Delay Fault Testing for VLSI Circuits

DELAY FAULT TESTING FOR VLSI CIRCUITS

ANGELA KRSTIĆ
Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106

KWANG-TING CHENG
Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106

Kluwer Academic Publishers
Boston/Dordrecht/London

Contents

Preface

This is an example preface. This is an example preface. This is an example preface. This is an example preface.

This is a preface section

This is an example of a preface. This is an example preface. This is an example preface.

Angela Krstić

This book is dedicated to my
daughter.

Acknowledgments

text...

INTRODUCTION

Angela Krstić

1 TEST APPLICATION SCHEMES FOR VLSI CIRCUITS

Unlike stuck-at fault testing, delay testing is closely tied to the test application strategy. This means that before tests for delay faults are derived it is necessary to know how these tests will be applied to the circuit. The testing strategy depends on the type of the circuit (combinational, scan, non-scan or partial scan sequential circuit) as well as on the speed of the testing equipment. Ordinarily, testing delay defects requires that the test vectors be applied to the circuit at its intended operating speed. However, since high speed testers require huge investments, testers currently used in test facilities are several times slower than the new designs that need to be tested on them. Testing high speed designs on slower testers requires special test application and test generation strategies.

In this chapter, we focus on different test application schemes for combinational and sequential circuits. We describe techniques used for testing scan as well as non-scan designs. Also, wee address the issue of testing high speed designs using slow testers and describe some of the currently available solutions to this problem.

1.1 COMBINATIONAL CIRCUITS

To observe delay defects it is necessary to create and propagate transitions in the circuit. Creating transitions requires application of a vector pair, $V = \langle v_1, v_2 \rangle$. The first vector initializes the circuit while the second vector causes the desired transitions. The test application scheme for combinational circuits

is shown in Figure 1.1. In normal operation, only one clock is used to control the input and output latche (system clock) and its period is T_c. In testing mode, the input nd output latches are controlled by two different clocks: the input and output clock, respectively. The period of these clocks, T_s, is assumed t be larger than T_c. The input and output clocks are skewed by an amount equal to T_c. The first vector, v_1, is applied to the primary inputs at time t_0.

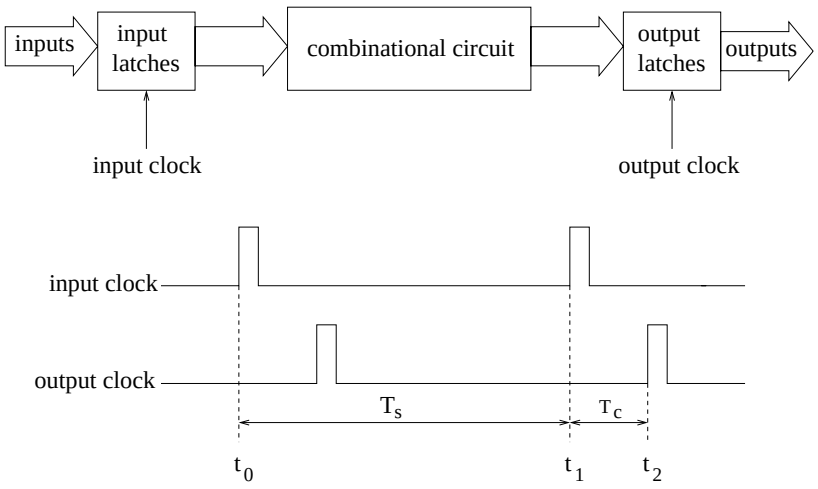
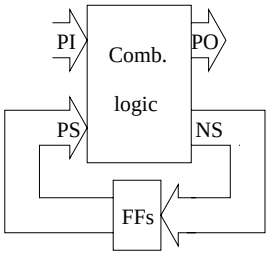


Figure 1.1. Testing scheme for combinational circuits.

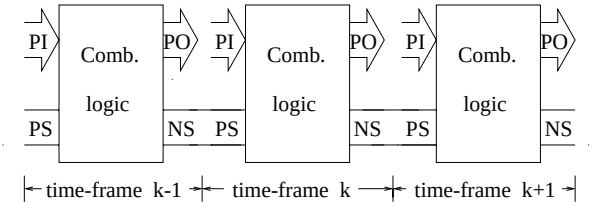
The second vector, v_2, is applied at time t_1. Time $T_s = t_1 - t_0$ is assumed to be sufficient for all values in the circuit to stabilize under the first vector. After the second vector is applied, the circuit is allowed to run for one clock cycle until time t_2, where $t_2 - t_1 = T_c$. At time t_2, the primary output values are observed and compared to a prestored response of a fault-free circuit to determine if there is a defect.

1.2 SEQUENTIAL CIRCUITS

Most of the delay testing research has concentrated on testing combinational circuits. Testing delay faults in sequential circuits is significantly more difficult than testing delay faults in combinational circuits. This is because application of an arbitrary vector pair is not possible to non-scan or standard scan sequential circuits. Figure 1.2(a) illustrates the model of a sequential circuit. Its operation can be represented using an iterative array of the combinational logic (shown in Figure 1.2(b)). Each copy of the combinational logic is called a **time-frame**. The present state (PS) values in time-frame k correspond to the next state (NS) values in time-frame $k-1$. In case of a sequential circuit, a vector pair, $V = \langle v_1, v_2 \rangle$, can be represented as pair $V = \langle i_1 + s_1, i_2 + s_2 \rangle$, where i_1, i_2 are the values of the primary input lines, s_1, s_2 are values of the present



(a)



(b)

Figure 1.2. Model for sequential circuits.

state lines and symbol "+" denotes concatenation of bit vectors. Therefore, vector i_1 is required to produce s_2 as the next state of the sequential machine.

There are several commonly used testing strategies for sequential circuits: *enhanced scan*, *functional justification* and *scan shifting* for standard scan, *slow-fast-slow strategy* and *at-speed strategy* for non-scan or partial can designs.

1.2.1 Enhanced scan testing

To solve this problem, Dervisoglu and Strong [17] propose using memory elements that can store two bits of state instead of just one. Such flip-flops are called **enhanced scan flip-flops**. The disadvantages of using enhanced scan flip-flops are high area overhead and long test application time.

1.2.2 Standard scan testing

Generating tests for delay faults for standard scan designs corresponds to a two time-frame sequential circuit test generation. In the first time frame, all primary inputs and present state lines are fully controllable. In the second time-frame, only the primary inputs are fully controllable. Testing schemes for standard scan have been proposed in literature [15, 73, 74, 75]. These techniques use **functional justification** (also called **broad-side test** [75]) or **scan shifting** [15] (also called **skewed-load test** [73, 74]) to obtain the second vector. In functional justification, the second vector represents the set of next

state values obtained after the application of the first vector. In scan shifting, the second vector is obtained by shifting the contents of the scan chain by one bit after the application of the first vector. Figure 1.3 illustrates the functional justification and scan shifting concepts.

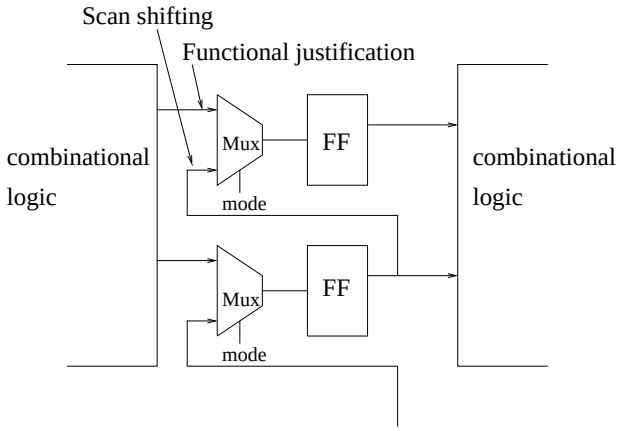


Figure 1.3. Standard scan design testing schemes.

Cheng *et al.* [15] propose a delay test generation algorithm for standard scan designs. It is modified from a PODEM-based combinational test generator. The modifications involve a two time-frame expansion of the combinational logic of the circuit, and the use of backtracking heuristics tailored to detecting delay faults. The present state values for the second vector are generated using functional justification or scan shifting. A fault that is redundant under scan shifting might be testable under functional justification and vice versa. On the average, the test generation complexity is lower when scan shifting rather than functional justification is used.

The order of flip-flops in the scan chain cannot affect the fault coverage when functional justification is used. However, when scan shifting is applied, the order of flip-flops in the scan chain affects the fault coverage. To find a good order of flip-flops in the scan chain, Cheng *et al.* [15] first run the test generation algorithm for standard scan designs using functional justification. If the fault is not detectable using functional justification, test generation in enhanced scan mode is tried. The test generator attempts to have as many *don't care* entries as possible in the present state lines in the second vector of the two vector sequence. Once the test pair, $\langle v_1, v_2 \rangle = \langle i_1 + s_1, i_2 + s_2 \rangle$, is generated, a set of constraints on the scan ordering is computed. These constraints, if satisfied, guarantee that s_2 can be obtained by scan shifting of v_1 in standard scan. In general, if the value of flip-flop FF_i is 0 (1) in s_1 and the value of flip-flop FF_j is 1(0) in s_2, then the constraint is that the flip-flop FF_i cannot be the immediate predecessor of flip-flop FF_j in the scan chain. If the circuit has n flip-flops, the constraints can be recorded using a quadratic matrix A of size n. Initially all entries in this matrix are set to zero. Given

a test vector pair for some target fault, if flip-flop FF_i is not allowed to be the immediate predecessor of flip-flop FF_j in the scan chain, then entry A_{ij} is increased by one. The matrix A is updated after each fault in the fault list is processed, until the fault list becomes empty. The final value A_{ij} represents the number of faults that will not be detected by scan shifting if flip-flop FF_i is the predecessor of flip-flop FF_j under the vector set used to construct the matrix A. The scan ordering is determined such that most of the constraints in matrix A are satisfied. Since a delay fault can have more than one test and only one of the tests is used to construct the matrix A, the fault might be detected even if the constraints in matrix A have not been completely satisfied.

Even with an efficient order of the flip-flops in the scan chain, a certain number of faults that can be detected under enhanced scan design, cannot be detected under standard scan design. To increase the fault coverage, Cheng *at al.* [15] propose **partial enhanced scan** design. In this design methodology, a subset of flip-flops is selected and made enhanced scan. The present state lines of enhanced scan flip-flops are fully controllable in both time-frames in test generation. Given a set of faults that are testable under enhanced scan but are redundant under functional justification or scan shifting scheme with a given ordering of flip-flops, the proposed heuristic attempts to minimize the number of flip-flops to be made enhanced scan in order to achieve a specified fault coverage.

1.2.3 Slow-fast-slow clock testing

Testing a fault in non-scan or partial scan sequential circuits requires a sequence of vectors. These vectors correspond to three different phases of the test generation process: fault initialization, fault activation and fault propagation. Fault initialization sets the signal values to the required values for fault activation. In the fault propagation phase, the fault effect is propagated from a next state line to some primary output. Fault initialization and fault propagation require a test sequence while the fault activation requires a vector pair. The existence of delay defects in the initialization and propagation phases can interfere with activation or the observation of the fault. A common solution is to apply a **slow-fast-slow clock** testing strategy. It assumes that the vectors for initialization and propagation of the fault effect are applied at a slow speed such that the circuit can be considered delay fault-free in these test phases. In the activation phase the first vector is applied under the slow clock while the second vector is applied at the rated speed. Figure 1.4 illustrates the slow-fast-slow testing strategy.

Testing methodologies for non-scan sequential designs using the slow-fast-slow scheme have been proposed in [18, 2, 11, 81]. The methodology proposed by Devadas [18] is based on extracting the complete or partial state transition graph. A known reset state is required. Due to the need for extracting the state transition graph, this methodology cannot handle large circuits. Agrawal *et al.* [2] propose inserting a logic block into the sequential circuit netlist for each fault such that testing a delay fault becomes equivalent to testing a certain

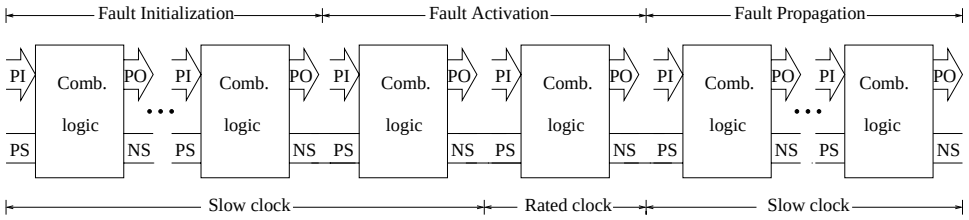


Figure 1.4. Slow-fast-slow testing strategy.

stuck-at fault. Chakraborty *et al.* [11] propose a delay test generator based on the iterative logic array model for sequential circuits. It considers two time-frames at a time.

Using a slow clock in fault initialization and fault propagation phases significantly simplifies the test generation for delay faults. However, the need for two clocks (slow and fast) complicates the test application. Testing delay faults in non-scan or partial scan design is further complicated by the fact that it is usually not practical to apply a single fault assumption for delay faults. Therefore, in slow-fast-slow clock testing scheme it could happen that at the end of the fault activation phase more than one flip-flop latches a faulty value. The test generator has to account for this possibility in the fault propagation phase. Chakraborty *et al.* [11] consider different initial conditions for the fault propagation phase.

1.2.4 At-speed testing

At-speed testing strategy assumes that the fault is initialized, activated and propagated under a fast clock. Therefore, delay faults are present in all three phases.

At-speed testing strategies for sequential circuits have been proposed in [64, 13]. Pomeranz *et al.* [64] assume that multiple delay faults can simultaneously be present in the circuit and develop a value system for testing delay faults under these conditions. In their experiments several fast clocks (up to 3) were embeded in sequences of slow clocks. The at-speed test methodology proposed by Cheng [13] uses a single fault assumption.

Some faults that are untestable under the slow-fast-slow clock testing scheme might become testable under the at-speed scheme and vice versa.

1.3 TESTING HIGH PERFORMANCE CIRCUITS USING SLOWER TESTERS

Testing a design at its intended operating speed requires high speed testers. High cost of fast testers makes it impossible for the testers to follow the designs in terms of speed increase. The problem of testing high performance circuits without high speed testers has been addressed by a number of researchers [84, 6,

1, 41, 27, 4, 22, 3]. The proposed strategies include tester pin multiplexing [1],

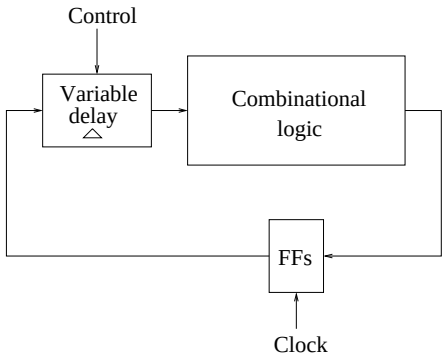


Figure 1.5. Inserting a controllable delay in the combinational logic.

built-in self-test [4], use of a high speed clock and shift registers [41], use of special test fixtures [6], reducing the supply voltage [84, 27], use of on-chip test circuitry for testing high bandwidth memories [22]. The technique proposed by Agrawal *et al.* [3] involve adding extra logic to the combinational logic such that the speed of the circuit in the testing mode becomes slower and comparable to the speed of testers. The amount of the added delay can be controlled by a test input signal. Figure 1.5 illustrates the concept. The extra logic for inserting

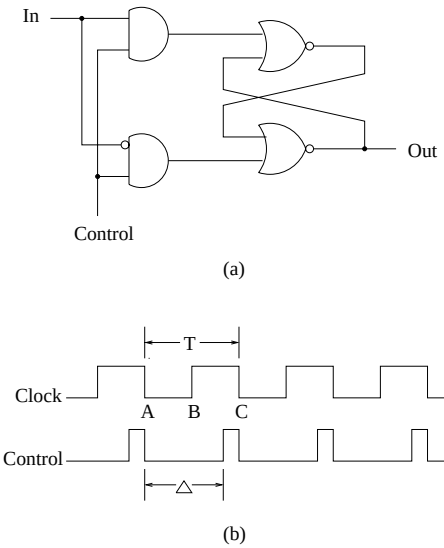


Figure 1.6. A controlled delay element and a waveform applied to the control input.

the variable delay should: (1) be controllable, (2) have a minimum normal

mode delay, (3) be testable and (4) use minimum logic. The following example describes one possible implementation of such logic.

EXAMPLE 1.1 Consider the circuit in Figure 1.6(a) [3]. When the control input is set to 1, the input signal propagates to the output. When the control input is 0, the output holds its value. During normal operation, the control input is held at 1. If single clock master-slave flip-flops are assumed and the clock waveform is as shown in Figure 1.6(b), the falling edge A is the time when the data is transferred from the master to slave flip-flop and the data stored in the slave flip-flop is applied to the combinational logic. The new data stored in the slave flip-flop will stay there until the next falling edge C. The rising edge B opens the master flip-flop to the input data. The time between the two falling edges (A and C) represents the clock period T. Figure 1.6(b) also shows the waveform for the control signal for the inserted logic. At the falling edge A of the clock, control signal drops to 0 and blocks the application of the data from the slave flip-flop to the combinational logic. After a delay Δ, the control signal rises to 1 and thus, allows the value from the slave flip-flop to be applied to the combinational logic. From Figure 1.6(b) it is clear that if T represents the clock period of the tester, then the clock period of the circuit can at most be $T_{rated} = T - \Delta$. In the test mode, delay Δ can be varied by changing the pulse width of the waveform.

The use of slow-fast-slow and at-speed testing schemes for testing high performance designs on slow testers has been discused by Krstić *et al.* [46]. They assume that the speed of the circuit is k (k is a positive integer) times higher than the speed of the tester and that an internal fast clock matching the speed of the circuit is available. If there is no fast clock available on the tester, the fast clock can be generated using frequency multiplier and the tester's clock.

1.3.1 Slow-fast-slow testing strategy on slow testers

The slow-fast-slow testing scheme can, under certain constraints, be used to test high performance circuits on low speed testers. In this scheme, the testable set of faults is affected by the presence or absence of latches on primary outputs. This is because to observe a fault, after activation, it has to be propagated to some primary output.

DEFINITION 1.1 Faults that in the activation time-frame can be propagated only to a primary output are called **PO-logic faults**.

DEFINITION 1.2 Faults that in the activation time-frame can be propagated to either a primary output or to a next state line and faults that in the activation time-frame can be propagated only to a next state line are called *NS-logic faults*.

Next, we consider the use of slow-fast-slow scheme on slow testers for testing non-scan, scan and partial scan designs.

Testing non-scan designs. The test application scheme for non-scan designs with latched PI/PO is shown in Figure 1.7(a). The primary inputs can be latched but it is not essential. The primary inputs are applied and the primary outputs are observed at the tester's speed. The tester's clock is also used in the slow phases (fault initialization and fault propagation). The tester's clock is assumed to be slow enough for the circuit to be fault-free in these phases. Fault activation is performed with a fast clock.

EXAMPLE 1.2 Consider the waveform in Figure 1.7(b). It illustrates the case when when the tester's clock is 2 times slower than the operating speed of the circuit under test, i.e., $k = 2$. Also, it is assumed that the test sequence for the target fault consists of two initialization vectors (v_1 and v_2), one activation vector (v_3) and two propagation vectors (v_4 and v_5). Initialization vectors, v_1 and v_2 are applied at times t_1 and t_2, respectively. After the application of the activation vector at time t_3, the values of the primary outputs and next states are latched at time t_4. Next, the propagation vectors v_4 and v_5 are applied at times t_5 and t_6, respectively. Finally, at time t_7, the primary outputs are observed.

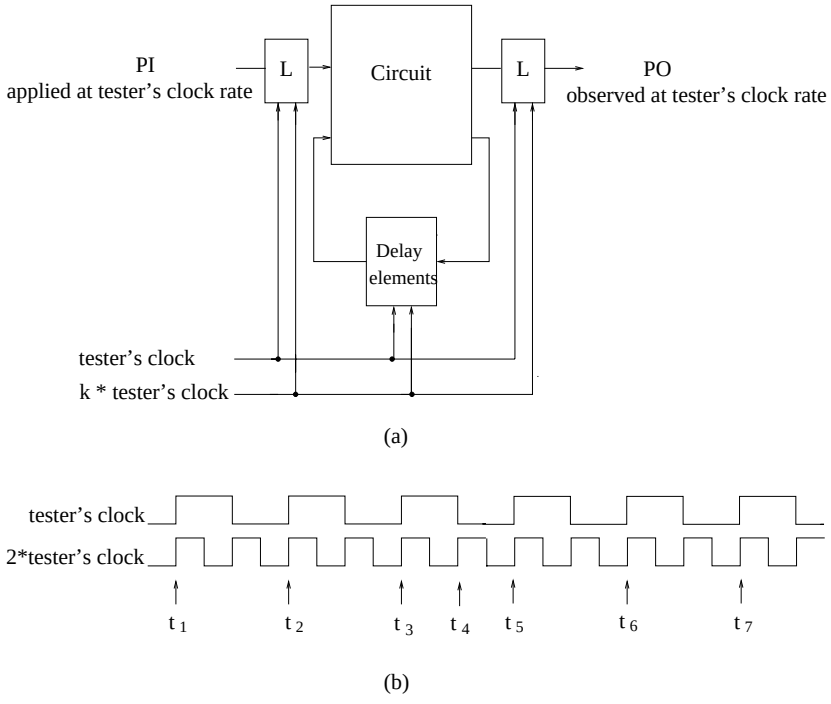


Figure 1.7. Non-scan designs with latched PI/PO.

Since the primary outputs can be latched at the end of the activation phase, this methodology can test both NS-logic and PO-logic faults.

When the primary outputs are not latched, PO-logic faults might not be testable on a slow tester using slow-fast-slow testing scheme. Only faults that are larger than a certain size can be tested. For example, PO-logic faults in the circuit in Figure 1.7(a) have to be larger than $t_5 - t_4$ to be testable.

Testing Scan Designs. Test application scheme that allows testing high speed scan designs on a low speed tester is illustrated in Figure 1.8(a). The tester's clock is used for applying the primary inputs, for the scan-in operation as well as for observation of the primary outputs and next state values, i.e., scan-out operation. The fast clock is used for latching the values into primary outputs and next states.

EXAMPLE 1.3 The waveform in Figure 1.8(b) illustrates the case when $k = 2$. First the present state values, v_1, are scanned into the registers. Next, vector $v_1 = (i_1 + s_1)$ is applied at time t_1. If standard scan is used, the state values s_2 of the second vector can be obtained through functional justification [15]. The second vector, $v_2 = (i_2 + s_2)$, is applied at time t_2. Time $t_2 - t_1$ is assumed to be sufficient for the signal values to settle to their final values after the application of vector v_1 and before application of v_2. Next, one fast clock cycle is applied and at time t_3, the values of the primary outputs and next states are latched. At time t_4, the primary outputs can be observed and the scan-out operation can start. Then, the same cycle repeats for the next test.

Since the primary outputs can be latched after the application of the fast clock, both PO-logic and NS-logic faults can be tested using this scheme. However, if the scan circuit in Figure 1.8(a) does not have latches at the primary outputs, from the waveform in Figure 1.8(b) we get that PO-logic faults have to be larger than $t_4 - t_3$ in order to be detectable.

Testing Partial Scan Designs. Testing scheme for partial scan designs represents a combination of the schemes described for non-scan and scan designs. The testing strategy depends on the target fault. For faults that can be tested through paths between the non-scan flip-flops, faults between non-scan flip-flops and POs and faults between PIs and non-scan flip-flops, the testing process is similar to the process described for faults in non-scan designs. It consists of initialization, activation and propagation phase. However, since some of the memory elements are scanned, the initialization and propagation phases might be shorter than in the non-scan case. For faults that can be tested through paths between the scanned flip-flops, faults between scanned flip-flops and POs and faults between PIs and scanned flip-flops, the testing strategy is the same as the one described for scan designs.

