A CAD Tool for Test Generation in Distributed Computing Environment

A Thesis
submitted in partial fulfillment of the requirements for the award of degree of

MASTER OF ENGINEERING
in
SOFTWARE ENGINEERING

Under the Supervision of:

Ms. Seema Bawa
Assistant Professor & Head
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology

Submitted By

Shaveta Datta
M.E. (Software Engineering)
8023118

Computer Science and Engineering Department
Thapar Institute of Engineering and Technology
(Deemed University) Patiala -147 004

May 2004

Abstract

The work done here is inspired by the emergence of high performance, massively parallel computers and distributed computing environments, which demands the development of new parallel and distributed algoritms to take advantage of these technologies. Many complex problems can be solved efficiently by massive parallelism.

On the other hand, Advancements in integrated circuit (IC) technology have made it possible to fabricate digital circuits with a very large number of devices on a single chip. IC technology namely, Very Large Scale Integration (VLSI) involves the fabrication of millions of components and interconnections on a chip. The main advantages of VLSI are reduced system cost, improved performance, and high reliability. These advantages would be lost until the chips are economically tested. The availability of affordable parallel machines and distributed network of idle workstations in most of the design and test environments has prompted us for the development of efficient parallel and distributed algorithms for the compute-intensive test generation problem.

Test generation is known to be hard i.e. NP-complete problem. While the test generation for purely combinational circuit is challanging due to the high circuit complexity of VLSI circuits, thus, there exists a critical need to develop efficient test generation algorithms that can handle VLSI chips at a resonable computing cost and provide high fault coverage.

Literature review of the existing approaches for test generation has been done. The approaches have been classified into conventional and unconventional ones. The conventional test generation approaches for combinational circuits are reviewed. The most successful combinational test generation algorithms are found to be mainly based upon two steps: First, create a change at the fault-site and second, search for consistent value on all signal lines in the circuit such that the fault-effect is successfully propagated to at least one of its primary output (PO). This is called the Path Sensitization approach.

Most of the work done by researchers is on Stuck-at fault models; we found that 95% fault coverage was through stuck-at fault models. Then no body took concentration on rest 5% fault. That faults was due to delay, memory, register problems. But with in

change in time, researchers took these 5% faults i.e. Delay Faults under consideration. They added lot of research on this area. These delays are not due to logical problem of input-output but due to presence of hazards (static as well as dynamic hazards), delay defects like

- GOS defects
- Resistive shorting defects between nodes and to the supply rails etc.

Due to pressence of these delay faults, there was always performance degradation. Researchers proposed different-2 techniques, approaches methods to solve these delay faults. These delay faults are categorised as Path, Gate, Transition, Line, Segment, and Functional delays. In this thesis we are focussing on Path delay faults. Various approaches, methods used to solve path delay fault problem are described fully in literature review. We are using here, Algebra for test generation of these delay faults. In this thesis, we are focussing on ten-valued logic for robust tests and three-valued logic for non-robust tests. From the earlier research as mentioned in literature, by using sequential algorithm for path delay faults, we are not gaining efficient results. CPU utlization and memory consumption is coming high.

To cop this problem, several parallelization techniques for test generation problem have been investigated in the past. These techniques have been tried to parallelize some of the portions of the conventional uniprocessor algorithms and execute them in parallel. Although these techniques have shown some promising results, but much work remains in this direction since no effective parallel or distributed system has yet been found.

The thesis addresses to develop efficient parallel and distributed algorithms for the test generation problem. The scope of this work is limited only to combinational circuits.

Here in this thesis, we develop a parallel algorithm for test generation of path delay faults using 10-valued logic for robust tests and 3-valued logic for non-robust tests using master-slave approach in parallel and distributed environment (PVM) on LINUX platform.

We have successfully tested the prototype of the proposed solution for different ISCAS'85 benchmark circuits as well as some example circuits. The experimental results are encouraging. The plots of computational time vs number of processors, speedup vs.

number of processors and the computation time vs. no. of tested paths are shown. Average CPU time is coming low and memory utlization is less, speed-up is increasing as the no. of processors increases. These results clearly demonstrate the effectiveness of the proposed algorithm.

Candidate's Declaration

I hereby certify that the work which is being presented in the thesis entitled, **"A CAD Tool for Test Generation in Distributed Computing Environment"**, in partial fullfillment of the requirements for the award of degree of Master of Engineering in Software Engineering and submitted in Computer Science and Engineering Department of Thapar Institute of Engineering and Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of Ms. Seema Bawa.

The matter presented in this thesis has not been submitted by me for the award of degree of any other degree of this or any other University.

Shaveta Datta

(Shaveta Datta)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Seema Bawa 05/05/2004

Ms. Seema Bawa,
Supervisor,
Assistant Professor and Head,
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology,
Pataiala.

Countersigned by

Seema Bawa 05/05/2004

Ms. Seema Bawa,
Assistant Professor and Head,
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology
Pataiala.

5.5.2004

Dr. D.S.Bawa,
Dean of Academic Affairs,
Thapar Institute of Engg. and Tech.,
Patiala.

Acknowledgements

I wish to express my earnest gratitude to Ms. Seema Bawa, for providing me invaluable guidance and suggestions, which inspired me to submit this thesis report on time.

I would also like to thank all the faculty and staff members of Computer Science and Engineering Department of Thapar Institute of Engineering and Technology, Patiala for providing me all the facilities required for the completion of this thesis.

Last but not least I wish to thank all my classmates and friends for their timely suggestions and cooperation during the period of my thesis.

Shaveta Datta

Shaveta Datta

Regd. No. 8023118
M.E. (Software Engineering)

List of Figures

List of Tables

Contents

1.1 Introduction to VLSI Testing

The origin of computing devices lies in computing problems. However, once a computing device is built, it requires algorithms through which the problems would be solved. Since the advent of electronic computers, the past decades have seen intense activity in the development of algorithms. Through this activity have emerged a class of problems that seem to require the cooperative use of multiple computers. Many problems in the design of large digital circuits fall in this category.

A VLSI circuit consists of millions of components on a single chip. The design process transforms abstract functional specifications into a manufacturable assembly of known parts. **Verification** checks the correctness of design by analysing its simulated performance. Efficient chip fabrication involves thorough testing in order to make the chip fault-free. The **testing** process detects the physical defects produced during the fabrication of a VLSI chip. Such a chip is tested by a sequence of input stimuli, known as test vectors, which check for possible defects in the chip by producing observable faulty responses at primary outputs. Test generation involves the generation of test vectors to detect failures in the VLSI chip. A normal requirement of these tests is that they detect a large fraction of the faults. The detected fraction of faults is called the fault coverage. Also, since the same set of test vectors will be applied to a large number of copies of the chip, short test sequences are desirable.

No effective parallel architecture or algorithm for test generation has as yet been found. At least two possibilities exist, however. First, a conventional uniprocessor algorithm can be parallelized by finding portions where it can be pipelined or directly executed in parallel. Second the testing problem can be reformulated and new parallel methods can be developed.

Terms Associated with VLSI Testing

Input Vector: Set of values for all primary input signals.

Fault: Fabrication-related failure. A typical fault is the stuck-at-fault,

Test Vector: Input vector that results in a faulty circuit being differentiated from a fault-free one.

The Importance of Test Generation

The advantages of VLSI would be lost unless VLSI chips can be economically tested. An obvious reason for testing is to separate the good product from the faulty one. The dramatic increase in the ratio of internal devices to I/O terminal pins drastically reduces the **controllability** (ease of producing a specific internal signal value by applying signals to the circuit input terminals) and **observability** (ease with which the state of internal signals can be reduced from the signals at the circuit output terminals) of the circuit.

1.2 Introduction to Delay Fault Models

We know that most of faults are detected by stuck-at fault model approximately 80%, but remaining are delay faults, memory faults and bridge faults. For this work, we have chosen delay fault models.

Failures that cause logic circuits to malfunction at the desired clock rate and thus violate timing specifications are currently receiving much attention. Such failures are modeled as delay faults. They facilitate delay testing [37]. The use of delay fault models in VLSI test generation is very likely to gain industry acceptance in the near future.

A stuck-at-0 fault on a signal means that the signal can be set to 0, but then cannot be changed to 1. Alternatively, this situation can be described by saying that the signal will take an infinite" amount of time to rise from 0 to 1. Thus, a stuck-at fault is an infinite delay fault and, indeed, a circuit that passes stuck-at fault tests is not likely to have any infinite delay fault [74]. For digital systems that work at any appreciable speed, this is not sufficient. The operation of such systems is usually synchronized by clock signals and it is necessary that all combinational logic elements attain steady state within some specified clock period. Application of stuck-at fault tests at higher speed can uncover some delay defects. A number of defects that can cause delay faults:

- GOS defects
- Resistive shorting defects between nodes and to the supply rails

- Parasitic transistor leakages, defective pn junctions and incorrect or shifted threshold voltages
- Certain types of opens
- Process variations can also cause devices to switch at a speed lower than the specification.

However, at least four recent studies show that even that may not be sufficient and tests specifically generated to detect delay defects may be necessary.

Delay Fault Problem

Figure 1.1 shows a schematic of a digital system. Some inputs and outputs can be state variables connected to I/P-O/Ps (FF) (not shown) and others are primary inputs (PI) and primary outputs (PO.) All input changes are synchronized with a clock signal and all outputs are expected to attain their final steady state values within one clock period after the inputs change. Thus, for a correct operation the delay of the combinational logic should not exceed the clock period [74].

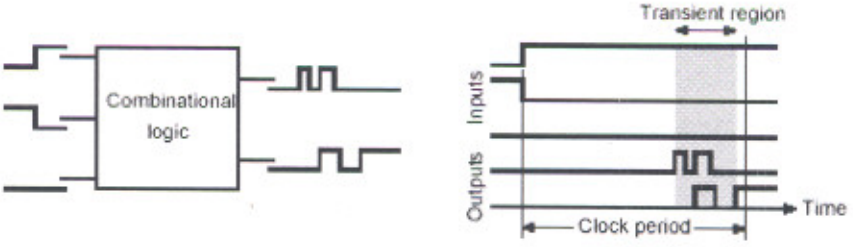


Fig 1.1 **Delay Fault Problem Defined**

Typical outputs of logic circuits contain transients as shown in Figure 1.1. There are several observations:

■ In order to examine the timing operation of a circuit we should examine signal transitions. The input signal in Figure 1 consists of two vectors: 010 → 100. Delay tests consist of vector-pairs.

■ All input transitions occur at the same time in Figure 1.1. Thus, the duration of the transient region at the input is zero. This, of course, is an idealized illustration though it closely represents the real situation. The transient region at the output contains multiple

transitions that are separated in time. As we will see later on, the position of each output transition depends upon the delay of some input to output combinational path.

- **Test Definition:**
  - At time t_1, the initializing vector of the two-pattern test, V_1, is applied through the input latches or PIs and the circuit is allowed to stabilize.
  - At time t_2, the second test pattern, V_2, is applied.
  - At time t_3, a logic value measurement (a sample) is made at the output latches or POs.
  - $T_C = (t_3 - t_2)$ represent the time interval between the application of vector V_2 at the PIs and the sampling event at the POs The nominal delay of each of these paths be defined as pd_i. The slack of each path be defined as $sd_i = T_C - pd_i$. This is the difference between the propagation delay of each of the sensitized paths in the nominal circuit and the test interval.

- The right edge of the output transition region (grey shaded area in Figure 1.1) is determined by the last transition, or the delay of the longest combinational path activated by the current input vector-pair. Considering all possible input vector-pairs, the longest delay combinational path of the circuit is known as the critical path. There can be more critical paths than one if several paths meet the maximum delay criterion. The delay of critical paths determines the smallest clock period at which the circuit can function correctly.

- For a manufactured circuit to function correctly, the output transition region for any input vector-pair must not expand beyond the clock period. Otherwise, the circuit is said to have a delay fault. A delay fault means that the delay of one or more paths (not necessarily the critical path) exceeds the clock period.

Digital system designers have traditionally maximized the frequency of system clocks in order to obtain the highest performance from the hardware. The maximum allowable clock rate is determined by the propagation delays of the combinational logic block between latches [37]. Consider the circuit under test shown in Figure 1.2. During the normal operation of the circuit, the input clock C_1 is the same as the output clock C_2 and the period (T_c) of C_1 and C_2 corresponds to the system clock. This period should be greater than the maximum propagation delay DP_{max} for the circuit.

However, during testing for delay faults, we use two separate test clocks, C_1 and C_2, running at a frequency that is slower than the normal system clock. Thus, the period of test clocks, T_t, is longer than T_c. The two test clocks are skewed by the amount T_c. The activation of the output clock C_2 must follow the activation of the input clock C_1 by at least DP_{max} time units (i.e., $T_c = t_2 - t_1 \geq DP_{max}$). If the output clock is activated sooner, then unstabilized and possibly incorrect logic values may be latched in output latches.

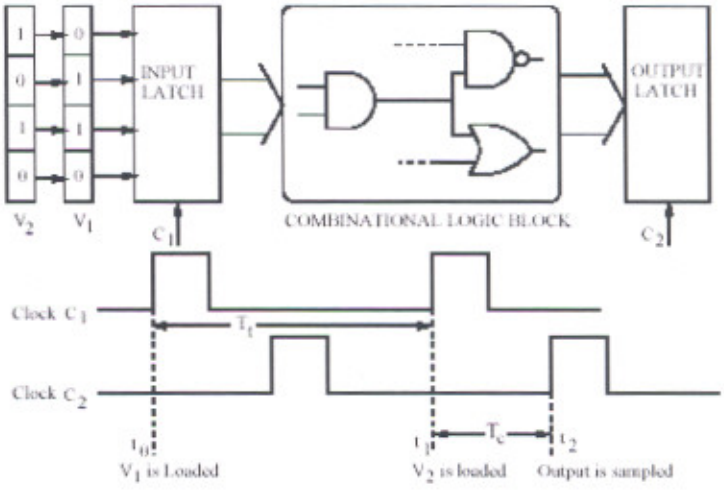


Fig 1.2 **Hardware Model & Clock Timings**

Since delay faults do not alter the logic function realized by a circuit and since the tests for stuck-at faults are normally applied at a slow clock rate (due to the ATE limitation or other reasons), they are inadequate for detecting delay faults. Special two-pattern test vectors are required for detecting delay faults. The hardware model used in delay fault testing is shown in Figure 1.2. Here the vector pair $< V_1, V_2 >$ constitutes a delay test and signals C_1 and C_2 are used to clock the input and output latches, respectively. At time t_0, an initializing input vector V_1 is applied, and the circuit is allowed to stabilize under input V_1. At time t_1, the propagation vector V_2 is applied, and the outputs are sampled at time t_2, where $(t_2 - t_1)$ is the intended time interval between the input and output clocks, called the rated clock interval T_c.

Exhaustive testing is quite impractical for delay faults since the total number of pattern-pairs required will be $(2^n)(2^n - 1)$, which is of the order 2^{2n}, for a circuit having n inputs. One must derive suitable and reasonable delay fault models and devise algorithms that can generate tests for the modeled faults.

Delay Fault Test Generation:

- Difficulties with delay fault test generation:
  - Test generation requires a sensitized path that extends from a PI to a PO.
  - Path selection heuristics must be used because the total number of paths is exponentially related to the number of inputs and gates in the circuit.
  - The application of the test set must be performed at the rated speed of the device.
    - This requires test equipment that is capable of accurately timing two-vector test sequences.
  - The detection of a defect that introduces an additional delay, ad_i, along a sensitized path is dependent on satisfying the condition:
    - $ad_i > sd_i$ (or $pd_i + ad_i > T_C$)
    - Therefore, the effectiveness of the delay fault test is dependent on both the delay defect size and the delay of the tested path.

Hazards

- A path sensitized by a delay test consists of on-path nodes and off-path nodes.
  - The nodes along the sensitized path are referred to as on-path nodes.
- Static sensitization defines the case when all off-path nodes settle to non-controlling values (0 for OR/NOR, 1 for AND/NAND) following app. of V_2.
  - This is a necessary condition to test a path for a delay fault.
- The gates along the sensitized path have exactly one on-path input and zero or more non-controlling off-path inputs.
  - Delay fault tests are classified according to the voltage behavior of the off-path nodes.
  - Such tests can be invalidated under certain conditions.
- Hazards can invalidate tests:

- Static hazard: describes a circuit condition where off-path nodes change momentarily when they are supposed to remain constant.
- Dynamic hazard: describes a circuit condition where off-path nodes make several transitions when they are supposed to make a single transition.

Static Hazards

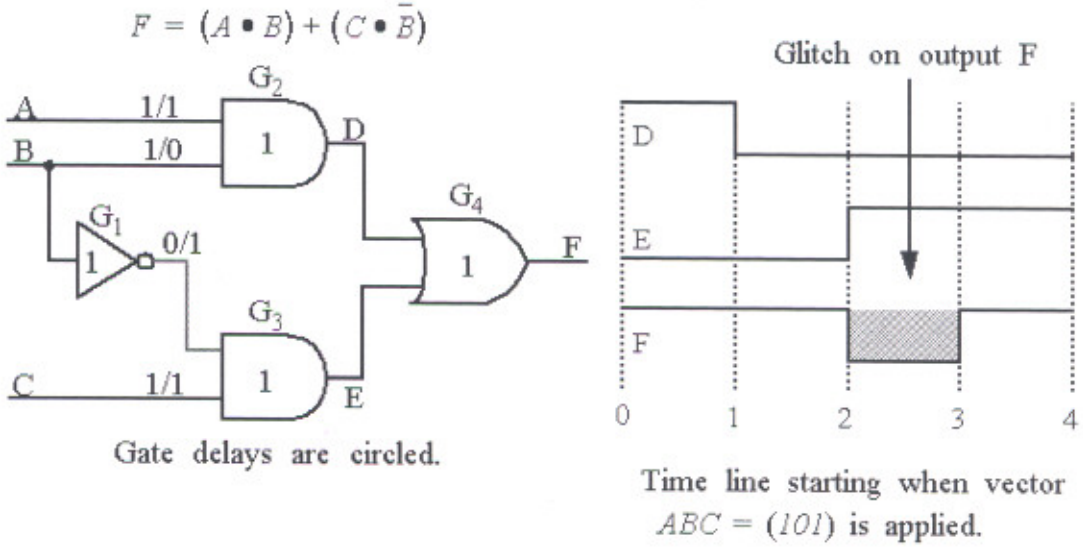


Fig 1.3 **Static Hazards**

Two-vector sequence is ABC = (111), (101).

- Gate G_1 introduces an additional delay of 1 unit.
- Output E of gate G_3 is driven to logic 1, one time unit behind D $\rightarrow$ 0.
- Produces a glitch on F.

Dynamic Hazards

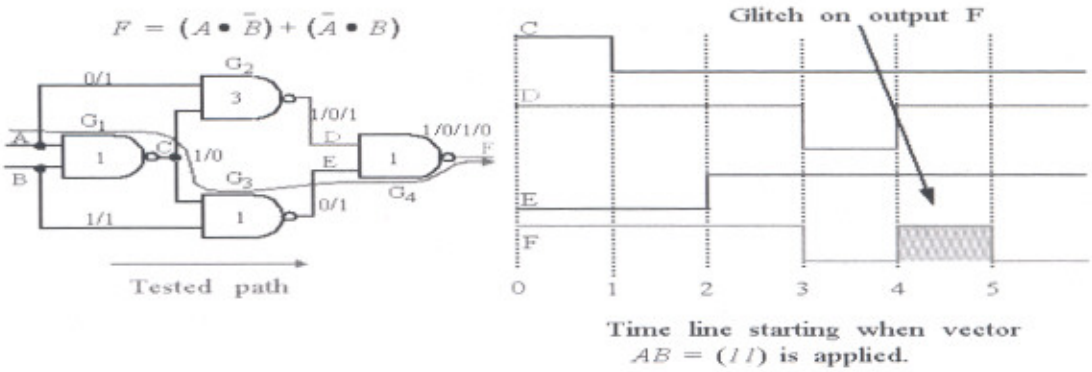


Fig 1.4 **Dynamic Hazards**

Two-vector sequence is AB = (01), (11).

- Gate G_2 has a delay value of 3 time units, due either to a defect or a different physical implementation of the NAND gate.

Hazards and Invalidation

- Static hazards can create dynamic hazards along tested paths and need to be considered during test generation.

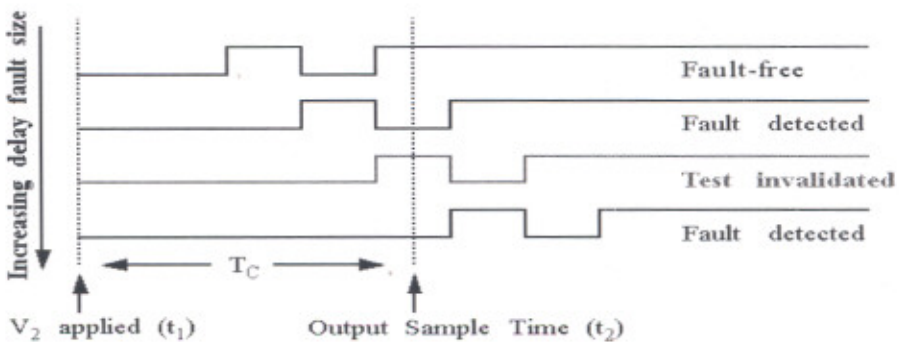


Fig 1.5 **Hazards and Invalidation**

- Note, unlike the previous example, the glitch occurs before the intended transition in this case, and can invalidate the test (e.g. fault is not detected).

Delay Tests and Invalidation

- The critical path(s) of this circuit is 6 time units.
    - Let's set the clock period T = 7.

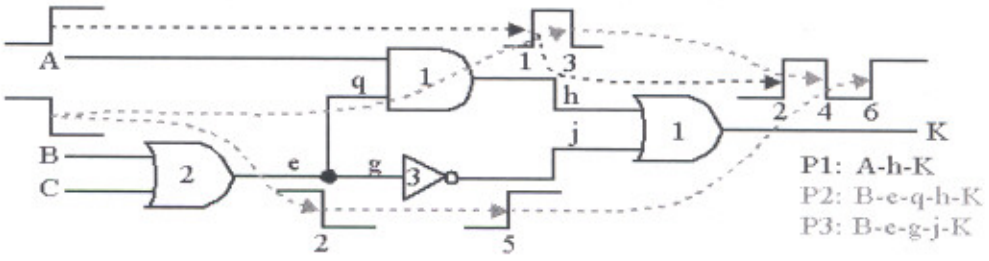


Fig 1.6 **Transitions along Paths**

- Assume only one faulty path.
    - No delay fault is detected if path delay along P3 is less than 7 units.
    - This test will not detect single delay faults along paths P1 or P2.
- Assume there can be multiple faulty paths.
    - Assume P2 and P3 are faulty and P2 extends the "static glitch" at the output beyond 7 units, then it masks P3's delay fault.
- This test is called a non-robust test for delay fault P3.

An understanding of delay fault models is essential in today's VLSI design and test environment. Various fault models used in delay fault testing, their classifications, and coverage metrics are discussed in the following sections. The delay fault models, namely, gate delay, transition, path delay, line delay and segment delay faults, shows their benefits and limitations. The details of these models are written below: -

1.2.1 Path Delay Fault Model

The path-delay fault is an important fault model used in delay testing. The following definitions characterize it.

Definition 1.1 Path-delay fault. The delay defect in the circuit is assumed to cause the cumulative delay of a combinational path to exceed some specified duration. The

combinational path begins at a primary input or a clocked I/P-O/P, contains a connected chain of gates, and ends at a primary output or a clocked I/P-O/P. The specified time duration can be the duration of the clock period (or phase), or the vector period. The propagation delay is the time that a signal event (transition) takes to traverse the path [74]. Both switching delays of devices and transport delays of interconnects on the path contribute to the propagation delay.

For each combinational path in a circuit, there are two path-delay faults corresponding to rising and falling transitions, respectively. These faults for a path consisting of gates a, b, and c are specified as ↑ a – b- c and ↓ a- b-c, where the arrow gives the direction of the transition at the input of the path. The total number of path-delay faults is twice the number of physical paths in the circuit. In general, any combination of paths can be faulty. However, similar to the "single stuck-at" fault model" we consider delay faults of single paths. In practice, though, multiple paths can be faulty.

Definition 1.2 Non-robust path-delay test. A test that guarantees to detect a path-delay fault, when no other path-delay fault is present, is called a non-robust test for that path. A path-delay fault for which a non-robust test exists is called a singly-testable path-delay fault [74]."

A non-robust path delay test applies a transition (two-vectors) at the input of the path and measures the output value after a specified interval (clock period.) For the test to be an effective measure of the path delay, the "expected or correct" output value must be uniquely controlled by the transition propagating through the path. Consider the path-delay fault ↓ P3 shown with bold lines in Figure 1.7.

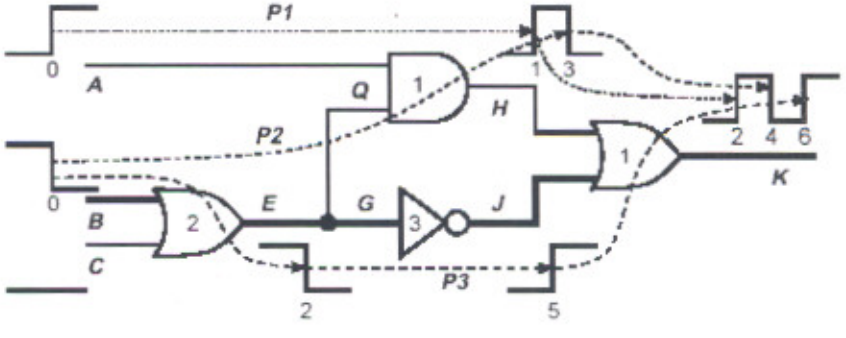


Fig 1.7 **An Example of Transition Propagation through Paths**

Signals B, E, G, J, and K are called the on-path signals. Signals that are not in the path P3 but feed the gates on the path are called off-path signals. Thus, C and H are off-path signals for P3. A non-robust test consists of a vector-pair V_1; V_2, such that:

(1) The change $V_1 \rightarrow V_2$ initiates the appropriate transition at the beginning of the path under test. For example, in Figure 3 the vector-pair (V_1; V_2) = (010; 100) produces a falling transition at B to test the fault $\downarrow$ P3.

(2) All off-path input signals for the path under test assume non-controlling values (0 when feeding into OR/NOR gate, and 1, into AND/NAND gate) in the steady state following the application of the second vector V_2. This condition is known as static sensitization of a path. We may point out that the static sensitization of paths should not be confused with the "static timing analysis," which simply refers to a topological analysis of physical paths without the application of any signals.

In Figure 1.7, transitions are applied to faults $\uparrow$ P1 and $\downarrow$ P3 but static sensitization is achieved only for the latter. Therefore, only the fault $\downarrow$ P3 is non-robustly tested. The fact that the two conditions listed above indeed produces a non-robust test can be easily verified. First, by the definition of non-robust test, only a single path is faulty. Hence, all transitions arriving through other paths ending at the same destination must arrive prior to completion of the clock period (shown as the grey region in Figure 1.8.) This implies that by the end of the clock period, all signals other than the on-path signals of the path under test must be in their steady state. Since the off-path steady-state signals sensitize the entire path under test, the path destination signal is uniquely controlled by the transition propagating through the path. If the path delay exceeds the clock period, then the observed values at the path destination at the end of the clock period will differ from the steady-state output due to V_2, which is the correct expected value. This is illustrated in Figure 1.8.

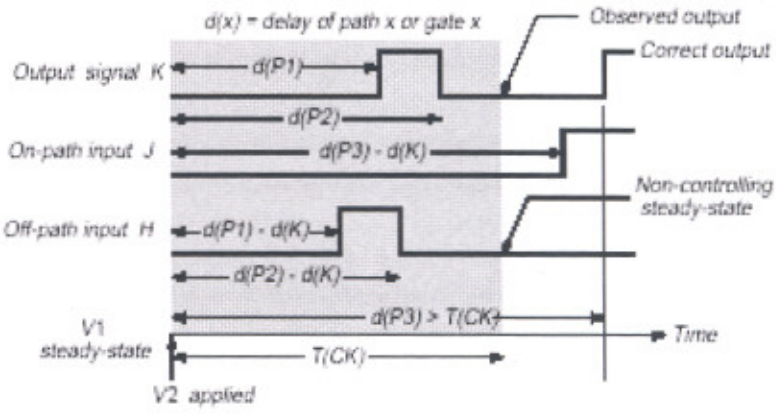


Fig 1.8 **Non-Robust Path Delay Test Output for ↓ P3 being Tested in Figure 1.7**

Consider the example of Non-Robust and Robust Tests:

Non-Robust Test

Figure 1.9 shows a non-robust delay test for the path delay fault ↑ A-B-C. The AND gate has rise and fall delays of one unit each, shown as 1/1. The rise and fall delays of the inverter are 2 units. A vector-pair (0; 1) is derived to satisfy the conditions of a non-robust test and is derived without the consideration of the specific gate delays. The first three waveforms are sketched for the fault-free circuit. The last two waveforms show the signals for a delay fault caused by the inverter delay increasing to 4 units. We notice that the test does not produce a steady-state signal change in the output, which is 0 for all inputs. This logically trivial circuit is a pulse generator whose pulse width is controlled by the inverter delay. If the position and width of the pulse have timing requirements with respect to the clock period, then the delay fault in the inverter path may be important and such a test would be useful. Since this is a non-robust test, it is not guaranteed to work when other paths are faulty. For example, if an additional delay fault ↑ A - C is present (either due to increased routing delay or due to increase in the delay of the AND gate), then the signal C may remain as constant 0. In some delay distributions the output pulse will be produced but will be pushed to the right

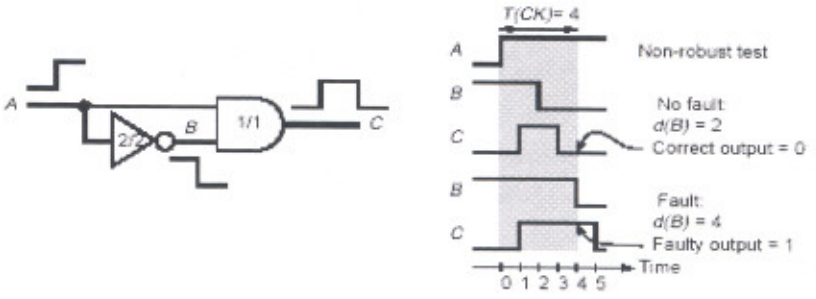


Fig 1.9 **An Example of a Non-Robust Test**

and out of the clock period (grey region.) In either case, the correct logic value 0 will be observed at the end of the clock period. The presence of the fault ↑ A - C, therefore, \invalidates" the non-robust test for fault ↑ A - B - C. We also observe that a non-robust test is not possible for the fault ↑ A-C, because when we apply the rising transition at A, the off-path input B of the AND gate assumes the controlling value 0.

The notion of robust delay test, though implicit in Smith's 1985, was formally defined by Lin and Reddy.

Robust Test

A robust path-delay test guarantees to produce an incorrect value at the destination if the delay of the path under test exceeds a specified time interval (or clock period), irrespective of the delay distribution in the circuit.

