LFSR Seed Computation and Reduction Using SMT-Based Fault-Chaining*

Dhrumeel Bakshi and Michael S. Hsiao
Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 24061, USA
{dvb, mhsiao}@vt.edu

***Abstract*—We propose a new method to derive a small number of LFSR seeds for Logic BIST to cover all detectable faults as a first-order satisfiability problem involving extended theories. We use an SMT (Satisfiability Modulo Theories) formulation to efficiently combine the tasks of test-generation and seed-computation. We make use of this formulation in an iterative seed-reduction flow which enables the "chaining" of hard-to-test faults using very few seeds. Experimental results demonstrate that up to 79% reduction in the number of seeds can be achieved.**

***Index Terms*—LFSR Reseeding, Logic BIST, Test generation, Satisfiability Modulo Theories.**

I. Introduction

LBIST (Logic Built-in Self Test) refers to a technique which enables a chip to test itself. As the size and complexity of functional logic being built into chips keep increasing, the cost of conventional ATE (Automated Testing Equipment) based testing becomes more challenging. LBIST can help to either reduce ATE complexity or eliminate ATEs altogether. In BIST, generation of test patterns and analysis of output responses are both performed on-chip. BIST allows for at-speed testing of the chip which is needed to test performance-related defects. Additionally, as test mechanisms are embedded into the chip, BIST has a better access to the core logic to be tested as compared to external testing. Finally, a self-testable chip can test itself even after it is part of a system.

While BIST offers the aforementioned advantages, there are challenges that come with it. Insertion of BIST circuitry on chip may reduce the circuit performance and increase area from the test structures such as the TPG (Test Pattern Generator) and ORA (Output Response Analyzer). To reduce the costs of BIST, low hardware-overhead TPG structures capable of producing pseudo-random patterns are preferred. However, the patterns produced by these structures are not optimal. Most critically, achieving a high fault-coverage is a challenging problem for circuits containing random-pattern-resistant and/or hard-to-test faults. Various solutions have been proposed to tackle this problem. Some solutions involve modification of the circuit by redesigning or by test-point insertion to improve the fault coverage. Others propose the use of *weighted-random patterns* to increase the probabilities of detecting random-pattern-resistant faults ([1], [2], etc.). A third category uses what is commonly known as *mixed-mode BIST*. This technique involves testing the circuit first using pseudo-random patterns followed by deterministic methods to cover faults missed by the random patterns.

*This work was supported in part by SRC grant 2011-TJ-2134.

This paper proposes novel techniques for a sub-class of mixed-mode BIST called *reseeding*, in which test-patterns are encoded as initial states or *seeds* of a TPG based on an LFSR (Linear Feedback Shift Register). A great amount of work has been done on computing seeds for LFSR-based pattern-generation. Most of these techniques attempt to compute seeds from pre-computed ATPG patterns. We propose a new way to look at the problem of reseeding, wherein the seeds are computed by considering target faults rather than target test patterns. We shall show that this approach can help us cover all faults in the circuit using very few number of seeds.

In our proposed method, we cast the problem as a first-order satisfiability problem involving extended theories. For solving this kind of a constraint-problem, we make effective use of SMT-solving (Satisfiability Modulo Theories). SMT allows us to encode our problem by combining the domains of Boolean logic and the theory of Bit-Vectors, allowing us to try to chain *faults* instead of pre-computed vectors. Results show that we can achieve high quality solutions with low computational costs - up to 79% reduction in seeds can be achieved.

The rest of the paper is organized as follows: we shall first give some background and specifics of the LFSR-reseeding problem followed by a discussion of the motivation behind our method in Section II. The SMT formulation and complete flow for computing LFSR seeds will be described in Section III. Section IV presents the experiments and results.