1.3.2 At-speed testing strategy on slow testers

Conventional at-speed testing strategies for sequential circuits [64, 13] assume that the inputs are applied and the outputs are observed at the circuit's rated

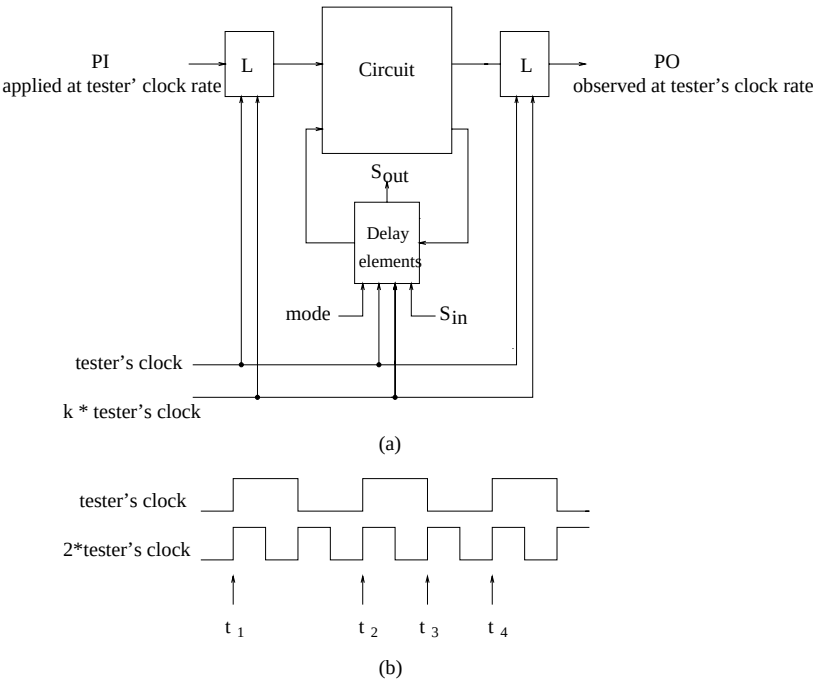


Figure 1.8. Scan designs with latched PI/PO.

speed. This is impossible to do on a low speed tester. Krstić *et al.* [46] propose an at-speed methodology that accommodates the slow speed of the tester. Figure 1.9 illustrates the proposed at-speed scheme. The inputs to the circuit are applied and the outputs are observed at the slow tester's rate. Using the internal fast clock makes the circuit go through k states between applying the inputs and observing the outputs. This is equivalent to saying that the same set of primary input values are applied for k clock cycles and that the primary outputs are observed only at the end of each k-th cycle. Since the circuit runs at-speed between each application of inputs and observation of outputs, delay faults are constantly present in the circuit.

EXAMPLE 1.4 Figure 1.10(a) illustrates the proposed at-speed testing scheme for $k = 3$. The same set of primary input values is applied for three fast clock cycles and the primary outputs are only observed after the third cycle. The delay elements are clocked with the fast clock and the circuit passes through three different states before the application of the next set of primary inputs.

Since the observation of the outputs is performed at the tester's speed, the existence or non-existence of latches at the primary outputs does not affect which faults can be tested using this at-speed scheme. This means that the

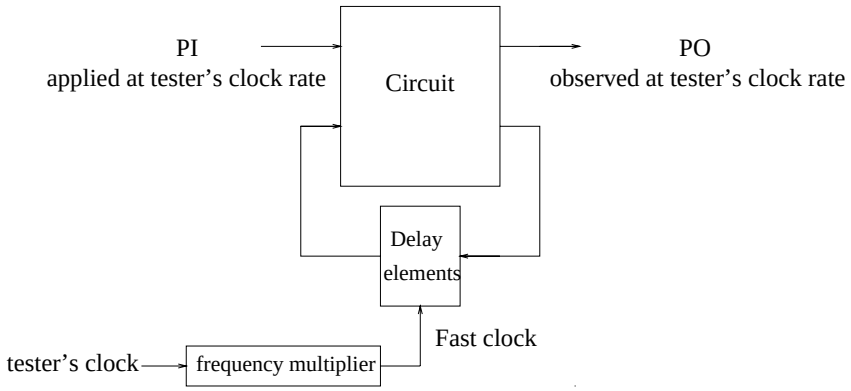


Figure 1.9. At-speed testing strategy for slow testers.

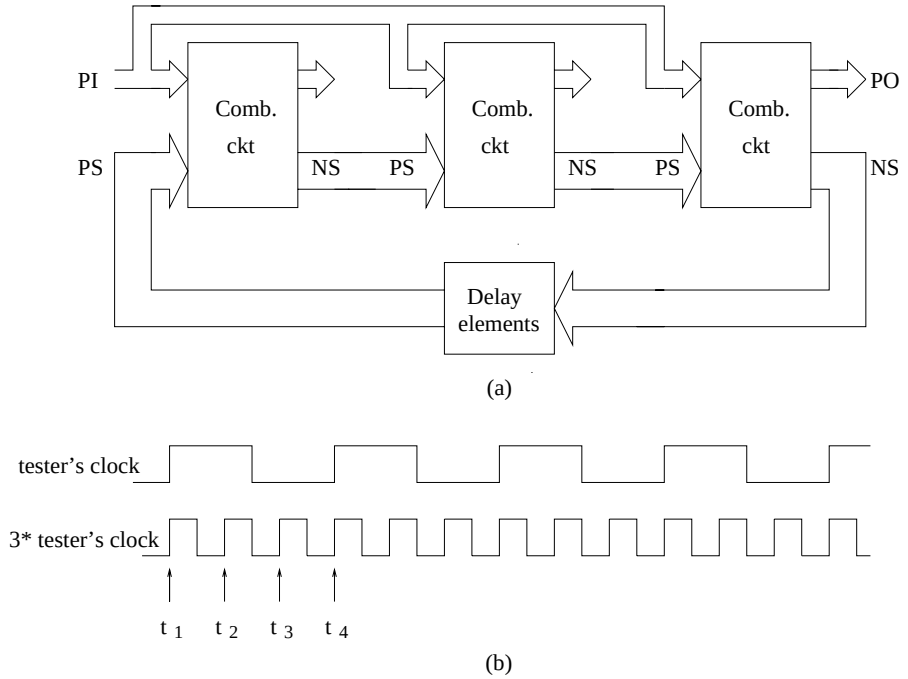


Figure 1.10. At-speed testing scheme for $k = 3$.

proposed at-speed scheme can be used to test PO-logic faults that cannot be tested using slow-fast-slow scheme.

Next, we consider the application of the at-speed testing scheme to non-scan, scan and partial scan designs.

Testing Non-Scan Designs. Let the design in Figure 1.10(a) be a non-scan design and consider the waveform shown in Figure 1.10(b). At time t_1, vector $v_1 = (i_1 + s_1)$ is applied to the circuit. Next, at time t_2, the primary input values stay unchanged but the state values have changed. Therefore, at time t_2, vector $v_2 = (i_1 + s_2)$ is applied to the circuit. Similarly, at time t_3, vector $v_3 = (i_1 + s_3)$ is applied. Finally, at time t_4, the primary outputs can be observed. A new vector, $v_4 = (i_2 + s_4)$, is applied to the circuit at time t_4 and the cycle repeats. In this scheme, if the test sequence contains n test vectors, where n is a positive integer, the circuit actually changes $k \times n$ states. For example, the circuit in Figure 1.10(a) must go through 3 or 6 or 9, ... states. Therefore, the test generation process for this at-speed testing strategy has to be different than the test generation process for at-speed schemes that assume fast testers.

Testing Scan Designs. Let the design in Figure 1.10(a) be a scan design and consider the waveform shown in Figure 1.10(b). The application of primary inputs, scan-in, scan-out and observation of the primary output values are performed at the tester's speed. However, between the scan-in and scan-out operations, the circuit is allowed to run with the fast clock and it goes through three states while the primary inputs are kept constant. At time t_1, the first set of state values, s_1, is assumed to be already scanned-in and i_1 is applied at the primary inputs. The state values for the second and third vector, $v_2 = (i_1 + s_2)$ and $v_3 = (i_1 + s_3)$, are obtained through functional justification and these vectors are applied at times t_2 and t_3, respectively. At time t_4, the values of the primary outputs are observed and the scan-out operation starts. The test sequence for scan designs contains k vectors.

Testing Partial Scan Designs. As with slow-fast-slow scheme, the at-speed testing strategy for partial scan designs can be described as a combination of testing strategies for scan and non-scan designs (depending on the target fault).

Since in this at-speed testing strategy the primary outputs are observed only after each k-th cycle, the signal observability is smaller than if the primary outputs are observed after each cycle. Also, since the primary inputs are kept unchanged for k clock cycles, the controllability of the signals is negatively affected as well. This can lead to lower fault coverage than if a high speed tester was available. Therefore, the described at-speed technique should not be used as a stand-alone technique. Instead, it can be combined with the slow-fast-slow testing strategy to obtain higher overall fault coverage. In the case when there are no latches on the primary outputs, the at-speed technique can be used to test PO-logic faults that would stay untestable under slow-fast-slow strategy. In addition, some NS-logic faults might also be untestable by a slow-fast-slow scheme but testable by the at-speed scheme. The proposed at-speed scheme can be used to detect them as well. Also, there exist faults that can be tested by both slow-fast-slow testing strategy and by the at-speed strategy. If these two strategies require that the circuit passes through a comparable

number of states when testing a given fault, the at-speed scheme would clearly be superior in terms of the testing time.

Summary

Test application strategy is integral part of delay test generation. This is especially true for testing sequential designs for which several different strategies exist. Enhanced vs. standard scan schemes show the trade-offs between the overhead in area and test application time versus fault coverage. Enhaced scan requires high area and test application time overhead but it also results in a higher fault coverage than standard scan techniques. In slow-fast-slow testing scheme, the assumption that the circuit is fault-free in the fault initialization and propagation phases greatly reduces the complexity of test generation but it complicates the test application process (when compared to the at-speed testing scheme).

An important factor in delay fault test generation is also the tester's speed. The speed of the testers usually lags behind the speed of the new designs. Therefore, developing new techniques that would allow testing high speed designs on slower testers is of great practical importance.

2 DELAY FAULT MODELS

The focus of this chapter is on the ways to model delay faults. Five delay fault models are considered: transition fault model, gate delay fault model, path delay fault model , segment delay fault model and line delay fault model. It is assumed that each gate can have an arbitrary fall (rise) delay from each input to the output pin. Also, the interconnects are assumed to have arbitrary rise (fall) delays. Since the gate pin-to-pin delays and the interconnect delays can be combined together, we will only talk about delays of gates. Transition , gate and line delay models are used for representing delay faults lumped at gates while the path and segment delay model address faults that are distributed over several gates. The advantages and disadvantages of each model are discussed.

2.1 TRANSITION FAULT MODEL

Transition fault model [13, 49, 76, 86] assumes that the delay fault affects only one gate in the circuit. There are two transition faults associated with each gate: a slow-to-rise fault and a slow-to-fall fault. It is assumed that in the fault-free circuit each gate has some nominal delay. Delay faults result in an increase or decrease of this delay. (Thoughout this book only delay faults caused by an increase of the delay will be considered.) Under transition fault model, the extra delay caused by the fault is assumed to be large enough to prevent the transition from reaching any primary output at the time of observation. In other words, the delay fault can be observed independent of whether the

transition propagates through a long or a short path to any primary output. Therefore, this model is also called *gross delay fault model* [60]. In addition to being a model for delay faults, transition fault model is also used as a logic model for transistor stuck-open faults in CMOS circuits [83]. CMOS transistor stuck-open faults can be treated as faults that either suppress or delay the occurrence of certain transitions. In practice, the extra delay caused by a stuck-open fault depends on the electrical characteristics of the defective component.

To detect a transition fault in a combinational circuit it is necessary to apply two input vectors, $V = \langle v_1, v_2 \rangle$. The first vector, v_1, initializes the circuit, while the second vector, v_2, activates the fault and propagates its effect to some primary output. During the application of the second vector the fault behaves as a stuck-at fault and vector v_2 can be found using stuck-at fault test generation tools. For example, for testing a slow-to-rise transition, the first pattern initializes the fault site to 0, and the second pattern is a test for stuck-at-0 fault at the fault site. A transition fault is considered detected if a transition occurs at the fault site and a sensitized path extends from the fault site to some primary output.

The fault equivalence rules for transition faults are more restrictive than those for stuck-at faults [86]. This is because, as mentioned above, testing a transition fault requires more than one vector. Only two rules can be applied for fault equivalence collapsing for transition faults: (1) if a gate has one input, then the input transition faults are equivalent to the output transition faults, and (2) if a gate has only one fanout, then the output transition faults are equivalent to the input transition faults on the fanout gate. As a result, the number of collapsed transition faults for a given circuit is larger than the number of collapsed stuck-at faults.

The main advantage of the transition fault model is that the number of faults in the circuit is linear in terms of the number of gates. Also, the stuck-at fault test generation and fault simulation tools can be easily modified for handling transition faults. On the other hand, the expectation that the delay fault is large enough for the effect to propagate through any path passing through the fault site might not be realistic because short paths may have a large slack. The assumption that the delay fault only affects one gate in the circuit might not be realistic, either. A delay defect can affect more than one gate and even though none of the individual delay faults is large enough to affect the performance of the circuit, several faults can together result in a performance degradation. For practical simplicity, the transition fault model is frequently used as a qualitative delay model and circuit delays are not considered in deriving tests.

Transition fault model for sequential circuits. The transition fault model described above cannot be used for sequential circuits if the clock is applied at the rated speed because it does not take into account the fault size. We now discuss a transition fault model [13] that is suitable for the at-speed test application scheme.

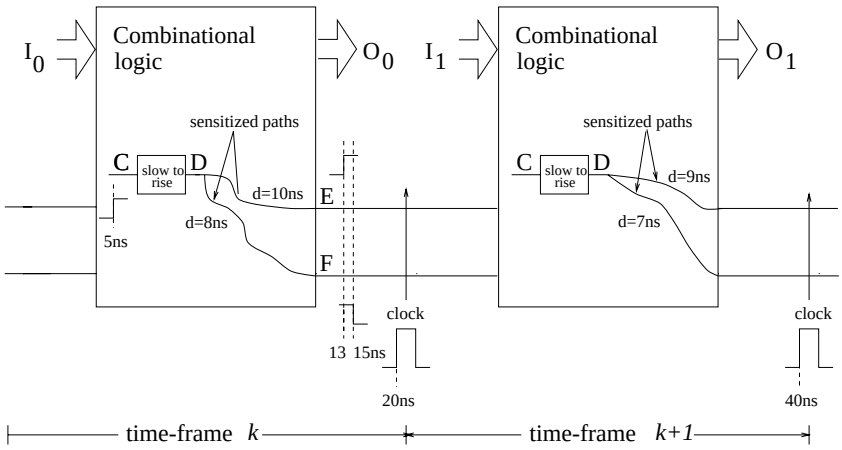


Figure 2.1. Faults of different size result in different next states.

The transition fault model for a sequential circuit [13] is characterized by the fault site, the fault type and the fault size. As before, the fault type is slow-to-rise or slow-to-fall transition. The fault size represents the amount of extra delay caused by the defect. In sequential circuits, different fault sizes will result in different faulty next states.

EXAMPLE 2.1 Consider the circuit in Figure 2.1. Figure 2.1 shows two time-frames of the given sequential circuit. It is assumed that input vectors are applied at the rated speed. The clock pulse for latching the next state is applied before the next input vector is applied. Suppose there is a slow-to-rise fault between the signal C and signal D. The clock interval is 20 nanoseconds (ns). The inputs are applied at 0ns (reference time) for time-frame k and a rising transition occurs at signal C at 5ns. There are two sensitized paths from C to the next state signals, E and F. The propagation delays of the transitions along these two paths are 10 and 8 ns, respectively. The transitions at E and F for the fault free-circuit and the times at which they occur are shown in the figure. If the fault size of the slow-to-rise fault at C is less than 5ns, the next state of the faulty circuit will be the same as that of the fault-free circuit, i.e., $(E, F) = (1, 0)$. If the fault size is greater than 5ns but less than 7ns, flip-flop E will catch the fault effect but flip-flop F will not (when the clock is applied at 20ns). The faulty next state will be $(E, F) = (0, 0)$. If the fault size is greater than 7ns, the faulty next state will be $(E, F) = (0, 1)$. The faulty next state, along with the next input vector, will produce a new logic value at each signal in time frame $k + 1$. Next, let the longest and shortest sensitized paths from C to any next state signal in time-frame $k + 1$ have delays of 9ns and 7ns, respectively. If the value at C in time-frame $k + 1$ is a logic 1 and if the fault size is in the range of (7ns, 26ns), the value at signal D in time frame $k + 1$ will be 1. The effects of the delayed transition will be stabilized at the next state signals in time-frame $k + 1$ before the following clock pulse is applied at 40ns.

On the other hand, if the new value at C is a logic 0, regardless of the fault size, the delayed transition will be completely suppressed and the value at D in time-frame $k+1$ will be 0. If the value at C in time-frame $k+1$ is a logic 1 and if the fault size is in the range of (26ns, 28ns), the effects of the delayed transition will be propagated to signal E and stabilized before the next clock pulse is applied. However, the next state signal F will not catch the fault effect in this case.

Clearly, it is not possible to guarantee the detection of a transition fault in a sequential circuit under the at-speed test application scheme without considering the size of the fault. Different fault sizes result in completely different circuit behaviors. However, the computation costs of dividing the fault sizes into hundreds of fine-grained ranges and simulating them are prohibitive. This problem can be solved by dividing the fault using units of clock cycles [13].

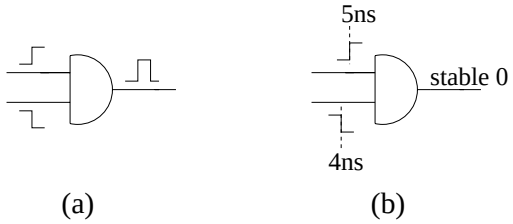


Figure 2.2. Advantage of considering circuit timing.

2.2 GATE DELAY FAULT MODEL

Gate delay fault model [9, 33, 34, 35, 69, 70] assumes that the delay fault is lumped at one gate in the circuit. However, unlike the transition model, gate delay fault model does not assume that the increased delay will affect the performance independent of the propagation path through the fault site. It is assumed that long paths through the fault site might cause performance degradation. Gate delay fault model is a quantitative model since it takes into account the circuit delays. The delays of the gates are represented as intervals. The gate delay fault model has the following characteristics [35]:

(1) The delay through a gate depends on the logic values applied to the gate.

(2) Multiple copies of a gate have different delays due to manufacturing variations.

(3) A gate has some inertia in responding to changes at its inputs. Transients of short duration at the gate inputs get filtered out from the response at the output.

Taking the timing into consideration when deriving tests for gate delay faults allows application of some tests that would otherwise not be considered.

EXAMPLE 2.2 Consider the AND gate shown in Figure 2.2. If no information about the delays in the circuit is available, it might be assumed that there is a static hazard at the output of the AND gate, as shown in Figure 2.2(a). This static hazard might prevent the propagation of some target fault elsewhere in the circuit. However, if the information about delays is given and the arrival times of the transitions at the inputs to the AND gate are as shown in Figure 2.2(b), the output will clearly have a stable 0 value, which may be favorable for the propagation of the target fault effect.

To determine the ability of a test to detect a gate delay defect it is necessary to specify the delay size of the fault. Methods for computing the smallest delay fault size (detection threshold) guaranteed to be detected by some test have been reported in the literature [33, 34, 69, 70].

The limitations of the gate delay fault model are similar to those for the transition fault model. Namely, because of the single gate delay fault assumption a test may fail to detect delay faults that are result of the sum of several small delay defects. The main advantage of this model is that the number of faults is linear in the number of gates in the circuit.

2.3 PATH DELAY FAULT MODEL

Under **path delay fault model** [79] a combinational circuit is considered faulty if the delay of any of its paths exceeds a specified limit. A path is defined as an ordered set of gates $\{g_0, g_1, \ldots g_n\}$, where g_0 and g_n are a primary input and primary output, respectively. Also, gate g_i is an input to gate g_{i+1} $(0 \leq i \leq n-1)$. A delay defect on a path can be observed by propagating a transition through the path. Therefore, a path delay fault specification consists of a physical path and a transition that will be applied at the beginning of the path. The delay or length of the path represents the sum of the delays of the gates and interconnections on that path.

Tests for the path delay fault model can detect small distributed delay defects caused by statistical process variations. A major limitation of this fault model is that the number of paths in the circuit can be very large (possibly exponential in the number of gates). For this reason testing all path delay faults in the circuit is not practical. Two strategies are commonly used for selecting the set of path delay faults for testing. One is to select a minimal set of paths such that for each signal s in the circuit the longest path containing s is selected for testing [50, 55, 54]. The other is to select all paths with expected delays greater than the specified threshold. The reason behind selecting the longest paths is that the delay defects on shorter paths might not be large enough to affect the circuit performance. Also, if the defects on short paths are large and could affect the performance, one expects that such defects would be detected by other tests (e.g., at-speed tests and gate delay tests) that precede the path delay fault testing. This strategy might work for circuits whose paths have very different delays so that there is a small percentage of long paths. However, often in performance optimized designs almost all paths have long delays and in these circuits not even all longest paths can be tested [61]. Therefore, even after

path delay fault testing, the temporal correctness of the circuit under test often cannot be guaranteed. The problem can be alleviated by developing techniques for resynthesizing the circuits such that the path count is reduced [44, 68].

EXAMPLE 2.3 Consider the path distribution for some circuit to be as shown in Figure 2.3 [44]. Horizontal axis represents the path length relative to the longest sensitizable path delay in the circuit. Vertical axis shows the number of paths whose delay is longer than the corresponding percentage of the critical path length. Let curve (a) represent the path distribution in the original circuit and curve (b) represent the path distribution in the resynthesized circuit such that the path count is reduced. Let us also assume that it is possible to test 5000 paths for delay defects. Selecting 5000 paths in the original circuit means that all paths longer than 75% of longest sensitizable path length can be checked for delay faults. On the other hand, selecting 5000 paths in the resynthesized circuit means that all paths longer than 59% of the critical path length can be checked. Therefore, the tests derived for the resynthesized circuit will be able to cover a larger portion of all possible path delay faults.

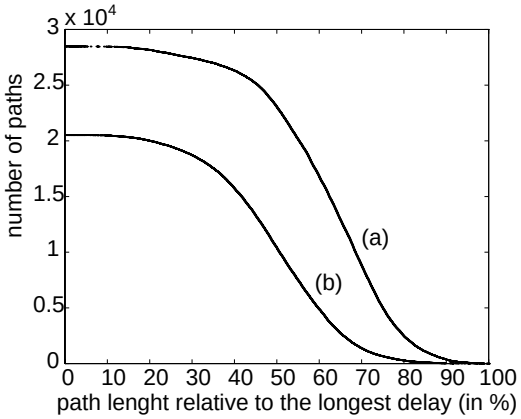


Figure 2.3. Path distributions for a circuit (a) before and (b) after resynthesis for path count reduction.

Additional problems with the use of the path delay fault are: (1) Tests that guarantee that the given path will not affect the performance of the circuit can be generated using reasonable resources only for a small set of paths in the circuit. For most circuits, there exists a large number of paths that can impact the performance of the circuit but these paths cannot be easily tested. Classification of path delay faults based on their testability characteristics is considered in Chapter 4. (2) Most path delay fault testing research has concentrated on testing combinational circuits. Extending these techniques to non-scan or partial scan designs is not straightforward.

2.4 SEGMENT DELAY FAULT MODEL

Segment delay fault model [29, 30] represents a trade-off between the transition delay fault model and path delay fault model. The assumption in this model is that the delay defect affects several gates in a local region of occurrence. Also, it is assumed that a segment delay fault is large enough to cause a delay fault on all paths that include the segment. The length of the segment, L, can be anywhere from 1 to L_{max}, where L_{max} represents the number of gates in the longest path in the circuit. The fault list consists of all segments of length L and all paths whose length is less than L. When $L = 1$, this model reduces to the transition fault model. When $L = L_{max}$, the segment delay fault model is equivalent to the path delay fault model. The idea of using the segment delay model is to combine the advantages of the transition and path delay fault models while avoiding their limitations. Since the number of segment delay faults for a given L can be much smaller than the number of all paths in the circuit, the explosion of the number of faults can be avoided. Also, the assumption that the fault is distributed over several segments is more realistic than the transition fault assumption about the lumped delay fault at one segment. In addition, in practice, many segments are testable while the entire paths containing those segments may not be testable.

The length of the segment can be decided on the basis of available statistics about manufacturing defects. All segments of a given length can be counted and identified using the method in [29].

2.5 LINE DELAY FAULT MODEL

Line delay fault model [54] tests a rising (falling) delay fault on a given signal (line) in the circuit. The fault is propagated through the longest *sensitizable* path passing through the given line. Similar to transition and gate delay fault models, line delay fault model assumes a single delay fault. Therefore, the number of faults equals twice the number of lines in the circuit. Sensitizing the longest path through the target line allows detecting the delay fault of the smallest size on the target line. In general, a test will cover several line delay faults. Therefore, this fault model can also detect some distributed delay defects on the propagation paths. However, since only one propagation path through each line is considered, this model can fail to detect some distributed defects [53].

Summary

Fault models represent an approximation of the effects that defects produce on the behavior of the circuit. An ideal model should provide a high confidence level that faulty circuits will be detected. The test generation process for such a fault model should allow handling of very large designs with reasonable amount of computing resources. Detecting timing defects requires models other than the well known stuck-at fault model. Several different delay fault models have been proposed in literature. Each of these models has its advantages and

disadvantages. The main characteristics of the delay fault models are shown in Table 2.1. Path delay fault model is usually considered to be closest to the ideal

Table 2.1. Comparison of different delay fault models.