Figure 1.10 shows a hypothetical (though typical) output waveform produced by Combinational logic when a vector-pair (V_1; V_2) is applied at the input. If this logic is a part of a clocked sequential circuit, the output value at the end of the clock period T (C_K) is of interest. The initial value (0) is the steady-state output of V_1 and the final value is the steady-state output of V_2. Each transition produced by the vector-pair can potentially propagate through some path and produce a transition at the output at a time determined by the delay of that path. The transitions propagating through paths whose delays are smaller than T (C_K) are shown as "fast transitions" and those propagating through paths with delays greater than T (C_K) are shown as "slow transitions." If the delay of a path increases, the corresponding transition at the output will move to the right. If the delay reduces, the transition will move to the left. When two

neighboring transitions form a pulse, the pulse width equals the difference between the delays of the corresponding paths. If the pulse width is zero or negative (i.e., falling edge arrives earlier for a positive pulse), both transitions will disappear. In other words, the position of an output event is determined by the delay of the path the event travels through, while the existence of the event at the output depends upon the delays of other paths. A robust test that measures the delay of a path should produce an event at the output with following properties:

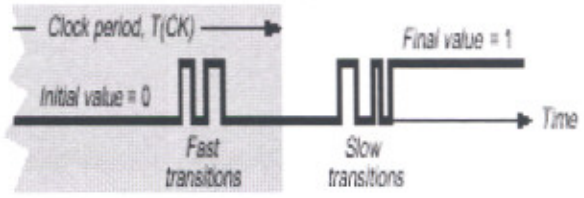


Fig 1.10 **Outputs Events produced by Combinational Logic**

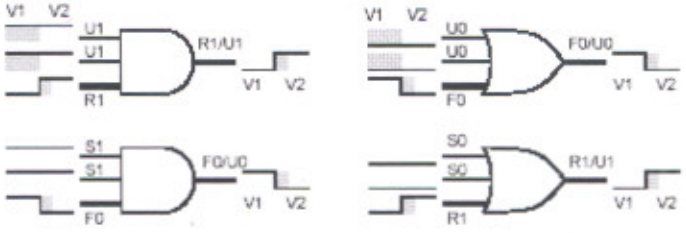


Fig 1.11 **Robust Path Delay Sensitization for Rising and Falling Transitions**

1. It should be a "real event" defined as a transition from the initial value to the final value. This is because a real event can exist without the help of any other event. For a falling transition in Figure 1.10, to appear it must be preceded by another event (a rising transition.) Notice that the falling event at the output in Figure 1.8 is not a real event.
2. It should be a "controlling event." A controlling event permits no other events to appear prior to its own appearance. Thus, the output will remain at the initial value until the controlling event occurs at the output.

Having set the requirements for the event the test must produce at the output, we construct the test by recursively moving backward along the path under test. The on-path

input of the gate contains the source of the output transition. It is a real transition of the same or the opposite type depending on whether or not the gate has an inversion. If the on-path event is a transition from the controlling value to non-controlling value, then it will prevent any output events prior to its own occurrence.

So, there is no specific requirement for off-path inputs in V_1. To ascertain that the output has a real event, all off-path inputs of the gate should have non-controlling value in V_2. When the on-path event is a transition from non-controlling value to controlling value, all off-path inputs must have a steady non-controlling value in both V_1 and V_2. This is because any transition (even a glitch) can be propagated to the output from the off-path input. These conditions are illustrated in Figure 1.11 for AND and OR gates. The reader can easily work them out for other types of gates. The grey regions in waveforms are the times when "don't care" values or transients (glitches) can occur. We notice that glitches are permitted in on-path signals (shown in bold lines.) This is because these are fault detection tests and not diagnostic tests." That means the output will not change from the initial value (due to V_1) during an interval that equals the delay of the path under test. However, an incorrect output at the end of the clock period can also be due to some delayed transition or glitch propagating through off-path inputs.

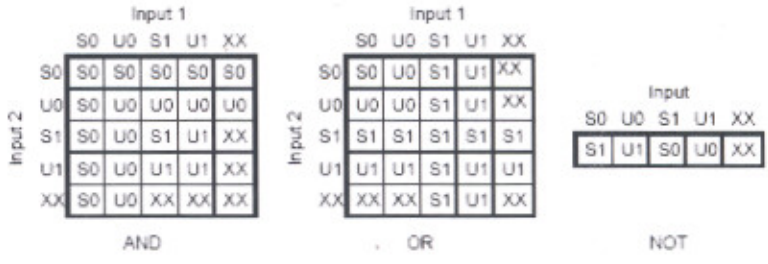


Fig 1.12 **Five-valued Algebra for Path-Delay Test**

The signal values shown in Figure 1.11 are due to Lin and Reddy. S0 and S1 are steady (without glitch) 0 and 1 values for both vectors V_1 and V_2. U0 and U1 specify the final value as 0 and 1, respectively, and leave the initial value as don't care or X. F0 and R1 are falling and rising transitions on the on-path signals. For an off-path signal, F0 and R1 are treated same as U0 and U1, respectively. In addition, XX is used to denote both vectors in the don't care state. The value set (S0, U0, S1, U1, XX) is a five-valued algebra. With

a careful examination, the reader can easily obtain the truth tables for AND, OR, NOT, NAND and NOR gates, the first three of which are shown in Figure 1.12.

Multi-valued algebras have useful applications in delay testing. Bose et al. give a theoretical treatment leading to optimal algebras for specific cases of test generation and fault simulation in combinational and sequential circuits.

Test Generation for Combinational Circuits

Generation of a test for a path-delay fault requires placing the appropriate transition at the origin of the path and justifying the required off-path inputs of all gates on the path. This is easily accomplished using the five-valued algebra.

Robust Test Generation Consider the path-delay fault ↓ P3 in Figure 1.11. We proceed as follows:

1. Place a transition at the path origin, B = F0.
2. Propagate value F0 to line E, from Figure 1.11, C = U0) E = F0.
3. G = F0) J = R1.
4. F0 is interpreted as U0 for off-path logic, Q = U0.
5. Propagate value R1 from J to K, using Figure 1.11 set H = S0) K = R1.
6. Justify H = S0, from Figure 1.12 set A = S0.
7. Test is A = S0, B = F0, C = U0; or V1 = 01X, V2 = 000.

We should remember that the value S0 implies that input A should hold its value steady for two vectors. We observe that this test is different from the one considered in Example 1. This test is robust and the reader can verify that it will not be invalidated irrespective of the delay of P2. The procedure of the above example can be implemented in many ways. The path can be sensitized starting at the output, or all off-path signals can be set at once and then justified. We chose to sensitize the path from input because if sensitization becomes impossible at some gate, then we can immediately conclude that no robust test is possible. This simple example has only limited choices. With increasing number of inputs of a gate, justification choices also increase. In general, when the circuit has reconvergent fanouts, the test generation procedure frequently has to use backtracks.

For some paths, robust tests are not possible and we must generate non-robust tests. As discussed before, non-robust tests only require static sensitization. That means all signals except the origin of the path under test can have arbitrary values in the first vector (V_1). This condition is easily incorporated in the multi-valued algebra by simple substitutions, S0 ← U0 and S1 ← U1.

Non-Robust Test Generation To generate a robust test for path-delay fault ↑ P2 in Figure 1.7. We proceed as follows:

1. Place a transition at path origin, B = R1.
2. Propagate R1 to E, from Figure 1.11 set C = S0.
3. R1 is interpreted as U1 for off-path logic, G = U1) J = U0.
4. Q = R1, Propagate R1 to H, from Figure 1.11 set A = U1.
5. H = R1, Propagate R1 to K, from Figure 1.11, must set J = S0) conflict since J = U0 in step 3.
6. Since no step has any alternatives, a robust test is not possible.

For a non-robust test we change S0 and S1 to U0 and U1, respectively (static sensitization.) Now the Step 5 requirement becomes J = U0, which is consistent with Step 3. The non-robust test is A = U1, B = R1, C = U0 (changed from S0); or V_1 = X0X, V_2 = 110.

An alternative and simpler method for generating non-robust tests is to derive single input change (SIC) tests. For a SIC test, the two vectors V_1 and V_2 in the test differ in exactly one bit. We first find V_2 to statically sensitize the entire path using any combinational ATPG procedure. V_1 is then obtained by just changing one bit in V_2 that corresponds to the origin of the path. It can be easily shown that every non-robustly testable path must have a SIC test.

The procedure of this example attempts to find a non-robust test only when a robust test is impossible. In view of the fact that the reliability of non-robust tests is questionable (see Example of Non Robust Tests), there is merit in finding as many robust tests as possible. The presence of robust tests for some paths can improve the reliability of non-robust tests for other paths. For example, in Figure 1.7 six path-delay faults,

↑ P1, ↓ P1, ↑ P3, ↓ P3, ↑ C - E - G - J - K and ↓ C- E -G -J -K, are robustly testable. Example of non-robust tests shows that ↑ P2 only has a non-robust test. By including the

six robust tests we can ensure that if the circuit passes those, there will be no delayed signal at off-path inputs of the path P2. We can conclude that in the presence of the other four tests, the non-robust test for ↑ P2 is as good as a robust test. Such a test is called a validatable non-robust (VNR) test.

Untestable Path Delay Fault

Consider the path-delay fault ↓ P2 in Figure 1.7. A falling transition (F0) is placed at B and is easily propagated to H by setting appropriate values on A and C. However, a forward implication sets the off-path input of the output OR gate to U1 (i.e., controlling value in V2.) This path-delay fault has no test.

A path for which both (rising and falling) path-delay faults (PDFs) are singly (i.e., non-robustly) testable is called a testable path. A path having one singly testable PDF and one singly untestable PDF is called a partially testable path. When no non-robust test exists for both PDFs of a path, that path is called a singly untestable path. Such a path can be eliminated by circuit transformations that preserve the logic function.

An untestable path is (and a partially testable path may be) associated with one or more redundant single stuck-at faults. The function-preserving transformation such as redundancy removal eliminates such paths.

The fault considered in this Example is on a partially testable path. We observe that the fault Q stuck-at-1 in this path is redundant. Removal of this fault removes the AND gate H feeding input A directly to the OR gate K. This eliminates the path completely. In general, a partially testable path may not have a redundant stuck-at fault.

However, there are procedures for modifying the circuit to expose redundant faults that can be removed. The resulting circuit always has fewer paths, a greater percentage of testable paths, and lower overall delay, but can be larger in size.

A combinational circuit may have paths whose delays cannot affect the time of signal change at the output. These paths are called false paths. The paths of singly untestable PDFs are not always false paths. For example, a singly untestable PDF may be co-sensitized (sensitized simultaneously) with other singly untestable PDFs and the timing of the circuit would be affected if all co-sensitized paths have excess delays. These paths belong to the classes of multiply testable PDFs and functionally sensitizable PDFs. That

is the reason why the delays of paths whose PDFs may be untestable are still taken into account while determining the clock period of the circuit (a point in favor of the static timing analysis.)

1.2.2 Gate Delay Fault Model

Carter introduced a quantitative model for delay faults, known as the gate delay fault. They assume that delays through logic gates are known with some precision. The characteristics (size and location) of likely delay faults are also known. The delays through a gate are represented by intervals in this model. A fault is an added delay of certain size (magnitude); say δ, in the propagation of a rising or falling transition from the gate input to output. The set of faults considered includes numerical delay information. An excessive delay of 3 nanoseconds at a point is not the same fault as an excessive delay of 5 nanoseconds at that point. See in Figure 1.13.

Most of the recent research in this area has concentrated on the determination of fault sizes detected by a given test. Given a particular fault of a fixed known size. *Carter* provides a method to determine whether a test T detects that fault. This is clearly a painstaking and inefficient method, and it would be more desirable to find a certain minimum fault size at a fault site such that given a test T for a fault at the above fault site, T is guaranteed to detect any fault at that site with a magnitude greater than the determined minimum size.

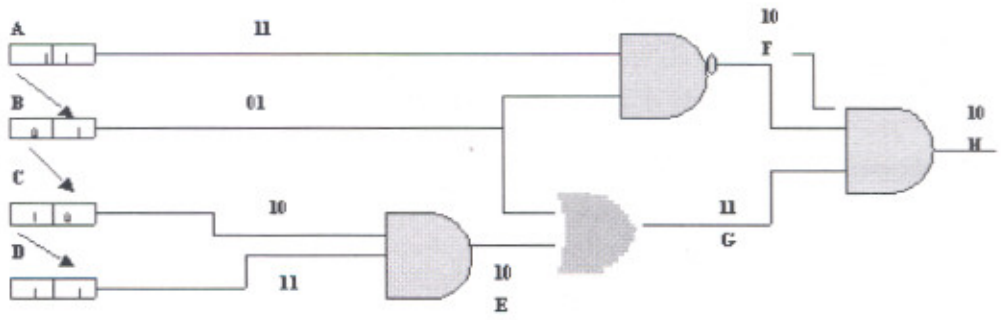


Fig 1.13 **Gate Delay Faults**

- Fault is associated with each gate.

- The transition and path delay fault model may fail to find the input sequence, given in the figure. Is a test, if delay at E is large, it is a test.
- Very time consuming because of too many parameters.

1.2.3 Transition Delay Fault Model

The transition fault model is considered as a logical model for a defect that delays a rising or falling transition at inputs and outputs of logic gates. There are two kinds of transition faults, i.e. slow-to-rise, slow-to-fall. The slow-to-rise (fall) transition fault temporarily behaves like a DC stuck-at-0 (1) fault. A test for a transition fault is a pair of input patterns, one (initialization pattern) to set up the initial state for the transition and another (propagation pattern) to cause the appropriate transition and observe its effect at a primary output. The propagation pattern is identical to a pattern that detects the corresponding DC stuck-at fault. The transition fault coverage is a measure of the effectiveness of the delay test in detecting large delay variations. Transition fault model defects for which the delay is large enough to cause a logical failure when the signal propagates along any path through the site of the fault. The main drawback of this model is the assumption of a large gate delay defect. Also, it is difficult to tell how small a delay fault can be, before it is not detectable. In practice, delay variations tend to be distributed over many circuit elements. Thus, many small gate delay faults, each undetectable as a transition fault, can give rise to a large path delay fault.

1.2.4 Segment Delay Fault Model

Heragu et al. have proposed a model that considers slow-to-rise and slow-to-fall defects on segments [20], whose length L, can be chosen from available statistics about the type of manufacturing defects. L can be as small as 1 (transition faults) or as large as the maximum logic depth (path faults). Once L is chosen, the fault list will comprise of all segments of length L and all paths whose entire length is less than L.

The segment delay fault model is an attempt to combine the advantages of the classical delay fault models while avoiding their limitations. Unlike the path delay fault model, this model can prevent an explosion of the number of faults to be considered. At the same time, a defect over a segment may be large enough to affect any path through it.

This assumption seems more realistic than the transition delay fault model that requires the defect on a single line to be large enough to affect any path passing through it. Due to process variations, every gate in the circuit is affected and their delays increase only by a small amount. Again, randomly occurring defects are of very small sizes. Defects of this nature may produce delay faults only on longest paths. The segment delay fault tests may not be able to detect some of these defects.

1.2.5 Functional Delay Fault Model

In *1995, Pomeranz and Reddy* proposed a functional fault model for delay faults in combinational circuits and described a *functional test generation procedure* based on this model [16]. The proposed method is most suitable when a gate-level description of the circuit-under-test, necessary for employing existing gate-level delay fault test generators, is not available. It is also suitable for generating tests in early design stages of a circuit, before a gate-level implementation is selected. It can also potentially be employed to supplement conventional test generators for gate-level circuits to reduce the cost of branch and bound strategies. A parameter called Δ is used to control the number of functional faults targeted and thus the number of tests generated. If A is unlimited, the functional test set detects every robustly testable path delay fault in any gate- level implementation of the given function. An appropriate subset of tests can be selected once the implementation is known. The test sets generated for various values of Δ are fault simulated on gate-level realizations to demonstrate their effectiveness.

1.2.6 Line Delay Fault Model

We can combine the relevant features of the transition and path delay fault models to define a line delay model. A rising (falling) line delay test will test the longest sensitizable path passing through a target line producing a rising (falling) transition on it. With this model, the coverage is measured for all lines with two possible transitions. Thus, the maximum number of faults (or tests) is twice the number of lines. (For example, in c6288, we will consider only 12576 line delay faults whereas the total number of possible path faults is $\approx 1.98 * 10^{20}$.) Yet, the test criterion is similar to path delay fault, and not like gate or transition delay fault. In general, a test will cover several

lines. Conventional path delay test generators attempt to derive robust tests for a subset of paths in the circuit, based on some path selection criterion such as the worst-case path selection or a threshold-based path selection. However, a large number of these paths may not be robustly testable and hence the test coverage of the targeted paths can be very low. The new *coverage metric* [37] seeks to remove this deficiency by attempting to derive a pair of line delay tests for each line in the circuit.

The basic idea of an iterative approach for generating a robust test was first proposed by *Park and Mercer*. They have followed an approximate method where the search space for test generation is biased to and a test along a path whose propagation delay is greater than or equal to a predefined threshold value. *Bose* preselects paths in a given range of lengths and shows that despite low path coverage, a high gate (or line) coverage can be obtained. In that case, however, lines are not tested through longest sensitizable paths. The method of *Majhi* et al., on the other hand, is an exact method for generating a robust test for the longest testable path through each line. To facilitate the simultaneous consideration of robust and non-robust tests, they use a 9-value logic system.

A *line delay coverage metric* [37] proposed by *Majhi and Agrawal* in *1997*. The motivation of defining the line delay test is to robustly detect the smallest incremental delay defect associated with a rising or falling transition at any line. Suppose, Δ_L is the incremental delay of a rising or falling transition through line L. Then, for detection of this delay fault,

$$\Delta_L + T_P > T_C \quad \text{or} \quad \Delta_L > T_C - T_P \qquad (1)$$

Where T_C = system clock period and T_P = nominal delay of the path P through which L is tested. From relation (1), we determine that the smallest incremental delay fault on L can be detected via the path through L having the longest nominal delay $(T_P)_{max}$, i.e.,

$$(\Delta_L)_{min} = T_C - (T_P)_{max} \qquad (2)$$

By sensitizing the longest path through L, we are able to detect the delay fault of the smallest size. However, simultaneous delay variations are possible for other gates on P due to correlation with L. Suppose, delays of other gates increase. Then the line delay test for L will detect a delay fault of even smaller size. If the delay of other gates reduces

while that of L increases, the sensitivity of the test reduces. Considering correlation of delays, this later case is less probable.

The basic assumption associated with the line delay fault model is that the delays of all gates are not reduced below their nominal values. Main advantages of this fault model are that the number of faults is limited to twice the number of lines in the circuit and almost all lines can be tested. Since the fault is tested along the longest propagation path, the system timing failures caused by the smallest localized delay defects or the accumulation of distributed delay defects can be detected. In transition fault model, a delay test is obtained along any arbitrary path because the size of delay fault is assumed to be large enough to be tested via any path through the fault site. For the gate delay fault model one must specify exact sizes of delay defects. Difficulties arise when accurate information on delays is not available. Transition and gate delay faults do not model the distributed delay defects along a target path. The line delay model, on the other hand, retains many advantages of the transition and gate delay fault models, while alleviating the major drawback of the path delay model (viz., too many paths to be tested and the low fault coverage).

In order to derive a line delay test for a given line, we first target the longest structural path. If that path is not robustly testable then we go for the next longest path and soon, until we find a test for the longest robustly testable path. The limitation in this approach is that in addition to this robustly testable path there could exist a nonrobust test for some longer path through the target line. In such a case, the robust test would miss a line delay fault that only causes the longer nonrobustly testable path to fail. To overcome this limitation one may include any possible nonrobust tests for all paths that are longer than the longest robustly testable path. Another limitation of this model is that in case of certain distributed delay defects the derived tests will fail to detect some of the delay faults that are not targeted. We consider only one path through any given line for determining a line delay test. However, there may be some other paths of the same length (or shorter) through the target line, with distributed delay defects exceeding the permissible propagation delay. Consider the three paths shown in Figure 1.4. Suppose, all three paths have the same delay. Let us assume that paths 1 and 2 are the longest structural paths selected for testing lines A and B, respectively. Let us further assume that

the smallest incremental delay that is detectable for each path is Δ (i.e., $\Delta = T_C - T_P$), where T_P is the nominal delay of each of the paths and T_C is the clock period. If the incremental delay of nodes A and B are $\Delta - \varepsilon$, where ε is small, then paths 1 and 2 will pass their tests. However, path 3 has a fault that is not detected by the test vectors though it is a detectable delay fault.

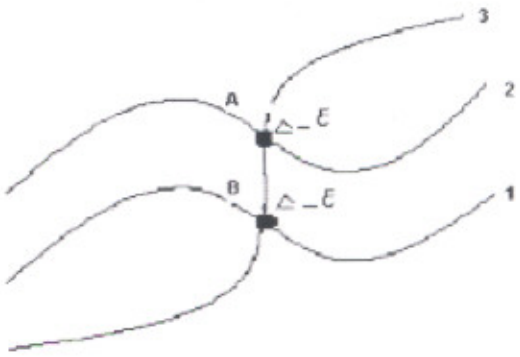


Fig 1.14. **Line Delay Faults Limitation**

The basic assumption associated with the line delay fault model is that the delays of all gates are not reduced below their nominal values. More than one faulty gate can occur in practice due to correlation between delays of gates. In that case many gates in paths 1 and 2 will have increased delays and it is more likely that the tests for those faults will show failures. There can be several ways of dealing with the situation depicted in Figure 1.14. When there are several longest paths of equal length through a target line, we can consider all such paths to increase the confidence level for the tests obtained. However, this can lead to a potentially large number of paths to be tested in some circuits.

1.3 Hazard Algebra

The two elements Boolean algebra, B_2, is the standard algebra for circuit analysis and design. However, it is unable to directly detect hazards. Several other algebras have been proposed for hazard detection. Certain of these hazard algebras can only represent static hazards, while others are capable of representing dynamic hazards. Little attention has been payed to the mathematical structure of these hazard algebras. There are several hazard algebras and examines their completeness and usefulness for hazard detection.

The first hazard algebra was the three-valued ternary algebra, T_3, introduced

by Goto in 1949. It included a value to represent an unknown circuit state. A two-pass algorithm for detecting static hazards using ternary algebra was developed by Eichelberger. Metze described a four-valued algebra, M_4, for use in the design of switching circuits. It had two non-binary values 0/1 and 1/0, which were introduced to represent transitions. A five - valued algebra, L_5, was presented by Lewis. It includes a value representing an unknown state and two values representing clean transitions. Lewis gives an algorithm for detecting static and dynamic hazards, and race conditions. The algebra is unable to directly distinguish which of the many types of hazards has occurred. Fantauzzi gave a nine-valued algebra, F_9, with elements representing each of the static and dynamic hazards and an unknown value.

Muth presented a nine-valued algebra for test generation. It's elements represented pairs of values, one from a good circuit and one from a faulty circuit. Hayes presented two methods for producing new algebras from old ones. Using these methods, he showed how to construct T_3 and L_5 from B_2. In the same paper Hayes introduced the idea that each value in the algebra should represent an initial state, a transition and a final state. We follow the nomenclature used by Hayes, with each algebra represented by the first letter of the last name of the author who suggested it, subscripted with the number of elements in the algebra. Using his generation methods, Hayes constructed a new six-valued algebra, H_6. He showed that this algebra is useful in the analysis of static hazards. Hayes also presented an eight-valued algebra, H_8, for detecting and distinguishing both static and dynamic hazards. By adding an unknown value to this algebra and taking the closure, he also showed a thirteen-valued algebra, H_{13}, for complete simulation. Breuer and Harrison proposed a twenty-seven valued algebra for eliminating static and dynamic hazards in test generation. It uses a modified D-algorithm to produce hazard-free tests on sequential circuits.

Here is the brief introduction of the laws, which describe the properties of many of the hazard algebras, the mathematical and hazard detecting properties of each of the hazard algebras. The algebras are reviewed in ascending order by number of elements. It also includes the generating new algebras by the methods of Hayes.

1.3.1 Laws of Hazard Algebras

For a general algebraic system P=(A, +, *, ¯, 0,1, Φ) where A is a set of elements, + and * are binary operations on A, ¯ is a unary operation on A and 0,1 and Φ are constants in A, the following laws have been used to describe a variety of algebraic structures which occur in the study of hazard algebras.

	For all $a,b,c \in A$:	
Idempotence	T1 $a+a=a$	T1' $a*a=a$
Commutativity	T2 $a+b=b+a$	T2' $a*b=b*a$
Associativity	T3 $a+(b+c)=(a+b)+c$	T3' $a*(b*c)=(a*b)*c$
Absorption	T4 $a+(a*b)=a$	T4' $a*(a+b)=a$
Identity	T5 $a+0=a$	T5' $a*1=a$
Bounding	T6 $a+1=1$	T6' $a*0=0$
Distributivity	T7 $a+(b*c)=(a+b)*(a+c)$	T7' $a*(b+c)=(a*b)+(a*c)$
Involution	T8 $\overline{(\overline{a})}=a$	
De Morgan's laws	T9 $\overline{(a+b)}=\overline{a}*\overline{b}$	T9' $\overline{(a*b)}=\overline{a}+\overline{b}$
Complement laws	T10 $a+\overline{a}=1$	T10' $a*\overline{a}=0$
Ternary laws	T11 $(a+\overline{a})+\Phi=a+\overline{a}$	T11' $(a*\overline{a})*\Phi=a*\overline{a}$
Self-Complementation	T12 $\overline{\Phi}=\Phi$	

Fig 1.15 **Laws pertaining to Hazard Algebras**

A *semigroup* is a system S=(A, +) where T3 is satisfied.

A *bisemigroup* is a pair of *semigroups* S_1=(A, +) and S_2=(A, *) which have the same element set A.

A *semilattice* is a semigroup where T1and T2 is satisfied.

A *bisemilattice* is a pair of semilattices S_1=(A, +) and S_2=(A, *) which have the same element set A.

A *lattice* is a bisemilattice such that T4 and T4' are satisfied.

A *lattice* is said to be bounded if T5, T5', T6 and T6' hold.

A *lattice* is said to be distributive if T7 and T7' hold.

A *De Morgan Algebra* is a bounded, distributive lattice such that T8, T9 and T9' hold.

A *De Morgan Algebra* is a Boolean algebra if T10 and T10' hold.

A *De Morgan Algebra* is a ternary algebra if T11, T11' and T12 hold.

A binary relation on A which is reflexive, antisymetric and transitive is called a partial order on A. an ordered pair (A, ≤) such that A is a set and ≤ is a partial order on A is called a poset, short for partially ordered set. For a pair of elements a and b in A we say that c in A is the *least upper bound* of a and b if a ≤ c, b ≤ c and for any d such that a ≤ d

and $b \leq d$, we have $c \leq d$. if the least upper bound exists for a pair of elements a and b, then it is unique.

1.3.2 The Three-valued Algebra

The ternary algebra of Goto and Eichelberger has three values, 1,0 and X. 1 and 0 represent logic 1 and logic 0 respectively, while X represents an unknown value. Eichelberger gave an efficient two-pass algorithm for detecting static hazards. In 1986, this algorithm was shown to be correct by Brzozowski and Seger. The operations of the three-valued algebra are shown in Figure 1.16(a). As it's name suggests, the three-valued algebra is a ternary algebra. Idempotence, Commutativity, absorption, identity, bounding, involution and the self -complementation of X are all immediate from Figure 1.16(a). Exhausting all possible cases can rapidly check all associativity, distributivity, De Morgan's Laws and ternary laws.

To verify that the three-valued algebra is not a Boolean algebra, simply note that $X + X' = X+X = X \neq 1$, which violates the complement law. The partial order for the three-valued algebra is given in Figure 1.16(b).

+	0	X	1
0	0	X	1
X	X	X	1
1	1	1	1

·	0	X	1
0	0	0	0
X	0	X	X
1	0	X	1

a	$\bar{a}$
1	0
X	X
0	1

Fig 1.16(a) **Operations of the Three-valued Algebra**

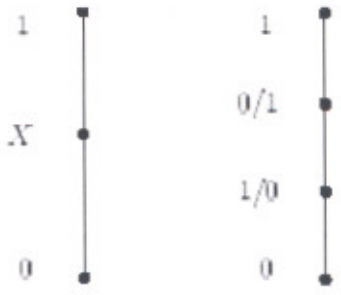


Fig 1.16 (b) **Partial Orders for M_4 and T_3**

Three-valued algebra is an excellent tool for detecting static hazards and its theory and applications have been studied. Brzozowski, Lou and Negulescu gave a characterization of finite ternary algebras as subset-pair algebras.

Eichelberger introduced a ternary simulation method for detecting static hazards. The method consists of two algorithms, A and B. it makes the assumption that the circuit begins in a stable state, though Brzozowski and Seger later generalized these algorithms to a method that does not require the initial state to be stable.

Algorithm A uses the uncertainty partial order given below.

$$0 \leq X, 1 \leq X, 0 \leq 0, 1 \leq 1, X \leq X \qquad (1)$$

This partial order puts values with more uncertainty higher in the partial order, for instance, X, being a more uncertain value than 1 is above it. At each step in the algorithm, the output of a gate is set to the least upper bound under the uncertainty partial order of the excitation of the gate and its current output. Hence any gate undergoing a change between the two Boolean values would have its output set to X, since X=*lub* (0,1). Since the output of a gate can only remain at its current value or change to X, this procedure is guaranteed to terminate in n+1 steps, where n is the number of outputs of gates in the circuit.

Algorithm B begins with the output of Algorithm A and repeatedly sets the output of each gate to the value of its excitation. If algorithm B reaches a stable state and the simulated value of an output is X then that output could have either a value 0 or a value of 1, depending on the delays in the circuit. A static hazard is detected if an output has identical values in the initial state of Algorithm A and the final state of Algorithm B, but has value X in the final state of Algorithm A. Brzozowski and Seger proved that Eichelberger's algorithm was equivalent to the General-Multiple Winner method.

1.3.3 The Four-valued Algebra

The four-valued algebra of Metze consists of a set A = {0, 1/0, 0/1, 1}, two binary operations + and * with identities 0 and 1, respectively, and the unary operation ¯. Figure 1.17 shows the entries for operators ¯, + and * on the four-valued algebra, Metze notes that the algebra is closed under these operations.

+	0	1/0	0/1	1
0	0	1/0	0/1	1
1/0	1/0	1/0	0/1	1
0/1	0/1	0/1	0/1	1
1	1	1	1	1

*	0	1/0	0/1	1
0	0	0	0	0
1/0	0	1/0	1/0	1/0
0/1	0	1/0	0/1	0/1
1	0	1/0	0/1	1

a	$\overline{a}$
0	1
1	0
0/1	1/0
1/0	0/1

Fig 1.17 **Operations of the Four-valued Algebra**

Metze observed that the four-valued algebra is idempotent, commutative and associative. Hence, it is a bisemilattice, however the term bisemilattice was not invented until 1971, so Metze did not use that nomenclature. Metze further observed that the distributive, involution and De Morgan's laws hold. By inspection of Figure 1.17 we can immediately see that Metze's algebra follows the absorption, identity and bounding laws. Hence the four-valued algebra is De Morgan Algebra.

The following counter example to the complement law shows that Metze's algebra is not Boolean algebra.

$$0/1 + \overline{0/1} = 0/1 + 1/0 = 0/1 \neq 1 \qquad (2)$$

Since the four-valued algebra does not have a self-complementary element, it is also not a ternary algebra.

Metze believed that hazards were due, at least in part, to the simplification of Boolean expressions. He writes

For feedback-free circuits, the problem is usually one of the "simplifying" a given Boolean Function. Noting that valid simplifications in Boolean algebra, which introduce hazards in the related circuit, are sometimes not valid in logic algebras with more than two elements. Metze's algebra is a least as powerful as T_3 since the homomorphism Φ: $M_4 \rightarrow T_3$, defined in (2), is surjective.

$$\phi(1) = 1 \quad \phi(0) = 0 \quad \phi(0/1) = \phi(1/0) = X \qquad (3)$$

This homomorphism preserves the algorithm for static hazard detection that was proved correct for T_3. The partial order for the four-valued algebra is given in Figure 1.16(b). Under the interpretation that Metze gives, 0/1 represents "the transition of state 0 to state 1 of the two-valued switching algebra." that is, a 0 to 1 transition. Similarly 1/0

represents a 1 to 0 transition. This interpretation seems to be inconsistent with the structure of the algebra. Having a 0 to 1 transition and a 1 to 0 transition on the inputs of an OR gate should give a static 1-hazard, but instead 0/1 + 1/0 =0/1, a 0 to 1 transition. Metze suggested using sequences of values to detect hazards. If the output of a gate yields one of the following sequences: ...1,1/0,0... or... 0,0/1,1... Then a transition without hazards has occurred, but if instead one of the following sequences occurs then a hazard is present.

$$\begin{array}{lll} \ldots,1,0/1,\ldots & \ldots,0/1,0/1,\ldots & \ldots,1/0,1,\ldots \\ \ldots,0,1/0,\ldots & \ldots,1/0,1/0,\ldots & \ldots,0/1,0,\ldots \end{array}$$

Metze states that the sequences in the first column represent hazards. However if we use the method given by Eichelberger for ternary simulation, M_4 correctly detects the static hazards.