II. Background and Overview

LFSRs are a common choice for pseudo-random pattern generation in BIST. The main advantages of LFSRs are their low hardware overhead and high degree of randomness in the vectors generated. For example, a maximal-length LFSR of length 40 can produce a sequence of period greater than one trillion vectors, while also ensuring that the generated vectors satisfy most random-distribution tests. However, this also means that LFSR patterns are far from optimal for testing VLSI circuits. In order to achieve a suitable fault-coverage, either exhaustive or a very long test sequence would need to be applied. Thus, good quality patterns, such as deterministic patterns, are typically 'scattered' over the huge LFSR pattern space. The method of reseeding aims to alleviate exactly this problem, by allowing a trade-off between on-chip hardware overhead and test lengths for a desired fault coverage. LFSR patterns, though predictable, are seemingly *chaotic*. The randomness property also makes it harder for us to reason about long sequences of LFSR patterns at a time.

LFSRs can be used in various configurations for pattern generation and application. In this paper, we have used the *Fi-*

bonacci LFSR configuration (aka. *external* LFSR). We assume a test-per-clock scheme, where the contents of the LFSR at every cycle constitute exactly one test-vector to be applied to the circuit under test. However, our method is applicable to other LFSR configurations (e.g., Internal or Galois-type LFSR), as well as to other BIST configurations (e.g., STUMPS), via changes to the SMT formulation discussed in Section III-A.

A. Related work

Mixed-mode BIST has been shown to be an effective approach to alleviate the fault-coverage problem in BIST. The biggest challenge in this method is the hardware overhead required to store the seeds on chip, especially if the number of seeds is large. The problem of deterministic testing using LFSRs has been widely studied. Seed computation for deterministic testing was first described in [3]. In [4], [5], techniques have been proposed to encode deterministic test-patterns as seeds of an LFSR whose configuration (polynomial) can be altered during test. Techniques for enhancing test-generation for reseeding schemes have been proposed in [6], [7]. Other methods for reducing the number of required seeds rely on identification and reduction of seeds by simulation and seed ordering/encoding methods [8], [9]. A large number of methods such as [10] describe techniques for efficient hardware encoding of the seeds.

Most of the previous methods for seed computation attempt to compute seeds based on pre-computed test patterns. This involves grouping and ordering of test-patterns followed by solving of linear-equations to get a seed for each group. There exist various problems with this approach which will be discussed in the next subsection. Our method removes these problems by computing seeds for faults instead of pre-computed vectors. The method described in [11] works by identifying the exact location of test-patterns in the LFSR state-cycle by computation of Discrete Logarithms. However, these logarithms need to be computed for all possible vectors for target faults, which is impractical for modern designs.

A recent work attempts to combine the processes of test-generation and seed-computation using SMT solving [12]. As will be discussed in Section III-A, the method in [12] is time-consuming due to large SMT formulae and search space involved. The method also has challenges of fault-masking due to considerations of multiple fault injections in each frame. In this paper, we propose a hybrid method based on simulation and SMT analysis which eliminates these problems and makes the search tractable for larger circuits.

B. Motivation/Idea

Most of the existing approaches to the computation of LBIST seeds attempt to chain a set of pre-determined test vectors. There are inherent problems with such approaches, which we describe using the following terms. We use the term *LFSR* to represent an LFSR configuration along with its polynomial.

L-Distance Given a test set, T, comprising K vectors $\{T_1, \ldots, T_K\}$ (the vectors may contain don't-care bits), and an LFSR, we define the *L-Distance* of the set T with respect to the given LFSR to be no greater than an integer L if and only if there exists a *seed*, S, which can generate all the vectors of T in any order, when the LFSR is run for L number of cycles.

F-Distance Given a set of K faults, F, $\{F_1, \ldots, F_K\}$, and an LFSR, we define the *F-Distance* of the set F with respect to the LFSR to be no greater than an integer L if and only if there exists a *seed*, S, such that the LFSR can produce a set of test-vectors sufficient to detect all faults in F, in any order, when run for L cycles starting from S,

Chainability A set of test-vectors (faults) is defined to be *chainable* within L cycles of a given LFSR if and only if the *L-Distance (F-Distance)* of the set is no more than L.