Delay fault model	*number of faults w.r.t. number of gates*	*faults that can be tested*	*size of detectable faults*	*test generation*
transition	linear	lumped at gate	large	modified stuck-at ATPG
gate	linear	lumped at gate	larger than threshold	takes timing into account
path	exponential (worst case)	distributed along paths	small to large	hard
segment	linear to exponential	distributed along segments	small to large	depends on the segment length
line	linear	lumped at gate or distributed along certain paths	small to large	requires finding longest sensitizable path through line

model for delay defects. However, testing all path delay faults that can affect the performance of the circuit is impractical. Currently used path delay fault model is oversimplified for deep submicron devices for which the interconnect and cell delays are highly pattern dependent. Developing a more accurate fault model and selection of critical paths in new designs that are highly sensitive to process variations, circuit defects and coupling effects are important research problems for the future.

3 MOTIVATIONS FOR DELAY TESTING

This chapter discusses the motivations for subjecting a design to delay testing. The conclusions of several case studies carried out by various research groups are presented.

Research and experiments have shown that for high design quality requirements it is not sufficient to test a design only for stuck-at faults [8, 56, 82, 57]. There exist some faults that can only be detected if multiple test strategies are used. For example, an IBM experiment has shown that "randomly occuring gross delay defects can allow chips to pass full stuck-at fault testing at both wafer and module levels, but cause them to fail when operated at system speeds" [8]. In the experiment 60,000 dc good modules repesenting chips from the same IBM computer system have been subjected to delay testing. The modules were designed using CMOS standard cell/gate array and the experiment was performed for transition fault model. The test fault coverage was 75%. There were 97 dc good modules that have failed the delay testing, i.e., that had gross delay defects but were not detected with stuck-at fault testing. The histogram in Figure 3.1 [8] shows the distribution of fault sizes for 91 out of 97 modules that have failed the delay test. The clock cycle time was 120ns. In this experiment, 42.86% of the gross delay defects had size between 1 and 200 ns, while 65.93% of the delay defects was between 1 and 200ns. The experiment has clearly demonstrated the benefit of delay testing.

A possible cost effective strategy for delay testing would include:

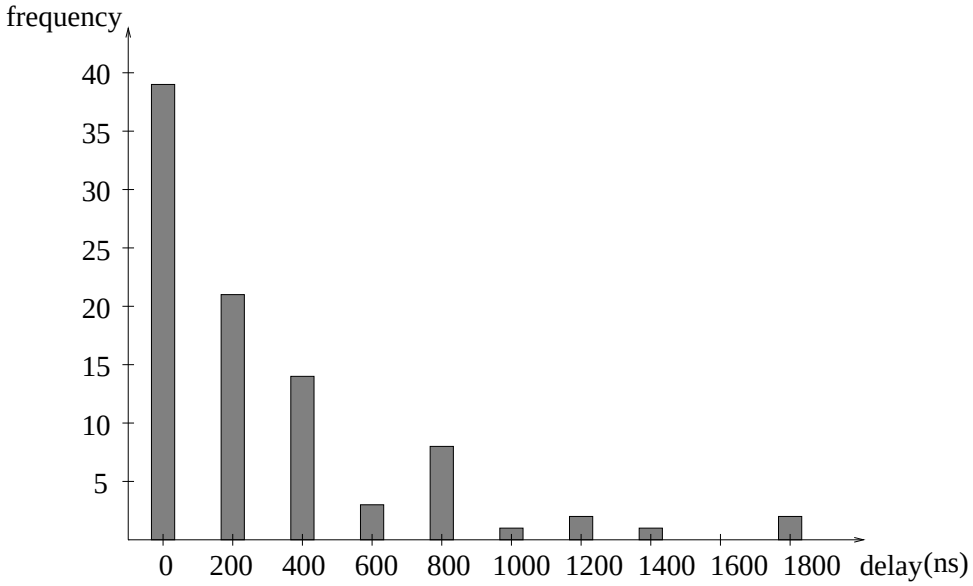


Figure 3.1. Fault size distribution in an IBM experiment.

- use of functional vectors that could be applied at-speed and should catch some delay defects. Functional vectors should be evaluated for transition fault coverage.
- application of tests for undetected transition faults
- application of tests for long path delay faults.

Several recent studies have investigated detection of timing defects in CMOS circuits by I_{DDQ} tests [56, 82, 57]. The experiment performed by Maxwell *et al.* has shown that I_{DDQ} testing can detect some delay defects. However, there exist some delay defects (distributed small delay defects caused by process vaiation, elevated series resistance in interconnects, elevated inteconnect capacitance) that can only be detected by delay testing. For example, the experiment described in [56] has tested a sample of 26,415 die. There were three types of tests applied: functional, scan and I_{DDQ}. Testing has identified 4,349 devices as faulty. Figure 3.2 [56] illustrates the distribution of the failing die in each test class. A total of 21 parts passed scan and low speed (2 MHz) functional tests but failed at-speed (20 and 32 MHz) functional tests. Only 10 out of 21 delay defective parts was detected by I_{DDQ} tests. Therefore, this study has suggested a testing strategy that combines high static stuck-at coverage, I_{DDQ} tests and delay tests.

The delay and overcurrent effects of resistive faults have been investigated by Vierhaus *et al.* [82]. Detailed simulations of resistive stuck-on, stuck-open and bridging faults have been performed for typical CMOS circuits. Figure 3.3

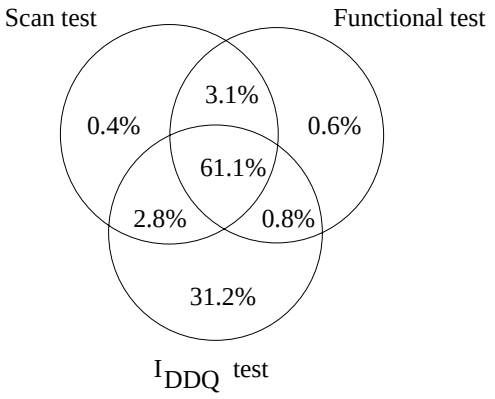


Figure 3.2. Distribution of failing die.

illustrates the simulated faults in a 2-input CMOS NAND gate. The summary

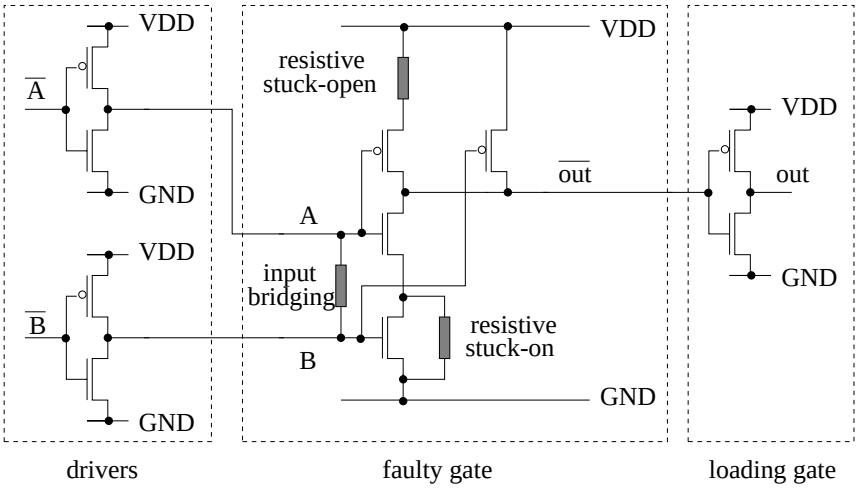


Figure 3.3. Considered resistive faults in 2-input CMOS AND gate.

of the delay and overcurrent effects for the 2-input AND gate in $1.5 - 2$ micron CMOS technology is shown in Figure 3.4. For example, for resistive transistor stuck-open faults for resistor values between 5 and 70 kω, the defects can be detected only by delay testing (overcurrent testing would not detect these faults).

Summary

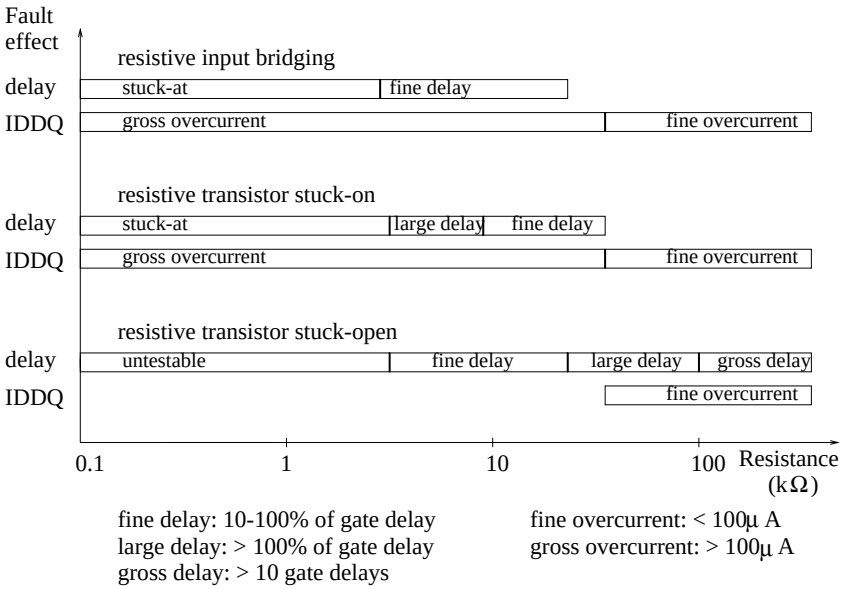


Figure 3.4. Delay and overcurrent effects for resistive faults in a 2-input CMOS AND gate.

4 PATH DELAY FAULT CLASSIFICATION

This chapter is devoted to a discussion on the classification of path delay faults. Paths are classified according to their testability characteristics. A given path delay fault can be tested by many different tests. Unlike a stuck-at fault for which all tests have the same quality (fault is certainly detected by the test), in path delay fault testing different tests for a given fault have different levels of quality (probability of detection). For example, some tests can guarantee detection of a fault while others can detect the fault only under restricted conditions. Not every path can be tested with a highest quality test. This is because higher quality path delay fault tests require more stringent conditions for path sensitization. To ensure the highest quality of path delay fault testing, each path delay should be tested under the most stringent sensitization criterion for which a test exists. Given various path sensitization criteria, paths are generally classified into several classes: robust, non-robust, functional sensitizable and functional unsensitizable.

Some path delay faults do not need to be tested to guarantee the performance of the circuit. This is because these path delay faults can never independently affect the performance. There are many different ways to partition the set of paths into the set that needs to be tested and the set that does not need to be tested.

This chapter describes two criteria for classifying path delay faults. The first one is based on the path sensitization and the second is based on whether

or not the given path needs to be tested to guarantee the performance of the circuit. Both single and multiple path delay faults are considered.

4.1 SENSITIZATION CRITERIA

Testing delay faults requires two vector patterns. Accordingly, path sensitization criteria are defined with respect to two vectors. This section addresses the sensitization of single path delay faults. Multiple path delay faults are addressed in Section 4.3.

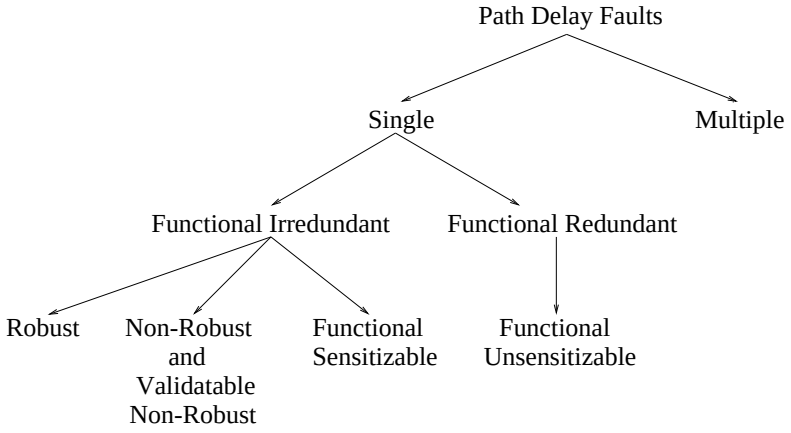


Figure 4.1. Path delay fault classification.

There exist several classes of path delay faults according to the sensitization criteria: robust, non-robust, validatable non-robust, functional sensitizable and functional unsensitizable faults. These classes have different testability characteristics based on the specific fault detection conditions. The robust, non-robust, validatable non-robust and functional sensitizable faults can affect the performance of the circuit and they are together called **functional irredundant faults**. Functional unsensitizable faults, also called **functional redundant faults**, can never independently determine the performance and they do not have to be tested. This section considers only functional irredundant faults, while functional unsensitizable faults will be addressed in Section 4.2. The path delay fault classification used in this book is illustrated in Figure 4.1. Note that most of the literature on path delay faults considers the non-robust set as a superset of the robust set of paths and the functional sensitizable set as a superet of non-robust testable set. However, in this book the set of robust testable, non-robust testable and functional sensitizable path delay faults are considered to be disjoint. The terminology that will be used in the rest of the book is given next.

Terminology. An input to a gate is said to have a **controlling value** (denoted as cv) if it determines the value of the gate output regardless of the values

on the other inputs to the gate. If the value on some input is a complement of the controlling value, the input is said to have a **non-controlling value** (denoted ncv). For example, in the case of an AND gate, the controlling value is 0 and the non-controlling value is 1, while for an OR gate the controlling value is 1 and the non-controlling value is 0. If the value of an input changes from the controlling to the non-controlling value, then the transition is denoted as cv→ncv. The ncv→cv transition changes the input from ncv to cv value. Each path delay fault is associated with a path and terms "path" and "path delay fault" will be used intechangebly. A signal is an **on-input** of path P if it is on P. A signal is an **off-input** of path P if it is an input to a gate in P but it is not an on-input. A path P is said to be **static sensitizable** if there is at least one input vector pair $V = \langle v_1, v_2 \rangle$ such that all off-inputs in P settle at respective non-controlling values under vector v_2. A path is said to be a **false path** if it can never propagate a transition to the primary output. The logic functions computed by the gates and their propagation delays preclude false paths from being sensitized [19]. Paths that are not false are called **true paths**.

Figure 4.2. Robust propagation through an AND gate.

4.1.1 Robust testable path delay faults

The **robust sensitization criterion** [52, 79] allows unconditional detection of a path delay fault. In other words, if there is a fault on the target path that is sensitizable under robust sensitization criterion the fault will be observed independent of the delays on signals outside the target path.

EXAMPLE 4.1 Consider an AND gate with two inputs, a and b, as shown in Figure 4.2. Let this gate be part of the target path P with input a as the on-input and input b as an off-input for P. Symbol S0 (S1) is used to denote a stable 0 (1) value on some signal under $V = \langle v_1, v_2 \rangle$, while values containing symbol X denote unspecified values. For example, if the value on some signal is X1, it means that the value of the signal is unspecified for vector v_1 and it is 1 for vector v_2. If input a is assigned a rising transition and the off-input b is also assigned a rising transition, the fault effect from a will propagate to the output of the AND gate, whether or not there is a fault on the off-input b. This is because the output of the AND gate is determined by the later of the two rising transitions. Similarly, if the off-input b has a stable non-controlling value and the on-input a has either a falling or a rising transition, the fault effect from the target path will always be observable at the output. These sensitization conditions are called the robust sensitization conditions.

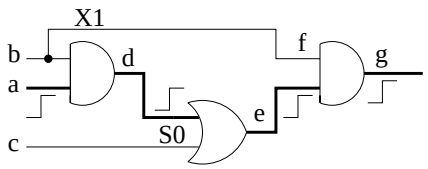


Figure 4.3. Robust path.

DEFINITION 4.1 Let f denote the on-input to gate g in the target path. Let h denote an off-input to gate g. The off-input h is called **robust off-input** if:

(1) there is a cv→ncv transition or stable non-controlling value on h when the on-input f has a cv→ncv transition, and

(2) there is a non-controlling value on h when the on-input f has a ncv→cv transition.

DEFINITION 4.2 Path delay fault for which there exists an input vector pair such that it activates the required transitions on the path and all off-inputs in the path are robust off-inputs is called **robust testable path delay fault**.

Therefore, a robust sensitizable path is static sensitizable.

EXAMPLE 4.2 Consider the circuit in Figure 4.3. Path {rising, *adeg*} is a robust testable path because there exists a vector pair such that all off-inputs for this path are robust off-inputs.

For each robust testable path there could be more than one input vector pair that robustly sensitizes the path. However, to test a robust path it is sufficient to apply one such vector pair since the fault is guaranteed to be detected. Robust tests are highest quality tests for path delay faults and should be applied whenever they exist. However, experience shows that for most circuits only a small portion of path delay faults can be tested under the robust condition [14, 21]. Paths that cannot be tested under robust sensitization criterion are called **robust untestable path delay faults**. All robust testable paths have to be selected for testing to guarantee the performance of the circuit.

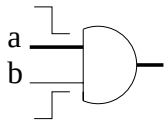


Figure 4.4. Non-robust sensitization criterion for AND gate.

4.1.2 Non-robust testable path delay faults

The **non-robust sensitization criterion** [52] is less stringent than the robust criterion. This is because the detection of a fault on a path that is sensitizable under a non-robust criterion depends on the delays on certain signals outside the target path.

EXAMPLE 4.3 Figure 4.4 illustrates the non-robust propagation criterion for an AND gate. If there is a falling transition on the on-input a and a rising transition on the off-input b, the transitions on the output of the AND gate depend on the arrival times of the input transitions. If the transition on the off-input occurs later than that on the on-input, it will mask the propagation of the fault from the on-input to the output. In this case, the non-robust test is said to be **invalidated**. On the other hand, if the transition on the off-input occurs before the one on the on-input, the fault effect from the on-input will be observable at the output.

DEFINITION 4.3 Let f denote the on-input to gate g in the target path and h denote an off-input to gate g. The off-input h is called a **non-robust off-input** if there is a ncv→cv transition on the on-input f and a cv→ncv transition on the off-input h.

DEFINITION 4.4 A robust untestable path delay fault for which there exists at least one vector such that it activates the required transitions on the path and at least one off-input is a non-robust off-input while the rest of the off-inputs are robust is called **non-robust testable path delay fault**.

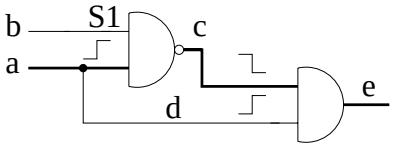


Figure 4.5. Non-robust path.

Therefore, non-robust testable paths are also static sensitizable.

EXAMPLE 4.4 Consider the circuit in Figure 4.5. Path {rising, *ace*} is a non-robust testable path and for the test shown in Figure 4.5 signal d is the non-robust off-input. If the rising transition on signal d arrives later than the transition on signal c, it will mask the propagation of the falling transition from signal c to signal e. In this case the test shown in the figure will not be able to detect the faulty target path (shown in bold).

There could be many different ways to non-robustly sensitize a given path, i.e., a non-robust testable path can have several possible non-robust tests.

These non-robust tests differ in the number and position of non-robust off-inputs in the given target path. A better non-robust test is the one for which the transitions on the non-robust off-inputs have a lower chance to mask the transitions on the target path. Finding such better non-robust tests requires knowledge about the delays in the circuit. Therefore, including the timing information into the test generation process for non-robust paths can result in higher quality non-robust tests. A technique for generating such high quality non-robust tests is described in Chapter 6.

Paths that cannot be tested under robust or non-robust sensitization criteria are called **non-robust untestable paths**. All non-robust testable paths have to be selected for testing to guarantee the performance of the circuit.

4.1.3 Validatable non-robust testable path delay faults

Let a target path be $\{g_1, f_1, g_2, \ldots, g_n\}$, where g_1 and g_n denote a primary

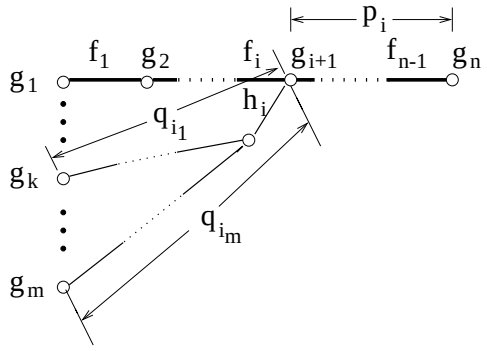


Figure 4.6. Validating a non-robust test.

input and primary output, respectively, and f_i denotes the on-input feeding gate g_{i+1} (Figure 4.6). Suppose that under some non-robust input vector pair V for this path, signal h_i is a non-robust off-input for gate g_{i+1}. Let the partial path from g_{i+1} to g_n be p_i. Under a non-robust test the off-input h_i must be assigned a cv→ncv transition. This transition has to propagate to h_i through one or more partial paths from primary inputs under vector pair V. Let these paths be $q_{i_1}, \ldots q_{i_m}$. If all paths $p_i{+}q_{i_1}, \ldots, p_i{+}q_{i_m}$, where symbol "+" denotes path concatenation, can be robustly tested and if the circuit passes these tests, it can be guaranteed that the non-robust test for the target path will not be invalidated. The target path delay fault is called a **validatable non-robust path delay fault** [71]. The robust tests for the concatenated paths together with the non-robust test V for the target path form a validatable non-robust test. For example, consider again the circuit in Figure 4.5. Path {rising, *ade*} is the only path through signal d that can invalidate the non-robust test shown in the figure. However, this path is robust testable and can be checked for faults prior to applying the non-robust test. If it is faulty, the circuit will be

classified as defective and applying the non-robust test is not necessary. If it is not faulty, the non-robust test in Figure 4.5 is guaranteed to detect the fault on the target path.

Not every non-robust testable path can be tested under validatable non-robust condition. However, a validatable non-robust test is the highest quality non-robust test and should be applied whenever it exists for testing non-robust testable paths. Automatic test generation for validatable non-robust tests is a complex problem. An algorithm for generating validatable non-robust tests will be presented in Chapter 6.

Similar to robustly testable faults, many circuits have only a small percentage of non-robust testable faults.

4.1.4 Functional sensitizable path delay faults

As with the non-robust sensitization criterion, detection of faults on paths that are sensitizable under **functional sensitization criterion** [14] depends on the

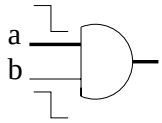


Figure 4.7. Functional sensitizable propagation for AND gate.

delays on signals outside the target path. The functional sensitization criterion requires that there exists more than one faulty path in the circuit in order for the target fault to be detected.

EXAMPLE 4.5 Figure 4.7 illustrates the functional sensitizable criterion for an AND gate. When the on-input a and off-input b both have falling transitions, in order to propagate the fault to the output of the AND gate both transitions have to be late. This is because the arrival time of the signal at the output is determined by the earlier of the two falling transitions.

DEFINITION 4.5 Let a signal f be the on-input to gate g in a target path. Let signal f be the on-input to gate g. The off-input h is called **functional sensitizable (FS) off-input** if there is a ncv→cv transition on both on-input f and the off-input h.

DEFINITION 4.6 A non-robust untestable path delay fault for which there exists an input vector pair such that it activates the required transitions on the path and at least one off-input is FS off-input while the rest of the off-inputs are either robust or non-robust is called **functional sensitizable path**.

A functional sensitizable path is not static sensitizable. If for a given input vector pair that functionally sensitizes some FS path, all of its FS off-inputs are

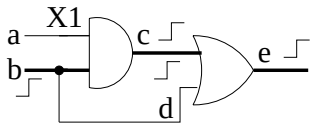


Figure 4.8. Functional sensitizable path.

late, the FS path might determine the performance of the circuit. However, if at least one FS off-input is not late, the FS path cannot impact the performance, i.e., it is a false path.

EXAMPLE 4.6 Consider the circuit in Figure 4.8. Let the target path be {rising, *bce*}. This path is functional sensitizable because under any input vector pair V that propagates a rising transition on this path, the off-input d must be assigned ncv→cv transition (d is an FS off-input).

For each FS off-input there must exist one or more partial paths from primary inputs through which ncv→cv transition can reach the FS off-input. These paths are said to be *associated* with the given off-input.

EXAMPLE 4.7 In Figure 4.8 partial path {rising, *bd*} is associated with the FS off-input d. The value of signal e in the target path is determined by the input that is arriving earlier. Therefore, in order to detect the fault on the target FS path, the transition on the FS off-input d must also be late. This means that the fault effect on the target path can only be observed in the presence of multiple path delay faults.

An FS path can be tested with several different tests. Since the detection of a fault on an FS path depends on the delays on certain other signals in the circuit, for the same target path, different FS tests have a different probability of detecting the defect. In Chapter 6 we describe a methodology for generating good quality FS tests using the circuit timing information.

