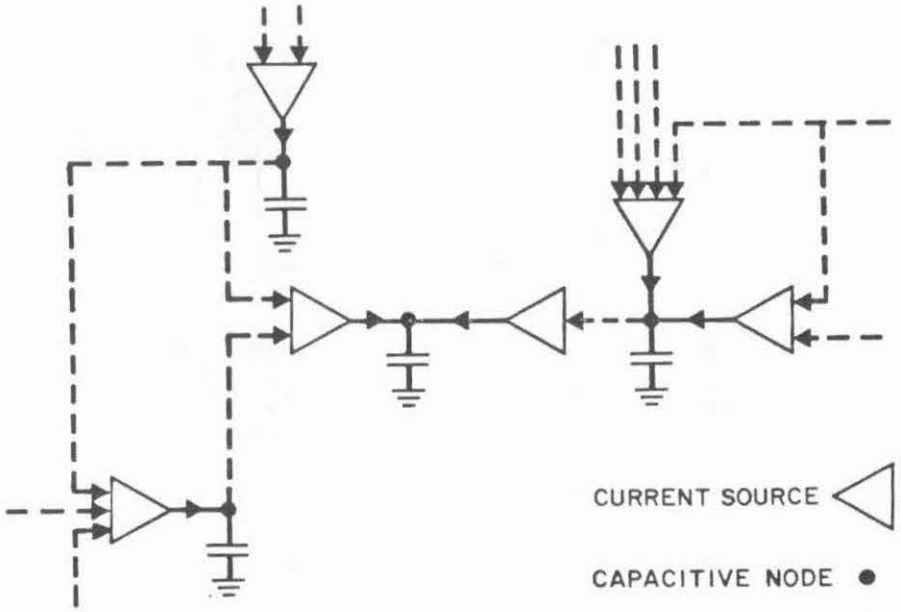

**Figure 4.** An example of initial node numbering

```
.SUBCKT  nand2 ( vss  vdd  A  B  Z  )
MN1  IØ  A  vss  VSS   N7A
MN2  Z  B  IØ  VSS   N7A
MP1  Z  A  vdd   vdd   P7A
MP2  Z  B  vdd   vdd   P7A
CMvss vss Ø CMTOSH 42
CNTvss vss Ø CN+H 12
CMvdd vdd Ø CMTOSH 42
CPTvdd vdd Ø CP+H 12
CPA A Ø CPTOSH 36
CPB B Ø CPTOSH 36
CMZ Z Ø CMTOSH 42
.FINIS
```

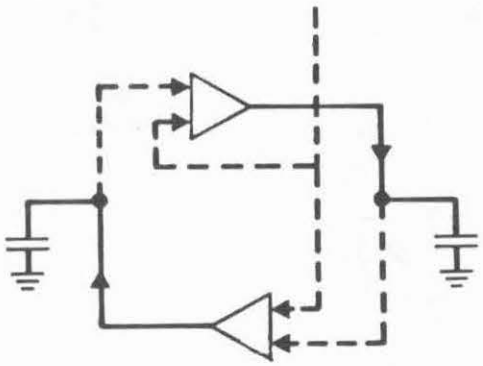
**Figure 5.** SPICE description of 2-input nand gate

## 4. SIMULATION

Analog circuit simulators such as *SPICE* give very accurate reliable feedback as to the functional operation of a circuit. They tend, however, to be very expensive in terms of computer and engineer time. In an interactive design environment, ease of operation and fast turnaround are of paramount importance and some accuracy can often be sacrificed to achieve this. For these reasons, a *UNIX* based MOS timing simulator known as *EMU* was developed. An important feature of the simulator is the fact that it is a resident part of the design station software and is therefore capable of giving the designer fast feedback concerning the operation of his circuit.




(a) Generalized showing current sources and voltage nodes



(b) Bi-directional circuit element

**Figure 6.** EMU circuit model

Timing simulators fall somewhere in between circuit simulators and logic simulators. They model digital circuits as collections of idealized transistors which may be grouped in a defined manner to form simple logic functions. Unlike logic simulators, they generate an analog waveform and are able to deal with limited analog effects such as charge storage and bidirectional circuit elements. Performance, however, is typically one to two orders of magnitude faster than analog circuit simulators. *EMU* is a MOS timing simulator which, like *MOTIS* [3], uses certain properties of the MOS transistor to greatly simplify the circuit model. These properties are represented by the following approximations:

1. The input resistance of the gate terminal is infinite, i.e. the input impedance is purely capacitive.
2. The leakage current of a MOS device is zero.
3. The channel may be represented as a voltage controlled d.c. current source.
4. In a self-aligned digital process, Miller effects are negligible.
5. The impedance to ground at any node is dominated by diffusion, gate and wiring capacitances which are voltage independent.