Note that the term *F-Distance* implicitly assumes that a specific fault-model has been selected for consideration. We have assumed the LFSR to be used to generate tests using a *test-per-clock* scheme where the contents of the LFSR in each cycle represent one bit-vector to be used as a test-vector. The terms may be easily generalized for other BIST architectures.

Given a stuck-fault in a circuit, there usually exist multiple vectors that can detect the fault. For a set of hard-to-chain faults, the choice of the exact vectors to use for chaining these faults has a direct impact on the number of seeds needed. Realizing that LFSR vectors are chaotic (but predictable from an initial seed), we can see that even making slight changes in the test-vector bits has the potential to alleviate or magnify the problem of *vector-chaining*. Thus, there may exist alternatives to vector-chaining and give smaller set of seeds.

In this work, we propose a method of seed-computation which tries to chain faults rather than pre-computed vectors. We believe this to be a more effective way to view the problem of seed computation. We use an SMT formulation for *fault-chaining* and integrate it into a broader cost-effective method for computing LFSR seeds. This method allows us to select a small set of very high quality seeds. In effect, by considering the chainability of faults (*F-distance*) instead of chainability of vectors (*L-distance*), we offer a tighter integration of the test-generation process with seed computation. In addition, this method overcomes the problems associated with vector-chaining and gives quality solutions in significantly smaller computational costs.

III. Iterative Seed Reduction

A. SMT-based fault-chaining

We first describe our SMT formulation for test generation under LFSR constraints. Given the circuit under test and a set of single-stuck-faults to *chain*, we build an SMT model. Running an SMT solver on this model can help us estimate the *F-Distance* of the set of faults. In this work, we model the problem by combining the domains of Boolean (propositional) logic and quantifier-free bit-vector theory ($\mathcal{QF}$-$\mathcal{BV}$).

Suppose we have the gate-level netlist of a circuit, and a set of K single-stuck-faults $F = \{F_1, \ldots, F_K\}$. Also given to us is the LFSR-structure, and a bound L which is a heuristic bound set on the *F-Distance*. The SMT formulation to determine whether the set F has an *F-Distance* $\leq L$, is a formula that consists of the following sub-formulae:

(a) LFSR constraints $\mathbf{C}$ in the theory of bit-vectors.
(b) Fault detection formula $\mathbf{D_i}$ for every fault F_i.
(c) Constraints to connect every $\mathbf{D_i}$ to the bit-vector variables in $\mathbf{C}$.

In (a), $\mathbf{C}$ consist of L bit-vector variables $S_1, \ldots, S_L$. S_1 represents the *seed*, while the rest represent the contents (*state*) of the LFSR that can be derived in successive cycles. We know that an external-LFSR can be seen as a circular shift-register that shifts right in every cycle, with the leftmost bit defined by the feedback XOR-network. Thus, every bit-vector variable S_i is defined in terms of S_{i-1} using a bit-vector right-shift operation. Additionally, the leftmost bit of S_i is a function of some bits of S_{i-1}. These constraints are easily expressible in SMT, since $\mathcal{QF}$-$\mathcal{BV}$ allows for *extraction* of certain bits from bit-vectors, and specifying Boolean propositions using them. The initial seed, S_1, is also an unspecified variable, and each one of the S_i variables can potentially detect one or more of the faults in F.

Next, in (b), each formula $\mathbf{D_i}$ comprises a formula $\mathbf{G_i}$ (*good circuit*) and a formula $\mathbf{B_i}$ (*faulty circuit*). The constraints in both $\mathbf{G_i}$ and $\mathbf{B_i}$ are modeled using Boolean propositions, using the gate-level structure of the circuit. $\mathbf{G_i}$ models fault-free operation while $\mathbf{B_i}$ is the same formula with the node at the fault location being replaced by constant-value drivers. Both $\mathbf{G_i}$ and $\mathbf{B_i}$ receive the same input vector T_i. The outputs of the two circuits are then fed, pair-wise, via XOR gates, effectively forming a *miter circuit*. In this miter (of G_i and B_i), the output is *true* if and only if the vector T_i causes at least one output of $\mathbf{G_i}$ and $\mathbf{B_i}$ to differ. Further optimizations are made to reduce the number of variables in $\mathbf{B_i}$ as in SAT-based test generation methods such as [13]. In particular, we make arrangements to remove variables for gates which do not participate in the *injection* or *propagation* of fault F_i, by pre-computing the fan-in and fan-out *cones of influence* of F_i.

