Switch-level fault detection and diagnosis environment for MOS digital circuits using spectral techniques

G. Ruiz
J. Michell
A. Burón

Indexing terms: Fault detection, Digital circuits

Abstract: A switch-level fault detection and diagnosis environment for MOS digital circuits using a compression data method based on a spectral signature is described. The selected fault model includes an MOS transistor permanently On and Off, breaks in internal gate lines, and shorts between two internal nodes of different logic gates, or between the internal nodes within the same complex gate. Circuit editing is performed in modules containing simple switch-level descriptions of the transistors. From the module structure a fault list is created, which will later be processed to eliminate all equivalent faults (fault collapsing). Simulation of the faults contained in this list, and Walsh or Haar spectral analysis of the outputs, allow a data file to be created, containing a list of faults detected, a list of diagnosed fault groups and the spectral signature for each of these groups. The circuits are tested by comparing the information contained in this file and the data provided by a logic analysis system (LAS).

1 Introduction

The aim of testing an integrated circuit is to detect the permanent physical faults introduced in the different phases of production. The method most widely used to show up physical faults, once the circuit has been manufactured, involves applying a series of stimuli (test vectors) to its inputs and testing if the response is correct. In this respect, fault simulators have proved to be an important tool in evaluating test sequence efficiency, and in providing information on the number of faults detected; CAD VLSI systems, therefore, usually have this kind of simulator built into their structure. Fault simulators carry out three basic tasks: preprocessing, fault simulation and output analysis. Preprocessing includes circuit editing and the generation of a fault list, which is compatible with the fault model selected (switch, logic gate or functional level). This fault list is then processed to remove all the equivalent faults (fault collapsing). In fault simulation, the circuit is excited with the same test sequence for each of the fault situations contained in the list. Subsequent analysis of the circuit response allows the compilation of a dictionary of faults, classified into three categories: detected, nondetected and unstable.

The accuracy of the results obtained with fault simulators is directly related to the fault model selected [1]. The 'stuck-at' fault model at logic gate level has been the most widely used in test vector generation and fault simulation, as it is a simple model, technology-independent and compatible with the main test vector generation algorithms [2]. This model is inappropriate for MOS circuits [8, 9], where the principal failure modes result from transistor degradation, short nodes (short) and open-lines (break) [3, 4]; indeed, shorts and breaks at the metallisation and diffusion level are the commonest fault types in this technology [5]. In digital CMOS circuits, these faults produce error-specific behaviour such as memory and analogue effects not considered in the classic stuck-at model [6]. Wadsack [7] broadened the classic stuck-at model and proposed a new structure at logic gate level that allowed the behaviour of some of these faults to be modelled. With this new structure it is possible to use the generation and simulation algorithms developed for the stuck-at model. However, it has the drawback of requiring a considerable increase in circuit logic, and its application is limited to relatively small circuits.

At the transistor level, the switch model has emerged as the best alternative to deal with digital MOS circuits [1, 10, 11]. At this level of abstraction, MOS transistors are modelled by means of switches controlled by gate voltage, with the possibility of adding resistances and capacitances to improve accuracy when necessary. This enables most failure modes in MOS circuits to be dealt with directly and fairly accurately, without an undue increase in computation time. Dealing with shorts and breaks is rather more complicated, as it is also necessary to know the distribution of the connection lines and nodes in the circuit. To deal with these faults, circuit descriptions at layout level, which are not included in most current fault simulators, must be used. Therefore, fault simulation at the transistor level, starting from circuit descriptions at the layout level, seem to be the most appropriate for dealing with the commonest types of faults in MOS circuits directly and efficiently [5]; this generally requires a moderate increase in computation time.

Testing techniques for digital circuits based on data compression allow the circuit response to be analysed more simply, and require less memory than the conventional bit-by-bit comparison [12]. The spectral fault signature method uses one or several spectral coefficients of the circuit response for this purpose. The Walsh spectrum

Paper 8911E (E10), first received 11th October 1991
The authors are with the Department of Electronic, University of Cantabria, Avda, Castros s/n, Santander 39005, Spain

has proved to be the most appropriate of these methods [17]. Using this approach, Hsiao and Seth [13] analysed the efficiency of the method when using random test sequences and the highest value coefficient of the spectrum as the signature. Susskind [14] showed that the use of two coefficients guarantees the detection of any stuck-at faults in unate combinational circuits; for the same kind of circuit, Miller and Muzio [15] proposed three general signature methods using $n + 1$ coefficients (n being the number of input variables). Later, the same authors developed a new strategy to obtain spectral fault signatures for stuck-at fault detection in combinational circuits with a single output [16]. Recently, Hurst [18] studied the use of spectral techniques in a compaction testing method for combinational circuits with multiple outputs.

This paper describes a test environment for the detection and diagnosis of faults in digital NMOS and CMOS circuits by means of spectral analysis techniques. First, the selection of the transistor-level fault model is discussed, and a form of circuit description is introduced which uses transistor-level modular structures based on the matrix model [19]. Next, there is a description of the complete test environment structure, which includes pseudographic circuit editing, the creation of fault lists by means of automatic fault generation and the elimination of equivalent faults (fault collapsing), simple switch-level simulation, spectral analysis of the circuit response and testing of the circuits from the data provided by a logic analysis system (LAS).

The matrix description of the circuits makes it easier to develop the simulation algorithms, whose most important features are low memory requirement and high-speed processing. The process of spectral fault evaluation allows efficient results to be obtained for fault coverage and diagnosis, and determines the most appropriate spectral signature for the selected fault list [24]; this phase can be performed using the Rademacher–Walsh set as the basis [20], or the normalised Haar set (complete or reduced) [21]. In previous works, some of the software tools have been used to generate pseudorandomly ordered test sequences which give acceptable results in the detection and diagnosis of stuck-open faults in CMOS combinational circuits [22]. They have also been used to obtain more suitable test sequences for the diagnosis of sequential circuits [23]; as an extension to this latter treatment, an automatic test sequence generator has been incorporated which uses a functional description of the sequential circuit.

2 Fault model and matrix model for NMOS cells

Testing techniques for integrated circuits (ICs) require fault models that reflect with sufficient accuracy the behaviour of the circuits in the presence of the commonest faults for a given technology. However, when the selection of any specific model is made, a compromise has to be reached between accuracy when modelling real failures, and simplicity in computation. The most frequent faults in digital MOS circuits mainly affect the signal propagation times, noise margins and power consumption; faults with this origin are called parametric, as they simply degrade the IC parametric performance. Of the total set of failures, only a small percentage modify the logic behaviour; to this group, however, belong most of the failures observable in MOS circuits. The interpretation of these failures at the circuit and logic levels allows five groups of faults to be defined: stuck-at line faults, stuck-at-on (stuck-on) and stuck-at-open (stuck-open) transistor faults, line break faults (breaks), bridge faults (shorts), and diverse faults not included in these categories. The selection of a fault model to include the first four types (the fifth is extremely complex) conditions the characteristics of any fault simulator for two reasons, namely, that stuck-on/open faults are dealt with more simply using transistor-level circuit descriptions, and that shorts and breaks require a knowledge of the circuit topology.

When these points had been considered, a fault model was chosen including the following:

(*a*) MOS permanently open (stuck-open); fault modelled as an open circuit between its drain and source terminals.

(*b*) MOS permanently closed (stuck-on); fault modelled as a short between the drain and source terminals.

(*c*) Breaks in the internal lines of a gate, including the physical break of connection lines between two internal nodes of a gate whether it is a metal line, a diffusion line or a polysilicon line.

(*d*) Short between two internal nodes of different logic gates or between the internal nodes within the same complex gate.

The simple switch model representing the transistors is appropriate for dealing with the faults included in this model. This model deals readily with transistor faults, short-nodes and open-lines, and simplifies the simulation processes. Choosing the simple switch model means that the matrix model can be used to represent digital MOS circuits. In this model, each basic cell consists of a reticular distribution of transistors and horizontal connection branches. Fig. 1 represents an NMOS cell.

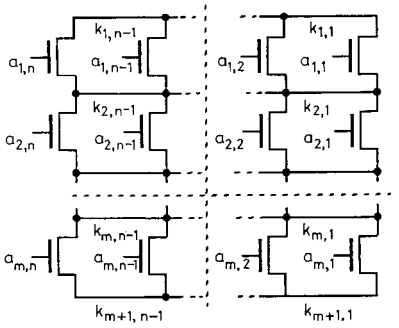

Fig. 1 *The basic NMOS cell matrix model*

Each transistor is assigned an identifier $a_{i,j}$, with $i = 1, \ldots, m$ and $j = 1, \ldots, n$, which defines its logic state; the pair i, j indicates the placement of the transistor in the matrix structure. In this structure, the elements of the parallel branches connect the terminals of continguous transistors, the lowest horizontal branch is connected to the ground, and the highest, which represents the output, should be connected to Vdd by means of a load transistor. Each horizontal connection element has an identifier $k_{i,j}$, with $i = 1, \ldots, m + 1$ and $j = 1, \ldots, n - 1$, which represents the presence or absence of an interconnection; the pair i, j indicates the placement of each span of connection (disconnection) in the reticular structure. With this matrix model, the basic cell is represented by two matrices, one of transistors, M, and the other of horizon-

tal branches, K:

$$M = \left[\begin{array}{c|cc} a_{1,n} & a_{1,2} & a_{1,1} \\ a_{2,n} & a_{2,2} & a_{2,1} \\ \hline a_{m,r} & a_{m,2} & a_{m,1} \end{array}\right]$$

$$K = \left[\begin{array}{c|cc} k_{1,n-1} & k_{1,2} & k_{1,1} \\ k_{2,n-1} & k_{2,2} & k_{2,1} \\ \hline k_{m+1,n-1} & k_{m+1,2} & k_{m+1,1} \end{array}\right]$$

where $a_{i,j} \in \{0, 1\}$ and $k_{i,j} \in \{0, 1\}$. With this notation, $a_{i,j} = 1$ represents the transistor when on (in a conduction state), and $a_{i,j} = 0$ when off (in a cutoff state). Likewise, $k_{ij} = 1$ indicates the presence of a horizontal branch and $k_{ij} = 0$ its absence.

