# A Novel, Low-Cost Algorithm for Sequentially Untestable Fault Identification*

**Manan Syal and Michael S. Hsiao** <{msyal, mhsiao}@vt.edu>
Bradley Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA.

***Abstract***

*This paper presents a new and low-cost approach for identifying sequentially untestable faults. Unlike the single fault theorem, where the stuck-at fault is injected only in the right-most time frame of the k-frame unrolled circuit, our approach can handle fault injection in any time frame within the unrolled sequential circuit. To efficiently apply our concept to untestable fault identification, powerful sequential implications are used to efficiently extend the unobservability propagation of gates in multiple time frames. Application of the proposed theorem to ISCAS '89 sequential benchmark circuits showed that more untestable faults could be identified using our approach, at practically no overhead in both memory and execution time.*

## 1. Introduction

The current state-of-the-art automatic test pattern generators (ATPGs) for sequential circuits spend a lot of time in trying to generate a test sequence for the detection of untestable faults, before aborting on them (or identifying them as untestable, given enough time). For untestable faults there does not exist a test sequence that can detect them, due to the unjustifiability of their excitation condition, their propagation condition, or both.

A number of approaches have been proposed in the past for the purpose of identifying untestable faults. Single Fault Theorem [1] states that if a single fault injected in the last time frame of a k-frame unrolled sequential circuit is found to be untestable using combinational ATPG, then the fault would be sequentially untestable. Three new procedures were also introduced in [2] as an extension to the single fault theorem, and a larger set of untestable faults were identified with the aid of these new procedures. FIRE [3], which is a fault-independent algorithm, identifies combinationally untestable faults as the faults that require conflicting assignments on a single line as a necessary condition for their detection. Since it is not possible for a line to have complementary values at the same time, faults requiring such an impossible assignment are untestable. FIRES [4], was introduced as an extension of FIRE for sequential circuits and used illegal state information as an additional criterion for identifying untestable faults. FILL [5] was introduced as a BDD based approach to identify illegal state information and FUNI [5] used that information to identify untestable faults. Since the success of algorithms such as FIRE and FUNI depends upon the number of implications associated with each line, it is important to have as large an implication set associated with each line as possible. A number of approaches have been proposed for learning implications. A 16-value logic algebra and reduction list method was used in [6] to determine node assignments. A more complete implication engine is based on recursive learning [7]. However, in order to keep simulation time within reasonable bounds, the recursion depth has to be kept low. A graphical representation of the implication graph was proposed in [8], and the concept of indirect implications based on the transitivity property of implications, along with extended backward implications [9] was used to increase the number of implications learnt. This form of representation also had the advantage of the ability to be used for sequential circuits, without suffering from memory explosion.

In this paper, we introduce an efficient and low-cost implementation to a powerful theorem, which states that if a *single fault* injected in *any time frame* of the k-frame unrolled sequential circuit is found to be untestable (i.e. there does not exist a k - frame combinational test to detect this single fault), the fault would be truly sequentially untestable. With this theorem, a target fault is untestable if and only if: (1) the target fault does not recombine with its copy in any time frame greater than the fault excitation frame or (2) the recombination occurs, the combined fault effect gets blocked. To make the theorem practical, we propose a very inexpensive method of ensuring that the fault declared as untestable does not become testable after recombining with its own copy in a higher time frame. The approach uses a sequential implication engine and the single-line conflict analysis similar to that used in FIRE as the basis for identifying untestable faults. Application of the proposed theorem to ISCAS '89 sequential benchmark circuits showed that significantly more untestable faults can be identified using our approach, at practically no overhead in both memory and execution time.

The rest of the paper is organized as follows. Section 2 gives an introduction to static implications along with single line conflict (FIRE) algorithm. Section 3 describes our algorithm used to extend the single-fault theorem and FIRE to find more untestable faults. Section 4 describes the implementation along with the overall algorithm. Section 5 reports the experimental results and section 6 concludes the paper.

## 2. Preliminaries

### 2.1 Static Implications

Static implications are the implications associated with both binary values of every gate in a given circuit, and comprise of direct, indirect and extended backward implications. Though direct implications for a gate can be easily found, indirect and extended backward implications require the extensive use of the transitive property and the contrapositive property [8]. These concepts can be understood from the following example.

We use the following terminology:

1. [N,v,t] : Indicates logic value *v* assigned to gate *N* during time frame *t*. Also, [N,v,0] is expressed as [N,v].
2. impl[N,v,t] : Set of implications resulting from assigning logic value *v* to gate *N* during time frame *t*.
3. a/v : Indicates line *a* stuck-at logic value *v*.

