STUDIES ON ISOMORPHIC-REDUNDANCY AND TESTING OF NON-SCAN SEQUENTIAL CIRCUITS

DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE M. TECH.(COMPUTER SCIENCE) DEGREE

OF

INDIAN STATISTICAL INSTITUTE

BY

JAYADEEP DAS

1992

ACKNOWLEDGEMENTS

I am deeply indebted to my supervisor, Dr.Bhargab B.Bhattacharya for his invaluable guidance without which it would not have been possible to complete this dissertation.

I thank my friends Rajan Gangadharan, M.Suresh, Ashutosh K.Jha and Milind B.Kamble for their constant moral support and encouragement.

Jayadeep Das.

CONTENTS

PART I

ABSTRACT 1

1. INTRODUCTION 2

2. PRELIMINARIES 4

3. THE BASIC APPROACH 10

4. CONCLUSION 17

PART II

ABSTRACT 18

1. INTRODUCTION 19

2. PRELIMINARIES 21

3. COMBINATIONAL TEST GENERATION 22

4. FAULT-FREE STATE JUSTIFICATION 27

5. STATE DIFFERENTIATION 30

6. IMPLEMENTATION 33

7. CONCLUSION 35

REFERENCES 36

PART I

STUDIES ON ISOMORPHIC-REDUNDANCY

ABSTRACT.

Synthesizing a sequential circuit from the description of state transition graph (STG) involves the steps of state minimization, state assignment and logic optimization. One of the basic goals in synthesis of non-scan sequential circuit is to make the circuit fully testable. Test sequences for all single stuck-at faults in the synthesized machine can be derived using test generation algorithms on the combinational logic blocks of the machine. A circuit is said to be 100% testable if test vectors can be generated for all the lines in the circuit under the single stuck-at fault model. The presence of redundant faults prevent the circuit from being fully testable. There are two classes of redundant faults in a non-scan sequential circuit, namely, combinationally and sequentially redundant faults.

This part of the dissertation deals with the study of one special class of sequential redundant faults (SRFs), called isomorphic faults (Isomorph-SRF). Such a fault causes the STG of the faulty machine to be isomorphic to that of the fault-free machine. No input-output experiment can thus detect an isomorphic fault. An open question in this context is whether there exists a real sequential circuit that is reduced and combinationally irredundant, in which a single-stuck at fault creates an isomorphic faulty machine. This dissertation settles this open question which shows that such a machine indeed exists.

1

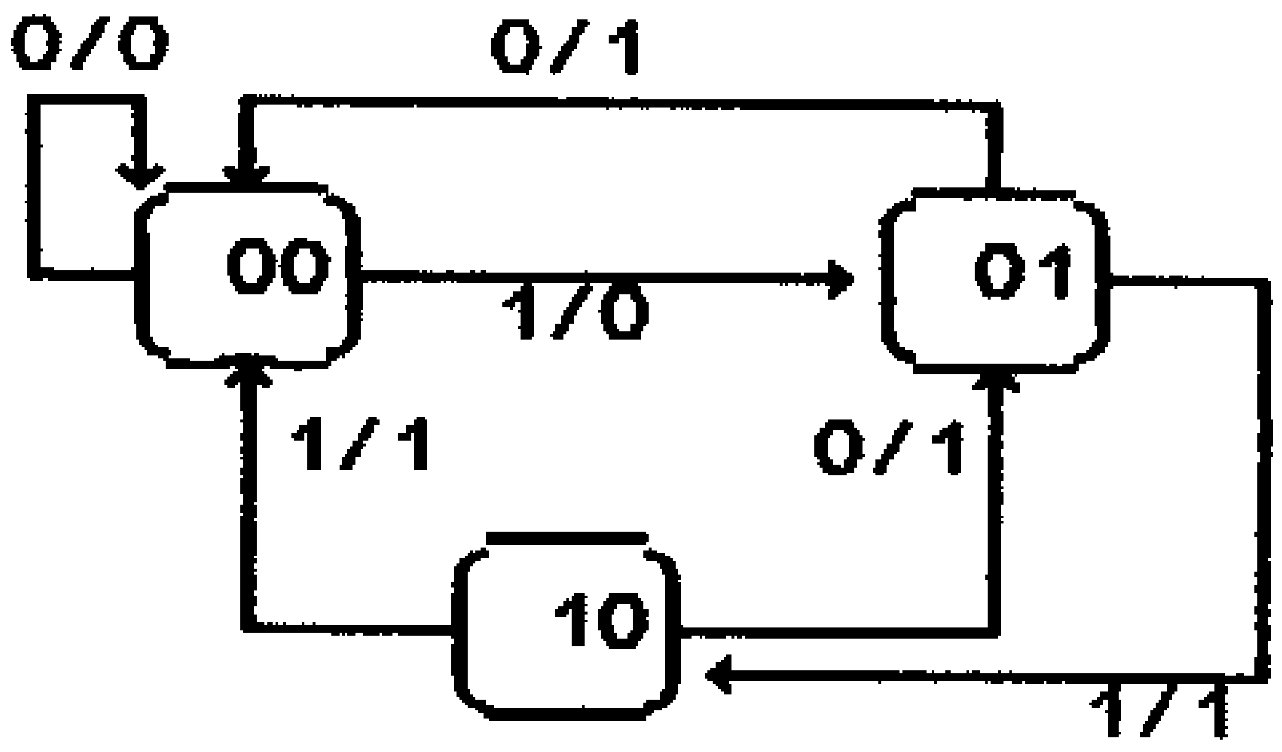
INTRODUCTION.

Combinationally redundant faults (CRFs) are due to the presence of lines in the logic circuit that do not contribute to the primary output or the next state functions. Replacement of these lines by constants do not change the functional behaviour of the combinational logic part of the sequential circuit. Sequentially redundant faults (SRF's), on the other hand, are related to the temporal characteristics of the sequential circuit. The faulty behaviour of the circuit in the case of a SRF, propagates to the next state lines, keeping the primary output unchanged. Hence, a SRF is a fault that cannot be detected by any input sequence and is not combinationally redundant.

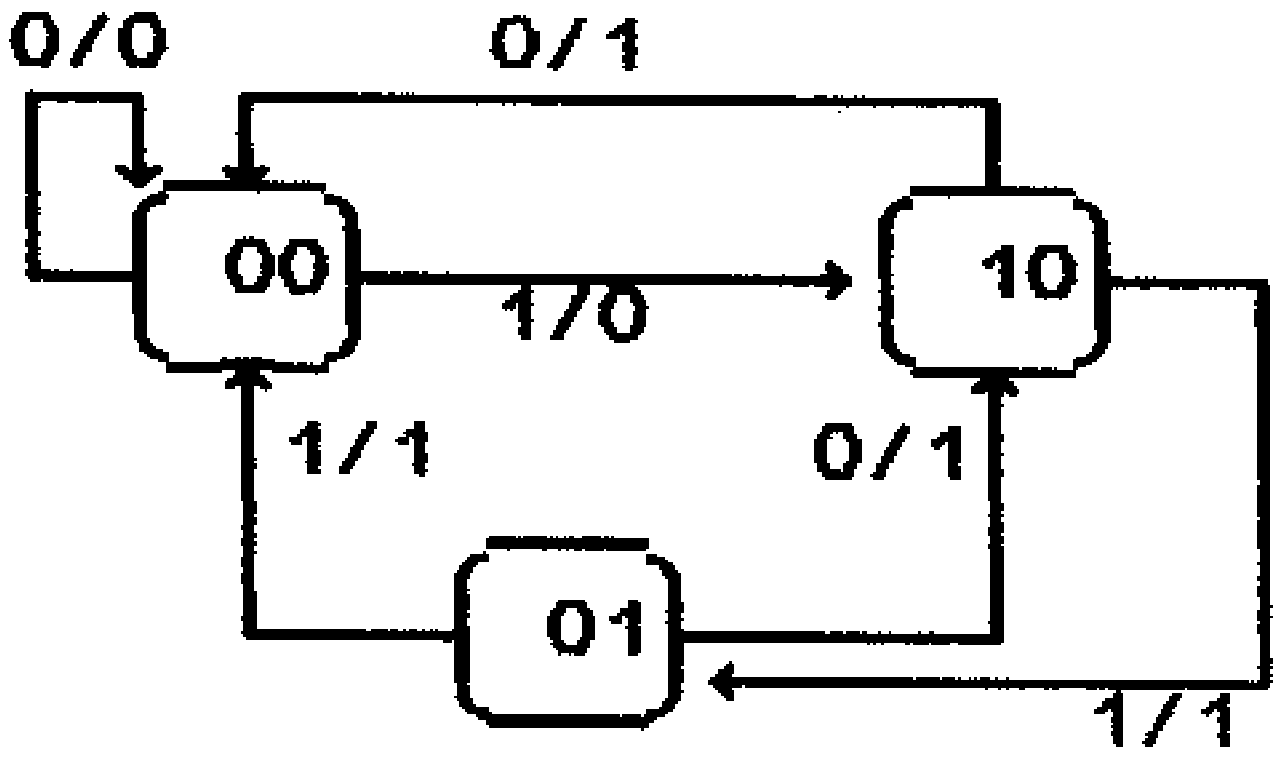
An isomorphic SRF transforms the original machine isomorphically, i.e., the faulty machine is equivalent to the good machine in the sense that the functional behaviour of both machines remain same, but the original machine gets transformed to a machine with a different encoding of states. Fig.1.1 and fig.1.2 show the STGs of a true machine and faulty machine respectively.

An isomorphic fault under single stuck-at fault possibly occurs very rarely. An open question in this area is the follwing :

" Does there exist a sequential machine which is reduced and combinationally irredundant, but transformable to an isomorphic faulty machine under a stuck-at fault ?." This part of the dissertation settles the above problem by exploring an interesting case, under which such an isomorph-SRF might occur.



TRUE MACHINE
Fig.1.1



FAULTY ISOMORPHIC MACHINE
Fig.1.2

2

PRELIMINARIES.

