Extended Forward Implications and Dual Recurrence Relations to Identify Sequentially Untestable Faults*

Manan Syal (msyal@vt.edu)
Intel Corporation
Hillsboro, Oregon, US

Rajat Arora (raarora@cadence.com)
Cadence Design Systems
San Jose, California, US

Michael S. Hsiao (mhsiao@vt.edu)
Department of ECE, Virginia Tech
Blacksburg, Virginia, US

Abstract

*In this paper, we make two major contributions: First, to enhance Boolean learning, we propose a new class of logic implications called **extended forward implications**. Using a novel concept called implication-frontier, extended forward implications efficiently capture those non-trivial relationships which previous techniques failed to identify. Secondly, we introduce the concept of **dual recurrence relations** in sequential circuits, and propose a **new theorem** which uses this concept to quickly identify sequentially untestable faults. Our tool based on the proposed extended forward implications and the new theorem was applied to identify untestable faults in benchmark circuits. Significantly more untestable faults than reported by earlier techniques, low memory overhead and low computational complexity are the noteworthy features of our tool.*

1. Introduction

Logic implications capture the effect of asserting logic values throughout a circuit. Over the past few decades, logic implications have been successfully applied in several areas of electronic design automation (EDA) including: test-pattern-generation [1-3], logic and fault simulation [4], fault diagnosis [5], logic verification [6-8], logic optimization [9-10], untestable fault identification [11-15], etc. As a powerful implication engine can have a wide impact on EDA applications and tools, much effort has been invested in efficient computation of implications.

Schulz *et al.* were the first to improve the quality of implications by computing indirect implications in SOCRATES [1]. To further the application of these indirect implications, static learning was extended to dynamic learning [16] [17]. Cox *et al.* introduced the use of 16-valued algebra and reduction lists to determine node assignments in [18]. A transitive closure procedure on implication graph was proposed by Chakradhar *et al.* in [19]. A complete implication engine based on recursive learning was proposed by Kunz *et al.* [20] that can capture all pair-wise relations in a circuit. However, to keep simulation time within reasonable limits, the recursion depth is generally limited to low values. A graphical representation to store implications was proposed by Zhao *et al.* [21] and the concept of indirect implications based on transitivity of implications, along with extended backward implications were used to increase the number of implications learnt.

We propose an efficient technique to enhance Boolean learning through static implications, and present a new class of indirect implications which we refer to as *extended forward implications.* We introduce a novel concept called "*implication-frontier or I-Frontier*" and use it to efficiently identify those relationships (via extended forward implications) which were missed by previous techniques. While numerous applications can benefit from implications, in this paper, we demonstrate the impact of extended forward implications by applying them to identify untestable faults. The significance of our new learning criterion is clearly presented through the substantial leap achieved in the number of sequentially untestable faults identified.

*supported in part by NSF grants 0196470, 0305881, 0417340, and an Intel grant

Untestable faults are faults for which there exists no test sequence that can simultaneously excite the fault-effect and propagate it to a primary output (PO). In combinational circuits, untestable faults result only from redundant logic; in sequential circuits, untestable faults may also result from unreachable states. Automatic test pattern generators (ATPG) can spend exponential amount of time targeting untestable faults (especially for sequential circuits) before aborting or eventually identifying them as untestable. Thus, the performance of fault-oriented tools such as test-pattern-generators and fault-simulators can be enhanced if the knowledge of untestable faults is available *a priori*. In addition, the information about untestable faults can also benefit other applications such as combinational equivalence checking [6], logic optimization, etc.

Techniques used for untestable fault identification can broadly be classified into fault-oriented methods based on ATPG [22-24], and fault-independent techniques [11-15] based on conflict analysis. In general, ATPG-based methods outperform fault-independent methods for small circuits; however, the computational complexity of branch-and-bound algorithms (ATPG) makes them impractical for large sequential circuits. Conflict based analysis has thus been researched and improved over the years. Iyer *et al.* introduced FIRE [11] as a technique to identify untestable faults as faults that require a conflict on a single line as a necessary condition for their detection. FIRES [12] was introduced as an extension of FIRE to identify untestable faults in sequential circuits without explicit search. MUST [13] introduced by Peng *et al.* was built over the framework of FIRES as a fault-oriented approach to identify untestable faults. However, the memory requirement for MUST can be exponential. Hsiao [14] presented a fault-independent technique to identify untestable faults using multiple-node impossible value combinations. Recently, the approach in [14] was improved by Syal and Hsiao [15]. They also introduced the concept of recurrence relations that aid in identifying sequentially untestable faults. While recurrence relations are useful, we believe that their full potential has not been explored in [15].

In addition to enhancing learning through extended forward implications, in this paper we explore recurrence relations further. We present a new concept called *dual recurrence relations*, and propose a powerful theorem that uses this concept to bring about an additional increase in sequentially untestable faults. Since sequentially untestable faults form the performance bottleneck (speed and fault-coverage) for test pattern generation, the impact of identifying a large set of untestable faults can be significant. We demonstrate the usefulness of our results in the context of the performance gain provided by our tool to ATPG. Finally, as shown through the experiments, low computational overhead makes our tool scalable and practical.

The rest of the paper is organized in the following manner: Section 2 discusses basic concepts about static implications and single-line-conflict analysis. Section 3 presents extended forward implications and in Section 4 we propose a new theorem that uses dual recurrence relations to identify untestable faults. Section 5 illustrates experimental results, and section 6 concludes the paper.

2. Preliminaries

In this section, we give an overview of static logic implications [21] and single line conflict analysis [11].

2.1 Static Logic Implications

Logic implications identify the effect of asserting logic values within a circuit. In the past, static logic implications have broadly been classified into direct, indirect and extended backward implications [21]. Direct implications for $g = v$ (assigning logic v to gate g) are logic relations that can be identified without circuit simulation. Unlike direct implications, indirect and extended-backward implications are non-trivial, require circuit simulation, and identify implications on circuit elements which may not be directly connected to g. In the discussion that follows, we use the following terminology for implications [21]:

a) **Impl[N,v,t]**: Set of implications resulting from assigning logic value v to gate N in time frame t.
b) **[N,v,t] → [M,w,t$_1$]**: Assigning logic value v to N in time frame t implies that M would be assigned value w in time frame t_1.

To better understand the concept of direct, indirect and extended backward implications, consider the following example:

Example 1: Consider the implications of {A = 1} in Figure 1. The set {(B = 0), (C = 0)} forms the direct implications for {A=1} (relationships learnt without circuit simulation). Next, {B = 0} and {C = 0} together imply {D = 1}. Since {(B = 0), (C = 0)} ∈ Impl[A, 1, 0], [A, 1, 0] → [D, 1, 0]. This is an indirect implication and can be learnt through logic simulation of the circuit under the implications of {A = 1}.