1.3.4 The Five-valued Algebra

Lewis introduced a five-valued algebra with values representing 0, 1, a 0 - to -1 transition without hazards, a 1 - to - 0 transition without hazards and an unknown value. He denoted these 0,1, 0/1, 1/0 and X, respectively. Figure 8 shows the entries for operations +, * and ¯ on the five-valued algebra.

Lewis claimed that his algebra detects all hazards and races in both combinational and sequential circuits. He gave a detailed algorithm for doing this. The five-valued algebra is not a lattice, since the absorption law does not hold. A counter example is

$$0/1 * (1/0 + 0/1) = 0/1 * X = X \neq 0/1 \qquad (4)$$

+	1	X	1/0	0/1	0
1	1	1	1	1	1
X	1	X	X	X	X
1/0	1	X	1/0	X	1/0
0/1	1	X	X	0/1	0/1
0	1	X	1/0	0/1	0

*	1	X	1/0	0/1	0
1	1	X	1/0	0/1	0
X	X	X	X	X	0
1/0	1/0	X	1/0	X	0
0/1	0/1	X	X	0/1	0
0	0	0	0	0	0

x	$\overline{x}$
1	0
0	1
X	X
0/1	1/0
1/0	0/1

Fig 1.18(a) **Operations of the Five-valued Algebra**

Distributivity does not hold in L_5 either. The following is a counterexample.

$$0/1 * (1/0 + 1) = 0/1 * 1 = 0/1 \neq X = X + 0/1 = (0/1 * 1/0) + (0/1 * 1) \quad (5)$$

The laws of the five-valued algebra, given in Figure 1.18(a), are those for a DeMorgan bisemilattice. The partial order for the five-valued algebra is given in Figure 1.18(b). De Morgan bisemilattices were studied by Brzozowski and he provided a set-theoretic characterization for locally distributive De Morgan bilattices.

Lewis developed the five-valued algebra from Metze's four-valued algebra and the three-valued ternary algebra. M_4 had no element to represent signals for which "neither the actual value ("0" or "1") nor the direction of a transition (if there is one) is known", such as race conditions. T_3 had no values to represent transitions without hazards and so could not differentiate between these and dynamic hazards. To be able to model these situations, Lewis introduced the value X into Metze's algebra. He also changed the operations so that 0/1 and 1/0 would only represent transitions without hazards. As with T_3, the value X represents both an unknown signal and any hazard. However, unlike in T_3, X does not represent a transition without hazards, as these are represented by the values 0/1 and 1/0.

The other major change that Lewis made with his algebra is to model device delays and incorporate these into his hazard and race condition algorithm. Lewis used the bounded delay model, where each device has a minimum and maximum number of time steps of delay.

After introducing his algebra Lewis examines the hazard detection capabilities of T_3 and M_4. He concluded, " The three symbols of the ternary algebra offer no distinction between hazards, race conditions, and simple transitions." And so "the algebra is unsuitable for hazard detection by simulation". This conclusion seems unfounded since there is an algorithm for the detection of static hazards using the three-valued ternary algebra, which has been proven correct. Lewis identified the inability of M_4 to represent race and unknown conditions as a motivation to improve upon it.

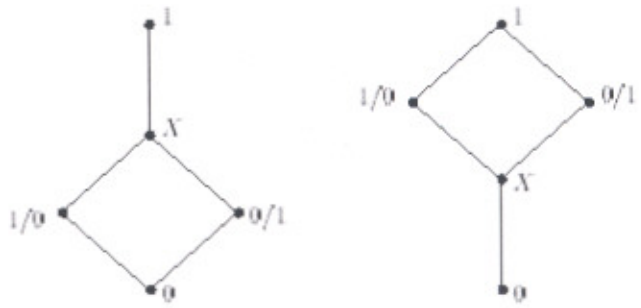


Fig 1.18(b) **Partial Order for L_5**

Lewis gave a quinary simulation algorithm for detecting hazards and races. He first described a method for evaluating the output of a feedback-free device with n inputs using the associative law. Second, Lewis gave a generalized model of a flip-flop and a flow chart for evaluating the output of such a device. Third, Lewis described a queue like structure for calculating the effects of the bounded delay model on the output. The final part of the algorithm is the general control algorithm, which calls the three previous parts as subroutines.

There are two reasons for which this proof cannot be taken as a guarantee that L_5 does infact, detect all hazards and races. The first is that Lewis has not included two sequences, which Metze has taken to represent hazards. The sequences ...0/1,1/0... Or... 1/0,0/1...occur where a hazard is generated in the simulation of a re-cycling counter. He second reason is that Metze has not proved the correctness of his methods for hazard detection and so any proof based on those methods is not rigorous.

1.3.5 Hazard Algebra Generation

Hayes described two methods for generating new hazard algebras from previous ones. He called them the Product Set method and the uncertainty Set Method. We digress to examine these methods here because the rest of the hazard algebras examined are constructed using them.

The Product Set method creates a set A of elements by taking the cross product of the sets of elements A_1, A_2, ... A_n from several algebras, and defining the operations on A coordinate-wise from the operations on the other algebras. The set of algebras that

is used in the creation is called basis set of the new algebra.

This is the method that was used to create H_6, H_8 and H_{13}. A law holds for an algebra created in this manner if and only if the law holds for every algebra in the basis set.

There are several interpretations for the Product Set method. Hayes discussed applications in fault simulation, where an ordered pair of values could have the first coordinate representing the state if a fault was present. In the context of hazard detection, changes in the signal are the information we wish to track. Interpreting the coordinates as a chronological sequence of signals seems to be appropriate. The three algebras constructed using the Product Set method each have three coordinates representing the signal at an initial, intermediary and final time.

The second algebra creation method given by Hayes creates a set of elements by taking the power set of the original algebra. Each of the sets in the power set, or a subset thereof, is an element in the new algebra. With A' c 2^A, an n-ary operation Φ from A can also be defined on A' by

$$\Phi(a'_1, \ldots, a'_n) = \cup_{a \in a'_1 \times \ldots \times a'_n} \{\Phi(a)\} \qquad (6)$$

This method is called the Uncertainty set method, because it has an interpretation involving values of uncertain signals. An element, $a_i' = \{a_1, a_2, \ldots\ldots a_n\}$ of A' can be thought of as representing a signal in the circuit that is one of $a_1, a_2, \ldots. a_n$ but it is uncertain which.

An example of this method is the generation of T_3 from B_2. The value X in the three-valued algebra represents a signal that is in either state 0 or state 1, but which of the two is unknown. Thus, X can be represented as {0,1}. Hence we can generate T_3 from B_2 in the following manner. We note that

$$\mathbf{T_3} \cong \{\{\mathbf{0}\}, \{\mathbf{1}\}, \{\mathbf{0}, \mathbf{1}\}\} \subset \mathbf{2^{B_2}} \qquad (7)$$

and define the operations on T_3 from those on B_2 as shown in Equation 6. Hayes similarly shows that $\mathbf{L_5} \subset \mathbf{2^{B_2^2}}$.

This method obviously does not have the nice properties of the product Set Method, as T_3 follows a different set of laws than B_2.

1.3.6 The Six-valued Algebra

The six-valued algebra of Hayes consists of a set of ordered triples A = {(0,0,0), (0,X, 0), (0,X, 1), (1,X, 0), (1,X, 1), (1,1,1)}, two binary operations + and * with identities (0,0,0) and (1,1,1) respectively, and the unary operation ¯. The first and third coordinates of the triple are from B_2 and the second coordinate is from T_3. The operations +, * and ¯ are defined coordinate wise by the corresponding operations from B_2 and T_3. The set A c B_2 * T_3 * B_2 is closed under these coordinate wise operations.

The structure of the six-valued algebra and the laws it follows are much easier and faster to discover than those of the four-valued algebra due to the construction of the six-valued algebra using the Product Set method.

It follows immediately from this that six-valued algebra is De Morgan algebra, but not a Boolean or ternary algebra, since B_2 is not a ternary algebra, T_3 is not a Boolean algebra, but both are De Morgan algebras. The partial order of the six-valued algebra is given in Figure 1.19.

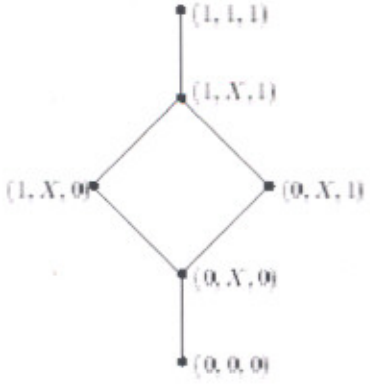


Fig 1.19 **Partial order for H_6**

The six-valued algebra realizes the idea that a hazard is the possibility of a transient signal between two static signals. The static signals are represented by the first and third coordinates of H_6 and the transient signal is represented by the second coordinate. Hayes was the first to put forward this model and makes use of its explanatory power in the eight-valued algebra as well.

Hayes noted that six-valued algebra explains the model used by many three-valued simulators. Users of these simulators "supply binary input sequences…the simulator

automatically inserts the third value X between 0 and 1 in any 0-to-1 or 1-to-0 transition it encounters in the user-supplied sequences". Unlike any of the smaller algebras, H_6 has distinct values for representing static 0- hazards and static 1-hazards. This leaded Hayes to conclude, "H_6 represents the smallest algebra for analysis of static hazards".

1.3.7 The Eight-valued Algebra

The eight-valued algebra of Hayes consists of a set of ordered triples A ={(0,0,0), (1,1,1,), (0,0/1,1), (1,1/0,0), (0,X, 0), (1,X, 1), (0,X, 1), (1,X, 0)}, two binary operations + and * with identities (0,0,0) and (1,1,1,) respectively, and the unary operation $\bar{}$. The first and third coordinates of the triple are from B_2 and the second coordinate is from L_5. As with the six-valued algebra, the operations +, * and $\bar{}$ are defined coordinatewise by the corresponding operations from B_2 and L_5 and the set A is closed under these coordinate-wise operations. The partial order for the eight-valued algebra is given in Figure 1.20.

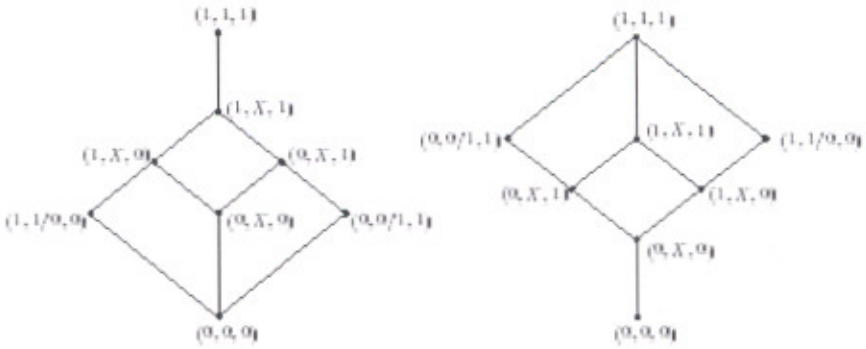


Fig 1.20 **The Partial Orders for H_8 under + and ***

B_2 is a Boolean algebra, and thus a De Morgan bisemilattice. L_5 is a De Morgan bisemilattice, but is neither a lattice, nor is it Boolean. It is therefore ensure that H_8 is De Morgan bisemilattice. Just as H_6 is the smallest algebra for analysis of dynamic hazards. Hayes explained the reason that L_5, though capable of detecting all hazards, cannot analyze them. For the general element of H_8, (a_1, a_2, a_3), Hayes remarked that the knowledge of a_2 alone is insufficient to identify a hazard condition; the associated static values a_1 and a_3 must also be known. Indeed we note that the four values of H_8 with

second coordinate X each correspond to a hazard, and their static original and final states clearly identify the hazard to which each value corresponds.

1.3.8 The Nine-valued Algebra

Two nine valued algebras have been introduced. One by Fantauzzi for hazard detection and one by Muth for test generation.

1.3.8.1 Fantauzzi's Algebra

In 1974, a nine-valued algebra, F_9, was introduced by Fantauzzi. This algebra has an eight-valued subalgebra isomorphic to H_8 in which pairs of values of values represent Boolean states, hazard-free transitions, and static hazards and dynamic hazards. To this, Fantauzzi added a value to represent an unknown state. As the notation used by Fantauzzi differs significantly from the notation of Hayes used throughout this section, Figure 1.21 shows which values of Fantauzzi correspond to which values of Hayes, and the state to which those values corresponds. We will continue to use the notation of Hayes for clarity. Hayes argued that the set $\{H_8, X\}$ is not closed under the operations * and +. When F_9 is viewed as 3 tuples it is easy to see that is indeed the case. While (1,X, 0) *(X, X, X) should give (X, X, 0) this value is not among those in F_9. This leads to a variety of difficulties. The * and + operations on F_9 are not associative or distributive and the absorption law does not hold. To deal with these problems, Fantauzzi introduced a different pair of operations to represent gates with more than two inputs. This did not solve the difficulty, as two AND gates can still be arranged so that associativity occur in a circuit, but did not occur under Fantauzzi's model.

The failure of the associative law guarantees that F_9 is not a semigroup under the binary operations given and so is not any of the other structures described as above.

Fantauzzi Value	Hayes Value	State
F	$(0,0,0)$	Binary zero
$\uparrow$	$(0,0/1,1)$	Hazard-free 0-to-1 transition
$\downarrow$	$(1,1/0,0)$	Hazard-free 1-to-0 transition
$\cap$	$(0,X,0)$	Static 0-hazard
$\cup$	$(1,X,1)$	Static 1-hazard
/	$(0,X,1)$	Dynamic 0-to-1 hazard
\	$(1,X,0)$	Dynamic 1-to-0 hazard
*	(X,X,X) [1]	Unknown state or race

Fig 1.21 **Values of F_9**

1.3.8.2 Muth's Algebra

The nine-valued algebra of Muth M_9, is a Product Set Algebra such that $M_9 = T_3 * T_3$. Since T_3 is a ternary algebra, it is immediate ensure that M_9 is a ternary algebra. The first coordinate represents the value of the signal in a good circuit, while the second coordinate represent the value in a faulty circuit.

This algebra leads to a generalization o the D-algorithm for test generation, permitting faults with multiple or repeated effects to be correctly handled.

1.3.9 The Thirteen-valued Algebra

The last hazard algebra which Hayes introduced consists of a set A $\square$ $T_3 * L_5 * T_3$ and A is the closure of H_8 U {X} under coordinatewise * and +. As with the other product algebras, all operations are defined coordinatewise and the identities are (0,0,0) and (1,1,1) for * and + respectively. Chakraborty, Agrawal and Bushnell seem to have discovered the thirteen-valued algebra independently.

The laws for a ternary algebra are a superset of those for a De Morgan bisemilattice. The three-valued algebra is a ternary algebra and the five-valued algebra is a De Morgan bisemilattice. So thirteen valued algebra is also a De Morgan bisemilattice. A verification of these facts is given by Brzozowski. The partial order under + for thirteen valued algebra is shown in Figure 1.22.

The thirteen-valued algebra can represent all of the states represented by the eight-valued algebra. In addition, there are values to represent a completely unknown signal, (X, X, X), a signal starting at 0 and becoming unknown, (0,X, X), a signal starting at 1 and becoming unknown, (1,X, X), a signal which begins in an unknown state and changes to 0, (X, X, 0) and a signal which begins in an unknown state and changes to 1, (X, X, 1).

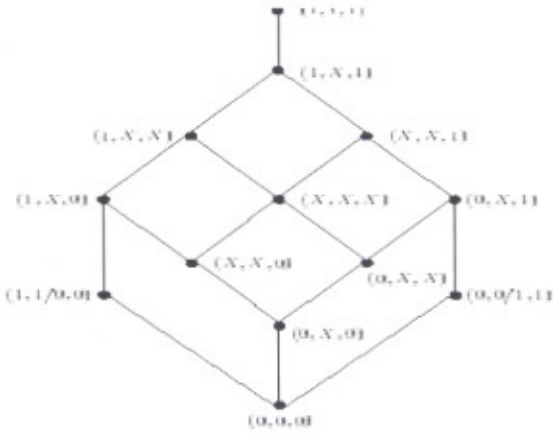


Fig 1.22 **The Partial Order under + of H_{13}**

Chakraborty, Agrawal and Bushnell gave an alternate method of constructing the thirteen-valued algebra. They examined all triples in T_3 * T_3* B_2. The first coordinate indicates the initial signal, the second coordinate represents the final signal and third coordinate indicates whether or not a hazard occurred in the transition. They reduced the number of elements from eighteen to thirteen by noting that for signal with unknown initial or final states, distinguishing hazards in not important and so any pair of signals (a, b, 0) and (a, b, 1) are identified if either a or b or both is X.

This algebra seems to be most complete of the algebra, of those discussed, for analyzing the circuits with respect to hazards and unknown signals where counting the number of hazard pulses is not important.

1.3.10 The Twenty-Seven-valued Algebra

Breuer and Harrison introduced a twenty-seven-valued algebra for eliminating the static and dynamic hazards in test generation. They built their algebra from T_3 * T_3 by attaching a third coordinate containing one of {hf, hsu, hp} which means hazard free, hazard state unknown, hazard present respectively. The operation tables for the algebra are rather large.

The algebra seems flawed as (0,1,hp) * (1,1,hsu) should give (0,1,hp), but instead is defined as (0,1,hsu). This leads to the algebra being non-associative, as shown by the following counter example.

$$
\begin{aligned}
& ((0,1,hp) * (1,1,hsu)) * (0,1,hsu) && (8)\\
= \; & ((0,1,hsu) * (0,1,hsu)) && (9)\\
= \; & (0,1,hsu) && (10)\\
\neq \; & (0,1,hp) && (11)\\
= \; & ((0,1,hp) * (0,1,hsu)) && (12)\\
= \; & (0,1,hp) * ((1,1,hsu) * (0,1,hsu)) && (13)
\end{aligned}
$$

In addition, the algebra does not follow the absorption or distributive laws, both of which have similar counter-examples. All other laws for a ternary algebra hold in the twenty-seven-valued algebra. Hence the algebra is not a semigroup and so not any of the other structures as described above.

1.4 Introduction to PVM

1.4.1 Parallelization

Three approaches to parallelizing a task are evident:

- **Fault Parallelism**
- **Search Parallelism**
- **Element-level Parallelism**

In case of **fault parallelism**, the fault list is partitioned for distribution among processors. To avoid interprocessor communication, the processors may independently generate tests. However, duplication of work occurs if the test generated in one processor can detect a fault that is assigned to another processor. A close-to-ideal speedup occurs only if the fault list is carefully partitioned and interprocessor communication delay can be neglected.

In **search parallelism** all processors in the multiprocessing environment work in unison to find a test for the same given fault. Every processor has a copy of the circuit. The amount of memory on a single processor in a multiprocessor system limits the size of circuits that can be tested by this method. The search space for this problem can be divided into disjoint subspaces and each processor searches one subspace for a test vector. If a processor finds a test vector it sends messages to all other processors to abort further search. This is an obvious way to parallelize any branch-and-bound method but

some ingenuity is required to isolate subspaces that have a higher likelihood of containing a solution. By searching subspaces we have a higher chance of finding a test vector quickly.

In **element-level parallelism** the gates of the circuit are distributed on multiple processors. A recent study concludes that fault parallelism should be preferred over element-level parallelism mainly because of the excessive inter-processor communication required in the element-level parallelism.

Parallel processing, the method of having many small tasks solve one large problem has emerged as a key enabling technology in modern computing. The acceptance has been facilitated by two major developments: massively parallel processors (MPPs) and the widespread use of distributed computing.

MPPs combine a few hundred to a few thousand CPUs in a single large cabinet connected to hundreds of gigabytes of memory.

The second-major development affecting scientific problem solving is distributed computing. It is a process whereby a set of computers connected by a network are used collectively to solve a single large problem.

Common between distributed computing and MPP is the notion of message passing. All parallel processing data must be exchanged between cooperating tasks. The Parallel Virtual Machine (PVM) system uses the message-passing model to allow programmers to exploit distributed computing across a wide variety of computer types including MPPs. PVM is a software system that permits a heterogeneous collection of Unix computers networked together to be viewed by a user's program as a single parallel computer. PVM is designed to link computing resources and provides users with a parallel platform for running their computer applications, irrespective of the number of different computers they use and where the computers are located.

With heterogeneous network computing environment we mean heterogeneity in terms of

- Architecture
- Data format
- Computational speed
- Machine load
- Network load

PVM transparently handles all message routing, data conversion and task scheduling across a network of incompatible computer architectures. The user writes his application as a collection of cooperating tasks. Tasks access PVM resources through a library of standard interface routines. These routines allow initiation and termination of tasks across the network as well as communication and synchronization between tasks. Communication constructs includes those for sending and receiving data, structures as well as high-level primitives such as broadcast, barrier-synchronization and global sum. At any point in the execution of a concurrent application, any task in existence may start or stop other tasks or add or delete computers from the virtual machine.

Other Packages

Various other systems with similar capabilities are also in existence. Among the most well-known efforts are:

- P4
- MPI
- Linda system
- Express

1.4.2 The PVM System

The principles on which the PVM is based include the following:

- **User-configured Host Pool:** the application's computational tasks execute on a set of machines that are selected by the user for a given run of the PVM program. Both single CPU machines and hardware multiprocessors including shared memory and distributed memory computers may be part of the host pool. The host pool may be altered by adding and deleting machines during operation (an important feature for fault tolerance). We used a heterogeneous host pool comprising Sun Solaris, Windows and Linux platforms.
- **Translucent Access to Hardware:** application programs either may view the hardware environment as an attribute-less collection of virtual processing elements (which is the strategy employed by us); or may choose to exploit the capabilities of a specific machine in the host pool by positioning certain

computational tasks on the most appropriate computers. Our strategy has been to let the PVM decide which processor to spawn the task to.

- **Process-based Computation:** the unit of parallelism in PVM is a task (often, but not always, a Unix process), an independent sequential thread of control that alternates between communication and computation. No process-to-processor mapping is implied or enforced by PVM; in particular, multiple tasks may execute on a single processor.
- **Explicit Message-passing Model:** collections of computational tasks, each performing a part of the application's workload, using data, functional or hybrid decomposition, cooperate by explicitly sending and receiving messages to one another. Message size is limited only by the amount of available memory. We have used data parallelism, rather that functional parallelism, and the decomposition is of the data.
- **Heterogeneity Support:** the PVM system supports heterogeneity in terms of machines, networks, and applications. With regard to message passing, PVM permits messages containing more than one data type to be exchanges between machines having different data representations.
- **Multiprocessor Support:** PVM uses the native message-passing facilities on multiprocessors to take advantage of underlying hardware.

1.4.3 Composition of PVM System

The first part is a daemon, called **pvmd3** and sometimes-abbreviated **pvmd that** reside on all the computers making up the virtual machine. Pvmd3 is designed so any user with a valid login can install it on a machine. When a user wishes to run a PVM application, he first creates a virtual machine by starting up PVM. The PVM application can then be started from a Unix prompt on any of the hosts. Multiple users can configure overlapping virtual machines and each user can execute many PVM applications simultaneously.

The second part of the system is a library of **PVM interface routines.** It contains a functionally complete repertoire of primitives that are needed for cooperation between tasks of an application. This library contains user-callable routines for message passing, spawning processes, coordinating tasks and modifying the virtual machine.

The PVM computing model is based on the notion that an application consists of several tasks. Each task is responsible for a part of the application's computational workload. Sometimes an application is parallelized along its functions; that is, each task performs a different function, for example, input, problem setup, solution, output, and display. This process is often called functional parallelism. A more common method of parallelizing an application is called data parallelism (the approach used by us). In this method all the tasks are the same, but each one only knows and solves a small part of the data. This is also referred to as the **SPMD (single-program multiple-data)** model of computing. PVM supports either or a mixture of these methods. Depending on their functions, tasks may execute in parallel and may need to synchronize or exchange data, although this is not always the case. An exemplary diagram of the PVM computing model is shown in following Fig 1.23.

The PVM system currently **supports C, C++, and Fortran languages**. This set of language interfaces have been included based on the observation that the predominant majority of target applications are written in C and Fortran, with an emerging trend in experimenting with object-based languages and methodologies.

The unit of parallelism is the task. The computation is divided into many tasks. All PVM tasks are identified by an **integer *task identifier* (TID)**. Messages are sent to and received from tids. Since tids must be unique across the entire virtual machine, they are supplied by the local pvmd and are not user chosen. Although PVM encodes information into each TID the user is expected to treat the tids as opaque integer identifiers. PVM contains several routines that return TID values so that the user application can identify other tasks in the system.

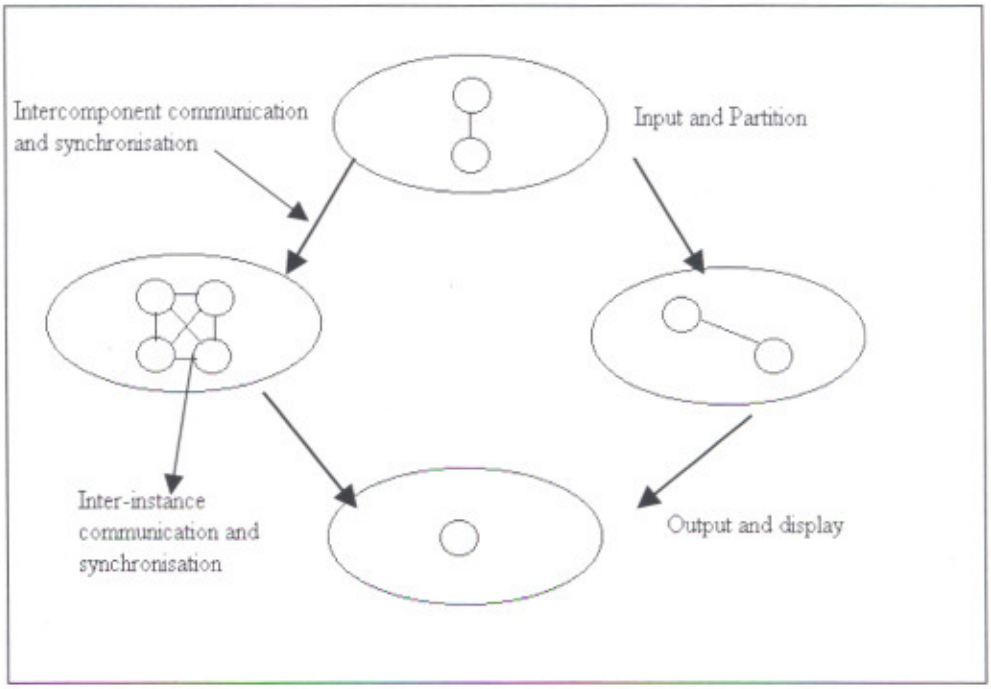


Fig 1.23 **PVM Computational Model**

There are applications where it is natural to think of a *group of tasks*. And there are cases where a user would like to identify his tasks by the numbers **0 - (p - 1)**, where **p** is the number of tasks. PVM includes the concept of user named groups. When a task joins a group, it is assigned a unique ``instance" number in that group. Instance numbers start at 0 and count up. In keeping with the PVM philosophy, the group functions are designed to be very general and transparent to the user. For example, any PVM task can join or leave any group at any time without having to inform any other task in the affected groups. Also, groups can overlap, and tasks can broadcast messages to groups of which they are not a member.

The general paradigm for application programming with PVM is as follows. A user writes one or more sequential programs in C, C++, or Fortran 77 that contain embedded calls to the PVM library. Each program corresponds to a task making up the application. These programs are compiled for each architecture in the host pool, and the resulting object files are placed at a location accessible from machines in the host pool. To execute an application, a user typically starts one copy of one task (usually the ``master" or

``initiating" task) by hand from a machine within the host pool. This process subsequently starts other PVM tasks, eventually resulting in a collection of active tasks that then compute locally and exchange messages with each other to solve the problem. Note that while the above is a typical scenario, as many tasks as appropriate may be started manually. As mentioned earlier, tasks interact through explicit message passing, identifying each other with a system-assigned, opaque TID.

1.5 Problem Specification

Effective testing is an essential part of any development process and VLSI is no exception. In VLSI circuits probabilities of occurring the faults is significantly high due to the smaller size and higher complexity of the circuits. Therefore chip testing is very difficult and essential. VLSI Testing is the broad area, there are various testing techniques used to test the chip, from one of them is Automatic Test Pattern Generation (ATPG).

The ATPG technology, in addition to generating high-quality tests for various fault models, offers efficient techniques for analyzing the logic networks. The technology has been successfully applied in several other areas of electronic design automation such as logic optimization, design verification and timing analysis. The problems of interest in these applications are transformed and modeled as a search problem for a test of a stuck-at fault or a delay fault. We know that most of the faults are detected by stuck-at fault model (approximately 80%), but remaining faults are delay faults, memory faults and bridge faults as seen in Figure 1.24.

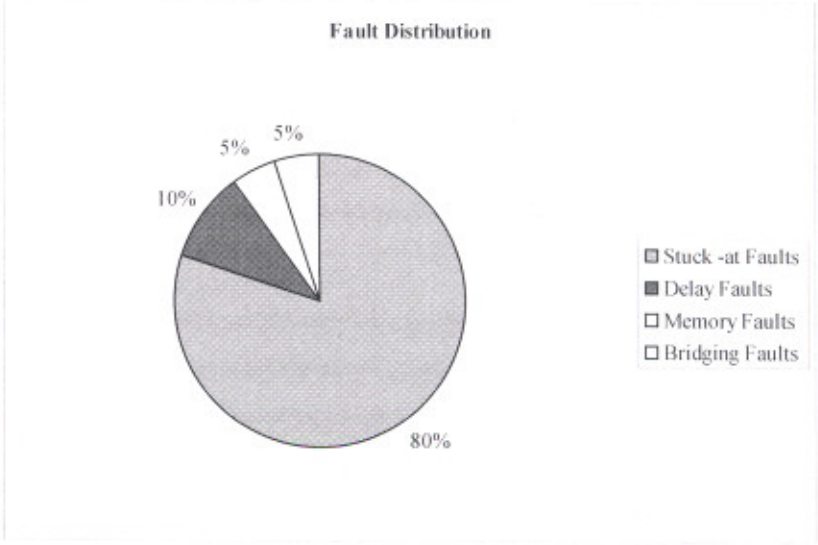


Fig 1.24 **Fault Distribution**

In this thesis, our main concentration is on delay faults. These delays are not due to logical problem of input-output but due to presence of hazards (static as well as dynamic hazards) and delay defects like

- GOS defects
- Resistive shorting defects between nodes and to the supply rails
- Parasitic transistor leakages, defective PN junctions and incorrect or shifted threshold voltages
- Certain types of opens
- Process variations can also cause devices to switch at a speed lower than the specification.

Research has been done on delay fault testing and is described in Chapter 2: Literature Survey. We found that path delay faults are taking more attention. Various techniques and methods being used for path delay faults have been described. In this thesis, we have used 10-valued logic for robust tests and 3-valued logic for non-robust test. But from literature research we found that path delay faults are taking more CPU time and memory consumption for detecting test vectors. So a parallelized approach is needed. Some parallel approaches have been reported earlier. In this thesis, we propse a new parallelized (master-slave) approach for path delay faults. The proposed algorithm has

been implemented in parallel and distributed environment using Parallel Virtual Machine (PVM).

1.6 Organisation of Thesis

The Chapter 1 INTRODUCTION, describes VLSI Testing, delay fault models like path, gate, line, segment, line, transition, functional, various hazard algebras like three-valued, four-valued, nine-valued, ten-valued, thirteen-valued, parallel and distributed environemnt i.e. parallel virual machine (PVM).

The Chapter 2 LITERATURE SURVEY describes the research or work done on various delay fault models, and which delay fault model is taking more attention.