A State Transition Graph (STG) G is a 3-tuple (V, E, W(E)), where V is the set of vertices corresponding to the set of states S, E is the set of edges where each edge joins v_i to v_j if there is a primary input that causes the finite-state machine to evolve from state v_i to state v_j and W(E) is the set of labels attached to each edge, each label carrying the information of the value of the input that caused the transition and the values of the primary outputs corresponding to that transition. Each label is an ordered 4-tuple $\langle i, s, s', o \rangle$ where i is a minterm, over the primary inputs, s and s' are minterms over the state variable, called the fan-in and fan-out states respectively, and o is a minterm over the primary outputs.

An edge of a STG is said to be corrupted by a fault if either the the fan-out state or the output label of this edge is changed because of the existence of the fault.

Sequentilly redundant faults can be classified into three categories.

1) **Invalid state faults:** The fault does not corrupt any fan-out edge of a valid state in the STG, but does corrupt the fan-out edge of an invalid state.

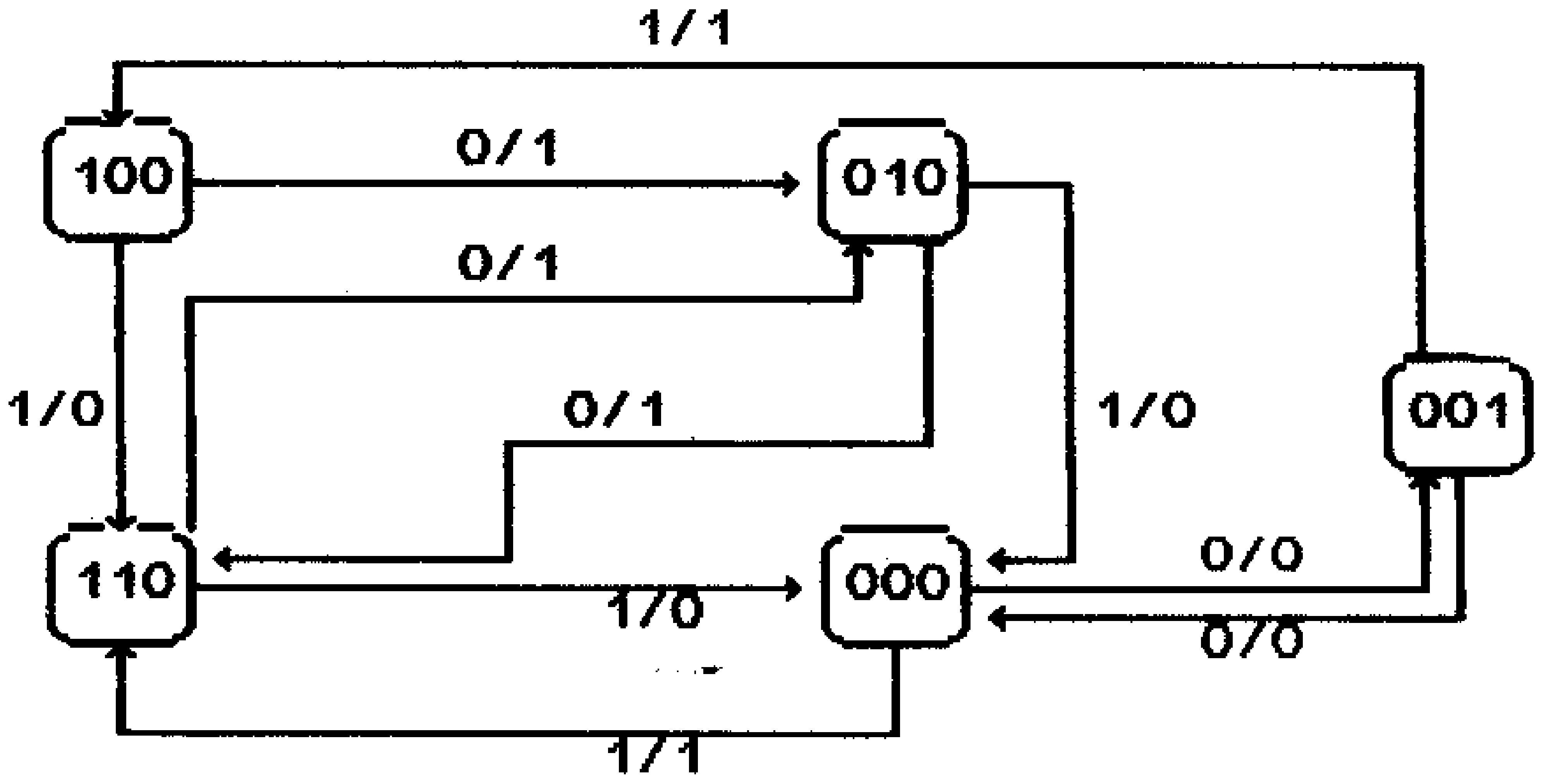
2) **Equivalent state faults:** The fault causes only interchange and/or creation of equivalent states in the state transition graph of the finite-state machine.

3) **Isomorphic faults:** The fault results in a faulty machine that is isomorphic (with a different encoding) to the original

fault-free machine.

Following examples illustrated in Fig 2.1 through Fig.2.5 will explain the SRFs.

The state transition graph of the finite-state machine is shown in Fig.2.1. The states 010 and 110 are equivalent. Fig.2.2 shows the logic implementation of the combinational part of the machine. The fault w1 (s-a-0) changes the original STG to the one shown in Fig.2.3. The corrupted edge is shown as a dotted line. Since 010 and 110 are equivalent states, the fault w1 causes an interchange of two equivalent states of the machine.The fault w2 (s-a-1) changes the machine to the one shown in Fig.2.4. The fault creates an extra state 111, which was originally an invalid state and equivalent to the true state 110. Therefore, the fault is sequentially redundant, ~~and is known as invalid state SRF.~~ Fig.2.5 shows the occurrence of an isomorph SRF, where the state codes 001 and 000 have been swapped.



ORIGINAL FINITE STATE MACHINE.
Fig.2.1.