The evaluation of a basic cell involves finding out whether a conduction path exists between the lowest horizontal branch, connected to the ground, and the output node of the cell situated in the highest horizontal branch. To do this, it is necessary to analyse each of the $a_{i,j}$ and $k_{i,j}$ elements and to create a path map on which it is possible to find out whether at least one path exists in the cell. This procedure can be greatly simplified by using a mathematical transformation which we shall term ***KOR***. This converts the original matrix M row by row into another $M^* = \{a^*_{i,j}\}$, on which it is much easier to determine the presence of possible conduction paths. The ***KOR*** transformation is defined by the relationships:

$$\begin{aligned} b_{i,j} &= [a_{i,j} \wedge a_{i,j+1}] \bullet k_{i,j} \wedge a_{i,j} \bullet k_{i,j} \\ b_{i,j+1} &= [a_{i,j} \wedge a_{i,j+1}] \bullet k_{i,j} \wedge a_{i,j+1} \bullet k_{i,j} \\ &\text{with } j = 1, 2, \ldots, n-1 \end{aligned} \quad (1)$$

$$\begin{aligned} a^*_{i,j} &= [b_{i,j} \wedge b_{i,j+1}] \bullet k_{i,j} \wedge b_{i,j} \bullet k_{i,j} \\ a^*_{i,j+1} &= [b_{i,j} \wedge b_{i,j+1}] \bullet k_{i,j} \wedge b_{i,j+1} \bullet k_{i,j} \\ &\text{with } j = n-1, n-2, \ldots, 1 \end{aligned} \quad (2)$$

where $\wedge$ represents the Boolean function OR and $\bullet$ the function AND. This is applied downwards from $i = m$ to $i = 1$. For each row of transistors in a cell, the ***KOR*** transformation process begins with a search from bottom to top, and from right to left in Fig. 1, to identify the states of each pair of consecutive transistors and their associated horizontal connection state (eqn. 1). Each transistor is given an intermediate identifier $b_{i,j}$, which indicates the presence or absence of a connection path between the current level and the higher one. Then the same search is performed downwards (eqn. 2), from left to right in Fig. 1. When this operation ends, each transistor will have been given an identifier $a^*_{i,j}$, which can be used to locate the different conduction paths between one level and the next. With the matrix produced by the ***KOR*** transformation, M^*, the presence of connections between the output and the ground can easily be determined; in fact, if there is at least one column j in which $a^*_{i,j} = 1\ \forall i$ is verified, there will be at least one conduction path between the cell output and the reference terminal. To obtain a better understanding of the usefulness of this transformation, Fig. 2 shows an example cell where the transistors have been replaced by boxes in which their logic state is shown: 1 (on) and 0 (off). With the matrix notation used so far, the information contained in this cell can be expressed in terms of M and K. A search for conduction paths, like the one shown in the cell in the figure, would be complicated for an analysis of these matrices. However, the M^* matrix produced by the ***KOR*** transformation shows such a path, as one of its columns (the middle one) is unitarian.

The characteristics of the cell matrix model are as follows:

(i) The information about the states of the transistors and about the basic cell topology can be represented at the bit level. The M and K matrices of the unitary elements can be stored using small integer arrays. Thus, this kind of representation is much simpler than the system of nodal description of the circuit used by other simulators. As a result, the search process for conduction paths in the cell can be implemented fairly simply.

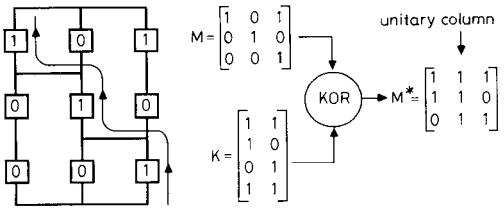

Fig. 2 *Example of **KOR** transformation*

(ii) The faults included in the selected model can be dealt with. Stuck-open/on faults produce modifications of the M matrix, whereas breaks in lines and shorts between nodes introduce modifications of the M and K matrices. Thus a stuck-open fault in the transistor $a_{i,j}$ is equivalent to considering $a_{i,j} = 0$, and a stuck-on fault, $a_{i,j} = 1$. Similarly, $k_{i,j} = 0$ represents a horizontal break and $k_{i,j} = 1$ a horizontal short.

(iii) The grouping of the information in numerical arrays, and the fact that the search process is based on binary logic operations, means that the processing of these cells is carried out at high computational speed with a low memory requirement.

3 General description of the test environment

The test environment is a set of aids to test tools for digital MOS circuits, the block diagram for which is shown in Fig. 3. This environment has been created with the aim of making fault detection and diagnosis easier, and a test strategy has been devised to be carried out in two phases. In the first, simulation of the faults in the circuit is performed and the result obtained is a data file containing the classification of the faults from their spectral signatures. In this phase, the tasks of graphic editing of the circuit, fault-list generation and elimination of equivalent faults (fault collapsing), fault simulation and spectral analysis of the circuit response, are all performed. This phase is applied repeatedly until the percentage of detected and diagnosed faults is within the limits established. In the second phase, the circuits are tested by evaluating the data provided by the logic analysis system (LAS) and those previously obtained by simulation; the LAS is used to apply the test sequence to the circuit and to store its responses.

4 Graphic editing of the circuit

Graphic editing is carried out on circuits that are divided modularly into cells, as shown in Fig. 4. Each cell is composed of connection lines and a level module. The connection lines may be internal, that is, lines connected to the module's own outputs, or external, that is, the input

lines of the cell. A level module is represented graphically in a similar way to that used in the matrix model, but appropriately modified to include multiple outputs in its structure. In this module, the topological description of the circuit at the transistor level is defined from externally supplied data; in this phases, there are a number of tools to make circuit editing easier. The level modules may be of three types: NMOS, CMOS and pass logic.

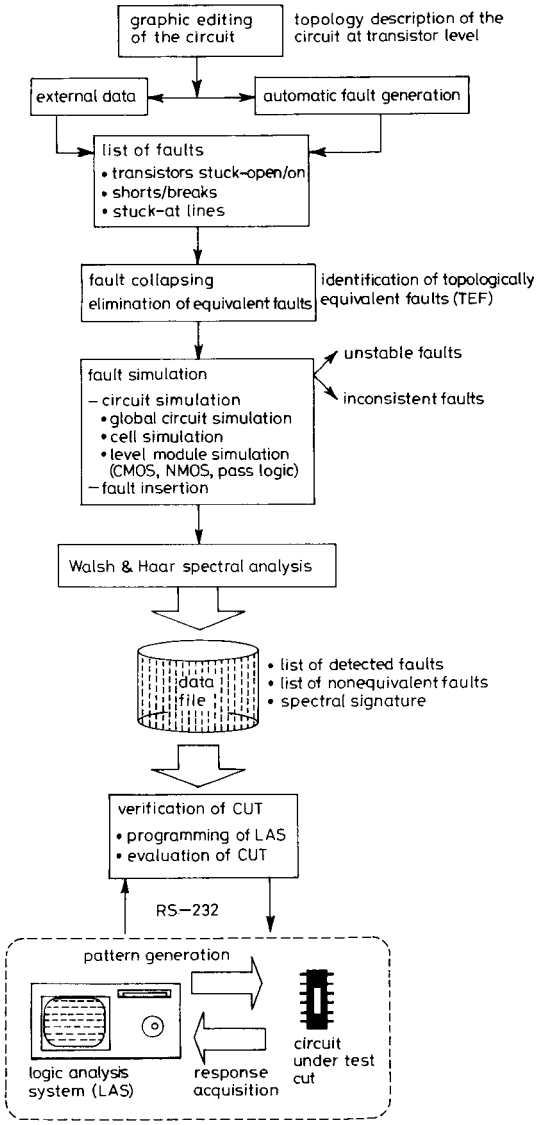

Fig. 3 *General structure of test environment*

NMOS level modules are made up of basic cells of N transistors whose lowest horizontal branches should be connected to the reference terminal, and whose uppermost branches define the output nodes. At each output node there should be a load element, which is not considered in the level module since in the simulation process it is assumed to be fault-free. Fig. 5*a* shows a typical example of this type of module. Each pair of characters Nk represents an N transistor whose gate is controlled by variable k; the state of this variable coincides with that of an internal or external line of the cell. The characters ! found in the upper and lower part of each transistor represent the drain and source terminals, respectively. Fig. 5*a* shows an NMOS level module with two output nodes, the first defined by the drain of transistor 4 and the second defined by the parallel branch joining the drains of transistors 1 and 2.

CMOS level modules are composed of two basic submodules: a submodule of P transistors, or submodule-P, situated in the upper part, and the other of N transistors, or submodule-N, situated in the lower part. In submodule-P, the highest horizontal branch should be connected to the power supply and the output nodes are defined in the lowest horizontal branch. In submodule-N, the lowest horizontal branch should be connected to ground and the output nodes are defined in the uppermost horizontal branch. The outputs of this level module are obtained by grouping pairs of output nodes of the two submodules. Fig. 5*b* shows two basic CMOS gates defined in the same level module. In submodule-P, one transistor can be seen to have been replaced by ! to represent a short between its drain and source terminals; with this procedure, vertical connection lines can be defined, making the topological description of the circuit easier. This level module has two outputs: the drain of transistor $N4$ and an extension of drain $P4$ define the first output node, and the other output node takes in the horizontal branch common to $N3$ and $N1$ and the drain of $P3$.