Functional sensitizable paths can affect the performance only if groups of FS paths are simultaneously faulty. These groups of FS paths belong to the class of faults called **primitive faults**. Primitive faults will be described in greater detail in Section 4.3. A given FS path can belong to many primitive faults. All these primitive faults have to be tested to guarantee that a given FS path delay fault will not affect the performance of the circuit. Therefore, a number of different FS tests have to be applied to test a functional sensitizable path.

Some functional sensitizable paths do not have to be tested to ensure the temporal correctness of the circuit. The properties and identification of such FS paths are discussed in the next section.

4.2 PATH DELAY FAULTS THAT DO NOT NEED TESTING

The main disadvantage of the path delay fault model is the large number of paths in the circuit. For this reason test generation and synthesis for path delay fault testability usually cannot be done for large designs using reasonable amount of resources. Also, large number of faults can imply a large test set and long test application time.

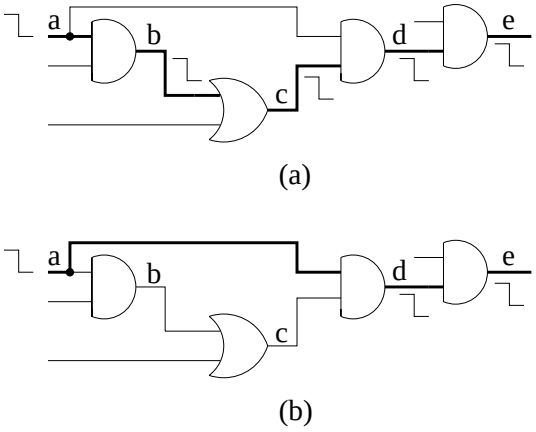


Figure 4.9. Example of a path that does not have to be tested.

Recent research [14, 23, 48, 80] shows that it is usually not necessary to test all path delay faults to guarantee the circuit performance. This is because there exist path delay faults that can never impact the circuit performance unless some other paths also have delay faults. These paths do not have to be tested if the other paths have been tested.

EXAMPLE 4.8 Consider the circuit in Figure 4.9(a). Path P_1 (shown in bold lines) consists of gates a, b, c, d and e. Let P_2 be the path consisting of gates a, d and e (shown in bold lines in Figure 4.9(b)). Clearly, if path P_2 does not have a delay defect that slows down the propagation of a falling transition, then the value on gate d is determined by signal a and not by signal c. Therefore, delay defects on path P_1 can affect the delay of the circuit only if path P_2 also has a delay defect. This implies that path P_1 for a falling transition does not have to be tested if path P_2 for falling transition is tested.

There are many different ways to partition the set of all path delay faults into the set that needs to be tested and the set that does not need to be tested. The test set size and the test generation effort depend on the number of faults in the set that needs to be tested. Therefore, it is important to find a set with minimum number of faults. However, identifying such a minimum set of paths that need to be tested is a complex problem. This section describes several methodologies that have been proposed for partitioning the set of path delay

faults. The classification into robust testable and robust dependent path delay faults proposed by Lam *et al.* [48] is decribed first. After that, the classification into functional irredundant and functional redundant path delay faults proposed by Cheng *et al.* [14] is outlined. Next, the methodologies proposed by Sparmann *et al.* [80] and by Gharaybeh *et al.* [25] for identifying the paths that do not need to be tested are given. Finally, the classification into primitive and non-primitive path delay faults first proposed by [39] is described. Given the path delay fault classification in Figure 4.1, all these techniques agree that all robust, non-robust and validatable non-robust path delay faults have to be selected for testing to guarantee the circuit performance. They also agree that none of the functional unsensitizable faults need to be tested since they can never independently determine the circuit critical path delay. The difference in these techniques is in partitioning the set of FS paths. Many designs have a very large percentage of functional sensitizable paths and finding a minimal set of the FS paths that have to be tested represents an important problem. Comparison of the experimental results obtained using the above techniques shows the trade-offs between the speed and the accuracy of the proposed algorithms.

4.2.1 Robust vs. robust dependent path delay faults

Lam *et al.* [48] partition all path delay faults into two disjoint sets: the set of robust testable delay faults and the set of **robust dependent (RD) faults**. Robust dependent faults are faults that cannot impact the performance of the circuit unless a fault also occurs on some robust testable path.

DEFINITION 4.7 ([48]) Let $\mathcal{D}$ be the set of all path delay faults in circuit C and $\mathcal{R}$ a subet of $\mathcal{D}$. If for all τ, where τ is the delay of the circuit, the absence of delay faults in $\mathcal{D} - \mathcal{R}$ implies that the delay of C is smaller or equal τ, $\mathcal{R}$ is said to be **robust dependent (RD)**.

The RD set is independent from the assignment of delays to the signals, i.e., faults in the RD set can be eliminated from testing under any delay assignment in the circuit.

The methodology in [48] finds an RD set given that the circuit is represented as a leaf-dag. **Directed acyclic graph (dag)** is a directed graph with no directed cycles [77]. **Leaf-dag** represents a rooted dag in which paths that start from the root reconverge only at the input vertices (leaves) [58]. Therefore, leaf-dag represents a circuit consisting of AND and OR gates with multiple fanouts and inverters permitted only at the primary inputs, and with each inverter allowed only a single fanout. Every circuit can be represented as a leaf-dag by gate duplication. There is a one-to-one correspondence between the paths in a circuit and its leaf-dag. The **I-edge** of a path in a circuit is either a connection from the primary input if no inverter is present or the connection immediately after the inverter. A **falling (rising) RD path-set** represents a set of RD path delay faults with falling (rising) transitions at the primary outputs. The

following theorem gives sufficient conditions under which a set of paths is an RD path-set.

THEOREM 4.1 ([48]) Let C be a given circuit and η its leaf-dag. Let $\mathcal{R}_\eta$ be the paths in η corresponding to the set of paths $\mathcal{R}$ in C. Let $\mathcal{M}_\eta$ be the I-edges of $\mathcal{R}_\eta$. If multiple stuck-at-0 (stuck-at-1) fault on $\mathcal{M}_\eta$ is redundant in η, then $\mathcal{R}$ is a rising(falling) RD path-set in C.

The above theorem links the identification of RD set for circuit C to finding redundant multiple stuck-at faults in a leaf-dag of C. In practice, due to high CPU-time and memory requirements, identification of redundant multiple stuck-at faults and the transformation of the circuit to a leaf-dag are not easy to perform. Therefore, Lam *et al.* [48] propose an approximate technique to find an RD set. It is based on identifying redundant multiple stuck-at faults by iteratively identifying redundant single stuck-at faults. It also eliminates the need to unfold the given circuit into a leaf-dag. The algorithm, which finds a maximal RD set (with no claims on how close it is to the maximum RD set), operates on an internally noninverting circuit. **Internally noninverting circuit** is a circuit that has inverters only at the primary inputs. The algorithm is based on the following theorem:

THEOREM 4.2 ([48]) Let C be an internally noninverting circuit and M be a redundant multiple stuck-at-0 (stuck-at-1) fault in C (faults considered on or after I-edges). Let C_M be the circuit obtained by replacing each connection in M by 0 (1). If P is rising (falling) robust path delay fault in C, then P is a rising (falling) path delay fault in C_M.

An internally noninverting circuit C' can be obtained for a given circuit C by duplicating gates in C. C' is at most twice the size of C. Identification of the RD set is done by applying the following two steps. First, replace each redundant stuck-at-0 (stuck-at-1) connection by a 0 (1) to obtain an irredundant circuit. Second, duplicate selected gates to obtain a fanout-free circuit. Since in the second step, new redundancies might be created, the two steps are iterated until the resulting circuit is fanout-free and irredundant. The paths in the resulting circuit that do not pass through any constant connection form the non-RD set.

This procedure identifies a near maximum RD set. However, it is very time and space consuming and can be applied only to small scale circuits. The procedure that will be described next is very efficient and can be applied to much larger circuits. However, the identified paths that do not have to be tested form a superset of the RD set.

4.2.2 Functional irredundant vs. functional redundant path delay faults

Cheng *et al.* [14] propose dividing the set of all paths into two sets: one that contains all robust, non-robust and functional sensitizable paths and the other that contains functional redundant (also called functional unsensitizable) paths.

Paths in the first set are the ones that might affect the performance of the circuit while the functional unsensitizable paths can never influence the performance of the circuit. **Functional unsensitizable path delay faults** are defined as the faults for which under all possible input vector pairs (1) at least one off-input

Figure 4.10. Functional unsensitizable off-inputs for an AND gate.

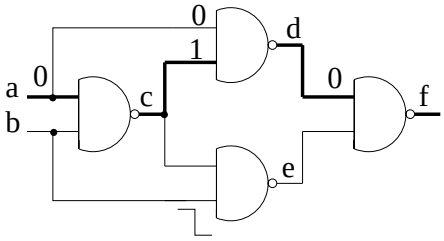


Figure 4.11. Functional unsensitizable path.

in the path has a controlling value under vector v_2 when the corresponding on-input has a non-controlling value or (2) at least one off-input is assigned a stable controlling value. Such off-inputs are called **functional unsensitizable off-inputs**.

EXAMPLE 4.9 Figure 4.10 illustrates functional unsensitizable off-inputs for the case of an AND gate. As an example of a functional unsensitizable path consider the circuit in Figure 4.11. The values in the figure are the values for the second input vector of the vector pair. Path {falling, *acdf*} is a functional unsensitizable path because the propagation of the transition on the target path is stopped at gate d due to the assignment of a controlling value at the off-input and a non-controlling value at the on-input.

To identify the set of functional unsensitizable paths Cheng *et al.* [14] identify a set of non-overlapping, functional unsensitizable prime segments.

DEFINITION 4.8 A partial path $Q = \{f_s, g_{s+1}, f_{s+1}, \ldots, f_{t-1}, g_t\}$ is called a **functional unsensitizable zero (one) segment** if there exists no input vector such that:

(1) it stabilizes f_s at logic 0 (logic 1), and

(2) for each gate g_{i+1} in Q whose on-input is stabilized at a non-controlling value, it stabilizes all off-inputs at their respective non-controlling values.

Any path that covers a functional unsensitizable segment is a functional unsensitizable path. Identification of functional unsensitizable prime segments (defined bellow) allows the counting of the number of functional unsensitizable path delay faults in a given circuit.

DEFINITION 4.9 A functional unsensitizable zero (one) segment Q is a **prime segment** if no proper sub-segment of Q is a functional unsensitizable zero (one) segment.

The algorithm for identifying functional unsensitizable paths has two phases [14]. The first phase finds a partial path $Q = \{g_0, f_0, g_1, \ldots, f_{t-1}, g_t\}$, where g_0 is a primary input, f_i is the on-input to gate g_{i+1} and g_t is the destination of the partial path Q, such that Q is a functional unsensitizable segment. The second phase locates gate g_{s+1} in Q closest to g_t such that $S = \{f_s, g_{s+1}, \ldots, f_{t-1}, g_t\}$ is a functional unsensitizable prime segment. In both phases, the partial path is expanded forward connection by connection (gate by gate). In the beginning, partial path Q is initialized as $\{g_0, f_0, g_1\}$. The primary input g_0 is assigned value 0 or 1. If signal f_0 is set to a non-controlling value, it is required that all off-inputs of Q are also set to non-controlling values. If f_0 is set to controlling value, there are no requirements for assigning the off-inputs of Q. If the current requirements for Q cannot be satisfied, partial path Q is a functional unsensitizable segment, and the first phase stops. Otherwise, the partial path Q is expanded forward to include the next pair consisting of a signal and a gate. Again, if the newly included signal is set to its non-controlling value, all off-inputs to the newly included gate must be set to the non-controlling value as well. If the expanded partial path Q is not a functional unsensitizable segment, the procedure continues. Let the final partial path after the first phase be $Q = \{g_0, f_0, g_1, \ldots, f_{t-1}, g_t\}$. In the second phase, partial path S is initialized to be $\{f_{t-1}, g_t\}$. If on-input f_{t-1} is assigned a non-controlling value, all off-inputs for gate g_t must be assigned non-controlling values. There are no requirements on off-inputs, if the on-input f_{t-1} is assigned a controlling value. If S is determined to be functional unsensitizable segment, a prime segment closest to g_0 is located and the second phase stops. Otherwise, partial path S is expanded backwards and the procedure repeats until S is a prime segment. Once a prime segment is identified, the algorithm continues to search for the next segment until the search space is exhausted. Suppose the last segment generated in the first phase is $Q = \{g_0, f_0, g_1, \ldots, f_{t-1}, g_t\}$. The new Q is formed by removing the last pair of signal and gate, $\{f_{t-1}, g_t\}$, and by adding $\{f'_{t-1}, g'_t\}$, where f'_{t-1} represents the next fanout of gate g_{t-1} and a fanin to gate g'_t. If g_{t-1} does not have any more unexplored fanouts, $\{f_{t-2}, g_{t-1}\}$ is removed and g_{t-2}'s next fanout is added to Q, etc. To determine if a path is functional unsensitizable, the value requirements on off-inputs must be justified. Since a complete justification process is very time consuming for large circuits, to make the procedure efficient only mandatory assignments and their implications are used to find paths that are functional unsensitizable.

The identified set of paths that do not have to be tested using the above procedure can be suboptimal (too large) for two reasons. First, only the second vector of the vector pair required to launch a transition is considered to identify functional unsensitizable paths. This means that some paths that under all possible input vector pairs have a stable controlling value on some off-input are classified as functional sensitizable even though they can never affect the performance of the circuit. Second, only local implications are used to identify functional unsensitizable faults. Fast algorithms for identification of functional unsensitizable faults based on computing static logic implications were recently proposed by Li *et al.* [51] and Heragu *et al.* [31].

4.2.3 Path classification based on input sort heuristic

Sparmann *et al.* [80] propose a technique that shows a trade-off between the efficiency and the size of the identified set of faults that do not require testing. This technique combines the method described in Section 4.2.2 and a heuristic that implicitely orders the paths to classify them into the set that has to be tested and set that does not need to be tested. In comparison with the method in Section 4.2.2, this method can be applied to larger designs and it identifies a larger set of paths that do not have to be tested. As the method Section 4.2.2, this procedure also relies only on mandatory assignments and their implications to find the set of paths that do not require testing. It also uses only the second vector of the vector pair to perform the classification but in addition to identifying functional unsensitizable paths, it also identifies some functional sensitizable paths that do not have to be tested. Functional sensitizable faults must occur together with some other faults in order to affect the performance. As will be explained in Section 4.3, for each such group of simultaneously faulty paths it is sufficient to select one functional sensitizable path for testing. To find a minimal path cover, the heuristic in [80] orders the inputs of each gate in the circuit. This order of inputs is called **input sort**.

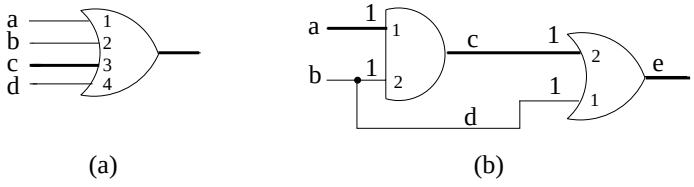


Figure 4.12. Using the *input sort* heuristic.

Given a target path and an input sort, the on-input partitions the off-inputs into higher and lower order off-inputs. For example, consider the four-input OR gate of Figure 4.12(a). Inputs a, b, c and d are assigned integers 1, 2, 3 and 4, respectively. If input c is the on-input, then (for any target path through c) inputs a and b are lower order off-inputs while d is a higher order off-input. Given a path delay fault and an input sort, if a lower order off-input cannot be assigned a non-controlling value, then the off-input is **partially**

static unsensitizable (PSUS). It can be shown that if an FS path has at least one PSUS off-input for all possible input vectors, then the FS path does not have to be tested [80]. These paths will not affect the performance of the circuit unless some robust, non-robust or other functional sensitizable path is faulty at the same time. For example, consider the circuit shown in Figure 4.12(b). The path consisting of gates a, c and e is functional sensitizable for a rising transition. However, the off-input d has a lower order than on-input c and d always assumes a controlling value. Therefore, the target path does not have to be tested. Defects on this path cannot affect the performance unless path {rising, bde} is also defective.

Different input sorts can lead to different partitions of paths into the "to be tested" and "need not be tested" sets. A "good" input sort results in a smaller set of paths that have to be selected for testing. Several different heuristics for finding a "good" input sort have been proposed [80].

4.2.4 Path classification based on using single stuck-at fault tests

Gharaybeh *et al.* [23] derive a new logic model for delay faults and show that path delay faults can be classified using single stuck-at fault test generation process on this model. They classify the set of path delay faults into three categories: singly-testable, multiply-testable and singly-testable dependent. Singly- and multiply-testable paths must be tested to guarantee the performance of the circuit, while singly-testable dependent paths do not have to be tested. **Singly-testable path delay faults** are those that can be guaranteed to be detected under the assumption that no other delay fault exists in the circuit. These faults are either robust, non-robust or validatable non-robust. **Singly-testable dependent path delay faults** cannot affect the performance unless a fault simultaneously happens on some singly-testable path. This set is a superset of the functional unsensitizable set of paths [14] and a subset of the RD set [48]. In addition to functional unsensitizable faults it also contains some of the functional sensitizable faults that do not have to be tested. *Multiply-testable path delay faults* are faults that can affect the performance only together with some other paths and none of these paths is singly-testable dependent. This set contains the functional sensitizable faults that are not in the singly-testable dependent faults set.

4.2.5 Primitive vs. non-primitive path delay faults

The classification of path delay faults into the set of primitive and set of non-primitive faults was first proposed by Ke and Menon [39]. Later a similar classification was proposed by Krstić *et al.* [45] and Sivaraman and Strojwas [78]. *Primitive faults* [39, 40] represent faults that have to be tested in order to guarantee the temporal correctness of the circuit. On the other hand, testing of *non-primitive faults* is not required if tests for primitive faults have been derived. This is because non-primitive faults can never independently affect the performance of the circuit. Primitive faults can be single or multiple. Single

primitive faults usually represent a small portion of all primitive faults. Most of the primitive faults consist of more than one faulty path. Figure 4.13 illustrates

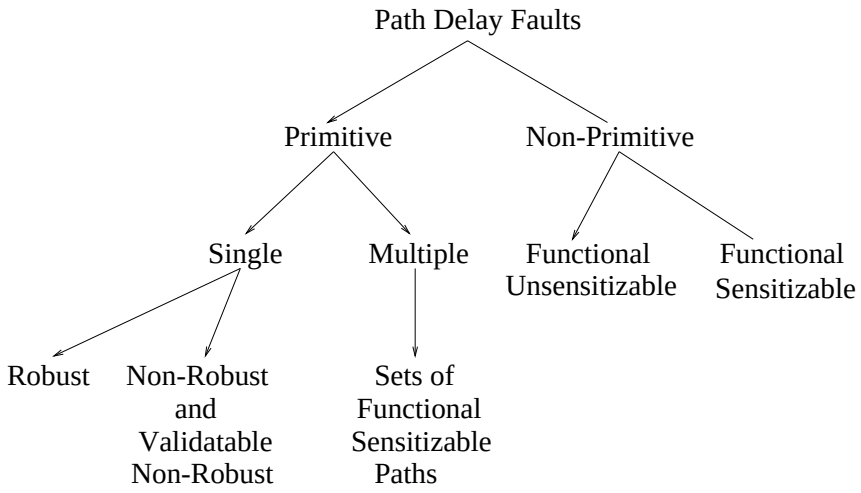


Figure 4.13. Primitive faults vs. non-primitive faults.

the classification of path delay faults into primitive and non-primitive faults. Primitive faults will be discussed in detail in the next section.

4.3 MULTIPLE PATH DELAY FAULTS AND PRIMITIVE FAULTS

The following terminology, taken from [39] is needed in order to formally define primitive faults. A **multiple path** Π is a set of single paths that end at the same primary output. A **multiple path delay fault** (MPDF) on Π is a condition under which every single path in Π has a fault. A signal is called an **on-input of a multiple path** Π if it is on Π. Therefore, a multiple path delay fault can have a gate such that several of its inputs are on-inputs (**multiple on-input**). A signal is called an **off-input of a multiple path** Π if it is an input to a gate in Π but it is not an on-input. An MPDF on π is said to be static sensitizable if there exists at least one input vector pair $V = \langle v_1, v_2 \rangle$ such that it launches the required transitions on all primary inputs in Π and each off-input of Π assumes a final non-controlling value.

EXAMPLE 4.10 In the circuit in Figure 4.14, single path delay faults $P_1 = \{bceghi$, falling$\}$ and $P_2 = \{dghi$, falling$\}$ form a double path delay fault $\Pi = \{P_1, P_2\}$. Gate g has a multiple on-input consisting of two signals, d and e. This MPDF is static sensitizable.

DEFINITION 4.10 ([39]) A **primitive fault** is defined as a multiple path delay fault that satisfies the following two conditions:

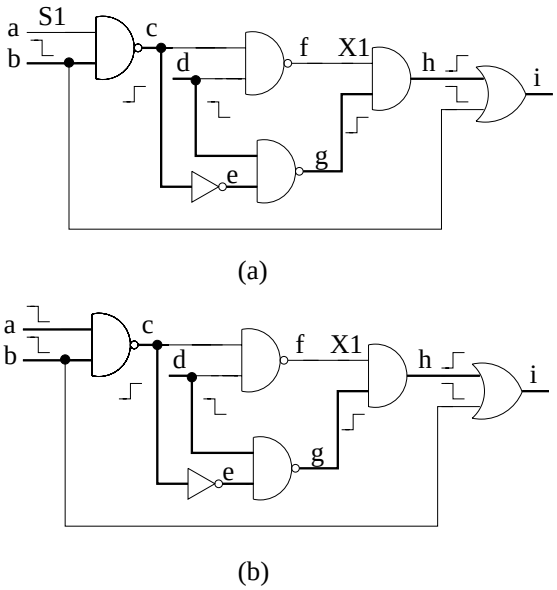


Figure 4.14. Primitive faults.

(1) the MPDF is static sensitizable, and

(2) no proper subset of this MPDF is static sensitizable.

Number of single path delay faults contained in a primitive fault represents the ***cardinality*** of a primitive fault. Robust and non-robust path delay faults satisfy the above two conditions and can be regarded as primitive faults of cardinality 1. Since functional sensitizable faults are not static sensitizable they do not satisfy the first condition for a primitive fault. However, several FS paths together can form a primitive fault. In fact, from the second condition required for primitive faults it follows that all single paths in a primitive fault of cardinality higher than 1 must be functional sensitizable paths.

EXAMPLE 4.11 Consider again the circuit in Figure 4.14. Faults $P_1 = \{bceghi$, falling$\}$ and $P_2 = \{dghi$, falling$\}$ are FS faults. When considered independently, none of these two faults is static sensitizable because it is impossible to find an input vector pair, $V = \langle v_1, v_2 \rangle$, such that all off-inputs to this multiple delay fault are assigned non-controlling values under vector v_2. However, the double path delay fault $\Pi_1 = \{P_1, P_2\}$ is static sensitizable. Gate g has a multiple on-input and no off-inputs. Therefore, MPDF Π_1 is a primitive fault of cardinality 2. If both single paths (P_1 and P_2) are faulty, the test for Π_1 will make the fault observable at the primary output.

Figure 4.13 illustrates the relationship between different classes of single path delay faults and primitive faults.

Since all single paths in a primitive fault end at the same primary output, every primitive fault of cardinality higher than 1 must have at least one gate with a multiple on-input. Such gates are called **merging gates** of the primitive fault [45, 43]. In the above example, gate g is a merging gate for primitive fault $\Pi_1 = \{P_1, P_2\}$. All signals in a multiple on-input must be assigned ncv→cv transition. Single paths in a primitive fault are said to be **co-sensitized** at the merging gates. v

EXAMPLE 4.12 Consider the circuit in Figure 4.14(b). Fault P_3 = {falling, *aceghi*} is also an FS fault. The test shown in the figure sensitizes three FS paths, $\Pi_2 = \{P_1, P_2, P_3\}$ and they merge at gates c and g. However, Π_2 is not a primitive fault because it does not satisfy the second condition for primitive faults (since $\Pi_1 \subset \Pi_2$). To explain this condition, consider first that the circuit has passed the test for Π_1. It means that at least one of the single path delay faults (P_1 or P_2) is not faulty. Let path P_1 be faulty and path P_2 be delay fault free. The test for Π_2 cannot detect the fault on P_2 because it requires that paths P_1 and P_3 are faulty together with P_2. Therefore, if test for Π_1 cannot detect the fault in the circuit, neither can the test for Π_2. Next, consider that the circuit did not pass the test for Π_1. In this case, the circuit will be classified as delay-defective and no testing of Π_2 is required.

Identifying and testing primitive faults is a complex problem. Testing strategies for primitive faults will be addressed in Chapter 6.