These approximations lead to the model shown in Figure 6(a). The circuit consists of a number of capacitive nodes interconnected by various voltage controlled current sources. Any number of current sources may drive a single node. Bidirectional circuit elements are represented by two sources as shown in Figure 6(b). Current sources may be transistors, load devices or compound gate structures.

### 4.1 Compound Gates

MOS gates typically consist of driver transistors connected in series/ parallel combinations as shown in Figure 7. Parallel branches pass current if any of the component elements conduct. This is equivalent to an OR function. Similarly, series branches conduct only if all component elements conduct - hence an AND function.

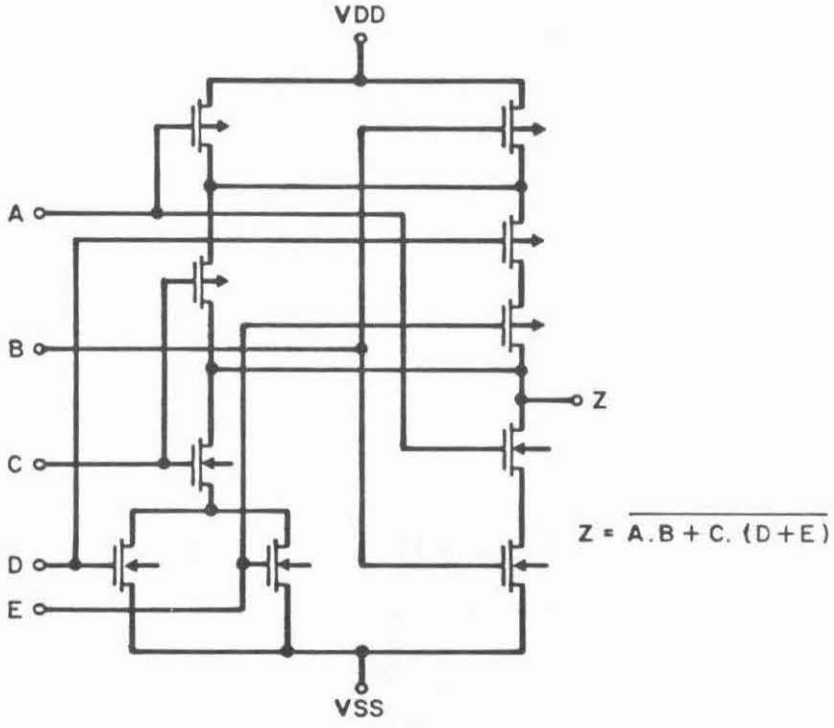
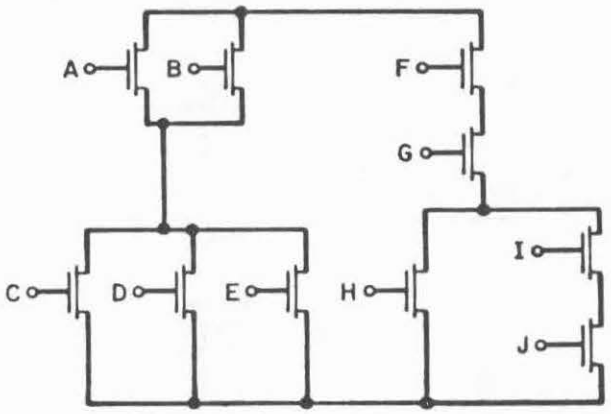



**Figure 7.** Typical CMOS gate construction

Although a logic gate can be simulated as a collection of individual transistors, much data space and simulation time can be saved if it can be modelled as a single current source. *MOTIS* uses the concept of a compound gate in which series/ parallel combinations of driver transistors are lumped together into a single data structure. It is based on the following approximations:

1. The total current sourcing a node is the sum of the currents sourced by all parallel branches connected to that node.
2. The current sourced by a branch containing several elements in series is the inverse sum of the currents that would be sourced by each element if it were the only element in the branch.
3. All transistors in the gate structure operate independently. Their only point of interaction is through the gate output node.