With the level modules described above, it is not possible to deal with circuits with pass transistors, but the concepts established will be of great use when the problem is approached. It is important to decide whether its construction should be based on N and P submodules or whether it is advisable to combine the two submodules into one by using as the basic element the simple switch for the pass transistor. Although the second choice means losing part of the information about the real distribution of the different transistors in the integrated circuit, the use of submodules N and P does not represent an appreciable advantage over the use of only one submodule; on the other hand, this may mean a reduction in computation time. For these reasons, the level module chosen for pass logic has the structure of a basic cell in which each transistor is replaced by a pair of P and N transistors in parallel. Fig. 5*c* shows the level module corresponding to a multiplexor where pass logic has been used; the label kTt indicates that the corresponding pass key is composed of a P transistor controlled by variable k and an N transistor controlled by variable t. Unlike the others, this level module has input nodes defined in the lowest horizontal branches, and output nodes defined in the uppermost horizontal branches. To describe NMOS pass logic with this structure, only the N transistors need be used.

5 Generation of reduced fault lists

The fault list contains the set of faults used in the circuit simulation process. This list consists of transistor faults, shorts between nodes and breaks in internal lines to the level module, and may also contain cell input and output line faults. This last type of fault, which can be considered within the classic stuck-at model, is easily dealt with by the simulator, giving greater flexibility to the application of the tools.

The fault list may be introduced externally or may be created by an automatic process which, starting from the cell and level module structure, generates the possible faults in the circuit. The search of the circuit topology makes it possible to identify all the selected faults that must be included in the list. The faults short and break are more difficult to deal with because of the great diversity that may exist in the level module. To optimise their generation, a process has been devised to eliminate topologically equivalent faults, and thus reduce the number to be dealt with. To illustrate this process we have chosen the CMOS circuit shown in Fig. 6, whose layout is given in Fig. 7. When, for example, the faults 'node-short' and 'line-open' between adjacent metal lines (as shown in Fig. 7) are considered, it is necessary to adapt the level modules to the circuit layout, which is not a requirement, for instance, with cell input and output line faults and transistor faults. The level module shown in Fig. 8 represents a possible solution for dealing with the shorts and breaks mentioned; this figure explicitly shows the shorts and breaks in the horizontal and vertical connection lines representing the faults previously indicated in the layout.

Fault collapsing is a technique that means the number of faults processed in the fault simulation can be reduced using the identification of equivalent faults as a basis. In our case, and given the fault model selected, the application of this technique makes it necessary to define the concept of topologically equivalent faults (TEF) in the level module. Two faults, F1 and F2, are TEFs if the logic functions associated with the level module under

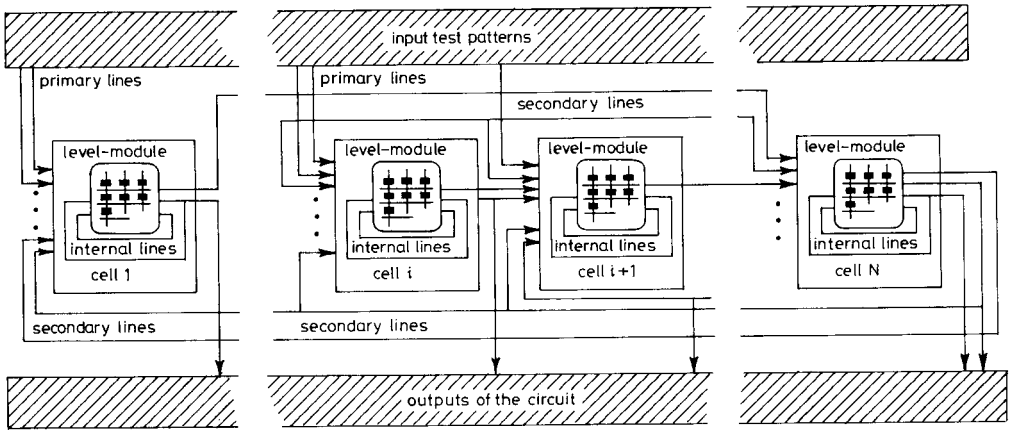

Fig. 4 *General structure of multilevel modules*

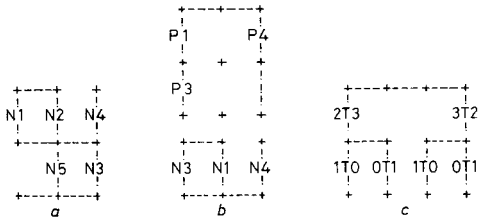

Fig. 5 *Examples of level modules*
a NMOS
b CMOS
c Pass logic

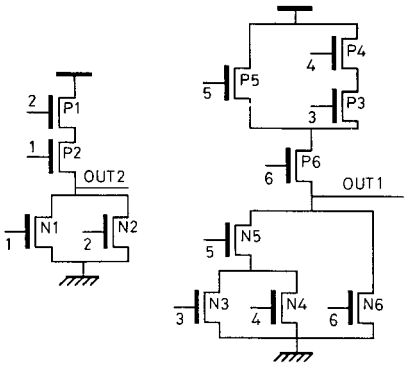

Fig. 6 *CMOS circuit*

these two fault conditions are identical. The identification of TEFs is carried out by analysing the level module topology, using the following operations:

(i) TEF in transistors stuck-open. Let $F1$ and $F2$ be two stuck-open faults; $F1$ and $F2$ are then TEFs if both transistors are in series. In Fig. 6, the stuck-open faults $P2$ and $P1$ are TEFs.

(ii) TEF in transistors stuck-on. Let $F1$ and $F2$ be two stuck-on faults; $F1$ and $F2$ are then TEFs if both transistors are in parallel. In Fig. 6, the stuck-on faults $N3$ and $N4$ are TEFs.

(iii) TEF between a transistor stuck-open and a breakline. Let $F1$ be a stuck-open fault and $F2$ be a break; $F1$ and $F2$ are TEFs if $F2$ leaves an open drain or the source of the transistor affected by $F1$. Several examples appear

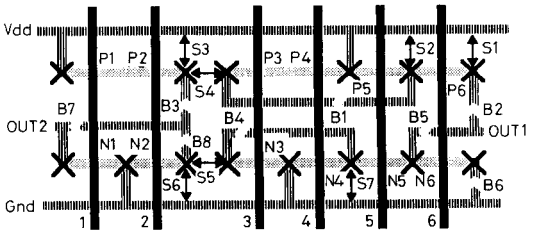

Fig. 7 *Stick layout of Fig. 6*
Polysilicon
Metal
× Contact
P–N diffusion

in Fig. 8; stuck-open faults $P2$ and $B3$, stuck-open faults $P6$ and $B2$, stuck-open faults $N3$ and $B4$. However, $B5$ is not a TEF with stuck-open faults $N5$ and $N6$, because two transistor terminals intervene.

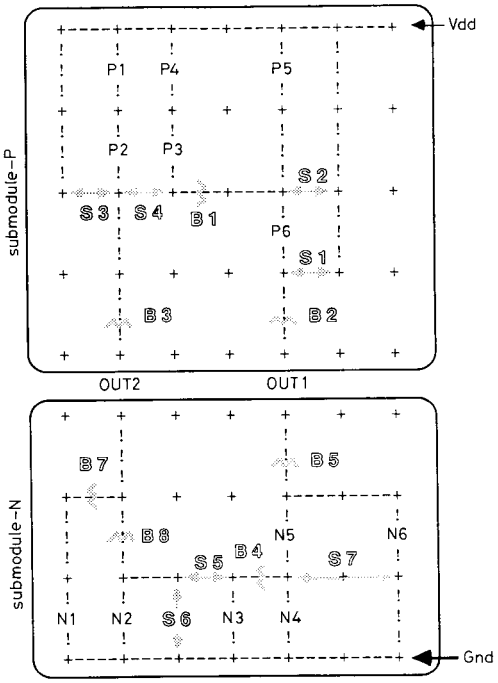

Fig. 8 *Level module for the previous stick layout*

(iv) TEF between a transistor stuck-on and a short-line. Let $F1$ be a stuck-on fault and $F2$ be a short; if $F2$ short-circuits the drain and source terminals of the transistor affected by $F1$, then $F1$ and $F2$ are TEFs. In Fig. 8, stuck-on faults $N2$ and $S6$, stuck-on faults $P5$ and $S2$, and stuck-on faults $N4$ and $S7$ are TEFs.

TEF identification has the property of transitiveness; that is, if $F1$ and $F2$ are TEFs and $F2$ and $F3$ are also TEFs, then $F1$ and $F3$ are TEFs. Examples of various fault types which are TEFs can be seen in Fig. 8. Thus $P1$ stuck-open, $P2$ stuck-open and $B3$ are TEFs; $P4$ stuck-open, $P3$ stuck-open and $B1$; $P5$ stuck-open, $P6$ stuck-open and $B2$; $N1$ stuck-on, $N2$ stuck-on and $S6$. In this way, the fault collapsing effected on the original fault list means a reduction of between 15% and 35% in the number of faults that intervene in the simulation process.

6 Fault simulation

The description of the circuits in cells means that serial fault simulation is the most appropriate method of fault simulation. This choice was motivated mainly because the modular fragmentation into cells, which may contain several complex gates and a large number of transistors, conditions the use of other fault-simulation methods. In serial fault simultion, the fault-free circuit and the faulty circuit are simulated separately with the same test sequence. This method is simple, but the simulation of one circuit at a time may slow up the process; nevertheless, the simulation algorithms developed are fast enough for it to be viable.

