Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

LSI COMPONENTS MODELLING IN A THREE-VALUED FUNCTIONAL SIMULATION.

G. Alia, P. Ciompi, E. Martinelli and F. Bernardini.

L78/9

Estratto da: Design Automation Conference Proceedings, June 19,20&21, 1978 LAS VEGAS, NEVADA.

LSI COMPONENTS MODELLING IN A THREE-VALUED FUNCTIONAL SIMULATION*

G. Alia, P. Ciompi, E. Martinelli
Istituto di Elaborazione della Informazione
C.N.R., Pisa, Italy
and
F. Bernardini
Selenia S.p.A., Roma, Italy

ABSTRACT

This paper deals with the problem of describing the behaviour of LSI components for a three-valued functional simulation.

The proposed functional description uses a set of predefined functional modules, named primitives, for handling data signals, and test blocks for handling control signals.

Some primitives with relative algorithms are described and a procedure for test blocks management with three-valued control signals is proposed.

1. INTRODUCTION

The technological evolution of integrated digital circuits, with the introduction of LSI components, makes the use of gate-level logic simulators highly inefficient, or rather impossible, in terms of memory and time requirements, when large networks are to be simulated for logic and design verification and fault analysis.

Hence it is necessary to develop a component description method, with a lower detail level than gate level description, which allows to reduce memory and computing time of the host computer [1,2].

A functional description, viewed as a connection of predefined functional modules, named primitives, promises to meet the above mentioned requirements.

The aim of this work is to propose a functional description of IC's based on the use of primitives for a three-valued simulation. In fact the following reasons, even if with different degrees of importance, make the introduction of the third value X necessary in a functional simulation:

a) necessity of assigning a value X to outputs and states at the initialization phase of the simulated circuit;

b) necessity of stopping a recognized oscillation, by setting the oscillating signals to X;

c) necessity of assigning unspecified values to outputs and states, when inputs take forbidden values;

d) necessity of assigning unspecified values to signals during certain time intervals, due to timing tolerances.

2. COMPONENT MODELLING

In general, at any level of detail, a component can be described in terms of its structure (inputs, outputs, states and so on), of its functionality (input/output relations) and of its timing.

All these characteristics must get rise from descriptions with a detail level that is fixed by a compromise between the necessity of a simulation system able to treat signals with a sufficient precision and the requirements of efficiency in respect of memory and computing time.

From our point of view only the following features of a component must be known in order to describe it completely:

a) input/output signals;

b) internal states;

c) timing relations among inputs;

d) functions and relative timing.

It is just worthy to mention that these features are always given be manufactures.

Considering these features, a component can be modelled, in a suitable hardware description language, by a structural description that defines inputs, with their timing relations, outputs and states, specifying their type, dimension and other similar attributes, by a functional description that represents the functions performed by the component, with their propagation delays, according to various input and state patterns.

In a three-valued simulation environment the correct propagation of signals from input to output can be obtained either by a comparison among all the results that may be got, specifying input X's anyhow and using an algorithm operating on two-valued data and assigning X to every output bit, whenever the value of that bit is different in two results at least, or by a specialized algorithm operating on three-valued data. However the exhaustive procedure would require an excessive

* This work has been sponsored by the Convention between Selenia S.p.A. and Consiglio Nazionale delle Ricerche.

computing time and the development of algorithms would be very inefficient, since an algorithm for each component would be designed, and very difficult, since the number of the functions performed by a component and the number of its inputs are normally quite large.

For these reasons our approach to a functional description of a component is a compromise between the two solutions mentioned above. It is based on the use of a number of simpler three-valued functions, named primitives, by which more complex functions can be implemented, according to a modular technique. This choice generally produces less precise outputs than an exhaustive procedure, since X values may be assigned as results, whereas these would be known. However if a signal value of the physical circuit is unknown, the one of the simulated circuit is unknown as well.

The functional description we propose consists of executive blocks, within which there are only primitives to be executed in a predefined order, alternated to test blocks within which input signals, commonly named control signals, internal states and dummy signals are tested; test blocks are points of the flow diagram at which the execution order of executive blocks can be modified.

From this approach, that distinguishes control signals from data signals, it arises the possibility of processing the two types of signals separately.

In the next sections the necessity of such a distinction is shown, with related problems and advantages; the criteria to implement primitives and a heuristic criterion to process control signals are also introduced.

3. THREE-VALUED PRIMITIVES

Since MSI and LSI IC's have very frequently a high parallelism degree on input/output data, it is convenient to adopt computing methods for three-valued functions that maintain such a parallelism in order to reduce computing time.

Three-valued functions can be computed either by boolean equations or by tabular methods, as well as by algorithms.

The first method is to be discarded because bits of the same output must often be valued by different boolean equations (for instance consider the sum of two numbers, represented as binary vectors of n bits).

The second method, however interesting, is useful only if the number of function arguments is small, because of memory occupancy.

Our choice for the implementation of primitives is to use algorithms. In a project for a logic simulator design we are presently developing, several relevant primitives have been picked out and their algorithms designed and implemented.

The chosen primitives can be grouped according to the classes of functions they represent:

a) logic and comparison functions (and, or, not, etc.)

b) arithmetic functions (addition, multiplication, etc.)

c) count and shift functions

d) decoding and multiplexing functions

e) assignment functions

f) storage and bidirectional elements modelling

g) transition sensitive inputs modelling.

