# Fault Nodes in Implication Graph for Equivalence/Dominance Collapsing, and Identifying Untestable and Independent Faults

Rajamani Sethuram
rms@qualcomm.com
Design Automation Group,
Qualcomm, San Diego, CA-92122

Michael L. Bushnell
bushnell@caip.rutgers.edu
Dept. of ECE,
Rutgers University, Piscataway, NJ-08854

Vishwani D. Agrawal
vagrawal@eng.auburn.edu
Dept. of ECE,
Auburn University, Auburn, AL-36849

## Abstract

*This paper presents a new* fault node *for implication graph that represents the Boolean detectability status of a fault in the circuit. An implication graph with fault nodes is termed* functional fault graph *(FFG) because such a graph stores both the functional information and the fault information of the circuit. By computing the transitive closure and graph condensation of the FFG of a circuit, we show that we can collapse faults, and identify untestable faults and independent fault pairs in the circuit. Compared to prior fault independent-based approaches for fault collapsing, our technique gives the best result by reducing the fault-set size by 66%. Additional advantages of our technique compared to previous techniques are: a) It can also identify independent fault pairs in the circuit, and b) It can be extended for other fault models and has a variety of applications. Our experiment with c7552 also found more than 268K independent fault pairs. This work also introduces the first fault-independent polynomial-time approach for identifying untestable transition delay faults.*

## 1 Introduction

Fault $f_1$ dominates another fault $f_2$ if every test of $f_2$ also detects $f_1$. When two faults dominate each other, they are called equivalent [6]. Fault collapsing is the process of generating a reduced fault set using the equivalence and the dominance relationships and is classified as structural or functional. Structural collapsing uses only the topology of the circuit. For example, a *stuck-at 0* (sa0) fault at the output of an AND gate (see Figure 1(a)) is equivalent to all of the input sa0 faults. Functional fault collapsing uses the circuit functional information to establish equivalence and dominance relationships. Figure 1(b) represents two faults $l_0$ ($l$ sa0) and $m_0$ that are functionally equivalent. This is because $a$=1, $b$=1 is the only vector that detects $l_0$ and $m_0$. Note that here and in the rest of the paper, we use the subscript fault notation. Thus, $m_0$ means $m\ sa0$.

There are several applications of fault collapsing. A collapsed fault set reduces the fault simulation time during test generation and helps the *automatic test pattern generation* (ATPG) tool to generate a smaller test set for achieving the

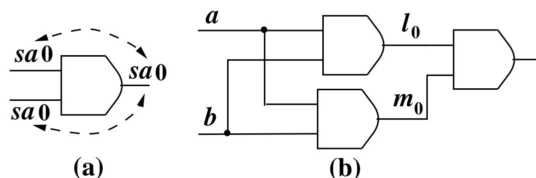


Figure 1: Example of Equivalent Faults: a) Structural and b) Functional Equivalence

desired *fault coverage*. Another application of fault collapsing is fault diagnosis (defined as the process of isolating the source of failure in a defective chip). The fault simulation time during diagnosis can be reduced by using a collapsed fault set, thereby enabling large volume diagnosis.

Lioy [1] showed that the algorithmic complexity for identifying functionally equivalent faults is similar to that of ATPG. He presented a fault collapsing algorithm that explicitly identifies the D-chains and necessary assignment conditions for all faults in the circuit. He, then, uses an ATPG analysis for each fault in the fault set for identifying functional equivalence. Amyeen *et al.* [5] use a similar approach along with the dominator theory for identifying more functionally equivalent fault pairs. Recently, Vimjam and Hsiao [15] enhance Lioy's work using their generalized dominance theorem. Sandireddy and Agrawal [11] presented a novel diagnostic fault equivalence and dominance technique. Adapa *et al.* [2] presented a technique to speed-up diagnosis via dominance relations between sets of faults using function-based techniques. Due to the high memory and time complexity, Sandireddy's and Adapa's approach is applicable only for small circuits. All of these techniques are fault oriented approaches, i.e., they consider a fault-pair at a time and perform ATPG-like analysis to establish fault equivalence and dominance relationships.

Boolean fault variables were first used by Poage [9]. Prasad *et al.* [10] presented the first fault-independent, polynomial time approach that uses a *dominance graph* and a transitive closure algorithm to obtain equivalence and dominance information. A dominance graph $D(F,\, E)$ is a directed graph where the nodes $f \in F$ represent faults and the

directed edge $e(f_i, f_j) \in E$ means fault $f_j$ dominates $f_i$. Figure 2 represents the dominance graph for the six stuck-at faults of an AND gate. This is the first graph theoretic

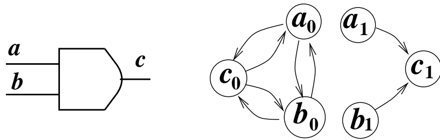


Figure 2: 2-input AND Gate and its Dominance Graph

approach that relies on the transitive closure algorithm for fault collapsing. In the sequel paper by Agrawal *et al.* [3], they describe how functional fault equivalence in standard cells and transitive closure can be used for hierarchical fault collapsing.