For simulation purposes, a cell consists of a level module and its input lines. Input lines may in turn consist of primary lines (circuit input lines), secondary lines (lines connected to the outputs of other cells) and internal lines (lines connected to the outputs of the cell itself). The same output node of the cell may act on a secondary line and/or an internal line and/or an output of the circuit. The simulation process begins with the classification of the cells in terms of the type of input lines they have; since they are difficult to evaluate, cells may be:

(i) Primary cells, which only have primary lines;

(ii) Secondary cells, which have secondary lines and may also have primary lines;

(iii) Mixed cells, which have internal lines and may also have primary lines;

(iv) Complete cells, which have secondary and internal lines and may also have primary lines.

One of the fundamental tasks of the simulator is to extract information about the connections between cells, the object of which is to optimise the exploration processes of the circuit. To achieve this, each cell output is assigned a data structure consisting of two lists of cells, $\boldsymbol{A}$ and $\boldsymbol{B}$, created from the secondary lines of the circuit. List $\boldsymbol{B}$ contains the cells connected to this output and list $\boldsymbol{A}$ is made up of those $\boldsymbol{B}$-list cells whose secondary lines do not imply a feedback path to the cell being dealt with. The simulation of a circuit is performed according to a hierarchical scheme, shown in Fig. 9, which has three

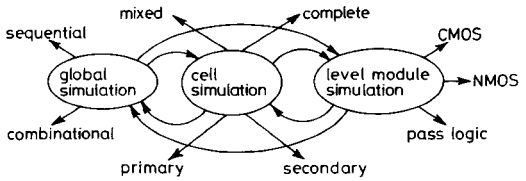

Fig. 9 *Hierarchy of the circuit simulation process*

operational levels: global simulation, cell simulation and level module simulation. Relaxation algorithms are used to simulate only those cells whose logic state has changed, thereby reducing reducing computation time considerably; these algorithms process the information contained in lists $\boldsymbol{A}$ and $\boldsymbol{B}$ and optimise the search process of the circuit.

6.1 Global circuit simulation

In this process, the inclusion and order of evaluation of the different cells in the circuit is controlled using the $\boldsymbol{A}$ and $\boldsymbol{B}$ interconnection lists; in this way, it is possible to perform successive searches in all the cells until a stable state is found. The first task is to identify the circuit, in terms of the connections ascribed to each of the cells that composite it, as a combinational or sequential circuit. Thus a circuit is identified as combinational when the entry lists for all cells are circuit input lines (primary lines) or secondary lines connected to preceding cell outputs; otherwise the circuit is identified as sequential. To simulate combinational circuits it is not necessary to create lists $\boldsymbol{A}$ and $\boldsymbol{B}$, as their cells can only be primary or secondary; the absence of feedback lines between cells, therefore, means that the simulation process is converted

into a simple simulation of all the cells in the circuit. The simulation process for sequential circuits is more complex, as it will usually be necessary to evaluate some cells more than once.

For a fixed state in their input lines, the simulation of sequential circuits is carried out in three different phases or approaches. The aim of these approaches is to find a stable state; this occurs when no output has changed its state compared with that obtained in the previous approach. This process is performed using the procedure *CircuitGlobalSimulation*, given in List 1 below. In the first approach, the state of all cells is set up according to the new values at the circuit inputs; this operation is performed by evaluating the cells, and those with feedback lines will, therefore, not be set up correctly. The second approach begins when the first has not found a stable state. First, the cells that have changed the logic state of their outputs during the previous approach are identified; then, from the ***A*** lists of these cells, a new list ***A**** of all the selected cells is created, and each of these cells is evaluated individually. When a stable situation is not reached in the first two stages, a recursive procedure constituting the third phase begins. The procedure followed is similar to that described in the second phase. First, the output nodes of the cells that have changed their logic state in the previous approach are identified. Next, from the ***B*** lists of these nodes, a new list ***B**** is created, which includes the cells to be evaluated. After each selected cell has been evaluated individually, the nodes that have undergone a change are identified, creating a new list from the ***B*** lists of these nodes so that the selected cells may be evaluated again. This process is repeated until either a stable state is found or the number of iterations exceeds a set value (the iteration parameter). If the number of iterations is exceeded, the circuit will be identified as unstable and the simulation process will end.

List 1

```
procedure CircuitGlobalSimulation ( ) {
  /* FIRST APPROACH */
  forall (cell j) {
    CellSimulation (j);  /* simulation of the cell j */
  }
  if (circuit state = STABLE) {
    return;  /* stable state of the circuit is found */
  }
  /* SECOND APPROACH */
  forall (cell j) {
    if (∃output ∈ cell j) has changed) {
      forall (cell i ∈ list A of this output) {
        to add cell i to the list A*;
      }
    }
  }
  forall (cell i ∈ list A*) {  /* simulation of list A* cells */
    CellSimulation (i);
  }
  if (circuit state = STABLE) {
    return;  /* stable state of the circuit is found */
  }
  /* THIRD APPROACH */
  while (k < iteration) {  /* iterative process */
    forall (cell j) {
      if ((∃output ∈ cell j) has changed) {
        forall (cell i ∈ list B of this output) {
          to add cell i to the list B*;
        }
      }
    }
    forall (cell i ∈ list B*) {  /* simulation of list B* cells */
      CellSimulation (i);
    }
    if (circuit state = STABLE)
      return;  /* stable state of the circuit is found */
    else
      k = k + 1;
  }
  /* output of the previous loop indicates unstable state */
  UNSTABLE CIRCUIT. END OF SIMULATION;
}
```

6.2 Cell simulation

Cell simulation is an iterative process of simulations of level modules whose object is to find a stable state; the process is performed by the procedure *CellSimulation*, summarised in List 2. A different evaluation method is applied according to cell type: primary, secondary, mixed or complete. Thus primary and secondary cells only require simulation of the level module; mixed and complete cells, however, require more complex treatment owing to the existence of internal lines. The search for a stable state in the cells belongs to the latter group is performed using an iterative process, whose limit is fixed by the number of internal lines of the cell; once this limit is passed, it will be identified as unstable.

List 2

```
procedure CellSimulation (J) {
  if (cell J ∈ {Primary, Secondary}) {
    switch (type of level module of cell J) {
      case "NMOS":
        LevelModuleSimulation ( ) for NMOS level module;
      case "CMOS":
        LevelModuleSimulation ( ) for CMOS level module;
      case "pass logic":
        LevelModuleSimulation ( ) for pass logic level module;
    }
    return;  /* end of procedure */
  }
```

```
if (cell J ∈ {mixed, complete}) {
  while (k ≤ internal lines of cell J) {   /* iterative process */
    switch (type of level module of the cell J) {
      case "NMOS":
        LevelModuleSimulation ( ) for NMOS level module;
      case "CMOS":
        LevelModuleSimulation ( ) for CMOS level module;
      case "pass logic":
        LevelModuleSimulation ( ) for pass logic level module;
    }
    if (circuit state = STABLE) {
    return;  /* end of procedure */
    }
    k = k + 1;
  }
}
/* output of the previous loop indicates unstable state */
UNSTABLE CIRCUIT. END OF SIMULATION;
}
```

6.3 Level module simulation

Simulation of the level module constitutes the lowest level in the circuit simulation hierarchy, and its aim is to explore the cell topology. This process involves the search for conduction paths through the level modules by means of algorithms based on the ***KOR*** transformation. In CMOS and pass transistor levels modules, conflicting situations may arise at their outputs. In the former, memory states occur when there are no conduction paths in the two submodules, and conduction states occur when conduction paths exist in both submodules at the same time. In pass-transistor level modules, the conflict arises when an output node is connected to two or more input nodes with different logic states; such a situation gives rise to logic indetermination (inconsistent fault) not considered in the simulator.

A description will now be given of the simulation algorithm for a NMOS level module. This algorithm will later be extended so that CMOS and pass-transistor modules can be dealt with, and the modifications needed to solve the conflicting situations arising in each case will be introduced.

6.3.1 Simulation of NMOS level modules

One or several complex NMOS gates can be dealt with by means of the level module whose general plan is shown in Fig. 10; this graphic representation is similar to that used in the basic cell matrix model. The lowest horizontal branch is connected to the reference terminal and, in the uppermost horizontal branch, the output nodes, each of which should be connected to a load element, are defined. Two vector sets are defined in this structure: the ***GL*** set, which represents the conduction states for transistors, and the ***LL*** set, which represents the horizontal connection topology between transistors. The i element of ***GL***, $\boldsymbol{GL}[i]$, defines the state of the transistors in row i. The j bit of $\boldsymbol{GL}[i]$, $\boldsymbol{GL}[i]_j$, represents the logic state of the transistor i, j; the pair i, j indicates its relative placement within the matrix structure. $\boldsymbol{GL}[i]_j = 1$ indicates that transistor i, j is in a conduction state (on), and $\boldsymbol{GL}[i]_j = 0$ indicates a cutoff state (off). Horizontal connections are dealt with in the same way. The i vector of ***LL***, $\boldsymbol{LL}[i]$, represents the state of the horizontal connection branches existing in row i. The j bit of i, $\boldsymbol{LL}[i]_j$, indicates the presence ($\boldsymbol{LL}[i]_j = 1$) or the absence ($\boldsymbol{LL}[i]_j = 0$) of horizontal branches situated in position i, j. Vertical connections are made using virtual transistors which are set permanently in a conduction state (on) or a cutoff state (off). Thus the nonexistence of the transistor i, j in the level module structure is equivalent to considering $\boldsymbol{GL}[i]_j = 0$ permanently; on the other hand, a transistor in a permanent conduction state (vertical connection branch) implies considering $\boldsymbol{GL}[i]_j = 1$ to model a short-circuit between its drain and its source.