All the algorithms for these primitives have been developed keeping in mind a three logic values coding previously defined [3]; it represents a three-valued variable a by two boolean variables a^0 and a^1 and by the following correspondence:

$$\begin{aligned} &a^0=1,\ a^1=0 && \text{iff} \quad a=0 \\ &a^0=0,\ a^1=1 && \text{iff} \quad a=1 \\ &a^0=1,\ a^1=1 && \text{iff} \quad a=X \\ &a^0=0,\ a^1=0 && \text{impossible} \end{aligned} \qquad (1)$$

This coding allows us to represent a three-valued function f by two boolean f^0 and f^1, which are two-valued and are generated from function f and its dual $\bar{f}$, both in SP form, substituting each argument x_i with x_i^1, if affirmed and with x_i^0, if negated. For example, let be

$$y=a\bar{b}+bc+\bar{b}\bar{c}$$

then

$$\bar{y}=\bar{c}b+\bar{a}\bar{b}c$$

and

$$y^1=a^1b^0+b^1c^1+b^0c^0$$

$$y^0=c^0b^1+a^0b^0c^1$$

According to the coding, we shall represent an n-bits three-valued variable, say c, with two n-bit two-valued variables c^0 and c^1; for example let c=10X110X, then

$$c^0=0110011$$

$$c^1=1011101$$

In the next sections some primitives and relative algorithms will be described, in order to show how different implementation criteria are applied to cases of different complexity.

3.1 Logic and comparison functions

3.1.1 Logic functions. Logic functions are implemented making use of their equations and their dual ones; for instance the equations of a logic AND are

$$c^1=a^1\wedge b^1, \qquad c^0=a^0\vee b^0 \qquad (2)$$

where a, b are the two binary variables to be anded, c the result; the apexes have the previously mentioned meaning.

It is immediate to note that only two machine instructions, namely an AND and an OR, are necessary to implement the function, whereas other codes would require a larger number of machine instructions.

In Tab. 1 the equations and the number of necessary machine instructions for several logic functions are shown:

function	coded equations	n° of instr.
AND	$c^0=a^0 \vee b^0$, $c^1=a^1 \wedge b^1$	2
OR	$c^0=a^0 \wedge b^0$, $c^1=a^1 \vee b^1$	2
NOT	$b^0=a^1$, $b^1=a^0$	2
NAND	$c^0=a^1 \wedge b^1$, $c^1=a^0 \vee b^0$	2
NOR	$c^0=a^1 \vee b^1$, $c^1=a^0 \wedge b^0$	2
XOR	$c^0=(a^1 \wedge b^1) \vee (a^0 \wedge b^0)$, $c^1=(a^1 \wedge b^0) \vee (a^0 \wedge b^1)$	6

Tab. 1

3.1.2 Comparison function. The comparison function has been implemented by the algorithm shown in the flow-chart of Fig.1. Such an algorithm arises from the following easy considerations: it is sufficient to know only two order relations, namely equality and consequence, to get all others; for the equality and consequence relation between two variables, represented as binary vectors the bits of which can assume values 0,1,X, the correlations summarized in Tab.2 can be established:

equality	consequence
true	false
false	true false indeterminate
indeterminate	false indeterminate

Tab. 2

The notation of Fig. 1 has the following meaning:

A,B input data variables

U work variable

OUT_1,OUT_2 output bits with the meaning:

$OUT_1=1$ A=B

$OUT_1=0$ $A \neq B$

$OUT_1=X$ indeterminate equality

$OUT_2=1$ $A > B$

$OUT_2=0$ $A \not> B$

$OUT_2=X$ indeterminate consequence

3.2 Arithmetic functions.

Among the four arithmetic functions, only the addition has been chosen to be described here, because adders are very commonly used IC's.

The procedure to perform the sum of two numbers a and b follows these steps:

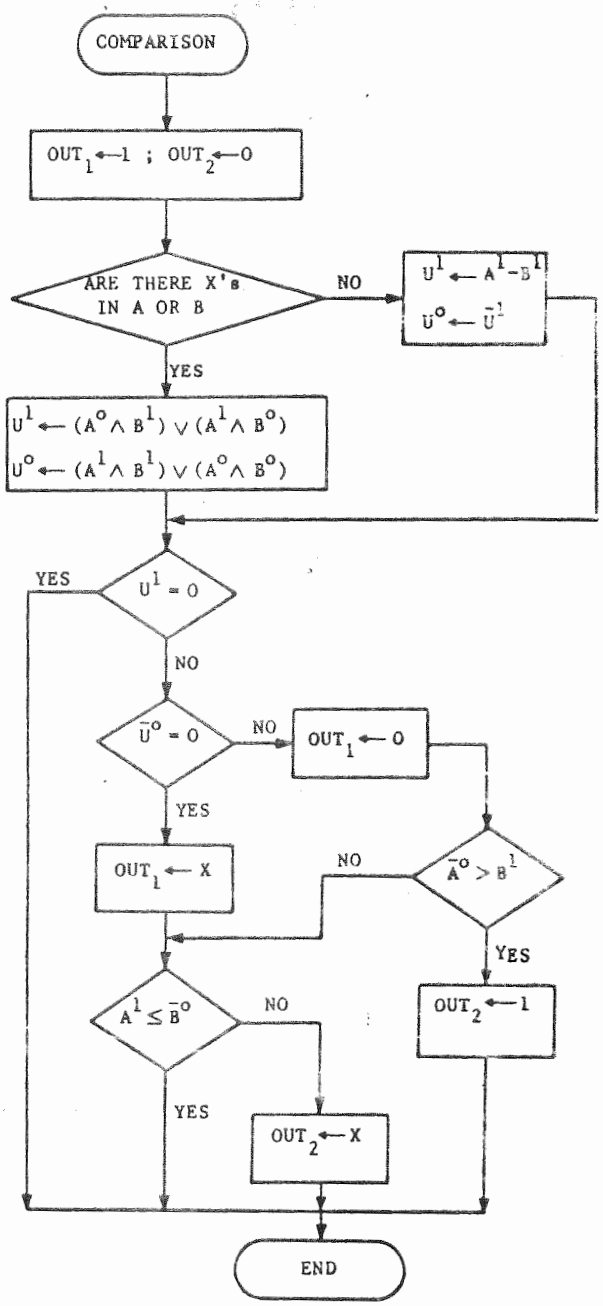

FIG. 1

a) the maximum and the minimum values of operands are obtained, by substituting the X' with 1 and 0 respectively(*);

b) the sums between maximum values and between minimum values are performed, let they be S_{max} and S_{min};

(*) With our coding the maximum and the minimum value of a variable c are $c_{max}=c^1$ and $c_{min}=\bar{c}^0$ respectively.

c) the result is computed by assigning the value 1 (0) to each bit, if the corresponding bits in S_{max} and S_{min} are both 1 (0), the value X if the corresponding bits in S_{max} and S_{min} are different or if at least one of the corresponding bits in the operands has value X.