The Chapter 3 PATH DELAY FAULTS describes introduction to path delay faults, robust test generation using five-valued logic, seven-valued logic, nine-valued logic and ten-valued logic and non-robust test generation using three-valued logic.

The Chaper 4 PARALLELIZATION OF TEST GENERATION FOR PATH DELAY FAULTS, describes other methods for paralleization of path delay faults and a new approach i.e. Master-Slave approach in parallel and distributed environment (PVM). It also includes sequential algorithm, parallel algorithm, and comparision between two algorithms, implementation and experimental results.

The Chapter 5 CONCLUSIONS conclude the thesis followed by highlighting the future scope of path delay faults.

2.1 Introduction

Digital system designers have traditionally maximized the frequency of system clocks in order to obtain the highest performance from the hardware. The maximum allowable clock rate is determined by the propagation delays of the combinational logic block between latches [37]. Consider the circuit under test shown in Figure 1. During the normal operation of the circuit, the input clock C_1 is the same as the output clock C_2 and the period (T_c) of C_1 and C_2 corresponds to the system clock. This period should be greater than the maximum propagation delay DP_{max} for the circuit. However, during testing for delay faults, we use two separate test clocks, C_1 and C_2, running at a frequency that is slower than the normal system clock.

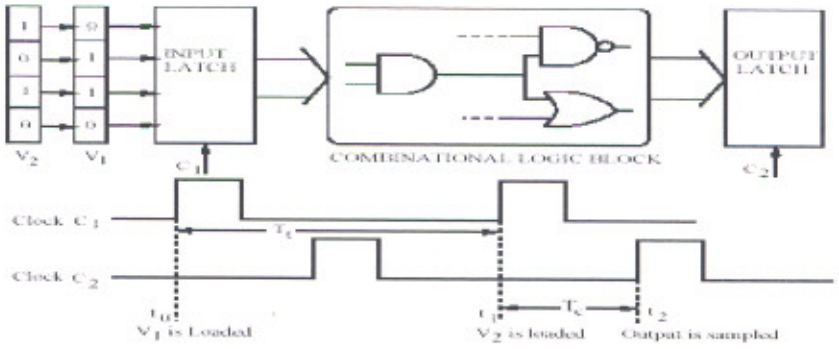


Fig 2.1 **Hardware Model & Clock Timings**

Thus, the period of test clocks, T_t, is longer than T_c. The two test clocks are skewed by the amount T_c. The activation of the output clock C_2 must follow the activation of the input clock C_1 by at least DP_{max} time units (i.e., $T_c = t_2 - t_1 \geq DP_{max}$). If the output clock is activated sooner, then unstabilized and possibly incorrect logic values may be latched in output latches.

Since delay faults do not alter the logic function realized by a circuit and since the tests for stuck-at faults are normally applied at a slow clock rate (due to the ATE limitation or other reasons), they are inadequate for detecting delay faults. Special two-pattern test vectors are required for detecting delay faults. The hardware model used

in delay fault testing is shown in Figure 2.1. Here the vector pair < V_1, V_2 > constitutes a delay test and signals C_1 and C_2 are used to clock the input and output latches, respectively. At time t_0, an initializing input vector V_1 is applied, and the circuit is allowed to stabilize under input V_1. At time t_1, the propagation vector V_2 is applied, and the outputs are sampled at time t_2, where (t_2 - t_1) is the intended time interval between the input and output clocks, called the rated clock interval T_c [37].

Exhaustive testing is quite impractical for delay faults since the total number of pattern-pairs required will be $(2^n)(2^n - 1)$, which is of the order 2^{2n}, for a circuit having n inputs. One must derive suitable and reasonable delay fault models and devise algorithms that can generate tests for the modeled faults. Various fault models used in delay fault testing are discussed in following sections.

2.2 Delay Fault Models

There are three classical fault models that have been developed in recent past to represent the delay defects (i.e., transition fault, gate delay fault and path delay fault). In recent years two more fault models (i.e., line delay fault and segment delay fault) have been developed, which are basically derived from classical ones. Each of these fault models has their own limitations and advantages in various aspects that have been discussed in detail in this chapter.

2.2.1 Path Delay Fault Model

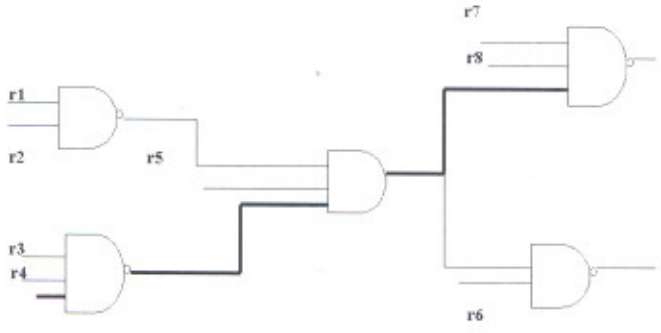


Fig 2.2 **Path Delay Faults**

Smith first proposed the path delay fault model. This model has received greater attention than the gate delay and transition fault models and has been quite extensively studied in [6], [10], [11], [12], [14], [22], [23], [24], [25], [29], [31], [33], [34], [35], [36], [37], [38], [39], [41], [44], [53], [57], [63], [70], [75].

It includes considerable amount of research is being done on path delay faults. In path delay fault model, any path with a total delay exceeding the system clock interval is said to have a path delay fault. These models distributed defects that affect an entire path. See in Figure 2.2, for each physical path P, connecting a primary input to a primary output of the circuit, there are corresponding two delay paths. The rising path (falling path) is the path traversed by a transition whenever it passes through an inverting gate. We present the following definitions that are frequently used in path delay fault testing [74].

Definition 1: Let G be a gate on path P in a logic circuit, and let r be an input to gate G; r is called an *off-path sensitizing input* if r is not on path P.

Definition 2: A two-pattern test $<V_1, V_2>$ is called a *robust test* for a delay fault on path P, if the test detects that fault independently of all other delays in the circuit.

Definition 3: A two-pattern test $<V_1, V_2>$ is called a *nonrobust test* for delay fault on path P, if it detects the fault under the assumption that no other path in the circuit involving the off-path inputs of gates on P has a delay fault.

As we mentioned above that most of the research is going on *path delay faults*. There are various models, algorithms, methods, and procedures proposed path delay faults. Lets take a survey on these faults.

The very first research has given by *Smith* in *1985*, he invented firstly that why we didn't get 100% performance in automatic test pattern generation (ATPG), as we know that 95% of faults are covered by stuck-at fault model but no much concentration had been given on remaining 5% faults, these faults are due to various delay defects known as delay faults. Smith gave introduction about delay faults, different categories of delay faults i.e. robust, non-robust tests, delay defects, hazards (including static as well as dynamic) due to which these faults occur. Smith emphasis on the paths on which delay occurs. He did the *analysis to find out the paths where critical faults* are more.

After that in *1990*, *Devdas and Keutzer* [1] developed necessary and sufficient conditions for robust path-delay fault testability to develop synthesis procedures, which

produce two-level and multilevel circuits with high degrees of robust path delay fault testability. For circuits, which can be flattened to two levels, they gave a *covering procedure*, which optimizes for robust path delay fault testability. These two-level circuits can then be algebraically factored to produce robustly path-delay-fault testable multilevel circuits. For regular structures, which cannot be flattened to two levels, they gave a *composition procedure*, which allows or the construction of robustly path-delay fault testable regular structures. They also showed how these two techniques can be combined to produce cascaded combinational logic blocks that are robustly path-delay fault testable. They demonstrated these techniques on a variety of examples. It is possible to produce entire chips that are fully path delay testable using these techniques.
In *1992*, they added more modifications; they applied the same conditions to the less restrictive gate-delay fault models to find out the performance comparison between two models on the basis of testability [4].

In *1992*, *Girard* gave a *novel approach to delay-fault diagnosis* [2], he presented a new method based on critical path tracing from a six-valued simulation. This method showed the perfect experimental results, it does not require any timing evaluations and can be very accurate.

Delay testing at the operational system clock rate can detect system-timing failures caused by delay faults. However delay fault coverage in terms of percentage of the number of tested faults may not be an effective measure of delay testing because, unlike a stuck-at-fault, the impact of a delay fault on a proper system operation is dependent on its delay defect size rather than on its existence. The effectiveness of a delay test is dependent on the propagation delay of the path to be tested, the delay defect size, and the system clock interval. *Park, Mercer, Williams* [5] proposed a *quantitative delay fault coverage model* which provides a figure of merit for delay testing. System sensitivity of a path to a delay fault along that path and the effectiveness of a delay test are described in terms of the propagation delay of the path under test and the delay defect size. A new statistical delay fault coverage (SDFC) model is established. A new defect level model is also proposed as a function of the yield of a manufacturing process and the new statistical delay fault coverage. They also proposed a new delay testing strategy driven by the defect level for delay faults.

In same year *Fink, Fuchs, Schulz* [6] proposed a *simulation approach* for path delay faults. The distinct feature of the proposed fault simulation method consist in the application of parallel processing of patterns at all stages of calculation procedure, its versatility to account for both robust and nonrobust detection of path delay faults and its capability of efficiently maintaining large number of path faults to be simulated. They used the six-valued simulation. Approach.

In 1993, as the complexity of integrated circuits was growing rapidly, that lead to spent more and more time and money spent on the test of these circuits. *Fey, Shi, Drechsler* [9] gave a new approach, besides minimizing the logic needed for a given function the testability of the resulting circuit become a major issue during synthesis. One way to synthesize a circuit for a given function is to directly convert the Binary Decision Diagram (BDD) of that function into a circuit. It is known that *optimizations of the BDD transfer to the derived circuit*. Therefore they evaluated different optimization techniques for BDD' s based on variable reordering with respect to the path delay fault testability of the resulting circuit. They showed an optimization strategy that allows to compromise during synthesis between logic size and testability.

In *1994, KraSniewski, Wrofiski* [10] derived the relationships between *testability of path-delay faults and gate delay faults*. They used the theoretical results to develop a fast procedure for comprehensive assessment of a given test sequence under the different assumptions on what "detection of a delay fault by an input-pair" means and the experimental results demonstrated that a fair comparison of different test strategies requires several coverage metrics, corresponding to the different delay fault models and testability requirements.

In same year, *Henftling, Wittmann, Antreich* [11] gave an efficient *approach to path delay fault simulation*. They accelerated fault simulation by more than one order of magnitude with a new speed-up technique called *path hashing*. An intelligent path identification method is proposed that allows dealing with circuits containing two orders of magnitude more paths than state – of the – art tools. Using these techniques large circuits can be handled with a reasonable amount of time and memory.

In 1995, Majhi, Jacob, Patnaik and Agarwal [12] developed new *test pattern generation system for path delay faults* in combinational logic circuits considers robust

and nonrobust tests, simultaneously. In this system, once a robust test is obtained for a path with a given transition, another test for the same path with the opposite transition is immediately derived with a small extra effort. To facilitate, the simultaneous consideration of robust and nonrobust tests, they derived a new nine-value logic system. An efficient multiple backtrace procedure satisfies test generation objectives. They also used a path selection method, which covers all lines in the logic circuit by the longest and the shortest possible paths through them. A fault simulator in the system gave information on robust and nonrobust detection of faults either from a given target set or all path faults. Experimental results on *ISCAS'85* and *ISCAS'89* benchmark circuit's substantiated the efficiency of their algorithm in comparison to other published results.

In this year, *Henftling and Wittmann* [14] adds more implementation in research on path delay faults. They proposed a method to apply *bit parallel processing at all stages of robust and nonrobust test generation of path delay faults*. Two different modes of bit-parallel processing are combined: fault parallel test pattern generation (FPTPG) and alternative parallel test pattern generation (APTPG). They discussed the problems that appear while exploiting bit-parallelity and they also described how to overcome them. Experimental results showed that a reduction of aborted faults and an acceleration up to a factor of nine.

To improve the *quality of delay fault testing* [17], they examined various delay models used in VLSI Testing. Their study included electrical-level simulation experiments with *HSPICE*. They showed the phenomenon which significantly affect the actual delays, but which are not taken into account by the existing models used in testing. Their analysis questions the test quality offered by test generation procedures used so far.

For the *high performance systems*, it is the requirement to do rigorous testing for path delay faults, in this year *Lam* proposed a *synthesis algorithm* [18], which produces a 100% path delay fault testable function with a minimal set of test pins. Various test generation applications for delay fault models in described by *Cheng, Krstic* [19].

In *1996*, an *algebraic method* for delay fault testing is proposed by *Sophie, Mireille and Rene* [21]. The aim of this method is to allow an *accurate analysis of delay faults*. This method is based on the fact that some input values remain constant when two

successive input vectors are applied. For a transition between two input states, the output function is reduced to a function of few variables. An analysis of reduced function allows obtaining the delay faults, which are detected by the corresponding transition. The analysis allows knowing if a fault is robustly testable or non-robustly testable and validatable, or weekly verifiable: in very case the corresponding tests are obtained. An application of the results to random testing of faults allows observing that some nonrobustly testable faults are easier to detect than some robustly testable faults.

In the same year, the problem involved in handling large numbers of path delay faults were alleviated in previous work by developing a *fault simulation and test generation procedure* that do not require paths to be explicitly considered by *Pomeranz, Reddy* [22]. Thus the methods developed allow the set of all path delay faults to be targeted during test generation and fault simulation. With the problems related to the number of paths removed, a new limiting factor in test generation for path delay faults is revealed, namely the number of test required to detect all path delay faults. They investigated the problems related to the number of tests. A procedure for computing a lower bound on the number of tests is described, and the methods for synthesizing circuits with reduced lower bounds on the number of tests developed.

Conditions are derived for robust testing of a path delay fault via a sequence of test vectors applied at-speed. For that a *simulator* proposed by *Hsu, Gupta* [23]. This simulator uses these above conditions along with the knowledge of paths that are robustly tested by the previous vectors, to determine the fault coverage obtained by such testing. The results show that existing fault simulators could overestimate robust path-delay fault coverage by 5-15%.

In many designs a large portion of path delay faults is not robustly testable. *Krstic and Cheng* investigated *test strategies* for robustly untestable faults [24]. They showed that the quality of non-robust tests could be obtained by including timing information into the process of test generation. A good non-robust test can tolerate larger timing variations on the off-inputs. They also showed that not all non-robustly untestable path delay faults might be ignored in high quality delay testing. Functional sensitizable paths are non-robustly untestable but under some fault conditions, may degrade the performance of the circuit. However, up till now, there was no strategy for generating

tests for such faults. They presented the algorithms for generating high quality non-robust and functional sensitizable tests. They also devised an algorithm for generating tests for validatable non-robust faults, which have a high quality in detecting defects but are hard to be generated automatically. Their experimental results show that the quality of delay testing increases if validatable and high quality non-robust tests, as well as tests for functional sensitizable path delay faults are included.

A new formulation to generate robust tests for path delay faults in combinational circuits based on *Boolean Satisfiability* is given by *Chen, Gupta* in *1996* [25]. Conditions to detect a target path delay faults are represented by a Boolean formula. Unlike the other techniques described by other researchers for stuck-at faults, which extracts the formula for each path delay fault, the proposed formulation needs to extract the formula only once for each circuit cone. Experimental results show that tremendous amount of time saving on formula extraction compared with other satisfiability-based algorithms. This also leads to the low test generation time, especially for the circuits having many numbers of paths and few outputs. The proposed formulation also modified to develop other type of tests for path delay faults.

Luong described test generation for delay faults caused by global process disturbances [26]. They represented structural and spatial correlation between path delays is used to reduce the number of paths that must be tested.

A *novel algorithm* is proposed to rapidly identify untestable delay faults using pre-computed static logic implications by *Heragu, Patel* [28]. Their fault-independent analysis identifies large sets of untestable faults, if any, without enumerating them. The cardinalities of these sets are obtained by using a counting algorithm that has quadratic complexity in the number of lines. Since their method is based on an incomplete set of logic implications, it gives only a lower bound on the number of untestable faults. A post-processing step can list the untestable faults, if desired. Targeting untestable delay faults for test generation by an automatic test pattern generation (ATPG) tool can be avoided. The method works for the segment delay fault model and its special case, the path delay fault model, and identifies robustly untestable, non-robustly untestable, and functionally unsensitizable delay faults. Results on benchmark circuits show that many delay faults are identified as untestable in a very

short time. For the benchmark circuit *c6288*, their algorithm identified $1.978*10^{20}$ functionally unsensitizable path faults in 3 CPU seconds.

A *coverage metric* and a *two-pass test generation method* for path delay faults in combinational logic circuits proposed by *Majhi, Jacob, Patnaik* [29]. The coverage is measured for each line with a rising and a falling transition. However, the test criterion is different from that of the slow-to-rise and slow-to-fall transition faults. The test, called "line delay test", is a path delay test for the longest sensitizable path producing a given transition on the target line. The maximum number of tests (and faults) is limited to twice the number of lines. However, the line delay test criterion resembles path delay test and not the gate or transition delay test. Using a two-pass test generation procedure, they began with a minimal set of longest paths covering all lines and generate tests for them. Fault simulation is used to determine the coverage metric. For uncovered lines, in the second pass, several paths of decreasing length are targeted. They presented a theorem stating that a redundant stuck-at fault makes all path delay faults involving the faulty line untestable for either a rising or falling transition depending on the type of the stuck-at fault. The use of this theorem considerably reduces the effort of delay test generation. They gave results on benchmark circuits.

As we found that detailed circuit simulations have demonstrated that a classical two-pattern robust test for a path delay fault may not excite the worst-case delay of the target path. *Chen, Gupta* [31] in *1997* developed a new definition of robust test that maintains the desirable properties of classical robust tests while incorporating *two additional considerations*, namely *side-fan-in transitions and pre-initialization*, which are shown to have a significant impact on the delay of target path. The associated test generation problem was formulated as a constraint optimization problem, and an ATPG system developed to generate three-pattern robust tests that excite the worst-case delay of the target path. The ATPG works on a gate level model that is augmented to capture the necessary switch level details. Experimental results shows that quality of robust delay tests varies dramatically and that the proposed high quality robust delay tests are needed for improving test quality.

To make path delay fault test generation memory efficient, a *memory efficient test pattern generator* for path delay faults, DTPG, is presented by *Long, Li,*

Yang and Min [34], which uses the efficient path identifier to represent a path. A compact bit table, path information table, is proposed to store test information efficiently. Furthermore, DTPG is capable of identifying functional sensitizable paths, which account for large percent of paths in many circuits. The experimental result shows that DTPG is memory efficient. It generates tests for c*3540* with 57 million paths and preserves the testability information for all paths. Experimental results show the influence of stepwise mandatory sensitization, multiple backtrace, and backtracking limits on the CPU line consumed by delay test generation process.

A recent method proposed by *Kagaris, Tragoudas, Karayiannis* in same year [35] that a *lower bound on the number of path delay faults* excited by a given test set can be computed using a set independent lines that form a cut. For each line in the cut a sub circuit consisting of all paths that contain the line is defined, and a lower bound to the number of excited path delay faults can be obtained by working on the respective sub circuits. A polynomial time algorithm is presented here for computing the maximum cardinality set of independent circuit lines. Experimental results show that the more the sub circuits the better the lower bound on the number of excited path delay faults is. More sub circuits may be generated only in a heuristic manner. It was proposed to consider two or more line-disjoint cuts C_i. They proposed a technique where only one C_a must be a cut. This scheme is based on novel algorithms, and results in more sub circuits than the previous one.

A *method of identifying primitive path-delay faults* in combinational circuits is proposed by *Tekumalla and Menon* in *1997* [36]. They derived robust tests for all robustly testable primitive faults. It uses the concept of sensitizing cubes to reduce the search space. This approach helps identify faults that cannot be part of any primitive fault, and avoids attempting test generation for them. Sensitization conditions determined for primitive fault identification are also used in test generation, reducing test generation effort. Experimental results on some of the *ISCAS'85* and *MCNC'91* benchmark circuits indicate that they contain a fair number of primitive multiple path delay faults, which must be tested.

A *review* on different delay fault models has been given by *Majhi, Agrawal* in *1998* [37]. They also discussed classifications and proposed fault coverage metrics.

At-speed robust path delay testing is more desirable than the commonly employed *slow-fast* delay testing due to lower test application time, lower area overhead, and increased possibility of detecting un-modeled faults that cause errors only during at-speed circuit operation. However, it has been observed in practice that at-speed application of tests generated by existing test generators can lead to their invalidation. *New techniques* presented by *Hsu, Gupta* [38], and the aim is to modify tests generated by existing test generators to avoid invalidation during at-speed testing. They also presented a new procedure to generate tests suitable for at-speed delay testing of combinational circuits. Experimental results show that (a) at-speed application of test sets generated by existing generators leads to significant test invalidation, where the degree of invalidation is approximately proportional to the degree of compactness of the test set, and (b) the at-speed robust path delay tests generated by the proposed test generator are significantly shorter than those obtained by modifying the tests generated by existing generators.

Testing path delay faults (PDF's) in VLSI circuits is becoming an important issue as we enter the deep sub-micron age. However, it is difficult in general, since the number of faults normally is very large and most faults are hard to sensitize. To make delay fault testing and test synthesis easier, a model is proposed for path delay faults by *Cheng, Chih* [39] known as *Probabilistic Model for path delay faults*. They investigated probability density functions for wire and path delay size to model the fault effect in the circuit under test. In their approach delay fault size is assumed to be randomly distributed. An analytical model is proposed to evaluate the PDF coverage. They showed that the fault size of undetected paths could be greatly reduced if these paths co joined with other detected paths. Therefore, by their approach, path selection and synthesis of PDF testable circuits can be done more accurately and for a test set, fault coverage can be predicted by calculating the mean delay of the paths.

In *1998, Bose and Agrawal* [41] presented an algorithm to *derive logic systems for various classes of path delay fault* test problems. In these logic systems, the value of a signal represents the relevant conditions that occur during a set of consecutively applied vectors. Starting from a set of basic values for valid signals at primary inputs, a state transition graph is constructed to enumerate all possible signal states relevant to path

activation that are reachable by Boolean operations. These states include all incompletely specified states, composed as combinations of basic values. A distinguishability analysis then finds all state-pairs that need to be distinguished during test generation. The final step minimizes the number of states. For forward and backward implications of test generation in combinational or sequential circuits, the procedure provides optimal logic systems. They defined optimality as the smallest set of logic states that provides the least possible ambiguity in implications. Thus, an optimal set of logic states will minimize the number of backtracks in test generation. A 10-valued logic described in the literature is found to be optimal for generating tests for single path delay faults. They also addressed other problems in include compact test generation through activation of many single path delay faults, test generation for rated-clock test application, and test generation for multiple path delay faults. The limitations and capabilities of various logic systems are illustrated by examples.

A flexible path selection procedure for path delay testing proposed by *Pomeranz and Reddy* [44] in *1999*. This procedure selects target faults for path delay fault test generation. Since large number of path delay faults may be untestable, the proposed procedure does not select a fixed set of paths. Instead, it provides compactly represented subsets of paths, referred to as super-paths, and allows the test generation procedure to select one path out of each subset based on testability considerations.

To do the fault simulation of delay faults, *Ravikumar and Mittal* [47] proposed a fault simulation technique i.e. *Hierarchal Fault Simulation*. As we know that increasingly, VLSI systems are being designed using macro blocks and predesigned cores. Since the clock rate at which these circuits operate is steadily increasing over the years, it is important to perform delay testing on modern VLSI chips and systems. Algorithms for delay test generation and delay fault simulation are known to be compute-intensive. Many of these algorithms require gate-level descriptions of circuits, which are difficult to generate, and may be even impossible to provide when the designer has made use of predesigned cores. Hierarchical testing appears to be an attractive alternate in such cases. Tests generated for logic blocks may be reused to generate tests for larger systems comprising of the logic blocks, hence reducing the total effort in test generation. They showed the computational effort spent in fault simulation could also be reduced using a

hierarchical approach. The simulator HIDEFS described which exploits the modular nature of the circuit to save on the memory requirement as well as execution time requirement of fault simulation.

To improve the delay faults diagnosis, new adaptive techniques proposed by *Dastidar and Touba* [51]. These techniques reduce the search space for direct probing which can save a lot of time during failure analysis. For a given set of two-pattern tests that resulted in faulty output responses, a procedure for deriving additional two-pattern tests that will improve the diagnostic resolution of delay faults is described. They presented two new techniques based on adjacency testing and delay-size bounding. These techniques can be used to greatly reduce the number of suspect lines and thereby provide a more precise diagnosis that is valid for either single or multiple delay faults. Experimental results show that the number of suspects can be reduced dramatically for both single and multiple delay faults.

In *2000* to deal with the problem of delay fault testing, *Sosnowski, Wabia, Bech* [53] presented results obtained with newly developed *test pattern generator*. This generator based on the use of binary decision diagrams (ROBDDs) and reveals many advantages as compared with other ATPGs published in the literature. Their important contribution was the analysis of various testability features of digital circuits in respect to path delay faults. It was applied to *ISCAS89* benchmark and other circuits. The proposed testability measures are helpful in co designing deterministic and pseudorandom delay tests as well as in improving circuit testability.

Many techniques has been implemented for path delay faults, from one of them is *Boolean satisfiability*, The Boolean satisfiability problem (SAT) has various applications in electronic design automation (EDA) fields such as testing, timing analysis and logic verification. SAT has been typically applied to EDA as follows: 1) formulation of the given problem as a SAT instance 2) solution of the SAT instance. *Joonyoung, Jeese, Joao and Karem* [54] presented a method to simultaneously solve several closely related SAT instances using incremental satisfiability (ISAT). In ISAT, the decision sequence made for a "prefix" function is used to solve another set of functions, which have a number of new constraints (extensions) added to the prefix function. Their experiments show that they can achieve significant gains in total runtime when they use

this methodology as opposed to resetting the decision sequences and solving each instance from scratch. Application of ISAT to delay fault testing is presented by formulating incremental path sensitization as an ISAT problem. Non-robust tests for the combinational portion of *ISCAS'89* circuits are generated using this method.

In *2001*, the *fault-grading problem* formulated as combinatorial problem by *Padmanaban, Michael, and Tragoudas* [57], [64], that amounts to storing and manipulating sets on a special type of Binary Decision Diagrams (BDDs), called zero-suppressed BDDs (ZBDDs), that represent sets in a unique and compact manner. A simple modification of the basic scheme allows overcoming memory problems that may arise by complex set representation. Experimental results on the *ISCAS's* benchmarks show considerable improvement over all existing techniques for exact PDF grading. The main advantages of the proposed methodology are the simplicity of the approach, in terms of it being expressed by a polynomial number of increasingly efficient BDD-based operations, its organization, and its ability to handle very large test sets.

Noise effects such as power supply and cross talk noise can significantly impact the performance of deep sub-micrometer designs. Existing delay testing and timing analysis techniques cannot capture the effects of noise on the signal/cell delays. Therefore, these techniques cannot capture the worst case timing scenarios and the predicted circuit performance might not reflect the worst-case circuit delay. More accurate and efficient timing analysis and delay testing strategies needed to be developed to predict and guarantee the performance of deep sub micrometer designs. A new *pattern generation technique* for delay testing and dynamic timing analysis proposed by *Krstic, Jiang and Cheng* [60] in *2000* that can take into account the impact of the power supply noise on the signal propagation delays. In addition to sensitizing the selected paths, the new patterns also cause high power supply noise on the nodes in these paths. Thus, they also cause longer propagation delays for the nodes along the paths. Their experimental results on benchmark circuits show that the new patterns produce significantly longer delays on the selected paths compared to the patterns derived using existing pattern generation methods.

We can also generate test pattern for path delay faults *using stuck-at TPG*, an approach for generating test patterns for the path delay fault model based on a stuck-at

TPG tool is implemented by *Meyer, Anheier and Sticht* [62] which has been used for industrial purposes for years. Furthermore, the constraints to the definition of non-robust delay tests discussed. They showed that under some circumstances, the requirements accepted in literature are not sufficient to derive non-robust tests due to possible test invalidation, even in the absence of any additional faulty delay. They discussed possible constraints that have influence on the test quality and showed that a test generation for robust tests cannot be tackled if a stuck-at TPG tool is used without any delay fault simulation.

In *2002*, in current industrial practice, critical path selection is an indispensable step for AC delay test and timing validation. Traditionally, this step relies on the construction of a set of worse case paths based upon discrete timing models. The assumption of discrete timing models can be invalidated by delay effects in the deep sub-micron domain, where timing defects and process variation are statistical in nature. *Krstic, Liou, Cheng and Wang* [63] studied the problem of *optimizing critical path selection*, under both fixed delay and statistical delay assumptions. With a novel problem formulation and new theoretical results, they proved that the problem in both cases is computationally intractable. Then they discussed practical heuristics and their theoretical performance bounds, and demonstrate that among all heuristics under consideration, only one is theoretically feasible. They also provided consistent experimental results based upon defect-injected simulation using an efficient statistical timing analysis framework.

Delay fault tests are classified regarding to their probability of invalidation. Even if robust tests are preferred they exist only for a small number of faults. Non-robust test are therefore also a matter of interest. *Meyer, Sticht, Weigl and Anheier* [65] addressed the question *how non-robust tests can be strengthen* and how compact test pattern generation affects the test quality.

In *2003*, to improve more delay fault testing quality different *test strategies* implemented by *Krstic, Liou, Cheng and Wang* [68]. A structurally testable delay fault might become untestable in the functional mode of the circuit due to logic or timing constraints or both. Experimental data suggests that there could be a large difference in the number of structurally and functionally testable delay faults. However, this difference is usually calculated based only on logic constraints. It is unclear how this

difference would change if timing constraints were taken into consideration, especially when using statistical timing models. Their goal is to better understand how structural and functional test strategies might affect the delay test quality and consequently, change their perception of the delay test results.

The known methods of transition fault diagnosis usually suffer from the drawback of many candidates. The method presented by *Majhi, Gronthoud and Valer* [69] whose aim was to reduce the number of suspects. The transition fault patterns were generated by Philips in-house ATPG tool and applied on the tester. The fail information from tester was subjected to fault diagnosis resulting in a small list of faulty candidates. They then injected the delay faults into the golden net list of the test chip and confirmed through simulation whether or not their behavior matched with the tester results. Upon successful matching, they proceeded with the selection of few testable paths through the suspect faulty node and created corresponding path delay patterns using the path delay ATPG (a prototype at the University of Bremen, developed in cooperation with Philips Semiconductors GmbH, Hamburg). Finally, they verified those path delay patterns on the tester to increase the confidence level of the diagnosis method. The experimental results show the effectiveness of their novel approach for improving diagnostic resolution.

As we said earlier that test pattern for path delay faults can be find by using stuck-at fault test generation algorithms. A new *test generation method* proposed by *Ohtake, Ohtani and Fujiwara* [70] in *2003*. This method is for non-robust path delay faults using stuck-at fault test generation algorithms. The idea behind this method is, first transform an originals combinational circuit into a circuit called a partial leaf-dag using path-leaf transformation. Then they generated test patterns using a stuck-at fault test generation algorithm for stuck-at faults in the partial leaf-dag. Then transform the test patterns into two-pattern tests for path delay faults in the original circuit. They proved the correctness of the approach and experimental results on several benchmark circuits show the effectiveness of it.

Gupta, Hsiao [72] presented a *novel technique* for generating effective vectors for delay defects. The test set achieves high path delay fault coverage to capture small-distributed delay defects and high transition fault coverage to capture gross delay defects. Furthermore, non-robust paths for ATPG are filtered (selected) carefully so that

there is a minimum overlap with the already tested robust paths. A relationship between path delay fault model and transition fault model has been observed which helps us reduce the number of non-robust paths considered for test generation. To generate tests for robust and non-robust paths, a deterministic ATPG engine is developed. Clustering of paths has been done in order to improve the test set quality. Implications were used to identify the untestable paths. An incremental propagation based ATPG is used for transition faults. Results for *ISCAS'85* and full-scan *ISCAS'89* benchmark circuits show that the filtered non-robust path set can be reduced to 40% smaller than the conventional path set without losing delay defect coverage. Clustering reduces vector size in average by about 40%.