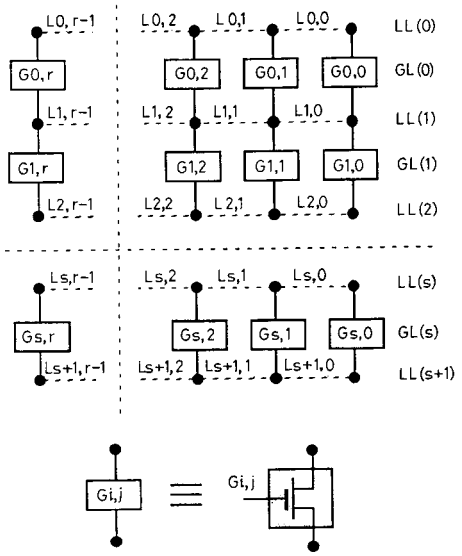

Fig. 10 *Structure of an NMOS level module*

Horizontal connections in each level module can be dealt with in a simplified way using the concepts of independent terminal and equipotential terminal:

(*a*) A terminal of a transistor is said to be *independent* (IT) if it has no horizontal connections ascribed to it. That is, the transistor drain i, j is defined as an independent terminal, and is represented by IT $\{i, j\}$ if $\boldsymbol{LL}[i]_{j-1} = \boldsymbol{LL}[i]_j = 0$.

(*b*) A set of transistor terminals connected by contiguous horizontal connections is called an *equipotential* terminal (*ET*). That is, the transistor terminals i, j, $(i, j + 1)$,

..., $(i, j+k)$ constitute an equipotential terminal and are represented by ET$\{i, j-k\}$ if $\boldsymbol{LL}[i]_j = \boldsymbol{LL}[i]_{j+1} = \cdots = \boldsymbol{LL}[i]_{j+k-1} = 1$.

Output nodes are obtained by identifying the ITs and ETs defined by the horizontal connection branches in row zero; output nodes are built in an orderly way from right to left along the zero row transistor drains.

The level module exploration process involves searching for conduction paths between the reference terminal and the output node or nodes. The algorithm implemented in based on the previously described ***KOR*** transformation. In this search process, an auxiliary variable termed 'state' is used to represent the state of the drains and sources of the transistors in a row. Each bit of this variable is associated with a transistor terminal; state_j (bit situated in state position $j+1$) represents the state of the transistor terminal situated in column j. The least significant bit corresponds to the transistor terminal situated on the extreme right of the row, represented in state_0 (bit situated in state position 1). With this notation, $\text{state}_j = 1$ indicates the presence of a conduction path between the j transistor and the reference terminal; $\text{state}_j = 0$ indicates the opposite.

The conduction path search in the level module is carried out through the simultaneous search of each row of transistors and its corresponding row of horizontal connections, a process which is repeated until all rows have been evaluated. The individual exploration of rows is based on the following operations:

(*a*) *STEP 1:* An exploration of the conduction paths between the sources and drains of a row i of transistors. To do this, the operation AND is carried out at bit level between the state variable and the **GL** vector corresponding to this row:

state = state AND $\boldsymbol{GL}[i]$;

(*b*) *STEP 2:* An exploration of the conduction paths through the connection branches of row i. This process is converted into a search of the conduction paths between the drains of the transistors in row i and the sources of the transistors in row $i-1$, through the ETs existing in the common horizontal connections. If, given that $ET\{i, j-k\}$, $\exists t$, $t \in (j, j+1, \ldots, j+k)$ so that $\text{state}_t = 1$, then $\text{state}_j = \text{state}_{j+1} = \cdots = \text{state}_{j+k} = 1$; this operation must be extended to all the ETs in this row. This step is equivalent to applying the ***KOR*** transformation to a row of transistors and their respective horizontal connection branches.

The procedure *MKOR* summarised in List 3 performs the exploration of the submodules from a knowledge of their dimensions $(s+1) \times (r+1)$, for example. First, the $r+1$ first bits of the state variable are initialised and the auxiliary variable k is identified with the number of rows in the level module. Next, STEP 1 and STEP 2 are performed on the rows of transistors from $k = s$ to $k = 1$. Finally, STEP 1 is carried out on the row of transistors $k = 0$ to determine the state of their drains. In the final state value, if $\exists j$, $j = 1, \ldots, r+1$, so that $\text{state}_j = 1$, then at least one conduction path exists between the drain terminal of the j transistor in the zero row and the reference terminal. This result is equivalent to finding a unitary column at $\boldsymbol{M}^*$ when ***KOR*** procedure is applied. This search process is interrupted when state = 0 is detected at any intermediate stage, in which case it can be guaranteed that no conduction paths exist.

List 3

```
procedure MKOR () {
  state = 1_{r+1} ··· 11;  /* initialisation of the state value */
  k = s;  /* initialisation of k with row number of the level module */
  while (k ≥ 1) {
    STEP 1;  /* exploration of row k of transistors */
    STEP 2;  /* exploration of row k of horizontal connection branches */
    if (state = 0) {
      There are no conduction paths in the level module;
      return;  /* end of the procedure */
      }
    k = k − 1;
    }
  STEP 1  /* last process; exploration row 0 transistors */
  StateOutput ();  /* state of the output node or nodes */
  }
```

Once the procedure *MKOR* has ended, the state of the output nodes is determined by executing procedure *StateOutput*, summarised in List 4. The presence of a conduction path between an output node defined by $IT\{0, t\}$ and the reference terminal is reflected in state. If $\text{state}_t = 1$, such a path exists and the output takes the low value (0); if, on the other hand, $\text{state}_t = 0$, there is no such path and the output takes the high value (1) since, as assumed earlier, there is a fault-free load transistor at each output. For an output node defined by $ET\{i, j-k\}$, if $\exists t$, $t \in \{j, \ldots, j+k\}$, such that $\text{state}_t = 1$, then the output will take the low value (0); otherwise, the output will take the high value (1). Fig. 11 shows an example of the NMOS level module search process.

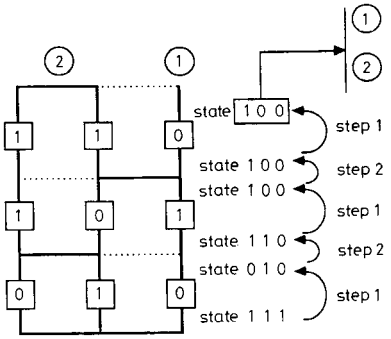

Fig. 11 *Example of the NMOS level module search process*
① Output 1 identified by IT {0, 0}; $\text{state}_0 = 0 \rightarrow$ output 1 is *high*
② Output 2 identified by ET {0, 1-2}; $\exists t \in \{1, 2\}/\text{state}_t = 1 \rightarrow$ output 2 is *low*

List 4

```
procedure StateOutput () {
  forall (∀output node j ∈ level module)}
    if (output node j is IT{0, k}) {/* independent terminal */
      if (state_t = 1)
        output low or 0;   /* conduction path exists */
      else
        output high or 1;   /* no conduction path */
    }
    else {
      if (output node j is ET{0, j − k}) {/* equipotential terminal */
        if (∃t(t ∈ {j, ..., j + k})/state_t = 1)
          output low or 0;   /* conduction path exists */
        else
          output high or 1;   /* non-conduction path */
      }
    }
  }
}
```

6.3.2 Simulation of CMOS level modules

The level module for CMOS cells consists of two submodules, as shown in Fig. 12: one is of type *P*, situated in the upper part, and the other of type *N*, situated in the lower part; both submodules are similar in structure to the NMOS level module. The outputs of this module are obtained by grouping pairs of output nodes of the two submodules. The simulation process is applied independently to each submodule, using the procedures described above. Once the presence or absence of conduction paths has been confirmed, the state of each of the outputs is determined according to the following criteria:

(i) A path exists in submodule-*P* but not in submodule-*N* (outut high or 1), or vice versa (output low or 0). The level module functions correctly and the output will take one of the logic states, depending on the submodule that conducts.

(ii) No paths exist in either submodule (memory state). A memory state arises when there is electrical isolation between the output and the corresponding supply terminals; this type of situation is generally solved by maintaining the previous logic state.

(iii) Both submodules conduct simultaneously (analogue state); the output state will be set by the dominant submodule.

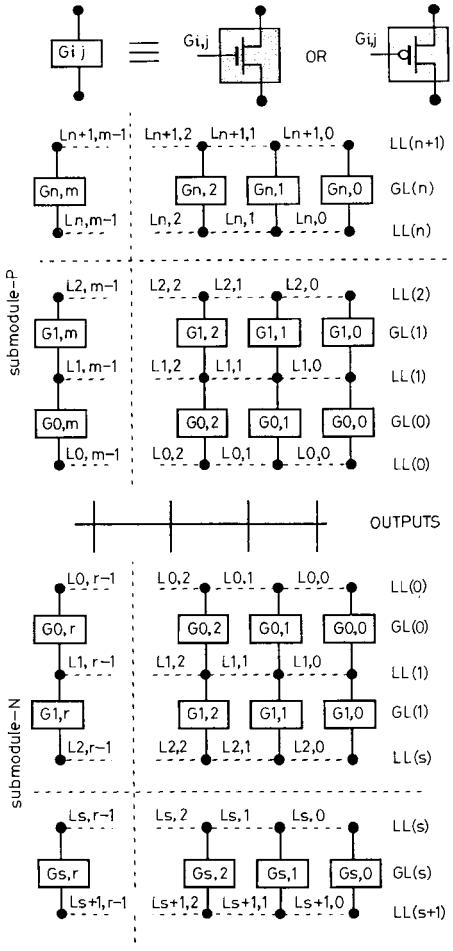

Fig. 12 *CMOS level module*

To solve memory and analogue states, the following decision parameters are used:

(i) mem1_0. For an output which is in a memory state, this parameter indicates the maximum time this node can remain in the high state; when this limit is exceeded, it changes to the low state. This time is measured by counting the number of test patterns applied to the circuit.