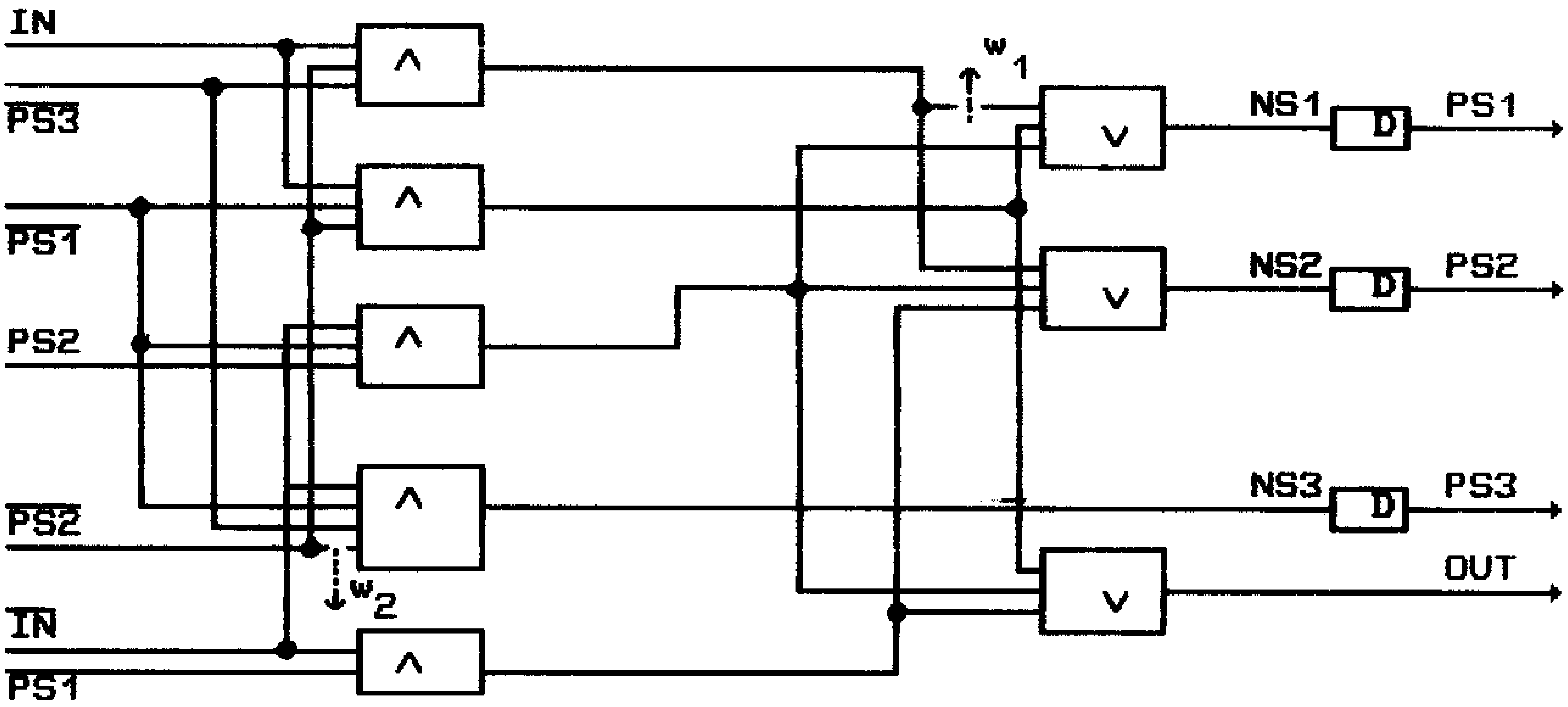


Fig.2.2. COMBINATIONAL LOGIC OF MACHINE OF FIG.2.1

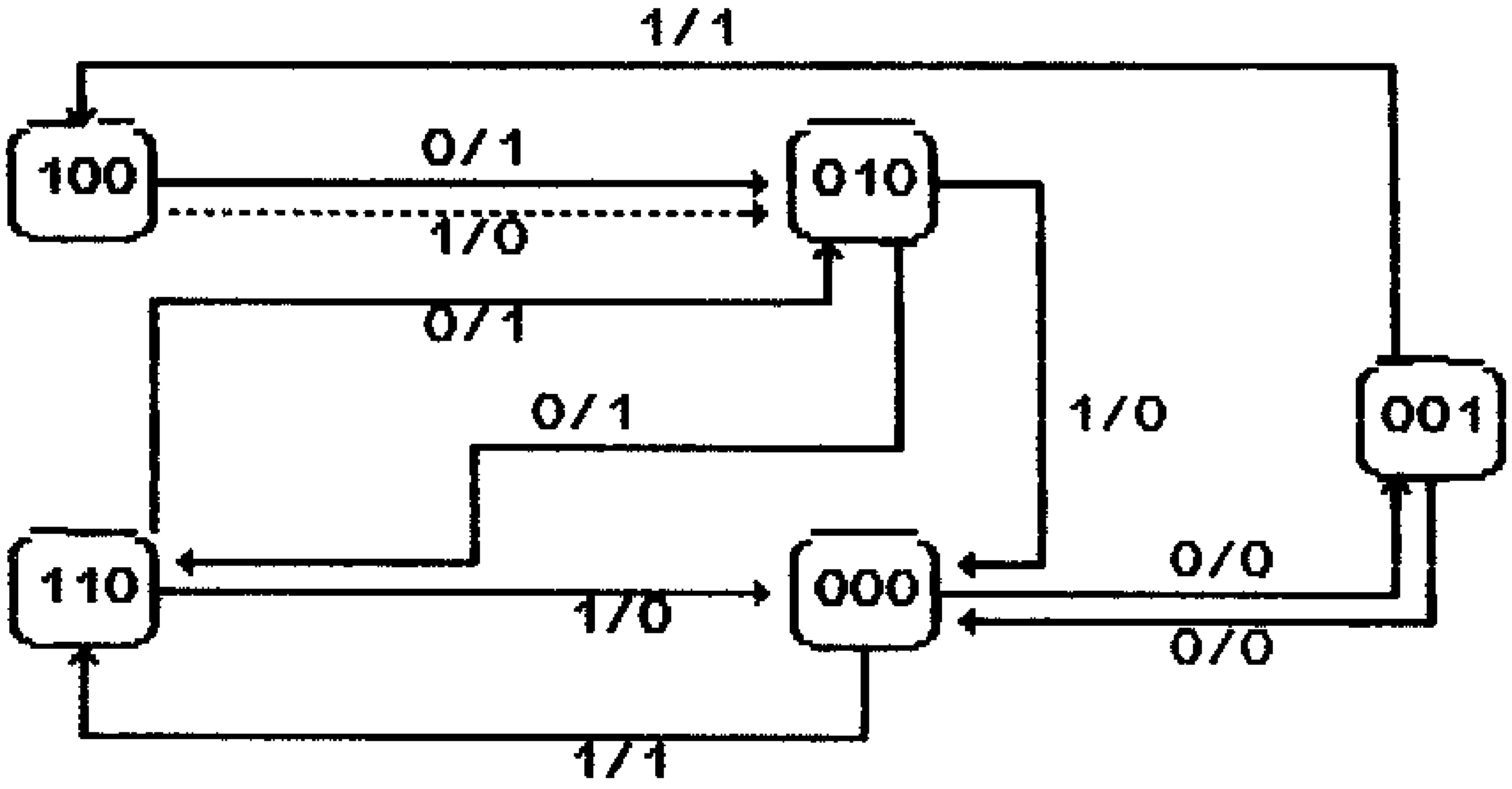


Fig.2.3. FAULTY MACHINE WITH w1 S-A-0 (EQUIVALENT SRF).

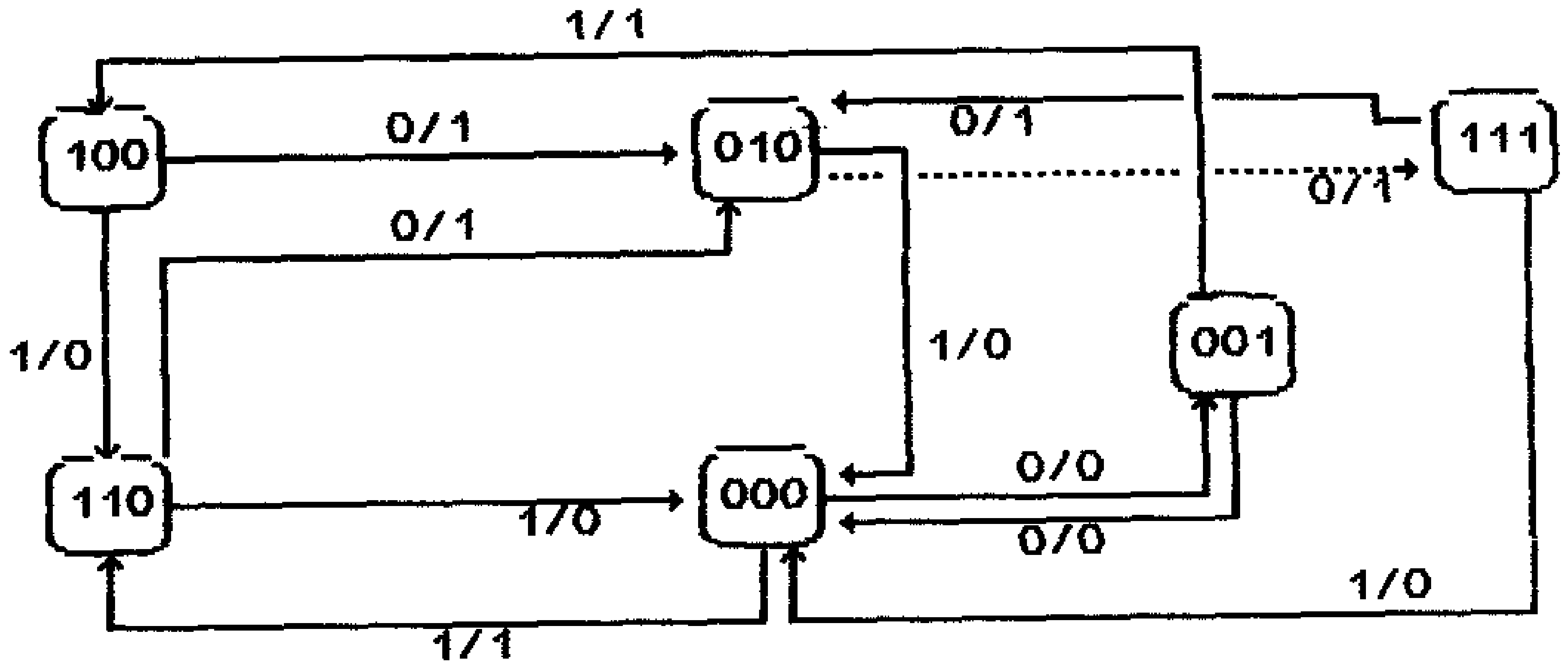


Fig.2.4. FAULTY MACHINE WITH w2 S-A-1 ~~(INVALID SRF)~~.

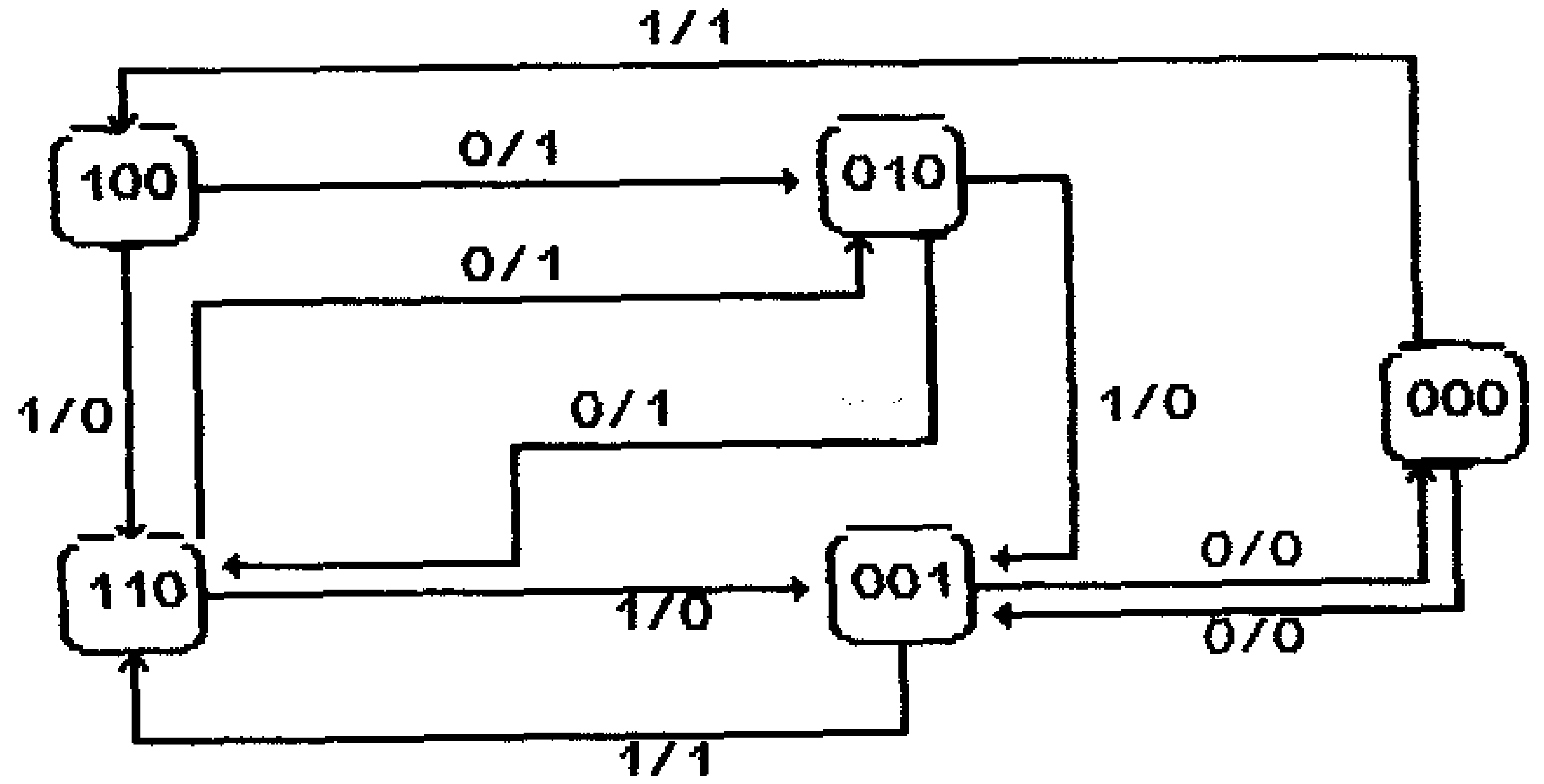


Fig.2.5. FAULTY MACHINE WITH ISOMORPH SRF.