The constraints in (c) effectively model the mapping between vector T_i and the S_i LFSR state vectors. We add constraints to ensure that every test vector T_i is covered by at least one vector that can be generated by the LFSR. Formally, the constraint

$$\bigwedge_{i=1}^{K} \left[\bigvee_{j=1}^{L} (T_i = S_j) \right]$$

ensures that every bit-vector T_i is equal to one of the S_j variables. Along with the detection constraints $\mathbf{D_i}$, this ensures that every fault in set F is detected within L cycles of the LFSR starting from seed S_1. To further constrain the search space for the SMT solver, we add the constraint $\left[\bigvee_{i=1}^{K} (T_i = S_1) \right]$ which ensures that the computed seed (S_1) detects at least one fault in F. This constraint helps avoid (and prune from the search space) those variable assignments in which LFSR cycles are 'wasted' due to fault-detection starting at a vector S_i such that $i > 1$.

Note that the above SMT model is satisfiable if and only if all faults in set F are testable and the *F-Distance* of F is $\leq L$. Hence, the value of S_i in the satisfying assignment gives a seed that can *chain* the faults in F within L cycles from the computed seed. Also, an *unsat* returned by the solver would indicate that the *F-Distance* of F is greater than L.

The above SMT formulation for *fault-chaining* is the core of our method for seed computation and reduction. It promises significant advantages over previous approaches. First, our method does not require test-vectors to be pre-determined and pre-ordered, as is frequently required by methods based on solving linear equations. In fact, it does not even impose any restrictions on the order in which faults in set F are to be detected. Since test-vectors are not required to be fixed in advance, the method implicitly allows for a single LFSR vector to potentially detect multiple faults. It also allows for 'don't care' cycles between test-vectors in the LFSR cycle, where the LFSR vector does not detect any of the faults under consideration. Consequently, the method is very general and gives us a better chance of finding a seed to cover a given set of faults. Additionally, if the SMT solver returns an *unsat*, it suggests that the faults in F are indeed *hard-to-chain*.

We now compare our method with previous work on seed computation using SMT. The work in [12] also attempts to combine the processes of seed computation and test-generation. However, the problem is cast as one big SMT formula involving L time-frames, resulting in a resource expensive call to the SMT solver. In contrast, our method is more cost-effective as it no longer needs to have L copies of the circuit but is only dependent on the number of faults.

Given the SMT formulation, we need to identify the K faults that we wish to chain. Although any set of K faults can be chained, we would like to be more clever about it to reduce the number of seeds obtained at the end. The next section describes how we select these faults to be chained.

B. Fault-selection via clustering using independence graph

In order to reduce the number of seeds, a seed that can chain those faults currently undetected by any single seed would be helpful. We describe a fault-selection step which is critical to the efficacy and efficiency of our overall method.

With our SMT formulation described in the preceding section, the proposed *fault-chaining* method considers K faults within an LFSR 'window' of length L cycles. However, a seed returned by the SMT solver is usually able to detect additional faults. We can determine all faults that a seed covers by simulating the LFSR (starting from the seed) on the entire faultlist. This also helps us identify faults which were not explicitly considered while computing the seed, yet are detected by the seed sequence. The idea behind fault-selection is - we aim to generate additional seed candidates to chain those faults currently not simultaneously detected by any existing seed. The newly computed seed may render some previously computed seeds unnecessary, thereby reducing the cardinality of the set of seeds.