For example, let a=0X0X01, b=0X0010 then a_{max}=010101, b_{max}=010010 a_{min}=000001, b_{min}=000010 and $S_{max}=a_{max}+b_{max}$=100111, $S_{min}=a_{min}+b_{min}$=000011; from this we have: S=XX0X11.

The above procedure can be justified with the following considerations:

1) if at least a bit of an operand has the value X, the corresponding bit of the sum is X, because 0+X=X, 1+X=X, X+X=X;

2) if S_{max} and S_{min} differ for a certain bit, that bit must be X, since there are at least two different pairs of operand configurations, i.e. a_{max}, b_{max} and a_{min}, b_{min}, that produce different results in the sum for that bit;

3) if S_{max} and S_{min} are equal for a certain bit and in the corresponding bit of the operands there are no X, the value of that bit in S_{max} or S_{min} is the correct value for the sum. In fact, if the above hypothesis holds, since the pairs of operands (a_{max}, b_{max}; a_{min}, b_{min}) are equal for that bit, also the carries must be equal. It follows that the carries are equal also for any other configuration obtained specifying the operand X's anyhow.

The algorithm for the addition is shown in the flow-chart of Fig. 2. It has been drawn making explicit reference to the adopted coding. The notation used in Fig. 2 has the meaning:

A, B	operands
C	carry output
R_x	vector containing one's anywhere at least one operand has value X
S_x	vector containing one's anywhere S_{max} and S_{min} are different

From the described procedure it is easy to recognize that it allows to perform the three-valued sum between two numbers of n bits in parallel, requiring a number of machine instructions much lower than an equivalent gate level procedure.

3.3 Count and shift functions.

3.3.1 Count function. In order to implement a procedure describing the three-valued count function correctly, it is necessary to take into account the correlation between the X's number that arrive at the input and the X's present in the current configuration(*).

(*)Let us consider a counter with initial value equal to zero and at the first step let the input take value X: the counter will take the configuration 0..0X; at the next step with input equal to X, the counter takes the configuration 0..XX, which would mean 0..00 or 0..01 or 0..10, but not 0..11.

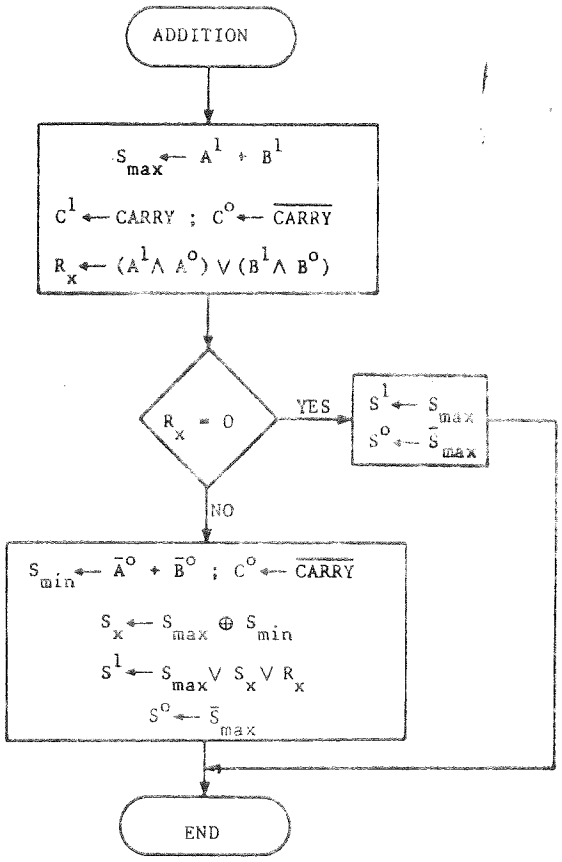

FIG. 2

In fact, if the count function is implemented making use of the addition algorithm iteratively, the counter will contain n bits with value X after a sequence of n X at the input, while the sum of n X is a vector containing $\lceil \lg_2 n \rceil$ X; in fact the maximum value of that sum is n, and the minimum is zero.

Hence to compute the counter value after k steps, in h of which the input has value X ($h \leqslant k$), these h steps are assumed to be the last ones and consecutive and the addition algorithm is executed, taking the counter configuration after k-h steps as an operand, and a vector containing the $s=\lceil \lg_2 h \rceil$ least significant bits of value X as the other one. The maximum and the minimum values of the latter operand are equal to 2^s-1 and to zero respectively.

Let c_i be the initial content of the counter, c the current content and i the input; the following implementation of the algorithm takes into account that the initial configuration may contain any number of X's and it is so summarized:

$c_{min} = \bar{c}_i^0$, $r_x = s = \bar{c}_i^0 \wedge \bar{c}_i^1$ (preset step)

if i=0 then no operation;

if i=1 then begin

$c_{min} = c_{min} + 1$,

$c_{max} = c_{min} + s$,

$c^1 = c_{min} \vee r_x \vee (c_{min} \oplus c_{max})$,

$c^0 = \bar{c}_{min} \vee r_x \vee (c_{min} \oplus c_{max})$,

end

if i=X then begin

$s = s + 1$

$r_x = r_x \vee s$,

$c_{max} = c_{min} + s$,

$c^1 = c_{min} \vee r_x \vee (c_{min} \oplus c_{max})$,

$c^0 = \bar{c}_{min} \vee r_x \vee (c_{min} \oplus c_{max})$,

end;

The above procedure, as well as the addition one, is able to compute the relative function exactly, without introducing more indeterminateness than it is necessary, at a low cost in terms of time and memory.