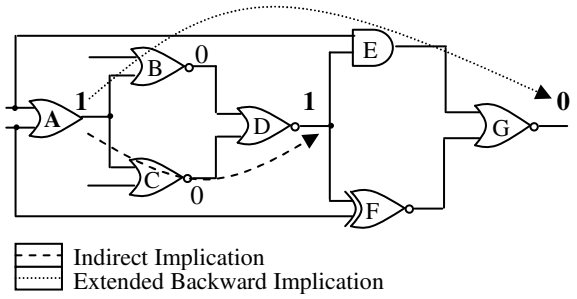

Figure 1: Circuit fragment to illustrate implications

Finally, since {A = 1} is currently unjustified, it forms a candidate for the application of extended backward implications. Extended backward implications are obtained by simulating {A = 1} together with each of its unspecified inputs. Applying extended backward implications on {A = 1}, we learn that [A, 1, 0] → [G, 0, 0]. Thus, the set Impl[A, 1, 0] is {(A=1), (B = 0), (C = 0), (D = 1), (G = 0)}.□

We store these implications as a graph [21] because this graphical representation can efficiently be applied to enumerate sequential implications without suffering from memory explosion.

2.2 Single Line Conflict Analysis

The underlying concept behind single line conflict analysis [11] is the following: faults that require a conflict on a single line as a necessary condition for their detection are untestable. In this analysis, for every gate g (or stem s), two sets are computed:

Set_0: Set of faults untestable with $g = 0$.
Set_1: Set of faults untestable with $g = 1$.

The faults in Set_0 (Set_1) require the assignment of $g = 1$ ($g = 0$) as a necessary condition for detection. The set of untestable faults then is the intersection of the two sets. For better understanding, consider Example 2.

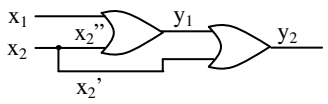

Figure 2: Illustration of single-line-conflict analysis

Example 2: In the following discussion, assume that g/v denotes the fault 'g stuck at v'. Consider stem x_2 in the circuit shown in Figure 2. The set of faults that become untestable (unexcitable and unobservable) when $x_2 = 0$ are: Set_0: {x_2/0, x_2"/0, x_2'/0}.
Similarly, the set of faults that would become untestable when x_2=1 are: Set_1: {x_2/1, x_2"/1, x_2'/1, y_1/1, y_2/1, x_1/0, x_1/1, x_2'/0, x_2''/0, y_1/0}. The faults in the intersection of Set_0 and Set_1, i.e. {x_2'/0, x_2''/0} would be untestable according to single-line-conflict analysis (these faults require conflicting assignments on x_2 for their detection). □

As observed from the example above, untestable faults are identified via identification of unexcitable (y_1/1, y_2/1 etc. are unexcitable when $x_2 = 1$) and unobservable nets (x_2'/0 and x_2'/1 etc. are unobservable when $x_2 = 1$). It should also be noted that the implications of $x_2 = 0$ and $x_2 = 1$ are used to enumerate unexcitable and unobservable nets. For sequential circuits, we implement unobservable net identification by the approach indicated in [25] to rule out false positives due to fault-reconvergence across multiple time-frames (false positives refers to declaring a net as unobservable, when it is actually observable).

3. Enhancing Implications

For single-line conflict analysis (and also for other conflict based approaches [13-15]), unexcitable/unobservable nets due to a particular logic assignment {$g = v$} are enumerated using implications of {$g = v$}. Therefore, increasing the set of logic implications would enable identification of potentially more unexcitable and unobservable nets due to a given assignment, translating to an increased number of untestable faults identified.

In this section, we propose a new technique to improve Boolean learning. This results in a new class of implications, termed as *extended forward implications*. Before explaining extended forward implications, we define a new concept called implication frontier (I-Frontier), and provide a motivating example.

Definition 1: Implication Frontier (I-Frontier)
The *I-Frontier* of a logic assignment $g = v$ is the set of gates that are currently unspecified but each gate in the set has one or more inputs implied by $g = v$. If g_{ik} represents the kth input of an n-input gate g_i, then I-Frontier can be represented as:
I-Frontier[g=v]: {$g_i \mid g_i \notin$ Impl[g,v,0], $\exists g_{ik} \in$ Impl[g,v,0], $1 \le k \le n$} □

The motivation behind extended-forward implications can be understood through the following example:

Example 3: A = 1 in Figure 3 does not imply any assignments (by direct, indirect and extended backward implications). Thus, Impl[A, 1, 0] = {(A, 1, 0)} (reflexive property).

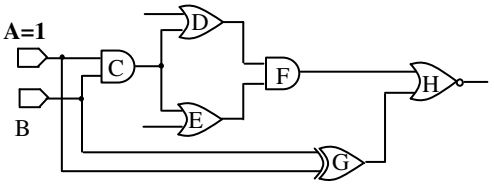

Figure 3: Motivation for extended forward implications

The I-Frontier of A = 1 consists of two gates: gate C and gate G. Each of these gates has one specified input (gate A), and one unspecified input (gate B). Let us focus on gate C. If we set gate B

to logic 0 (the unspecified input of gate C), and simulate the circuit with Impl[A,1,0]∪ Impl[B,0,0], a set (R_0) of new logic assignments can be learnt: {(C = 0), (G = 1), (H = 0)}.
Next, if we set B = 1, and again simulate the circuit, another set (R_1) of new assignments can be learnt: {(C = 1), (D = 1), (E = 1), (F = 1), (G = 0), (H = 0)}.
Now, {Impl[A,1,0] ∪ Impl[B,0,0]} implies R_0 while {Impl[A,1,0] ∪ Impl[B,1,0]} implies R_1. Thus, $R_0 \cap R_1 \in$ {(Impl[A,1,0] ∪ Impl[B,0, 0]) ∩ (Impl[A,1,0] ∪ Impl[B,1,0])}.
Or, $R_0 \cap R_1 \in$ {Impl[A,1,0] ∪ (Impl[B,0,0] ∩ Impl[B,1,0])}, since Impl[B,0,0] ∩ Impl[B,1,0] = ϕ, $R_0 \cap R_1 \in$ Impl[A,1,0].
Finally, because $R_0 \cap R_1$ = (H = 0), we can learn that [A,1,0] → [H, 0, 0]. By setting the unspecified input of C (a gate in the I-Frontier[A=1]) to both logic values, we were able to identify a new logic implication which was missed by all of direct, indirect and extended backward implications. □

The main idea behind extended forward implications is to learn new relations using the I-Frontier for an assignment $\{g = v\}$. Now we formally define extended forward implications (EF for short):

Definition 2: Extended Forward (EF) Implications
If $g_i \in$ I-Frontier$[g = v]$, then:

a) *If g_{ik} is the only input of g_i that is currently unspecified,*

R_0 = logic_simulate(Impl[g, v, 0] ∪ Impl[g_{ik}, 0, 0])
R_1 = logic_simulate(Impl[g, v, 0] ∪ Impl[g_{ik}, 1, 0]) ---- (1)

b) *If g_i has more than one unspecified input,*

R_0 = logic_simulate(Impl[g, v, 0] ∪ Impl[g_i, 0, 0])
R_1 = logic_simulate(Impl[g, v, 0] ∪ Impl[g_i, 1, 0]) ---- (2)

EF$\{g = v\}$ = $R_0 \cap R_1$; Impl[g, v, 0] ∪= EF$\{g = v\}$. □

The motivation behind extended-forward (EF) implications is to push the envelope of implications for $\{g = v\}$ beyond the I-Frontier. For case (a) above, the attempt to go beyond the I-Frontier is performed by trying *both* logic values for the unspecified input of g_i {g_{ik} = 0 and g_{ik} = 1} and taking an intersection of the set of new logic assignments for each logic value. For case (b) more than one inputs of g_i are unspecified: it would be too expensive (computationally) to try the complete value combinations for all unspecified inputs. Instead, both logic values for the gate output {g_i = 0 and g_i = 1} are simulated. Since the underlying concept behind extended forward implication tries to *extend* implications beyond the point *forward* implications reach (bounded by the I-Frontier), hence the name.