In this paper, we also present a graph theoretic, fault-independent, and polynomial-time technique for functional fault collapsing. Hence, this work is a direct extension of Prasad's and Agrawal's earlier work described above. We introdue *fault nodes* in an implication graph to build a *functional fault graph* (FFG) that stores the functional and fault information of the circuit. It can be used to collapse faults, and identify both untestable faults and independent fault pairs. Two faults $f_1$ and $f_2$ are independent if every vector that detects $f_1$ does not detect $f_2$.

The rest of the paper is organized as follows. In Section 2, we introduce the proposed FFG. In Section 3, we describe how a FFG can be used to collapse faults, identify untestable faults, and independent fault pairs. In Section 4, we present the overall algorithm to build the FFG and collapse faults. In Section 5, we describe how our technique can also be extended for other fault models. Section 6 presents the result of the proposed technique and shows that our technique can reduce the fault set by additional 19.9% compared to Prasad *et al.*'s [10] work. Finally, we conclude in Section 7.

## 2 The New Functional Fault Graph

In this section, we will describe how to build the proposed *functional fault graph* (FFG). Then we will analyze its memory complextiy. The FFG for a circuit is constructed by building the FFG for all of the individual gates and merging them together. Hence, we will now explain the procedure to construct a FFG for a simple two-input AND gate (see Figure 3). This procedure can be easily extended to build the FFG for other primitive gates (OR, NAND, NOR, NOT, BUF etc) as well. As mentioned earlier, the FFG stores both the functional and the fault information of the circuit.

**Functional Information.** It is represented using the controllability ($a$, $\overline{a}$, $b$, $\overline{b}$, $c$, $\overline{c}$) and the observability ($O_a$, $\overline{O_a}$, $O_b$, $\overline{O_b}$, $O_c$, $\overline{O_c}$) nodes. Node $a(\overline{a})$ tells whether line $a$ is controllable to Boolean 1(0) and the node labeled $O_a(\overline{O_a})$

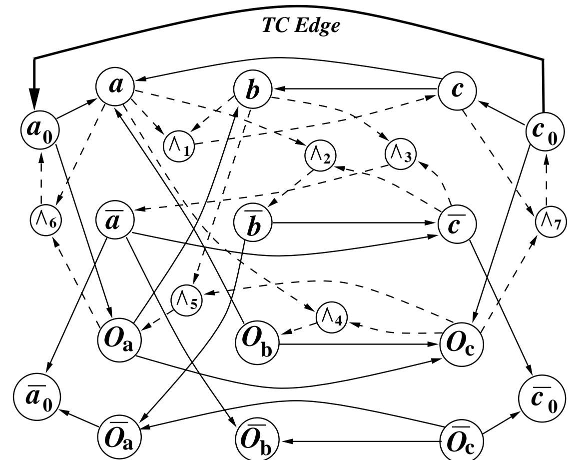


Figure 3: FFG of a 2-input AND Gate

tells whether line $a$ can (cannot) be observed at any *primary output* (PO). Since this is similar to an *implication graph* (IG) [14], the FFG is an extension of the IG. In Figure 3, the edge $c \rightarrow a$ ($b$) means if $c$=1, then $a$ ($b$)=1. The edge $\overline{a}(\overline{b}) \rightarrow \overline{c}$ represents its contrapositive relation, i.e., if $a(b)$=0 then $c$=0. The nodes labeled $\Lambda_i$, $i$=1...7, are partial nodes and the dotted edge is called a partial edge. The partial edges connecting $a\Lambda_1 b \rightarrow c$ mean if $a$=1 and $b$=1 then $c$=1. The partial nodes labeled $\Lambda_2$ and $\Lambda_3$ represent the contrapositive relationships of $\Lambda_1$. The edge $O_a \rightarrow b$ means if $a$ is observable then $b$=1 and the edge $O_a \rightarrow O_c$ means if $a$ is observable then $c$ is also observable. The node $\Lambda_4$ represents the partial implication that if $a$=1 and $c$ is observable then $b$ is also observable. The contrapositive of $\Lambda_4$ is not shown here to keep the figure legible.

However, note that using the observability nodes to represent the functional information makes the FFG graph incomplete. To understand this, consider a stem signal $s$ that fans out to signals $p$ and $q$. It is not possible to establish any relationship among $O_s$, $O_p$, and $O_q$ [8]. This limitation makes the FFG incomplete in its representation of the functional information. In Section 3, we will present a theorem that will make the proposed FFG less incomplete.