The isomorphic faulty machine given in Fig.2.5 is hypothetical, and does not arise as a result of some real stuck-at fault in the circuit of the fault free machine.

Substantial work on synthesis of sequential machines with an aim to eliminate SRFs have been done in [SD90] and [SD91]. Following sufficient conditions for elimination of isomorph-SRF are well-covered in [SD90].

Lemma1: The Stuck-at faults on the primary input (PI), primary output (PO), present state (PS) and next state (NS) lines cannot produce a faulty state transition graph G^F that is isomorphic to G.

Definition1: A multilevel network is **inversion-parity invariant** if for any fault in the network, other than on the PI lines, the parity of inversions is the same (either odd or even) for all paths to the POs.

Obviously any two level network is inversion-parity invariant. Also, networks that are synthesized by algebraic factorization from two-level networks are also inversion-parity invariant.

Theorem1: If the two level combinational circuit implementing the NS lines and output logic functions is prime and irredundant, then any fault F in the circuit cannot produce a G^F that is isomorphic to G. Also, if a prime and irredundant multilevel circuit is synthesized such that it is inversion-parity invariant, then any fault F in the circuit cannot produce a G^F that is isomorphic to G.

Outline of the proof : The circuit being inversion-parity invariant, it has the property that, faults on the intermediate lines and outputs will produce a D or $\overline{D}$ at the outputs of the network, uniformly for all test vectors that detect the fault. The concept of the logic values D and $\overline{D}$ is given in [ROTH66]. The above reason suffices the invalidation of swapping of state codes under a single fault condition. For example, a s-a-0 fault might result in the next state logic in the faulty machine producing state code 001 rather than 101, but the same fault could not also cause the next state logic in the faulty machine to produce state code 101 rather than 001. This is because, if a single fault produces a 0 at the first bit position of the state code rather than 1, i.e, produces a D, then the same fault cannot produce a 1 at the same bit position (NSL) of the state code rather than a 0, i.e, a $\overline{D}$. (Q.E.D.).

Definition2: A state assignment of G^M is deemed to be **locally optimal** with respect to a subset of states $Q_I \in G^M$, if interchanging of codes of $q \in Q_I$ does not produce, after optimization, a logic implementation that is exactly the same as the previous one, except with one less literal.

If a state assignment of G is locally optimal, with respect to interchanging the codes of a subset of states Q_I, then there will be no swapping of state codes involved in the set of states Q_I, that will cause isomorphism. This is because, the isomorphism produces a G^F which is a better implementation (after optimization) than that of G (with at least one less line), and this contradicts the fact that the initial state assignment for G^M that produced G locally optimal under the exchange(s) of the codes of state Q_I. This is quite a strong result, but finding such an state assignment is computationally very expensive. In the next section, we will find a case under which an isomorph SRF is actually possible.

3

THE BASIC APPROACH.

To formulate an example wherein an isomorph-SRF occurs under faulty condition, we start with a simple circuit with only two next state lines. The corresponding STG of such a sequential circuit will have a maximum of four states, out of which two possible states could be assigned codes '01' and '10'. If we can show that under a single stuck-at fault, the next state logic changes in such a way that the codes '01' and '10' get swapped, as well as the output logic gets modified to corrupt all the fanout edges of the state codes '01' and '10' in such a way that the STG under the faulty condition remains isomorphic to the true STG, then we are done.

If a fault swaps the two next state functions Y_1 and Y_2, then the swapping of codes '01' and '10' is possible, whereas the codes '11' and '00' remain unaffected. Such type of swapping function in irredundant combinational circuits, under stuck-at faults, has been suggested in [SC92].

Definition : Two boolean functions are said to be in **P-equivalence** class if one can be transformed to another by a permutation (P) of two or more input variables.

Definition : A network is said to be **p-redundant** if it is possible to realize the same function by introducing some stuck-at faults and permuting some input literals.

The function realized by the network of Fig.3.1 is given by:

$$Y_1 = x_1x_2+x_2x_3+x_3x_4+ x_2x_4+ x_1x_4 \qquad (I)$$

A stuck-at zero (s-a-0) fault at line k now changes Y_1 to Y_2

where $Y_2 = x_1x_3+x_2x_3+x_3x_4+ x_2x_4+ x_1x_4$ (II)

In the figures, boxes represent logic gates. The disjunction sign '$\vee$' on a gate represents an 'OR' gate and the conjunction sign '$\wedge$' on a gate represents an 'AND' gate.

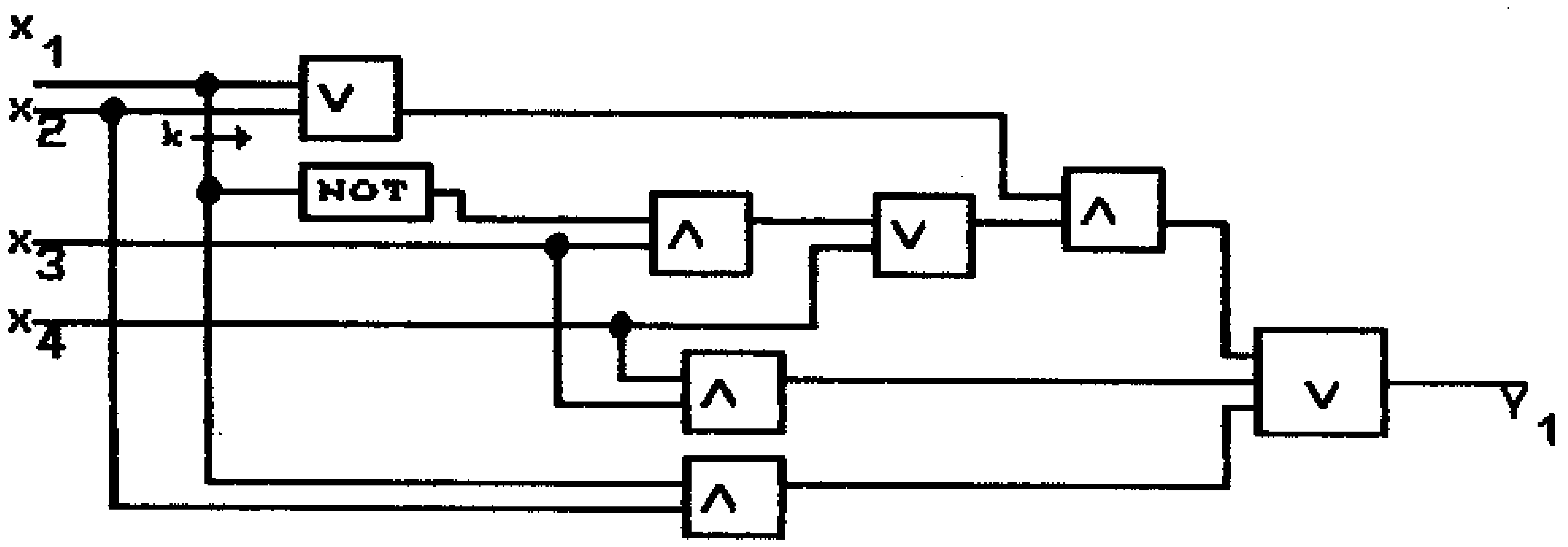


Fig.3.1. An interesting combinational circuit.

Clearly, Y_1 and Y_2 are in P-equivalence class, because by permuting x_2 and x_3 in Y_1, we get Y_2. Similarly, a single stuck-at fault is shown in Fig.3.2 (line k s-a-0) which changes Y_2 to Y_1. The circuits shown in Fig.3.1 and 3.2 have an interesting property. They realize unate functions. Even though they have lines with unequal parity, they are combinationally irredundant i.e., all stuck-at faults, single and multiple are detectable. However, they are p-redundant which is a new kind of redundancy, as observed in [SC92].