*EMU* uses this same compound gate construction. Driver transistors are specified in a "reverse polish" manner as shown in Figure 8. Transistor position in the gate is defined by the operators **push, parallel** and **series. Push** places a transistor on to an imaginary stack. **Parallel** means that the named device is in parallel with the top element of the stack. The resulting parallel combination replaces the element on the stack. **Series** means that the named device is in series with the top element of the stack. The resulting series combination replaces the element on the stack. **Parallel** or **series** without an operand, operates on the top two elements of the stack and pops the stack one position.




PUSH (A). PARL (B). PUSH (C). PARL (D). PARL (E). SERS.

PUSH (F). SERS (G). PUSH (I). SERS (J). PARL (H). SERS. PARL

**Figure 8.** Reverse Polish gate specification

This data structure is simply interpreted by a "reverse polish" current calculator. The calculator uses a real stack to store transistor conduction currents. The operator **push** causes a new current to be pushed on to the stack. The operator **parallel** causes two currents to be added. Similarly, the operator **series** causes two currents $I_a$ and $I_b$ to be combined according to:

$$\frac{1}{I} = \frac{1}{I_a} + \frac{1}{I_b}$$

### 4.2 Gate Advancement

Gate advancement is the technique by which node voltages are updated for each new time step of the simulator. Referring to Figure 6, each node in the circuit model is dominated by a capacitance to ground $C$. This means that for a sufficiently small time step, node voltages within the circuit are essentially constant. Source currents, which are in turn functions of circuit voltages are therefore also constant. Suppose a node is driven by $n$ current sources $I_1,...I_n$. The total current into the node is then $I = \sum_{k=1}^{n} I_k$. For a sufficiently small time step $(t - t_0)$, the new node voltage $V(t)$ is given by:

$$v(t) = V(t_0) + \frac{I(t - t_0)}{C}$$

The accuracy of this simple forward integration scheme depends critically on the choice of time step. If the time step is too large, circuit voltages and currents may change significantly during one iteration and errors will be introduced. In addition, voltages tend to overshoot leading to numerical instability - especially with bidirectional circuit elements such as transmission gates. On the other hand, if the time step is too small, much simulation time is wasted iterating over unnecesarily small time intervals.

Rather than leaving this delicate choice of time step to the operator, *EMU* automatically adjusts the time step to maintain simulation accuracy. It does this by monitoring the maximum voltage step occuring from one time instant to another and adjusting the timestep accordingly. Simulation thus proceeds rapidly during periods of low circuit activity and then slows down to critically examine those periods when changes are taking place.

### 4.3 Device Model

The basic Sah model is used to calculate MOS transistor channel current. The equations, as applied to an N channel device are:

*Cutoff*: $|V_{GS}| < V_t$

$$I_{DS} = 0$$

*Non−saturation*: $|V_{GS} - V_t| > |V_{DS}|$

$$I_{DS} = \beta \left(\frac{w}{l}\right) \left[(V_{GS} - V_t)\ V_{DS} - \frac{(V_{DS})^2}{2}\right]$$

*Saturation*: $|V_{GS} - V_t| \leq |V_{DS}|$

$$I_{DS} = \beta \left(\frac{w}{l}\right) \frac{(V_{GS} - V_t)^2}{2}$$

where $w$ and $l$ are the channel width and length respectively.

Back-gate bias effects are taken into account using table lookup techniques to calculate perturbations in threshold voltage.

### 4.4 OPERATION

The operation of *EMU* is characterized by four software states. Initially, *EMU* enters the **command** state. This is the common state from which all others can be entered. It is used to set simulation parameters, define clocks, display portions of the data base, initialize inputs and format the output of results. The **circuit** state is used to create a circuit description in the data

base. It is used to define inputs and nodes, assign gates and set circuit capacitances. The **process** state is used to set process parameters such as transistor threshold voltage and transistor gain. The **execute** state represents the actual simulation. Following execution, the simulator returns to the **command** state. **Command, input** and **process** states each have their own input language which may be entered interactively or via an predefined input file.