**Fault Information.** It is represented using the fault nodes. In this example, we have modeled only two faults ($a_0$ and $c_0$) using the nodes labeled $a_0$, $\overline{a_0}$, $c_0$, $\overline{c_0}$, $\Lambda_6$, and $\Lambda_7$. Here $a_0$ is true if $a$ $sa0$ is detectable. The edge $a_0 \rightarrow a$ means to detect the fault $a_0$, $a$ should be 1 and the edge $a_0 \rightarrow O_a$ means $a$ should be observable to detect $a_0$. Note that we have also added the contrapositive edges $\overline{a} \rightarrow \overline{a_0}$ (if $a$=0, then $a_0$ is not detectable) and $\overline{O_a} \rightarrow \overline{a_0}$ (if $a$ is unobservable then $a_0$ is not detectable). Also the partial node $a\Lambda O_a \rightarrow a_0$ means if $a$=1 and is observable then the fault $a_0$ is detectable. The contrapositive of the nodes $\Lambda_6$ and $\Lambda_7$ is not shown here. The thick edge labeled *TC Edge* will be described in Section 3.

**Memory Complexity.** For any directed graph $G(V, E)$ the total memory complexity is $O(|V| + |E|)$. We will now analyze the total memory required to store a FFG for a circuit with $n$ stem signals, $m$ fanout signals and $k$ faults. To represent a signal $s$, we need two controllabilty nodes and two observability nodes leading to $4n$ nodes. Two additional nodes are required to represent the observability status of each of the fanout signals because the observability status of a stem and its branches are different. Hence, $2m$ nodes are required to represent observability of the fanout signals and $4n$ nodes are required to represent the controllability and the observability ofthe stem signals. $2 \times k$ nodes are required to represent all of the fault nodes. Also, let $l$ be the average number of gate fanins of the circuit. Then, the total number of partial nodes required to represent the $n$ gates is $n\ (2 \times l + 2)$ and $k$ faults is $3 \times k$ ($k$ forward ANDing nodes and $2k$ nodes to represent their contrapositive). Hence the total number of nodes in the graph is $n \times (2l + 6) + 5k + 2m$. We will now count the total number of edges in the graph. Every $l$-input gate will have $l$ direct edges and $(l{+}1)\times(l{+}1)$ partial edges (there will be $l{+}1$ partial nodes each with $l{+}1$ partial edges) leading to $nl + n(l + 1)^2$. There will be at most $m$ additional direct edges due to the $m$ fanout signals. Four direct edges and nine partial edges (three partial nodes each having three partial edges) are required to represent each fault. Hence, the total number of edges is $n \times (l^2 + 3l + 1) + 13k + m$. Therefore, the total memory required to store the graph is $n \times (l^2 + 5l + 7) + 18k + 3m \approx O(n)$ because $l$ is a constant and both $m$ and $k$ are proportional to $n$.

## 3 Fault Collapsing Using FFG

The first step in our approach comprises building the FFG. This is done by determining the total number of nodes $|V|$ (see Section 2) and allocating memory for these nodes. We then add edges into the graph to represent the functional information corresponding to each gate in the design. Then, we add edges to model each fault in the circuit. The next step is to compute the graph condensation and the transitive closure of the FFG, which will now be explained in detail.

*Graph condensation* (GC) is the process of computing the condensed graph $G_C$ from the original graph $G$. A *condensed graph* is a graph $G_C(V_C, E_C)$ based on the graph $G(V, E)$ where each vertex in $V_C$ corresponds to a *strongly connected component* (SCC) in $G$ and edge $(u, v)$ is in $E_C$ if and only if there exists an edge in $E$ connecting any of the vertices in the component of $u$ to any of the vertices in the component of $v$. Here an SCC means a subgraph $G_{SCC}$ of a given directed graph $G$, such that every node in $G_{SCC}$ is reachable from every other node in $G_{SCC}$.

*Transitive closure* (TC) is a procedure that adds an edge between the node pair $v_1$, $v_2$ if the original graph has a directed path connecting nodes $v_1$ and $v_2$. The edge added is called a *TC edge*. Note that for all of the partial nodes $v_P$ in FFG, the $TC$ procedure adds edges connecting each of the *common ancestors* of $v_P$ to the children of $v_P$. To understand this let us suppose that $p_1$, $p_2$ are the parent nodes of $v_P$ and $c$, its child. If $v$ is the common ancestor of $v_P$, then it means that $v \rightarrow p_1$ and $v \rightarrow p_2$. But, $p_1 \wedge p_2 \rightarrow c$. Hence, $v \rightarrow c$.

**Example 1.** Figure 4 illustrates graph condensation and transitive closure. Nodes $b$, $c$, and $d$ belong to a SCC because every node in this set is reachable from every other node in that set. This SCC corresponds to node $C_1$ in the condensed graph shown in Figure 4(b). Similarly, $a$ and $e$

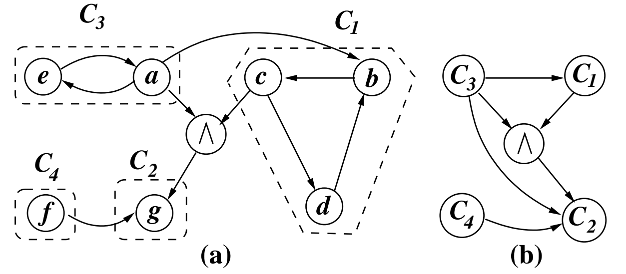


Figure 4: Graph Condensation Example: a) The Graph before Condensation and b) The Graph after Condensation

are a SCC that corresponds to $C_3$. The SCCs corresponding to $C_2$ and $C_4$ contain only one node, i.e., $f$ and $g$, respectively. The edge connecting $C_3$ and $C_2$ in the condensed graph is a TC edge because $C_3$ is a common ancestor of $C_2$. Detailed descriptions of the algorithm for these two operations are given elsewhere [13, 14].