To calculate the *delay fault probabilities* in parallel manner, *Ganz, Tafertshofer and Wittmann* [73] presented an improved measure for the dynamic functionality of a logic circuit, called delay fault probability (DFP). The new measure reflects both the nominal delay of the paths and the fact that only few paths are critical for path delay fault testing. An efficient distributed algorithm for computing DFP is presented. The experimental results show that, in contrast to DFP, the conventional fault coverage is an inadequate criterion to assure the dynamic functionality of a circuit.

2.2.2 Gate Delay Fault Model

Carter introduced a quantitative model for delay faults, known as the gate delay fault. They assume that delays through logic gates are known with some precision. The characteristics (size and location) of likely delay faults are also known. The delays through a gate are represented by intervals in this model. A fault is an added delay of certain size (magnitude); say δ, in the propagation of a rising or falling transition from the gate input to output. The set of faults considered includes numerical delay information. An excessive delay of 3 nanoseconds at a point is not the same fault as an excessive delay of 5 nanoseconds at that point. See in Figure 2.3.

Most of the recent research in this area has concentrated on the determination of fault sizes detected by a given test. Given a particular fault of a fixed known size. *Carter* provides a method to determine whether a test T detects that fault. This is clearly a painstaking and inefficient method, and it would be more desirable to find a certain

minimum fault size at a fault site such that given a test T for a fault at the above fault site, T is guaranteed to detect any fault at that site with a magnitude greater than the determined minimum size.

There are many implementations, research is going on gate delay fault model, and Lets take a survey on this.

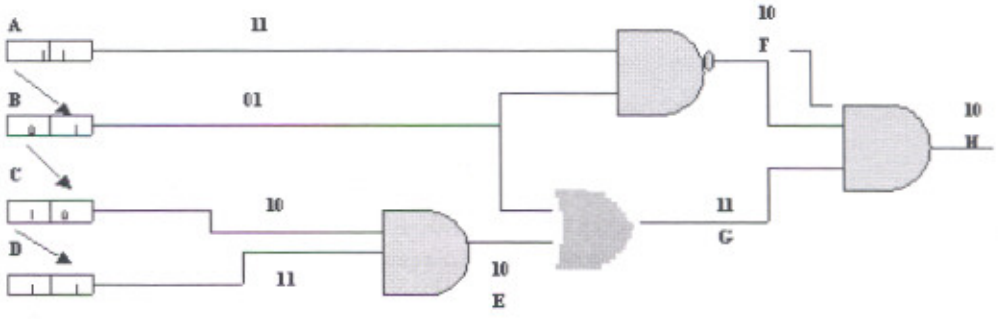


Fig 2.3 **Gate Delay Faults**

Carter has given the very first research in *1987*, which introduced a *quantitative model* for delay fault,

After that in *1993*, *Mahlstedt* [7] presented an *efficient approach* to generate tests for gate delay faults. Unlike other known algorithms, which try to generate a 'good' delay, test the presented algorithm is complete in the sense that if a delay test exists it will generate an optimal delay test. An optimal delay test for a gate delay fault is a test that sensitizes the longest functional path through the fault site. Especially the cone-oriented test generation – each output cone is processed separately - and the delay graph - a new method to keep track of all possible paths in a given situation - contribute to the efficiency of the algorithm. Although it is an NP-hard problem to generate optimal delay tests, experimental results show that it is tractable for a wide class of circuits. Close to optimal delay test sets could be generated for most ISCAS benchmark circuits containing up to 38,000 nodes.

In *1994*, *KraSniewski*, *Wrofiski* [10] derived the relationships between testability of path-delay faults and gate delay faults. They used the theoretical results to develop a fast procedure for *comprehensive assessment* of a given test sequence under the

different assumptions on what "detection of a delay fault by an input-pair" means and the experimental results demonstrated that a fair comparison of different test strategies requires several coverage metrics, corresponding to the different delay fault models and testability requirements.

In *1995*, *Takahashi, Watanabe* [15] presented a test for small gate delay faults in combinational circuits, called a *tenacious test* and described a method for generating tenacious tests. They considered a single gate delay fault in a circuit on the assumption of that each gate has some appropriate gate delay. First, They introduced a tenacious test $< V_1, V_2 >$ for a small gate delay fault on line L. The tenacious test $< V_1, V_2 >$ can propagate the effect of a small gate delay fault at line L to primary outputs by the delay effect. Next, they presented a method for generating tenacious tests by using a timed seven-valued calculus with consideration of delay of each gate in a circuit under test. Finally, experimental results are demonstrated for gate delay faults on *ISCAS'85* benchmark circuits. Experimental results show that they can obtain tenacious tests for small gate delay faults with high fault coverage.

In same year, to improve the *quality of delay fault testing* [17], they examined various delay models used in VLSI Testing. Their study included electrical-level simulation experiments with *HSPICE*. They showed the phenomenon which significantly affect the actual delays, but which are not taken into account by the existing models used in testing. Their analysis questions the test quality offered by test generation procedures used so far. Various test generation applications for delay fault models in described by *Cheng, Krstic* [19].

In 1998, a review on different delay fault models has given by *Majhi, Agrawal* in *1998* [37]. *Takahashi, Boateng and Takamatsu* [40] proposed a method of *diagnosing gate delay faults* using delay fault simulation. In the method, suspected faults are deduced by fault simulation and backward path tracing using diagnostic test-pairs with observed faulty responses. Also, by fault simulation using diagnostic test-pairs with fault-free responses, non-existent faults are deduced, and they are removed from the set of suspected faults. They presented experimental results on the *ISCAS'85* benchmark circuits. The experimental results show that by simple processes of backward path tracing

and fault simulation, this method achieves reasonable diagnostic resolutions in a short time.

Most industrial digital circuits contain three-state elements besides pure logic gates. A *gate delay Fault simulator* for combinational circuits was proposed by *Pohl, Walter* [42] that can handle three-state elements like bus drivers, transmission gates and pulled busses. The well-known delay faults -"slow-to-rise" and "slow-to-fall" -are considered as well as delayed transitions from isolating signal state "high impedance" to binary states '0' and '1' and vice versa. The presented parallel delay fault simulator distinguishes between non-robust, robust and hazard free tests and determines the quality of a test. Experimental results for *ISCAS85/89* benchmark circuits are presented as well as results for industrial circuits containing three-state elements.

In *1999, A.Chatterjee* et al. proposed tests with *linearity property for gate delay faults* to determine, at a required clock speed, whether a circuit under test is a marginal chip or not. The latest transition time at the primary output is changed linearly with the size of the gate delay faults when the proposed test is applied to the circuit under test. To author's knowledge no reports on algorithmic method for generating tests with linearity property for gate delay faults. *Takahashi, boating and Takamastu* proposed a new method to solve this problem [49]. The proposed method introduces a new extended timed calculus to calculate the size of a given gate delay fault that can be propagated to the primary output. The method has been applied to *ISCAS* benchmark circuits under the unit delay model.

In *2001*, to add more advancement in gate delay fault model, *Chen, Gupta and Breuer* [58] presented a new model to capture the delay phenomena associated with simultaneous to-controlling transitions. The proposed delay model accurately captures the effect of the targeted delay phenomena over a wide range of transition times and skews. It also captures the effects of more variables than table lookup methods can handle. The model helps improve the accuracy of static timing analysis, incremental timing refinement, and timing-based ATPG.

2.2.3 Transition Delay Fault Model

The transition fault model is considered as a logical model for a defect that delays a rising or falling transition at inputs and outputs of logic gates. There are two kinds of transition faults, i.e. slow-to-rise, slow-to-fall. The slow-to-rise (fall) transition fault temporarily behaves like a DC stuck-at-0 (1) fault. A test for a transition fault is a pair of input patterns, one (initialization pattern) to set up the initial state for the transition and another (propagation pattern) to cause the appropriate transition and observe its effect at a primary output. The propagation pattern is identical to a pattern that detects the corresponding DC stuck-at fault. The transition fault coverage is a measure of the effectiveness of the delay test in detecting large delay variations. Transition fault model defects for which the delay is large enough to cause a logical failure when the signal propagates along any path through the site of the fault. The main drawback of this model is the assumption of a large gate delay defect. Also, it is difficult to tell how small a delay fault can be, before it is not detectable. In practice, delay variations tend to be distributed over many circuit elements. Thus, many small gate delay faults, each undetectable as a transition fault, can give rise to a large path delay fault.

A satisfiable research is also going on this fault model; lets take a survey on this.

In *1995*, to improve the quality of delay fault testing [17], they examined various delay models used in VLSI Testing. Their study included electrical-level simulation experiments with *HSPICE*. They showed the phenomenon which significantly affect the actual delays, but which are not taken into account by the existing models used in testing. Their analysis questions the test quality offered by test generation procedures used so far.

In *1996*, a new delay fault model, called *double transition fault model* proposed by *Pomeranz, Reddy and Patel* [30]. Under this model, a fault is associated with a pair of lines and a pair of transitions on these lines. The model captures the effects of defects that increase the delays of two (or more) individual lines by an amount that causes the circuit to fail when signals are propagated through both lines. It thus provides a simplification of the path delay faults model that does not suffer from the exponential behavior of this model. They proposed a test generation procedure for double transition faults, based on reordering of a given test set for stuck-at faults. The procedure does not require enumeration of all double transition faults, and is thus applicable to circuits with

large numbers of lines. They presented experimental results of this procedure for several benchmark circuits.

In *1998*, a review on different delay fault models has given by *Majhi, Agrawal* in *1998* [37]. The speed of new VLSI designs is rapidly increasing. Assuring the performance of the circuit requires that the circuit be tested at its intended operating speed. The high cost of high-speed testers makes it impossible for the testers to follow the designs in terms of speed increase. This gap between the speed of the new circuits and the speed of the testers is not likely to disappear. *Krstic and Cheng* [43] in *1999*, focused on at-speed strategies for *testing high-speed designs on slower testers*. Conventional at-speed testing strategies assume that the primary inputs/outputs can be applied/observed at the circuit rated speed. This requires a high-speed tester. Their assumption is that a fast clock matching the speed of the designs is available. They described two classes of at-speed strategies that can be used on a low speed tester. The first class consists of testing schemes for which the test generation procedure is independent of the speed of the tester. These methods apply multiple input patterns in one tester cycle and the test application time for them can be long. The strategies in the second class of at-speed testing schemes integrate the tester's speed limitations with the test generation process. Due to constraints placed at the test generation process, these schemes might result in reduced fault coverage. To increase the fault coverage and reduce the test application time, the slow-fast-slow and at-speed strategies can be combined for testing high-speed designs on slower testers. They presented preliminary experimental results for at-speed schemes for slow testers for transition faults.

In *2002*, *Liu, Hsiao, Chakravarty and Thadikaran* [66] proposed *novel algorithms* for computing test patterns for transition faults in combinational circuits and fully scanned sequential circuits. The algorithms are based on the principle that s@ vectors can be effectively used to construct good quality transition test sets. Several algorithms are discussed. Experimental results obtained using the new algorithms show that there is a 20% reduction in test set size, test data volume and test application time compared to a state-of-the-art native transition test ATPG tool, without any reduction in fault coverage. Other benefits of this approach, viz. productivity improvement, constraint handling and design data compression are highlighted.

2.2.4 Segment Delay Fault Model

Heragu et al. have proposed a model that considers slow-to-rise and slow-to-fall defects on segments [20], whose length L, can be chosen from available statistics about the type of manufacturing defects. L can be as small as 1 (transition faults) or as large as the maximum logic depth (path faults). Once L is chosen, the fault list will comprise of all segments of length L and all paths whose entire length is less than L.

The segment delay fault model is an attempt to combine the advantages of the classical delay fault models while avoiding their limitations. Unlike the path delay fault model, this model can prevent an explosion of the number of faults to be considered. At the same time, a defect over a segment may be large enough to affect any path through it. This assumption seems more realistic than the transition delay fault model that requires the defect on a single line to be large enough to affect any path passing through it. Due to process variations, every gate in the circuit is affected and their delays increase only by a small amount. Again, randomly occurring defects are of very small sizes. Defects of this nature may produce delay faults only on longest paths. The segment delay fault tests may not be able to detect some of these defects.

A much research is going on this delay fault model; lets take a survey on this.

In *1995*, to improve the *quality of delay fault testing* [17], they examined various delay models used in VLSI Testing. Their study included electrical-level simulation experiments with *HSPICE*. They showed the phenomenon which significantly affect the actual delays, but which are not taken into account by the existing models used in testing. Their analysis questions the test quality offered by test generation procedures used so far.

In *1996*, *Heragu, Patel and Agrawal* [27] proposed an efficient combinational circuit *simulation technique* for the recently proposed segment delay fault model [20]. After simulation of a vector pair, activated segments are traced using a depth first search. A segment-numbering scheme finds the number of faults to be simulated. A labeling technique generates edge labels to compute a unique label for each segment fault. The use of labels avoids explicit storing of fault lists and allows efficient access to previously detected segment faults. Experimental results demonstrate several advantages of the segment delay fault model. First, the total number of faults remains manageable for small segment lengths. Second, many segments, not included in any robustly testable path fault,

may have robust segment delay fault tests. Generating tests for such segments may increase the delay defect coverage.

In *1999*, they proposed another *simulation-based technique* that uses a genetic algorithm (GA) to generate tests for delay faults on segments of any given length [46]. At every line, they assumed that an upper bound on the number of testable segment faults that originate there is known. Such a bound is efficiently computed by an implication-based technique. The fitness function for the GA is derived from an objective function that favors vectors, which might detect a large number of faults. This is accomplished by a simulator used as abase engine, by dynamically identifying a line m with the highest upper bound for the number of segments on which faults can and are yet to be tested, and by ranking vectors according to their ability to target the simultaneous objectives of invoking a transition on m and maximizing the number of signals that propagate robustly in the fanout cone of m. Rather than limiting the number of generations of evolution in the GA, they obtain improved results by using the diversity of the individuals in a population as a stopping criterion. Results indicate that for small segment lengths, reasonable robust segment delay test coverages can be obtained for most benchmark circuits. Also, the tests generated using the segment delay fault model detect a large number of transition and path delay faults. For example in the benchmark circuit *c3540*, tests generated for faults on segments of length 5 had a transition fault coverage of 96.1% and were able to detect 9,246 path faults.

2.2.5 Functional Delay Fault Model

In *1995*, *Pomeranz and Reddy* [16] proposed a functional fault model for delay faults in combinational circuits and described a *functional test generation procedure* based on this model. The proposed method is most suitable when a gate-level description of the circuit-under-test, necessary for employing existing gate-level delay fault test generators, is not available. It is also suitable for generating tests in early design stages of a circuit, before a gate-level implementation is selected. It can also potentially be employed to supplement conventional test generators for gate-level circuits to reduce the cost of branch and bound strategies. A parameter called Δ is used to control the number of functional faults targeted and thus the number of tests generated. If A is

unlimited, the functional test set detects every robustly testable path delay fault in any gate- level implementation of the given function. An appropriate subset of tests can be selected once the implementation is known. The test sets generated for various values of Δ are fault simulated on gate-level realizations to demonstrate their effectiveness.

In *1999*, *Michalel* [45] proposed and evaluate two frameworks for functional level ATPG for delay faults in combinational circuits. Although functional delay fault models have been proposed in [16], no systematic methodologies for ATPG have been presented in the literature. The proposed frameworks apply to any proposed fault model and utilize established techniques such as *Reduced Ordered Binary Decision Diagrams* (ROBDDs) and Boolean Satisfiability (SAT).

They added the further implementation in this work and presented an *ATPG tool for functional delay faults* in combinational circuits in *2002*, which applies to the single-input transition (SIT) and the multi-input transition (MIT) fault models, and is based on Reduced Ordered Binary Decision Diagrams (ROBDDs). They were able, for the first time, to identify all faults that did not have any SIT tests, and generated all SIT tests for nonredundant faults in combinational circuits. They also provided methodologies for efficient generation of MIT tests. Their experimental results on the *ISCAS'85* benchmarks is by far superior to existing methods as well as a Satisfiability-based tool that we have developed for comparative purposes. The presented tool, coupled with advancements in path delay fault coverage, shows that both the SIT and MIT functional models are very useful in ATPG for robust path delay faults for synthesized circuits.

Several functional delay fault models have been proposed before to allow functional test generation for delay faults. *Pomeranz, Reddy* [52] extended these models to accommodate functional descriptions where inputs and outputs are more naturally represented by vectors carrying non-binary values. Such vectors are typical of high- level functional descriptions. Experimental results show that using the vector-based models does not result in loss of gate-level path delay fault coverage.

Existing gate-level fault models are not well suited to test generation for circuits that contain modules whose logic implementation and timing behavior are unspecified. A high-level fault model called the *coupling fault (CF) model* presented by *Hayes* [61] in *2001*, which aims to cover both functional and timing faults in an

integrated manner. Intuitively, a (single) CF denoted $x_i|z_j$ exists between input x_i and output z_j of a module if $x_i|z_j$ blocks any dynamic effect of x_i on z_j .The set of test vectors $CTS_{x_i|z_j}$ that detect $x_i|z_j$ is represented by the Boolean difference of z_j with respect to x_i .A pair of adjacent vectors in $CTS_{x_i|z_j}$ constitutes a coupling delay test. They presented the basic properties of coupling faults and test sets, focusing on the relationship between coupling tests and other high-level tests. A coupling test set provides powerful, realization-independent coverage of stuck-at faults. Coupling delay tests can detect all robust path delay faults in any realization of a function.

2.2.6 Line Delay Fault Model

We can combine the relevant features of the transition and path delay fault models to define a line delay model. A rising (falling) line delay test will test the longest sensitizable path passing through a target line producing a rising (falling) transition on it. With this model, the coverage is measured for all lines with two possible transitions. Thus, the maximum number of faults (or tests) is twice the number of lines. (For example, in c6288, we will consider only 12576 line delay faults whereas the total number of possible path faults is $\approx 1.98 * 10^{20}$.) Yet, the test criterion is similar to path delay fault, and not like gate or transition delay fault. In general, a test will cover several lines. Conventional path delay test generators attempt to derive robust tests for a subset of paths in the circuit, based on some path selection criterion such as the worst-case path selection or a threshold-based path selection. However, a large number of these paths may not be robustly testable and hence the test coverage of the targeted paths can be very low. The new *coverage metric* [37] seeks to remove this deficiency by attempting to derive a pair of line delay tests for each line in the circuit.

The basic idea of an iterative approach for generating a robust test was first proposed by *Park and Mercer*. They have followed an approximate method where the search space for test generation is biased to and a test along a path whose propagation delay is greater than or equal to a predefined threshold value. *Bose* preselects paths in a given range of lengths and shows that despite low path coverage, a high gate (or line) coverage can be obtained. In that case, however, lines are not tested through longest sensitizable paths. The method of *Majhi* et al., on the other hand, is an exact method for

generating a robust test for the longest testable path through each line. To facilitate the simultaneous consideration of robust and non-robust tests, they use a 9-value logic system.

A *line delay coverage metric* [37] proposed by *Majhi and Agrawal* in *1997*. The motivation of defining the line delay test is to robustly detect the smallest incremental delay defect associated with a rising or falling transition at any line. Suppose, Δ_L is the incremental delay of a rising or falling transition through line L. Then, for detection of this delay fault,

$$\Delta_L + T_P > T_C \quad \text{or} \quad \Delta_L > T_C - T_P \qquad (1)$$

Where T_C = system clock period and T_P = nominal delay of the path P through which L is tested. From relation (1), we determine that the smallest incremental delay fault on L can be detected via the path through L having the longest nominal delay $(T_P)_{max}$, i.e.,

$$(\Delta_L)_{min} = T_C - (T_P)_{max} \qquad (2)$$

By sensitizing the longest path through L, we are able to detect the delay fault of the smallest size. However, simultaneous delay variations are possible for other gates on P due to correlation with L. Suppose, delays of other gates increase. Then the line delay test for L will detect a delay fault of even smaller size. If the delay of other gates reduces while that of L increases, the sensitivity of the test reduces. Considering correlation of delays, this later case is less probable.

The basic assumption associated with the line delay fault model is that the delays of all gates are not reduced below their nominal values. Main advantages of this fault model are that the number of faults is limited to twice the number of lines in the circuit and almost all lines can be tested. Since the fault is tested along the longest propagation path, the system timing failures caused by the smallest localized delay defects or the accumulation of distributed delay defects can be detected. In transition fault model, a delay test is obtained along any arbitrary path because the size of delay fault is assumed to be large enough to be tested via any path through the fault site. For the gate delay fault model one must specify exact sizes of delay defects. Difficulties arise when accurate information on delays is not available. Transition and gate delay faults do not model the distributed delay defects along a target path. The line delay model, on the other hand, retains many advantages of the transition and gate delay fault models, while

alleviating the major drawback of the path delay model (viz., too many paths to be tested and the low fault coverage).

In order to derive a line delay test for a given line, we first target the longest structural path. If that path is not robustly testable then we go for the next longest path and so on, until we find a test for the longest robustly testable path. The limitation in this approach is that in addition to this robustly testable path there could exist a nonrobust test for some longer path through the target line. In such a case, the robust test would miss a line delay fault that only causes the longer nonrobustly testable path to fail. To overcome this limitation one may include any possible nonrobust tests for all paths that are longer than the longest robustly testable path. Another limitation of this model is that in case of certain distributed delay defects the derived tests will fail to detect some of the delay faults that are not targeted. We consider only one path through any given line for determining a line delay test. However, there may be some other paths of the same length (or shorter) through the target line, with distributed delay defects exceeding the permissible propagation delay. Consider the three paths shown in Figure 2.4. Suppose, all three paths have the same delay. Let us assume that paths 1 and 2 are the longest structural paths selected for testing lines A and B, respectively. Let us further assume that the smallest incremental delay that is detectable for each path is Δ (i.e., $\Delta = T_C - T_P$), where T_P is the nominal delay of each of the paths and T_C is the clock period. If the incremental delay of nodes A and B are $\Delta - \varepsilon$, where ε is small, then paths 1 and 2 will pass their tests. However, path 3 has a fault that is not detected by the test vectors though it is a detectable delay fault.

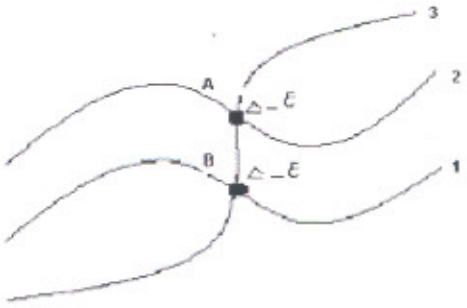


Fig 2.4 **Line Delay Faults Limitation**

The basic assumption associated with the line delay fault model is that the delays of all gates are not reduced below their nominal values. More than one faulty gate can occur in practice due to correlation between delays of gates. In that case many gates in paths 1 and 2 will have increased delays and it is more likely that the tests for those faults will show failures. There can be several ways of dealing with the situation depicted in Figure 4. When there are several longest paths of equal length through a target line, we can consider all such paths to increase the confidence level for the tests obtained. However, this can lead to a potentially large number of paths to be tested in some circuits.

In *2002, Krishnamachary and Abraham* developed new techniques for detecting both stuck-open faults and resistive open faults, which result in increased delays along some paths. The improved detection of CMOS open defects is made possible by a new delay fault model which combines the advantages of the gate delay fault model and the path delay fault model. They develop a test generation methodology for this fault model, which enables generation of test vectors that test a percentage of the longest sensitizable paths in the design and also test each net for spot defects through their longest sensitizable paths. Real delay values are used to determine the true critical paths in the circuit. The high degree of effectiveness of this fault model under realistic assumptions for process characteristics is first enumerated, and experimental results demonstrate the improved coverage possible with the proposed approach.

2.3 Comparison of Various Delay Fault Models

A comparison is given in tabular form of all fault models that discussed above; it includes the comparison according to their advantages and limitations. See in Table 2.1:

Fault Models	Advantages	Limitations
Path Delay	Distributed failures are considered.	Impossible to enumerate all possible paths.

Gate Delay	All gates can be modeled.	Distributed failures not considered. Exact defect size not possible.
Transition Delay	Easy to model all gates.	Distributed failures not considered.
Segment Delay	Considers general delay defect from spot to distributed failures.	Longest delay path may not be tested.
Functional Delay	Fault effect modeled at a higher level than logic for function modules.	No systematic methodologies for ATPG are efficiently present.
Line Delay	All gates are modeled. Distributed failures considered. Better coverage metric. Additional fault coverage by using multi-pass technique.	Existence of non-robust test. May fail for some shorter paths (hard constraint).

TABLE 2.1 **Comparison of Various Delay Fault Models**

3.1 Introduction to Path Delay Faults

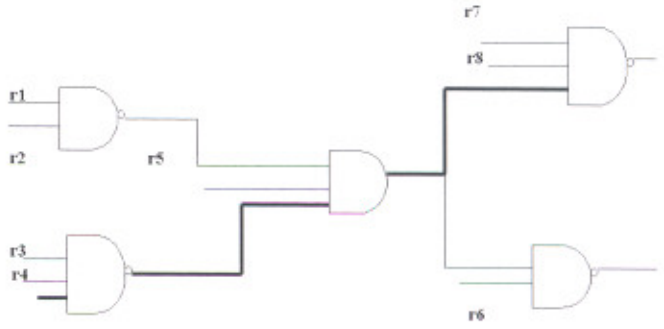


Fig 3.1 **Path Delay Faults**

Smith first proposed the path delay fault model. This model has received greater attention than the gate delay and transition fault models and has been quite extensively studied in [6], [10], [11], [12], [14], [22], [23], [24], [25], [29], [31], [33], [34], [35], [36], [37], [38], [39], [41], [44], [53], [57], [63], [70], [75].

It includes considerable amount of research is being done on path delay faults. In path delay fault model, any path with a total delay exceeding the system clock interval is said to have a path delay fault. These models distributed defects that affect an entire path. See in Figure 1.7, for each physical path P, connecting a primary input to a primary output of the circuit, there are corresponding two delay paths. The rising path (falling path) is the path traversed by a transition whenever it passes through an inverting gate. We present the following definitions that are frequently used in path delay fault testing [74].

Definition 1: Let G be a gate on path P in a logic circuit, and let r be an input to gate G; r is called an *off-path sensitizing input* if r is not on path P.

Definition 2: A two-pattern test $<V_1, V_2>$ is called a *robust test* for a delay fault on path P, if the test detects that fault independently of all other delays in the circuit.

Definition 3: A two-pattern test $<V_1, V_2>$ is called a *nonrobust test* for delay fault on path P, if it detects the fault under the assumption that no other path in the circuit involving the off-path inputs of gates on P has a delay fault.

As we mentioned above that most of the research is going on *path delay faults*. There are various models, algorithms, methods, and procedures proposed path delay faults. Lets take a survey on these faults.

3.2 Robust and Non-Robust Path Delay Fault Test Generation

3.2.1 Five-valued Logic for Robust Tests and Three-valued Logic for Non-Robust Tests

Generation of a test for a path-delay fault requires placing the appropriate transition at the origin of the path and justifying the required off-path inputs of all gates on the path. This is easily accomplished using the five-valued algebra [80].

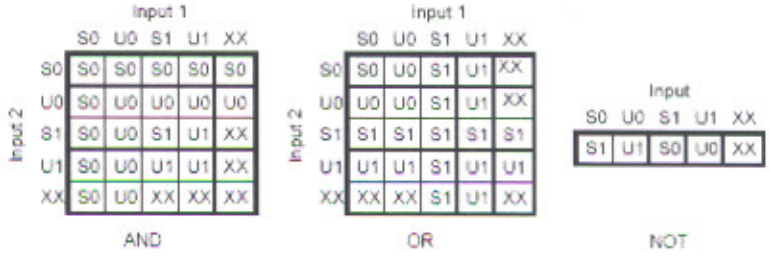


Fig 3.2 **Five-valued Algebra for Path-Delay Test**

Robust Test Generation Consider the path-delay fault ↓ P3 in Figure 1.11. We proceed as follows:

1. Place a transition at the path origin, B = F0.
2. Propagate value F0 to line E, from Figure 1.11, C = U0) E = F0.
3. G = F0) J = R1.
4. F0 is interpreted as U0 for off-path logic, Q = U0.
5. Propagate value R1 from J to K, using Figure 1.11 set H = S0) K = R1.
6. Justify H = S0, from Figure 1.12 set A = S0.
7. Test is A = S0, B = F0, C = U0; or V1 = 01X, V2 = 000.

We should remember that the value S0 implies that input A should hold its value steady for two vectors. We observe that this test is different from the one considered in Example 1. This test is robust and the reader can verify that it will not be invalidated irrespective of the delay of P2. The procedure of the above example can be implemented in many ways. The path can be sensitized starting at the output, or all off-path signals can be set at once and then justified. We chose to sensitize the path from input because if sensitization becomes impossible at some gate, then we can immediately conclude that no robust test is possible. This simple example has only limited choices. With increasing number of inputs of a gate, justification choices also increase. In general, when the circuit has reconvergent fanouts, the test generation procedure frequently has to use backtracks.

For some paths, robust tests are not possible and we must generate non-robust tests. As discussed before, non-robust tests only require static sensitization. That means all signals except the origin of the path under test can have arbitrary values in the first vector (V_1). This condition is easily incorporated in the multi-valued algebra by simple substitutions, S0 ← U0 and S1 ← U1.

Non-Robust Test Generation To generate a robust test for path-delay fault ↑ P2 in Figure 1.7. We proceed as follows:

1. Place a transition at path origin, B = R1.
2. Propagate R1 to E, from Figure 1.11 set C = S0.
3. R1 is interpreted as U1 for off-path logic, G = U1) J = U0.
4. Q = R1, Propagate R1 to H, from Figure 1.11 set A = U1.
5. H = R1, Propagate R1 to K, from Figure 1.11, must set J = S0) conflict since J = U0 in step 3.
6. Since no step has any alternatives, a robust test is not possible.

For a non-robust test we change S0 and S1 to U0 and U1, respectively (static sensitization.) Now the Step 5 requirement becomes J = U0, which is consistent with Step 3. The non-robust test is A = U1, B = R1, C = U0 (changed from S0); or V_1 = X0X, V_2 = 110.

An alternative and simpler method for generating non-robust tests is to derive single input change (SIC) tests. For a SIC test, the two vectors V_1 and V_2 in the test differ in exactly one bit. We first find V_2 to statically sensitize the entire path using any

combinational ATPG procedure. V_1 is then obtained by just changing one bit in V_2 that corresponds to the origin of the path. It can be easily shown that every non-robustly testable path must have a SIC test.

The procedure of this example attempts to find a non-robust test only when a robust test is impossible. In view of the fact that the reliability of non-robust tests is questionable (see Example of Non Robust Tests), there is merit in finding as many robust tests as possible. The presence of robust tests for some paths can improve the reliability of non-robust tests for other paths. For example, in Figure 7 six path-delay faults,

↑ P1, ↓ P1, ↑ P3, ↓ P3, ↑ C - E - G - J - K and ↓ C- E -G -J -K, are robustly testable. Example of non-robust tests shows that ↑ P2 only has a non-robust test. By including the six robust tests we can ensure that if the circuit passes those, there will be no delayed signal at off-path inputs of the path P2. We can conclude that in the presence of the other four tests, the non-robust test for ↑ P2 is as good as a robust test. Such a test is called a validatable non-robust (VNR) test.

3.2.2 Nine-valued Logic for Robust Tests and Three-valued Logic for Non-Robust Tests

In this system, we will consider three logic states 0, 1, X for two clock periods for three intervals (initial, intermediate and final). So we have

$$3^3 = 27 \text{ states}$$

- The transition states from 1 - 4 and 10 - 13, in which the logic values are fully specified are represented by unique values i.e. (S0, G0, FT) and (S1, G1, RT) respectively.
- The composite states (transition states 5 - 9 and 14 - 18) are abosrbed together and represnted by X0 and X1 respectively.
- The last 9 states (19 - 27) are collapsed to a single logic value XX since the final logic value is X (don't care).

G0 and G1 represent the static 0 and static 1 hazard represented, as 0-1-0 and 1-0-1 and these are explicity used for ***Non-Robust Tests.***

S0 and S1 represnts ***no Static Hazard*** represnted as 0 0 and 1 1 respectively.