**Example 1:** Let us consider the implications of gate 'A' in Figure 1, set to logic value 1.

* *Supported in part by NSF Grant CCR-0196470 and NJ Commission on Science & Technology*

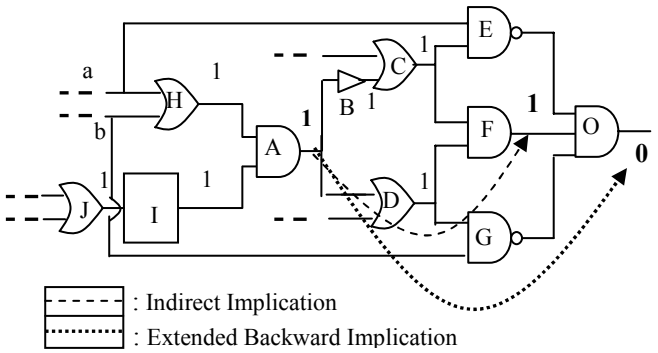


**Figure 1: Segment of a sequential circuit**

*a) Direct Implications:*

A logic value of '1' present at the output of gate A would imply B = D = 1. Also A = 1 → H = 1 and I = 1.
Thus, the set {(A,1,0), (B,1,0), (D,1,0), (H,1,0), (I,1,0)} represents the direct implications of A = 1.
Similarly, direct implications associated with B = 1 would be {(B,1,0), (C,1,0), (A,1,0)} and so on.
These implications are stored in the form of a graph with each node representing a gate (with value), a directed edge between nodes representing an implication, and a weight along with an edge representing the relative time-frame associated with the implication. Figure 2 shows the graphical representation of the direct implications for A = 1.

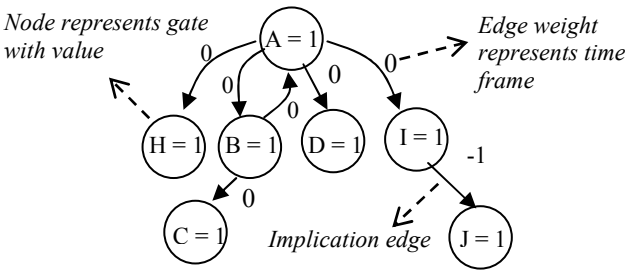


**Figure 2: Implication graph (shows direct implications)**

The complete set of implications resulting from setting A to 1 can be obtained by traversing the graph rooted at node A = 1 (transitive closure on A = 1). Thus, the complete set of direct implications associated with A = 1 is:
{(A,1,0), (H,1,0), (B,1,0), (C,1,0), (D,1,0), (I,1,0), (J,1,-1)}

*b) Indirect Implications:* Although C = 1 or D = 1 do not imply anything on gate F individually, together, they imply F = 1. Thus, indirectly, A = 1 would imply F = 1 (shown as a dashed line in Figure 1). This is called an indirect implication, and this implication is added as an additional outgoing edge from A = 1 in the implication graph.

Another non-trivial implication inferred from each indirect implication derives its roots from the contrapositive law. According to the contrapositive law, if [N,v] → [M,w,$t_1$], then [M, w'] → [N,v',-$t_1$] (here, v' represents the logical complement of v). Since A = 1 → F = 1 in time frame 0, then by contrapositive law, F = 0 → A = 0 in time frame 0.

*c) Extended Backward (EB) Implications:*

Extended backward implications apply to unjustified gates in the implication list. For the circuit shown in Figure 1, gate H = 1 is an unjustified gate in the implication list for A = 1, as none of H's inputs is implied to a value of logic 1. Thus, H is a candidate for the application of extended backward implications. To perform extended backward implications on H, a transitive closure is first performed for each of its unspecified inputs (i.e. 'a' and 'b'), obtaining impl[a=1] and impl[b=1], respectively. The implications of A=1 are simulated together with each of H's unspecified inputs' implication sets in turn, creating a set ($set_i$) of newly found implications for each input i. All new implications (not currently in the implication set for A = 1) that are common among these sets, are the extended backward implications and added as new edges to the original node A=1.
For our example, when the implications of (a = 1) and (A = 1) are simulated, then new implications found are (E,0,0) and (O,0,0). For the combined implication set of (b=1) and (A= 1), new implications found are (G,0,0) and (O,0,0). The set of implications common between the two sets is (O,0,0) and is hence added as a new edge to the implication graph for A = 1.

### 2.2 Untestable Fault Identification

The method used in our approach is the same as the original FIRE algorithm, based on single line conflicts. The basic concept is to find faults that require conflicting assignments on a line as a necessary condition for their detection.