3.3.2 Shift functions. As far as shift functions are concerned, they can be implemented very easily using shift instructions of the host computer. Only the relation between word length of the host computer and the length of the operand is needed to be taken into consideration, in order to define the implementation procedure.

3.4 Read/Write functions in Memory.

Unlike the functions so far described, the primitive for memory modelling would require a very large amount of computing time, to perform the three-valued read/write functions with a correct propagation of the third value.

Hence our approach is a probabilistic one and the consequent procedures may give more indeterminateness than it is required.

Consider a storage device, having a capacity $C=2^N$, where N is the bit number of the address a and assume that each address bit can take the value 0,1,X with the same probability, while assume that each bit of input/output data can take only the values 0 and 1 with the same probability; in respect to the problem of introducing more indeterminateness in data this is the worst case hypothesis.

3.4.1 Memory read. The procedure to implement the read function follows the below described criteria:

- if address a does not contain X's, the content is immediately read out;
- if address a contains only one X, the output is given by a comparison between the contents of the words addressed specifying the X in the address as zero and as one. Note that with our coding one address is a^1 and the other is $\bar{a}^0$;
- if address a contains two or more X's, the result is completely indeterminate. The reason is that, under this hypothesis, the probability P that one of the four or more addressable words differ from the others for a certain bit, i.e. that the corresponding bit of the result assume the value X, is $P \cong 0.88$.

3.4.2 Memory write. According to hypothesis and notation of paragraph 3.4, the number of different address vectors with n bits of value X is:

$$C_x(n) = \binom{N}{n} 2^{N-n} \qquad (3)$$

The probability that n bits have value X at the same time is:

$$p(n) = C_x(n) \Big/ \sum_{n=0}^{N} C_x(n) = (2/3)^N \cdot \binom{N}{n} \cdot 2^{-n} \qquad (4)$$

From (4) the probability that at most k bits have value X at the same time is:

$$p(k) = \sum_{n=0}^{k} p(n) = (2/3)^N \cdot \sum_{n=0}^{k} \binom{N}{n} \cdot 2^{-n} \qquad (5)$$

Equation (5) was tabulated and results were drawn in the graph of Fig. 3; it shows that for $N > 6$ and $k = N/2$ (*) it is $p(k) > 0.9$.

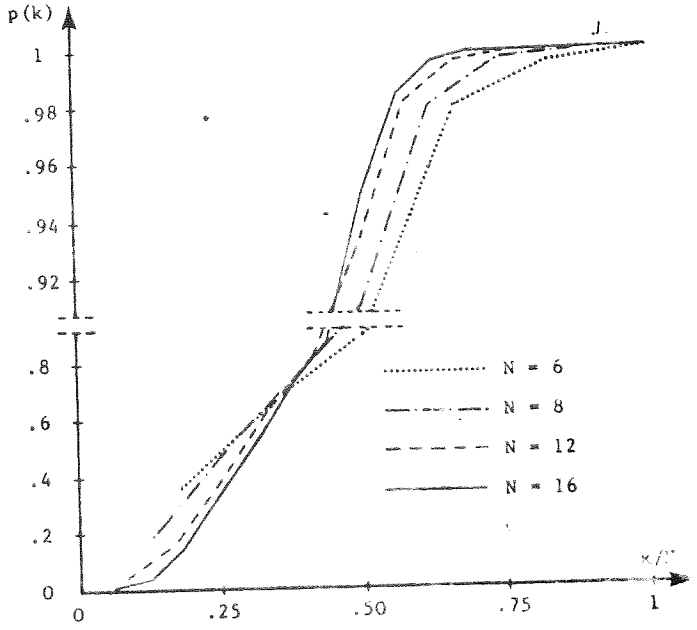

FIG. 3

(*) If the value X has a lower probability to appear in an address than the values 0 and 1, the probability p(k) increases, any other condition being equal.

The procedure to implement the write function is the following:

- if the number N_X of X's in the address is $N_X \leq \lceil N/2 \rceil$, all the possible addresses are specified and the configuration coming out from the comparison between the input datum and the content of each addressed word is stored in each of them;
- if it is $N_X > \lceil N/2 \rceil$ a completely undetermined datum is stored at all the specifiable addresses.

In the former case the propagation of the third value is correct, while in the latter it is a conservative one, but at most it concerns a 10% of cases.

Let us estimate the mean value E(I) of the instruction number needed to specify all the addresses from an address containing 0,1,X with the same probability:

$$E(I) = \sum_{n=0}^{N} i.2^n.C_x(n) \;/\; \sum_{n=0}^{N} C_x(n) = i(4/3)^N \qquad (6)$$

where $0 \leq n \leq N$ denotes the number of X's in the address and i is the number of instructions per address.

The procedure for address specification can be the following; let Inc be a vector containing 1 in the least significant X position and 0 elsewhere; $A_{min} = \bar{A}^0$ the minimum address value:

$B_x = \begin{cases} A^0 \wedge \bar{A}^1 \vee Inc \\ C = A_{min} \end{cases}$

L1 $C = C \vee Inc$ C contains an address

$C = C + B_x$

if $C = 0$ then goto L2

$C = C \wedge A^1$

$C = C \vee A_{min}$ C contains an address

goto L1

L2 . . .