After performing GC and TC, we traverse the edges of the condensed graph $G_c$ to identify the following relationships:

1. *Equivalent faults.* If fault nodes, $F = \{f_1, f_2, \ldots f_k\}$ belong to a SCC then they are equivalent faults. Since $F$ is a SCC, every fault node $f_i \in F$ is reachable from every other fault node $f_j \in F$. This means that if $f_i$ is detectable then so is $f_j$ and vice-versa. Hence, faults in $F$ are all equivalent.

2. *Dominant fault.* If $f$ is the topologically minimal fault node then $f$ is dominated by all of the faults reachable from $f$. Hence, if $f_1 \rightarrow f_2$, $f_2 \rightarrow f_3, \ldots, f_{k-1} \rightarrow f_k$ then $f_1$ is dominated by the faults $f_2, \ldots, f_k$.

3. *Untestable fault.* If there is an implication $f \rightarrow \overline{f}$, then it means that $f$ is untestable.

4. *Independent fault.* If $f_1 \rightarrow \overline{f_2}$ then it means that the faults $f_1$ and $f_2$ are independent pairs. This is because if $f_1$ is detected then $f_2$ cannot be detected and vice-versa.

Thus, by simply going through the edges in the condensed graph $G_c$, we collapse faults. For example, TC of the graph

shown in Figure 3 found $c_0 \rightarrow a_0$. This is because $c_0 \rightarrow a$ and $c_0 \rightarrow O_a$ (as $c_0$ is the common ancestor of $\Lambda_5$). Hence, $c_0$ is the common ancestor of $\Lambda_6$ meaning $c_0 \rightarrow a_0$. This means $a_0$ dominates $c_0$ (a graph theoretic proof for structural fault collapsing!). Similarly, we can also see that $a_0 \rightarrow c_0$ but this is not shown in the figure. We will illustrate more dominance examples using an example netlist (see Figure 5). But before that, it is important to note that the proposed FFG is incomplete because it does not contain any edge connecting the stem observability and its branch observabilities. This is a major deficiency because we cannot establish any equivalence/dominance relationship between a fault at a signal $s$ and its recovergent fanout. This is because the node $O_s$ is not connected to any other observability node in its fanout cone. We will now partially fix this deficiency using a theorem presented below that enhance the FFG by adding additional edges connecting observability nodes.

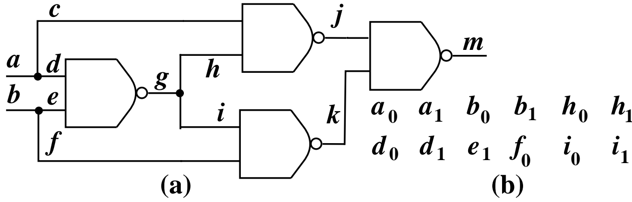


Figure 5: Example to Illustrate Theorem 1: a) The Circuit, and b) The Faults

**Theorem 3.1** *If $p$ and $q$ are two signals such that $p$ is the dominator of $q$ then $O_q \rightarrow O_p$.*

**Example 2.** We illustrate this theorem using an example shown in Figure 5. Since $m$ is the reconvergent fanout of $g$, we add the edge $O_g \rightarrow O_m$ into our $FFG$ graph. Due to this, the faults $i_0$, $i_1$, $h_0$, and $h_1$ can be dropped reducing the collapsed fault set size from 12 faults to eight faults. This is because it identifies the following four relationships:
**a)** $i_0$ dominates $d_1$
**Proof.**

1. $d_1 \rightarrow O_d \rightarrow O_g \rightarrow O_m$ and $d_1 \rightarrow O_d \rightarrow b$
2. $d_1 \rightarrow \overline{a} \rightarrow j$ and $d_1 \rightarrow \overline{a} \rightarrow i$
3. $O_m \Lambda j \rightarrow O_k$ and $O_k \Lambda b \rightarrow O_i$. Hence, $d_1 \rightarrow O_i$
4. $d_1 \rightarrow i$ and $d_1 \rightarrow O_i$. Hence, $d_1 \rightarrow i_0$.

The following three relationships can be similarly proved.
**b)** $i_1$ dominates $d_0$
**c)** $h_0$ dominates $e_1$
**d)** $h_1$ dominates $d_0$.