Now we provide important characteristics of EF implications and I-Frontier through the following Lemmas. As explained later, these characteristics are used in making EF implications efficient.

Lemma 1: If Impl[g_1, v, 0] ⊇ Impl[g_2, w, 0], I-Frontier$\{g_1 = v\}$ ⊇ I-Frontier $\{g_2 = w\}$.
Proof: By contradiction, assume that Impl[g_1, v, 0] ⊇ Impl[g_2, w, 0] and I-Frontier$\{g_1 = v\}$ ⊂ I-Frontier $\{g_2 = w\}$. Thus, there exists at-least one gate g_i, such that $g_i \in$ I-Frontier$\{g_2 = w\}$ and $g_i \notin$ I-Frontier$\{g_1 = v\}$. By definition of I-Frontier, there must exist at-least one input g_{ik} of g_i such that $g_{ik} \in$ Impl[g_2, w, 0] and $g_{ik} \notin$ Impl[g_1, v, 0]. However, this would violate our initial assumption that Impl[g_1, v, 0] ⊇ Impl[g_2, w, 0]; which proves the Lemma. □

Lemma 2: If Impl[g_1,v,0] ⊇ Impl[g_2,w,0], EF$\{g_1 = v\}$⊇EF$\{g_2 = w\}$.
Proof: Given Impl[g_1,v,0] ⊇ Impl[g_2,w,0]. Consider Definition 2: to enumerate EF$\{g_1 = v\}$ and EF$\{g_2 = w\}$, I-Frontier$\{g_1 = v\}$ and I-Frontier$\{g_2 = w\}$ are used. For each gate g_i in these I-Frontiers, R_0 and R_1 are created and intersected. Thus, *the number of EF implications learnt is directly proportional to the size of the I-Frontier, and the size of R_0 and R_1 for each gate in the I-Frontier*. From Lemma 1, since Impl[g_1, v, 0] ⊇ Impl[g_2, w, 0], I-Frontier$\{g_1 = v\}$ ⊇ I-Frontier $\{g_2 = w\}$. Also, since R_0 and R_1 are proportional to Impl[g_1, v, 0] and Impl[g_2,w, 0], it follows that R_0 and R_1 with $\{g_1 = v\}$ will be a superset of the corresponding sets for $\{g_2 = w\}$. Thus, EF$\{g_1 = v\}$ ⊇ EF$\{g_2 = w\}$. □

While learning additional Boolean relationships through EF implications can be useful, this learning should not come at the cost of high computational overhead. Observe from equations (1) and (2) that the computation of EF$\{g = v\}$ involves two passes of logic simulation corresponding to each gate in the I-Frontier of $\{g = v\}$. Although EF implications can significantly increase the number of implications learnt, the cost associated with logic simulation can potentially become prohibitive as circuit size grows. The time spent in logic simulation is worthwhile if all the gates in the I-Frontier result in new EF implications. However, we observed that a large subset of gates in the I-Frontier do not result in identification of new implications, but still account logic simulation time. Therefore, techniques that can quickly identify and prune out such 'useless' gates from the I-Frontier is desirable. In the following discussion, we describe *two efficient techniques* for this purpose.

3.1 EF Pruning Techniques

A. Pruning Based on I-Frontier: First, we present an important characteristic of I-Frontiers through Lemma 3. This characteristic can prove vital not only in improving the efficiency of learning EF implications, but also accelerate any other Boolean learning technique based on simulation (e.g. extended backward learning).

Lemma 3: If [I-Frontier$\{g_1 = v\}$ ∩ I-Frontier$\{g_2 = w\}$] = ∅, simulating {Impl[g_1, v, 0] ∪ Impl[g_2, w, 0]} would not result in the identification of any new value assignments.

Proof: We prove this Lemma by contradiction. Without loss of generality, consider a sequential circuit with only two-input gates and flip-flops. Refer to Figure 4 for the following discussion.

***Statement of contradiction**:* By contradiction assume [I-Frontier$\{g_1 = v\}$ ∩ I-Frontier$\{g_2 = w\}$] = ∅, but a set of *new* logic assignments, S, is obtained on simulating Impl[g_1, v, 0] ∪ Impl[g_2, w, 0].

Let, $g_i \in$ S (i.e. $\{g_i = v_i\} \in$ S is learnt through simulating {Impl[g_1, v, 0] ∪ Impl[g_2, w, 0]}). Note that if $\{g_i = v_i\} \in$ S, $\{g_i = v_i\} \notin$ {Impl[g_1, v, 0] ∪ Impl[g_2, w,0]}, and vice versa.

Step 1: Since $g_i \in$ S, either $\{g_i = 1\} \in$ S or $\{g_i = 0\} \in$ S. For ease of understanding, let g_i be AND gate as shown in Figure 4, and have two gates g_{i1} and g_{i2} as its inputs.

Case A: Assume $\{g_i = 1\} \in$ S. Now, $\{g_i = 1\}$ can be learnt through the simulation of {Impl[g_1, v, 0] ∪ Impl[g_2, w, 0]}, if:

I. g_{i1} = 1 and g_{i2} = 1 are specified prior to logic simulation; OR

II. g_{i1} = 1 and g_{i2} = 1 are learnt during logic simulation.

Consider part I. Both g_{i1} = 1 and g_{i2} = 1 can be specified prior to logic simulation only if one of these conditions is true:

1. $\{(g_{i1} = 1), (g_{i2} = 1)\} \in$ Impl[g_1, v, 0]
2. $\{(g_{i1} = 1), (g_{i2} = 1)\} \in$ Impl[g_2, w, 0]
3. $(g_{i1} = 1) \in$ Impl[g_1, v, 0] and $(g_{i2} = 1) \in$ Impl[g_2, w, 0]
4. $(g_{i1} = 1) \in$ Impl[g_2, w, 0] and $(g_{i2} = 1) \in$ Impl[g_1, v, 0]

If either condition (1) or (2) were true, then $(g_i = 1) \in$ {Impl[g_1, v, 0] ∪ Impl[g_2, w, 0]} (by indirect implications). Since $\{g_i = 1\} \in$ S, conditions (1) and (2) are not possible. If either condition (3) or (4) were true, then by Definition 1, $g_i \in$ [I-Frontier$\{g_1 = v\}$ ∩ I-Frontier$\{g_2 = w\}$]. This conflicts the assumption that [I-Frontier$\{g_1 = v\}$ ∩ I-Frontier$\{g_2 = w\}$] = ∅.