(ii) mem0_1. For an output which is in a memory state, this parameter indicates the maximum time this node can remain in the low state; when this limit is exceeded, it changes to the high state.

(iii) domp_n: This parameter indicates the dominant submodule, which may be the *P* submodule or the *N* submodule, or the submodule where the fault is located.

6.3.3 Simulation of pass-logic level modules

The structure of a pass-logic level module is shown in Fig. 13. As can be seen, the basic structure of the other modules is maintained, except that in this case the basic element is the CMOS pass transistor and there are two kinds of node: input nodes and output nodes. Input nodes are defined as the ITs and/or ETs ascribed to the lowest horizontal connection branches, and output nodes are similarly defined as the uppermost horizontal connec-

tion branches. In the search process three sets of vectors are used: set ***GL***, which represents the conduction states of the N transistors; set ***FL***, representing the P transistor conduction states; and set ***LL***, representing the horizontal connection branch topology. The same structure can also be used for NMOS pass transistor logic; this only requires the P transistors being maintained in a cutoff state.

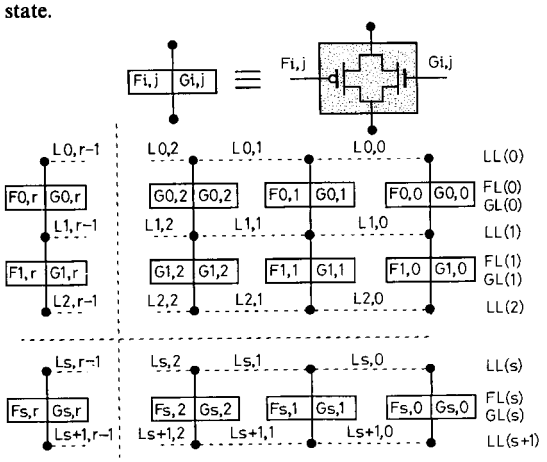

Fig. 13 *Pass logic level module*

The evaluation algorithm for these modules is similar to that previously described for CMOS and NMOS modules, but differs in three ways: the search process is performed from output nodes towards input nodes; it confirms the presence of conduction paths between the outputs and each of the associated inputs; and the logic state of each output node is obtained from the corresponding states of its inputs according to the following criteria:

(i) When there are no conduction paths, the previous logic state is maintained.

(ii) When there is a single path, the output node will take the logic state of the input node connected by this path.

(iii) When there are several input nodes with the same logic value connected to the same output node, the output node will take the same logic state as these inputs; if, however, the inputs connected to the output node do not have the same logic value, an inconsistent fault is produced which is not included in the proposed fault model.

7 Fault insertion

Faults affect the structure of the cells and level modules, so that fault insertion is performed by modification of the ***GL*** and ***LL*** vectors, depending on the nature of the fault. Thus transistor faults affect the ***GL*** vectors, whereas shorts and breaks modify the ***GL*** or the ***LL*** vectors, according to whether it is the horizontal or the vertical connection branch that is affected. Cell input and output line faults involve different treatment: the former are modelled as multiple transistor faults and the latter affect the variable representing the state of the cell outputs.

The insertion of a stuck-open fault (stuck-on) in a transistor situated in position i, j is performed by maintaining $\boldsymbol{GL}[i]_j = 0$ ($\boldsymbol{GL}[i]_j = 1$) during the whole simulation process. Shorts and breaks in the level module affect the horizontal (***LL***) and vertical (***GL***) connection branches. The insertion of a line-break is effected by nullifying a connection branch and a short-circuit is introduced by creating a new branch. A fault in a cell input line is transmitted to the transistors controlled by this line. Thus, a stuck-at-1 (stuck-at-0) type fault is modelled by setting all the cell's N transistors associated to this line at On (Off) and its P transistors at Off (On), and maintaining this state during its simulation. A fault in an output terminal modifies the variable that represents its state by maintaining the value 0 or 1, according to the fault type.

8 Haar and Walsh spectral analysis for digital signals

Let $f(i)$ be the sequence of length $N = 2^a$ formed from a Boolean function $f(x) = f(x_{a-1}, x_{a-2}, \ldots, x_2, x_1, x_0)$ of a variables obtained by changing the pair of values $\{0, 1\}$ for $\{1, -1\}$ for the function values and for the variables, with index i being defined by the expression

$$i = 2^{a-2}(1 - x_{a-1}) + 2^{a-3}(1 - x_{a-2}) + \cdots + (1 - x_1) + \tfrac{1}{2}(1 - x_0) \quad (3)$$

Let $\{H_k\}$ be the set of discrete Haar functions of N points quantified over $\{-1, 0, 1\}$, as given by the expressions

$$\begin{aligned} H_0(i) &= 1 \\ H_k(i) = H_{p,n}(i) &= +1, \frac{nN}{2^p} \leqslant i < \frac{nN}{2^p} + \frac{N}{2^{p+1}} \\ &= -1, \frac{nN}{2^p} + \frac{N}{2^{p+1}} \leqslant i < \frac{n+1}{2^p} N \\ &= 0, \text{ in all other cases} \end{aligned} \quad (4)$$

where

$$0 \leqslant p \leqslant (\log_2 N) - 1,\ 0 \leqslant n \leqslant 2^p - 1,\ k = n + 2^p,\ 0 \leqslant i \leqslant N - 1$$

With the previously established conditions, the discrete Haar transform of sequence $f(i)$ is defined as the real number sequence D_k, calculated using

$$D_k = \sum_{i=0}^{N-1} H_k(i) f(i), \quad 0 \leqslant k \leqslant N - 1 \quad (5)$$

The number of operations needed to find D_k can be considerably reduced if the sequences of length $N/2$ are considered:

$$f_1(i) = f(i);$$

$$f_2(i) = f\left(i + \frac{N}{2}\right); \quad 0 \leqslant i \leqslant \frac{N}{2} - 1 \quad (6)$$

and a strategy similar to the decimation in frequency is applied to eqn. 5, from which the following is obtained:

$$D_k = [D_0^1 + (-1)^{k+2} D_0^1]; \quad k = 0, 1 \quad (7)$$

and

$$D_{p,n''} = \sum_{i=0}^{N'-1} H_{p',n'}(i) f_r(i) = D^r_{p',n'} \quad (8)$$

where

$$r = 1, 2;\ N' = \frac{N}{2};\ p' = p - 1$$

$$1 \leqslant p \leqslant (\log_2 N) - 1;\ 0 \leqslant n' \leqslant 2^{p'} - 1;\ n'' = n' + 2^{p'} \log_2 r \quad (9)$$

Eqn. 7 indicates that the first two coefficients of the Haar spectrum of $f(i)$ are obtained by the addition and subtraction of the first coefficient of $f_1(i)$, D_0^1, with the first coefficient of $f_2(i)$, D_0^1. The remaining $N-2$ coefficients of $f(i)$ are obtained directly from the remaining coefficients of $f_1(i)$ and $f_2(i)$, as indicated in eqns. 8 and 9. This process can be repeated as many times as possible, until the basic sequences are of length 2, $(N-1)$ being the total number of additions and subtractions that have to be performed. The iterative application of eqns. 7 and 8 constitutes the fast Haar transform.

Calculation of the complete spectrum of $f(i)$ means generating a complete test set; however, eqn. 8 indicates that it is possible to determine groups of spectrum coefficients without the need to generate an exhaustive test. Thus, to determine the set of spectrum ocefficients $C_{q,m}$ defined by

$$C_{q,m} = \{D_{p,n}\}, \quad q > 0,\ q \leqslant p \leqslant (\log_2 N) - 1$$
$$2^{i-q}m \leqslant n \leqslant 2^{i-q}(m+1) - 1 \qquad (10)$$

it will be necessary to apply a test set of length 2^{a-q}, beginning at $i = m^{2a-q}$ after the circuit has been initialised.

Given the set of discrete Walsh functions, $\{W\}$, defined from the variables of f (after changing the pair of values $\{0, 1\}$ for $\{1, -1\}$) by the expressions

$$W_0 = 1$$
$$W_{j+1}(i) = x_j, \quad \forall j \in (0, 1, \ldots, a-1)$$
$$W_{j+1,k+1}(i) = x_j x_k, \quad \forall j, k \in (0, 1, \ldots, a-1), j < k$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$W_{1,2,\ldots,a}(i) = x_0 x_1 \cdots x_{a-1} \qquad (11)$$

the discrete Walsh transform of $f(i)$ is defined as the sequence of length N obtained from

$$R_{j,k,\ldots,m} = \sum_{i=0}^{N-1} W_{j,k,\ldots,m}(i) f(i),$$
$$j < k < \cdots < m \in (1, 2, \ldots, a) \qquad (12)$$

In this case, owing to the structure of the base set (eqn. 11), it is not possible to obtain any spectrum coefficient (eqn. 12) without performing an exhaustive exploration of the circuit.