For every gate 'g', the algorithm computes the following two fault sets:

*$Set_0$*: consisting of all faults that require gate 'g' to be 1 in order for each fault to be detected.

*$Set_1$*: consisting of all faults that require gate 'g' to be 0 in order for each fault to be detected.

Then, the set of untestable faults is simply all the faults common in the two sets, as these faults cannot be detected irrespective of the value on gate 'g'.

Let us consider the following example to understand the concept clearly:

**Example 2:** Consider the combinational portion of a circuit as shown in Figure 4. Let us consider the implications of x = 1.
Impl[x,1] = {(x,1,0), ($x_1$,1,0), ($x_2$,1,0), (b,1,0), (d,0,0), (e,0,0)}

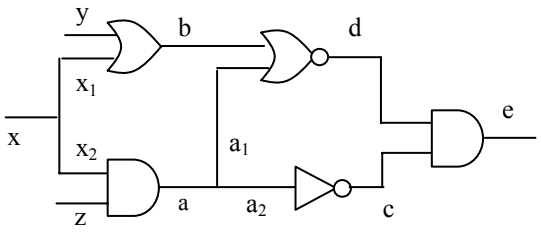


**Figure 4: Combinational portion of a circuit**

*Faults unexcitable due to x = 1:*

With x = 1, it would not be possible to set line b to 0, since x = 1 → b = 1. Thus, fault b/1 would be unexcitable with x=1, and would require x = 0 to be testable. *Essentially, if (k,v,t) is in the implication list for a node N, then fault k/v would be unexcitable in time frame t.* As a result, faults x/1, $x_1$/1, $x_2$/1, b/1, d/0, e/0 would all be unexcitable with x = 1.

*Faults unobservable due to x = 1:*

As x = 1 → d = 0, any fault value appearing at line c cannot be propagated to the next level. Hence faults c/0, c/1 require x to be 0 in order for them to be propagated/detected. Similarly, any faults appearing on lines y, $a_1$, $a_2$ etc would also be blocked due to the implications of x = 1. The complete set of faults that cannot be propagated because of x = 1 is {y/0, y/1, $a_1$/0, $a_1$/1, c/0, c/1, $a_2$/0, $a_2$/1, a/0, a/1, z/0, z/1, $x_2$/0, $x_2$/1}.

Thus, $Set_1$ = {x/1, $x_1$/1, $x_2$/1, d/0, e/0, b/1, y/0, y/1, z/0, z/1, $a_1$/0, $a_1$/1, c/0, c/1, $a_2$/0, $a_2$/1, a/0, a/1, z/0, z/1, $x_2$/0}

Now consider implications of x = 0.

Impl[x,0] = {(x,0,0), ($x_1$,0,0), ($x_2$,0,0), (a,0,0), ($a_1$,0,0), ($a_2$,0,0), (c,1,0)}

Similar to the analysis for x = 1, faults which are unexcitable and unobservable due to x = 0 are enumerated:

$Set_0$ = {x/0, $x_1$/0, $x_2$/0, a/0, $a_1$/0, $a_2$/0, c/1, z/0, z/1}

Thus, $Set_0 \cap Set_1$ = {$x_2$/0, a/0, $a_1$/0, $a_2$/0, c/1, z/0, z/1} forms the set of faults that are untestable, because these faults require an impossible (conflicting) assignment on line x as a necessary condition for their detection.

## 3. Our formulation for untestable fault identification

The iterative logic array (ILA) expansion of a sequential circuit consists of unrolled copies of the combinational portion of the circuit. Such an ILA expansion of a sequential circuit is shown in Figure 5. Here X defines the input bus, Y defines the output bus, $S_i$ defines the present state inputs to the $i^{th}$ frame, and $N_i$ defines the next state outputs of the $i^{th}$ frame. Let's assume that the initial state, $S_0$ as shown in Figure 5, has a reachable state space of size $|S_0|$. The subsequent reachable state space for successive time frames shrinks monotonically, i.e. $S_{i+1} \subseteq S_i$ for $0 < i < k-1$, where the circuit is unrolled in k frames.

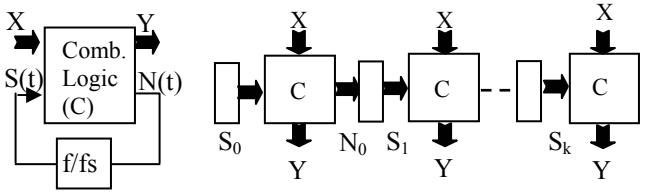


**Figure 5: ILA representation of a sequential circuit**

Before discussing our approach, let's illustrate how a combinationally untestable fault in a k-frame ILA circuit may become sequentially testable if the fault effect(s) combine with the same fault in any higher time frame.