Thus, part I cannot be true if $\{g_i = 1\} \in$ S. As a result, both g_{i1} = 1 and g_{i2} = 1 must also be learnt during logic simulation.

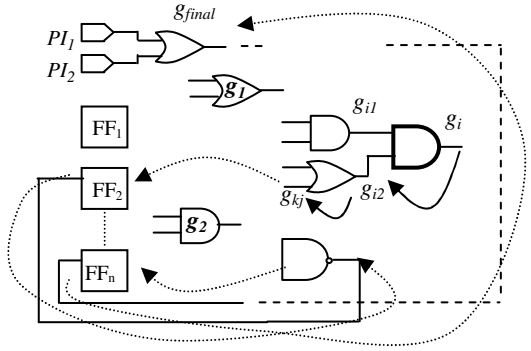

Figure 4: Illustration for Lemma 3

***Case B**:* Assume that $\{g_i = 0\} \in$ S. $\{g_i = 0\}$ can be learnt through the simulation of $\{\text{Impl}[g_1, v, 0] \cup \text{Impl}[g_2, w, 0]\}$, if:

I. $g_{i1} = 0$ or $g_{i2} = 0$ is specified prior to logic simulation;
OR

II. $g_{i1} = 0$ or $g_{i2} = 0$ is learnt during logic simulation.

Again, consider Part-I. If either $g_{i1} = 0$ or $g_{i2} = 0$ is specified prior to logic simulation, then $\{g_i = 0\}$ will also be specified prior to simulation. But, by assumption, $\{g_i = 0\} \in$ S (learnt after simulation). As a result, part I cannot be true. Thus, either $\{g_{i1} = 0\}$ or $\{g_{i2} = 0\}$ must be learnt during logic simulation.

From Case A and Case B, ***if $g_i \in$ S, at least one gate at the input of g_i must also be learnt through simulation***. That is, either $g_{i1} \in$ S and/or $g_{i2} \in$ S. Assume $g_{i2} \in$ S.

Step 2: The arguments used for g_i can also be applied for g_{i2}. Given $[\text{I-Frontier}\{g_1 = v\} \cap \text{I-Frontier}\{g_2 = w\}] = \varnothing$, $g_{i2} \in$ S is possible only if there exists some gate g_{kj} at the input of gate g_{i2} such that $g_{kj} \in$ S. *Since, $g_i \in S$ is possible only if $g_{i2} \in S$, by transitivity, it follows that $g_i \in S$ is possible only if $g_{kj} \in S$.*

***Step 3*:** Applying this argument inductively, we would reach gate g_{final} in the input-cone of g_i such that both inputs of g_{final} are primary inputs (say PI_1 and PI_2). Note that, as shown in Figure 4, g_{final} may be reached after passing through several gates, which may include flip-flops. From induction, *$g_i \in S$ is possible only if $g_{final} \in S$.*
From Step 1, if $g_{final} \in$ S either $PI_1 \in$ S or $PI_2 \in$ S. However, since both PI_1 and PI_2 are primary inputs, new assignments on PI_1 or PI_2 cannot be learnt through simulation. That is, $PI_1 \notin$ S and $PI_2 \notin$ S. As a result, $g_{final} \notin$ S. Since $g_i \in$ S is possible only if $g_{final} \in$ S, $g_i \notin$ S. Thus, if $[\text{I-Frontier}\{g_1 = v\} \cap \text{I-Frontier}\{g_2 = w\}] = \varnothing$, no new assignments can be identified by simulating $\{\text{Impl}[g_1, v, 0] \cup \text{Impl}[g_2, w, 0]\}$. □

Let us understand how Lemma 3 can be used to compute EF implications more efficiently. From equations (1) and (2) in the previous section, the implications of $g = v$ are combined with the implications of gate g_{ik} in (1) and g_i in (2), followed by simulation.

Proposition 1: Using Lemma 3, new EF implications would not be learnt if:

a) For equations (1): $[\text{I-Frontier}\{g = v\} \cap \text{I-Frontier}\{g_{ik} = 0\}] = \varnothing$ or if $[\text{I-Frontier}\{g = v\} \cap \text{I-Frontier}\{g_{ik} = 1\}] = \varnothing$;
b) For equations (2): $[\text{I-Frontier}\{g = v\} \cap \text{I-Frontier}\{g_i = 0\}] = \varnothing$ or if $[\text{I-Frontier}\{g = v\} \cap \text{I-Frontier}\{g_i = 1\}] = \varnothing$

In either case, g_i can be dropped from the I-Frontier of $\{g = v\}$ without loss of new EF implications. □

Consider the impact of Lemma 3: by performing a simple check (intersection) on the I-Frontiers, Lemma 3 enables the identification those gates which do not contribute to any new learning via EF implications. By removing such gates from the I-Frontier, saving in simulation cost is achieved. We observed that this pruning technique reduced the execution time of learning implications by as much as 30% - 40% for benchmark circuits.

Consider the following example for an illustration of Proposition 1.
Example 4: Figure 5 shows the combinational portion of a circuit and the I-Frontiers of some nets of interest.

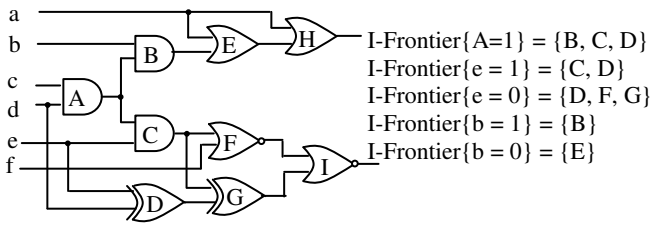

Figure 5: Illustration of pruning based on I-Frontier

By direct, indirect and extended backward implications, Impl[A, 1, 0] = {(A=1), (c = 1), (d = 1)}. Assume that EF{A=1} need to be enumerated. Candidates in the I-Frontier for {A=1} are {B, C, D}. First consider candidate B. Since [I-Frontier{A=1} ∩ I-Frontier {b=0}] = ∅, Proposition 1 allows us to drop gate B from I-Frontier{A=1}. It can be verified that simulating Impl[A, 1, 0] ∪ Impl[b, 0, 0] would not result in any new logic assignments. Now, consider candidates C and D. The unspecified input of gates C and D is gate e. Since [I-Frontier{A=1} ∩ I-Frontier {e = 1}] ≠ ∅ and [I-Frontier{A=1} ∩ I-Frontier {e = 0}] ≠ ∅, EF implication process is performed for both candidates C and D. EF for candidates C and D results in identification of new implications ({G = 1}, {I = 0}). □

B. Pruning Based on Value: In this heuristic, we use the characteristics of EF implications described earlier. Specifically, using Lemma 2, we propose the following Lemma to further prune out those gates which do not contribute to identification of EF implications.

Lemma 4: Assume that the value of gate g_i is controlled to logic w when one or more inputs $g_{ik} = c$. If:
(i) EF has been performed for all inputs $g_{ik} = c$, and
(ii) extended backward implications are performed for $\{g_i = w\}$;
Then: new EF implications would not be obtained for $g_i = w$.