Our approach is still incomplete because our approach uses a polynomial-time algorithm but the complexity of functional fault collapsing is exponential. Hence, currently our approach cannot establish $a_0$ dominates $d_0$ (see Figure 5) that an ATPG-based collapsing approach could identify. Note, however, that if we add the edge $O_m \rightarrow O_a$ and work out the transitive closure, we see that we can conclude $a_0$ dominates $d_0$. This means that the current deficiency of the proposed FFG can be improved but we will consider such enhancements in our future work.

# 4 Optimizations and the Overall Algorithm

The size of the FFG after computing the TC can become very large. Hence, in this section, we will present two optimizations: 1) Start with a reduced fault set, and 2) Cone-based decomposition. These two techniques help reduce the peak memory consumption of our technique. Then we will present the overall algorithm.

**1. Start with a reduced fault set.** As mentioned earlier, if there are $k$ faults in a circuit then a total of $5k$ nodes and $13k$ edges are required to model these faults in the FFG. Hence, the total memory required to model these faults can be reduced by reducing the value of $k$. So, we start with a structurally collapsed fault set, $F_m$. To obtain this, we start with an uncollapsed fault set $F_0$, which comprises all the stuck-at 0 and stuck-at 1 faults in the circuit. Thus, the size of $F_0$ is $2 \times n$, where $n$ is the number of signal lines in the circuit. Starting from the *primary inputs* (PIs) or *pseudo-primary inputs* (PPIs), we consider a fault $f \in F_0$ and remove all the faults $f_c$ from the set $F_0$ that are structurally equivalent to $f$ or dominated by $f$ to get a reduced fault set $F_1$. We continue reducing the fault set $F_j$, where $j$=0, 1, 2, ..., $m$ ($m$ is the last iteration count) until we are unable to reduce it any further.

**2. Cone-based decomposition.** Building the FFG for the entire circuit, at once, can consume enormous memory. Hence, we partition the circuit into a large number of *logic cones*. A logic cone (see Figure 6) comprises all of the signals in the fanin cone of a *primary output* (PO) or a *pseudo-primary output* (PPO). We build the FFG of one logic cone

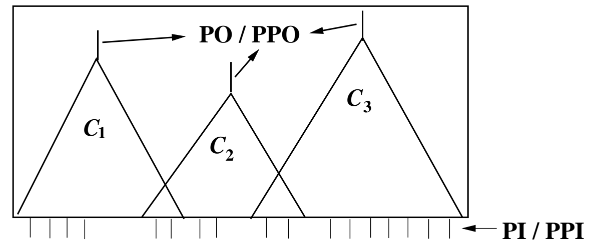


Figure 6: Example of Circuit with Three Logic Cones $C_1$, $C_2$, and $C_3$

and collapse all of the faults in that cone. We delete the memory allocated for the current FFG before building the FFG for the next logic cone. Since the size of a logic cone is much smaller than the size of the entire circuit, the peak run time memory is drastically reduced.

Figure 7 represents the overall algorithm. We use a breadth-first traversal-based algorithm to partition the

circuit into logic cones. In Step 7, the function $traverseCondensedGraph()$ traverses each edge in the final condensed graph to collapse faults, and identify independent fault pairs and untestable faults (see Section 3).

1. *Partition the circuit into logic cones*
2. *foreach cone $C_i$ in the circuit* {
3. $generateReducedFaultSet$ ()
4. $Build$ *FFG for this cone*
5. *Add edges due to Theorem 3.1*
6. $Compute$ *GC and TC*
7. $traverseCondensedGraph$ ()
8. }

Figure 7: The Overall Algorithm

# 5 FFG for Other Fault Models

One advantage of FFG is that it can be easily extended for other fault models. To illustrate this, we consider the *skewed load transition delay* (SLT) [12] fault model for full scan sequential circuits that are used to test the at-speed behavior of a device. The SLT models two types of fault at every signal line in the circuit. They are called *slow-to-rise* and *slow-to-fall* faults. A vector pair $< v_1, v_2 >$ detects a slow-to-rise (fall) fault at $l$ if it meets the following requirements: a) $v_1$ initializes $l$ to 0(1), b) $v_2$ detects $l_1$ ($l_0$), and c) $v_2$ is the logical right shift of $v_1$. In an actual chip, the vector $v_1$ is loaded by pulsing $n$ shift clock cycles where $n$ is the length of the scan chain. This initializes all of the internal signals in the circuit. Another shift pulse generates the vector $v_2$. The response is captured at-speed and is shifted out of scan chain using another $n$ shift clock cycles.