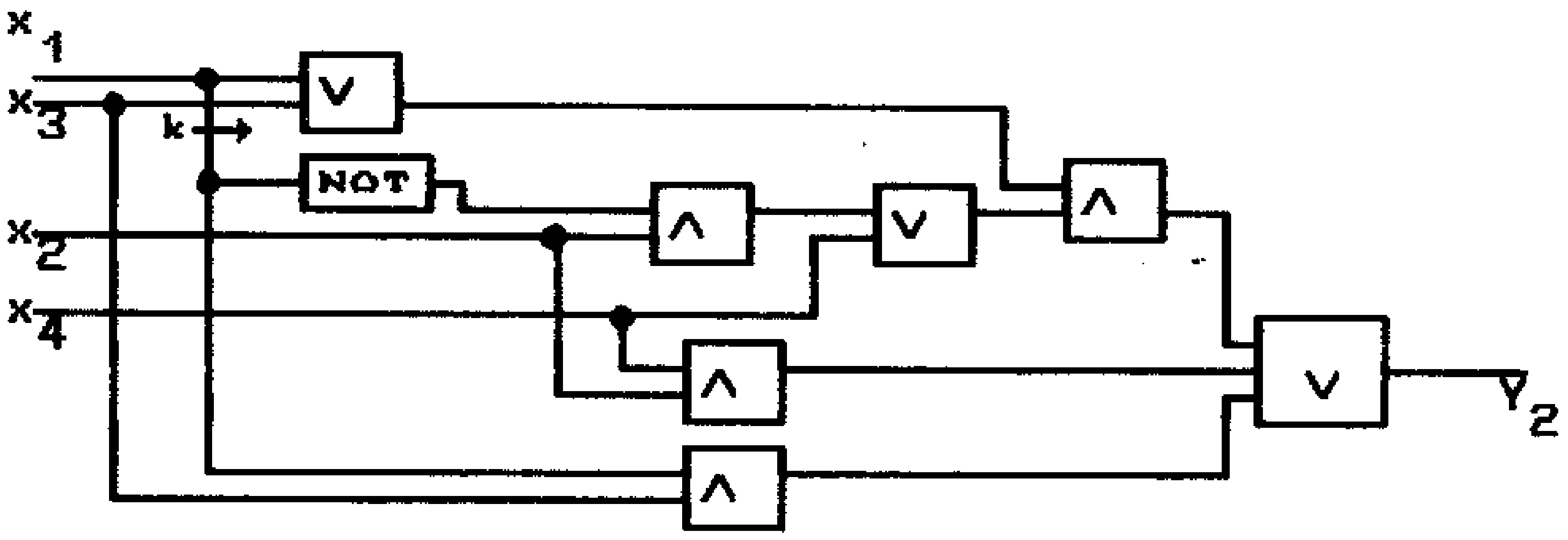


Fig.3.2

From the above setup, we can approach to formulate the example of occurrence of an isomorph-SRF by realizing the next state functions Y_1 and Y_2 by the networks given by Fig.3.1 and Fig.3.2, where Y_1 and Y_2 are represented as functions $f(x_1,x_2,x_3,x_4)$ and $f(x_1,x_3,x_2,x_4)$ in (I) and (II) respectively, where thefunction $f(a,b,c,d)$ is given by :

$$f(a,b,c,d) = ab + ad + bc + bd + cd \qquad \text{(III)}$$

As functions Y_1 and Y_2 are p-redundant, and under the same stuck-at fault k (s-a-0), they get interchanged and hence solve the purpose of next state functions in our example. Now, the output function should be so formulated that the output label, in the case of the true machine, for a particular primary input α and true present state β, should be equal to the output label of the faulty machine for the same input label α and present state γ, where γ is the code for the state in the faulty STG, corresponding to the state β in the true STG.

As shown earlier, the effect of the stuck-at fault in the given network realizing the function $f(a,b,c,d)$, isequivalent to interchanging the literals b and c in the original function. Exploring this information, we can realize the output function Z as

$$Z = f(x_1,y_1,y_2,x_4) \qquad \text{(IV)}$$

If the output logic is implemented in such a way that, the fault which affects the next state functions $Y_1$and Y_2, also affects the output logic in the same way, i.e., under faulty condition Z becomes equal to $f(x_1,y_2,y_1,x_4)$. Since the next state functions Y_1 and Y_2 get swapped by the fault, the present state value y_1y_2 also reverses after a delay of one clock-cycle. Hence, the faulty value of the output label remains unchanged for the same input label and swapped state code. This is the desired result for a machine, under faulty condition, to be isomorphic to the true machine.

Fig.3.3 shows the block diagram of a synchronous sequential circuit, which will behave in the above way. All the three logic blocks realizing Y_1, Y_2, Z get affected in the same way, by a single stuck-at fault, which will change the realized functions to p-equivalent functions as discussed earlier.

The circuit has been shown in Fig.3.4. Under fault-free condition

$$Y_1 = x_1x_2 + x_1x_4 + x_2x_3 + x_2x_4 + x_3x_4$$

$$Y_2 = x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_3x_4$$

and

$$Z = x_1y_1 + x_1x_4 + y_1y_2 + x_4y_1 + x_4y_2$$

Under faulty condition the functions realized are :

$$Y_1 = x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_3x_4$$

$$Y_2 = x_1x_2 + x_1x_4 + x_2x_3 + x_2x_4 + x_3x_4$$

and $Z = x_1y_2 + x_1x_4 + y_1y_2 + x_4y_1 + x_4y_2$

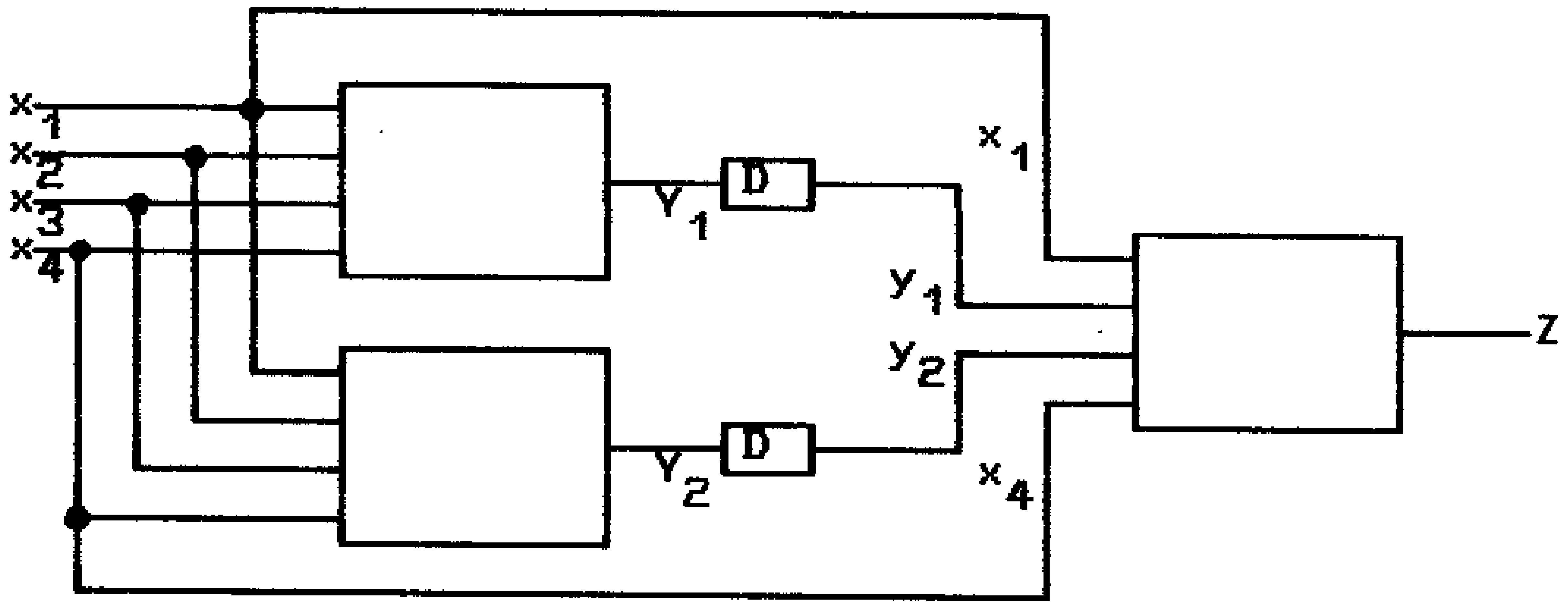


Fig.3.3. Block diagram of the logic circuit.
(D denotes a D flip-flop)

The above circuit is very simple, both structurally as well as functionally. The two next-state functions and the output function are unate boolean functions; moreover, the circuit is a feed-forward type of sequential machine. It has four input lines, one output line and four states.