At every iteration in our method, we start with a certain number of seed candidates in our pool. We fault-simulate the candidates to obtain the faults detected by each candidate. This information is represented in a fault-dictionary, which is a binary matrix. A sample dictionary with 3 seed candidates for a circuit with 5 faults is shown:

	F_1	F_2	F_3	F_4	F_5
Seed 1	1	0	1	1	1
Seed 2	0	1	1	1	0
Seed 3	1	0	1	0	0

In this example, seed 3 covers faults F_1 and F_3; fault F_4 is covered by Seed 1 and Seed 2, etc. We use this dictionary to determine the sets of faults to be chained in the next iteration of SMT solving. For this purpose, we build a *fault-independence graph*. We define two faults in the dictionary to be *independent* if there currently exists no seed candidate which can detect both faults. We identify all pairs of independent faults. Since each column in the dictionary represents one fault. the *dot product* of any two columns gives the number of seed candidates which detect both faults. By computing the dot product for every pair of faults, we identify independent faults (i.e., the columns with dot product = 0).

Algorithm 1 Clique-partitioning

```
Inputs: Independence graph $G$, Clique size bound $B$
Output: Set of cliques $C$
 1: $C \leftarrow \phi$
 2: for all nodes $N \in G$ do
 3:   $Q \leftarrow \phi$
 4:   if $N$ is marked covered OR degree($N$) $\leq 1$ then
 5:     skip $N$ and go to next node
 6:   else
 7:     $Q \leftarrow \{N\}$
 8:     mark $N$ covered
 9:     $common \leftarrow \{n \mid n$ is a neighbor of $N\}$
10:     while $|Q| < B$ AND $|common| > 0$ do
11:       $M \leftarrow$ max degree node from $common$
12:       $Q \leftarrow Q \cup \{M\}$
13:       $common \leftarrow common \cap \{$neighbors of $M\}$
14:     end while
15:     $C \leftarrow C \cup Q$
16:   end if
17: end for
```

Using the dot product computed for all fault-pairs in the dictionary, we can now build a fault-independence graph such as the one shown in Figure 1. Each node in the graph represents one fault. We add an edge between two faults if and only if their corresponding dot product is 0. With the independence-graph, we identify *cliques* in the graph. A clique of size N in the independence graph represents a set of N faults such that no pair of faults are detected by any seed candidate in the pool. It has been shown in [14] that the size of the largest clique (i.e., clique number) in the independence graph is a lower bound on the single-detection test set size.

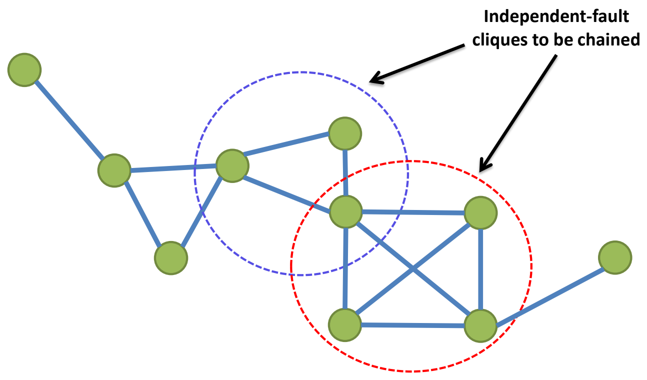

Fig. 1: Example of fault independence-graph

Generalizing this result, we can see that the clique number of the independence graph shown in Figure 1 gives the minimum number of seeds if we were restricted to using only those seed candidates from the current pool. This observation suggests that we are likely to benefit from chaining together those faults which form cliques in the independence graph. Additionally, due to reasons described earlier, we want to limit the sizes of these cliques. Thus, the next step in our method is the partitioning of the independence graph into non-singular cliques of bounded size. The clique partitions need not be mutually independent. Algorithm 1 describes a greedy approach used to identify cliques from the graph such that every fault node with a non-zero degree is included in at least one clique.