**Example 3**: Consider the stuck-at fault a/1 injected in time frame 'i', as shown in Figure 6. Assume that the fault effect for a/1 propagates to the next time frame (marked as E). Assuming that the excitation condition for fault a/1 also implied gate C equal to logic 0 in time frame i+1, the fault effect 'E' would seem to be blocked in time frame i+1 at gate D. However, it would be incorrect to mark the fault as untestable because the controlling off-path that blocked the fault effect at gate D, contained gate A (the fault site) itself, indicating that the injected fault can recombine with its copy in time frame i+1 and become testable.

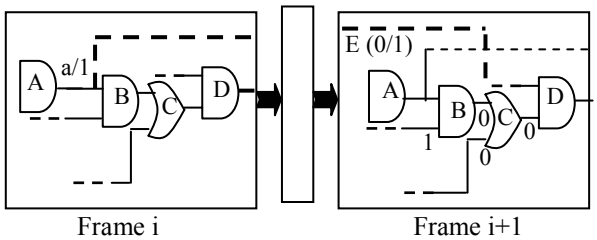


**Figure 6: A sequential circuit unrolled in two frames**

This recombination effect is shown in Figure 7.

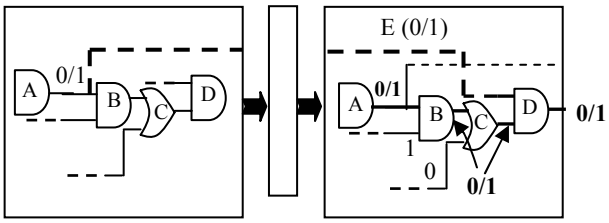


**Figure 7: Recombination of two copies of a fault**

Thus, if we inject a single fault, in order to avoid declaring a fault as untestable incorrectly, we must check if the fault effect recombines with itself in a higher time frame. However, recombination of a fault effect with its own copy does not necessarily imply that the fault is definitely testable. For example, assume that gate D is followed by gate F as shown in Figure 8. In this case, the fault effect would be truly blocked at gate F in time frame i+1, because the controlling off-path does not contain gate A, and the fault would not become testable even after recombination. Thus, declaring a fault as testable just because it recombines with itself in a higher time frame may be too conservative.

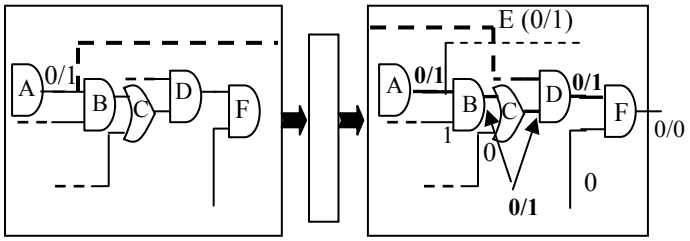


**Figure 8: True sequentially untestable fault**

In our approach, we guarantee that a fault that we declare as untestable either *does not recombine with itself in any time frame greater than the time frame in which the fault is injected*, or *even if recombination occurs, the combined fault effect truly gets blocked.* Thus, we present the following classifications that an untestable fault can fall under, as depicted by Figure 9.

a) If the fault is combinationally redundant (represented by region A in Figure 9).

b) If the fault can be excited in some time frame 'i', but the fault effect cannot be propagated across to time frame i+1 (represented by region B in Figure 9). (These can be captured by the Single Fault Theorem [2]).

c) If the fault effect crosses time frame boundaries, but does not recombine with a copy of itself in any higher time frame, and the fault effect is blocked before it can reach either the primary output or the flip-flop boundary in the last time frame (represented by region C in Figure 9).

d) If the fault effect recombines with itself in a higher frame, but the combined fault effect is eventually blocked (represented by region D).

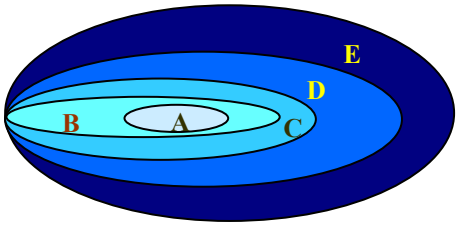


**Figure 9: Untestable fault model**

Region E in Figure 9 represents the entire set of sequentially untestable faults. In our work, we target regions C and D.