FT and RT represents ***no Dynamic Hazrad*** is present represnted as 1 0 and 0 1 respectively.

X0, X1 and XX represents that ***hazard is possible*** and represnted as X 0, X 1 and X X.

State id	Possible States	Fully Specified	Composite States
1	000	[S0]	-
2	010	[G0]	-
3	100	[FT]	-
4	110	[FT]	-
5	0X0	-	[S0, G0]
6	X00	-	[S0, FT]
7	XX0	-	[S0, G0, FT]
8	1X0	-	[FT, FT]
9	XI0	-	[G0, FT]
10	111	[S1]	-
11	101	[G1]	-
12	011	[RT]	-
13	001	[RT]	-
14	0X1	-	[RT, RT]
15	1X1	-	[S1, G1]
16	X01	-	[RT, G1]
17	XX1	-	[S1, G1, RT]
18	X11	-	[RT,S1]
19	00X	-	-
20	11X	-	-
21	X0X	-	-
22	X1X	-	-

23	0XX	-	-
24	1XX	-	-
25	01X	-	-
26	10X	-	-
27	XXX	-	-

TABLE 3.1 **Nine-valued Algebra**

Test Generation Procedure:

- First an implication w.r.t to transition (Rising or Falling) at the input of the path and other inputs unspecified XX.
- The offpath, set to the suitable logic, according to their robust and non-robust conditions, the main aim is to do path sensitization.
- Then a backtrace from PO to PI, in breath first manner, so as to saisfy the objectives.
- During backtrace if we create a new objective on an input of gate G and this input happens to be fanout branch, then we create the same logic on fanout stem.
- After doing the implication all fanout branches as tried.
- Hence before setting an objective on any fanout branch during backtrace, we check whether or not it has already been tried by previous objective.

Generation of Test of Opposite Transition:

- Once we find a robust test for a transition on a target path, instead of generating a completely new test for opposite transition we derive from second test by replacing X1 (X0) with steady values S1 (S0) in the already generated test and reversing the transition at the input of the target path.
- The principle is that since the second vector is a test for a stuck fault on the PI at the source of the pathn, the opposite transition will always reach the destination of the path if the sensitizing condition remains unchanged.
- In most cases this modified vector pair will be a valid robust/nonrobust test for the opposite transition.

- If the implied logic values are a subset of (S0 and G0 C X0 and S1 and G1 C X1) of the primary objective logic values or are the same, then the derived test will be a robust test.
- If at least one of the primary objective logic values is S1 (S0) and the implied logic value becomes G1 (G0), then the derived test will be a non – robust test.

In the following figure:

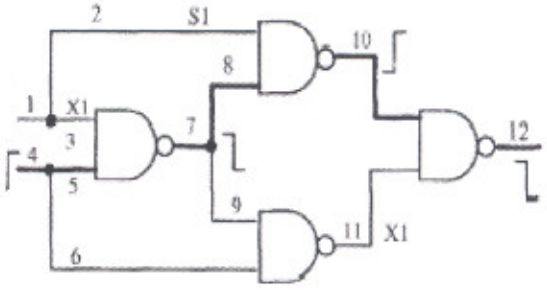


Fig 3.3 (a) **Generate Test with Opposite Transition –Rising Transition**

For this robust test vector consist of these values.

Path = 4-5-7-8-10-12 with rising transition

1 → U1
2 → S1
3 → U1
4 → RT
5 → RT
6 → U0 (Off path)
7 → FT
8 → FT
9 → U1
10 → RT
11 → U1
12 → FT

Corrsponding to same path, we will check the falling transition for that test value set is:

1 → S1
2 → U1
3 → S1
4 → FT
5 → FT
6 → S0
7 → RT
8 → RT
9 → U0
10 → FT

11 → S1 12 → RT

In this test value set primary objective is 11=S1, so it is ***a non-robust test.***

Consider another figure,

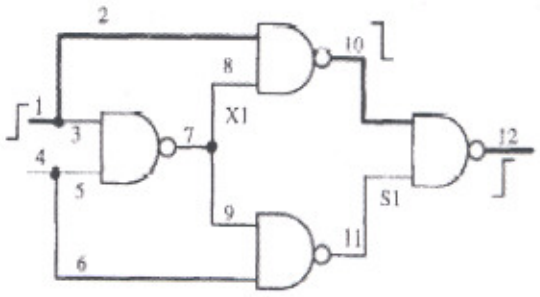


Fig 3.3 (b) **Generate Test with Opposite Transition – Falling Transition**

In this case

Path is, 1-2-10-12. For Rising Transition, for robust test vectors consiting values:

1 → U0	7 → U1
2 → RT	8 → U1
3 → U0	9 → U1
4 → S0	10 → FT
5 → S0	11 → S1
6 → S0	12 → RT

Consider the same path for falling transition, for that test value set is:

1 → U1	7 → S1
2 → FT	8 → S1
3 → U1	9 → S1
4 → S0	10 → RT
5 → S0	11 → U1
6 → U0	12 → FT

In this case for falling transition, ***there is robust test***, as we know that S1 and G1 ⊂ X1.

3.2.3 Seven and Ten-valued Logic for Robust Tests and Three-valued Logic for Non-Robust Tests

Based upon seven valued logic $A_7 = \{B_7, C_7\} = [\{0S, 0S', 1S, 1S'\}, \{0X, 1X, XX\}]$ where as 0X, 1X, XX consisting the subset of basic values 0S, 0S', 1S, 1S'.

- $0s$: stable 0
- $1s$: stable 1
- $0s$: final value 0 but not stable at 0
- $1s$: final value 1 but not stable at 1

Note that $0s'$ (1s') can represent a static-0 (static-1) hazard, or 1(0) to 0(1), transition with or without the presence of dynamic hazards. Using this basic system, the following composite values are included in L_7 [80], [81], [83].

- $U0 = \{0s, 0s'\}$: final value 0 may or may not be stable
- $U1 = \{1s, 1s'\}$: final value 1 may or may not be stable
- $X = \{0s, 0s', 1s, 1s'\}$: an unknown value with no constraints

The Hasse diagram for this seven-valued logic is shown below:

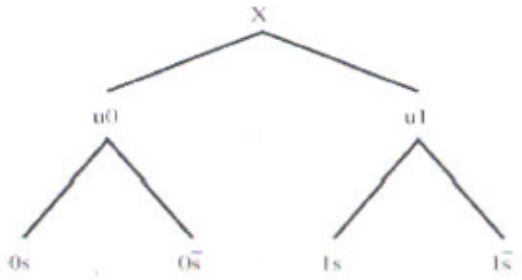


Fig 3.4 **Hasse Diagram for Seven-valued Logic**

0S' denotes the falling path, where as IS' denotes the rising path.

Consider the case of **AND Operation on 1S' and X.**

B_7 (1S' X) = B_7 (X) = {0S, 0S', 1S, 1S'}

Rather than doing this manner it can be expressed as:

B_7 (1S') B_7 (X) = 1S' {0S, 0S', 1S, 1S'} = {0S, 0S', 1S'}

So from here we found, B_7 (1S' X) ≠ B_7 (1S') B_7 (X).

Consider the case of **OR Operation on 0S' and X.**

B_7 (0S' ∨ X) = B_7 (X) = {0S, 0S', 1S, 1S'}

Rather than doing this manner it can be expressed as:

$B_7(0S') \vee B_7(X) = 0S' \vee \{0S, 0S', 1S, 1S'\} = \{0S', 1S, 1S'\}$

So from here we found, $B_7(0S' \vee X) \neq B_7(0S') \vee B_7(X)$.

Consider the case of **XOR Operation on 0S' and X.**

$B_7(0S' \oplus X) = B_7(X) = \{0S, 0S', 1S, 1S'\}$

Rather than doing this manner it can be expressed as:

$B_7(0S') \oplus B_7(X) = 0S' \oplus \{0S, 0S', 1S, 1S'\} = \{0S', 1S'\}$

So from here we found, $B_7(0S' \oplus X) \neq B_7(0S') \oplus B_7(X)$.

Due to these inequalities, three more logic values proposed, that's why **generation of 10 valued logic.**

A new ten-valued logic for ATPG for path delay faults has been proposed [80], [81], [83].

A ten-valued logic is represnted as $A_{10} = \{B_{10}, C_{10}\} = [\{0S, 0S', 1S, 1S'\}, \{0X, 1X, XX, XS'^{0}, XS'^{1}, XS'\}]$ with

The Hasse diagram for this ten-valued logic is shown below:

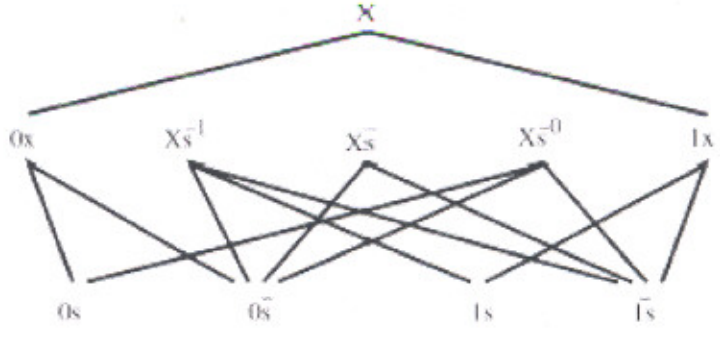


Fig 3.5 **Hasse Diagram for Ten-valued Logic**

$B_{10}(0X) = \{0S, 0S'\}$

$B_{10}(1X) = \{1S, 1S'\}$

$B_{10}(X) = \{0S, 0S', 1S, 1S'\}$

$B_{10}(XS') = \{0S', 1S'\}$

$B_{10}(XS'^{0}) = \{0S, 0S', 1S'\}$

$B_{10}(XS'^{1}) = \{1S, 0S', 1S'\}$

is complete with respect to F= { Λ, $\vee$, $\oplus$,$\neg$}.

Λ	0s	0s'	0x	1s	1s'	1x	Xs'	Xs'^{0}	Xs'^{1}	X
0s	0s	0s	0s	0s	0s	0s	0s	0s	0s	0s
0s'	0s	0s'	0x	0s'	0s'	0s'	0s'	0s	0s'	0x
0x	0s	0x	0x	0x	0x	0x	0x	0s	0x	0x
1s	0s	0s'	0x	1s	1s'	1x	Xs'	Xs'^{0}	Xs'^{1}	X
1s'	0s	0s'	0x	1s'	1s'	1s'	Xs'	Xs'^{0}	Xs'	Xs'^{0}
1x	0s	0s'	0x	1x	1s'	1x	Xs'	Xs'^{0}	Xs'^{1}	X
Xs'	0s	0s'	0x	Xs'	Xs'	Xs'	Xs'	Xs'^{0}	Xs'	Xs'^{0}
Xs'^{0}	0s	0s	0x	Xs'^{0}	Xs'^{0}	Xs'^{0}	Xs'^{0}	Xs'^{0}	Xs'^{0}	Xs'^{0}
Xs'^{1}	0s	0s'	0x	Xs'^{1}	Xs'	Xs'^{1}	Xs'	Xs'^{0}	Xs'^{1}	X
X	0s	0x	0x	X	Xs'^{0}	X	Xs'^{0}	Xs'^{0}	X	X

(a) AND-gate

$\vee$	0s	0s'	0x	1s	1s'	1x	Xs'	Xs'^{0}	Xs'^{1}	X
0s	0s	0s'	0x	1s	1s'	1x	Xs'	Xs'^{0}	Xs'^{1}	X
0s'	0s'	0s'	0s'	1s'	1s'	1s'	1s'	Xs'	Xs'	Xs'
0x	0x	0s'	0x	1x	1s'	1x	Xs'	Xs'	Xs'	Xs'
1s	1s	1s'	1x	1s	1s'	1x	1s'	1x	1x	1x
1s'	1s'	1s'	1s'	1s'	1s'	1s'	1s'	1s'	1s'	1s'
1x	1x	1s'	1x	1x	1s'	1x	1s'	1s'	1s'	1x
Xs'	Xs'	Xs'	1s'	1s'	1s'	1s'	Xs'	1s'	1s'	1s'
Xs'^{0}	Xs'	Xs'	Xs'	1x	1s'	1s'	1s'	1s'	1s'	1s'
Xs'^{1}	Xs'	Xs'	Xs'	1x	1s'	1s'	1s'	1s'	Xs'	1s'
X	X	Xs'	1s'	1x	1s'	1s'	1s'	1s'	1s'	X

(b) OR-gate

Fig 3.6 **Implication Tables of the 10-valued Logic for Robust Test Generation**

$\neg$	0s	0s'	0x	1s	1s'	1x	Xs'	Xs'^{0}	Xs'^{1}	X
0s	1s'	1s	1x	0s'	0s	0x	Xs'	1x	X	X
0s'	1s'	1s	1x	0s'	0s	0x	Xs'	1x	X	X
0x	1s'	1s	1x	0s'	0s	0x	Xs'	1x	X	X
1s	1s'	1s	1x	0s'	0s	0x	Xs'	1x	X	X
1s'	1s'	1s	1x	0s'	0s	0x	Xs'	1x	X	X
1x	1s'	1s	1x	0s'	0s	0x	Xs'	1x	X	X
Xs'	1s'	1s	1x	0s'	0s	0x	Xs'	1x	X	X
Xs'^{0}	1s'	1s	1x	0s'	0s	0x	Xs'	1x	X	X
Xs'^{1}	1s'	1s	1x	0s'	0s	0x	Xs'	1x	X	X
X	1s'	1s	1x	0s'	0s	0x	Xs'	1x	X	X

(c) NOT-gate

$\oplus$	0s	0s'	0x	1s	1s'	1x	Xs'	Xs'^{0}	Xs'^{1}	X
0s	0s	0s'	0x	1s	1s'	1x	Xs'	Xs'^{0}	Xs'^{1}	X
0s'	0s'	0s'	0s'	1s'	1s'	1s'	Xs'	Xs'	Xs'	Xs'
0x	0x	0s'	0x	1x	1s'	1x	Xs'	Xs'^{0}	Xs'^{1}	X
1s	1s	1s'	1x	0s	0s'	0x	Xs'	Xs'^{1}	Xs'^{0}	X
1s'	1s'	1s'	1s'	0s'	0s'	0s'	Xs'	Xs'	Xs'	Xs'
1x	1x	1s'	1x	0x	0s'	0x	Xs'	Xs'^{1}	Xs'^{0}	X
Xs'	Xs'	Xs'	Xs'	Xs'	Xs'	Xs'	Xs'	Xs'	Xs'	Xs'
Xs'^{0}	Xs'^{0}	Xs'	Xs'^{0}	Xs'^{1}	Xs'	Xs'^{1}	Xs'	Xs'^{0}	Xs'^{1}	X
Xs'^{1}	Xs'^{1}	Xs'	Xs'^{1}	Xs'^{0}	Xs'	Xs'^{0}	Xs'	Xs'^{1}	Xs'^{0}	X
X	X	Xs'	X	X	Xs'	X	Xs'	X	X	X

(d) XOR-gate

Fig 3.6 **Implication Tables of the 10-valued Logic for Robust Test Generation**

Logic Values needed for the Non-Robust ATG

In non-robust test we have to consider the final values only. So the test generation reduced to propagation vector V_2. But in case of robust tests it depends upon the static values (initial values) also.

So it can be represented as three-valued logic in which initial value, final value and transition signal.

In this case 0X reprsents the falling value, 1X represents the rising value.

∧	0x	1x	X
0x	0x	0x	0x
1x	0x	1x	X
X	0x	X	X

(a) AND-gate

∨	0x	1x	X
0x	0x	1x	X
1x	1x	1x	1x
X	X	1x	X

(b) OR-gate

¬	0x	1x	X
0x	1x	0x	X
1x	1x	0x	X
X	1x	0x	X

(c) NOT-gate

⊕	0s	1s	X
0s	0s	1s	0s
1s	1s	0s	X
X	X	X	X

(d) XOR-gate

Fig 3.7 **Implication Tables of the 3-valued Logic for Non-Robust Test Generation**

The Implication Procedure for 10-valued Logic

The performance of ATG approach depends upon the power of implication procedure.

- We found that, by maximizing the number of logic values, which can be uniquely determined by the implication procedure, is vital to perform the determiministic ATG process efficiently.
- This ATG Generator executes these both forward and backward implications.
- This implication procedure is restricted to 10 valued logic implications.
- This implication procedure follows the Completeness Criterion.
- Taking any path and assigning the appropriate values to the offpath inputs for sensitization can do this, as shown in following table 3.2.

Gate Type	Robust ATG		Non-Robust ATG	
	Rising	Falling	Rising	Falling
AND/NAND	1x	1s	1x	1x
OR/NOR	0s	0x	0x	0x
XOR/XNOR	0s/1s	0s/1s	0x/1x	0x/1x

TABELE 3.2 **Allowed Logic Values for Off-Path Sensitizing Inputs in Robust and Non-Robust ATG**

3.3 Proposed Logic for Robust Tests and Non-Robust Tests

In this chapter all types of logic like five-valued, seven-valued, nine-valued and ten-valued logic for robust test generation and for non-robust test generation three-valued logic for path delay faults has been fully described. From above we concluded that:

- We found that, by maximizing the number of logic values, which can be uniquely determined by the implication procedure, is vital to perform the determiministic ATG process efficiently.
- So 10-valued logic is efficient for robust test generation and 3-valued logic for non-robust test generation.
- But Sequential algorithm is taking more CPU time and more memory consumption.
- There is need of parallelized approach i.e. some parallel or distributed algorithm is needed.

The next chapter will describe parallelization of test generation for path delay faults.

Chapter 4 Parallelization of Test Generation for Path Delay Faults

4.1 Introduction

In earlier literature parallelization of path delay faults, parallelization is done by two methods; one is Bit level Parallelism mentioned in [14] and second is robust and non-robust path delay fault simulation by parallel processing of patterns [82].

Bit Parallel Test Generation

- This approach can only applied on Non-Robust Test Generation of PDF.
- In Non Robust Test Generation only final value will change, so three-valued logic system is better, i.e. 0,1, X. It means $\log_2 3 = 2$ bits. If machine word length is L then L no. Of logical values, can be store in 2 words.
- Each bit level represents one logic value.

Bit id	Logic value	0 – bit	1 – bit
0	0	1	0
1	1	0	1
2	X	0	0
3	Conflict	1	1

Fig 4.1 **Non-Robust Test Pattern Generation**

Fault Parallel Test Pattern Generation

- In this we will take L no. of faults simultaneously.
- First L paths will be sensitized and the resulting implications be performed.
- As long as there is at least one unjustified logic is there, a backtrace procedure is performed and bit parallel implications are made from primary inputs.

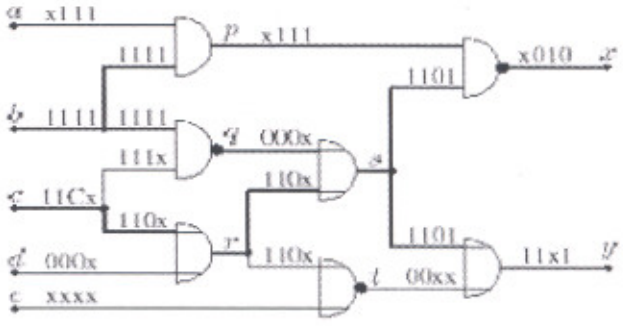


Fig 4.2 **Performing FPTPG**

These paths

1. b – p – x
2. b- q – s – x
3. c – r – s – x
4. c – r – s – y

will be treated as parallel from bit level 0 to bit level 3.

Bit level 3 Bit level 2 Bit level 1 Bit level 0

It means four-bit level, in which each bit level represents one logical value consisting two bits. **1 1 (C) 0 0 (X) 0 1 (1) 1 0(0)**

- These show the resulting logic values of the four bit levels after sensitizing the paths and performing the implications.
- The advantage of FPTG is that it is possible to treat multiple paths simultaneously and not sequentially.

Alternative Parallel Test Pattern Generation (APTPG)

- In order to examine several alternatives of test patterms simultaneously, APTPG is performed.
- If the result of backtrace indicates that at various primary inputs 0 and 1 is required, L alternatives can be considered simultaneously.

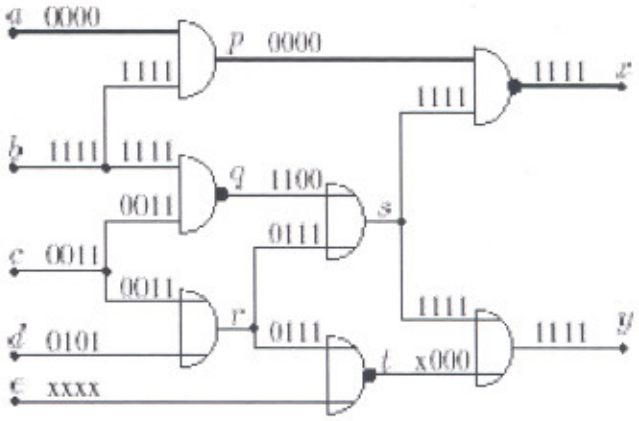


Fig 4.3 **Performing APTPG**

- In general we can consider all possible value assignments at Log_2L primary inputs.

Combination of FPTPG and APTPG

- **(Task 1)** In order to treat easy-to-test faults as efficient as possible, we start with FPTPG and identify a lot of testable and redundant paths; this leads to a speed-up of test pattern generation.
- **(Task 2)** If backtracking is necessary, we dynamically pass over APTPG.
- **(Task 3)** The effort of resensitizing a target path by simply flattening the active bit of a logical value to multiple bit levels.
- With the help of APTPG, we will be able to Detect Hard to detect faults.
- Combination of these approaches helps for faster test pattern generation, and faster redundacy identification.

To apply the same approach for ***Robust Tests***, there are some differnces:

- Rather than justifying the input only, we have to justify the stable values from the primary inputs also.

For robust tests, there will be ***total Four Tasks***.

Similarities:

- The technique to apply both FPTPG and APTPG is same.
- Path sensitization is same as by applying bit levels.

Another approach is **Robust and Non Robust Path delay Fault Simulation by Parallel Processing of Patterns [82].**

- This approach is used every time at the time of calucating the paths, which have, delay faults.
- This is useful, because there are a large number of paths where delay can occur.

For Robust as well as Non-Robust tests, six valued simulation and Four Valued Simulation is used.

These six values represented by vector pair (fv, pds)

First vector consist of the final value of signal k under the two-pattern test (V_1, V_2).

F v (k) ε {0,1}.

Second factor path detectability status pds (k) ε {s, p, -}

s for if transition values remains stable.

p for any path is present, which is robustly detectable.

- Neither it is stable nor there is robustly detectable path.

For robust test six-valued simulation is used, but for nonrobust test, there will be only one path, which will be faulty in circuit. So that's why s and - reduced to p'.

So

$$0s * 0- \rightarrow 0p'$$

$$1s * 1- \rightarrow 1p'$$

∧	0s	0p	0-	1s	1p	1-
0s	0s	0s	0s	0s	0s	0s
0p	0s	0-	0-	0p	0p	0p
0-	0s	0-	0-	0-	0-	0-
1s	0s	0p	0-	1s	1p	1-
1p	0s	0p	0-	1p	1p	1p
1-	0s	0p	0-	1-	1p	1-

(a)

∧	0p	0p'	1p	1p'
0p	0p'	0p'	0p	0p
0p'	0p'	0p'	0p'	0p'
1p	0p	0p'	1p	1p
1p'	0p	0p'	1p	1p'

(b)

Fig 4.4 **Six-valued Simulation for Robust Tests and Four-valued Simulation for Non-Robust Tests**

Encoding Logic Values:

pds(k) = (s(k) , p(k))

TABLE I
ENCODING OF LOGIC VALUES. (a) SIX-VALUED LOGIC. (b) FOUR-VALUED LOGIC

Logic Value of Signal κ	$fv(\kappa)$	$s(\kappa)$	$p(\kappa)$
0s/1s	0/1	1	0
0p/1p	0/1	0	1
0-/1-	0/1	0	0

(a)

Logic Value of Signal κ	$fv(\kappa)$	$p(\kappa)$
0p/1p	0/1	1
$0\overline{p}/1\overline{p}$	0/1	0

(b)

TABLE 4.1 **Encoding of Logic Values (a) Six-valued Logic (b) Four-valued Logic**

In order to support parallel processing of pattern pairs during fault simulation, we use three L-Bit wide machine words in the case of robust fault simulation and two L-Bit wide machine words in the case of non-robust fault simulation for representing logic values for signal k, which usually corresponds to L pattern pairs.

$<V_1, V_2>_1$, $<V_1, V_2>_2$,.....$<V_1, V_2>_L$ denotes the first word consisting final value.

Vec [f v (k)] = (fv (k) $_1$, fv (k) $_2$ fv (k) $_3$ ..... fv (k)$_l$)

Similarly

Vec [p(k)] = (p(k) $_1$, p (k) $_2$, p (k) $_3$, ... p (k)$_l$) , denotes the p-word.

Similarly

Vec [s(k)] = (s(k) $_1$, s (k) $_2$, s (k) $_3$, ... s (k)$_l$) , denotes the s-word.

Choose L=5

The set {1p, 0s, 0-, 1s, 0p}

In this Vec [f v (k)] ={1,0,0,1,0}.

Vec [p (k)] = {1,0, 0,0,1}

Vec [s (k)] = {0,1, 0,1,0}

Parallel Pattern Path Delay Fault Simulation

Consider the AND gate with input signal k_1 and k_2 and output signal n.

For Robust Path Delay Faults

f v (n) = fv (k_1). fv (k_2)

s(n) = fv (k_1)' . s(k_1) + fv (k_2)'. s(k_2) + fv (k_1) . s(k_1) + fv (k_2). s(k_2)

p(n) = fv (k_1). fv (k_2) . [p(k_1) + p(k_2)] + fv (k_1)'. p(k_1) . fv (k_2). s(k_2) + fv (k_2)'. p(k_2) . fv (k_1). s(k_1)

For Non-Robust Path Delay Faults

$f v(n) = fv(k_1).\ fv(k_2)$

$p(n) = fv(k_1)\ .\ p(k_2) + f v(k_2)\ .\ p(k_1)$

It summarizes the binary operations, which are necessary for determining the final value and the path detectability status by parallel processing of pattern pair in robust and nonrobust path delay fault simulation for all gate types considered. By applying law of associativity, these operations can easily be extended to gates with more than two inputs.

TABLE II
Binary operations for determining the final value in robust and nonrobust path delay fault simulation

Gate Type	Binary Operations Robust and Non-robust Path Delay Fault Simulation
AND	$vec[f v(n)] = vec[f v(k_1)]\ .\ vec[f v(k_2)]$
NAND	$vec[f v(n)] = \overline{vec[f v(k_1)]\ .\ vec[f v(k_2)]}$
OR	$vec[f v(n)] = vec[f v(k_1)] + vec[f v(k_2)]$
NOR	$vec[f v(n)] = \overline{vec[f v(k_1)] + vec[f v(k_2)]}$
BUF	$vec[f v(n)] = vec[f v(k)]$
INV	$vec[f v(n)] = \overline{vec[f v(k)]}$
XOR	$vec[f v(n)] = vec[f v(k_1)] \oplus vec[f v(k_2)]$
XNOR	$vec[f v(n)] = \overline{vec[f v(k_1)] \oplus vec[f v(k_2)]}$

TABLE II
Binary operations for determining the path detectability status in robust and nonrobust path delay fault simulation

Gate Type	Binary Operations	
	Robust Path Delay FS	**Non-robust Path Delay FS**
AND	$vec[s(z_1)] = \overline{vec[f \vee (k_1)]} \cdot vec[s(k_1)] + \overline{vec[f \vee (k_2)]} \cdot vec[s(k_2)] + vec[f \vee (k_1)] \cdot vec[s(k_2)]$	$Vec[p(z_1)] = vec[f \vee (k_1)] \cdot vec[p(k_2)] + vec[f \vee (k_2)] \cdot vec[p(k_1)]$
NAND	$vec[p(z_1)] = vec[f \vee (k_1)] \cdot vec[f \vee (k_2)] \cdot (vec[p(k_1)] \cdot vec[p(k_2)]) + \overline{vec[f \vee (k_1)]} \cdot vec[p(k_1)] \cdot vec[f \vee (k_2)] \cdot vec[s(k_2)] + \overline{vec[f \vee (k_2)]} \cdot vec[p(k_2)] \cdot vec[f \vee (k_1)] \cdot vec[s(k_1)]$	
OR	$vec[s(z_1)] = vec[f \vee (k_1)] \cdot vec[s(k_1)] + vec[f \vee (k_2)] \cdot vec[s(k_2)] + \overline{vec[f \vee (k_1)]} \cdot vec[s(k_2)] + \overline{vec[f \vee (k_2)]} \cdot vec[s(k_1)]$	$vec[p(z_1)] = \overline{vec[f \vee (k_1)]} \cdot vec[p(k_2)] + \overline{vec[f \vee (k_2)]} \cdot vec[p(k_1)]$
NOR	$vec[p(z_1)] = \overline{vec[f \vee (k_1)]} \cdot \overline{vec[f \vee (k_2)]} \cdot (vec[p(k_1)] \cdot vec[p(k_2)]) + vec[f \vee (k_1)] \cdot vec[p(k_1)] \cdot \overline{vec[f \vee (k_2)]} \cdot vec[s(k_2)] + vec[f \vee (k_2)] \cdot vec[p(k_2)] \cdot \overline{vec[f \vee (k_1)]} \cdot vec[s(k_1)]$	
BUF	$vec[s(z_1)] = vec[s(k)]$	$vec[p(z_1)] = vec[p(k)]$
INV	$vec[p(z_1)] = vec[p(k)]$	
XOR	$vec[s(z_1)] = vec[s(k_1)] \cdot vec[s(k_2)]$	$vec[p(z_1)] = vec[p(k_1)] + vec[p(k_2)]$
XNOR	$vec[p(z_1)] = vec[p(k_1)] \cdot vec[s(k_2)] + vec[p(k_2)] \cdot vec[s(k_1)]$	

Operations will be performed in table I, II, III in such a way such that this parallel pattern delay fault simulation should be minimized.

Path Delay Fault Detection

By using above formulas for parallel processing of Patterns, we can detect the both robust as well as non-robust tests, in which L pattern pairs are simulated.

As we know that for robust path delay fault simulation, the identification of detected path faults is a little bit more involved in non-robust path delay fault simulation.

We cannot simply extend our search for detected path faults over all inputs of g with logic value 0p or 1p. In this case we have to determine those gate inputs ki which have pds (k_i) = p and whose final value fv (k_i) is observable at the gate output n.

i.e. for Local Path Sensitivity

$$lps(n_{ki}) = p(k_i).n_{ki}^{fv} = 1;$$

$$n_{ki}^{fv} = fv(n(k_i)) + fv(n(k'_i))$$

Denotes the Boolean difference nki with regard to final value n and k_i.

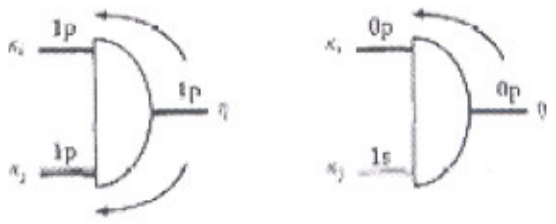


Fig 4.5 **Identification of Robustly detected Path Delay Faults**

$$\eta_{\kappa_i}^{fv} = (fv(\kappa_i) \wedge fv(\kappa_j)) \oplus (\overline{fv(\kappa_i)} \wedge fv(\kappa_j))$$
$$= (0 \wedge 1) \oplus (1 \wedge 1) = 1$$

And $pk_i = 1$, so lps (n_{ki})=1.