Spectral analysis includes the calculation of the Walsh and/or Haar (depending on the case) spectra of the circuit responses obtained for each of the fault situations simulated, and the comparative analysis of these spectra. This process is applied individually to each output, and the fault-free circuit spectra are compared with those for each of the fault situations. Spectral analysis is applied in two phases:

(i) *Identification of equivalent faults.* Equivalent faults are those faults which, having different origins and/or different placements in the circuit, generate the same state in the output; these sets implicitly include topologically equivalent faults (TEFs). From these faults, coverage can be calculated. Coverage indicates the fraction of faults detected, that is, the number of faults detected in terms of the total number of faults dealt with. The sets of nonequivalent faults are also obtained and used as a measurement of diagnosis.

(ii) To find *the minimum set of spectrum coefficients* which allows univocal identification for each of the nonequivalent faults considered. To perform this selection, the spectrum coefficients are first ordered according to their diagnosis value, placing the best results first. This ensures that the coefficients selected are the most appropriate for the circuit type, test sequence and fault list under consideration.

When the spectral analysis process has ended, a list of detected faults, a list of nonequivalent faults and the spectral signature for each of the nonequivalent equivalent faults set are all obtained. All this information will later be used to test the circuit.

The processes of circuit simulation and spectral analysis for each of the selected fault situations result in the generation of two sets of identifiers: $\{F\}$ and $\{S^i\}$. The $\{F\}$ set composed of $m+1$ identifiers $\{F_0, F_1, F_2, \ldots, F_m\}$ represents each of the faults dealt with in the simulation process; F_0 is the identifier of the fault-free circuit. Each identifier $F_1 \in \{F\}$ contains the fault type, its placement in the circuit and the list of TEFs for this fault. The $\{S^i\}$ set contains the spectra obtained from the i output of the circuit and is composed of $(m+1)$ identifiers, $\{S_0^i, S_1^i, S_2^i, \ldots, S_m^i\}$; S_0^i is the spectrum of the i output of the fault-free circuit.

The process of identifying equivalent faults is applied to each circuit output operating on the sets $\{F\}$ and $\{S^i\}$. To do this, the set $\{FS^i\}$ is created: $\{FS^i\} = \{\{F_j, S_j^i\}\}$, for $j = 0, \ldots, m$, and a comparison is made of the set of $m+1$ spectra so that the fault identifiers F_j which have the same spectrum can be grouped together. The procedure *IdentifyinEquivFault*, summarised in List 5, obtains this identification. It uses three variables: m^*, which indicates the current number of spectra in the list, and p and q, which select spectra for their comparison.

List 5

```
procedure IdentifyinEquivFault () {
  m* = m   /* initialises m* */
  foreach (p ∈ {0, ..., m* − 1}) {   /* selects first spectrum */
    foreach (q ∈ {p + 1, ..., m*) {   /* selects second spectrum */
      if (S_p^i = S_q^i) {   /* equivalent fault identified */
        F_q is added to the list {F_p, S_p^i} → {F_q, F_p, S_p^i};
        {F_p, S_p^i} is removed from {FS^i} set;
        m* = m* − 1;   /* number spectra in list decreases */
      }
    }
  }
}
```

In this procedure, if two spectra are identical ($S_p^i = S_q^i$, $p < q$), then the F_q identifier is added to S_p^i giving $\{F_q, F_p, S_p^i\}$, with the pair $\{F_q, S_q^i\}$ then being taken from $\{FS^i\}$. At the end of the process the original $\{FS^i\}$ set of $m+1$ pairs of identifiers will contain $g(\leqslant m+1)$ subsets of nonequivalent fault identifiers, with their correspond-

ing spectra. With this information, fault coverage at output i is obtained using the following equation:

$$\text{Fault coverage} = 1 - \frac{niff - 1}{m}$$

where *niff* represents the number of identifiers associated with fault-free circuit spectrum and m is the total number of faults.

To determine the spectral signature of the g subsets of $\{FS^i\}$, the spectral coefficients are first reordered according to the number of faults that they diagnose, and then a minimum set of coefficients is selected to identify univocally the g subsets.

9 Testing the circuit

The fault simulation and spectral analysis processes, which are performed on a workstation, allow the selection of the spectral signatures appropriate for carrying out the test process of the circuit prototypes. To apply the test sequences to the prototypes and to collect their responses, a logic analysis system (LAS) (Hewlett Packard HP16500A) is used as an interface between the workstation and the circuit prototypes, as shown in Fig. 14.

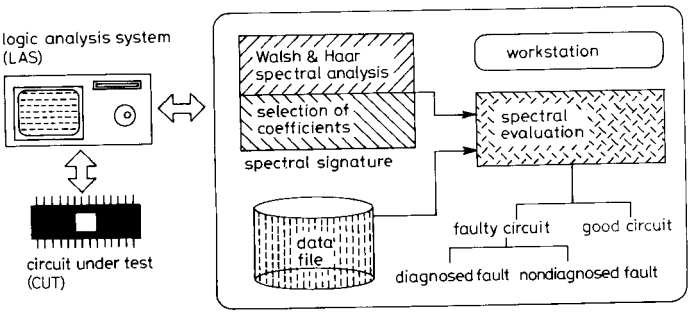

Fig. 14 *Testing the CUT*

The prototype testing process begins with the programming of the LAS from the workstation to ensure that the circuit is tested in the same conditions as those used in the simulation process; this programming includes assigning channels and labels, selecting probes, setting clock rates and loading the test vectors used in the simulation process, among others. When programming is finished, the LAS applies the electrical stimuli to the circuit and stores the responses, which are later read from the workstation, where their spectra are computed. Of these spectra, only those coefficients that constitute the spectral signature are used in the comparison with those stored in the data file during the simulation process; as a result of this process, the circuit is classified as fault-free or faulty, in which case the diagnosis results are also included.

The problems of aliasing, and/or the presence of faults not detected by the test sequence, may result in the erroneous identification of a faulty circuit as good. The phenomenon of aliasing is caused by loss of information, which is characteristic of any compression method; it arises when the fault-free and the faulty circuit have the same spectral signature. In our case, the spectral signature selection process used ensures that aliasing does not occur for the fault set and the selected test sequence.

10 Example of application

The CMOS serial adder shown in Fig. 15 was used. This circuit consists of a complete adder, shown in the upper part of the figure, and a master–slave D flip-flop to store the carry, in the lower part. The input test sequence used has a length of 32 test patterns applied synchronically with each clock cycle. A total of 240 simple faults were selected: 48 stuck-on and 48 stuck-open faults, 76 shorts and breaks, and 68 input and output line stuck-at faults; of this set, 64 faults (26.6%) were eliminated by fault collapsing. During the simulation process, 18 inconsistent and 6 unstable faults were identified.

The results in the SUM output for the 216 (240-18-6) faults finally considered were as follows:

(i) Groups of nonequivalent faults: 46 (21.3%).

(ii) Faults detected 191 (88.4%): 39 stuck-open (81.3%), 40 stuck-on (100%), 52 shorts and breaks (76.5%), 60 stuck-at line faults (100%). Fig. 16 shows the solution of the percentage of detected faults in terms of the number of test patterns applied during the 64 half-clock cycles of the test sequences.

(iii) Statistics of the spectral signature: Walsh coefficients selected: R_0 and R_5; Haar coefficients selected: H_0, H_5 and H_{22}.

11 Performance analysis

An evaluation of the cost in terms of CPU time was carried out for the fault collapsing, fault simulation and spectral analysis modules; there was no point in doing so for the rest, as there is a 'human interface'. Most computation time is taken up by fault simulation. One of the most important features of simulation tools is the execution speed, which allows a large number of faults to be dealt with; this is due to the use of the simple switch model for transistors and to the simplicity of the level module exploration process, in which rapid simulation algorithms have been used in order to handle a large number of faults. Table 1 gives the execution times for the simulation of a set of typical circuits using the software tools implemented on a SUN 3/260 (~4 MIP) workstation with a UNIX operating system. The table contains information about the number of transistors and cells in each circuit, the length of the test sequence used, the number of faults dealt with in each case, the number of collapsed faults, and the total time for the complete process.

With the chosen fault model, fault situations may arise that are difficult to deal with and the mechanisms developed to overcome these difficulties may, therefore, not be appropriate. In order to assess the model of the faulty

Table 1: Execution times for the simulation of a set of typical circuits

Circuits	NITR	NTR	NCELL	NFAULT	NCF	TIME
2-bit comparator	16	36	4	142	24	5, 8
4-bit multiplexer	64	28	3	112	18	10
4-state bidirectional parallel/serial input/output register	32	328	12	1019	218	1020
4-bit full-adder	512	148	5	736	186	3635
4-bit BCD full-adder	512	204	16	848	176	4075
4-bit parallel multiplier	256	384	16	1560	469	5010
Presettable 4-bit up-down counter with BCD-to-seven segment decoder	512	384	19	1618	429	12182
4-bit arithmetic logic unit	1024	300	11	1364	386	26542

NITP: Number of input test pattern. NTR: Number of transistors. NCELL: Number of cells. NFAULT: Number of faults. NCF: Number of collapsed faults. TIME: CPU time in s.