Summary

Unlike the stuck-at fault tests that can unconditionally detect the target faults, tests for delay faults differ in their level of quality. This is because detection of delay defects depends on the circuit timing, as well as on the fault model and

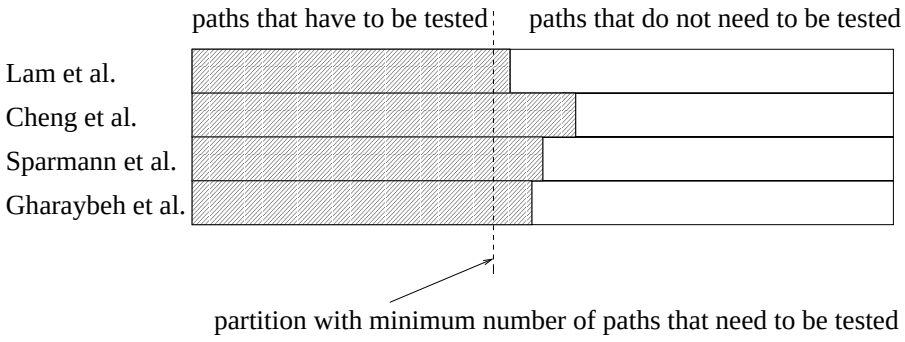


Figure 4.15. Comparison of different techniques described in Section 4.2.

fault size. Path delay faults can be classified into several classes with respect to the test quality. Robust and validatable non-robust tests have the highest quality because they can detect a fault independent of the delays on the signals

outside the target path. Non-robust faults can detect faults if certain other paths in the circuit are not faulty. Functional sensitizable paths may affect the performance only if there are also other faults that simultaneously exist in the circuit.

Testing a path delay fault under the most stringent sensitization condition under which a test exists may lead to erroneous conclusions about the timing correctness of the design. Main problem is in the use of the oversimplified fault model that cannot take into account the dependence of the path delay on the applied test pattern. For example, Pierzynska and Pilarski [63] have shown that a path can have a considerably longer delay if it is tested non-robustly than if it is tested by a robust test. In deep submicron designs these kinds of effects cannot be longer tolerated. The fault models have to be updated to better reflect the nature of defects.

Research on path delay fault testing shows that there exist paths that can never independently impact the performance of the circuit and they do not have to be tested. Several different methodologies [48, 14, 80, 23] have been proposed for finding the set of faults that do not need to be tested and they differ in the size of the identified set as well as in their efficiency and ability to handle large designs. Figure 4.15 shows a comparison of the path delay fault classification methods described in Section 4.2. The size of the primitive fault set does not depend on the specific technique that is used to identify the primitive faults. One way to test all primitive faults is to select one path from each primitive fault and show that it is not faulty. Selecting a more than minimal set of single paths to cover all primitive faults would result in some primitive faults being tested more than once.

5 DELAY FAULT SIMULATION

5.1 TRANSITION FAULT SIMULATION

Transition fault simulation can be performed using adapted stuck-at fault simulators such that they identify logic gates that have transitions in the applied test pattern (relative to the preceding pattern). Simulating transition faults has been addressed by [86, 76, 13]. Waicukauski *et al.* [86] use enhanced parallel-pattern, single fault propagation stuck-at fault simulator [85] to simulate transition faults. The flowchart of the algorithm is shown in Figure 5.1 [86]. The parallel-pattern, single fault propagation fault simulator combines the concepts of multiple pattern evaluation and single fault propagation. It simulates 256 patterns per pass. Single fault propagation minimizes the number of gate evaluations to determine if a given fault is detectable with the set of 256 patterns. Fault values are calculated starting from the fault location and continuing forward only for gates that continue to propagate differences. The rules for finding eqivalence classes for transition faults are given in Section 2.1.

The transition fault model for sequential circuits proposed by Cheng [13] has been discussed in Section 2.1. Cheng [13] also proposes a fault simulation algorithm for these faults (called TFSIM). The at-speed test application strategy is assumed (see Chapter 1). The inputs to the algorithm include: gate-level circuit description, the input sequence and the sizes of transition faults for simulation. The overall flow of TFSIM is shown in Figure 5.2. The algorithm is based on PROOFS stuck-at fault simulation algorithm for sequential

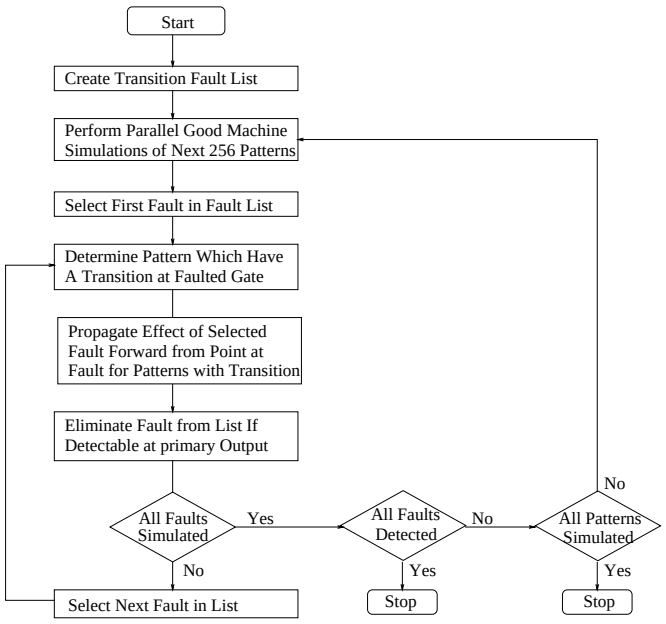


Figure 5.1. Parallel-pattern, single fault propagation transition fault simulation algorithm.

circuits [59]. PROOFS combines the advantages of differential fault simulation, single fault propagation and parallel fault simulation. To fully utilize the bit spaces in a computer word, faults are dynamically grouped to simulate several faulty machines at once (a group represents 32 faulty machines). The dynamic strategy avoids wasting space in the machine word for faults already detected. For each test vector, the algorithm performs the good circuit simulation and does the faulty evaluation for each fault group.

Fault list generation. The fault list contains slow-to-rise and slow-to-fall faults on all signals in the circuit. However, since this transition fault model considers different fault sizes (specified as number of clock cycles), the sizes of transition faults that need to be simulated have to be passed to the fault list generator. For example, if sizes 1 to 5 clock cycles are specified, the fault list generator will generate 5 slow-to-rise and 5 slow-to-fall faults for each signal. Fault collapsing is then performed among the faults of the same size.

Fault injection. After scheduling the state events that contain the differences of the present states between the fault-free and faulty circuits, fault is performed. There are two situations when the faulty event should be injected: (1) A transition occurs at the fault site. The transition should be suppressed for the current time-frame. (2) A transition occured at the fault site in the previous time-frame(s) and is delayed to occur in the current time-frame. To

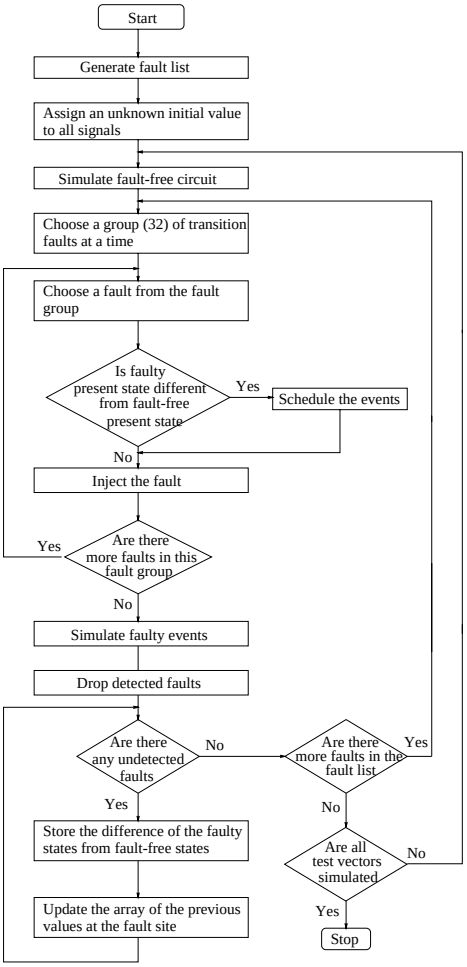


Figure 5.2. Flowchart of the TFSIM algorithm.

examine these conditions, it is necessary to know the value(s) of the faulty signal in previous time-frame(s). If the size of the target transition fault is k clock cycles, the values of the faulty signal in the previous k time-frames have to be known. These values are recorded and updated at the end of the simulation for each vector. At the beginning of the simulation for the first vector, the initial value of the faulty signal is assumed to be an unknown value (U).

A traditional implementation of fault injection uses a flag for each gate. The flag indicates whether the associated gate is faulty or not. This implementation requires that the flags be examined for every gate evaluation even though only one gate is faulty. The fault injection method proposed in [13] does not require the use of special flags. For each fault, an extra gate is inserted into the circuit. The concept of inserting an extra gate for fault injection of stuck-at faults has

been given in [59]. To inject a slow-to-rise fault of size k clock cycles, a $(k+1)$-input AND gate is inserted at the faulty signal. This is illustrated in Figure 5.3. The values of the first k inputs of the extra gate are set to the values of the faulty signal in previous k time-frames. Note that if the value of signal A is a logic 0 in any of the previous k time-frames, the value of signal B in the current time-frame will be a logic 0. If the value of signal A is a logic 1 in all previous k time-frames, the value of signal B in the current time-frame will be the current value of signal A. Similarly, to inject a slow-to-fall fault of size k clock cycles, a $(k+1)$-input OR gate is inserted at the faulty line and the values of first k inputs to the extra gate are set to the values of the faulty signal at the previous k time-frames.

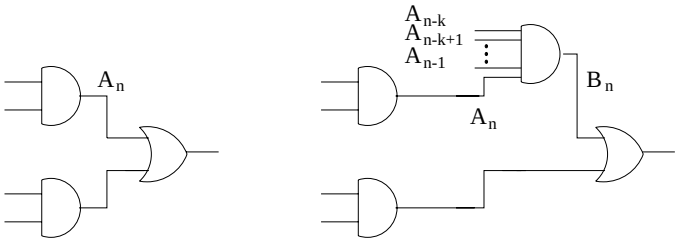


Figure 5.3. Fault injection at time-frame n for a slow-to-rise fault at signal A of size k clock cycles.

EXAMPLE 5.1 Consider the example given in Figure 5.4. The values at signal B can be expressed in terms of the values at signal A as:

$$B_n = A_n \wedge A_{n-1} \wedge A_{n-2}$$

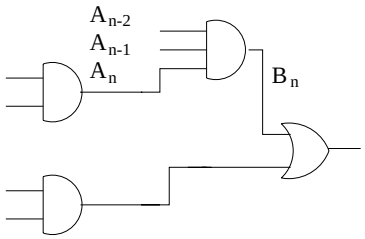


Figure 5.4. Fault injection example.

Therefore, the suggested fault injection technique produces a correct waveform at the fault site.

When simulating 32 faulty circuits in paralell, except for the injected fault, the values of the first k inputs to the extra gate are set to a logic value if an AND gate is inserted and to a logic value 0 if an OR gate is inserted.

Storing states of faulty circuits. At the end of each vector simulation, in addition to storing the difference between the faulty state and the fault-free state, the value at the faulty signal (the $(k+1)$-th fanin of the inserted gate) should be also stored. For each fault of size k clock cycles, an array of size k is used to store the values of the faulty signal in the previous k time-frames. The arrays are updated for all undetected faults at the end of the simulation for each vector. These arrays are used for fault injection as described above.

The TFSIM algorithm has very low memory requirements. In addition to the memory for storing the values of the fault-free circuit for all signals for one time-frame and the memory for storing the differences between the fault-free and each undetected faulty circuit at the next state signals (as used in PROOFS [59]), TFSIM requires an array of k bytes for each undetected fault of size k clock cycles to store the values of the faulty signal in the previous k time-frames.

Unknown initail value. In TFSIM, an unknown initial value (U) is assumed for each signal. Under this assumption, a transition fault may prevent a circuit from fault initialization. As an illustration, consider transition faults of size one clock cycle.

In three-valued logic, there are nine possible combinations of the previous value and current value at a signal: 00, 01, 0U, 10, 11, 1U, U0, U1 and UU. If the value-pair at the fault site is 01, a rising transition occurs and the slow-to-rise fault should be injected. The current faulty value becomes 0. If the value pair is U1, 0U or UU, a rising transition may or may not occur, depending on the power-up states of the flip-flops. For example, if the faulty value-pair is U1 before fault injection, the slow-to-rise fault is injected and the current faulty value becomes U. Similarly, if the faulty value-pair is 0U before fault injection, the slow-to-rise fault is injected and the current faulty value becomes 0. Table 5.1 summarizes the actions of fault injections for all value-pairs.

If teh value-pair of the faulty signal is U1 before injection, the current value of the signal becomes U after the slow-to-rise fault is injected. Similarly, if the value-pair of the faulty signal is U0, the current value of the signal becomes U after the slow-to-fall fault is injected. The following example illustrates that a transition fault may prevent the circuit from being initialized, and thus it is untestable, while the corresponding stuck-at fault is initializable and testable.

EXAMPLE 5.2 Consider a three-bit counter with a reset input. The most significant bit is a primary output. The input sequence is the reset vector followed by a sequence of clock pulses. Table 5.2 lists the states of the fault-free circuit, the states of the circuit with a slow-to-fall fault of size one clock cycle at the least significant bit (denoted LSB in the table), and the states of the circuits with

Value-pair at the faulty signal before injection	after injection	
	slow-to-rise	slow-to-fall
00	-	-
01	00	-
0U	00	-
10	-	11
11	-	-
1U	-	11
U0	-	UU
U1	UU	-
UU	-	-

-: No injection

Table 5.1. Fault injection.

stuck-at-1 and stuck-at-0 faults at the least significant bit. As it can be seen from the table, neither a stuck-at-0 nor a stuck-at-1 fault prevents the circuit from initialization, while the circuit with a slow-to-fall fault is not initialized and is undetected.

input event	fault-free circuit	LSB slow-to-fall	LSB stuck-at-1	LSB stuck-at-0
initial state	UUU	UUU	UUU	UUU
reset	000	00U	001	000
clock	001	0UU	011	000
clock	010	UUU	101*	000
clock	011	UUU	111	000
clock	100	UUU	001	000*
clock	101	...	...	...
clock	110	...	...	...
clock	111	...	...	...
clock	000	...	...	...

*: Fault is detected

Table 5.2. Faulty states for unknown initial values.

However, if we enumerate the two possible initial binary values (0 and 1) at the fault site and simulate them separately, the problem is solved for this example. Table 5.3 lists the faulty states for different initial values. The sequence detects the slow-to-fall fault for both cases at the same vector.

input event	fault-free circuit	LSB slow-to-fall initial value = 0	LSB slow-to-fall initial value = 1
initial state	UUU	UU0	UU1
reset	000	000	001
clock	001	001	001
clock	010	011	011
clock	011	100*	101*
clock	100	101	110
clock	101	111	...
clock	110	...	...
clock	111	...	...
clock	000	...	...

: Fault is detected

Table 5.3. Faulty states for different initial values.

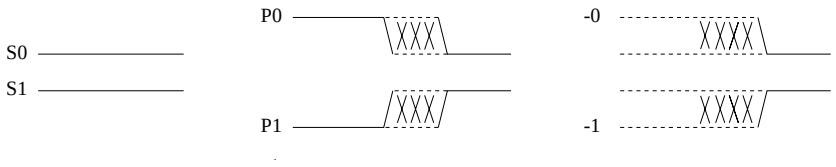
As illustrated, the reported transition fault coverage may be too pessimistic if an unknown initial value is assumed for the faulty signal. A higher fault coverage can be achieved if each transition fault is treated as two faults: one that assumes an initial value 0 at the faulty signal and the other that assumes an initial value of 1.

5.2 GATE DELAY FAULT SIMULATION

5.3 PATH DELAY FAULT SIMULATION

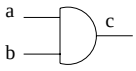
Path delay fault simulation has been investigated by several research groups [79, 10, 7, 65, 67, 38, 32, 24, 36, 28]. Smith [79] proposes a six-valued algebra for simulating robust path delay faults in combinational circuits. It consists of the following values: S0, S1, P0, P1, -0 and -1. Symbol S is used for signals with stable value during the application of a two-vector input. These signals do not have any hazards. The output of a gate is assigned value P if the transition on it cannot occur before

all inputs with value P have changed from their initial values. These signals have a different initial and final value but static or dynamic hazards might exist before the signal stabilizes on its final value. Value – is used for signals that cannot be assigned neither value S nor value P. These signals can have none, one or more transitions. The six-valued logic system is illustrated in Figure 5.5(a). The implication tables for this system for AND, OR and NOT gates are given in Figure 5.5(b). For each two-vector test, the simulation algorithm by Smith [79] traces all undetected paths in the fault list and the paths with value P1 or P0 on each signal in the path are marked as tested and removed from the fault list. The fault list contains all or selected paths in the circuit.

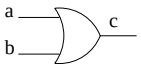


(a)

a / b	S0	P0	-0	S1	P1	-1
S0	S0	S0	S0	S0	S0	S0
P0	S0	-0	-0	P0	-0	-0
-0	S0	-0	-0	-0	-0	-0
S1	S0	P0	-0	S1	P1	-1
P1	S0	-0	-0	P1	P1	P1
-1	S0	-0	-0	-1	P1	-1



a / b	S0	P0	-0	S1	P1	-1
S0	S0	P0	-0	S1	P1	-1
P0	P0	P0	P0	S1	-1	-1
-0	-0	P0	-0	S1	-1	-1
S1	S1	S1	S1	S1	S1	S1
P1	P1	-1	-1	S1	-1	-1
-1	-1	-1	-1	S1	-1	-1



a	b
S0	S1
P0	P1
-0	-1
S1	S0
P1	P0
-1	-0

(b)

Figure 5.5. Implication tables for six-valued logic.

In non-scan sequential circuits, the path delay fault simulation procedure depends on the assumed test application scheme (see Chapter 1). If slow-fast-slow clock testing scheme is assumed, the fault is activated only once during the application of the test sequence for that fault. In at-speed scheme, the fault can be activated many times during the application of the test sequence. In sequential circuits, the path source can be a primary input or a flip-flop. The path destination can be a primary output or a flip-flop. Fault from the target path can affect some flip-flops other than the destination flip-flop. Therefore, the fault simulators have to decide how to update the states of the flip-flops other than the destination flip-flop after the activation of the target fault. For example, if during simulation a rising transition arrives to the destination flip-flop, the correct value of the flip-flop will be 1. If the rising transition arrives late, the destination flip-flop will latch in a value 0. If a transition also reaches some flip-flops other than the destination flip-flop, these flip-flops might latch in an incorrect value [2, 11].

Chakraborty *et al.* [10] address path delay fault simulation for slow-fast-slow and at-speed testing strategy in non-scan sequential circuits. Two different fault models are considered. In the first model, it is assumed that the fault effect reaches only the destination flip-flop. All other flip-flops are assumed to latch in their fault-free circuit values. Tests generated for this fault model can be invalidated by the presence of some faults other than the target fault. In

the second fault model, only those flip-flops that have stable values without hazards during fault activation are updated with their fault-free circuit values. All other flip-flops (except the destination flip-flop) are assigned an unknown value in the faulty circuit. Tests generated for this fault model cannot be invalidated. However, the fault coverage obtained with this fault model can be pessimistic.

Bose *et al.* [7] also consider path delay fault simulation in non-scan sequential circuits. At-speed testing scheme is assumed. The fault simulator is based on the six-valued logic system shown in Figure 5.5. Both robust and non-robust path delay faults are simulated. The only difference in the implication table for AND gate for non-robust path delay fault propagation is for the case (a = P0, b = -1) for which the output c evaluates to P0. This is because it is assumed that input b settles on its final value 1 quickly and the transition from input a propagates to the output c. Similarly, for the OR gate, case (a = P1, b = -0) evaluates to P1. For robust simulation, the *optimistic update rule* is used to update the values of the flip-flops after the fault activation. According to this rule, all flip-flops with non-steady values when the fault is activated are updated with their fault-free circuit values given that they are destinations of at least one robustly activated path. Flip-flops with non-stable values that are result of a non-robust propagation are updated with an unknown value. Flip-flops with steady values are assigned their fault-free values. It can be proven that faults found detectable using the optimistic update rule are guaranteed to cause a failure and cannot be masked by other faults in the circuit [7]. The fault coverage using this rule is higher than if all the flip-flops with non-steady values are updated with an unknown value. For non-robust simulation, all flip-flops, other than the fault destination flip-flop, are updated with their fault-free circuit values.

A variation of slow-fast-slow clocking strategy for simulation of non-scan sequential circuits has been considered by Pomeranz *et al.* [67]. For a test sequence consisting of k vectors all possible $(k-1)$ schemes that have a single fast clock are simulated in parallel (only $(k-1)$ schemes are considered since the first vector is needed for initializing the circuit). In each of the $(k-1)$ schemes the fast clock is applied during a different test vector. Also, application of multiple fast clocks is considered. Usually, paths in the fault list are represented as a sequence of signals that belong to the path. Once a path is detected by a given test sequence it needs to be compared to the list of previously detected and stored path delay faults to find the fault coverage. This in turn, requires comparing sequences of signals. To reduce the memory requirements and increase the speed of the path delay fault simulation process, paths can be represented using *path numbers* [7, 66]. Faults are then represented as pairs (path number, transition type), where the transition type is rising or falling. By assigning a unique path number for each path, paths detected by the given vector sequence can be found by comparing the path numbers rather than comparing a sequence of signal numbers.

The number of path delay faults can be exponential in the number of lines in the circuit. Therefore, fault simulation methods that rely on enumerating the path delay faults both in the fault list and in the list of detected faults cannot process large number of faults. The problem can be overcome by using nonenumerative methods for estimating the path delay fault coverage. Nonenumerative methods for combinational circuits have been proposed by several research groups [65, 37, 28]. Pomeranz *et al.* [66] propose a method that can estimate the fault coverage in time which is polynomial in the number of lines in the circuit. The computed fault coverage is pessimistic in the sense that the exact fault coverage is never smaller than the estimated one. The quality of the estimation can be improved by increasing the polynomial complexity of the method.

The total number of path delay faults can be found in linear time in the number of lines in the circuit [65]. The procedure can be illustrated by the following example.

EXAMPLE 5.3 Consider the circuit in Figure 5.6. To find the number of paths, the gates are processed in a topological order from primary outputs toward primary inputs. During processing, all signals in the circuits are assigned integer numbers as follows: (1) Each primary output is assigned value 1. (2) Each input to a gate is assigned the same value as the gate's output. (3) The values on the fanout stems are obtained as a sum of the values on all of the fanout branches. For example, the value on the fanout stem f is obtained as a sum of the values on the fanout branches f_1 and f_2. The value assigned to each signal in the circuit is shown within square brackets. The number of paths in the circuit can be found as a sum of the values assigned to the primary inputs. In our example, the total number of paths is $4 + 3 + 1 + 2 + 1 = 11$. Then, the number of path delay faults is twice the number of paths.

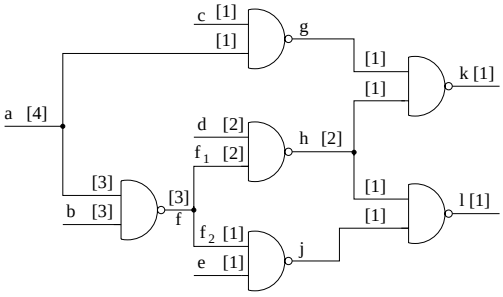


Figure 5.6. Counting the number of paths in a circuit.

To find the fault coverage we need to find the number of detected path delay faults with the given test. The procedure for finding the number of path delay faults robustly detected by a two-input test pattern without enumerating the faults is illustrated by the following example.

EXAMPLE 5.4 Consider the two-vector test shown in Figure 5.7. The values assigned under the test are shown inside the small squares next to each signal. Only primary inputs b, c and e are assigned transitions under the test, while primary inputs a and d are assigned stable 1 (denoted S1) and stable 0 (denoted S0) values, respectively. The list of signals in parentheses indicates the inputs to the given gate that robustly propagate the transition to the gate output. Only signals that are assigned transitions can have a list of signals associated with them. A list on a primary input contains the name of the primary input. For example, primary input b is assigned a falling transition and a list containing signal b ($\{b\}$). The rising transition on signal f is due to the robustly propagated transition from input b. Therefore, signal f is assigned list $\{b\}$. The transitions on signals e and f robustly propagate to signal i and the list on i is $\{e, f\}$. Finally, the primary output k is assigned a transition that robustly propagates from signal i and signal k is assigned list $\{i\}$.