Proof: Without loss of generality, assume g_i to be AND gate. Let g_{ik} be the gate at k^{th} input of gate g_i. Thus, $\forall_k \{g_{ik} = 0\} \rightarrow \{g_i = 0\}$ (g_i is controlled to logic 0, when one or more inputs $g_{ik} = 0$). As a result, $\forall_k \text{Impl}[g_{ik}, 0, 0] \supseteq \text{Impl}[g_i, 0, 0]$. From Lemma 2, $\forall_k \text{EF}\{g_{ik} = 0\} \supseteq \text{EF}\{g_i = 0\}$.
Let us assume that gate g_i has n gates g_{i1} - g_{in} as inputs. Thus,

$[\text{EF}\{g_{i1} = 0\}] \supseteq [\text{EF}\{g_i = 0\}]$ ---------- (a)

Intersecting $\text{EF}\{g_{i2} = 0\}$ on both sides of equation (a),

$[\text{EF}\{g_{i1} = 0\} \cap \text{EF}\{g_{i2} = 0\}] \supseteq [\text{EF}\{g_i = 0\} \cap \text{EF}\{g_{i2} = 0\}]$-- (b)

Since $\text{EF}\{g_{i2} = 0\} \supseteq \text{EF}\{g_i = 0\}$ (note right side of eq. (b)),

$[\text{EF}\{g_{i1} = 0\} \cap \text{EF}\{g_{i2} = 0\}] \supseteq [\text{EF}\{g_i = 0\}]$ ---------- (c)

Performing this intersection operation for all inputs,

$[\text{EF}\{g_{i1} = 0\} \cap \text{EF}\{g_{i2} = 0\} \ldots \cap \text{EF}\{g_{in} = 0\}] \supseteq \text{EF}\{g_i = 0\}$--(d)

Assume that each term on the left-hand-side has been individually evaluated (i.e. $\text{EF}\{g_{ik}=0\}$ has been evaluated for all inputs). By definition of extended backward implications, left side of equation (d) can be identified by performing extended backward implications on $\{g_i = 0\}$. Thus, equation (d) implies that if $\text{EF}\{g_{ik} = 0\}$ has been performed for all inputs of g_i, performing extended-backward implications on $\{g_i = 0\}$ would be sufficient; new EF implications would not be identified for $\{g = 0\}$. Similar reasoning applies to other gates for which $g_{ik} = c$ controls the output of g_i to logic w (e.g. if g_i is OR gate, any input $g_{ik} = 1$ controls output of g_i to logic 1). □

Consider the impact of Lemma 4: For any gate g_i with n inputs g_{i1}-g_{in}, once both conditions in Lemma 4 are satisfied, EF$\{g_i = w\}$ can be completely avoided. By not performing EF$\{g_i = w\}$, logic simulation is avoided for all gates in the I-Frontier for $\{g_i = w\}$.

For better understanding, consider the following example:

Example 5: Let us identify EF{A = 0} in Figure 6 using two procedures: in procedure 1 we explicitly perform EF{A=0} using Definition 2 and in procedure 2, we use Lemma 4.

Procedure 1: Since B is the only gate in the I-Frontier of {A=0}, EF{A=0} would be enumerated as:

R_0 = logic_simulate(Impl[A, 0, 0] ∪ Impl[c, 0, 0]);
Or, R_0 = {(B = 0), (C = 1), (D = 0)}
R_1 = logic_simulate(Impl[A, 0, 0] ∪ Impl[c, 1, 0])
Or, R_1 = {(B = 1), (D = 0)}

Thus, EF{A=0} = $R_0 \cap R_1$ = {(D = 0)}.

Procedure 2: It can be verified that EF{a = 0} = EF{b = 0} = {(D = 0)}. Now, by performing extended backward implications on {A=0}, we can directly learn the implication [A, 0, 0] → [D, 0, 0]. Unlike Procedure 1, Procedure 2 learns EF{A=0} without explicitly performing logic simulation. Once EF{a=0} and EF{b=0} are performed, by Lemma 4, only extended backward implications on {A=0} are needed to identify EF{A=0} (eliminating the need for additional logic simulations). □

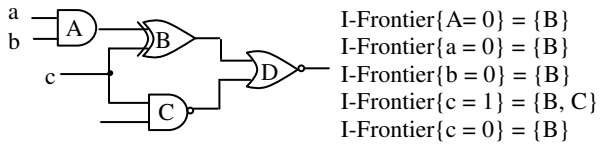

Figure 6: Illustration of pruning based on value

This pruning technique based on value is more powerful than the pruning based on I-Frontier and helped achieve an additional 10% - 15% reduction in execution time on top of the latter. These two pruning techniques together make EF implications an efficient way of improving the learning capabilities of our implication engine. Shown below is complete algorithm for enumerating implications:

/*Implication Engine*/

For all gate assignments in levelized order ($\{g = w\}$, $w \in \{0,1\}$),
1. Perform Direct, Indirect and Extended Backward Implications
2. If w is the non-controlling value for g (Pruning based on value)
   2.1. Identify I-Frontier for $g = w$
   2.2. Prune I-Frontier based on Proposition 1
   2.3. Perform EF implications using eqns (1) and (2)

4. Theorem for Sequentially Untestable Faults

Learning non-trivial Boolean relationships via EF implications can prove instrumental only when these relationships are analyzed and applied suitably. In this section, we propose a new Theorem that efficiently utilizes the extra knowledge provided by EF implications towards identifying untestable faults. This Theorem is based on a new concept called dual recurrence relations. Dual recurrence relations explore certain unique characteristics of sequential implications to help derive conclusions about untestable faults. Before discussing the new theorem, the following two terms [15] need to be defined to facilitate better understanding of the theorem.

Definition 3: The value assignment $g = v$ is said to be unachievable if there exists no input sequence that can set gate g to value v with the initial state of all flip-flops being unknown or X. □

Definition 4: A recurrence relation exists for a gate g with value v if $[g, v, t] \rightarrow [g, v, t\text{-}t_1]$. For $t_1 < 0$, this relation is called a forward recurrence relation, while for $t_1 > 0$, the relationship is called a backward recurrence relation. □

Syal and Hsiao [15] used the following Lemma based on the knowledge of recurrence relations to identify unachievable assignments:

For a gate g, if
a) The backward recurrence relation exists for g = v, and
b) g = v is not a constant assignment;
Then the value assignment g = v is unachievable.

If condition (a) is true for a gate g (i.e. a backward recurrence relation exists for $\{g = v\}$), logic simulation is performed in [15] using random vectors (starting from unknown initial state) to determine if condition (b) also holds true for g. Condition (b) is declared true *only if* $g = v'$ is achieved during simulation (v' represents the logical compliment of logic value v; $\{v, v'\} \in \{0, 1\}$). When $g = v'$ is achieved, conditions (a) and (b) evaluate to true, and assignment $g = v$ is declared as unachievable according to the Lemma [15].

However, it is possible that during simulation (as employed in [15]) gate g always remains un-initialized, i.e. g always remains X. In such a scenario, the simulation based technique described in [15] cannot make any decisions regarding the controllability of gate g. In this paper we present a more powerful theorem than the Lemma in [15] which would enable meaningful decisions to be made about the controllability of a gate *without performing random simulation*.