**Theorem 1:** *If a target fault injected in any time frame 'i' is found to be combinationally untestable, and if it can be guaranteed that the injected fault either does not recombine or does not become testable after re-combination with a copy of the same fault in any time frame greater than 'i', the fault would be truly sequentially untestable.*

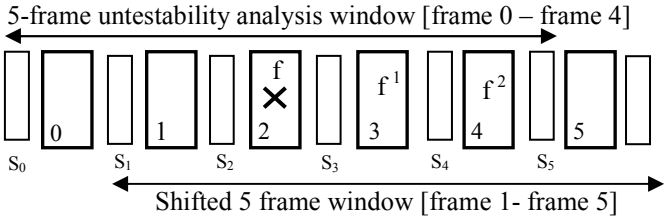


**Figure 10:** ILA representation of a circuit, with a 5- frame window for untestability identification

**Proof:** Consider the ILA representation of a circuit as shown in Figure 10. A k-frame window is selected to identify untestable faults. For illustration, assume k = 5. Also assume that the fault f is injected in time frame i = 2, as shown. Faults $f^1$ and $f^2$ represent copies of *f* in higher time frames. Let us assume the initial state $S_0$, to be fully controllable. If after excitation, the fault effect does not propagate across the current time frame boundary, then, in accordance with the single fault theorem [2], the fault is truly sequentially untestable. However, if the fault effect crosses the current time frame boundary (crosses frame 2), then one of the following may occur:

**a)** Assume that the fault effect *f* does not recombine with any of its copies ($f^1$ and $f^2$) in any higher time frame. Since $f^1$ and $f^2$ would not affect the injected fault *f*, every copy of *f* for time frames greater than 2 (i.e. $f^1$ and $f^2$) can be ignored from the perspective of *f*. If the injected fault *f* does not propagate to the primary outputs or to the D flip-flops in the last time frame of the k-frame window (i.e. to $S_5$), then the single fault would be combinationally untestable. Now we must prove that this fault is the sequentially untestable too. Consider the same fault injected in a higher time frame i+m. The analysis can be performed simply by shifting the 5-frame window over which the analysis is performed by m time units. In Figure 10, let's consider the same fault injected in time frame 3, so the window would shift by one time unit and would now extend from time frame 1 to time frame 5. Since the reachable state space at $S_1$ is a proper subset of the state space at $S_0$, it follows that if the fault injected in time frame 2 is untestable in the first 5-frame window, the same fault injected in time frame 3 will definitely not be observed over the shifted window as well. This is true for any fault injected in any time frame because if a fault injected in time frame i gets blocked in time frame i+m, then the same fault injected in any other time frame (i ± n) would also be blocked in time frame (i ± n)+m. Thus, if the fault injected in any time frame does not recombine with its own copy in a higher time frame and is found to be combinationally untestable, it would be sequentially untestable too.

**b)** Assume that the fault effect does recombine with one or more copies of the same fault in a higher time frame. However, if the combined fault effect gets blocked, then again, fault effects present in time frames greater than the fault injection frame (*i*) would not affect the detection of *f*. This is true because even though recombination of fault effects occurs, the fault effect (both combined and single) does not become observable. In order to prove that the fault is sequentially untestable, we can consider fault injection in any time frame i+m, and shift the analysis window by m time units. Again, the fault effect would recombine with a copy in some frame greater than i+m, but the combined fault effect would eventually get blocked within the analysis window. Thus, if a fault injected in any time frame gets blocked even after recombination, it would truly be sequentially untestable.

## 4. Implementation

Since our implementation is fault independent, unlike the implementation of the single fault theorem, we do not inject any fault in the circuit.

Whenever any input(s) to a gate is a controlling value for the gate, all the other inputs become unobservable because values present at these inputs would be blocked by the controlling value present at the other input. This unobservability propagates along the input cone of each of the "unobservable lines/inputs". The concept of unobservability propagation is shown in Figure 11. Whenever a stem is encountered in this unobservable cone, an analysis (in accordance with lemma 1 introduced in FIRE [3]) is performed to determine if all paths starting from the stem are blocked or not. However, we take special care when this unobservability cone extends beyond time frame boundaries. This is required to prevent declaring a signal as unobservable, when it actually becomes observable after re-combining with itself in a higher time frame.