It is immediately seen that three instructions per address are required to specify X's, so that, if N=10, equation (6) yields $E(I) \simeq 50$.

3.5 Transition Modelling.

Since in general component inputs are sensitive, apart from levels, also to transitions, as for instance timing inputs are, it is necessary to provide tools able to recognize if a signal undergoes a transition and the sense of transition.

The solution we propose is a primitive that compares two signal levels v_p and v_s at two consecutive simulation steps and that translates the transition into a dummy variable Trans which has a meaning according to Tab. 3:

v_p	v_s	Trans	Meaning
0	0	00	no transition occurred
0	1	10	low-to-high transition
0	X	X0	uncertain low-to-high transition
1	0	01	high-to-low transition
1	1	00	no transition occurred
1	X	0X	uncertain high-to-low transition
X	0	0X	uncertain high-to-low transition
X	1	X0	uncertain low-to-high transition
X	X	XX	uncertain low-to-high or high-to-low transition

Tab. 3

4. CONTROL VARIABLES MANAGEMENT

4.1 Definition.

Although control inputs and states can be handled as any other signals, the following considerations show the opportunity to distinguish between controls and data.

For example, suppose that a chip can perform four boolean functions f_1, f_2, f_3 and f_4, selected by two variables a and b; then in the whole the behaviour of the chip can be described by the relation:

$$f = ab f_1 + a\bar{b} f_2 + \bar{a} b f_3 + \bar{a}\bar{b} f_4$$

If the number of functions to be executed and input signal number are large or if the function is to be described by algorithms to take advantage of parallelism on output, it is not convenient to describe the function f by a single primitive operating on three-valued data, because storage requirements for tables or machine code would increase excessively.

It is possible to consider control inputs and states to be a vector address pointing to the function to be selected and to compare all the results that may be got in a three-valued simulation environment, when some control signals have the value X. If the number of control signals assuming the value X is large an exhaustive comparison needs an excessive computing time. On the other hand, when complex chips are to be described, it is often necessary to test some intermediate results of the function. For these reasons we extend the definition of control variables and propose a heuristic solution to their processing.

The behaviour of a component is described by a flow diagram Δ, as defined in section 2.

We define control variable of a component each variable appearing in a test block of Δ, at least. In the following for the sake of simplicity we assume that all the control variables are one bit-variables; however the proposed procedure can be easily extended to n bits-variables.

4.2 The heuristic procedure.

Each solution simplifying the evaluation of chip outputs in terms of time and memory, even if it renounces a correct propagation of X's, would have the following points:

1) it would not yield specified values for variables that surely would be undetermined with a correct propagation of X's in controls;

2) it would not modify variables not involved in any operation selected by specifying the X's in controls anyhow.

The latter condition is especially important at the initialization phase of component state when, even if many X's are present, already specified and not involved registers need to be saved.

In order to make our solution clearer, some considerations need to be introduced here.

If some control variables take the value X independently N times during the execution of the flow diagram, then it is not known what function is performed among the set F(N) of all the functions, affecting the same output, that can be defined specifying the N occurrences of the value X anyway.

Let us assume that:

a) an output bit of a complex function having all the operands specified has about a 0.5 probability of being 1 (0);

b) the results of $|F(N)|$ distinct functions evaluated with the same input configuration have a very low probability of being correlated.

It follows that the probability that a bit of the result, obtained by comparison among the outcomes of $|F(N)|$ functions, is 0, 1 or X, assumes the values shown in Table 4.

$\|F(N)\|$	Probability of value 0 $P_0=\left(\frac{1}{2}\right)^{\|F(N)\|}$	Probability of value 1 $P_1 = P_0$	Probability of value X $P_X = 1-2P_0$
1	0.5	0.5	0
2	0.25	0.25	0.5
3	0.125	0.125	0.75
4	0.0625	0.0625	0.875
5	0.03125	0.03125	0.9375

Tab. 4

Hypotheses a) and b) are not always strictly true, but in practice they are approximately true for the majority of chips.

Basing ourselves on previous considerations, we introduce an algorithm which does not compute the results by a comparison among $|F(N)|$ possible results, but which sets the value X into all output bits when the number $|F(N)|$ is greater than a limit k and thus makes a percent error decreasing very quickly as k grows.

Let V be the set of variables appearing in Δ and partition it into two subsets I and L. Subset I contains all the input variables, i.e. input signals and preceding states that we suppose not appearing as result in any Δ statement, and all the variables obtained only by the execution of statements appearing in executive blocks and containing only input variables and preceding states as arguments. Note that the flow diagram Δ can be always redefined, using dummy variables, in such a way as each variable obtained in the old diagram by the execution of at least one statement of above mentioned type appears in the new diagram in that statement only, and thus it belongs to I. Subset L is defined as V-I and contains, among the others, output variables, i.e. output signals and next states, not contained in I.

We name "total state" of the algorithm after the execution of each test block the set of values taken by variables in V and the name labelling the branch of Δ to be run.

Moreover we associate each executive block P of Δ, that is a branch of Δ, with an executive block Q formed by a sequence of statements such that:

- a statement in P which sets an L-variable is associated with a statement in Q setting the value X into that variable;
- a statement p_i in P setting an I-variable α is associated with a statement in Q that performs the comparison between the value of α set by the statement p_i and the preceding one (with our coding the comparison corresponds to perform two OR instructions between the two variables α^1 and between the two variables α^0 obtained before and after the execution of p_i; the result of comparison is stored into α).