We first define *dual-recurrence relations* and discuss Lemma 5 before stating the Theorem.

Definition 5: A dual recurrence relation exists for a gate g if a backward recurrence relations exist for $\{g = v\}$ and $\{g = v'\}$ (i.e. $[g, v, t] \rightarrow [g, v, t\text{-}t_1]$ and $[g, v', t] \rightarrow [g, v', t\text{-}t_1]$ for $t_1 > 0$). □

Definition 6: If $\{g = v\}$ is a constant assignment, $\{g = v\}$ is true in every time frame of the sequential circuit after circuit synchronization. If the sequential machine powers up into an illegal starting state that sets gate g to v', the machine eventually goes into a state that sets $g = v$ after which g retains value v in all time frames.

Lemma 5: If a dual recurrence relation exists for a gate g, then neither $\{g = v\}$ nor $\{g = v'\}$ is a constant assignment.
Proof: If a dual recurrence relation exists for gate g, then
$[g, v, t] \rightarrow [g, v, t\text{-}t_1]$ --------- (a) AND
$[g, v', t] \rightarrow [g, v', t\text{-}t_1]$ --------- (b)
Applying contrapositive law to (b), we obtain
$[g, v, -t] \rightarrow [g, v, t_1\text{-}t]$ --------- (c)
Adding (2*t) to the time portion of equation (c),
$[g, v, t] \rightarrow [g, v, t+t_1]$ ------- (d)
Equations (a) and (d) indicate that *if the sequential machine enters a state that sets {g = v}, the assignment {g = v} would appear indefinitely at an interval of every t_1 time frames.* Thus, according to Definition 6, it can be concluded that the assignment $\{g = v'\}$ cannot be constant. Using a similar argument, since the dual recurrence relation exists on gate g, $\{g = v\}$ cannot be constant. □

Theorem 1: If a dual recurrence relation exists for a gate g, then both logic assignments $\{g = v\}$ and $\{g = v'\}$ are unachievable.

Proof: Since a dual relationship exists for gate g, $[g, v, t] \rightarrow [g, v, t\text{-}t_1]$ (with $t_1 > 0$). Also, since $\{g = v\}$ is not constant (Lemma 5), then according to the Lemma presented in [15], $\{g = v\}$ is unachievable. Using a similar argument, $\{g = v'\}$ is unachievable. □

It should be noted that if a dual recurrence relation exists for gate g, random logic simulation employed in [15] would return with $\{g = X\}$. In such a scenario, the Lemma in [15] would not be able to make any decision about the controllability characteristics of g.

However, Theorem 1 enables us to quickly learn that both logic assignments on gate g ($g = v$ and $g = v'$) are unachievable. As shown in the results, this additional knowledge results in a significant increase in the number of untestable faults identified.

Finally, we provide the following Lemma that applies Theorem 1 towards identification of untestable faults.

Lemma 6: If a dual recurrence relationship exists for gate g, all faults that require $\{g = v\}$ or $\{g = v'\}$ as a necessary condition for their detection would be untestable.

Proof: Since a dual recurrence relation exists for gate g, assignment $g = v$ is unachievable. Thus, by the hypothesis on unachievable nets presented in [15], all faults that require assignment $\{g = v\}$ as a necessary condition for their detection are untestable. Similar argument can be used for $\{g = v'\}$. □

Implementation of the Theorem: Implementation of Theorem 1 does not add any memory overhead or significant overhead in terms of execution time (unlike Lemma in [15] which required random logic simulation). The algorithm for implementation of Theorem 1 (on top of the Lemma proposed in [15]) is shown below:

For all gates assignments ($g = v$)

1. Perform transitive closure on $\{g = v\}$ to obtain Impl[g, v, 0]
2. If [g, v, 0] → [g, v, -t_1], and [g, v', 0] → [g, v', -t_1], mark $\{g = v\}$ and $\{g = v'\}$ as unachievable and skip step-3;
3. If [g, v, 0] → [g, v, -t_1]
   a. Perform logic simulation using k random vectors (In our analysis, k = 10,000)
   b. If $\{g = v'\}$ is achieved during simulation, mark $\{g = v\}$ as unachievable (according to Lemma in [15]);

Note that we also implement the Lemma proposed in [15] in our framework to estimate the increase in the number of untestable faults identified by Theorem 1 on top of Lemma in [15]. As we show in our results, the contribution of Theorem 1 is significant.

5. Experimental Results

The proposed techniques were implemented in C++ and experiments were conducted on ISCAS '85 and ISCAS '89 circuits on a 3.2 GHz, Pentium-4 workstation with 1 GB RAM, with Linux as the operating system. Table 1 illustrates the experimental results (# of untestable faults identified and execution times) obtained using our techniques. All results reported in Table 1 are on the same workstation for a fair comparison of execution times between our results and those obtained from [15]. For each circuit listed in Column 1, Columns 2 and 3 respectively report the # of untestable faults identified (UNT) and the time taken by the techniques proposed in [15]. Our engine implemented the techniques proposed in [15] along with extended-forward implications, and Theorem 1.

Column 4 shows the # of untestable faults identified for each of the circuits with only extended forward implications (no Theorem 1). It can be seen from Column 4 that for most of the circuits, the # of untestable faults identified increased when extended forward implications were incorporated into the framework. Column 5 shows the # of untestable faults identified only through the use of Theorem 1. It is easy to observe that Theorem 1 cannot aid in the identification of additional untestable faults for combinational circuits because the theorem uses *sequential recurrence relations* to identify untestable faults. For the larger sequential circuits (except for s38417), Theorem 1 significantly increases the # of identified untestable faults. For example, for s9234, s13207, s38584, Theorem 1 increased the # of identified untestable faults by multiples of thousands. Finally, columns 6 and 7 report the # of untestable faults identified and the time taken for analysis when both extended-forward implications and Theorem 1 are applied in conjunction.

Key observations made from Table 1 are:

- Through the techniques proposed in this paper, we could identify a lot of additional faults as untestable, with little overhead in terms of execution time. As an example, for circuit s13207 we could identify an additional 2915 untestable faults (over [15]) with an overhead of less than 120 seconds. Even for large circuits such as s38584, we could additionally identify more than 3000 additional untestable faults with an additional overhead of about 600 seconds. An overhead of 600 seconds is insignificant considering that deterministic sequential ATPGs would potentially spend hours targeting untestable faults for such designs.
- By enhancing Boolean learning through EF implications, we were able to identify several critical recurrence relationships which were missed earlier. Knowledge of these additional recurrence relations in turn increases the number of untestable faults considerably. As an example, for s13207, without EF implications, we could identify 2833 faults as untestable using Theorem 1 (Column 5). When EF implications were also used in conjunction with Theorem 1, new recurrence relationships were uncovered, which increased the number of untestable faults by almost 1000 (Column 6) via Theorem 1. Similar results can be observed for s9234, s15850, etc. These results illustrate the significance of EF implications in learning critical and useful sequential relationships which cannot otherwise be learnt.