In this section, we will describe how FFGs can model SLT faults of a given circuit. This is the first work that presents a polynomial time algorithm to identify untestable SLT faults. The following three enhancements are required:

**1. Representing two time frames.** Since two time-frames are required to detect a SLT fault, we need a scheme to represent information pertaining to multiple time frames. For this, we use the idea of concurrent graph that has been used by Sethuram *et al.* for Sequential ATPG [13] and false path identification [14]. Concurrent graphs represent multiple time frames simultaneously using edges that are annotated by a $k$-bit flag $B[1 \ldots k]$. The bit $B[i]$ of an edge $e$ tells whether $e$ exists at time frame $i$ or not. Figure 8(a) shows a concurrent graph where edges are annotated by 2-bit flags ($k$=2). The edge $a \stackrel{01}{\rightarrow} b$ means if $a$ is true in the second time frame then $b$ is also true in the second time frame.

**2. Representing the scan chain.** To represent scan chains, we should first understand *time frame* (TF) edges [14, 16]. A TF edge represents implication relationships between nodes belonging to different time frames. They are represented by directed edges that are annotated by $+N$ or a $-N$ flag, where $N$ is a whole number. For example, an edge represented by $A \stackrel{+N}{\rightarrow} B$ means that if $A$ is true in the current time frame then $B$ is true after $N$ time frames. The TF edge was first used by Zhao *et al.* [16] for identifying untestable stuck-at faults in non-scan sequential circuits and, later, by Sethuram [14] for sequential ATPG. They used it to represent the relationship between the $D$ input and the $Q$ output of a non-scan flop by adding the edge $D \stackrel{+1}{\rightarrow} Q$ and its contrapositive $\overline{Q} \stackrel{-1}{\rightarrow} \overline{D}$ (meaning if $Q$=0 in the current time frame then $D$=0 in the previous time frame).

In this work, we use TF edges to represent the scan chain information in the proposed FFG. Let us assume that there are $k$ *scan flip-flops* (SFFs) labeled $F_1$, $F_2$, $\ldots$, $F_k$ and let $S_1$, $S_2$, $\ldots$, $S_k$ represent their outputs. Consider two *scan flip-flops* (SFFs) $F_i$ and $F_{i+1}$ where $F_i$ is the left neighbor of $F_{i+1}$, i.e., one shift clock will shift the value of $F_i$ into $F_{i+1}$. To represent controllability relationships between $S_i$ and $S_{i+1}$, we use TF edges that represent relationships between nodes across different time frames as shown in Figure 8(b). Here $S_1 \stackrel{+1}{\rightarrow} S_2$ means if $S_1$=1 in the current time frame then $S_2$=1 in the next time frame. To model the entire scan chain, we add the edges $S_1 \stackrel{+1}{\rightarrow} S_2$, $S_2 \stackrel{+1}{\rightarrow} S_3$, $\ldots$, $S_{k-1} \stackrel{+1}{\rightarrow} S_k$ to represent the 1-controllability of all of the SFFs. Similarly, we also add edges $\overline{S_1} \stackrel{+1}{\rightarrow} \overline{S_2}$, $\overline{S_2} \stackrel{+1}{\rightarrow} \overline{S_3}$, $\ldots$, $\overline{S_{k-1}} \stackrel{+1}{\rightarrow} \overline{S_k}$ to represent the 0-controllability. Finally, we also add TF edges representing the contrapositive relationships of all of these edges.

**3. Modeling SLT faults.** We use fault nodes labeled $l_{SR}$ and $\overline{l_{SR}}$ in Figure 8(c) to model faults. Here, $l_{SR}$ node represents a slow-to-rise fault at $l$. The edge $l_{SR} \stackrel{-1}{\rightarrow} \overline{l}$ means $l_{SR}$ is detectable if $l$=0 in the previous time frame, $l_{SR} \stackrel{01}{\rightarrow} l$ and $l_{SR} \stackrel{01}{\rightarrow} O_l$ means $l_{SR}$ is detectable in second time frame if $l$=1 and $l$ is observable in the second time frame, respectively. The other three edges in Figure 8(c) represent their respective contrapositive edges.

This completes the description of how a FFG is built for modeling SLT faults. A graph condensation-based TC algorithm can infer additional edges in the proposed FFG that can be used to identify fault dominance, untestable faults, and independent fault pairs. Due to lack of space, we do not describe the algorithms to compute transitive closure and graph condensation of concurrent graphs with TF edges but they are described elsewhere [13].

# 6 Results and Analysis

The proposed FFG, Theorem 3.1 and its enhancement for SLT faults were implemented in C and experiments with ISCAS '85, ISCAS '89, and ITC '99 benchmark circuits were conducted on a Sun Sparc machine running the SunOS operating system.