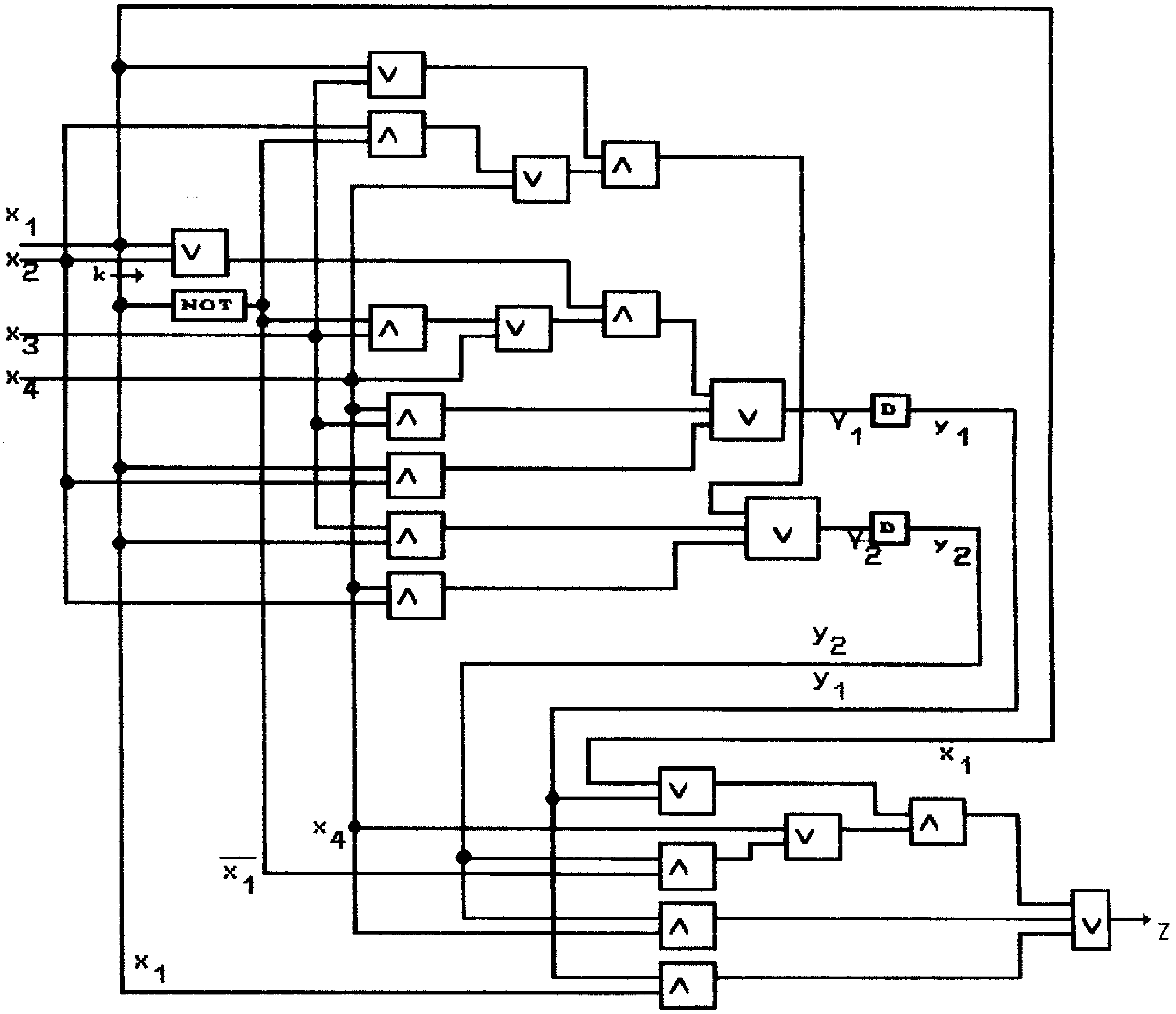


Fig.3.4. An example of a sequential circuit.

(Line k is s-a-0)

Table-1

	Decimal equivalent of input $x_1x_2x_3x_4$															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	A,0	A,0	A,0	D,0	A,0	D,0	D,0	D,0	A,0	D,1	B,0	D,1	C,0	D,1	D,0	D,1
B	A,0	A,1	A,0	D,1	A,0	D,1	D,0	D,1	A,0	D,1	B,0	D,1	C,0	D,1	D,0	D,1
C	A,0	A,1	A,0	D,1	A,0	D,1	D,0	D,1	A,1	D,1	B,1	D,1	C,1	D,1	D,1	D,1
D	A,1	A,1	A,1	D,1	A,1	D,1	D,1	D,1	A,1	D,1	B,1	D,1	C,1	D,1	D,1	D,1

STATE TABLE OF FAULT-FREE MACHINE.

Table-2

	Decimal equivalent of input $x_1x_2x_3x_4$															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	A,0	A,0	A,0	D,0	A,0	D,0	D,0	D,0	A,0	D,1	C,0	D,1	B,0	D,1	D,0	D,1
C	A,0	A,1	A,0	D,1	A,0	D,1	D,0	D,1	A,0	D,1	C,0	D,1	B,0	D,1	D,0	D,1
B	A,0	A,1	A,0	D,1	A,0	D,1	D,0	D,1	A,1	D,1	C,1	D,1	B,1	D,1	D,1	D,1
D	A,1	A,1	A,1	D,1	A,1	D,1	D,1	D,1	A,1	D,1	C,1	D,1	B,1	D,1	D,1	D,1

STATE TABLE OF FAULTY MACHINE.

Table-3

	y_1y_2
A	00
B	01
C	10
D	11

CODES OF CORRESPONDING STATES.

State tables of the true and the faulty machine are given in Table-1 and Table-2 respectively. Table-3 shows the corresponding codes of the states of the machine. From Table-1 it is clear that the fault-machine is reduced, i.e., no two states are equivalent. Notice that Table-2 is will be identical to Table-1 when the states B and C are interchanged. Hence, the faulty machine under the effect of a single stuck-at fault(k^o) becomes isomorphic to the fault-free machine. Furthermore, the circuits realizing the next state functions Y_1 and Y_2 and the output function Z are combinationally irredundant. Thus the above circuit serves as an example of a combinationally-irredundant, reduced sequential machine, that has an isomorphic type of sequentially redundant single stuck-at fault.

4

CONCLUSION.

The circuit shown earlier is the first example of an isomorphic-redundant sequential machine ever reported in the literature. Since the behaviour of the machine depends on the starting state, a different state assignment perhaps would not produce an isomorphic SRF under a single stuck-at fault. Though certain sufficient conditions for non-occurrence of isomorphic faults have been found, yet the necessary conditions for the occurrence of an isomorphic SRF are unknown. The approach described in this part may be extended for the generalised case of multiple next-state codes swapping and appropriate modification of the corresponding output logic, to find out cases for isomorphic SRFs.

PART II

TEST GENERATION FOR NON-SCAN SEQUENTIAL CIRCUITS

ABSTRACT.

Test generation for sequential circuits has always been a difficult problem. Early methods in tackling this problem involved the use of both random and deterministic techniques. The popular approach for enhancing the testability of sequential circuits has been the scan design methodology.

In contrast to the previous approaches, the approach implemented to generate test vectors for sequential circuits has been decomposed into three subproblems of combinational test generation, fault-free state justification and state differentiation under faulty condition.

Circuit specifications conforming to the ISCAS-'89 benchmark sequential circuits have been used for testing in the test generation program.

1

INTRODUCTION.

Given a fault for which a test sequence has to be generated, first a combinational test vector is generated, that propagates the effect of the fault to the next state lines or primary outputs. The process of finding an input sequence which takes the machine from the reset state to the fault excitation state is called **state justification**. After the combinational test vector is generated, a justification step is then performed, which involves finding a justification sequence for the state corresponding to the generated test vector. This step is carried out efficiently using a sequence of cube intersections on the complete or partial ON/OFF-set representations of the next state lines by implicitly searching the state space. Implicit searching of state space is done by the use of PODEM algorithm [PG81]. Thus a fault-free justification sequence is found. If the effect of the fault-free justification sequence, under faulty condition, propagates the effect of the fault to next state lines alone, then the true-faulty state pair produced by the test vector is found. A diffentiation sequence under faulty condition for this true-faulty state pair is obtained using another sequence of cube intersections, this time using the ON/OFF-set representations of the primary outputs. A **differentiating sequence** for a pair of states, S_1 and S_2, in a sequential circuit is a sequence of input vectors, such that, if the sequence is applied to the circuit when the circuit is initially in S_1, the last vector in the sequence produces a different primary output if the circuit initially were in state S_2. A test sequence for a fault is obtained by concatenating the justification sequence, the excitation vector, and the differentiation sequence. The approach followed is given in [AG91]. In the original work as given in [AG91], fault-free state differentiation has been suggested. In this implementation, state differentiation under the fault condition has been considered. Empirical evidence has shown that over 99% of the real time, in real circuits, a justification sequence, valid in a fault-free machine, is also valid in the faulty machine or is in itself a test sequence for the fault. The latter condition becomes true, when the original justification sequence, propagates the effect of the fault to one of the primary output lines, under faulty condition. In the unlikely event that a justification sequence is neither valid in the faulty machine nor a test sequence itself, the fault-free state justification approach has been implemented.

2

PRELIMINARIES.

A **state** is a bit vector of length equal to the number of flip-flops in the sequential circuit. A state with only 0's and 1's as bit values is called a **minterm** state. In general, a state is a **cube**, i.e., the valus in the different bit positions may be 0, 1 or x (don't care). A cube state is a group of minterm states.

A state is said to **cover** another state if the value of each bit position in the first state is either an x or is equal to the value of the corresponding bit position in the second state.

The **ON-set** of a primary output or a next state line is the complete set of input values such that the primary output or next state line is 1. similarly the **OFF-set** of a primary output or a next state line is the complete set of input values such that the primary output or next state line is 0. The set of cubes C, is said to be a **cover** for a ON-set if the ON-set Xon is a subset of C and C does not intersect the OFF-set Xoff.

In a sequential circuit the fault may be **combinationally** redundant or **sequentially** redundant. For a combinationally redundant fault, the effect of the fault cannot be propagated to the primary outputs or next state lines, beginning with any state and any input vector. A sequentially redundant fault is a fault which cannot be excited or whose effect cannot be propagated to the primary outputs using any sequence of input vectors starting from the reset state of the machine.

3

COMBINATIONAL TEST GENERATION.

Given a fault for which a test sequence is to be generated, the first step in sequential test generation is to generate a combinational test vector for the fault. The circuit is considered to be combinational with inputs being the primary inputs and the present state lines and the outputs being primary outputs and next state lines.

The most widely used algorithm for combinational test generation is the D-algorithm (DALG)[ROTH66]. This is a complete algorithm in the sense that it will generate a test for any logical fault if such a test exists. D-algorithm uses a five valued logic (0, 1, x, D, $\overline{D}$) to describe the behaviour of the circuits with faults. The value D designates a logic value 1 for a net in the good circuit and a 0 for the same net in the failing circuit. $\overline{D}$ is the complement of D and x represents a DONT-CARE. A difference in behaviour between the good circuit and the failing circuit propagates along a sensitized path. A test is generated when a sensitized path is built from the output of the gate under test (GUT) to some primary output (PO). Setting up of an sensitized path involves setting all inputs to the gates G_i's on the path, except an input on the path to 1 for an AND/NAND gate and 0 for an OR/NOR gate. This process is called the forward trace or error propagation phase of the method. Finally, an input pattern is found that realizes all the necessary gate input values. This is done by tracing backward from the inputs of the gates on the sensitized path to the primary inputs of the circuit. This process is called the backward trace or line justification phase of the method. Next, the implication is performed. The implication procedure specifies all gates, inputs and outputs that will be determined uniquely or implied by other line values in the current primary input assignment. An inconsistency occurs when the value implied on the line is different from the value that has already been specified on the line. If an inconsistency occurs, backtracking is performed to the last point at which a choice existed, all lines are reset to their values at this point, and one starts again with the next choice. The implication operation completely traces such signal determination both forward and backward through the circuit until the faulty signal D or $\overline{D}$ is propagated to a primary output.