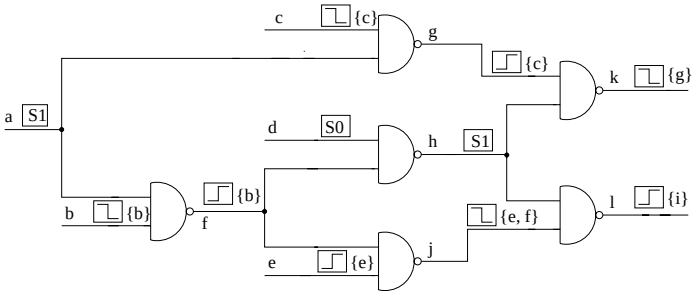


Figure 5.7. Counting the number of paths detected by a two-input test.

5.4 SEGMENT DELAY FAULT SIMULATION

Summary

6 TEST GENERATION FOR PATH DELAY FAULTS

This chapter concentrate on value systems and methodologies proposed for generating tests for single and multiple path delay faults. Robust, non-robust, validatable non-robust and functional sensitizable faults are considered as single path delay faults. These paths usually can be tested with many different tests, i.e., there are many different robust tests for a robust testable path, many different non-robust tests for a non-robust testable path, etc. Since robust tests are guranteed to detect a faulty target path independent on whether or not there are delay faults on paths other than the target path, most test generators do differentiate between robust tests for a given path. These test generators do not use any timing information. On the other hand, some non-robust tests have a higher probability of detecting a faulty non-robust testable path than the others. Similar holds for functional senitizable tests. This chapter presents test generation algorithms that can produce high quality tests based on using the timing information of the circuit for non-robust and functional sensitizable faults. A non-robust test for a given target path becomes invalid if certain other paths in the circuit are defective. If the faults that may invalidate the non-robust test for the target path can be robustly tested, these robust tests along with the non-robust test for the target path form a validatable non-robust test (VNR). An algorithm for automatic generation of validatable non-robust tests is outlined in this chapter.

Testing only single path delay faults is not sufficient to gurantee the circuit performance. Some multiple delay faults (multiple primitive faults) can also

affect the performance. Identifying and testing such faults for large multi-level designs is a hard task. An algorithm that can identify and test double primitive faults is presented.

6.1 ROBUST TESTS

Process variations usually affect delay of more than one gate or interconnect in the circuit. Therefore, it would be ideal if all path delay faults could be tested under conditions that are independent of the delays on signals and gates outside the target path. However, this is possible only for a subset of path delay faults called robust testable faults. Definition of robust testable path delay faults has been given in Section 4.1.

Robust testable path delay faults have been first considered by Smith [79]. He suggested a six-valued logic system to find the path delay faults tested by a given two-vector pattern. Test generation for robust path delay faults has been first addressed by Lin *et al.* [52]. They propose a five-valued logic for generating tests for robust testable paths. The logic system consists of values {S0, S1, U0, U1, XX}. These symbols represent the initial and final values of a signal under vector pair $V = \langle v_1, v_2 \rangle$. Symbol S0 (S1) is used for signals for which the initial and final value are 0 (1). Signals with S0 or S1 values are asssumed to be hazard-free under vector pair V. Symbol U0 (U1) represents signals for which the final value is 0 (1). The initial value can be 0 or 1 or there could be hazards on the signal before it settles on its final value. Symbol XX represents signals for which the initial and final value are unspecified. Figure 6.1 [52] illustrates the covering relations in the proposed logic sytem. Value U0 (U1) covers S0 (S1),

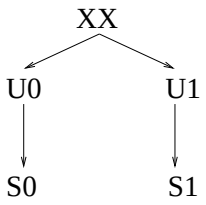


Figure 6.1. Covering relations in the five-valued logic system.

while XX covers both U0 and U1. In this value system a transition can only be represented using U0 (for falling transition) and U1 (for rising transition). Lin *et al.* [52] use the convention that for the signals on the target path symbols U0 and U1 can represent only a falling and rising transition respectively, while for the signals outside the target path these symbols are intrepreted as defined earlier. The implication tables for AND, OR and NOT logic gates for the five-valued logic system are shown in Figure 6.2.

A two-vector test, $V = \langle v_1, v_2 \rangle$, is a robust test for a given path delay fault if it lauches the desired transition at the source of the path and if the values on the path's off-inputs have logic values that are covered by the values given in Figure 6.3 [52]. For example, if the on-input to an AND or NAND gate is

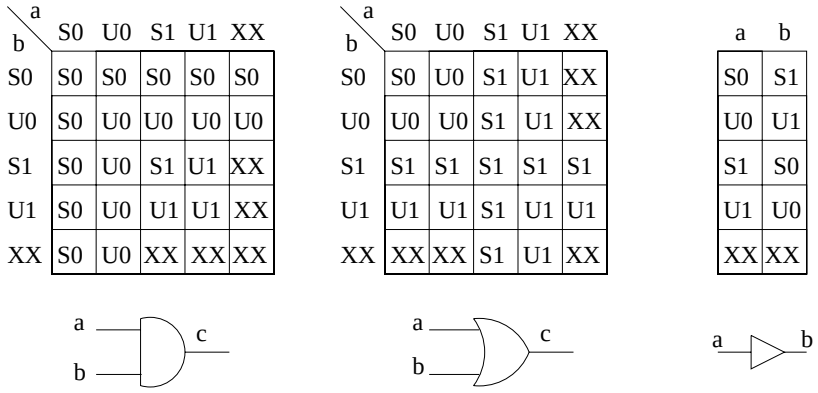


Figure 6.2. Implication tables for AND, OR and NOT gates.

assigned a rising transition, the off-input can be assigned any value covered by U1, i.e., value S1 or U1. As it can be seen from the figure, in the proposed

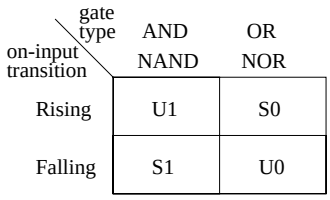


Figure 6.3. Values covering the off-input values.

five-valued logic system the values on the off-inputs are uniquely determined. They satisfy the conditions given by Definition 4.1 for robust off-inputs.

A PODEM-type test generation algorithm based on this logic system has been proposed by Patil *et al.* [62]. It starts with the list of paths for which tests have to be generated. Next, given the path and a transition at the source of the path, a list of signal values required for the on-inputs and off-inputs (according to the table in Figure 6.3) are generated. Given this list, the test generator attempts to justify the signals in the list using a PODEM-type algorithm [26]. After a partially specified test is generated, a heuristic procedure is used to convert the unspecified (XX) or partially specified (U0 or U1) values at primary inputs into specific values such that the probability of detecting the remaining path delay faults with the same test is maximized.

Test generation algorithms for robust path delay faults have also been proposed by many other researchers. For example, Fuchs *et al.* [21] propose a ten-valued logic system and a stepwise path sensitization procedure to generate robust tests. This procedure is capable of handling a large number of path delay faults. The experimental results on many benchmark and industrial designs have shown that usually only a small number of path delay faults can be

tested by a robust test. Saldanha *et al.* [72] propose a robust tests generation algorithm that employs a single-stuck at fault test generator on a modified network.

Recently, it was shown that a two-vector tests for robust path delay faults might not always excite the worse case delay of the target path [20, 63]. The off-input transitions and pre-initialization are shown to have significant impact on the delay of the target path. Chen *et al.* [12] propose a robust path delay fault test generator that incorporates these two additional considerations. It generates three-vector robust tests that excite the worst case delay of the target path.

6.2 HIGH QUALITY NON-ROBUST TESTS

Non-robust testable paths have been defined in Section 4.1. As it was shown, a non-robust test becomes invalid if the transition on *any* non-robust off-input arrives later than the transition on the corresponding on-input. A non-robust testable path can usually be tested with many different non-robust (NR) tests. Test generation techniques for non-robust testable path delay faults have been proposed in literature [71, 21]. However, these techniques cannot make a distinction between different non-robust tests for a given path. Cheng *et al.* [16] show that some non-robust tests have a higher quality than the others. Their algorithm for generating high quality non-robust tests is based on including timing information into the test derivation process. A metric, called *robustness* is introduced to measure how far a given non-robust (NR) test is from a robust test. Then, for each non-robust testable fault, this metric is used to generate a non-robust test which is *closer* to a robust test.

The following notation and definitions will be used in describing the algorithm for generating high quality non-robust tests [16]. Let $V = \langle v_1, v_2 \rangle$ be an input vector pair applied for delay testing of a given target path and let v_2 be applied at time $t = 0$. At some time after $t = 0$ the logic values on the signals in the circuit will become stable. It is assumed that the signal delays in the fault-free circuit are equal to the nominal signal delays.

DEFINITION 6.1 The time when the logic value on signal f becomes stable under v_2 is called **arrival time** of f under v_2. The arrival time at signal f under v_2 in the fault-free circuit is denoted with $AT(f, v_2)$.

DEFINITION 6.2 For a given off-input g and its corresponding on-input f, the difference $AT(f, v_2) - AT(g, v_2)$ is called **slack of the off-input g**.

A non-robust testable path under any test vector pair has at least one non-robust off-input while the rest are robust off-inputs (see Section 4.1.2). The number of non-robust off-inputs for a given target path can be different for different non-robust tests. Since the goal of the algorithm is generating non-robust tests that are *more robust*, its first objective is to find a test with minimal number of non-robust off-inputs for a given target path. Also, two non-robust

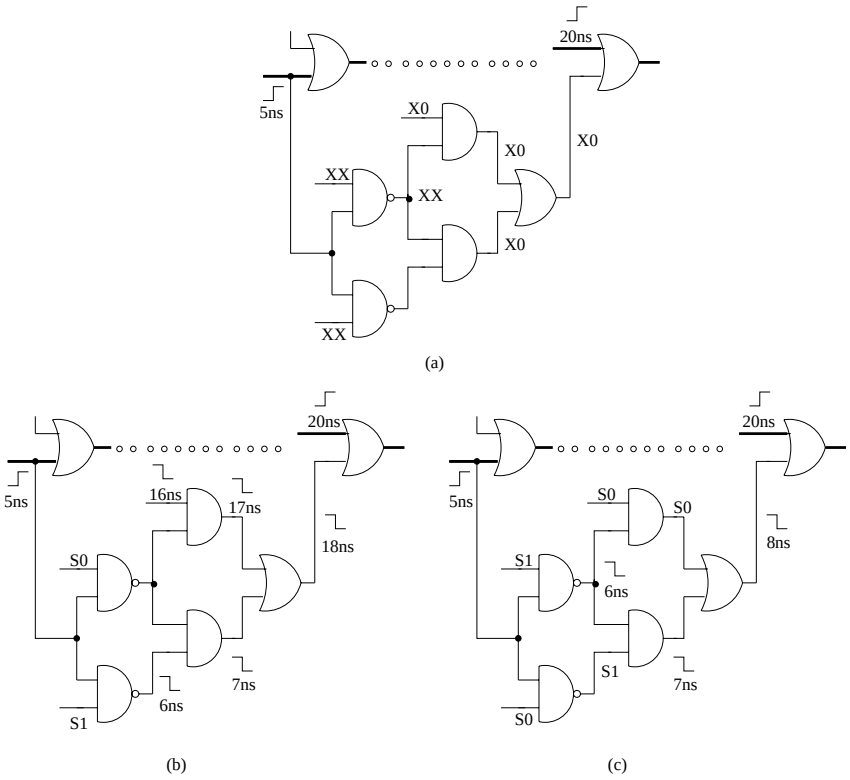


Figure 6.4. Different non-robust tests can tolerate different timing variations.

tests with the same number of non-robust off-inputs can have different quality with respect to their effectiveness in detecting defects. Slack of the non-robust test is defined to help guide the test generation process towards generating high quality non-robust tests.

DEFINITION 6.3 Let $g_1, g_2, \ldots, g_n$ denote the non-robust off-inputs for a given target path under a given vector pair V. Let $s_1, s_2, \ldots, s_n$ denote the slacks of those non-robust off-inputs. Then, the **slack of the non-robust test V** is defined as $\min_{i=1,\ldots,n} \{s_i\}$.

The second objective of the algorithm is to find the non-robust test with the maximum slack among all non-robust tests for a given path. If the slack at the non-robust off-inputs is larger, the probability that the non-robust off-input transition masks the on-input transition is lower. In other words, a non-robust test with a larger slack can tolerate larger timing variations at the non-robust off-inputs. For delay defects caused by process variation, the slack of a non-robust test should be closely related to the probability of fault masking at the non-robust off-inputs.

EXAMPLE 6.1 Consider the circuit in Figure 6.4(a). The path delay fault shown in bold is robustly untestable but non-robustly testable. Figures 6.4(b) and 6.4(c) show two different non-robust tests for the same path. The test given in Figure 6.4(c) can tolerate a 12 ns delay variation on the non-robust off-input, while the test in Figure 6.4(b) can tolerate only a 2 ns delay variation. Clearly, the test in Figure 6.4(c) has a higher quality than the test in Figure 6.4(b).

The **robustness** of a non-robust test is defined as its slack. The higher the robustness, the lower the probability of the test being invalidated. The robustness ranges from $-\infty$ to ∞. The robustness of a robust test is defined as ∞, while robustness of a vector pair which is neither a robust nor a non-robust test is defined as $-\infty$.

For each non-robustly testable path repeat:

1. Assign desired transitions at on-inputs.
2. Convert NR candidate off-inputs into robust off-inputs (if possible), one at a time. Leave the values of the candidate off-inputs which could not be converted into robust off-inputs unassigned. Timing information is used to determine the order of processing the candidates.
3. Convert the rest of the unassigned off-inputs into NR off-inputs one at a time and do backward justification. Timing information is used to determine the order of processing of the candidates and to make decisions during justification process.
4. Assign values to the remaining primary inputs that still have their values unassigned.

Figure 6.5. Summary of the algorithm for generating NR tests with high robustness.

6.2.1 Algorithm for generating non-robust tests with high robustness

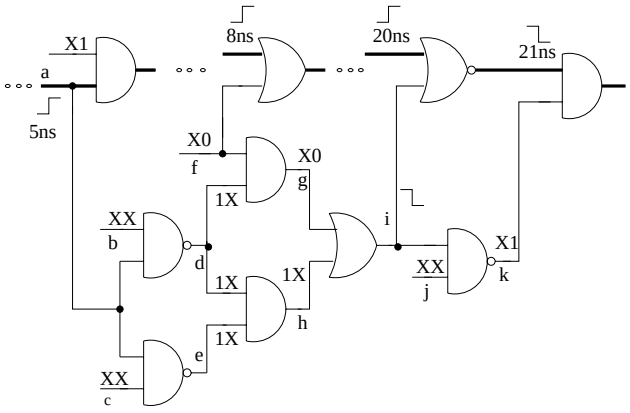
The algorithm for generating non-robust tests with high robustness [16] consists of several steps. A brief summary of the algorithm is given in Figure 6.5. The detailed algorithm and a step-by-step example are given next.

STEP 1: To non-robustly test a path delay fault, all off-inputs must have a non-controlling value under vector v_2 and a transition must be created at the source of the path under test. These requirements have to be satisfied by any test vector and they are called **mandatory assignments** [42]. Hence,

in the first step, all mandatory assignments and their implications [47] are found.

Next, the earliest arrival time of each signal *restricted to these mandatory assignments* are computed. Note that certain input vector pairs may produce values that violate the given set of mandatory assignments (SMA) at some signals. The **earliest arrival time at signal f under a given SMA** is defined as the earliest arrival time at f among all vector pairs not violating the SMA.

EXAMPLE 6.2 Consider the circuit in Figure 6.6(a). The target path is shown in bold. The arrival times and the transitions for the signals on



(a)

$a_r = 5$ns	$d_f = 6$ns	$g_f = 7$ns	$j_r = j_f = 0$ns
$b_r = b_f = 0$ns	$e_f = 1$ns	$h_f = 2$ns	$k_r = 1$ns
$c_r = c_f = 0$ns	$f_f = 0$ns	$i_f = 8$ns	

(b)

Figure 6.6. Computing the SMA and the earliest arrival times.

the target path are as shown. Signals b, c, f and j are primary inputs and the transitions on them are assumed to arrive at time $t = 0$. All gates are assumed to have a unit delay for both, the rising and falling transition. After assigning non-controlling values under the second vector, v_2, to all off-inputs and after implying these values, the obtained set of mandatory assignments is as shown. Next, the earliest arrival times for all signals under the given SMA are computed. These values are shown in Figure 6.6(b). Notation s_f and s_r is used to denote the arrival times of the falling and rising transitions on signal s, respectively. For example, $d_f = 6$ns means that the earliest arrival time for a falling transition on signal d under the given SMA is 6ns.

DEFINITION 6.4 For a non-robust testable path, the off-inputs whose corresponding on-input has a ncv→cv value are called **NR candidate off-inputs**.

EXAMPLE 6.3 In Figure 6.6(a), the off-inputs f, i and k are NR candidate off-inputs.

STEP 2: All NR candidate off-inputs for the target path are identified and an attempt is made to convert them, one at a time, into robust off-inputs by assigning a stable non-controlling value to them. To order the NR candidate off-inputs for processing, the slack of each NR candidate off-input (i.e., the difference between the arrival time of the corresponding on-input and the earliest arrival time of the transition on the NR candidate off-input) is computed. The NR candidate with the smallest slack is processed first. This is because a non-robust off-input with a smaller slack has a higher probability of masking its on-input transition than a non-robust off-input with a larger value of slack.

EXAMPLE 6.4 Consider again the circuit in Figure 6.6(a). The slacks of NR candidate off-inputs are: $slack(f) = 8$ns, $slack(i) = 12$ns and $slack(k) = 20$ns. Since the NR candidate off-input f has the smallest slack, signal f is first tried to be converted into a robust off-input by assigning a stable 0 (S0) value to it. The assignment of S0 value to f and its implications do not cause any conflicts and, therefore, signal f is successfully converted into a robust off-input.

Next, the signal earliest arrival times are incrementally updated. This is needed because for certain signals the signal earliest arrival times may change due to the augmented set of mandatory assignments.

EXAMPLE 6.5 In the circuit in Figure 6.6(a), the earliest arrival time of signal g is removed from the set since g has a stable value under the new SMA and the earliest arrival time of signal i becomes $i_f = 3$ns. The slacks of the NR candidate off-inputs now are: $slack(i) = 17$ns and $slack(k) = 20$ns. Therefore, the off-input i is selected as the next signal for processing. However, under the given SMA it is not possible to assign a stable non-controlling value to signal i and signal k is processed next. Off-input k can be converted into a robust off-input by assigning a stable 0 value to primary input j. The new SMA and the signal earliest arrival times are shown in Figure 6.7.

STEP 3: For the NR candidate off-inputs which cannot be converted into robust off-inputs, assigning non-robust transitions cannot be avoided *under the current, partially assigned test T*. To minimize the probability that the on-input transition is masked by the transition at a non-robust off-input, an attempt is made to find a test for which the arrival time of the transition

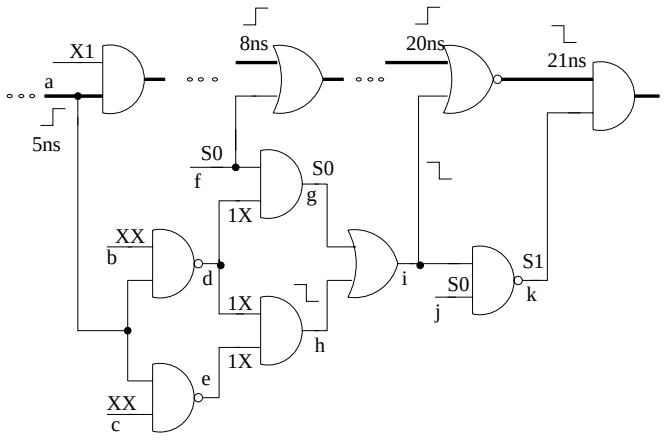


Figure 6.7. Converting non-robust off-inputs into robust off-inputs.

at the off-input is the earliest possible. This can be achieved by using the calculated earliest arrival times to guide the justification process. To justify a transition at the output of a gate, among all inputs that can have a transition under current partial test, a transition is assigned to the input with the earliest arrival time. All other inputs to the gate are assigned stable non-controlling values. This backward justification process continues until the primary inputs are reached or a conflict has occurred. In the latter case, the algorithm backtracks to the last decision point and justifes the transition at the input with the next earliest arrival time. The justification and backtracking process are very similar to those used in stuck-at test generation algorithms.

In this test generation process, the values at internal signals and primary inputs are gradually assigned. Therefore, those NR candidate off-inputs which are processed later will have a smaller search space than those that were processed first. This is why the most critical NR candidate off-inputs (the ones with the smallest slack) first are processed first.

EXAMPLE 6.6 Consider the circuit in Figure 6.7(a). Signal i is a NR off-input for the target path. The objective is to make sure that the falling transition on signal i arrives as early as possible. Since the value at signal h has to be justified using the calculated earliest arrival times for signals d and e it can be seen that the transition on signal i will be the earliest

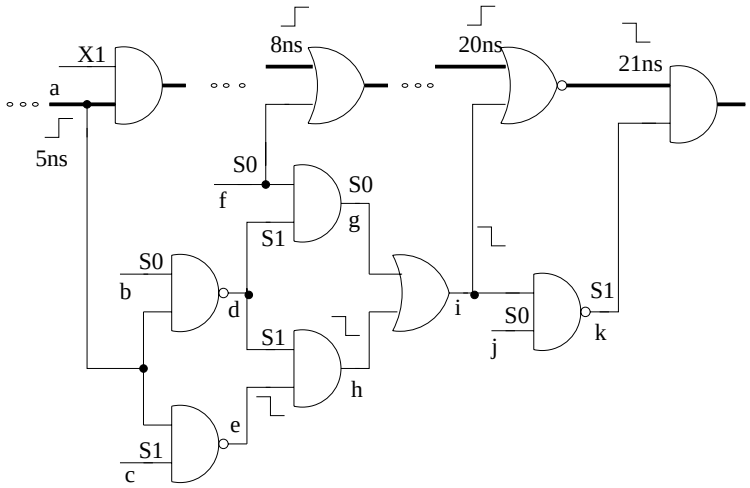


Figure 6.8. Backward justification and assigning unspecified values at PIs.

if a falling transition is assigned to signal e and stable 1 to signal d. This is illustrated in Figure 6.8.

STEP 4: Finally, to obtain a completely specified test and to minimize the number of sensitized paths associated with the non-robust off-inputs, it is necessary to check if there are some primary inputs that do not have values assigned and they are assigned such that the number of transitions at the primary inputs is minimized, i.e., value X1 is specified as 11, X0 is specified as 00, and XX as 00 or 11. Test compaction is not considered in this work. If test compaction is used, this step of the algorithm can be omitted.

EXAMPLE 6.7 The only primary input that does not have values completely specified is input c and in this step it is assigned a stable 1 value. The final values on all signals are shown in Fig. 6.8.

6.3 VALIDATABLE NON-ROBUST TESTS

Validatable non-robust path delay faults have been defined in Section 4.1.3. The validatable non-robust tests guarantee to detect a fault and therefore, should be used for non-robustly testable paths whenever they exist.

There may exist many validatable non-robust tests for a given fault. To reduce the test set size it is of interest to find a test such that the number of paths that have to be robustly tested to validate the non-robust test for the target path is minimal. An algorithm for automatic generation of such a set of two-pattern tests is proposed in [16]. It consists of several steps.

STEP 1: For each non-robust testable fault, the NR candidate off-inputs (defined in Section 6.2) are converted into robust off-inputs and a non-robust

test with a minimal number of non-robust off-inputs is obtained. This is done using the procedure described in Section 6.2. After processing all NR candidate off-inputs, a partially specified non-robust test is obtained.

STEP 2: The unspecified values at the primary inputs are specified such that the number of transitions at the primary inputs is minimized. The final non-robust test is denoted as T. Next, the implications of T are performed.

STEP 3: The non-robust off-inputs are examined and the paths that need to be robustly tested to validate T are identified (see Figure 4.6).

In developing test T, the number of transitions at primary inputs is minimized. Therefore, typically only a very small number of partial paths that end at a non-robust off-input is sensitized. Thus, in Step 3 only a small number of paths need to be examined. This not only reduces the computational complexity but it also reduces the cardinality of the validatable non-robust test.

There is a possible extension of this algorithm for identifying more VNR testable paths. The condition for robustly testing the sensitized partial paths (from primary inputs to non-robust off-inputs), can be relaxed so that they are tested under the VNR condition. However, if this extension is adopted, the following situation may occur: in generating a VNR test for path A, the VNR testability of path B is required and in generating a VNR test for path B, the VNR testability of path A is required. In this case, neither of the paths is VNR testable.