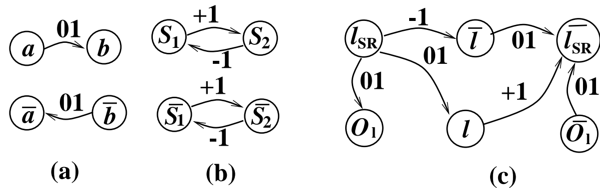


Figure 8: FFG for Skewed-load Transition Delay: a) Concurrent Edge, b) Time Frame Edge, and c) Fault Node

## 6.1 Fault Collapsing

Table 1 shows the results for the fault collapsing experiments. The column labeled *Total Flts.* represents the total number of stuck-at faults in that circuit. The column labeled *Struc. Flts.* represents the fault set size after applying structural fault collapsing. The next column labeled *Prasad et al.* represent the faut list size of Prasad *et al.*'s work. The last column labeled $Ours$ represent the result of our work. Here, *Flts* represents the fault set size and *Time* represents the CPU time (in seconds) of our work. Compared to Prasad *et al.*'s work, our technique reduced the fault set size by additional 19.9%. This is because Prasad *et al.*'s dominance graph had several missing implications that were captured by Theorem 3.1 and the partial nodes of the FFG. However, our technique could not identify any additional dominance relation in $c6288$. This is because $c6288$ has a large number of fanout stem signal and Theorem 3.1 could not detect additional implications. The strength of our technique is that: a) it uses a fault-independent approach and can be used as a pre-process for any fault-oriented ATPG-based fault collapsing algorithm, b) it can be easily extended for other fault models, and c) it has a variety of applications.

Table 1: Dominance Fault Collapsing Results

| CKT | $Total$ | $Struc.$ | *Prasad* | $Ours$ | |
|---|---|---|---|---|---|
| | $Flts.$ | $Flts.$ | *et al.* | $Flts.$ | *Time* |
| $c1355$ | 2710 | 1234 | 1210 | 808 | 46 |
| $c1908$ | 3816 | 1568 | 1566 | 753 | 14 |
| $c2670$ | 5340 | 2324 | 2317 | 1853 | 110 |
| $c3540$ | 7080 | 2882 | 2786 | 2092 | 831 |
| $c5315$ | 10630 | 4530 | 4492 | 3443 | 72 |
| $c6288$ | 12576 | 5840 | 5824 | 5824 | 4 |
| $c7552$ | 15104 | 6163 | 6132 | 4707 | 232 |

## 6.2 Independent Fault Pairs

Our experiments indicate that there are large numbers of independent fault pairs in a circuit. We used the reduced fault set (see Section 4) of each circuit, $C$, to compute the total number of independent fault pairs in $C$. Figure 9 shows a detailed distribution of all of the independent fault pairs in $c432$. Here, the $x$-axis (labeled *Logic Cone*) represents the seven logic cones (see Figure 6) of $c432$ each corresponding to a PO. The primary $y$-axis (labeled *# Ind. Flt. Pairs*) represents the number of independent fault pairs found in one cone. The secondary $y$-axis (labeled *Total Faults*) represents the total number of faults considered in that cone. Also, our experiment found 268K independent fault pairs in $c7552$. This number will be in the order of a million for large industrial circuits. Hence, a practical way to use our tool is to extract small portions of the netlist and conduct independent fault pair analysis on it.

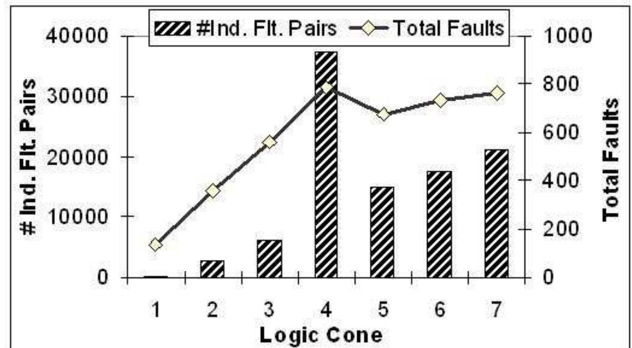


Figure 9: Independent Fault Pairs in $c432$

## 6.3 Skewed-load Untestable Faults

We used the full scan sequential circuits of ISCAS '89 and ITC '99 benchmark circuits with a single scan chain to identify SLT untestable faults in them. The *scan chain order* (SCO) in these circuits is the same as the order in which the flip-flops appear in the verilog description of the netlist. Also, in our experiment we only consider logic that is $K$ levels deep from the scan chain. A practical application of this experiment is to study the impact of SCO on SLT testability of the circuit (see Mao *et al.'s* [7] work to understand why SCO affects the SLT testability of a circuit). Table 2 shows the total number of untestable SLT faults for the cases $K$=5, $K$=10, and $K$=20. The column labeled $Unt.$ represents the total number of untestable SLT faults detected by our enhanced FFG and $Time$ represents the total CPU time in seconds. The large number of untestable faults for $s38584$ indicates that the design, currently, has an inferior SCO.