The D-algorithm has been pointed out to be ineffective for the class of combinational circuits used to implement error-correction-and-translation (ECAT) functions. ECAT-type circuits are characterized by consisting of some number of XOR trees with reconvergence. In generating a test, D-algorithm creates a decision structure in which there is more than one choice available at each decision node. Through an implicit enumeration, all alternatives at each decision node are capable of being examined. In ECAT-type circuits, corresponding to a particular assignment to input nets of some intermediate gates, the algorithm may have to enumerate input values exhaustively until the absence of the justification is confirmed, thus the algorithm backtracks tediously many times until it changes the original assignments at the intermediate gates and tries again for justification.

The PODEM (Path Oriented DEcision Making) test generation algorithm [PG81] has been found to be more efficient than the DALG. This is an implicit enumeration algorithm in which all possible primary input patterns are implicitly, but exhaustively examined as tests for a given fault.

Implicit enumeration refers to a subset of the branch and bound algorithm designed specifically for search of an n-dimensional state space. In the test generation procedure using PODEM, all primary inputs (PIs) are initially at x. The implicit enumeration process used in PODEM uses a decision tree for the branch and bound operations. An initial assignment (branching) of either 0 or 1 on a PI is recorded as an unflagged node in the decision tree. The decision tree is an ordered list of nodes with :

1) Each node identifying a current assignment of either a 0 or 1 to one PI, and
2) The ordering reflects the relative sequence in which the current assignments were made.

Forward implications of the present PI assignments is done as in DALG. A node is flagged if the initial assignment has been rejected and the alternative is being tried. When both assignment choices at a node are rejected, then the associated node is removed and the predecessor node's current assignment is altered, if possible. The last PI assignment made is rejected, if it can be determined that no test can be generated with the assignments made with the assigned PI's, regardless of the values that may be assigned to the as yet unassigned PI's. The rejection of a PI assignment results in a 'bounding' of the decision tree.

A two-step procedure is used to choose a PI and logic level for initial assignment.

Step-1: The initial objective is determined- the objective is to bring the test generator closer to its goal of propagating a D or $\overline{D}$ to a PO.

Step-2: Given the initial objective, a PI and the logic level is so chosen that the chosen logic level assigned to the chosen PI has a good likelihood of helping towards meeting the initial objective.

An objective is defined by: 1) a logic level 0 or 1 that is referred to as the objective logic level, and 2) an objective net which is the net at which the objective level is desired.

If the failing gate or gate under test (GUT) does not have a D or $\overline{D}$ on its output, the initial objective is directed towards promoting setup for the gate. Once the GUT has been set up, the initial objective is aimed at propagating a D or $\overline{D}$ on level of logic closer to a PO than before. In PODEM, determination of initial PI assignment, given an initial objective is known as back-tracking. In DALG, since the assignment of values is allowed to internal lines, more than one choice is available at each internal line or gate and backtracking could occur at each gate. In contrast, the PODEM algorithm allows assigning of values only to PI's. The values assigned to PI's are then propagated toward internal lines by the implication.

The backtracking procedure causes a path to be traced from the objective net backwards to a primary input. Since the initial PI assignment corresponds to making a decision at a node of the decision tree, the algorithm has been named Path Oriented **DEcision** making (PODEM) test pattern generator. The following flowchart in Fig.1 gives the high level description of the algorithm.

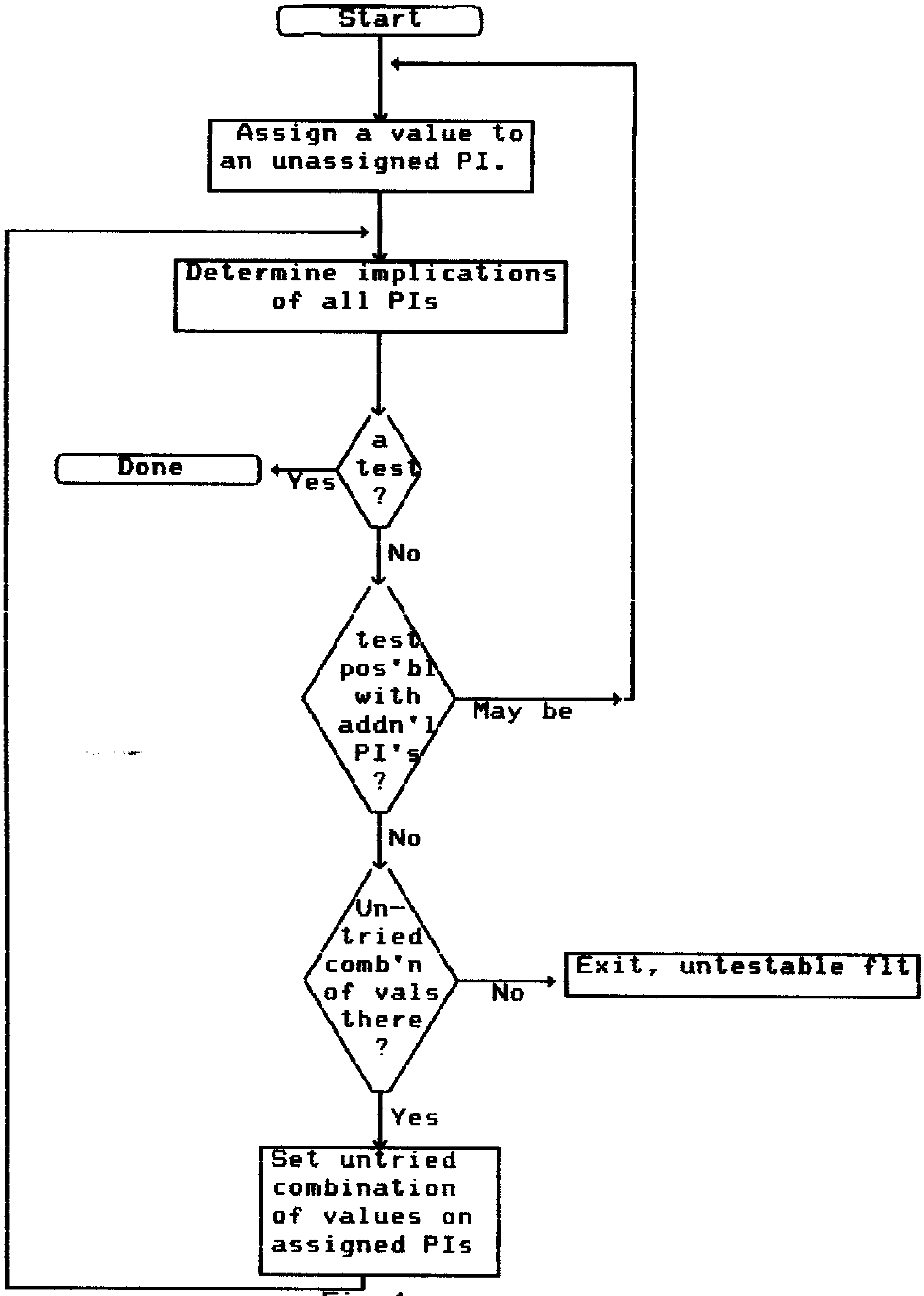

Start
Assign a value to an unassigned PI.
Determine implications of all PIs
a test ?
Yes
Done
No
test pos'bl with addn'l PI's ?
May be
No
Un-tried comb'n of vals there ?
No
Exit, untestable flt
Yes
Set untried combination of values on assigned PIs

Fig.1

4

FAULT-FREE STATE JUSTIFICATION.

The first step in test generation is the enumeration of complete ON and OFF-sets of each of the next state lines (NSLs) and POs of the sequential circuit. Cover enumeration has been done by using the implemented PODEM-based enumeration algorithm. Since on every backtrack PODEM sets an input line to a value different from what it had previously, the cover of the ON and OFF-sets are guaranteed to be single cube containment minimal. Partial covers can also be extracted instead of complete covers in the case the CPU time or memory requrement becomes prohibitive.

State justification has been implemented as follows :

After the combinational test is generated with the help of PODEM algorithm, the excitation vector is examined to see if the present state part of the vector covers the reset state. If it is so, then the fault can be excited from the reset state of the machine. If not, then the excitation state is justified using a backward justification algorithm as will be described later. Backward justification is performed by first finding all the fan-in states of the excitation state via repeated cube intersection. If the reset state is a member of the set of fan-in states, then a single vector justification sequence is found. Otherwise, the process is repeated for some state in the fan-in of the state being currently justified. Once a justification sequence is found, it is fault simulated to see if the required state is justified..if so, then the sequence is also valid in the fault machine. Otherwise, some edge in the justification path must have been corrupted and only a part of the sequence may be enough to be used as a justification sequence. If the effect of the fault under test has been propagated to the POs by the combinational test vector, and the excitation state is justifiable in the faulty machine, then a successful test has been generated. If however, the effect is propagated to one of the next states only, then the fault-effect is propagated to some PO by the state differentiation. If the excitation vector cannot be justified under fault, then justification of another excitation vector (found by PODEM), is carried out.

If justification of the state S_1 is carried out, the entire fan-in of S_1 can be found out by cube intersections. If nocube is found which covers the reset state, then a single vector justification sequence does not exist for any of the states in S_1. Thus an N-vector (N > 1) sequence is found by heuristically selecting a group of states S_2 which exist in the fan in S_1 and attempt is done to justify some state in S_2, via a single vector justification sequence. While selecting a state it is so chosen that it should not be covered by any of the states in the potential justification path built so far, to prevent cycles during justification. The procedure select_state does this heuristics as given in the algorithm next.