C. Seed reduction by set-covering using ILP

As seed candidates are computed using our SMT formulation and added to the pool, the fault-dictionary size keeps growing. In every iteration, following fault-simulation of the new seed candidates, we use the dictionary to determine a small subset of seeds that are sufficient to cover all the faults in the circuit. The problem of selecting the minimum number of seeds from the dictionary that can detect all faults can be cast as a standard *Set Covering* problem, with the columns being elements to cover and each row representing a subset of elements it covers.

We use an Integer Linear Programming (ILP) formulation for computing the set cover. Modern ILP solvers are quite efficient at solving optimization problems of this type. The formulation has been widely used and discussed in literature. Optimization procedures and LP-solving have been shown to be very effective for test compaction procedures (e.g., [15], [16]). The integer linear programs for selecting the set cover of seeds is described below:

$$\text{minimize} \sum_{i=1}^{N_S} x_i$$

under the constraints:

$$\forall \text{ faults } F_j, \quad \left(\sum_{i \in D_j} x_i \right) \geq 1$$

where:

$$\begin{aligned} N_S &= \text{Number of seeds in dictionary} \\ D_j &= \{i \mid \text{Seed } i \text{ covers fault } F_j\} \\ x_i &= \begin{cases} 0 \;\Leftrightarrow\; (\text{Seed } i \text{ excluded from solution}) \\ 1 \;\Leftrightarrow\; (\text{Seed } i \text{ included in solution}) \end{cases} \end{aligned}$$

D. Overall procedure

The overall flow for computing a small set of sufficient seeds is illustrated in Figure 2. An initial set of seeds (could be random) is used. Alternatively, one may start from the test set from an ATPG. This initial set of seed candidates is used to build a seed-fault dictionary as described in Section III-B, by simulating the LFSR for M cycles for each seed candidate. An ILP solver can be applied to determine the set-cover, which serves as an upper bound on the number of seeds needed. However, we use this set to construct the independence graph and compute independent faults.

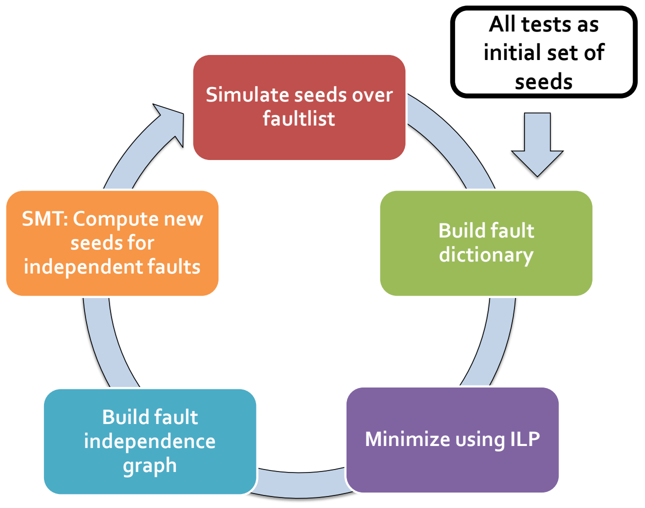
All tests as initial set of seeds
Simulate seeds over faultlist
Build fault dictionary
Minimize using ILP
Build fault independence graph
SMT: Compute new seeds for independent faults

Fig. 2: Iterative seed-reduction procedure

Next, we move into the main thrust of our framework in which we attempt to chain the currently unchained faults into a common seed. The faults to be chained are identified from the independence graph, as described in Section III-B. SMT solving as described in Section III-A gives us one seed for every group of faults chainable within L cycles of the LFSR. We add the new seeds to our current pool of seed candidates, and simulate these seeds to augment our fault-dictionary. In any iteration, we do not replace the seeds from the previous iterations. Since we keep adding new (possibly better) seeds to the pool, the size of the solution across iterations monotonically decreases. We can repeat this iterative procedure until either a sufficiently low number of seeds is selected by ILP, no edges exist in the independence graph, or the number of seeds is constant for a number of iterations. The following parameters are used by the method:

- LFSR - We choose an LFSR of length $= N_I$, the number of primary inputs of the circuit.
- LFSR polynomial - We choose a *primitive polynomial* from [17] to yield a maximal-length LFSR.
- Clique-size bound B.
- LFSR window size of L cycles for SMT based chaining.
- Fault-simulation window length of M cycles - the number of cycles after loading each new seed.