## 7 Conclusion

In this paper, we described a new *functional fault graph* (FFG) that can store both the functional information and the fault information of the circuit. We showed that such

Table 2: Untestable SLT Faults

| CKT | $K$=5 | | $K$=10 | | $K$=20 | |
|---|---|---|---|---|---|---|
| | $Unt.$ | $Time$ | $Unt.$ | $Time$ | $Unt.$ | $Time$ |
| $s5378$ | 34 | $< 1$ | 155 | $< 1$ | 315 | 10 |
| $s9234$ | 45 | $< 1$ | 47 | 2 | 89 | 1 |
| $s13207$ | 159 | $< 1$ | 307 | 14 | 520 | 29 |
| $s15850$ | 87 | $< 1$ | 232 | $< 1$ | 276 | 2 |
| $s35932$ | 0 | 1 | 0 | 1 | 0 | 2 |
| $s38417$ | 52 | 1 | 430 | 3 | 686 | 26 |
| $s38584$ | 205 | $< 1$ | 362 | 3 | 2385 | 35 |
| $b15s$ | 15 | 1 | 26 | 51 | 43 | 123 |
| $b17s$ | 115 | 5 | 278 | 196 | 298 | 852 |
| $b20s$ | 5 | $< 1$ | 12 | $< 1$ | 50 | 8 |
| $b21s$ | 6 | $< 1$ | 12 | 1 | 21 | 5 |
| $b22s$ | 19 | $< 1$ | 14 | $< 1$ | 26 | 2 |

a graph can be used to build polynomial-time algorithms for fault collapsing, identifying independent fault pairs, and untestable faults. We also presented an enhanced FFG to model *skewed-load transition delay* (SLT) faults. This is the first polynomial-time algorithm for identifying untestable SLT faults. Compared to prior fault independent-based approaches for fault collapsing, our technique gives the best result by reducing the fault-set size by 66%. The advantage of our technique compared to previous techniques is that it can also identify independent fault pairs in the circuit. It will be interesting to investigate the use of the FFG data structure for *concurrent test generation* [4].

## References


[1] Lioy. A. Advanced Fault Collapsing. *IEEE Design and Test of Computers*, 9(1):64–71, Jan. 1992.

[2] R. Adapa, S. Tragoudas, and M. K. Michael. Accelerating Diagnosis via Dominance Relations between Sets of Faults. In *Proc. of the VLSI Test Symposium*, pages 219–224, 2007.

[3] V. D. Agrawal, A. V. S. S. Prasad, and M. V. Atre. Fault Collapsing via Functional Dominance. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 274–280, 2003.

[4] S. B. Akers, C. Joseph, and B. Krishnamurthy. On the Role of Independent Fault Set to Generate Minimal Test Sets. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 1100–1107, 1987.

[5] M. E. Amyeen, W. K. Fuch, I. Pomeranz, and V. Boppana. Fault Equivalence Identification in Combinational Circuits Using Implication and Evaluation Techniques. *IEEE Trans. on Computer Aided Design*, 22(7):922–936, July 2003.

[6] M. L. Bushnell and V. D Agrawal. *Essentials of Electronic Testing*. Springer, Boston, 2000.

[7] W. Mao and M. D. Ciletti. Reducing Correlation to Improve Coverage of Delay Faults in Scan-Path Design. *IEEE Trans. on Computer Aided Design*, 13(5):638–646, May 1994.

[8] V. J. Mehta, K. K. Dave, V. D. Agrawal, and M. L. Bushnell. A Fault-Independent Transitive Closure Algorithm for Redundancy Identification. In *Proc. of the Int'l. Conf. on VLSI Design*, pages 149–154, Jan. 2003.

[9] J. F. Poage. Derivation of Optimum Tests to Detect Faults in Combinational Circuits. In *Proc. of the Mathematical Theory of Automata*, pages 483–528, New York, N. Y., 1963. Polytechnic Press of Polytechnic Institute of Brooklyn.

[10] A. V. S. S. Prasad, V. D. Agrawal, and M. V. Atre. A New Algorithm for Global Fault Collapsing into Equivalence and Dominance Sets. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 274–280, 2002.

[11] R. K. K. R. Sandireddy and V. D. Agrawal. Diagnostic and Detection Fault Collapsing for Multiple Output Circuits. In *Proc. of the Design Automation and Test in Europe Conf.*, pages 1014–1019, 2005.

[12] J. Savir. Skewed-load Transition Test: Part I, Calculus. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 705–713, 1992.

[13] R. Sethuram. Sequential Automatic Test Pattern Generation Using Concurrent Implications Over Time. Master's thesis, Rutgers, The State University of New Jersey, Dept. of ECE, New Brunswick, NJ-08554, Oct. 2005.

[14] R. Sethuram and M. L. Bushnell. A Graph Condensation Based Transitive Closure Algorithm for Implication Graphs to Identify False Paths. In *Proc. of the IEEE North Atlantic Test Workshop*, pages 52–58, 2007.

[15] V. C. Vimjam and M. S. Hsiao. Efficient Fault Collapsing via Generalized Dominance Relations. In *Proc. of the VLSI Test Symposium*, pages 258–265, 2006.

[16] J. K. Zhao, J. A. Newquist, and J. H. Patel. A Graph Traversal Based Framework for Sequential Logic Implication With an Application to C-cyle Redundancy Identification. In *Proc. of the Int'l. Conf. on VLSI Design*, pages 163–169, Jan. 2001.