The algorithm is designed as a supervisor procedure that controls the execution of Δ and works according the following criterion:

- as long as the number of possible executions of Δ, coming from any specification of X's in test blocks, is lower than or equal to k, all the possible branches are run in the proper order and with the proper value of specified control variables and all the possible final states in V are compared to find the result;
- when the number of the possible executions becomes greater than k, a rough, but faster version of Δ is activated and all the sequences Q associated to branches that can be run are executed, starting from branches that have not yet produced a final state. Once a test block, in which the variable to be

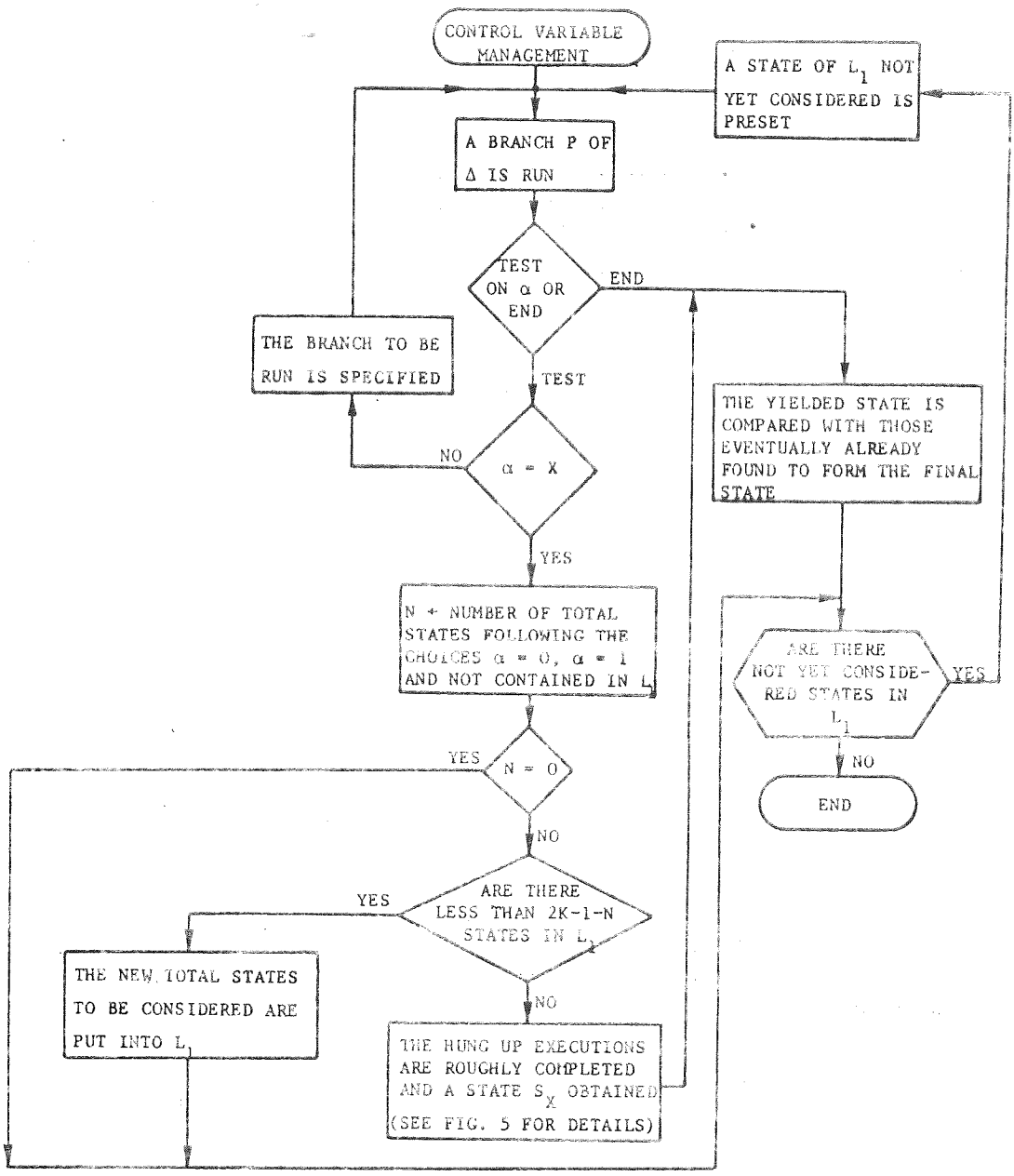

FIG. 4

examined has a determined value, is reached, this value is made equal to X, if the variable belongs to subset L; otherwise the branch pointed by the test is run and afterwords, if in any execution statement this variable takes a different value, also the other branch beginning at the mentioned test block is executed.

Note that because of the characteristics of sequences Q, the same branch never needs to be run twice, since each statement like p_i always yields the same result, having only input variables as arguments. The algorithm is shown in more detail by the flow-charts of Figg. 4 and 5.

As far as requirements 1) and 2) are concerned, the above explained method guarantees point 1) completely, while it may happen that the value of a variable becomes X, even if the variable is not involved by any operation selectable specifying X's in controls. However in our opinion the majority of these situations are handled by our procedure and it would be very difficult and expensive to take into account the remaining ones.

4.3 An Example.

In order to illustrate briefly the heuristic procedure proposed, it is shown in Fig. 6 a simple example describing the behaviour of

