Equivalence of Robust Delay-Fault and Single Stuck-Fault Test Generation

Alexander Saldanha  Robert K. Brayton  Alberto L. Sangiovanni-Vincentelli

University of California - Berkeley CA

Abstract
A link between the problems of robust delay-fault and single stuck-fault test generation is established. In particular, it is proved that all the robust test vector pairs for any path delay-fault in a network are directly obtained by all the test vectors for a corresponding single stuck-fault in a modified network. Since single stuck-fault test generation is a well solved problem, this result yields an efficient algorithm for robust delay-fault test generation.

1 Introduction
Due to the increasing need for reliability in high-performance VLSI circuits, static testing which only ensures logical correctness is becoming insufficient. In particular, defects and random variations in process parameters often cause delays to fall outside the specifications. Static testing may not detect such defects since there may be no impact on the logical (static) behavior of the circuit. However, since most circuits operate in a clocked environment, it is essential that the final logic value on the circuit be asserted before the clock arrives. This need for ensuring the temporal (dynamic) correctness of circuits, given delay specifications, has led to the development of delay-testing for chips.

The problem of delay-fault test generation has been studied in [1, 6, 2]. Lin and Reddy [1] and Schulz et al. [6] use multiple-valued calculi to generate robust delay-fault tests. The former uses a 5-value calculus, whereas the latter employs a 10-value calculus. Besides apparently increasing the complexity of the problem over stuck-fault test generation, both approaches require the development of new (implicit search) algorithms combined with modification of existing stuck-fault test generation programs. A recent approach of McGee et al. [2] is restricted to test generation for the strongest model of robust delay-fault testing, namely hazard-free single path propagation delay-fault testing. Although this approach is easily extended to the general model, there is a significant increase in size of the problem. In short, a satisfactory solution for delay-fault test generation (under the general model) has been lacking up to now. There are several proposals made for approximate delay-fault test generation in the literature (e.g., [4]). None guarantee completeness of the test generation process, and hence are excluded from consideration here.

The contribution of this paper is the establishment of a link between robust delay-fault test generation to single stuck-fault test generation. A robust test for any path delay-fault in a network is obtained by a single stuck-fault test generation on a slightly modified network. This allows the adapted application of mature and efficient static testing techniques [5] to delay-fault test generation.

2 Delay-faults
A few basic definitions are reviewed, before a formal definition of delay-fault testing conditions are provided. A controlling value for a gate \( f \) is the value at its input that determines the value at the output independent of the other inputs, and is denoted as \( A(f) \). For example, 0 is a controlling value for an AND gate. A non-controlling value for a gate \( f \) is the value at its input which is not a controlling value for the gate, and is denoted as \( I(f) \). For example, 1 is a non-controlling value for an AND gate. A simple gate is any one of AND, OR, NAND, NOR, and NOT. All the results described here apply only to simple gates. Only these gates have controlling vs. non-controlling values for each input. However, using the macro-expansion operator [2], any complex gate can be converted to an equivalent connection of simple gates.

Let \( P = \{ f_0, f_1, \ldots, f_m \} \) be a path. The inputs of \( f_i \) other than \( f_{i-1} \) are called side-inputs of \( f_i \) along \( P \) and denoted as \( S(f_i, P) \). A path that starts at a primary input and ends at a side-input of \( P \) is a side-path of \( P \). A path delay-fault is said to occur along path \( P \) if the delay along \( P \) falls outside its specified limits.

Definition 2.1 (RPDFT): A path \( P = \{ f_0, f_1, \ldots, f_m \} \) is said to be robust delay-fault testable for the rising (falling) transition at \( f_0 \) by the vector pair \( < v_1, v_2 > \) if at each node \( f_i \), \( f_i(v_1) \neq f_i(v_2) \) yields the desired transition being tested, and for each \( g_j \in S(f_i, P) \):

1. \( g_j(v_2) = I(f_i) \), and
2. if \( f_{i-1}(v_1) = I(f_i) \), then there is no transition on \( g_j \).

The vector \( v_2 \) is assumed to be applied after \( v_1 \), delayed by an amount greater than the delay of the circuit.

This general robust test may involve multiple path propagation with hazards. This definition does not specify the value on side inputs for \( v_1 \) when a path transition from a controlling value is being tested. This is because even in the presence of hazards on side inputs, a delay on this path will appear as a delay on the transition at the output of the gate. However, for the opposite transition (from non-controlling to controlling value) on the path input to the gate, no transitions are allowed on the side-inputs to this gate. Possible advantages of using delay-tests with hazards are a reduction in the size of the test set and a higher degree of fault coverage. This is because more paths can potentially be tested in the case when hazards are allowed compared to the hazard-free case: circuits are known with paths for which no robust test exists under the stronger delay-fault model, viz. hazard-free single-path propagation, while a general robust test does exist [3].