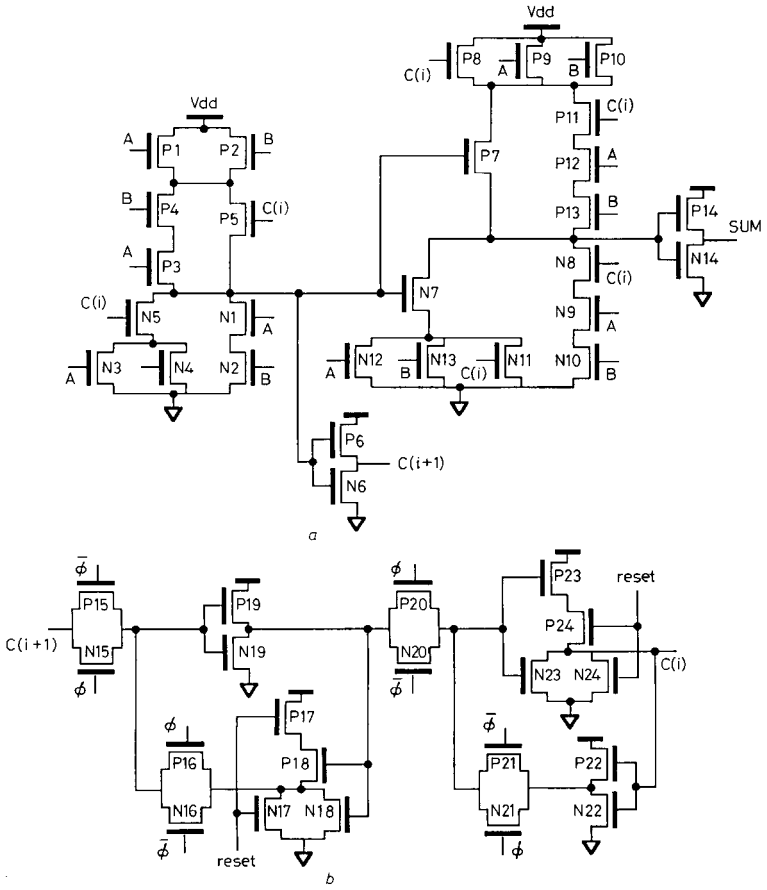

Fig. 15 *CMOS serial adder*
a Full adder *b* D flip-flop

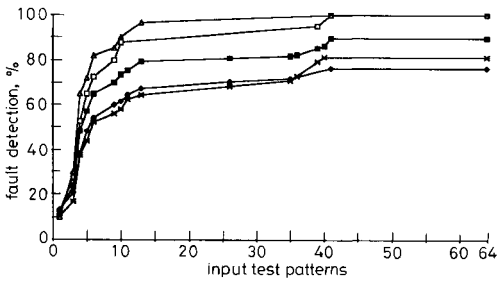

Fig. 16 *Fault detection versus number of input test patterns*
—×— Stuck-open —◆— Short/break
—□— Stuck-on —■— Total
—△— Stuck-at line

circuit's behaviour at logic level, several cases were studied and the results were compared with those obtained using a SPICE electrical simulator. In SPICE, stuck-on faults and shorts were modelled as resistive elements of 200 Ω, stuck-open faults as open circuits, and breaks as open lines. The comparative analysis showed that over 90% of the faults are dealt with correctly, which offers sufficient guarantee for the test environment to be applied.

12 Conclusions

A test environment has been presented for fault detection and diagnosis in MOS circuits. It consists of a set of software tools written in C under UNIX. These tools run on

a workstation and act on a logic analysis system connected to it for the physical testing of the circuits. The test process is carried out in two phases: in the first phase, simulation of the circuit faults is performed, as a result of which a dictionary of faults and the corresponding set of spectral signatures are obtained. To achieve this, a series of tasks is performed: circuit editing, fault collapsing at the transistor level, simple switch-level fault simulation, and spectral analysis of the response obtained by simulation. Circuit editing is carried out using transistor matrix modules, called level modules, which are appropriate for dealing with faults at the transistor level. Indeed, certain shorts and breaks faults existing in the circuit can easily be handled by these modules. This feature is very important, as these are the fault types that occur most often in MOS circuits. The fault-collapsing technique at the transistor level, which makes use of the process of the identification of topologically equivalent faults in the level module, allows a reduction in the number of faults to be simulated. Fault simulation deals with each of the preselected fault situations in the circuit following a hierarchical scheme that includes three operational levels: global simulation, cell simulation and level module simulation. The simulation algorithms are characterised by their low memory requirement and high processing speed, thus allowing a large number of faults to be dealt with in a relatively short computation time. Spectral analysis of the circuit's simulated response is performed with discrete Walsh and/or Haar transforms. The dictionary of faults contains a list of unstable faults, a list of inconsistent faults, a list of detected faults and a list of nonequivalent fault groups (diagnosis) as well as the spectral signature for each of the latter. To generate the spectral signatures, the spectrum coefficients most appropriate for diagnosis are chosen, which ensures that aliasing does not occur for the test sequence, circuit type and the fault list under consideration.

In the second phase, the physical process of testing the circuit is carried out. First, the logic analysis system is programmed to ensure that circuit testing is performed as foreseen in the simulation. Next, electrical stimuli are applied to the circuit under test and its responses are collected and sent to the workstation for its spectral signature to be computed. The result of the test is obtained by comparing this spectral signature with those obtained in the first phase; the circuit is classified as fault-free or faulty, in which case the diagnosis results are included.

13 References

1 HAYES, J.P.: 'Fault modeling', *IEEE Design & Test*, April 1985, pp. 88–95

2 FUJIWARA, H.: 'Logic testing and design for testability'. MIT Press, 1985

3 FANTINI, F., and MORANDI, C.: 'Failure modes and mechanisms for VLSI ICs — a review', *IEE Proc. G*, 1985, **1325**, pp. 78–81

4 FERGUSON, F.J., and SHEN, J.P.: 'A CMOS fault extractor for inductive fault analysis', *IEEE Trans. Computer-Aided Design*, 1988, **7**, pp. 1181–1194

5 GALIAY, J., CROUZET, Y., and VERGNIAULT, M.: 'Physical versus logical fault models MOS LSI circuits: impact on their testability', *IEEE Trans. Comput.*, 1980, **C-29**, pp. 527–531

6 BANERJEE, P., and ABRAHAM, J.A.: 'Characterization and testing of physical failures in MOS logic circuits', *IEEE Design & Test*, 1984, pp. 76–86

7 WADSACK, R.L.: 'Fault modeling and logic simulation of CMOS and MOS integrated circuits', *Bell System. Tech. J.*, 1978, **57**, pp. 1449–1474

8 BURGESS, N., DAMPER, R., TOTTON, K., and SHAW, S.: 'Physical faults in MOS circuits and their coverage by different fault models', *IEE Proc. E*, 1988, **135**, pp. 1–9

9 HAYES, J.P.: 'Fault modeling for digital MOS integrated circuits', *IEEE Trans.*, 1984, **CAD-3**, (3), pp. 200–207

10 HAYES, J.P.: 'An introduction to switch-level modeling', *IEEE Design & Test*, 1987, pp. 18–25

11 BRYAN, R.E.: 'A switch-level model and simulator for MOS digital systems', *IEEE Trans. Comput.*, 1984, **C-33**, pp. 160–177

12 ROBINSON, J.P., and SAXENA, N.R.: 'A unified view of test compression methods', *IEEE Trans. Comput.*, 1987, **C-36**, pp. 94–99

13 TEN-CHUAN HSIAO, and SETH, S.C.: 'An analysis of the use of Rademacher–Walsh spectrum in compact testing', *IEE Trans. Comput.*, 1984, **C-33**, pp. 934–937

14 SUSSKIND, A.K.: 'Testing by verifying Walsh coefficients', *IEEE Trans. Comput.*, 1983, **C-32**, pp. 198–201

15 MILLER, D.M., and MUZIO, J.C.: 'Spectral fault signatures for internally unate combinational networks', *IEEE Trans. Comput.*, 1983, **C-32**, pp. 1058–1062

16 MILLER, D.M., and MUZIO, J.C.: 'Spectral fault signatures for single stuck-at faults in combinational networks', *IEEE Trans. Comput.*, 1984, **C-33**, pp. 765–768

17 HURST, S.L., MILLER, D.M., and MUZIO, J.C.: 'Spectral techniques in digital logic' (Academic Press, 1985)

18 HURST, S.L: 'Use of linearisation and spectral techniques in input and output compaction testing of digital networks', *IEE Proc. E*, 1989, **136**, pp. 48–56

19 EL-ZIQ, Y.M., and SU, S.Y.H.: 'Fault diagnosis of MOS combinational networks', *IEEE Trans. Comput.*, 1982, **C-31**, pp. 129–139

20 RUIZ, G., BURÓN, A., and MICHELL, J.A.: 'NMOS & CMOS physical faults diagnosis by spectral techniques: simulation and associated tools'. IEEE Workshop on Languages for Automation, The Technical University of Vienna, August 26–27, 1987, pp. 191–194

21 RUIZ, G., MICHELL, J.A., and BURÓN, A.: 'Fault detection and diagnosis for MOS circuits from Haar & Walsh spectrum analysis: on the fault coverage of Haar reduced analysis'. Proc. 3rd International Workshop on Spectral Techniques, University of Dortmund, October 4–6, 1988, pp. 97–106

22 RUIZ, G., MICHELL, J.A., and BURÓN, A.: 'Diagnosis of stuck-open faults in CMOS by spectral techniques with pseudorandom excitation'. Proc. 7th IASTED International Symposium on Applied Informatics, Grindelwald, Switzerland, February 8–10, 1989, pp. 89–92

23 RUIZ, G., MICHELL, J.A., and BURÓN, A.: 'Automatic test vector set generation in the spectral testing of sequential circuits'. Proc. 40th ISMM International Symposium on Mini- and Microcomputers and their Applications, Lugano, Switzerland, June 1990, pp. 55–58

24 RUIZ, G., MICHELL, J.A., and BURÓN, A.: 'Fault spectral detection and diagnosis of MOS digital circuits at switch level', *in* MORAGA, C., and CREUTZBURG, R. (Eds.): 'Spectral techniques: theory and applications' (Springer-Verlag, 1991 (in press))