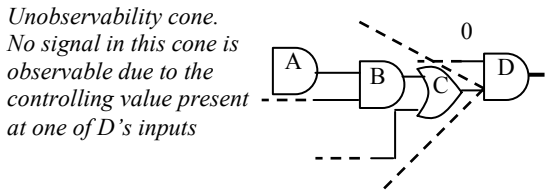


**Figure 11: Unobservability cone**

As stated in the theorem, if it can be guaranteed that a signal/fault does not re-converge with itself in a higher time frame or does not become observable/testable even after recombination, it can safely be marked as unobservable or untestable. Thus, we propagate unobservability backwards across time frame boundaries, and declare a signal as unobservable only if it is truly unobservable in accordance with our theorem. The key is to develop a technique that would determine if a signal is truly unobservable (even after recombination) or not, without causing extra overhead in terms of both execution time and memory requirements. We achieve this by performing a special stem analysis in a sequentially unrolled circuit. The concept can be better understood through the following example.

**Example 4:** Consider the circuit in Figure 12. Let us assume that the implications of a node, say z = 0, are as shown in Figure 12 ((C,1,i), (B,0,i+1), etc). The presence of a controlling value at the input of D in time frame i+1 (due to C = 0) would cause the input cone associated with the other input to become unobservable. Assuming that the unobservability propagates backward across the time frame boundaries, we would need to analyze the observability of the stem at the output of gate A in time frame *i*. The following steps are undertaken for this analysis:

**1)** We associate an ID (say 'X') with gate A in time frame *i*, and propagate it forward. Even though 'X' cannot propagate across path B-C-D (blocked by the controlling value present at the off-path input of C), the stem cannot be declared as unobservable at this stage because there is another path for the ID to propagate (bold, dashed path) across the time frame boundary.

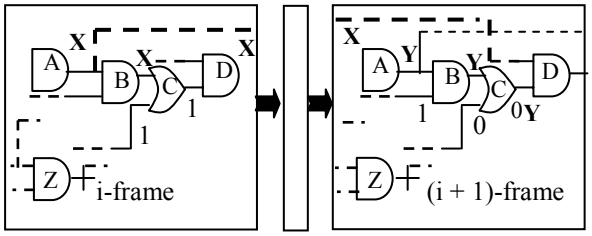


**Figure 12: Sequential circuit unrolled in two time frames**

**2)** Before we start analyzing the propagation path for ID 'X' in time frame i+1, we associate gate A (the gate from which ID propagation began in time frame *i*) in time frame i + 1 with a new ID 'Y'. We propagate this new ID until it gets blocked in the current time frame (i.e. frame i+1). Here, 'Y' would propagate across gates B, C and D.

**3)** Now we analyze the path of the previous ID, 'X' which came from time frame *i*. X would naturally propagate to gate D in frame i+1. Although the other input to gate D presents a controlling value, it actually has the other ID 'Y', implying that there exists a path for the fault at gate A in the time frame *i* to re-combine with the fault present at gate A in a higher time frame. Thus, the stem is not declared as unobservable and ID 'X' propagates across gate D in frame i+1. This analysis is in accordance with our theorem, because we declare a stem as unobservable only if the signal cannot become observable even after recombination with its copies in higher frames.

Clearly, this propagation of unobservability across frame boundaries increases the probability of finding more untestable faults.

**Algorithm**

The implication graph is generated as a preprocessing step to untestable fault identification. Initially a graph consisting of direct implications is generated. Indirect and extended backward implications are then computed for each gate and added to the implication graph. Single line conflict algorithm is then applied to identify untestable faults. The new theorem is applied wherever applicable to extend the unobservability cone across frame boundaries.

Generate sequential implication graph using:
Direct, indirect and extended backward implications.

//Perform single line conflicts with the addition of
//extended unobservability propagation.
$S_{untestable} = \phi$;
For every gate 'g' {
  compute $set_0$ = set of all faults that require g = 1;
  compute $set_1$ = set of all faults that require g = 0;
  (propagate unobservability across time frame boundaries when required)
  $S_{untestable} = S_{untestable} + (set_0 \cap set_1)$ }