### 4.5 Performance

*EMU* is written in *C* and will run on any *UNIX* based machine. In particular, it runs on the *LSI*−11/23 - the host processor in the *MULGA* design station. It has also been implemented, however, on a *VAX*−11/780 and a Motorola 68000. The 11/23 implementation occupies approximately 25K bytes of code space leaving 30K available for circuit definitions. This is sufficient to hold a circuit of about 2000 transistors. Table I shows some simulation run times for a sample circuit - a 32×1 bit CMOS static RAM. This circuit contains 260 gates which in turn contain some 770 transistors. Note that even on the 11/23 microcomputer, this type of circuit can be simulated in a time which compares favorably with the time needed to perform an off-line simulation on a larger machine.

| SIMULATION RUN TIMES (secs) | |
|---|---|
| LSI-11/23 | 1094 |
| VAX-11/780 | 129 |
| 68000 (4 mHz) | 1010 |
| 68000 (8 mHz) | 585 |

TABLE I SAMPLE RUN TIMES

## 5. GATE DECOMPOSITION

The nodal analysis program described in Section 3 produces a transistor net list which can be used to generate a transistor level circuit description for *EMU*. The speed of the simulator, however, is directly related to the number of active current sources in the circuit. Accordingly, a gate extraction program has been written to process this transistor net list and convert it, where possible, into the compound gate structures recognized by *EMU*.

As a first step, nodes are characterized as either inputs, outputs or internal nodes. Outputs are defined to be those nodes which connect (in the case of a CMOS design) to the drains of both an N and a P transistor. Inputs are those nodes which connect only to transistor gate terminals within the cell. All remaining nodes are assumed to be internal. The algorithm then examines each output node in turn, and searches for all N devices connected either directly or indirectly (through an N channel) to that node. Any branches that pass through other output nodes are assumed to contain transmission gates and are ignored. All such N branches must eventually terminate at the negative supply rail if they are indeed part of a compound gate structure. Branches that do not satisfy this condition are discarded. This is equivalent to traversing the graph of all potential gate transistors connected to the output node. Figure 9(a) shows a number of devices connected to an output node Z. The N transistor graph that is derived from this circuit is shown in Figure 9(b). Each device is represented by a simple PUSH operator, consistent with the notation described in Section 4.1.