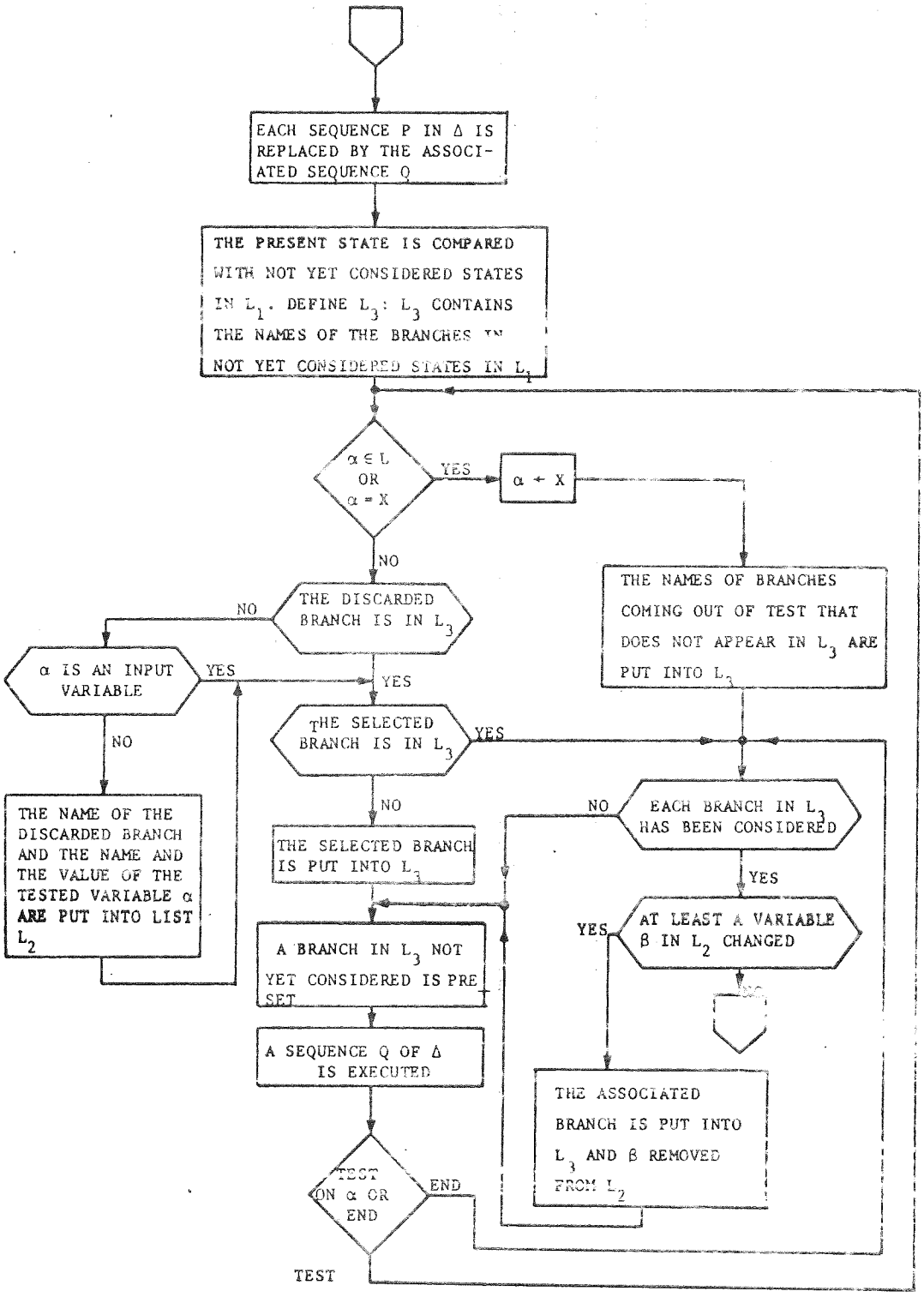
EACH SEQUENCE P IN Δ IS REPLACED BY THE ASSOCIATED SEQUENCE Q
THE PRESENT STATE IS COMPARED WITH NOT YET CONSIDERED STATES IN L_1. DEFINE L_3: L_3 CONTAINS THE NAMES OF THE BRANCHES IN NOT YET CONSIDERED STATES IN L_1
$\alpha \in L$ OR $\alpha = X$
YES
$\alpha \leftarrow X$
NO
THE DISCARDED BRANCH IS IN L_3
NO
α IS AN INPUT VARIABLE
YES
YES
NO
THE NAME OF THE DISCARDED BRANCH AND THE NAME AND THE VALUE OF THE TESTED VARIABLE α ARE PUT INTO LIST L_2
THE NAMES OF BRANCHES COMING OUT OF TEST THAT DOES NOT APPEAR IN L_3 ARE PUT INTO L_3
THE SELECTED BRANCH IS IN L_3
YES
NO
THE SELECTED BRANCH IS PUT INTO L_3
NO
EACH BRANCH IN L_3 HAS BEEN CONSIDERED
YES
YES
AT LEAST A VARIABLE β IN L_2 CHANGED
A BRANCH IN L_3 NOT YET CONSIDERED IS PRESET
A SEQUENCE Q OF Δ IS EXECUTED
THE ASSOCIATED BRANCH IS PUT INTO L_3 AND β REMOVED FROM L_2
TEST ON α OR END
END
TEST