## 5. Results

The proposed algorithm was implemented in C++ and experiments were conducted on ISCAS89 and ISCAS93 circuits on a 1.8 GHz, Pentium-4 workstation with 512 MB RAM, with Linux as the operating system. The results are reported in Tables 1 and 2. In Table 1, results were obtained for two implementations for each circuit. The first implementation (traditional method) did not consider the newly proposed theorem, and here, the unobservability propagation does not cross time frame boundaries. The second implementation, based on the proposed method, did not bound unobservability propagation to time frame boundaries, and we made sure that recombination of fault effects would not cause false positives via our theorem and algorithm. Since the implication graph for both the traditional and our approaches are exactly the same, the memory requirements are identical. For the traditional implementation, the untestable fault set $S_{untest}$, and execution time, are reported for time frames ranging from –1 to 1 (M or Maximum Edge Weight = 1) in columns 2 and 3 of table 1 and for time frames ranging –5 to 5 (M = 5) in columns 4 and 5. For the second implementation, which involves the implementation of our theorem, the implication graph is generated only over time frame range –1 to 1, and the set of untestable faults $S_{untest}$ identified and the corresponding execution times are reported in columns 5 and 6 in Table 1. It can be seen that for many circuits, our proposed approach could identify more untestable faults without too much additional computational effort. For example, in s5378, the proposed theorem could identify 877 untestable faults in only 46.02 seconds (with time frames ranging –1 to 1), while the traditional implementation could identify only 778 untestable faults over the same time frame range of –1 to 1, in 45.04 seconds. Even when the ILA size is increased to 11 (time frame –5 to 5), only 781 untestable faults were identified with the traditional method, at a cost of more than 900 seconds. This indicates that increasing the ILA size by the traditional method cannot capture untestable faults that recombine within the ILA. Note that our approach only required less than 1 seconds (in same ILA size) to perform unobservability propagation across time frame boundaries and detected nearly 100 more untestable faults. Likewise, for s3330, 73 more untestable faults were identified at practically no overhead. Not only does the implementation based on our new theorem identify more untestable faults without costing much additional effort, it also does not require additional memory overhead. For some circuits (s9234.1, s15850.1, etc), the number of untestable faults identified using the new method was the same as those identified using the traditional

approach. For these circuits, crossing the time frame boundaries did not benefit in identifying more untestable faults. Here, an implementation based on the single fault theorem would be sufficient to identify untestable faults, because faults injected in a time frame *i* that cross time frame boundaries do not get blocked in higher time frames with or without recombination with multiple copies of the same fault.

Table 2 compares our results with the untestable fault set obtained using FUNI+FIRE. Here, column 2 shows the number of untestable faults identified by the combination of FUNI and FIRE, when run on SUN sparc10 [5]. The column shows two quantities: number of untestable faults identified / number of time frames the sequential circuit was unrolled into. Column 3 shows the corresponding execution times. Column 4 through 7 show the number of untestable faults identified by our tool, using only 3 frames for all circuits, and the time taken for the analysis, respectively. We report results for 2 scenarios: with and without extended backward (EB) implications (since FUNI+FIRE did not use EB implications). It can be seen that although our tool does not explicitly enumerate a subset of the illegal state space, we can identify more untestable faults as compared to FUNI and FIRE combined, for quite a few circuits. FUNI outperforms our implementation for circuits which have a lot of un-initializable flip flops (such as s15850), and hence a lot of unreachable states. Since we do not enumerate unreachable states, we identify a smaller subset of untestable faults for these circuits. Note that for most circuits, we can identify more untestable faults using fewer number of frames. Finally, even when the tool is run without EB implications, we still identify a larger subset of untestable faults for a few circuits.

## 6. Conclusion

A novel, low-cost method for identifying untestable faults has been presented. Unlike the single fault theorem, we can handle fault injection in any time frame of a k-frame unrolled circuit. In order to make the implementation of the theorem feasible in terms of memory requirements and execution time, an implication-based algorithm is developed. More untestable faults were identified using this new theorem for many sequential circuits at low computational cost and no memory overhead. Future work includes identification of more untestable faults by combining our method with fault injection oriented algorithms.

| | Traditional Implementation | | | | New Approach | |
|---|---|---|---|---|---|---|
| **Circuit** | $S_{untest}$ M*=1 | Time (sec) | $S_{untest}$ M=5 | Time (sec) | $S_{untest}$ M=1 | Time (sec) |
| s386 | **60** | 0.53 | **60** | 0.92 | **60** | 0.55 |
| s400 | 7 | 0.22 | **8** | 0.81 | **8** | 0.26 |
| s499 | **89** | 0.84 | **89** | 7.04 | **89** | 0.89 |
| s713 | **32** | 0.38 | **32** | 0.64 | **32** | 0.41 |
| s953 | **5** | 3.36 | **5** | 8.41 | **5** | 3.99 |
| s991 | 0 | 0.48 | 0 | 1.69 | **6** | 0.50 |
| s1238 | **9** | 1.57 | **9** | 1.76 | **9** | 1.90 |
| s1423 | **9** | 0.43 | **9** | 1.24 | **9** | 0.58 |
| s3330 | 420 | 89.6 | 420 | 114.33 | **493** | 90.69 |
| s5378 | 778 | 45.04 | 781 | 965.46 | **877** | 46.02 |
| s9234.1 | 193 | 69.71 | 193 | 180.31 | **195** | 103.48 |
| s13207.1 | 374 | 343.12 | 381 | 1577.6 | **399** | 355.70 |
| s15850.1 | 315 | 201.85 | **317** | 1166.4 | **317** | 239.22 |
| s35932 | **3984** | 484.23 | **3984** | 880.08 | **3984** | 542.73 |
| s38417 | 328 | 365.27 | 332 | 3085.8 | **356** | 706.67 |
| s38584 | 1638 | 8113.5 | 1654 | 21341 | **1691** | 8415.1 |