As Cheng *et al.* [16] show, including the timing information into the process of non-robust test generation can substantially improve the quality of a non-robust test. However, experimental results on a large set of benchmark and industrial designs show that typically only a small number of paths in the circuit can be tested under non-robust and validatable non-robust criteria [21]. On the other hand, most circuits have a large number of functional sensitizable faults. These faults under can, certain conditions, affect the performance of the design. Therefore, to further improve of the quality of delay tests, it is necesary to derive tests for functional sensitizable paths.

6.4 HIGH QUALITY FUNCTIONAL SENSITIZABLE TESTS

Functional sensitizable path delay faults have been defined in Section 4.1.4. To *guarantee* that a given functional sensitizable (FS) path delay fault will not affect the performance of the circuit, a set of functional sensitizable tests has to be applied. These tests will be considered in Section 6.5. In this section the goal is to generate a small number of tests (e.g., one) for a given FS path such that they are *most likely* to detect the fault.

Different functional sensitizable tests have a different probability of detecting a defect on an FS path.

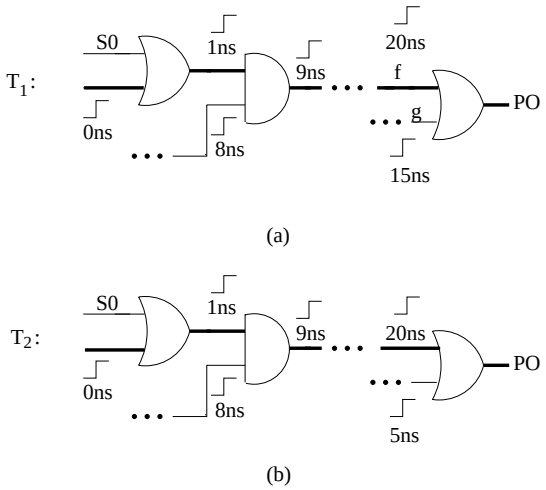


Figure 6.9. Different quality functional sensitizable tests.

EXAMPLE 6.8 Consider the functional sensitizable test T_1 shown in Figure 6.9(a). The target path is shown in bold and signal g is a functional sensitizable off-input under test T_1. Under this test, in the fault-free circuit, the rising transition at off-input g arrives at 15ns while the rising transition on the on-input f arrives at 20ns. The arrival time of the transition at the primary output will be determined by the earlier arriving signal between signals f and g. This means that, if in a faulty circuit, the transition at the off-input g arrives more than 5ns late, the fault on the target path will propagate to the primary output. Otherwise, the fault on the target path will not be detectable with test T_1. Next, consider the functional sensitizable test T_2 for the same target path (shown in Figure 6.9(b)). Signal g is again a functional sensitizable off-input. However, under test T_2 in the fault-free circuit, the transition at signal g arrives at 5ns. Therefore, in the faulty circuit, the rising transition on the off-input g needs to be more than 15ns late to observe the fault on the target path.

If one to can afford selecting only a small number of input vector pairs to test a functional sensitizable path, one should select those tests that have a higher probability of making the defects on the FS path observable. In the above example, test T_1 is better than test T_2 since it requires smaller delay faults on signals outside the target path to detect a defect on the target path. Therefore, for a good FS test, the nominal arrival times of the FS off-inputs should be as late as possible, i.e., the slack of the FS off-in[put should be as small as possible. An algorithm for generating high quality functional tests is given in [16] and it will be reviewed in this section. The algorithm is based on using the circuit timing information to derive high quality FS tests. For each tested FS path only one FS test is derived.

A necessary condition to detect a defective functional sensitizable path with a given test is that the transitions on the FS off-inputs arrive later than the

transitions on the on-inputs. Hence, in general, if the number of FS off-inputs is larger, the probability of detecting a defect is smaller. The number of functional sensitizable off-inputs in a given path depends on the applied test vector pair. Therefore, one of the goals of the algorithm in [16] is to minimize the number of FS off-inputs. Reduction of the number of FS off-inputs can be achieved by maximizing the number of off-inputs that are robust or non-robust. However, two FS tests with the same number of FS off-inputs can still have a different quality (as the above example shows).

DEFINITION 6.5 Let $g_1, g_2, \ldots, g_n$ denote the FS off-inputs under a given vector pair V, and let $s_1, s_2, \ldots, s_n$ denote the slacks of those FS off-inputs. Then the **slack of the FS test** V is defined as $\max_{i=1,\ldots,n} \{s_i\}$.

The second goal of the test generation algorithm is to find tests whose value of the slack is minimal. The slack in this case can be positive or negative. The algorithm searches for a test with the most negative slack, if it exists. Otherwise, it looks for one with the least positive slack. Such tests can tolerate only small timing variations on their FS off-inputs and are more likely to lead to the detection of a faulty FS path.

Algorithm for generating functional sensitizable tests. The set of FS paths that need to be tested can be identified using any of the algorithms described in Section 2.2. In the process of generating non-robust tests the off-inputs whose corresponding on-input has a ncv→cv transition were called NR candidate off-inputs. Here, in the context of generating tests for FS paths, such off-inputs are redefined as FS candidate off-inputs.

DEFINITION 6.6 The off-inputs whose corresponding on-input has a ncv→cv transition are called **FS candidate off-inputs**.

The algorithm for generating high quality functional sensitizable tests for a given FS path consists of several steps [16].

STEP 1: Find the FS candidate off-inputs.

STEP 2: Depending on the transitions assigned under the given test vector, an FS candidate off-input can become either robust, non-robust or functional sensitizable off-input. The goal is to minimize the number of FS off-inputs. Thus, an attempt is made to generate a test under which the highest number of FS candidate off-inputs is either robust or non-robust. Because the transition on a non-robust off-input can mask the transition on the corresponding on-input while this can never happen in the case of a robust off-input, it is preferred to have as large number of robust off-inputs as possible. The FS off-input candidates are processed one at a time. For each candidate, the first attempt is to convert it into a robust off-input by assigning a stable non-controlling value to it. Similar to the algorithm in Section 6.2, the forward implication and backward justification process are used to check if

the desired transition can be assigned. If the attempt to convert some FS candidate off-input into robust off-input fails, its value is left unspecified and the algorithm proceeds with the next candidate in the list.

STEP 3: After all FS off-input candidates have been processed, the algorithm starts over with the ones that have their values still unspecified and tries to convert them into non-robust off-inputs by assigning a cv→ncv transition to them.

STEP 4: If the attempt to convert some FS candidate off-input into non-robust off-input fails, a ncv→cv transition is assigned to it, i.e., it is converted into an FS off-input.

The order of processing the FS candidate off-inputs is very important for generating good quality FS tests. Therefore, the algorithm uses the partial timing information available under the current set of mandatory assignments and their implications to calculate the earliest arrival time of the transition on each of the FS candidate off-inputs. This information is then used for calculating the slack of each candidate. The candidates with the larger slack are processed first since they are less likely to allow the propagation of the defect to the primary output (see example in Figure 6.9). Therefore, it is desired that they be converted into robust or non-robust off-inputs. The FS candidate off-inputs that have a smaller slack have a higher chance of sensitizing the given FS path under faulty conditions and they can be converted into FS off-inputs. Each time after some FS candidate off-input has successfully been assigned a transition the set of mandatory assignments and their implications are updated and the slacks of the FS off-input candidates are recompured.

STEP 5: After the appropriate transitions have been assigned to all FS candidate off-inputs, the final justification process is done in a way which maximizes the chance that the non-robust off-inputs arrive as early as possible while the FS off-inputs arrive as late as possible. If the target path has both, non-robust and FS off-inputs, and if the fanin cones of some non-robust and FS off-inputs intersect, the justification process is done so that the non-robust off-input arrives as early as possible.

To generate tests under which the FS off-inputs arrive as late as possible, the fanins to a gate whose output needs to be justified are sorted such that the latest arriving signal is the first in the list. If the transition that needs to be justified is cv→ncv, an attempt is made to assign this transition to all fanins that could have a transition under the current, partially assigned vector pair, processing them as they appear in the fanin list. If the transition that needs to be justified is ncv→cv, the algorithm tries to assign this transition to the fanin with the latest arrival time among all the fanins that could have a transition under the partially assigned test and a stable non-controlling value to the rest of the inputs. If there is a conflict in this justification process, the algorithm backtracks to the last decision point and attempts

the justification with the next fanin in the list. In order to achieve that the non-robust off-input arrives as early as possible a similar strategy is used but the fanins are ordered such that the earliest arriving signal is first in the list.

STEP 6: After the justification process is done, if there are some primary inputs that still have unspecified values, they are assigned such that the number of transitions on the primary inputs is minimized.

The summary of the algorithm is given in Figure 6.10.

For each functional sensitizable path repeat:

1. Assign desired transitions at on-inputs.
2. Convert FS candidate off-inputs into robust off-inputs (if possible), one at a time. Leave the values of the candidate off-inputs which could not be converted into robust off-inputs unassigned. Timing information is used to determine the order of processing of the candidates.
3. Convert unassigned off-inputs into non-robust off-inputs (if possible), one at a time. Timing information is used to determine the order of processing of the candidates.
4. Convert the rest of unassigned off-inputs to FS off-inputs.
5. Do backward justification. Timing information is used to make decisions during justification process.
6. Assign values to the remaining primary inputs that still have their values unassigned.

Figure 6.10. Summary of the algorithm for generating FS tests.

This algorithm cannot guarantee that a fault on an FS path will be detected. Detection can be guaranteed only if all primitive faults have been tested.

6.5 TESTS FOR PRIMITIVE FAULTS

Primitive faults have been defined in Section 4.3. Since testing of primitive faults of cardinality 1 has been addressed in Sections 6.1, 6.2 and 6.3, this section focuses on testing primitive faults with two or more paths. Therefore, in the sequel, term "primitive fault" will be used to denote primitive faults of cardinality higher than 1.

Testing primitive faults was first addressed by Ke and Menon [39]. They propose a technique for synthesizing a two-level circuit such that all path delay faults in the circuit are primitive. Since representing a circuit in two-level form may not be practical, they use algebraic transformations to convert a two-level circuit into a multi-level circuit while preserving its testability. The test set generated for the two-level circuit can also detect all primitive faults in the multi-level circuit. The problem with this approach is that multi-level circuits obtained using only algebraic transformations are not area-efficient and synthesis tools usually use various Boolean transformations for area/performance/power optimization [5].

A method for identifying and testing primitive faults in multi-level circuits was first reported by Krstić *et al.* [45]. It will be reviewed in this section. Another method for testing primitive faults in multi-level circuits was presented by Sivaraman *et al.* [78]. This technique identifies primitive faults by identifying sets of paths which determine the signal stabilization time at the circuit outputs. To be able to handle medium sized benchmark circuits, iterative approach is proposed such that finding primitive faults of cardinality n requires previous identification of all primitive faults of cardinality $(n-1)$. For larger circuits, due to memory constraints only low cardinality (up to 3) primitive faults are possible to be identified.

All individual paths in primitive faults of cardinality higher than 1 are functional sensitizable paths. The methodology proposed by Krstić *et al.* [45] identifies primitive faults by considering individual FS paths and by identifying all primitive faults that the given FS path is involved in. The algorithm is based on the observation that single path delay faults contained in a primitive fault have to merge at one or more gates to form a multiple path delay fault and such gates can be quickly identified. The technique can be combined with timing analysis or any other method for selecting functional sensitizable paths that need to be tested (see Section 4.2).

The following notation and definitions will be used in the description of the algorithm for identifying and testing primitive faults. For a given signal s, two kinds of FS paths terminating at the same primary output can be defined: (1) FS paths for which the signal s assumes a a controlling value under vector v_2 (**cv-FS paths through *s***) and (2) FS paths for which signal s assumes a non-controlling value under vector v_2 (**ncv-FS paths through *s***). For simplicity reasons, the main ideas of the proposed methodology are explained assuming that the circuit contains only 2-input gates. Extension to circuits containing gates with arbitrary number of inputs is straightforward. Multiple output circuits are handled by processing each primary output and its fanin cone separately.

6.5.1 Co-sensitizing gates

Co-sensitizing gates with respect to a primary output. A primitive fault must have at least one merging gate. If one identifies gates that can never be the merging gates for any primitive fault, then one can significantly reduce

the effort required to identify and test primitive faults. This is especially true if the number of non-merging gates represents a significant fraction of the total number of gates in the circuit. Gates that could possibly be the merging gates in some primitive fault are called **co-sensitizing gates**. Gates on which no merging of FS paths to form a primitive fault is possible are called **non-co-sensitizing gates**. As it will be shown later, identification of co-sensitizing gates is easier than identification of merging gates.

DEFINITION 6.7 A 2-input gate g is a **co-sensitizing gate** if every input to the gate has at least one cv-FS path passing through it.

If a gate has more than two inputs, then the gate is a co-sensitizing gate if it has at least two inputs with non-zero number of cv-FS paths passing through them. If an input to some gate has no cv-FS paths passing through it, then all paths passing through this input with a controlling value under vector v_2 are either robust, non-robust or functional redundant. Therefore, this input cannot be part of any path in a primitive fault. For example, consider the circuit in Figure 6.11. There are six functional sensitizable paths: $P_1 = \{\text{falling}, aceghi\}$, $P_2 = \{\text{falling}, dghi\}$, $P_3 = \{\text{falling}, acfhi\}$, $P_4 = \{\text{rising}, acfhi\}$, $P_5 = \{\text{rising}, bcfhi\}$ and $P_6 = \{\text{falling}, bceghi\}$. The circled numbers adjacent to each signal show the number of cv-FS paths through that signal. Gates f, h and i have just

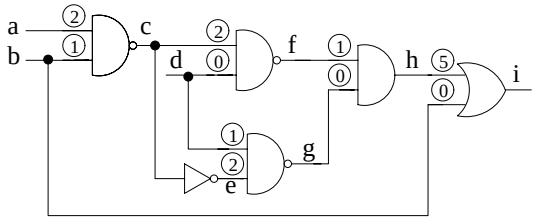


Figure 6.11. Co-sensitizing gates for a given PO.

one input with cv-FS paths and they cannot be co-sensitizing gates. The only co-sensitizing gates are c and g. If a circuit has multiple outputs, then a gate can be a co-sensitizing gate with respect to one PO and a non-co-sensitizing gate with respect to another PO.

Co-sensitizing gates with respect to an FS path. Condition given by Definition 6.7 identifies the co-sensitizing gates for any FS path in the fanin cone of a given primary output. To find the primitive faults that a given FS path is involved in, the information about co-sensitizing gates can be further refined. In this refinement process, some of the co-sensitizing gates with respect to a given PO might become non-co-sensitizing with respect to the target path. However, all non-co-sensitizing gates with respect to the given PO will stay non-co-sensitizing with respect to any target FS path ending at that PO. Several refinement conditions are illustrated through examples.

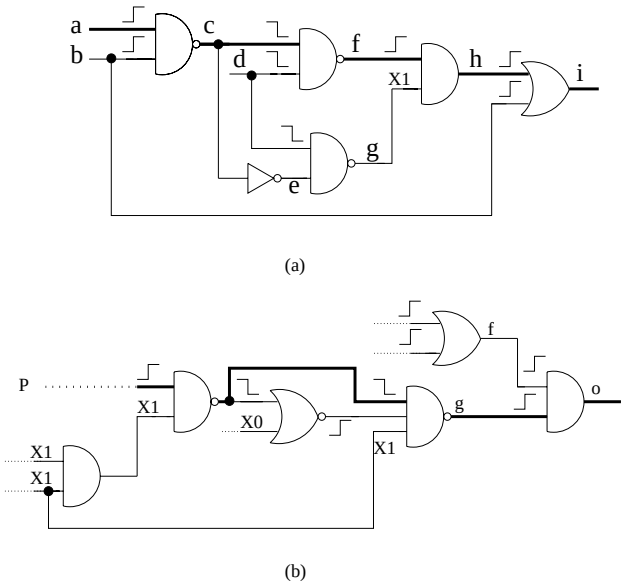


Figure 6.12. Co-sensitizing gates for a given FS path.

EXAMPLE 6.9 In the circuit in Figure 6.12(a), gate c is a co-sensitizing gate with respect to the primary output i. Let the FS path for which the primitive faults need to be found be $P_4 = \{\text{rising}, acfhi\}$. For a gate to be a co-senstitizing gate with respct to a given path, the on-input and the off-input must be assigned a non-controlling value under vector v_2. Since under the set of mandatory assignments (SMA) for P_4 in Figure 6.12(a) the transition on input a does not have a controlling value under vector v_2, gate c cannot be a co-sensitizing gate for path P_4.

EXAMPLE 6.10 Assume that the logic in Figure 6.12(b) is a part of some larger circuit. Let path P be the target FS path and let gates g and f be the co-sensitizing gates with respect to the primary output o. The signal values shown in Figure 6.12(b) belong to the SMA for path P. Since gate g is the last co-sensitizing gate in P (closest to the primary output), no gate outside the fanin cone of gate g can be a co-sensitizing gate for path P. Therefore, gate f cannot be a co-sensitizing gate for path P.

EXAMPLE 6.11 Also, since under the SMA for P all off-inputs to gate g are assigned a non-controlling value under vector v_2, gate g cannot be a co-sensitizing gate for path P.

6.5.2 Merging gates

There are several reasons why a co-sensitizing gate may not be a merging gate. First, computing the exact number of cv-FS paths through each signal in the circuit can be prohibitively expensive since it requires consideration of two vectors (v_1 and v_2). Approximate values for the number of cv-FS paths through each signal can be efficiently computed by using only the information about the second vector v_2 (see Section 4.2). However, these numbers are pessimistic in the sense that a functional redundant path may be counted as an FS path. On the other hand, FS paths can never be misclassified. Second, even if the cv-FS path count can be computed exactly, the information about the number of cv-FS paths passing through the inputs to a co-sensitizing gate does not take into account any correlations between these paths. Correlations between the FS paths can be twofold: correlations manifested by a co-sensitizing gate for which the cv-FS paths cannot pass through together under any input vector pair and correlations manifested by a co-sensitizing gate for which a multiple delay fault involving only FS paths exists, but it is a superset of some primitive fault. One simple way to account for some of the correlations between the paths passing through a given co-sensitizing gate is to assign ncv→cv transition to both of its inputs and to find the forward and backward implications of this assignment. If a conflict is detected, then the gate is not a co-sensitizing gate.

6.5.3 Identifying FS paths not involved in any primitive fault

As discussed in Section 4.2, not all path delay faults need to be selected for testing. If all robustly and non-robustly testable path delay faults are selected, then some FS paths do not have to be selected. In addition to the techniques for identifying the set of FS paths that need to be tested described in Section 4.2, the information about co-sensitizing gates can be used to perform the selection. If an FS path under the SMA does not have any co-sensitizing gate, then the path does not have to be tested.

EXAMPLE 6.12 Consider again the circuit in Figure 6.12(a). Gates c and g are co-sensitizing gates with respect to the primary output. The SMA and their implications for the functional sensitizable path P_4 = {rising, $acfhi$} are as shown. As discussed earlier, gate c is not a co-sensitizing gate for path P_4. Therefore, P_4 has no co-sensitizing gates and does not have to be tested. The fault on the target path can be observed only if paths {falling, $dfhi$} and {rising, bi} are also faulty. However, both of these paths are robustly testable and if any of them is faulty the robust test set will detect the fault. Similarly, it can be determined that path P_5 = {rising, $bcfhi$} does not have co-sensitizing gates and it does not have to be tested. Information about FS paths that do not require testing can be used to update the number of cv-FS paths through each signal. For example, paths P_4 and P_5 pass through signals c and h with a controlling value but these paths do not have to be tested. Therefore, the number of cv-FS paths through signals c and h reduces by 2. The updated values for cv-FS paths for our example circuit are shown in Figure 6.13.

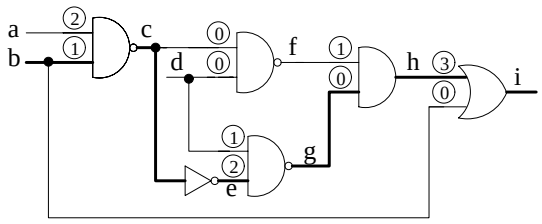


Figure 6.13. Updating the number of cv-FS paths.

This process of eliminating FS paths that do not have co-sensitizing gates and updating the number of cv-FS paths through each signal can be repeated for several iterations. Updating the values for cv-FS paths results in a smaller number of co-sensitizing gates and a smaller number of FS paths that have to be tested and helps reducing the test generation effort for primitive faults. However, note that there exist FS paths that do not have to be tested but cannot be identified by examining only the number of co-sensitizing gates. This is because not every co-sensitizing gate is a merging gate.

EXAMPLE 6.13 Consider the FS path P_3 = {falling, $acfhi$} in the circuit in Figure 6.13. It has one co-sensitizing gate (gate c) but it can be co-sensitized with a robust path {falling, $bcfhi$} and therefore, does not need to be tested.

6.5.4 Primitive faults of cardinality 2

A given FS path can participate in many primitive faults. The cardinality k of these primitive faults can be anywhere from 2 up to the total number of FS paths terminating at the given PO. To guarantee that a fault on an FS path will not affect the performance of the circuit, all primitive faults that involve the FS path have to be identified and tested. The probability that k long functional sensitizable paths will simultaneously be affected by defects significantly reduces as k increases. Therefore, identification and testing of primitive faults of low cardinality is of the highest significance. The following algorithm focuses on primitive faults of cardinality 2.

Algorithm for identifying primitive faults of cardinality 2. The algorithm for identifying and testing primitive faults of cardinality 2 consists of two parts [45]. First, the co-sensitizing gates and FS paths that need to be tested are identified. This part of the algorithm involves several steps.

STEP 1: The co-sensitizing gates with respect to the given PO are identified. This requires knowledge about the number of FS paths passing through each signal. This number can be found using the technique described in Section 4.2.2. It gives the upper bound on the number of cv-FS and ncv-FS paths through each signal. Therefore, the number of co-sensitizing gates we identify is also an upper bound.

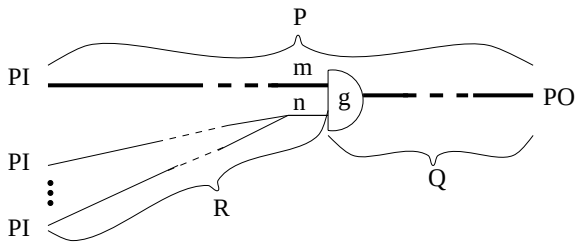


Figure 6.14. Testing path P.

STEP 2: Some of the correlations between the cv-FS paths that pass through a given co-sensitizing gate can be accounted for by assigning a ncv→cv transition to both of its inputs and by checking for conflicts during the implication process.

STEP 3: For each FS path, the information about co-sensitizing gates is further refined. Since, Step 1 identifies co-sensitizing gates with respect to a PO, it is possible that some of these gates may not be co-sensitizing gates with respect to the target FS path. If there are no co-sensitizing gates on the target path, then the target path does not have to be tested. The number of cv-FS and ncv-FS paths are updated for all the signals on the FS path. To reduce the test generation effort, this step can be repeated several times and it can also be combined with the path ordering heuristic described in Section 4.2.3.

Second, the knowledge about co-sensitizing gates is used to find primitive faults of cardinality 2 that include a given FS path. Circuit in Figure 6.14 will be used to explain the algorithm. Gate g is a co-sensitizing gate for the FS path P. The off-input n of g is an FS off-input. Partial paths R represent possible paths associated with the FS off-input n (paths that can propagate a transition from PI to gate g). The partial path of P from gate g to the primary output is denoted as Q. The primitive faults of cardinality 2 for the FS path P are found as follows:

STEP 1: Identify co-sensitizing gates under the SMA for the FS path P.

STEP 2: For each co-sensitizing gate g in P repeat:

(a) Derive a new set of mandatory assignments SMA' for P by assigning the following values:

(i) assign a ncv→cv transition to the on-input m and to the off-input n of the co-sensitizing gate g, and

(ii) (ii) assign a non-controlling value for the second vector v_2 to off-inputs of all other co-sensitizing gates in P. This limits the cardinality of primitive faults to 2.

(b) If SMA' is not consistent, go to Step 2. Else if SMA' is consistent, explore partial paths R from the off-input n to any PI in order to find a co-sensitizing path that together with P forms a primitive fault of cardinality 2. The partial path R must be statically sensitizable under SMA' and it also must be contained in some FS path. The strategy described in Section 4.2.2 can be used to quickly identify signals in the input cone of signal n for which, under the current SMA, no static sensitizable path can pass through. Also, in this step, the information about the number of cv-FS and ncv-FS paths passing through each signal can be used to prune the search for the partial path. Once a partial path R to PI is found, it is necessary to check if the path co-sensitized with P at gate g is functional sensitizable. If yes, a primitive fault of cardinality 2 is found. If the path obtained by concatenation of path R and path Q is an FS path and partial path R is not static sensitizable under SMA', it means that path P might be involved in a primitive fault of cardinality higher than 2. If this is the case, we proceed by exploring a different partial path R. The process continues until all statically sensitizable partial paths from n to any PI are found.