FIG. 5

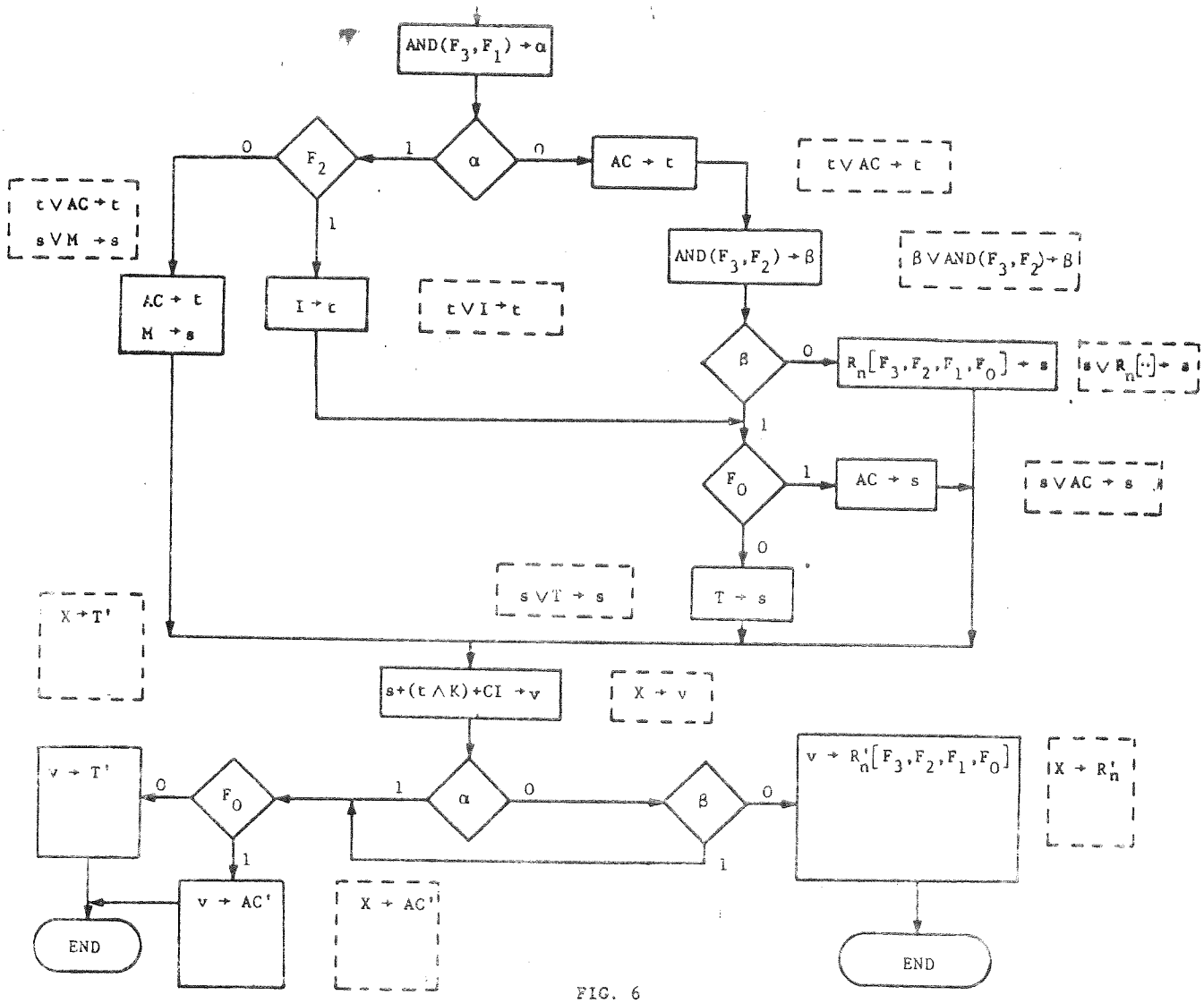

FIG. 6

Central Processing Element SIGNETICS N 3002, limited to group 3 functions, as defined in Tables 5,6. For the sake of simplicity, all the control variables in the example are one bit-variables.

Group 3 Functions	
Register Group	Equation
I	$R_n + (AC \wedge K) + CI \longrightarrow R_n$
II	$M + (AC \wedge K) + CI \longrightarrow AT$
III	$AT + (I \wedge K) + CI \longrightarrow AT$

Tab. 5

In Tab. 5 AT denotes registers AC or T, as specified by control variable F_0 (see Tab. 6).

Register Group	Register	F_3	F_2	F_1	F_0
I	R_0	0	0	0	0
	R_1	0	0	0	1
	⋮	⋮	⋮	⋮	⋮
	R_9	1	0	0	1
	T	1	1	0	0
	AC	1	1	0	1
II	T	1	0	1	0
	AC	1	0	1	1
III	T	1	1	1	0
	AC	1	1	1	1

Tab. 6

In the flow diagram the following variables appear, forming set V:

$F_0 \ldots F_3$ input single-bit control variables

AC accumulator (preceding state)

AC' accumulator (next state)

M M-bus data

I I-bus data

K K-bus data

CI carry input

$T, R_0 \ldots R_9$ internal registers (preceding state)

$T', R'_0 \ldots R'_9$ internal registers (next state)

α, β, s, t, v dummy variables

Set V is partitioned into two subsets I and L as defined in 4.2 composed as it follows:

$I: F_0 \ldots F_3, AC, M, I, K, CI, T, R_0 \ldots R_9, \alpha, \beta, s, t$

$L: V, AC', T', R'_0, \ldots R'_9$

With each P execution block of the diagram, the corresponding Q block is associated (dotted blocks in Fig. 6).

The data structure for the heuristic procedure proposed in 4.2 is now complete.

5. CONCLUSIONS

The problem of describing the behaviour of LSI components for a three-valued functional simulation has been faced.

In a three-valued functional simulation environment two objects are to be pursued:

1) a correct propagation of x signals from input to output that does not yield results with specified value if they would be undetermined and does not introduce indeterminateness if not necessary;

2) a low amount of memory and computing time of the host computer.

The two objects are hard to be both reached, as the first would require time-consuming algorithms and a great amount of memory for exhaustive procedures taking into account interdependence between X values of different signals.

The proposed functional description, based on two different techniques for handling data and control signals and on heuristic procedures for three-valued processing, is a satisfactory precision-cost tradeoff.

REFERENCES

[1] Abramovici, M., Breuer, M.A., Kumar, K., "Concurrent Fault Simulation and Functional Level Modeling", Proc. 14th Design Automation Conference, June 1977, pp. 128-135.

[2] Szygenda, S.A. and Lekkos, A.A., "Integrated Techniques for Functional and Gate-Level Digital Logic Simulation", Proc. 10th Design Automation Conference, June 1973, pp. 159-172.

[3] Alia, G., Bernardini, F., Ciompi, P., Martinelli, E., Simoncini, L., "Tecniche per il Calcolo di una Funzione Booleana a Tre Valori", Internal Report 77004, Convenzione Selenia-CNR, I.E.I., Pisa, May 1977.