**Table 1: Untestable faults (new thm. v/s the trad. method)**
* M : Maximum Edge Weight. M = 1 → frame range of –1 to 1

| | FUNI+FIRE | | Our Tool (W/O EB impl) | | Our Tool (W/ EB impl) | |
|---|---|---|---|---|---|---|
| **Circuit** | $S_{untest}$ / # Fr. | Time (sec) | $S_{untest}$ / #Fr.=3 | Time (sec) | $S_{untest}$ / #Fr.=3 | Time (sec) |
| s386 | 36/2 | 0.4 | **42** | 0.11 | **60** | 0.55 |
| s400 | **16**/3 | 2.0 | 8 | 0.42 | 8 | 0.26 |
| s713 | **91**/15 | 1.7 | 32 | 0.11 | 32 | 0.41 |
| s953 | 0/5 | 8.5 | **2** | 1.20 | **5** | 3.99 |
| s1238 | 6/3 | 4.1 | 6 | 0.85 | **9** | 1.90 |
| s1423 | 5/2 | 12.3 | **9** | 0.25 | **9** | 0.58 |
| s1196 | 0/3 | 3.5 | 0 | 0.32 | 0 | 0.96 |
| s5378 | 414/15 | 43.5 | **577** | 4.79 | **877** | 46.02 |
| s9234 | 257/15 | 108.8 | **272** | 43.95 | **274** | 180.27 |
| s13207 | 654/10 | 150.8 | **688** | 29.06 | **791** | 486.24 |
| s15850 | **816**/10 | 114.0 | 356 | 78.1 | 445 | 815.02 |
| s35932 | **3984**/0 | 237.4 | **3984** | 80.47 | **3984** | 542.72 |
| s38417 | **381**/5 | 373.4 | 337 | 297.1 | 356 | 706.67 |
| s38584 | 1582/5 | 405.4 | 1553 | 333.8 | **1691** | 8415.1 |

**Table 2: Comparison of our tool with FUNI +FIRE**

**Refrences:**


[1] V. D. Agrawal and S. T. Chakradhar, "Combinational ATPG theorems for identifying untestable faults in sequential circuits", *IEEE Trans. Computer-Aided Design, vol. 14, no. 9, Sept. 1995, pp. 1155-1160.*

[2] Sudhakar. M. Reddy, Irith. Pomeranz, X. Lim and Nadir Z. Basturkmen, "New procedures for identifying Undetectable and Redundant Faults in Synchronous Sequential Circuits", *VLSI Test Symposium, 1999. Proceedings. 17th IEEE, 1999 Page(s): 275 –281.*

[3] M. A. Iyer and M. Abramovici, "FIRE: a fault independent combinational redundancy algorithm", *IEEE Trans. VLSI, June 1996, pp. 295-301.*

[4] M.A. Iyer, D.E. Long, Abramovici, "Identifying sequential redundancies without search," *Design Automation Conference, 1996, pp. 457-462*

[5] D. E. Long, M. A. Iyer, M. Abramovici, "FILL and FUNI: Algorithms to identify illegal states and sequentially untestable faults," *ACM TODAES, pp 631-657*

[6] J. Rajski and H. Kox, "A method to calculate necessary assignments in ATPG". *Proc. Int'l. Test Conf. 1990, pp. 25-34*

[7] W. Kunz and D. K. Pradhan, "Recursive Learning: A new implication technique for efficient solutions to CAD problems-test, verification, and optimization", *IEEE Trans. On CAD, Sept 1994, pp. 1149-1158.*

[8] J. Zhao, J. A. Newquist and J. Patel, "A graph traversal based framework for sequential logic implication with an application to c-cycle redundancy identification", *Proc. VLSI Design Conf., 2001, pp. 163-169.*

[9] J. Zhao, M. Rudnik, J. Patel,"Static Logic Implication with application to fast redundancy identification", *Proc. VLSI Test Sym., 1997 pp. 288-293*