The flowchart of the algorithm is shown in Figure 6.15. Next, the second part of the algorithm is illustrated with an example.

EXAMPLE 6.14 Consider the circuit in Figure 6.16. The circled numbers in Figure 6.16(a) show the number of cv-FS paths passing through the given signal while the numbers inside the boxes show the number of ncv-FS paths. The values are shown only for those signals for which the number of cv- or ncv-FS paths is non-zero. Gates i, o and v are the co-sensitizing gates with respect to the primary output w. Let the target path P for which all primitive faults of cardinality 2 need to be found be {rising, $afnqruvw$}. This path is shown in bold in Figure 6.16(b). The SMA for P is also shown in the figure (ignore for now the values shown inside the boxes since they are not part of the SMA for P). The only co-sensitizing gate on P is gate v. Therefore, to find a co-sensitized path which together with path P forms a primitive fault of cardinality 2, only partial paths associated with off-input t have to be explored. Note that, if the information about co-sensitizing gates had not been used, all partial paths associated with off-inputs i, m and t (potential FS off-inputs in the target path) would have to been explored. Next, a non-controlling value is assigned for the second vector v_2 to all off-inputs other than off-input t and a ncv$\rightarrow$cv transition to off-input t. These values are shown within small boxes in Figure 6.16(b). Then, the partial paths from signal t to any primary input are examined. The partial path associated with off-input t must be static sensitizable under the current SMA. Since the path co-sensitized with P has to be functional sensitizable, the search effort can be reduced by using the information about the number of cv-FS and ncv-FS paths passing through each signal. For example, there are no ncv-FS paths through input m of gate p (Figure 6.16(a)) and gate p in Figure 6.16(b) assumes value 1 for vector v_2. Therefore, all partial paths passing through signal m can be eliminated from further consideration. On the

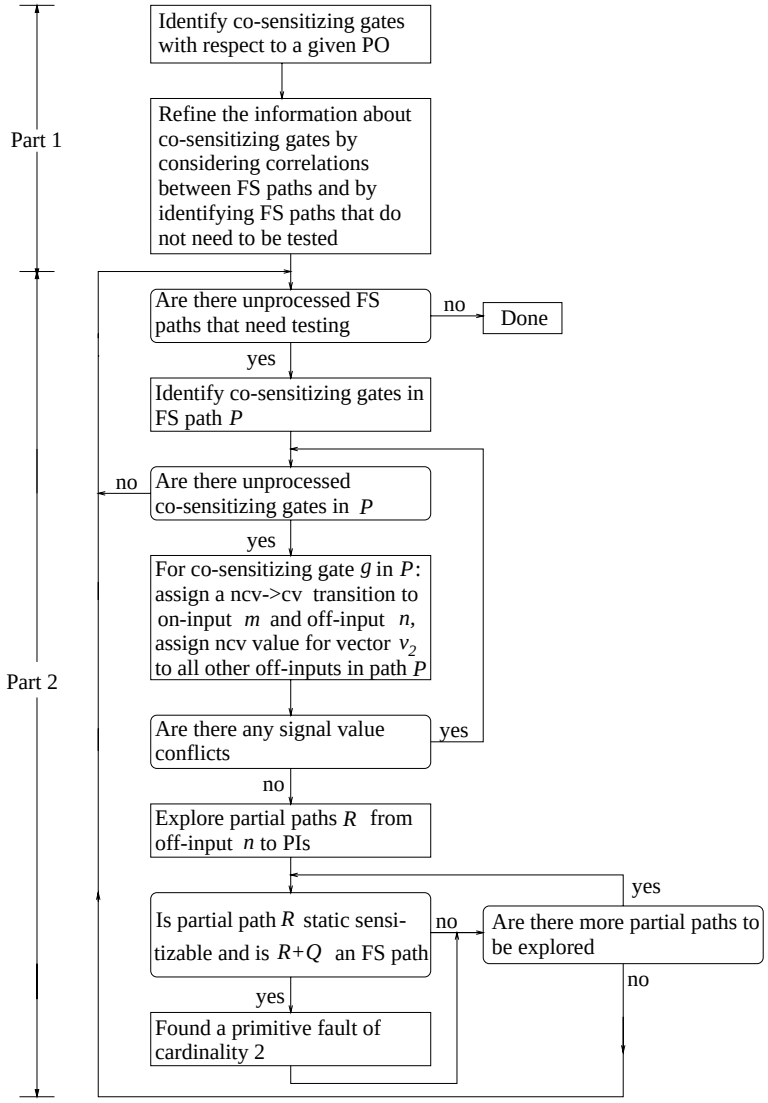


Figure 6.15. Flowchart of the algorithm for identifying and testing primitive faults of cardinality 2.

other hand, input i to gate p has 2 ncv-FS paths passing through it and the algorithm has to continue checking all paths passing through signal i. Once a partial path is identified, it is necessary to check if the path co-sensitized with P is functional sensitizable. In the example there are two primitive faults of cardinality 2 containing path P. The first primitive fault includes path P and {rising, $diptvw$}. The second primitive fault includes paths P and {rising, $eiptvw$}.

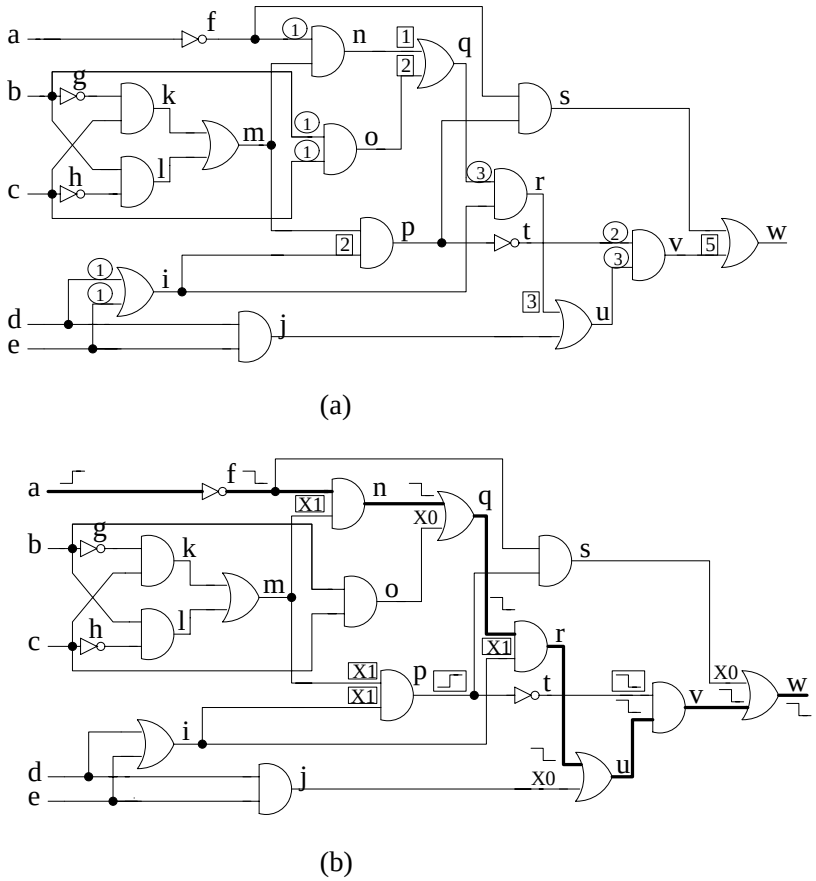


Figure 6.16. Testing primitive faults of cardinality 2.

Summary

Experimental results show that test sets generated under robust testability criteria cannot detect all timing defects. A defect on a non-robust target path will not be detected if the transitions on certain signals outside the target path arrive late. Among all possible NR tests for a non-robustly testable path some NR tests are better than others in detecting delay defects. A good NR test can tolerate larger timing variations and its probability of being invalidated is smaller. Generating such tests requires the use of circuit timing information.

Generating tests for robust and non-robust path delay faults is not sufficient to cover all possible delay defects. Functional sensitizable paths can, under certain conditions, be responsible for the circuit's performance degradation. Defects on these paths can be detected only if multiple delay faults exist.

Ignoring these multiple path delay faults, called primitive faults, in the process of test generation can lead to a poor delay test quality. It is very difficult to

identify and test all primitive faults in multi-level circuits. The existing techniques can efficiently deal only with primitive faults that consists of a small number of single paths.

7 DESIGN FOR DELAY FAULT TESTABILITY

8 SYNTHESIS FOR DELAY FAULT TESTABILITY

9 CONCLUSIONS AND FUTURE WORK

References

[1] L. Ackner and M. R. Barber. Frequency Enhancement of Digital VLSI Test Systems. *Proceedings of IEEE International Test Conference*, pages 444–451, October 1990.

[2] P. Agrawal, V. D. Agrawal, and S. C. Seth. Generating Tests for Delay Faults in Nonscan Circuits. *IEEE Design & Test of Computers*, pages 20–28, March 1993.

[3] V. D. Agrawal and T. J. Chakraborty. High-Performance Circuit Testing with Slow-Speed Testers. *Proceedings of IEEE International Test Conference*, pages 302–310, October 1995.

[4] V. D. Agrawal, C.-J. Lin, P. W. Rutkowski, S. Wu, and Y. Zorian. Built-In Self-Test for Digital Integrated Circuits. *AT&T Technical Journal*, 73:30–39, March 1994.

[5] K. A. Bartlett et al. Multilevel Logic Minimizing Using Implicit Don't Cares. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(6):723–740, June 1988.

[6] S. Barton. Characterization of High-Speed (Above 500 MHz) Devices using Advanced ATE - Technique, Results and Device Problems. *Proceedings of IEEE International Test Conference*, pages 860–868, October 1989.

[7] S. Bose, P. Agrawal, and V. D. Agrawal. Path Delay Fault Simulation of Sequential Circuits. *IEEE Transactions on VLSI Systems*, 1(4):453–461, December 1993.

[8] O. Bula, J. Moser, J. Trinko, M. Weissman, and Woytowich. Gross Delay Defect Evaluation for A CMOS Logic Design System Product. *IBM Journal of Research and Development*, 34(2–3):325–338, March–May 1990.

[9] J. L. Carter, V. S. Iyengar, and B. K. Rosen. Efficient Test Coverage Determination for Delay Faults. *Proceedings of IEEE International Test Conference*, pages 418–427, September 1987.

[10] T. J. Chakraborty, V. D. Agrawal, and M. L. Bushnell. Path Delay Fault Simulation Algorithms for Sequential Circuits. *Proceedings of First Asian Test Symposium*, pages 52–56, November 1992.

[11] T. J. Chakraborty, V. D. Agrawal, and M. L. Bushnell. Delay Fault Models and Test Generation of Random Logic Sequential Ciruits. *Proceedings of 30th Design Automation Conference*, pages 453–457, June 1993.

[12] L.-C. Chen, S. K. Gupta, and M. A. Breuer. High Quality Robust Tests for Path Delay Faults. *Proceedings of 15th IEEE VLSI Test Symposium*, pages 88–93, May 1997.

[13] K.-T. Cheng. Transition Fault Testing for Sequential Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(12):1971–1983, December 1993.

[14] K.-T. Cheng and H.-C. Chen. Classification and Identification of Nonrobust Untestable Path Delay Faults. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(8):845–853, August 1996.

[15] K.-T. Cheng, S. Devadas, and K. Keutzer. Delay-Fault Test Generation and Synthesis for Testability Under a Standard Scan Design Methodology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(8):1217–1231, August 1993.

[16] K.-T. Cheng, A. Krstić, and H.-C. Chen. Generation of High Quality Tests for Robustly Untestable Path Delay Faults. *IEEE Transactions on Computers*, 45(12):1379–1392, December 1996.

[17] B. I. Dervisoglu and G. E. Strong. Design for Testability: Using Scanpath Techniques for Path-Delay Test and Measurement. *Proceedings of IEEE International Test Conference*, pages 365–374, October 1991.

[18] S. Devadas. Delay Test Generation for Synchronous Sequential Circuits. *Proceedings of IEEE International Test Conference*, pages 144–152, September 1989.

[19] S. Devadas, K. Keutzer, and S. Malik. Computation of Floating Mode Delay in Combinational Circuits: Theory and Algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(12):1913–1923, December 1993.

[20] P. Franco and E. J. McCluskey. Three-Pattern Tests for Delay Faults. *Proceedings of 12th IEEE VLSI Test Symposium*, pages 452–456, April 1994.

[21] K. Fuchs, F. Fink, and M. H. Schulz. DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults. *IEEE*

Transactions on Computer-Aided Design of Integrated Circuits and Systems, CAD–10:1323–1335, October 1991.

[22] J. A. Gasbarro and M. A. Horowitz. Techniques for Characterizing DRAMS with a 500 MHz Interface. *Proceedings of IEEE International Test Conference*, pages 516–525, October 1994.

[23] M. A. Gharaybeh, M. L. Bushnell, and V. D. Agrawal. Classification and Modeling of Path Delay Faults and Test generation Using Single Stuck-Fault Tests. *Proceedings of IEEE International Test Conference*, pages 139–148, October 1995.

[24] M. A. Gharaybeh, M. L. Bushnell, and V. D. Agrawal. An Exact Non-Enumerative Fault Simulator for Path-Delay Faults. *Proceedings of IEEE International Test Conference*, pages 276–285, October 1996.

[25] M. A. Gharaybeh, M. L. Bushnell, and V. D. Agrawal. Classification and Modeling of Path Delay Faults and Test generation Using Single Stuck-Fault Tests. *Journal of Electronic Testing: Theory and Applications*, 11(1):55–67, August 1997.

[26] P. Goel. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. *IEEE Transactions on Computers*, C-30(3):215–222, March 1981.

[27] H. Hao and E. J. McCluskey. Very Low Voltage Testing for Weak CMOS Logic ICs. *Proceedings of IEEE International Test Conference*, pages 275–284, October 1993.

[28] K. Heragu, V. D. Agrawal, M. L. Bushnell, and J. H. Patel. Improving A Nonenumerative Method to Estimate Path Delay Fault Coverage. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(7):759–762, July 1997.

[29] K. Heragu, J. H. Patel, and V. D. Agrawal. Segment Delay Faults: A New Fault Model. *Proceedings of 14th IEEE VLSI Test Symposium*, pages 32–39, May 1996.

[30] K. Heragu, J. H. Patel, and V. D. Agrawal. SIGMA: A Simulator for Segment Delay Faults. *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 502–508, November 1996.

[31] K. Heragu, J. H. Patel, and V. D. Agrawal. Fast Identification of Untestable Delay Faults Using Implications. *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 642–647, November 1997.

[32] Y.-H. Hsu and S. K. Gupta. A Simulator for At-Speed Robust Testing of Path Delay Faults in Combinational Circuits. *IEEE Transactions on Computers*, 45(11):1312–1318, November 1996.

[33] V. S. Iyengar, B. K. Rosen, and I. Spillinger. Delay Test Generation 1 – Concepts And Coverage Metrics. *Proceedings of IEEE International Test Conference*, pages 857–866, September 1988.

[34] V. S. Iyengar, B. K. Rosen, and I. Spillinger. Delay Test Generation 2 – Algebra And Algorithms. *Proceedings of IEEE International Test Conference*, pages 867–876, September 1988.

[35] V. S. Iyengar, B. K. Rosen, and J. A. Waicukauski. On Computing The Sizes of Detected Delay Faults. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(3):299–312, March 1990.

[36] D. Kagaris, S. Tragoudas, and D. Karayiannis. Improved Non-Enumerative Path-Delay Fault-Coverage Estimation Based on Optimal Polynomial-Time Algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(3):309–315, March 1997.

[37] D. Kagaris, S. Tragoudas, and D. Karayiannis. Nonenumerative Path Delay Fault Coverage Estimation with Optimal Algorithms. *Proceedings of International Conference on Computer Design*, pages 366–371, October 1997.

[38] B. Kapoor. An Efficient Method for Computing Exact Path Delay Fault Coverage. *Proceedings of European Design and Test Conference*, pages 516–520, March 1995.

[39] W. Ke and P. R. Menon. Delay-Verifiability of Combinational Circuits Based on Primitive Faults. *Proceedings of IEEE International Conference on Computer Design*, pages 86–90, October 1994.

[40] W. Ke and P. R. Menon. Synthesis of Delay-Verifiable Combinational Circuits. *IEEE Transactions on Computers*, 44(2):213–222, February 1995.

[41] D. C. Keezer. Multiplexing Test System Channels for Data Rates Above 1Gb/s. *Proceedings of IEEE International Test Conference*, pages 790–797, October 1990.

[42] T. Kirkland and M. R. Mercer. A Topological Search Algorithm For ATPG. *Proceedings of 24th Design Automation Conference*, pages 502–508, June 1987.

[43] A. Krstić, S. T. Chakradhar, and K.-T. Cheng. Design for Primitive Delay Fault Testability. *Proceedings of IEEE International Test Conference*, pages 436–445, November 1997.

[44] A. Krstić and K.-T. Cheng. Resynthesis of Combinational Circuits for Path Count Reduction and for Path Delay Fault Testability. *Proceedings of European Design and Test Conference*, pages 486–490, March 1996.

[45] A. Krstić, K.-T. Cheng, and S. T. Chakradhar. Identification and Test Generation for Primitive Faults. *Proceedings of IEEE International Test Conference*, pages 423–432, October 1996.

[46] A. Krstić, K.-T. Cheng, and S. T. Chakradhar. Testing High Speed VLSI Devices Using Slower Testers. 1998. submitted to *International Test Conference*.

[47] W. Kunz and D. K. Pradhan. Recursive Learning: A New Implication Technique for Efficient Solutions to CAD Problems - Test, Verification and Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1143–1158, September 1994.

[48] W. K. Lam, A. Saldanha, R. K. Brayton, and A. L. Sangiovanni-Vicentelli. Delay Fault Coverage, Test Set Size, and Performance Trade-Offs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(1):32–44, January 1995.

[49] Y. Levendel and P. R. Menon. Transition Faults in Combinational Circuits: Input Transition Test Generation and Fault Simulation. *Proceedings of International Fault Tolerant Computing Symposium*, pages 278–283, July 1986.

[50] W.-N. Li, S. M. Reddy, and S. K. Sahni. On Path Selection in Combinational Logic Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(1):56–63, January 1989.

[51] Z. Li, Y. Min, and R. K. Brayton. Efficient Identification of Non-Robustly Untestable Path Delay Faults. *Proceedings of IEEE International Test Conference*, pages 992–997, November 1997.

[52] C. J. Lin and S. M. Reddy. On Delay Fault Testing in Logic Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(5):694–703, September 1987.

[53] A. K. Majhi and V. D. Agrawal. Tutorial: Delay Fault Models and Coverage. *Proceedings of The 11th international Conference on VLSI Design*, pages 364–369, January 1998.

[54] A. K. Majhi, J. Jacob, L. M. Patnaik, and V. D. Agrawal. On Test Coverage of Path Delay Faults. *Proceedings of The 9th International Conference on VLSI Design*, pages 418–421, January 1996.

[55] Y. K. Malaiya and R. Narayanswamy. Modeling And Testing for Timing Faults in Synchoronous Sequential Circuits. *Design & Test of Computers*, 1(4):62–74, November 1984.

[56] P. C. Maxwell, R. C. Aitken, V. Johansen, and I. Chiang. The Effectiveness of IDDQ, Functional And Scan Tests: How Many Fault Coverages Do We

Need? *Proceedings of IEEE International Test Conference*, pages 168–177, October 1992.

[57] P. C. Maxwell, R. C. Aitken, R. Kollitz, and A. C. Brown. IDDQ And AC Scan: The War Against Unmodelled Defects. *Proceedings of IEEE International Test Conference*, pages 250–258, October 1996.

[58] G. D. Micheli. *Synthesis And Optimization of Digital Circuits*. McGraw-Hill, New York, 1994.

[59] T. M. Niermann, W.-T. Cheng, and J. H. Patel. PROOFS: A Fast, Memory-Efficient Sequential Circuit Fault Simulator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(2):198–207, February 1992.

[60] E. S. Park and M. R. Mercer. Robust and Nonrobut Tests for Path Delay Faults in A Combinational Circuit. *Proceedings of IEEE International Test Conference*, pages 1027–1034, September 1987.

[61] E. S. Park, B. Underwood, T. W. Williams, and M. R. Mercer. Delay Testing Quality in Timing-Optimized Designs. *Proceedings of IEEE International Test Conference*, pages 897–905, October 1991.

[62] S. Patil and S. M. Reddy. A Test Generation System for Path Delay Faults. *Proceedings of IEEE International Conference on Computer Design*, pages 40–43, October 1989.

[63] A. Pierzynska and S. Pilarski. Pitfalls in Delay Fault Testing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(3):321–329, March 1997.

[64] I. Pomeranz and S. M. Reddy. At-Speed Delay Testing of Synchronous Sequential Circuits. *Proceedings of 29th Design Automation Conference*, pages 177–181, June 1992.

[65] I. Pomeranz and S. M. Reddy. An Efficient Nonenumerative Method to Estimate The Path Delay Fault Coverage in Combinational Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(2):240–250, February 1994.

[66] I. Pomeranz and S. M. Reddy. On Identifying Undetectable and Redundant Faults in Synchronous Sequential Circuits. *Proceedings of 12th IEEE VLSI Test Symposium*, pages 8–14, April 1994.

[67] I. Pomeranz and S. M. Reddy. SPADES-ACE: A Simulator for Path Delay Faults in Sequential Circuitswith Extensions to Arbitrary Clocking Schemes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(2):251–263, February 1994.

[68] I. Pomeranz and S. M. Reddy. On Synthesis-for-Testability of Combinational Logic Circuits. *Proceedings of 32nd Design Automation Conference*, pages 126–132, June 1995.

[69] A. K. Pramanick and S. M. Reddy. On The Detection of Delay Faults. *Proceedings of IEEE International Test Conference*, pages 845–856, September 1988.

[70] A. K. Pramanick and S. M. Reddy. On The Fault Coverage of Gate Delay Fault Detecting Tests. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(1):78–94, January 1997.

[71] S. M. Reddy, C. J. Lin, and S. Patil. An Automatic Test Pattern Generator for the Detection of Path Delay Faults. *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 284–287, November 1987.

[72] A. Saldanha, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Equivalence of Robust Delay-Fault and Single Stuck-Fault Test Generation. *Proceedings of 29th Design Automation Conference*, pages 173–176, June 1992.

[73] J. Savir. Skewed-Load Transition Test: Part I, Calculus. *Proceedings of IEEE International Test Conference*, pages 705–713, October 1992.

[74] J. Savir. Skewed-Load Transition Test: Part II, Coverage. *Proceedings of IEEE International Test Conference*, pages 714–722, October 1992.

[75] J. Savir. On Broad-Side Delay Testing. *Proceedings of 12th IEEE VLSI Test Symposium*, pages 284–290, April 1994.

[76] M. H. Schulz and F. Brglez. Accelerated Transition Fault Simulation. *Proceedings of 26th Design Automation Conference*, pages 237–243, June 1987.

[77] N. A. Sherwani. *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Academic Publishers, Boston, 1993.

[78] M. Sivaraman and A. Strojwas. Primitive Path Delay Fault Identification. *Proceedings of The Tenth International Conference on VLSI Design*, pages 95–100, January 1997.

[79] G. L. Smith. Model for Delay Faults Based upon Paths. *Proceedings of IEEE International Test Conference*, pages 342–349, November 1985.

[80] U. Sparmann, D. Luxenburger, K.-T. Cheng, and S. M. Reddy. Fast Identification of Robust Dependent Path Delay Faults. *Proceedings of 32nd Design Automation Conference*, pages 119–125, June 1995.

[81] G. van Brakel, U. Glaser, H. G. Kerkhoff, and H. T. Vierhaus. Gate Delay Fault Test Generation for Non-Scan Circuits. *Proceedings of European Design and Test Conference*, pages 308–312, March 1995.

[82] H. T. Vierhaus, W. Meyer, and U. Glaser. CMOS Bridges And Resistive Transistor Faults: IDDQ versus Delay Effects. *Proceedings of IEEE International Test Conference*, pages 83–91, October 1993.

[83] R. L. Wadsack, J. M. Soden, R. K. Treece, M. R. Taylor, and C. F. Hawkins. CMOS IC Stuck-Open Fault Electrical Effects and Design Considerations. *Proceedings of IEEE International Test Conference*, pages 423–430, September 1989.

[84] K. D. Wagner and E. J. McCluskey. Effect of Supply Voltage on Circuit Propagation Delay and Test Application. *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 42–44, November 1985.

[85] J. A. Waicukauski et al. Fault Simulation for Structured VLSI. *IEEE Design & Test*, pages 20–32, December 1985.

[86] J. A. Waicukauski, E. Lindbloom, B. Rosen, and V. Iyengar. Transition Fault Simulation. *IEEE Design & Test*, pages 32–38, April 1987.

Index