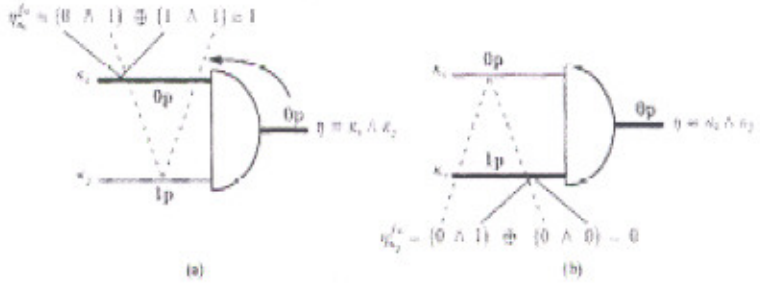


Fig 4.6 **Identification of Non-Robustly detected Path Delay Faults**

$$\eta_{\kappa_j}^{fv} = (0 \wedge 1) \oplus (1 \wedge 1) = 0$$

And pk_i=0, so lps (nk_i)=0

Path P1 and P2 are non robustly detectable, as there local path sensitivity is 1.

Path P3 is neither robustly detectable, nor non-robustly detectable, as their path sensitivity is 0.

In order to support parallel processing of patterns also during the path trace, there is vec [po(n)] as the path observability mask of signal n. initializing the path trace at a distinct PO σ is given by:

$$vec[po(\sigma)] = vec[p(\sigma)].$$

We recursively proceed in the mentioned depth-first manner from a gate output n to a gate input k by evaluating

$$vec[po(\kappa)] = vec[po(\eta)] \cdot vec[lps(\eta_\kappa)],$$

Where vec [lps (n_k)] is determined by parallel processing of patterns as expressed by:

$$vec[lps(n_k)] = vec[p(\kappa)] \cdot vec[n_\kappa^{fv}].$$

4.2 Sequential ATPG Algorithm for Robust and Non-Robust Tests

Phase 1:

Assign input/ouput values to the circuit.

Phase 2:

Do path sensitization by assigning the appropriate values at off-path inputs using 10-valued logic for robust tests and 3-valued logic for non-robust tests.

(If test generation is Robust and transition signal at on-path input changes from controlling to non-controlling values, then set the vector V_2 of off-path input to non-controlling value of corresponding gate associated.)

(If test generation is Robust and transition signal at on-path input changes from non-controlling to controlling values, then set the vector V_1 and V_2 of off-path input to non-controlling value of corresponding gate associated.)

(If test generation is Non-Robust then all off-path input signals for the path under test assume non-controlling values (0 when feeding into OR/NOR gate, and 1, into AND/NAND gate) in the steady-state following the application of second vector V_2. This condition is known as static sensitization of a path.)

Phase 3:

Justify all input signals using 10-valued logic for robust tests and 3-valued logic for non-robust tests. If a conflict occurs or redundancy occurs, do multiple path sensitization.

Following flow chart in Figure 4.7 describes that how CAD Tool works for Sequential Automatic Test Pattern Generation for Path Delay Faults.

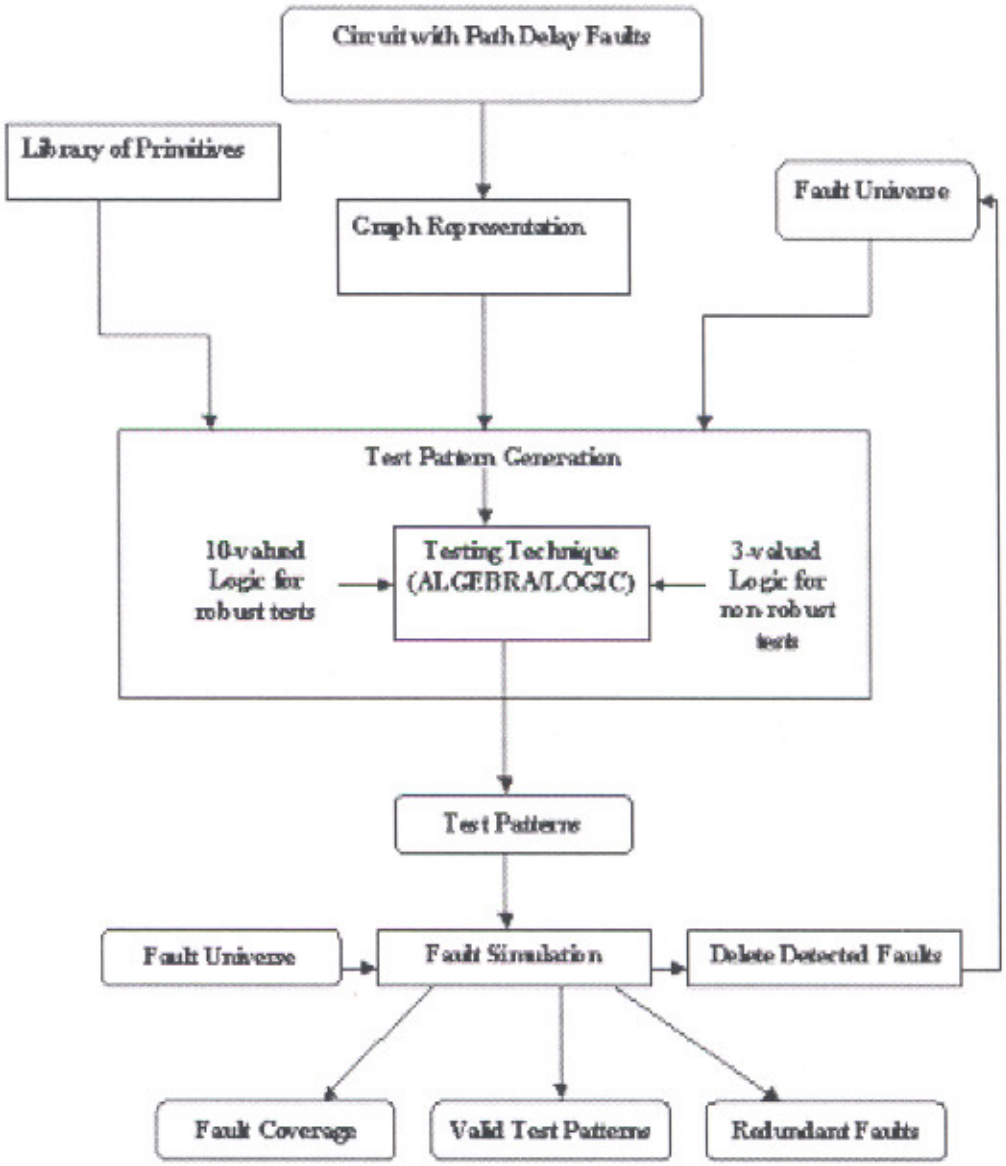


Fig. 4.7 **A CAD Tool for Sequential ATPG**

4.3 Proposed Parallel ATPG Algorithm for Robust and Non-Robust Tests

The mono-processor version of the algorithm spends most of its time in fault simulation, which detects path delay tests using 10-valued logic for robust tests and 3-valued logic for non-robust tests in sequential manner. So there was the need of parallelization. A

parallel algorithm using master-slave approach has been proposed and implemented on PVM (Parallel Virtual Machine) environment, which detects the path delay tests in parallel manner, by passing faults and paths in a circuit in parallel manner. By distributing the fault simulation task among the available processors we can exploit their computational power, this reduces CPU time as well as memory consumption and with result of increase in efficiency and performance and speed-up.

Master process executes thekernel of the overall algorithm on one processor, while fault simulation is performed by slave processes, which are distributed among all the other processors. Several techniques have been proposed in the literature to perform distributed fault simulation: fault partitioning, test vectors and test sequences partitioning ,paths partitioning we adopted a mixed approach, in which the most effective technique is used in each phase.

Phase 1:

The faults, paths, on-path I/O, off-path I/O, on-path vectors, off-path vectors to be simulated are partitioned among the slaves.

Phase 2:

Each slave will do the path sensitization and path justification related to corresponding data and different gates (AND, OR, NAND, NOR, NOT, EXOR, EXNOR) using 10-valued logic for robust tests and 3-valued logic for non-robust tests and result will be stored in a output file.

Phase 3:

If any conflict or redundancy occurs, then multiple path sensitization is done by slaves.

As a result, several fault simulation processes work in parallel in three phases.

Synchronization points are placed at the end of phase 1, at the end of each generation in phase 2, and at the end of phase 3. When master reaches these points, it waits until all the slave processes finish their work, thus guaranteeing the global correctness of the whole algorithm.

The Master process:

1. Does all the I/O operations, towards both the user and the file system: it writes the netlist, pathlist, and off-path values list, on-path values list, onpath vector list and off-path vector list.

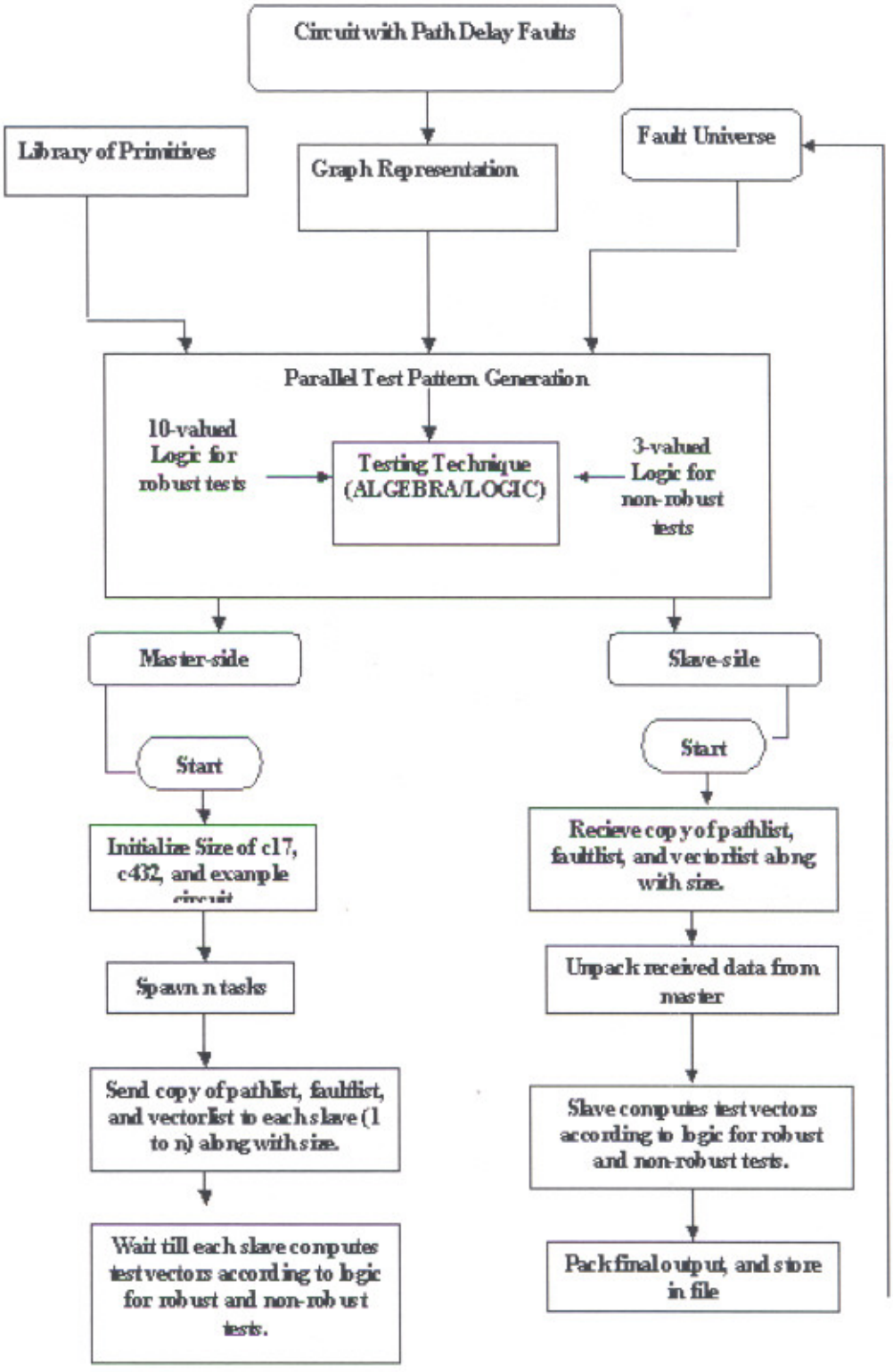
2. Initially spawns the slave processes on the available processors.

3. Distributes a copy of the internal format of the netlist, faultlist, pathlist, off-path values list, on-path values list, onpath vector list and off-path vector list to each slave process. Loops through the three phases: as soon as a sequence has to be fault-simulated, it sends an appropriate message to activate the slave processes, when the slavs finish their work, the master receives the results and updates the global data structures (i.e., the global fault list, pathlist, off-path values list, on-path values list, onpath vector list and off-path vector list).

The Slave processes:

The slave memories store a local copy of the pathlist, off-path values list, on-path values list, onpath vector list and off-path vector list. Each slave recieves data from master reads data and do the computations as specified in all three phase. When it finishes its task, it sends results to the master, and waits for a new job.

As faults and paths data is very large than number of processors, a very good load balancing is obtained in all three phases. Updating each slave local fault list with the detection information coming from other slaves is a critical problem in phase 1: it could save time by preventing repeated simulation of faults already detected by other slaves.


Circuit with Path Delay Faults
Library of Primitives
Graph Representation
Fault Universe
Parallel Test Pattern Generation
10-valued Logic for robust tests
Testing Technique (ALGEBRA/LOGIC)
3-valued Logic for non-robust tests
Master-side
Slave-side
Start
Start
Initialize Size of c17, c432, and example circuit
Recieve copy of pathlist, faultlist, and vectorlist along with size.
Spawn n tasks
Unpack received data from master
Send copy of pathlist, faultlist, and vectorlist to each slave (1 to n) along with size.
Slave computes test vectors according to logic for robust and non-robust tests.
Wait till each slave computes test vectors according to logic for robust and non-robust tests.
Pack final output, and store in file

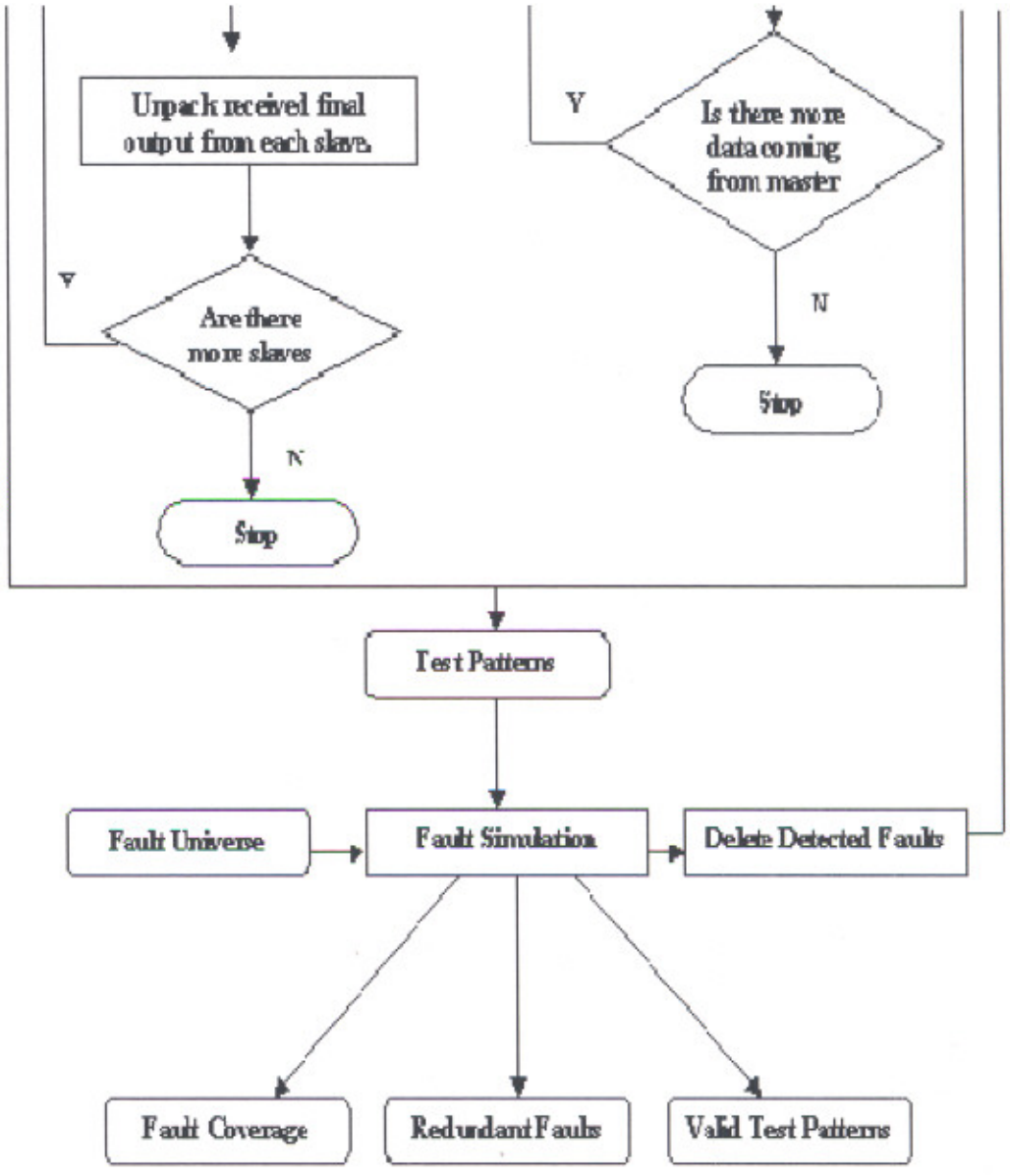


Fig. 4.8 **A CAD Tool for Proposed Parallel ATPG**

4.4 Comparision between Sequential and Parallel ATPG Algorithm

Type of Circuit	Sequential Average CPU Time (in sec)	Parallel Average CPU Time (in sec)
Example Circuit	24.57	0.22
C17	6.23	0.18
C432	53.24	0.59

TABLE 4.4 **Sequential vs. Parallel Average CPU Time**

Type of Circuit	1	2	3	4
Example Circuit	0.76	0.31	0.26	0.22
C17	0.25	0.21	0.21	0.18
C432	2.35	1.18	0.51	0.59

TABLE 4.5 **Average CPU Time vs. No. of Processors**

Type of Circuit	1	2	3	4
Example Circuit	1	2.45	3	3.5
C17	1	1.2	1.2	1.39
C432	1	2.78	3.45	3.99

TABLE 4.6 **Speed up vs. No. of Processors**

Type of Circuit	No. of Tested Paths	Sequential Average CPU Time (in sec)	Parallel Average CPU Time (in sec)
Example Circuit	28	24.57	0.22
C17	9	6.23	0.18
C432	1242	53.24	0.59

TABLE 4.7 **No. of Tested Paths vs. Average CPU Time**

4.5 Master-Slave Approach

4.5.1 Composition of PVM System

The first part is a daemon, called **pvmd3** and sometimes-abbreviated **pvmd that** reside on all the computers making up the virtual machine. Pvmd3 is designed so any user with a valid login can install it on a machine. When a user wishes to run a PVM application, he first creates a virtual machine by starting up PVM. The PVM application can then be started from a Unix prompt on any of the hosts. Multiple users can configure overlapping virtual machines and each user can execute many PVM applications simultaneously.

The second part of the system is a library of **PVM interface routines.** It contains a functionally complete repertoire of primitives that are needed for cooperation between tasks of an application. This library contains user-callable routines for message passing, spawning processes, coordinating tasks and modifying the virtual machine.

The PVM computing model is based on the notion that an application consists of several tasks. Each task is responsible for a part of the application's computational workload. Sometimes an application is parallelized along its functions; that is, each task performs a different function, for example, input, problem setup, solution, output, and display. This process is often called functional parallelism. A more common method of parallelizing an application is called data parallelism (the approach used by us). In this method all the tasks are the same, but each one only knows and solves a small part of the data. This is also referred to as the **SPMD (single-program multiple-data)** model of computing. PVM supports either or a mixture of these methods. Depending on their functions, tasks may execute in parallel and may need to synchronize

or exchange data, although this is not always the case. An exemplary diagram of the PVM computing model is shown Fig 4.9.

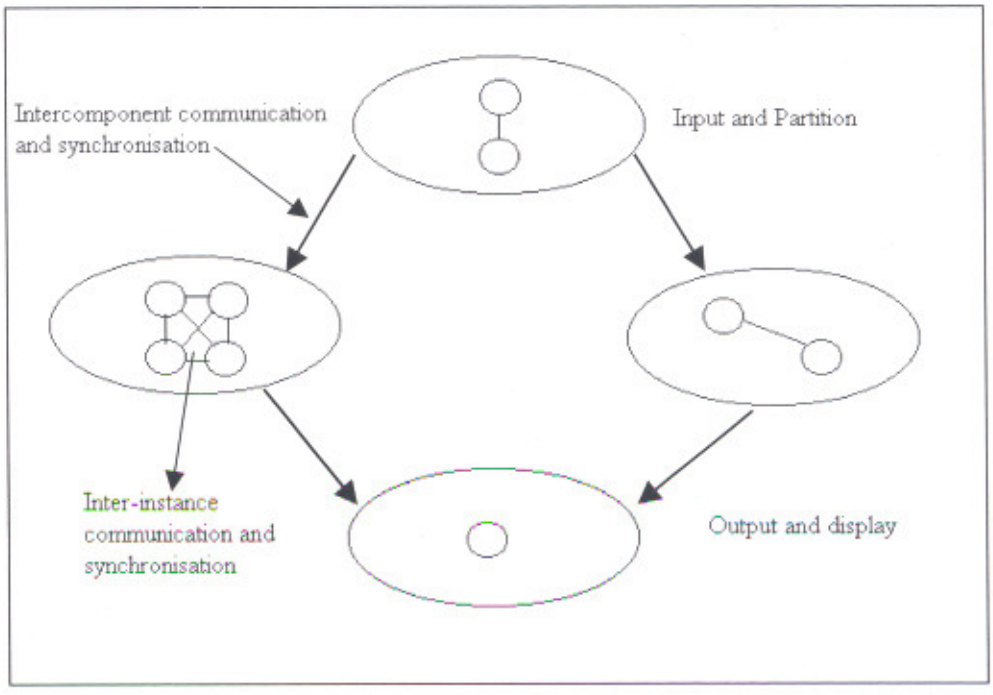


Fig 4.9 **PVM Computational Model**

The PVM system currently **supports C, C++, and Fortran languages**. This set of language interfaces have been included based on the observation that the predominant majority of target applications are written in C and Fortran, with an emerging trend in experimenting with object-based languages and methodologies.

The unit of parallelism is the task. The computation is divided into many tasks. All PVM tasks are identified by an **integer *task identifier* (TID)**. Messages are sent to and received from tids. Since tids must be unique across the entire virtual machine, they are supplied by the local pvmd and are not user chosen. Although PVM encodes information into each TID the user is expected to treat the tids as opaque integer identifiers. PVM contains several routines that return TID values so that the user application can identify other tasks in the system.

There are applications where it is natural to think of a *group of tasks*. And there are cases where a user would like to identify his tasks by the numbers **0 - (p - 1)**, where **p** is the

number of tasks. PVM includes the concept of user named groups. When a task joins a group, it is assigned a unique ``instance" number in that group. Instance numbers start at 0 and count up. In keeping with the PVM philosophy, the group functions are designed to be very general and transparent to the user. For example, any PVM task can join or leave any group at any time without having to inform any other task in the affected groups. Also, groups can overlap, and tasks can broadcast messages to groups of which they are not a member.

The general paradigm for application programming with PVM is as follows. A user writes one or more sequential programs in C, C++, or Fortran 77 that contain embedded calls to the PVM library. Each program corresponds to a task making up the application. These programs are compiled for each architecture in the host pool, and the resulting object files are placed at a location accessible from machines in the host pool. To execute an application, a user typically starts one copy of one task (usually the ``master" or ``initiating" task) by hand from a machine within the host pool. This process subsequently starts other PVM tasks, eventually resulting in a collection of active tasks that then compute locally and exchange messages with each other to solve the problem. Note that while the above is a typical scenario, as many tasks as appropriate may be started manually. As mentioned earlier, tasks interact through explicit message passing, identifying each other with a system-assigned, opaque TID.

4.6 Implementation Details and Experimental Results

The proposed algorithm has been implemented in a distributed computing environment consisting of workstations using Parallel Virtual Machine (PVM). A prototype of the proposed solution has been developed by implementing master-slave architecture in which the master processor first distributes the entire data to the available slave processors and asks each slave to compute the the test vectors using 10-valued logic for robust tests and 3-valued logic for nonrobust tests. The slave processors perform the required processing and return the computational result to the master processor.

We have successfully tested the prototype of the proposed solution for different ISCAS'85 benchmark circuits as well as example circuits.

4.6.1 Average CPU Time vs. No. of Processors

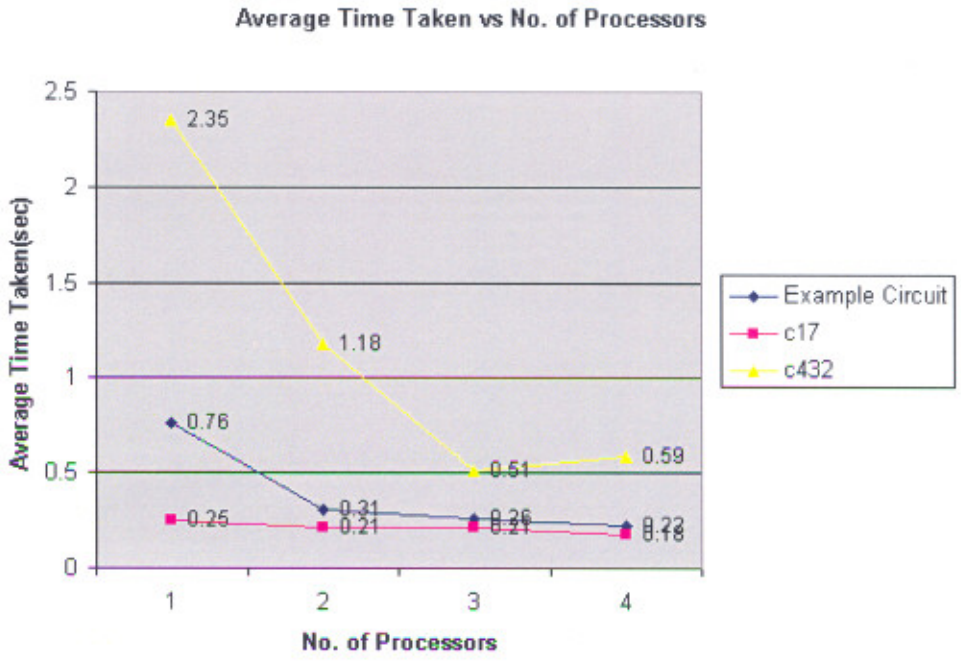


Fig 4.10(a) **Average CPU Time vs. No. of Processors**

4.6.2 Speed up vs. No. of Processors

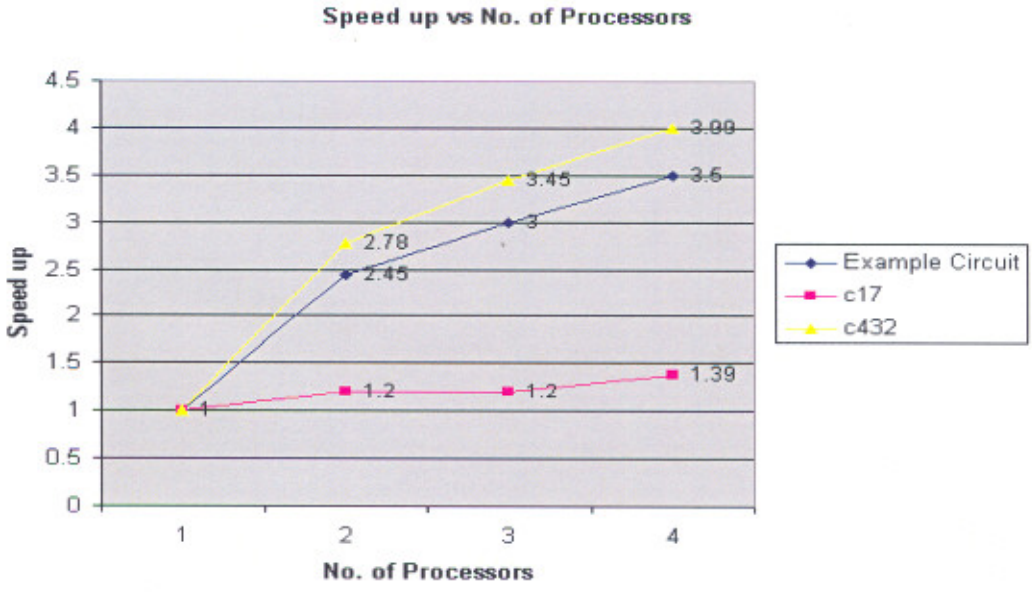


Fig 4.10(b) **Speed up vs. No. of Processors**

4.6.3 Parallel vs. Sequential Average CPU Time

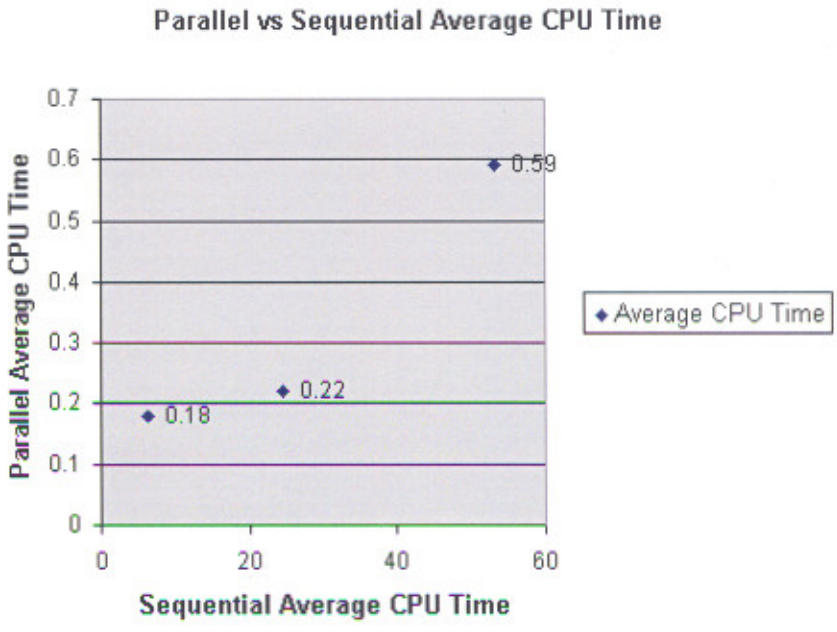


Fig 4.10(c) **Parallel vs. Sequential Average CPU Time**

4.6.4 Average CPU Time Taken vs. No. of Tested Paths

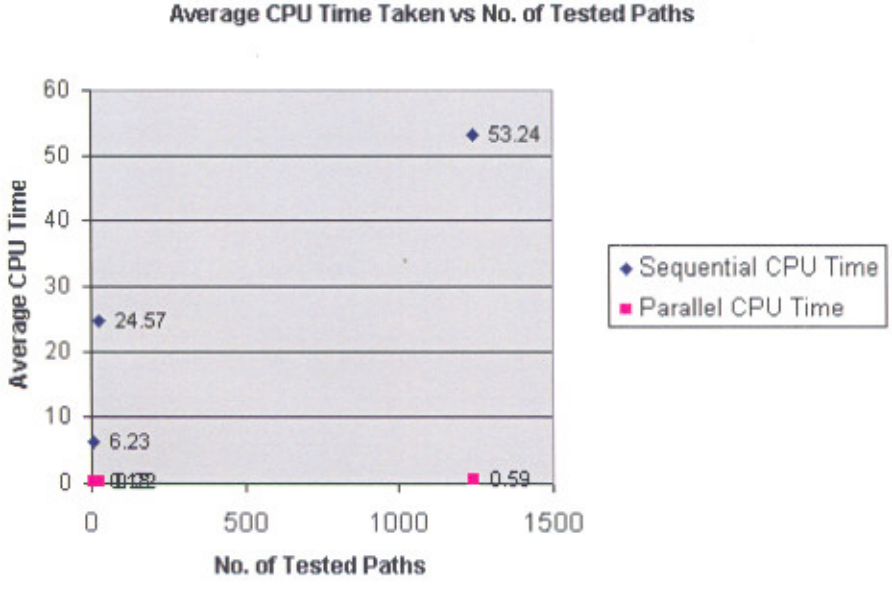


Fig 4.10(d) **Average CPU Time Taken vs. No. of Tested Paths**

The algorithm has shown good results as it has been tested by generating test patterns for some of the practical example circuit as well as ISCAS'85 benchmark circuits and example circuits. Memory consumption is also low. The experimental results reported in the thesis show the effectiveness of the proposed algorithm.