Table 1: Experimental Results

Circuit	Results [15]		EF (Unt)	Thm.1 (Unt)	EF + Thm. 1	
	Unt	Time			Unt	Time
c1908*	9	0.93	9	9	9	1.2
c2670	93	0.72	101	93	**101**	1.15
c3540*	137	5.8	137	137	137	7.5
c5315*	58	2.39	59	58	**59**	2.9
c6288*	34	1.8	34	34	34	2.0
c7552	66	6.76	67	66	**67**	15.1
s386	63	0.65	65	63	**65**	0.7
s400	10	0.22	10	10	10	0.25
s641	59	0.20	59	59	59	0.24
s713	101	0.21	101	101	101	0.29
s1238	25	4.5	28	25	**28**	5.12
s1423	14	0.96	14	14	14	1.07
s5378	882	15.6	884	882	**884**	20.25
s9234	434	142.0	438	3490	**3602**	211.9
s9234.1	371	99.1	382	371	**389**	134.3
s13207	897	127.2	937	2833	**3812**	242.1
s13207.1	453	232.1	457	885	**889**	425.9
s15850	835	394.3	838	4411	**4636**	454.2
s15850.1	951	192.6	954	1043	**1045**	257.1
s38417	511	787.2	511	511	511	1014.1
s38584	2283	2187.4	2308	5405	**5616**	2758.5

Time values are specified in seconds
**All redundant faults in these combinational circuits were identified*

Finally, even though there exist circuits for which the # of identified untestable faults did not increase, the total # of implications identified for each circuit increased by an average of 15-20% by using EF implications.

Table 2 compares our results with some of the previously published work. Columns 2 and 3 report the # of untestable fault identified by MUST [13] (a combination of fault-independent and fault-oriented approaches) and the time taken for analysis, while columns 4 and 5 report results for SFT or single-fault-theorem [23] (based on ATPG). It can be seen that for some small sequential circuits (such as s1238 and s386), ATPG-based techniques can outperform our approach. This can be attributed to: a) complete

branch-and-bound nature of ATPG which may suit some small designs; b) implications learnt in our engine, like most other fault-independent techniques, do not represent the complete set of logic relations (learning the complete set of implications is computationally very expensive). However, for larger sequential circuits, our technique outperforms both SFT and MUST by large margins in terms of untestable faults.

Table 2: Comparison with other engines

Circuit	MUST[13]		SFT [23]		EF + Thm. 1	
	UNT	Time	UNT	Time	UNT	Time
c1908	8	8.7	-	-	**9**	1.2
c2670	97	6.32	-	-	**101**	1.15
c3540	127	37.4	-	-	**137**	7.5
c5315	**59**	16.5	-	-	**59**	2.9
c7552	62	24.4	-	-	**67**	15.1
s386	-	-	**70**	51.5	65	0.7
s400	9	5.36	**10**	1.7	**10**	0.25
s641	-	-	0	8.0	**59**	0.24
s713	38	2.9	38	8.9	**101**	0.29
s1238	58	11.42	**69**	14.8	28	5.12
s1423	**14**	6.96	**14**	25.4	**14**	1.07
s5378	622	232	470	862.0	**884**	20.25
s9234	-	-	524	4328.7	**3602**	211.9
s13207	1125	238	961	3237.2	**3812**	242.1
s15850	-	-	448	4197.5	**4636**	454.2
s38417	189	972.9	391	6995.5	**511**	1014.1
s38584	-	-	2142	9076.3	**5616**	2758.5

Time values are specified in seconds

To illustrate the impact of our contribution, we show the performance of deterministic ATPG for the faults identified as untestable by our tool. Any tool designed for untestable fault identification can enhance the performance of ATPG if:

(a) Such a tool can identify those untestable faults which ATPG fails to identify as untestable. In this case, performance gain to ATPG will be in terms of increased effective fault coverage or EFC (EFC = (#faults detected by ATPG)/ (#Total Faults – #untestable faults));

(b) Such a tool can identify untestable faults faster than ATPG. In this case, performance gain to ATPG will be in terms of improved speed (by ignoring already known untestable faults).

We show the importance of our tool for ATPG with respect to both these criteria. We use an in-house ATPG (based on PODEM [26], using SCOAP [27] testability measures) to target only those faults that are identified as untestable by our tool. For a fair estimation of the performance of ATPG on untestable faults, we perform ATPG in an incremental manner: each sequential circuit is first unrolled into one time frame, and ATPG targets all faults identified as untestable by our tool. Faults identified as untestable by ATPG within one time frame are dropped for further consideration. Next, the circuit is unrolled into two time frames, and only those faults which were not identified as untestable by ATPG in one time frame are now targeted. Again, faults identified as untestable in two time-frames are dropped when the circuit is unrolled into three time frames. Results in the context of ATPG's performance on untestable faults are shown in Table 3. Column 2 in Table 3 shows the number of untestable faults fed to ATPG for analysis. Note that ATPG is made to target only those faults which we have already identified as untestable using our tool. Columns 3 and 4 show the number of untestable faults identified by ATPG in one time frame, and the corresponding time taken. Next, Columns 5 and 6 show similar results when ATPG was performed on the two time frame unrolled circuit, and Columns 7 and 8 show results for three time frames. Finally, Columns 9 and 10 show the total number of faults identified as untestable by ATPG over three-time frames and the total time taken by ATPG to analyze untestable faults. Key observations made from Table 3 are:

- For small circuits, such as s386 and s400, ATPG is also able to identify all faults as untestable (shown in bold in Column 9). Not only that, ATPG is also faster than our fault-independent tool. This is a result of the small-search space that ATPG has to explore for such circuits (these circuits have few inputs and few flip-flops). Thus, the branch and bound nature of ATPG works well for such small circuits.
- For medium sized circuits such as s641-s5378, ATPG is not able to identify all faults as untestable. Our tool can identify several untestable faults which ATPG either aborts or falsely detects in the unrolled circuit (untestable faults get detected because flip-flops are converted into primary inputs in unrolled circuits). Moreover, ATPG may spend significantly more time than taken by our tool to analyze these faults (e.g. s5378).
- For large circuits such as s9234-s38584, ATPG identifies a very small fraction of faults as untestable. The number of untestable faults identified by our tool are more than that identified as untestable by ATPG by several factors (e.g. s9234, s15850 etc.). Moreover, the amount of time spent by ATPG on untestable faults is significantly more compared to the time spent by our tool on these faults (e.g. s38584, s9234 etc).
- As the number of time frames is increased linearly, the search space for ATPG grows exponentially. Thus, ATPG takes more time to analyze untestable faults. Also, with an increase in time frames, ATPG identifies fewer untestable faults (ATPG aborts on most faults when the circuit is unrolled for 3 or more frames).

Experimental results show that our tool identifies those untestable faults which ATPG fails to identify as untestable; also for large circuits, ATPG spends exponential amount of time targeting the faults identified as untestable by our tool. Thus, our tool can benefit ATPG in both increasing the effective fault coverage and in increasing the efficiency of ATPG through *a priori* knowledge of untestable faults.