3 RPDFT and single stuck-fault test generation
In this section we present the main results of the paper, namely, a relationship between robust delay-fault and single stuck-fault test generation. For ease of exposition and proofs, we first consider only a particular circuit structure, functionally equivalent to any given circuit; this structure is termed a leaf-dag. A leaf-dag is a circuit composed of AND and OR gates with fanout and inverters only permitted at the inputs. An inverter is not allowed fanout. Every circuit (composed of simple gates,
The I-edge is said to be the first connection of a circuit. A falling-smooth-circuit is defined analogously, by considering replacing the I-edge of each circuit is shown at the top. A leaf-dag (shown in the middle) is a robust delay-fault test for the rising transition along P. Theorem 3.2 Let P be a path with input i in leaf-dag η. v is a test for the stuck-0 fault on the I-edge of P in the rising-smooth-network 77. for Pi if and only if < v, v >, where \( v_i = v_1, v_j = v_2 \), for \( i \neq j \), is a robust delay-fault test for the rising transition along P in η. Proof If part: Let < \( \theta, v \) > be a robust delay-fault test for the rising transition along P in η. Then by Theorem 3.1 it is a robust delay-fault test for the rising transition along P in 77.. The output of P is I when v is applied to the rising-smooth-circuit 77. P for Pi. Consider what happens in the presence of the test condition 1 and 2 of Definition 2.1, each side-input to OR gates along P is at a non-controlling value with no transitions on the I-edge of each path in P. This in turn implies that on application of the test < v, v > remains a test for P in 77.. Only if part: Only the I-edges of paths in P, are set to I in obtaining 77. from η. By condition 1 and 2 of Definition 2.1, since < v, v > is a robust delay-fault test for P in 77., each side-input to OR gates along P is at a steady non-controlling (0) value. This implies (since there are no inverters along the way) that the constant I asserted on the I-edge of each path in P does not propagate to the side-inputs of the OR gates. This in turn implies that on application of the test < v, v > remains a test for P in 77.. Figure 1: Example: Leaf-dag and rising-smooth network

A falling-smooth-circuit is defined analogously, by considering gates instead of OR gates, and 0 instead of I. Figure 1 illustrates the terms defined above. The initial circuit is shown at the top. A leaf-dag (shown in the middle) is derived by pushing all inverters to primary input leads. Note that gate 1 is first duplicated to gates 11 and 12; gate 12 becomes an AND gate when the inverter is pushed from the output to its inputs (using DeMorgan’s law). Similarly, the OR gate 5 becomes an AND gate. The rising-smooth network for the bold path P is shown at the bottom of the figure. (Note the I-edge of P is the connection from the inverter to the AND gate S.) This is obtained from the leaf-dag by replacing those I-edges associated with \( f \), of side-paths to all the OR gates along P by I. Thus, the I-edges of the paths starting at \( f \) through 11, 12, and 2 are replaced.

Theorem 3.1 Let P be a path in leaf-dag η. < v1, v2 > is a robust delay-fault test for the rising transition along P in the rising-smooth-circuit 77. P for Pi if and only if < v1, v2 > is a robust delay-fault test for the rising transition along P in η. Proof Let i be the input of P. Let \( P_i \) denote the side-paths to OR gates along P in η, such that the I-edge of each Q \( \in P_i \) is associated with input i.

Paper 10.3
/* If a path has transition i at input i */
compute.rpdtf.test(n, P, t1) {
    Make P fanout-free.
    For each node i, Qor = OR gates in P with rising transition,
    Qand = AND gates in P with falling transition.
    Q = transitive fanout of i in reverse topological order.
    compute.path.parity(P, Q, Qor, Qand, i, parity).
    For each (n ∈ Q) {
        If n has any fanout f with parity(f) = {0}
            n’ = duplicate(n).
            For each fanout f or n with parity(f) == {1}
                Replace connection n to f by n’ to f.
            parity(n) = {0}.
            For each gate g ∈ Q with input i
                If (parity(g) = {1})
                    Replace connection n to f by n’ to f.
            parity(g) = {0}.
        Else if (parity(g) = {0})
            Set g to 0.
        If (g is rising transition at t)
            v = test for stuck-0 fault on input of P.
        Else
            v = test for stuck-1 fault on input of P.
        RPDTF test for t is < v, w >.
    }
}