The justification algorithm in pseudo code is given below.

```
 function justify_state(state) {
   if (reset state covered by state) return (True);
   fanins = universal cube;
   for (each PS line that is a 0 or 1) {
     fanins = fanins $\cap$ (ON or OFF set of corresponding NS line)
   }
   if (reset state covered by some state in  fanins)  return(True);
   Pjs = NULL          /* Potential justification sequence */
   while (there are still cubes in fanins) {
      fanin_state = select_state();
      if (fanin_state is found) {
         Pjs = Pjs $\cup$ fanin_state;
         justify_state(fanin_state);
         if (justification sequence is found) return(True);
      }
      else {
            Pjs = Pjs - fanin_state;
      }
   }
   return(False);
}
```

Fig.2 : Justification algorithm.

5

STATE DIFFERENTIATION.

After justification, the true-faulty state pair (S_1^t, S_1^f) given by the test vector has to be differentiated. Some bit positions in S_1^t and S_1^f may be x (Don't care), which means that in general, two group of states are to be differentiated. Since differentiation is carried out under faulty condition, the ON and OFF-sets of all the POs and next-state lines (NSLs) are extracted by the use of PODEM, with the stuck-at fault taken into account. This is done only once, before the differentiation procedure is called, for each stuck-at fault.

The procedure for single vector differentiation is as follows:

1) Pick a (new) output line.

2) Inspect the true ON-set covers and faulty OFF-set covers of the output line and search for a primary input combination i_1, which appears concatenated with S_1^t in the true ON-set and concatenated with S_1^f in the faulty OFF-set (or vice versa). If such an input combination is found for some output line, then the state differentiation sequence can be constructed. If not, a single vector state differentiation sequence cannot be found. Multiple vector differentiation can be searched in the fillowing fashion. N is the # vectors in the current sequence.

3) N = 1;

4) Pick a next state line and attempt to find a primary vector (as outlined in step 2, above), i_N, that produces a 1 (0) when concatenated with S_N^t and a 0 (1) when concatenated with S_N^f.

Try another NSL if a vector cannot be found for the picked one. If an input combination cannot be found for any such NSL, then a state differentiation sequence cannot be found for (S_N^t, S_N^f).

5) Find the state pair (S_{N+1}^t, S_{N+1}^f) given by the fanout states of the primary vector i_N for the state pair (S_N^t, S_N^f). $N = N + 1$. Attempt to find a single vector propagation sequence for the pair (S_{N+1}^t, S_{N+1}^f).

Pseudo code for differenting the states S^t and S^f is given in the next page. The function if_inpvec_concats_true_ON_faulty_OFF returns a true flag and an input vector if such an input vector is found which when concatenated with S^t produces a 1 in a given PO or NSL and a 0 in the same line, when concatenated with S^f. Otherwise a false flag is returned. The function lf_inpvec_true_OFF_faulty_ON does the vice versa. Pds is the potential differentiating sequence built so far.

Once the differentiation sequence is found out, the test sequence is obtained concatenating the justification sequence, the excitation vector, and the differentiation sequence.

```
function differentiate($S^t$, $S^f$) {
    for (each PO line) {
        flag = if_inpvec_concats_true_ON_faulty_OFF($S^t$, $S^f$);
        if (flag) break;
        flag = if_inpvec_concats_true_OFF_faulty_ON($S^t$, $S^f$);
        if (flag) break;
    }
    if !(flag) {
     for (each NSL) {
        flag = if_inpvec_concats_true_ON_faulty_OFF($S^t$, $S^f$);
        if (flag) {
          Pds = Pds + inpvec;
          implyunderfault(&$S^{t'}$, &$S^{f'}$);
          flag = differentiate($S^{t'}$, $S^{f'}$);
          if not(flag) Pds = Pds - 1;
        }
       if not(flag) {
           flag = if_inpvec_concats_true_ON_faulty_OFF($S^t$, $S^f$);
           if (flag) {
             Pds = Pds + inpvec;
             implyunderfault(&$S^{t'}$, &$S^{f'}$);
             flag = differentiate($S^{t'}$, $S^{f'}$);
             if not(flag) Pds = Pds - 1;
           }
        }
        if (flag) break;
     }
    }
return (flag);
}
```

Fig.3. State differentiation algorithm.

6

IMPLEMENTATION.

A suitable data structure for representation of logic gates which helps in both forward implication and backtracking, has been implemented in the test generation algorithm. Each gate is represented by a record structure having the following fields.

1. **no** /* Gives the gate identification number */
2. **id** /* The identification code for the type of logic gate, such as AND, OR, NOR etc. */
3. **val** /* To store the logic value 0,1,x,D or $\overline{D}$ as implied by the logic gate */
4. **inps** /* # input lines that fan in to the gate */
5. **ops** /* # fanouts from the gate */
6. **fanin [1..inps]** /* An array of integers of valid entries upto the length of inps indicating the identification numbers of all the fan-in gates. */
7. **fout [1..ops]** /* An array of integers of valid entries upto the length of ops indicating the identification numbers of all the fan-out gates. */

As the gates are identified by a name (Character string) in the ISCAS circuit specification files, an array called ExtIntname is used in the program which stores the external names and the corresponding internal names of the gates become the index of the array at which the names are stored. The integer inernal name gives the gate identification number. The record structures of the gates are allocated dynamically as the logic circuit is constructed in the parsing phase of the circuit specification file and an array of pointers called gateptr[] is maintained to store the pointers to the structures allocated in the dynamic memory.

The basic steps of the test generation algorithm are :

1. Parse of input circuit specification file and construct of the combinational logic circuit.

2. Extract of the complete set of ON and OFF covers of each PO and NSL and store them in the array OnOffset[], by using the PODEM algorithm.

3. Generate a (new) test vector using PODEM. If no (more) test vector can be generated, exit with failure.

4. Find the justification sequence of the present state (PS) part of the combinational test vector. Fault simulate the justification sequence and see if the effect of the fault has been propagated to one of the POs or NSLs. If not, goto step 3. If the effect of the fault has been propagated to NSLs only, then the true and faulty state pairs are to be differentiated, otherwise exit with the current justification sequence as the test sequence.

5. If the ON and OFF-sets of the POs and NSLs under the given faulty condition has not been generated, generate them using the PODEM algorithm and taking the stuck-at fault into consideration.

6. Find a differentiating sequence under faulty condition for the true and faulty state pair. If the pair of states cannot be differentiated goto step 3, otherwise exit with the concatenation of the justification sequence, the excitation vector and the differentiating sequence as the test vector of the fault.

7

CONCLUSION.

The approach implemented for test generation is quite a novel approach, as suggested in [AG91]. This method also identifies the presence of sequentially redundant faults namely, *unjustifiable* faults and *undifferentiable* faults. If all the excitation states are unjustifiable, then the fault under test is sequentially redundant. If for all possible combinational tests for the fault, the true and the faulty state pairs are not differentiable, then the fault is also sequentially redundant. Studies of the implemented algorithm on benchmark circuits is underway.

REFERENCES.

[AG91] Abhijit Ghosh, Srinivas Devadas and A. Richard Newton, "Test generation and verification for highly sequential circuits," in IEEE Transactions on Computer Aided Design, Vol. 10, No. 5, May 1991.

[PG81] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," IEEE Trans. on Computers, pp 215-222, Mar 1981.

[ROTH66] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," IBM J. Res. Development, Vol 10, pp 278-291, July 1966.

[SC92] Susanta Chakraborty, Debesh K. Das, Bhargab B. Bhattacharya, "Logical redundancies in irredundant combinational circuits," (to be published).

Address for correspondence: Dr. B. B. Bhattacharya, Electronics Unit, Indian Statistical Institute, 203, B. T. Road, Calcutta-700 035, India.

[SD90] Srinivas Devadas, *et al.*, "Irredundant Sequential machines via optimal logic synthesis," IEEE Trans. on CAD, Vol 9, No. 1, Jan 1990.

[SD91] Srinivas Devadas, *et al.*, " A unified approach to the synthesis of the fully testable sequential machines," IEEE Trans. on CAD, Vol 10, No. 1, Jan 9091.

The following M.Tech(CS) II year students (1991-92) have submitted to the Dean's Office the copies of their dissertations for the Library. These are being sent herewith.

	Name	Title of Dissertation
1.	Jayadeep Das	Studies on Isomorphic-Redundancy and Testing of Non-Scan Sequential Circuits
2.	M. Suresh	A Page Management Scheme for an Edge Coloured Multigraph
3.	Rajan Gangadharan	Heuristic Algorithms for Determining Feasible Routing order in Nonslicible Floorplans.
4.	M. Devi Prasad	Analysis of Permutations for Routing in Multistage Interconnection Networks.
5.	Milind B. Kamble	Studies in Short Circuit Fault Diagnosis of Multistage interconnection Networks.
6.	Koushik Dasgupta	Some Studies on non Recursive Evaluation of Network Reliability.
7.	Debashree Ghosh	Location of Largest Empty Staircase Polygon Among Point Obstacles.
8.	Arani Sinha	Location of Largest Isothetic Empty Rectangle Among Arbitrarily Oriented Line Segments.
9.	Devendra Singh	Processing of Finger Print Image.
10.	Ashutosh Kumar Jha	Recognition of Well Formed Handwritten Devnagari Characters.
11.	Jyotishman Chatterjee	A Software for Maintaining a Conceptual Hierarchy of Objects to Aid in Theta-Map in a Natural Language Processing System For Bangla.
12.	Vuppala Sreenivas	Game Playing Using Expert Systems.
13.	Hemanta Ranjan Panda	Rule Based 'Sandhi Bicched' (de-euphon of Bengali.
14.	Prasenjit Pal	Gamera Calibration for Stereo Vision Using Imaging Geometry
15.	Rahul Bhattacharyya	Automatic Selection of Structuring Elem for Object Classification Through Morph
16.	Krishnendu Chakraborty	A Visual Object Representation Scheme.

(J.B. Choudhuri)
Executive Officer