6. Conclusion

In this paper we introduced a new class of implications called *extended forward implications (or EF implications)*. We also introduced the concept of I-Frontier and used properties associated with implications and I-Frontier to optimize the process for identifying these EF implications. We proposed a new theorem which utilizes recurrence relations of sequential implications to determine controllability characteristics of nets in a circuit. With the aid of the new theorem in conjunction with EF implications, significantly more untestable faults were identified for many circuits. These results can be of prime importance to many EDA tools (ATPG, fault-simulators, etc.). We showed the significance of our tool in terms of its impact on the performance of ATPG. Since we were able to identify several untestable faults which are missed by ATPG, our tool can benefit ATPG in terms of both coverage calculations and speed.

7. References

[1] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system," *IEEE Trans. Computer-Aided Design*, vol. 7, Jan 1998, pp. 126–137.

[2] A. El-Maleh, M. Kassab and J. Rajski, "A fast sequential learning technique for real circuits with application to enhancing ATPG performance," *Proceedings of DAC*, June 1998, pp. 625 – 631.

Table 3: Performance of ATPG for Untestable Faults

Circuit	# UNT Faults	ATPG* (1-TF)		ATPG* (2-TF)		ATPG* (3-TF)		Total (ATPG)	
		UNT	Time	UNT	Time	UNT	Time	UNT	Time
s386	65	0	0.01s	65	0.08s	-	-	**65**	0.09s
s400	10	6	0.01s	4	0.01s	-	-	**10**	0.02s
s641	**59**	0	0.02s	0	0.42s	0	1.6s	0	2.04s
s713	**101**	38	0.16s	0	0.45s	0	10.2s	38	10.81s
s1238	**28**	27	0.04s	0	0.01s	0	0.01s	27	0.06s
s5378	**884**	35	0.53s	286	3m45s	100	9m54s	421	13m39s
s9234	**3602**	396	3m25s	52	20m53s	3	50m13s	451	71m31s
s9234.1	**389**	311	2m38s	14	4m42s	0	7m18s	325	14m32s
s13207	**3812**	145	16.1s	776	1m1s	36	11m15s	957	12m32s
s13207.1	**889**	115	8.6s	241	14.4s	9	15m2s	365	15m25s
s15850	**4636**	375	1m5s	49	9m32s	4	72m2s	428	82m39s
s15850.1	**1045**	352	46s	37	1m42s	4	4m22s	393	6m50s
s38417	**511**	123	2m18s	43	7m20s	115	29m48s	281	39m26s
s38584	**5616**	1444	39s	300	61m31s	37	85m10s	1781	147m20s

*Time values are specified in minutes and seconds (**m**: minutes, **s**: seconds)*
**Number of backtracks used in ATPG: 100,000*

[3] P. Tafertshofer; A. Ganz and K. J. Antreich, **"**IGRAINE-an Implication GRaph-bAsed engINE for fast implication, justification, and propagation," *IEEE Transactions on CAD of Integrated Circuits and Systems*, August 2000 pp. 907 – 927

[4] S. Kajihara, K. K. Saluja and S. M. Reddy, **"**Enhanced 3-valued logic/fault simulation for full scan circuits using implicit logic values," *Proc. European Test Symposium,* May 2004, pp. 108-113

[5] M. Enamul Amyeen, W. K. Fuch, I. Pomeranz, V. Boppana, "Implication and evaluation techniques for proving fault equivalence," *Proc. IEEE VTS,* April 1999 pp. 201 – 207

[6] D. Paul, M. Chatterjee and Dhiraj K. Pradhan, "VERILAT: Verification Using Logic Augmentation and Transformations," *IEEE Trans. On CAD of Integrated Circuits and Systems*, vol. 19, no. 9, Sept. 2000

[7] J. Marques-Silva and T. Glass, "Combinational equivalence checking using satisfiability and recursive learning," *Proc. Design, Automation and Test in Europe Conf.*, March 1999, pp.145 – 149

[8] R. Arora, and M. S. Hsiao, "Enhancing SAT-based bounded model checking using sequential logic implications," *Proc. Of Intl' conference on VLSI Design*, Jan. 2004, pp. 784-787

[9] H. Ichihara, K. Kinoshita, "On acceleration of logic circuits optimization using implication relations," *Proc. Asian Test Symposium*, Nov. 1997 pp. 222 - 227

[10] W. Kunz, D. Stoffel, and P. R. Menon, "Logic optimization and equivalence checking by implication analysis," *IEEE Trans. On CAD of Integrated Circuits and Systems*, Volume: 16 , Issue: 3, March 1997, pp.266 – 281

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

[12] M.A. Iyer, D.E. Long and M. Abramovici, "Identifying Sequential Redundancies without Search," *Proceedings of DAC,* 1996, pp. 457-462

[13] Qiang Peng, M. Abramovici and J. Savir, "MUST: Multiple-Stem Analysis for Identifying Sequentially Untestable Faults," *Int'l Test Conference*, 2000. pp. 839-846

[14] M. S. Hsiao, "Maximizing Impossibilities for Untestable Fault Identification," *Proc. IEEE Design Automation and Test in Europe Conf.*, March 2002, pp. 949-953

[15] M. Syal, M. S. Hsiao, "Untestable Fault Identification Using Recurrence Relations and Impossible Value Assignments," *Proc. Of Intl' conference on VLSI Design*, Jan. 2004, pp. 481-486

[16] M. H. Schulz and E. Auth, "Improved deterministic test pattern generation with applications to redundancy identification," *IEEE Trans. Computer-Aided Design.*, vol. 8, pp. 811–816, July 1989.

[17] W. Kunz and D. K. Pradhan, "Accelerated dynamic learning for test pattern generation in combinational circuits," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 684–694, May 1993.

[18] J. Rajski and H. Kox, "A Method to Calculate Necessary Assignments in ATPG," *Proc. Int'l. Test Conf.* 1990, pp. 25-34

[19] S.T. Chakradhar and V. D. Agarawal, "A transitive closure algorithm for test generation", *IEEE Transactions on CAD*, 1993, pp. 1015 - 1028

[20] 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,* pp. 1149-1158. Sept 1994,

[21] 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.

[22] K.T. Cheng, "Redundancy Removal for Sequential Circuits without Reset States," *IEEE Tran. On CAD*, vol. 12, no. 1, Jan. 1993, pp 13-24

[23] V. D. Agrawal and S. T. Chakradhar, "Combinational ATPG Theorems for Identifying Untestable Faults in Sequential Circuits," *IEEE Trans. On CAD,* vol. 14, no. 9, Sept. 1995, pp. 1155-1160.

[24] S. M. Reddy, Irith. Pomeranz, X. Lim and Nadir Z. Basturkmen, "New procedures for identifying Undetectable and Redundant Faults in Synchronous Sequential Circuits," *Proc. VLSI Test Symposium, 1999.* pp. 275 -281.

[25] M. Syal, M. S. Hsiao, "A Novel, Low-Cost Algorithm for Sequentially Untestable Fault Identification," *Proc. IEEE Design Automation and Test in Europe Conf.*, March 2003, pp. 316-321.

[26] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. on Computers,* vol. C-30, no. 3, March 1981, pp. 215-222.

[27] L. H. Goldstein, "Controllability/Observability Analysis of Digital Circuits," *IEEE Trans. on Circuits and Systems,* vol. CAS-26, no. 9, Sept. 1979, pp. 685-693.