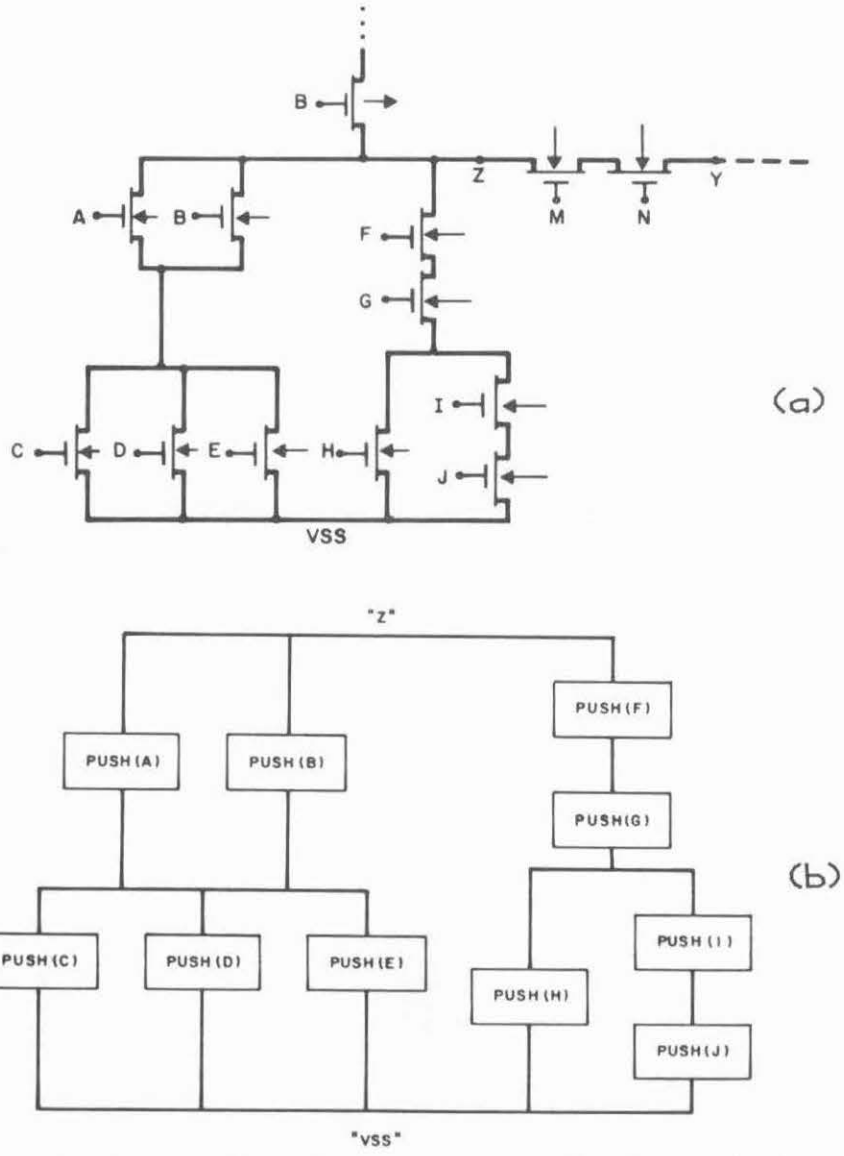



**Figure 9.** Example of gate device graph extraction from a circuit

The gate transistor graph is then reduced by successive parallel and series merging until a single branch remains. At each merge, the appropriate operator (SERIES or PARALLEL) is added to the branch description. The final result is the desired reverse polish specification of the transistor graph. Figure 10 shows the four steps required to reduce the graph of Figure 9.