Figure 2: Robust delay-fault test generation

A similar equivalence between RPDTF testing of the falling transition and single stuck-fault testing exists. The results stated apply to leaf-dags. Since most large Boolean networks cannot be converted to leaf-dags without an exponential (worst-case) increase in circuit size, an algorithm is presented in the next section that avoids this problem. Briefly summarized, the algorithm converts a Boolean network to another using a sequence of duplications and transfer of connections, on which stuck-fault test generation is performed. As shown later, the size of this resulting network is at most four times the size of the initial network; in actual practice, the increase is a few gates.

4 RPDTF test generation

In the previous section robust delay-fault testing is shown to be equivalent to single stuck-fault testing on leaf-dag networks. Here we extend the equivalence to arbitrary Boolean networks and develop an efficient process for generating the delay-fault tests.

Let P be a path in a Boolean network η for which a robust delay-fault test is required for transition i at the input i of P. η is composed of AND, OR, and NOT gates, where all inversions are explicitly represented. For ease of exposition, assume P is fanout-free; this is easily achieved by duplicating each gate along P at most once. For each gate along P, determine whether the transition along P is a rising or falling transition. Let Qor denote the OR gates along P that have a rising transition along P, and Qand the AND gates along P that have a falling transition.

Perform a reverse topological traversal of the gates Q in the transitive fanout of i, to determine the parity of all gates along side-paths to P. The algorithm for this is shown in Figure 3. A gate may have no parity, 0, 1, or both parities. For example,
Sketch of proof: Since each Boolean network has a unique equivalent leaf-dag circuit that is derived from it, let $\eta_i$ and $\eta_j$ denote the leaf-dag’s derived from $\eta$ and $\eta_j$, respectively. Assume the rising transition on input $i$ (of $P$) in $\eta$ corresponds to a rising transition along $P$ in $\eta_i$. Then it can be shown that the leaf-dag $\eta_i$ is (structurally and logically) identical to the rising-smooth circuit for $P$ in $\eta_i$. The result of the theorem follows from Theorem 3.2. Similar reasoning applies to the case when a rising transition on $i$ in $\eta$ corresponds to a falling transition along $P$ in $\eta_i$. Here the leaf-dag $\eta_i$ is identical to the falling-smooth circuit for $P$ in $\eta_i$.

An analogous statement holds for robust delay-fault test generation of falling transitions.

5 RPDFT test generation results

Table 1 gives the results of robust delay-fault test generation using equivalent single stuck-fault testing for some of the larger optimized circuits from the MCNC and ISCAS benchmark suite. All CPU times are on a DEC 5000. Circuits with very low testability have been excluded from the table, since they do not help in discriminating between the use of single-path or multiple-path propagation delay-fault tests. The first column is the name of the example, and the second is the number of path delay-faults for which test generation is attempted. The third and fourth column give the testability and CPU time (including path-tracing and Boolean satisfiability) using the test generation algorithm for hazard-free single-path propagation delay-faults [2]. The fifth column gives the testability using the general RPDFT model obtained using the approach described in this paper. The sixth column lists the total CPU time consumed by this approach, including path-tracing, network manipulations (duplications and connections movement) inside the SIS system, and Boolean satisfiability that is invoked to check single stuck-fault testability. The actual number of satisfiability calls made by the single-path approach is half the number made by the multiple-path approach. This is because both transitions are tested by the same vector pair along a path in the former case, if a test exists. The last column shows the CPU time spent on stuck-fault test generation after the circuit $\eta_j$ is constructed.

The following observations can be made from the table. Multiple-path propagation tests provide up to 10% additional coverage over single-path propagation tests. Further test set compaction has not been explored in this experiment. Unlike all other reported approaches for delay-fault test generation, notably [1, 6], as with [2], there are no aborted faults. Overall, the new proposal provides a very efficient and rugged technique for delay-fault test generation.

6 Conclusions

We have established a link between the problems of robust delay-fault and single stuck-fault test generation. It is proved that the robust test vector pairs for any path delay-fault in a network can be generated by all the test vectors for a single stuck-fault in a modified network. Since single stuck-fault test generation is a well-solved problem, this yields an efficient algorithm for robust delay-fault test generation. Experimental results demonstrate the efficiency of the proposed technique. Future work includes delay-fault test set compaction and synthesis for delay-fault testability in large Boolean circuits.

References