Chapter 5 Conclusions and Future Scope of Work

5.1 Conclusions

On the basis of:

- Study carried out on Automatic Test Pattern Generation (ATPG)
- Survey on Various Delay Fault Models
- Use of Hazard Algebra
- Parallelization of Test Generation for Path Delay Faults
- Implementation in PVM Environment

It can be concluded that

a) In VLSI circuits, to solve computationally intensive problem, a lot of research is going on. Especially on Automatic Test Pattern Generation (ATPG). The **testing** process detects the physical defects produced during the fabrication of a VLSI chip. Such a chip is tested by a sequence of input stimuli, known as test vectors, which check for possible defects in the chip by producing observable faulty responses at primary outputs. Test generation involves the generation of test vectors to detect failures in the VLSI chip.

b) 95% of fault coverage was done by stuck-at fault model. But to attain rest of fault coverage, we anlayzed delay fault models, which are logically correct but output come with delay. **Path Delay Fault Model** is gaining more attention to solve delay faults.

Path delay fault model have advantage of that it considers distributed failures, which can't be easily found by other delay fault models.

c) There are various approaches to solve path delay faults but **Hazard Algebra/ Logic System** has been found efficient.

Hazard Algebra/Logic System has been analyzed in detail in literature, we found 10-valued logic is best for robust test generation and 3-valued logic for non-robust test generation. Because, in view of the fact that maximizing the number of logic values, which can be uniquely determined by the implication procedure, is vital to performing the deterministic ATG process efficiently.

d) There are some limitations of sequential algorithm. CPU time and memory consumption is coming more. No effective parallel architecture or algorithm for this problem has as yet been found. At least two possibilities exist, however. First, a conventional uniprocessor algorithm can be parallelized by finding portions where it can be pipelined or directly executed in parallel. Second this problem can be reformulated and new parallel methods can be developed.

e) An efficient and practical parallel solution to the VLSI test generation problem has been proposed in this thesis. The proposed solution is based on the distributed test generation for path delay faults using 10-valued logic for robust tests and 3-valued logic for nonrobust tests. Computation in which the idle time of the processors is found very low as the computational work is almost evenly distributed among the available number of machines on the network. The parallel algorithm, so developed, has been implemented by using Parallel Virtual Machine (PVM) in a distributed computing environment mostly available for VLSI-CAD related activity. The algorithm has shown good results as it has been tested by generating test patterns for some of the practical example circuit as well as ISCAS'85 benchmark circuits. Memory consumption is also low. The experimental results reported in the thesis show the effectiveness of the proposed algorithm.

5.2 Future Scope of Work

Proposed approach has certain limitations that for example circuits, if we keep on adding the number of processors involved in the computation, there is an increase in the speedup but the speedup starts decreasing beyond a certain limit due to the large communication overheads. Thus the proposed approach recommends that number of processors to be employed should be based on the size of the problem assigned to each processor. Hence, the future scope of the work will be to implement the algorithm for generating test patterns for larger practical circuits in a more general-purpose parallel and/or distributed heterogeneous computing paradigm.

References

[1] Devadas Srinivas, Keutzer Kurt, "Synthesis and Optimization Procedures for Robustly Delay-Fault Testable Combinational Logic Circuits", 27th ACM/IEEE Design Automation Conference 1990, pp. 221-227.

[2] Girard P., Landrault C., Pravossoudovitch S., "A Novel Approach to Delay-Fault Diagnosis", 29th ACM/IEEE Design Automation Conference 1992, pp. 357-360.

[3] Tracy Larrabee, "Test Generation using Boolean Satisfiability", IEEE Transactions on Computer Aided Design, Vol. 11, No.1, pp. 4-15, January 1992.

[4] Devadas Srinivas, Keutzer Kurt, "Synthesis of Robust Delay-Fault Testable Circuits: Theory", IEEE Transactions on Computer-Aided Design, vol. 11, pp. 277-300, March 1992.

[5] Park Eun Sei, Mercer M. Ray, Williams Thomas W., "The Total Delay Fault Model and Statistical Delay Fault Coverage", IEEE Transactions on Computers, Vol. 41, No. 6, pp. 688-698, June 1992.

[6] Fink Franz, Fuchs Karl, Schulz Michael H., "Robust and Nonrobust Path Delay Fault Simulation by Parallel Processing of Patterns", IEEE Transactions on Computers, Vol. 41, No. 12, pp. 1527-1536,December 1992.

[7] Mahlstedt Udo, "DELTEST: Deterministic Test Generation for Gate Delay Faults", International Test Conference, October 1993, pp.972-980.

[8] Casar Ales, Meolic Robert, "Representation of Boolean Functions with ROBDDs", IEEE Region 8 Student Paper Contest, Paris-Evry, 1993.

[9] Fey Görschwin, Shi Junhao, Drechsler Rolf, BDD Circuit Optimization for Path Delay Fault-Testability, Proceedings of International Conference on Computer Aided Design, 1993, pp. 42-47.

[10] KraSniewski Andrzej, Wrofiski Leszek B., "Tests for Path Delay Faults vs. Tests for Gate Delay Faults: How Different They Are", Proceedings of the Conference on European Design Automation 1994, pp. 310-315.

[11] Henftling Manfred, Wittmann Hannes C., Antreich Kurt J., "Path Hashing to Accelerate Delay Fault Simulation", ACM/IEEE Design Automation Conference (DAC), June 1994, pages 522-526.

[12] Majhi Ananta K. Jacob James, Patnaik Lalit M., Agrawal Vishwani D., "An Efficient Automatic Test Generation System for Path Delay Faults in Combinational Circuits", Proceedings of the 8th International Conference on VLSI Design (VLSID '95), pp. 161-165.

[13] Raghunathan Anand, Ashar Pranav, Sharad Malik, "Test Generation for Cyclic Combinational Circuits", Proceedings of the 8th International Conference on VLSI Design (VLSID '95), pp. 104-109.

[14] Henftling Manfred, Wittmann Hannes, "Bit Parallel Test Pattern Generation for Path Delay Faults", Proceedings of the EDTC: The European Design and Test Conference (EDTC '95), pp. 521-525.

[15] Takahashi Hiroshi, Watanabe Takashi, and Takamatsu Yuzo, "Generation of Tenacious Tests for Small Gate Delay Faults in Combinational Circuits", Proceedings of the 4th Asian Test Symposium (ATS'95), pp. 332-338.

[16] Pomeranz Irith and Reddy Sudhakar M., "Functional Test Generation for Delay Faults in Combinational Circuits", Proceedings of the International Conference on Computer-Aided Design (ICCAD '95), pp. 687-694.

[17] Pierzynska Alicja, Slawomir Pilarski, "Quality Considerations in Delay Fault Testing", Proceedings of the European Design Automation Conference with EURO-VHDL '95 (EURODAC'95), pp. 196-201.

[18] Lam William K., "Synthesis of 100% Delay Fault Testable Circuits by Cube Partitioning", Hewlett – Packard Journal, February 1995, pp. 105-109.

[19] Cheng Kwang-Ting (Tim) and Krstic Angela, "Test Automation - Automatic Test Generation Technology and Its Applications", Volume 14, No. 7, pp. 909-916, July 1995.

[20] Heragu Keerthi, Patel Janak H., Agrawal Vishwani D., "Segment Delay Faults: A New Fault Model", Proceedings of the 14th IEEE VLSI Test Symposium (VTS '96), pp. 32-39.

[21] Sophie Crepaux-Mote, Mireille Jacomino, Red David, "An Algebraic Method for Delay Fault Testing", Proceedings of the 14th IEEE VLSI Test Symposium (VTS '96), pp. 308-315.

[22] Pomeranz Irith, Reddy Sudhakar M., "On the Number of Tests to detect all Path Delay Faults in Combinational Logic Circuits", IEEE Transactions on Computers, Vol. 45, No. 1, pp. 50- 62. January 1996.

[23] Hsu Yuan-Chieh, Gupta Sandeep K., "A Simulator for At-Speed Robust Testing of Path Delay Faults in Combinational Circuits". IEEE Transactions On Computers, Vol. 45, No. 11, pp. 1312-1318, November 1996.

[24] Krstic Angela, Cheng Kwang-Ting (Tim), Chen Hsi-Chuan, "Generation of High Quality Tests for Robustly Untestable Path Delay Faults", IEEE Transactions on Computers, Vol.45, No12, pp. 1379-1392, December 1996.

[25] Chen Chih-Ang, Gupta Sandeep K., "A Satisfiability-Based Test Generator for Path Delay Faults in Combinational Circuits", Proceedings of the 33rd Design Automation Conference 1996, pp.209-214.

[26] Luong G. M., Walker D. M. H., "Test Generation for Global Delay Faults", IEEE International Test Conference, Oct. 1996, pp. 433-442.

[27] Heragu Keerthi, Patel Janak H., Agrawal Vishwani D., "SIGMA: A Simulator for Segment Delay Faults", Proceedings of International Conference on Computer Aided Design (ICCAD) 1996, pp. 502-508.

[28] Heragu Keerthi, Patel Janak H., Agrawal Vishwani D., "Fast Identification of Untestable Delay Faults Using Implications", Proceeding of the International Conference on Computer Aided Design (ICCAD) 1996, pp. 642-647.

[29] Majhi Ananta K., Jacob James, Patnaik Lalit M., Agrawal Vishwani D., "On Test Coverage of Path Delay Faults", Proceedings of the 9th International Conference on VLSI Design: VLSI in Mobile Communication, January 1996,pp. 418-421.

[30] Pomeranz Irith, Reddy Sudhakar M., Patel Janak H., "On Double Transition Faults as a Delay Fault Model", Proceedings of the 6th Great Lakes Symposium on VLSI, 1996, pp. 282-287.

[31] Chen Liang-Chi, Gupta Sandeep K., Breuer Melvin A., "High Quality Robust Tests for Path Delay Faults", Proceedings of the 15th IEEE VLSI Test Symposium (VTS'97), pp. 88-93.

[32] Takahashi Hiroshi, Matsunaga Toshiyuki, Boateng Kwame Osei, Takamatsu Yuzo, "Method of Generating Tests for Marginal Delays and delay Faults in Combinational Circuits", Proceedings of the 5th Asian Test Symposium (ATS '97), pp. 320-325.

[33] Krstic Angela, Cheng Kwang-Ting (Tim), "Toward Testing Realistic Fault Behavior: Delay Fault Test", Draft 2 Tuesday, April 1, 1997, 1:36 pm Page 87/100 Chapter 5.

[34] Long Wangning, Li Zhongcheng, Yang Shiyuan, Min Yinghua, "Memory Efficient ATPG for Path Delay Faults", Proceedings of the 5th Asian Test Symposium (ATS '97), pp. 326-331.

[35] Kagaris Dimitrios, Tragoudas Spyros, Karayiannis Dimitrios, "Nonenumerative Path -Delay Fault Coverage Estimation with Optimal Algorithms", Proceedings of the 1997 International Conference on Computer Design (ICCD '97), pp. 366-371.

[36] Tekumalla Ramesh C., Menon Prem R," Test Generation for Primitive Path Delay Faults in Combinational Circuits", IEEE Transactions on Computer-Aided Design (ICCAD) 1997, pp. 636-641.

[37] Majhi Ananta K., Agrawal Vishwani D., "Tutorial: Delay Fault Models and Coverage", Proc. 11th International Conf. VLSI Design, pp. 364-369, January 1998.

[38] Hsu Yuan-Chieh, Gupta Sandeep K.," An Automatic Test Pattern Generator for At-Speed Robust Path Delay Testing", Asian Test Symposium 1998, pp. 88-95.

[39] Cheng-Wen Wu, Chih-Yuang Su, "A Probabilistic Model for Path Delay Faults", 7th Asian Test Symposium, December 02 - 04, 1998, pp. 70-75.

[40] Takahashi Hiroshi, Boateng Kwame Osei, Takamatsu Yuzo," Diagnosis of Single Gate Delay Faults in Combinational Circuits using Delay Fault Simulation", Asian Test Symposium. December 02 - 04, 1998, pp. 108-112.

[41] Bose Soumitra, Agrawal Prathima, Agrawal Vishwani D., "Deriving Logic Systems for Path Delay Test Generation", IEEE Transactions on Computers, Vol. 47, No. 8, pp. 829-846, August 1998.

[42] Pohl Frank, Walter Anheier, "Quality Determination for Gate Delay Fault Tests considering Three-State Elements", IEEE European Test Workshop, May 27-29, 1998.

[43] Krstic Angela, Cheng Kwang-Ting (Tim), Chakradhar Srimat T.,"Testing High Speed VLSI Devices using Slow Testers", Proceedings of the 17th IEEE VLSI Test Symposium, April 26 - 30, 1999, pp. 102-108.

[44] Pomeranz Irith and Reddy Sudhakar M., "A Flexible Path Selection Procedure for Path Delay Fault Testing", 17th IEEE VLSI Test Symposium April 26 - 30, 1999, pp. 108-116.

[45] Michael M., Tragoudas S.," ATPG tools for Delay Faults at the Functional Level ", Design, Automation and Test in Europe (DATE '99), March 09 - 12, 1999, pp. 631-636.

[46] Heragu Keerthi, Patel Janak H., Agrawal Vishwani D.," A Test Generator for Segment Delay Faults", Proceedings of the 12th International Conf. VLSI Design 1999, pp. 484-491.

[47] Ravikumar C.P. and Mittal Ajay, "Hierarchical Delay Fault Simulation", 12th International Conference on VLSI Design - January 1999, pp. 635-639.

[48] Krstic Angela, Liou Jing-Jia, Cheng Kwang-Ting (Tim), Wang Li-C, "Delay Testing Considering Power Supply Noise Effects", Proceedings IEEE International Test Conference 1999, pp. 181-190.

[49] Takahashi Hiroshi, Boateng Osei, Takamatsu Yuzo," A Method of Generating Tests with Linearity Property for Gate Delay Faults in Combinational Circuits", IEICE Transactions Inf. & Syst., Vol. ES2-D, No. 11, pp. 1466-1473, November 1999.

[50] Krstic Angela, Cheng Kwang-Ting (Tim),"Current Directions in Automatic Test Generation", IEEE Computers, Vol. 32, No. 11, 1999, pp. 58-65.

[51] Dastidar Jayabrata Ghosh, Touba Nur A.,"Adaptive Techniques for Improving Delay Fault Diagnosis", 17th IEEE VLSI Test Symposium, pp. 168-173, April 26 - 30, 1999.

[52] Pomeranz Irith, Reddy Sudhakar M.," Vector-Based Functional Fault Models for Delay Faults", Eighth Asian Test Symposium, November 16 - 18, 1999, pp. 41-46.

[53] Sosnowski Janusz, Wabia Tomasz, Bech Tomasz, "Path Delay Fault Testability Analysis", Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'00), pp. 338-347.

[54] Joonyoung Kim, Jesse Whittemore, Joao P. Marques-Silva, Karem Sakallah, "On Applying Incremental Satisfiability to Delay Fault Testing", Design, Automation and Test in Europe (DATE '00) March 27 - 30, 2000, pp. 380-384.

[55] Wiklund Kristian," Evaluation of Reduced Ordered Binary Decision Diagrams Using a Five-Valued Algebra", Technical Report 00-11, Department of Computer Engineering, Chalmers, Gothenburg, Sweden, 2000.

[56] Cheng Wu-Tung," Current Status and Future Trend on CAD Tools for VLSI Testing", Proceedings of the 9th Asian Test Symposium (ATS.00), pp. 10-11.

[57] Padmanaban S, Michael M., Tragoudas S.," Exact Path Delay Grading with Fundamental BDD Operations", ITC International Test Conference 2001 pp. 642-651.

[58] Chen Liang-Chi, Gupta Sandeep K., Breuer Melvin A.," A New Gate Delay Model for Simultaneous Switching and its Applications", Liang-Chi Chen, Sandeep K. Gupta, Melvin A. Breuer", Design Automation Conference (DAC), June 18-22, 2001, pp. 289-294.

[59] Rudy Garcla, "Rethink Fault Models for Sub-Micron IC Test" Test & Management World, October 2001,pp. 215473-215477.

[60] Krstic Angela, Jiang Yi-Min, Cheng Kwang-Ting (Tim), "Pattern Generation for Delay Testing and Dynamic Timing Analysis Considering Power-Supply Noise Effects", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 20, No. 3, pp. 416-425, March 2001.

[61] Joonhwan Yi, Hayes John P., "A Fault Model for Function and Delay Testing", Proceedings of the IEEE European Test Workshop 2001, pp. 27-35.

[62] Meyer Volker, Anheier Walter, Sticht Arne," Non-Robust Delay Test Pattern Generation based on Stuck-at TPG", 8th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2001), September 2-5, 2001.

[63] Krstic Angela, Liou Jing-Jia, Cheng Kwang-Ting (Tim), Wang Li-C, "On Theoretical and Practical Considerations of Path Selection For Delay Fault Testing", Proc. International Conference on Computer-Aided Design (ICCAD), November 2002.

[64] Padmanaban Saravanan, Tragoudas Spyros," Exact Grading of Multiple Path Delay Faults", Design, Automation and Test in Europe Conference and Exhibition (DATE'02) March 04 - 08, 2002, pp. 0084-0089.

[65] Meyer Volker, Sticht Arne, Weigl Andrew, Anheier Walter," How Robust Can Non-Robust Delay Tests Become", 7th European Test Workshop, Conference 26-29, May 2002.

[66] Liu Xiao, Hsiao Michael, Chakravarty S. Sreejit, Thadikaran Paul J," Novel ATPG Algorithms for Transition Faults", Proceedings of the Seventh IEEE European Test Workshop (ETW'02) May 26-29, 2002, pp. 47-53.

[67] Liou Jing-Jia, Wang LiC., Cheng Kwang Ting, "Enhancing Test Efficiency for Delay Fault Testing Using Multiple Clocked Schemes", 39th Design Automation Conference (DAC) 2002 June 10-14, 2002, pp. 371-374.

[68] Krstic Angela, Liou Jing-Jia, Cheng Kwang-Ting (Tim), Wang Li-C. ," On Structural vs. Functional Testing for Delay Faults", Symposium on Quality Electronic Design, March 2003.

[69] Majhi Ananta K., Gronthoud Guido, Valer Pop Eichenberger Stefan," Improving Diagnostic Resolution of Delay Faults using Path Delay Fault Model", Proceedings of the 21st IEEE VLSI Test Symposium (VTS.03).

[70] Ohtake Satoshi, Ohtani Kouhei, Fujiwara Hideo," A Method of Test Generation for Path Delay Faults Using Stuck-at Fault Test Generation Algorithms", Design, Automation and Test in Europe Conference and Exhibition (DATE'03) March 03 - 07, 2003, pp. 10310-10316.

[71] Xu Qiang, Nicolici Nicola," Delay Fault Testing of Core-Based Systems-on-a-Chip", Design, Automation and Test in Europe Conference and Exhibition (DATE'03) March 03 - 07, 2003, pp. 10744-10751.

[72] Gupta Puneet, Hsiao Michael S.," High Quality ATPG for Delay Defects", Proceedings of the IEEE International Test Conference, September 2003, pp. 584-591.

[73] Ganz Andreas, Tafertshofer Paul, Wittmann Hannes,"Parallel Computation of Delay Fault Probabilities",

[74] M. L. Bushnell and V. D. Agrawal, "Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits", Kluwer Academic Publishers, Boston, 2000, ISBN 0-306-47040-3.

[75] Krstic A., Cheng K. T.," Delay Fault Testing for VLSI Circuits ", Kluwer Academic Publishers, Boston, 1998, ISBN 0-7923-8295-1.

[76] Abramovici Miron, Breuer Melvin A., Friedman Arthur D., "Digital Systems Testing & Testable Design", Jaico Publishing House, Mumbai, 2002, ISBN 81-7224-891-1.

[77] Agrawal, P., Agrawal, V. D. and Bose, S.,"Deriving Logic Systems for Path Delay Test Generation", IEEE Transactions on Computers, 1998, Vol. 47, No. 8, pp .829-846.

[78] J.A. Brzozowski, Y. Iland, Algebras for Hazard Detection, 31st IEEE International Symposium on Multiple-Valued Logic, 22-24 May 2001, Warsaw, Poland, IEEE Computer Society, Los Alamitos, California, 2001, 3-12.

[79] Bawa, Seema, Sharma, G.K.,"A Parallel Transitive Closure Computation Algorithm for VLSI Test Generation", LNCS 2367, Applied Parallel Computing, Springer-Verlag Berlin Heidelberg, 2002, pp.243-252.

[80] Lin, C.J. and Reddy, S.M., "Delay fault testing in Logical Circuits", IEEE International Conference on Computer Aided Design, 1986, vol. 41,pp. 148-151.

[81] Schulz, M. H., Fuchs K. and Fink, F.,"Advance Automatic Test Pattern Generation Techniques for Path Delay Faults", 19th Fault Tolerance Communication System, 1989, pp. 44-51.

[82] Schulz, M. H., Fuchs K. and Fink, F.," Parallel Pattern Fault Simulation of Path Delay Faults," Proceedings of 26th Design Automation Conference, 1989, pp. 357-363.

[83] Schulz, M. H., Fuchs K. and Fink, F., DYNAMITE: "An Efficient Automatic Test Pattern Generation System For Path Delay Faults", Proceedings of IEEE Transactions on Computer Aided Design, 1989, pp.1323-1335.

Appendix A:

Parallelization of Path Delay Faults (Master-Slave Approach under PVM Environment)

I. Robust Test Generation using Ten-valued Logic

```
for(el=0;el<7;el++)
      {

      r1=strcmp(result[el],"robust");
      r2=strcmp(result[el],"nonrobust");


      if(r1==0)
      {
            for(el2=0;el2<7;el2++)
            {

            r3=strcmp(type_gate[el2],"AND");
            r4=strcmp(type_gate[el2],"OR");
            r5=strcmp(type_gate[el2],"NOT");
            r6=strcmp(type_gate[el2],"NAND");
            r7=strcmp(type_gate[el2],"NOR");
            r8=strcmp(type_gate[el2],"EXOR");
            r9=strcmp(type_gate[el2],"EXNOR");

            if(r3==0 || r6==0)
            {

            for(el1=0;el1<21;el1++)
            {
            r10=strcmp(result33[el1],"0s'");
            r11=strcmp(result33[el1],"1s'");

                  if(r10==0)
                  {
                  result55[el1]="1s";
            //    printf("%s\t%s\n",result55[el1],result33[el1]);
                  }

                  else if(r11==0)
                  {
                  result55[el1]="1x";
            //    printf("%s\t%s\n",result55[el1],result33[el1]);
                  }
                  result55androbust[el1]=result55[el1];

            //
      printf("%s\t%s\t%s\n",result55androbust[el1],result55[el1],result
33[el1]);


            }
      //    printf("%s\t%s\n",type_gate[el2],result[el]);
```

```
        }
```

II. Non-Robust Test Generation using Three-valued Logic

```
for(el=0;el<7;el++)
      {

      r1=strcmp(result[el],"robust");
      r2=strcmp(result[el],"nonrobust");


if(r2==0)
      {
            for(el2=0;el2<7;el2++)
            {

            r3=strcmp(type_gate[el2],"AND");
            r4=strcmp(type_gate[el2],"OR");
            r5=strcmp(type_gate[el2],"NOT");
            r6=strcmp(type_gate[el2],"NAND");
            r7=strcmp(type_gate[el2],"NOR");
            r8=strcmp(type_gate[el2],"EXOR");
            r9=strcmp(type_gate[el2],"EXNOR");

            if(r3==0 || r6==0)
            {

            for(el1=0;el1<21;el1++)
            {
            r10=strcmp(result33[el1],"0x");
            r11=strcmp(result33[el1],"1x");
                  if(r10==0)
                  {
                  result55[el1]="1x";
                  }

                  else if(r11==0)
                  {
                  result55[el1]="1x";
                  }
                  result55andnonrobust[el1]=result55[el1];
            //
      printf("%s\t%s\t%s\n",result55andnonrobust[el1],result55[el1],res
ult33[el1]);


            }
      //    printf("%s\t%s\n",type_gate[el2],result[el]);

            }


            else if(r4==0 || r7==0)
            {
```

```
            for(el1=0;el1<21;el1++)
            {
            r10=strcmp(result33[el1],"0x");
            r11=strcmp(result33[el1],"1x");
                  if(r10==0)
                  {
                  result55[el1]="0x";
                  }

                  else if(r11==0)
                  {
                  result55[el1]="0x";
                  }
                  result55ornonrobust[el1]=result55[el1];
            //
      printf("%s\t%s\t%s\n",result55ornonrobust[el1],result55[el1],resu
lt33[el1]);

            }
      //    printf("%s\t%s\n",type_gate[el2],result[el]);

            }
```

III. Output List of Test Vectors using 10-valued Logic for Robust Tests and 3-valued Logic for Non-Robust Tests

Onpath Test Vectors $<V_1,V_2>$	Offpath Test Vectors $<V_1,V_2>$	Final Output for Robust & Non-Robust Tests
0s	0s	0s
0s	0s	0s
0s	0s	1s'
0s	0s	0s
0s'	1s	0s'
0s'	0x	0s'
0s'	0s'	1s
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
0x	1x	0x
0x	0x	0x
0x	0x	1x
X	0x	0x
X	0x	0x
X	0x	1s'

X	0x	X
X	0x	1x
X	0x	1x
X	0x	X
X	0x	X
Xs'i	0x	0x
Xs'i	0x	Xs'
Xs'i	0x	1x
Xs'i	0x	Xs'i
0x	1x	0x
0x	0x	0x
0x	0x	1x
1s'	1x	1s'
1s'	0s	1s'
1s'	1s'	0s
X	X	X
X	X	X
X	X	X
X	X	X
X	X	X
X	X	X
X	X	X
X	X	X
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
0s	Xs'z	0s
0s	Xs'z	Xs'z
0s	Xs'z	1x
0s	Xs'z	Xs'z
1s	Xs'i	Xs'i
1s	Xs'i	1x
1s	Xs'i	X
1s	Xs'i	Xs'z
Xs'i	X	X
Xs'i	X	1s'
Xs'i	X	X
Xs'i	X	X
1x	1x	1x
1x	0x/1x	1x

1x	1x	0x
0x	1x	0x
0x	0x	0x
0x	0x	1x
1s'	1x	1s'
1s'	0s	1s'
1s'	1s'	0s
X	1x	X
X	1x	X
X	1x	1s'
X	1x	1x
X	1x	0x
X	1x	0x
X	1x	X
X	1x	X
0x	1x	0x
0x	0x	0x
0x	0x	1x
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
0s	1x	0s
0s	1x	1x
0s	1x	0x
0s	1x	1x
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
Xs'i	X	X
Xs'i	X	1s'
Xs'i	X	X
Xs'i	X	X
Xs'z	Xs'z	Xs'z
Xs'z	Xs'z	1s'
Xs'z	Xs'z	1x
Xs'z	Xs'z	Xs'z
0s'	1s	0s'
0s'	0x	0s'
0s'	0s'	1s
0x	1x	0x
0x	0x	0x
0x	0x	1x
1x	1x	1x

1x	0x/1x	1x
1x	1x	0x
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
0s'	1s	0s'
0s'	0x	0s'
0s'	0s'	1s
Xs'	1x	Xs'
Xs'	1x	1s'
Xs'	1x	0x
Xs'	1x	Xs'
0s	0s	0s
0s	0s	0s
0s	0s	1s'
0s	0s	0s
1s	0s	0s
1s	0s	1s
1s	0s	1s'
1s	0s	1s
Xs'	0s	0s
Xs'	0s	Xs'
Xs'	0s	1s'
Xs'	0s	Xs'
0x	1x	0x
0x	0x	0x
0x	0x	1x
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
Xs'	0s	0s
Xs'	0s	Xs'
Xs'	0s	1s'
Xs'	0s	Xs'
0s	0s	0s
0s	0s	0s
0s	0s	1s'
0s	0s	0s
1s	0s	0s
1s	0s	1s

1s	0s	1s'
1s	0s	1s
X	1s	X
X	1s	1x
X	1s	0s'
X	1s	X
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
0x	1x	0x
0x	0x	0x
0x	0x	1x
0x	1x	0x
0x	0x	0x
0x	0x	1x
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
X	1x	X
X	1x	X
X	1x	1s'
X	1x	1x
X	1x	0x
X	1x	0x
X	1x	X
X	1x	X
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
0x	1x	0x
0x	0x	0x
0x	0x	1x
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
0x	1x	0x
0x	0x	0x
0x	0x	1x
1x	1x	1x
1x	0x/1x	1x

1x	1x	0x
X	1x	X
X	1x	X
X	1x	1s'
X	1x	1x
X	1x	0x
X	1x	0x
X	1x	X
X	1x	X
0s	0s	0s
0s	0s	0s
0s	0s	1s'
0s	0s	0s
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
X	0x	0x
X	0x	0x
X	0x	1s'
X	0x	X
X	0x	1x
X	0x	1x
X	0x	X
X	0x	X
1s	Xs'i	Xs'i
1s	Xs'i	1x
1s	Xs'i	X
1s	Xs'i	Xs'z
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
X	Xs'z	Xs'z
X	Xs'z	1s'
X	Xs'z	1x
X	Xs'z	X
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
0x	1x	0x
0x	0x	0x
0x	0x	1x
X	1s'	Xs'z
X	1s'	1s'

X	1s'	0s
X	1s'	Xs'
0x	1x	0x
0x	0x	0x
0x	0x	1x
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
X	1x	X
X	1x	X
X	1x	1s'
X	1x	1x
X	1x	0x
X	1x	0x
X	1x	X
X	1x	X
X	X	X
X	X	X
X	X	X
X	X	X
X	X	X
X	X	X
X	X	X
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
0x	1x	0x
0x	0x	0x
0x	0x	1x
0x	1x	0x
0x	0x	0x
0x	0x	1x
0x	1x	0x
0x	0x	0x
0x	0x	1x
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
1x	1x	1x

1x	0x/1x	1x
1x	1x	0x
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
0s	0s	0s
0s	0s	0s
0s	0s	1s'
0s	0s	0s
1s	0s	0s
1s	0s	1s
1s	0s	1s'
1s	0s	1s
1x	1x	1x
1x	0x/1x	1x
1x	1x	0x
1s	1x	1x
1s	1x	1x
1s	1x	1x
1s	1x	0x
1s'	1x	1s'
1s'	0s	1s'
1s'	1s'	0s
0x	1x	0x
0x	0x	0x
0x	0x	1x
0x	1x	0x
0x	0x	0x
0x	0x	1x
0x	1x	0x
0x	0x	0x
0x	0x	1x
X	X	X
X	X	X
X	X	X
X	X	X
X	X	X
X	X	X
X	X	X
X	X	X

Papers Communicated

1. I have submitted paper on "Hazard Algebra for ATPG based on various Fault Models" to Journal of Universal Computer Science (JUCS), Springer Publications.
2. I have submitted paper on "A Parallel ATPG Algorithm for Path Delay Faults" to International Journal of Computational Methods (IJCM), World Scientific Publications.
3. I have submitted paper on "Delay Fault Testing : Past, Present and Future" to Journal of Electronic Testing : Theory and Applications (JETTA) , Kluwer Publications.
4. I have to be submit paper on "Use of Multi valued Algebra for Delay Fault Testing" at "18th International Conference on VLSI DESIGN " will hold on January 3-7, 2005, organized at Kolkata, India.