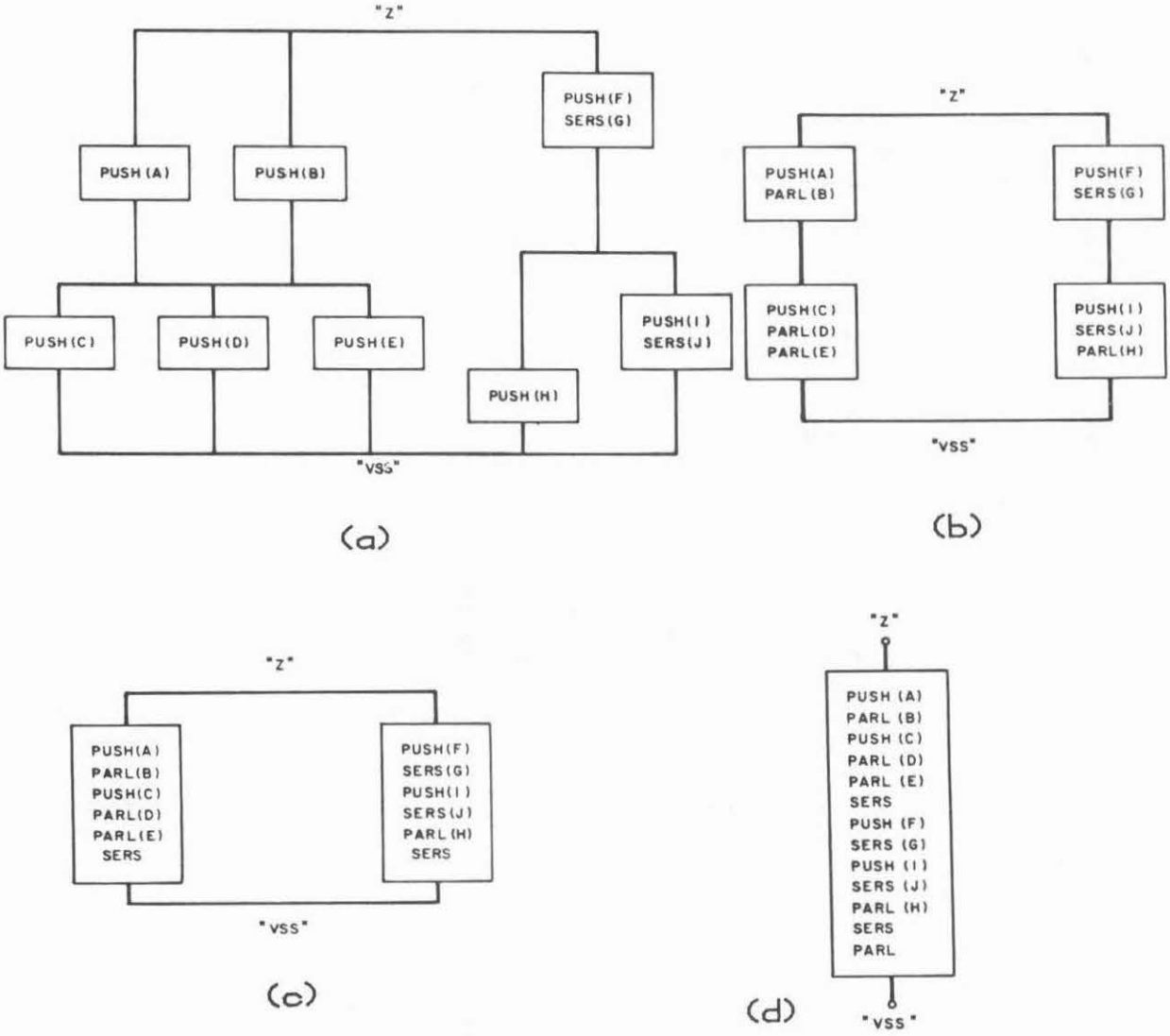



**Figure 10.** Series/parallel reduction of device graph

This process is then repeated for the P transistor chain. The two expressions are then combined to produce a complete gate description. Transistors not absorbed by this extraction process are retained as single transistor transmission gates. Figure 11 shows the gate level description that was extracted from the 2-input nand gate example of Figure 2. This description was given to *EMU* and Figure 12 shows the actual simulation result.

```
nand2(A, B, Z)
EXTERN A, B;
EXTERN Z;
{
     GATE(Z, 5);
          PUSH(A, 1); SER(B, 1); PCH();
          PUSH(B, 1); PAR(A, 1);
     INCAP(A, 30);
     INCAP(B, 30);
     OUTCAP(Z, 70);
     ROUCAP(Z, 9);
     ROUCAP(A, 11);
     ROUCAP(B, 11);
}
```

**Figure 11.** Gate level description of 2-input nand gate

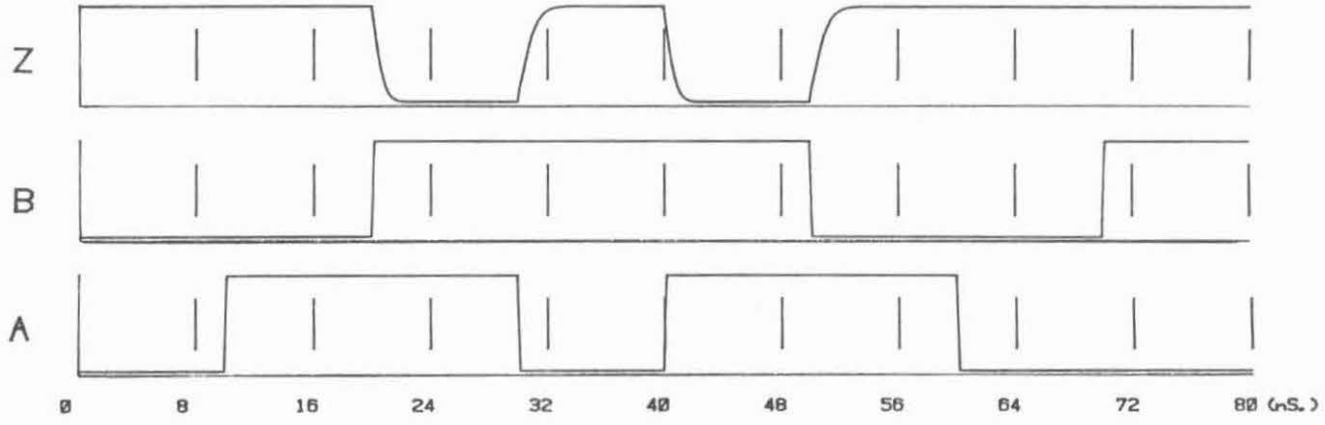



**Figure 12.** Simulation plot of 2-input nand gate

## 6. CONCLUSIONS

In an interactive design environment, verification tools must be fast and readily accessible in order to encourage the designer to perform this vital task. This paper has described three tools which meet these criteria by actually being part of the resident design station software. In addition, they have the advantage of being written in *C* and *UNIX*, making them readily transportable to other design systems.

## REFERENCES

[1] WESTE, N., "MULGA - An Interactive Symbolic System for the Design of Integrated Circuits", *Bell Sys. Tech. Journal,* to be published.

[2] BUCHANAN, I., "Modelling and Verification in Structured Integrated Circuit Design", *PhD Thesis, University of Edinburgh,* Scotland,1980.

[3] CHAWLA, B.R., GUMMEL, H.K., and KOZAK, P., "MOTIS - An MOS Timing Simulator", *IEEE Trans. on Circuits and Systems,* Vol. 22, No. 12, Dec. 1975, pp. 901-910.