IV. Experiments and Results

We have evaluated the proposed method on ISCAS'85, ISCAS'89 and ITC'99 benchmark circuits, and three circuits from OpenCores. Full-scan versions of sequential circuits were used. The SMT solver used for the fault-chaining step was Z3 v3.2 [18]. The ILP used for solving the set-covering was Gurobi [19]. All experiments were performed on an Ubuntu Linux workstation with an Intel ® Core ™ i7 3.33 GHz CPU, and 6 GB of memory.

Table I reports the results. A postfix f with a benchmark name indicates full-scan version of the circuit. For each circuit, the number of inputs is first reported, followed by the number of non-easy testable faults - we perform preliminary reduction by removing "easy" faults (faults detected by every random seed). Next, we report M, the size of the fault-simulation window, chosen empirically based on the value of N_I. We report three sets of results relevant to our method. First, we generate 1000 random seeds, simulate the LFSR for M cycles followed by fault-simulation as described earlier to build a fault-dictionary. Column 5 reports the minimum number of seeds required from 1000 random seeds to cover all faults using an ILP solver. Empty entries indicate that random seeds were insufficient to detect all faults. Next, we use an ATPG tool to generate tests for the entire faultlist. Each of the tests is then assumed to be an LFSR seed. This set of seeds is simulated over M cycles and reduced like the set of random seeds just described. Columns 6 and 7 report the number of tests and the required number of seeds, respectively. Since an ATPG is used, all faults could be detected. We use this set of deterministic tests as the starting point for our method. The last 3 columns report, for our overall procedure, the number of iterations, total runtime and size of the final solution (number of seeds). Our method ends when the number of seeds does not reduce for 3 consecutive iterations. In these experiments we have set the clique-size bound B to 3 and the LFSR window size for SMT (L) to 100. If the SMT-solver returns *unsat* for more than 90% of the number of cliques to be chained, L is increased by 50 and the SMT solver is called again.

It can be observed that our method produces very few seeds, with a low runtime. Many of the final solution sizes are a single-digit number of seeds. The total number of bits to be stored on-chip may be calculated by multiplying the solution size by N_I. The difference between the number of seeds before and after the method (columns 7 vs. 10) demonstrates the strength of fault-chaining. For example, for c7552, random seeds could not detect all faults, and ILP could select 37 out of 267 pre-computed ATPG vectors as the seeds that can detect all faults, but our method is able to obtain 10 seeds in 5 iterations (4207 seconds). This is a seed-reduction of 78%. Likwise, in s838f, we achieved a 79% reduction in the number of seeds (34 down to 7 seeds). When compared with the 1000 random seeds, results are also noteworthy. In c2670, for example, with 1000 random seeds, the ILP can select 18 as the number of seeds that can detect all faults. This is better than the deterministic seeds which required 29 from the 150 vectors. However, our method can generate 4 seeds that can detect all faults in less than 11 minutes of computation.

In Table II, we compare our results to a previous work on LFSR reseeding using SMT solving [12]. SMT-based seed computation was performed using an LFSR window size of $n + 20$, where n is the number of faults under consideration reported in Column 3. To make the comparison fair, we use the same value for LFSR-window ($L = n+20$). Additionally, we set the fault-simulation window size M equal to L. Even with virtually everything else being the same, the proposed method is able to produce equal or better results in most cases, with multiple orders of magnitude in speedup. For example, in circuit s1488f, 13 seeds were obtained in [12] taking 159,219 seconds, while our approach needed only 8 seeds in just 45 seconds. As discussed earlier, this may be attributed to the fact

TABLE I: Experimental results on ISCAS, ITC and OpenCores circuits

Circuit	# PIs (N_I)	# Faults	M	1000 Rand Seeds	All tests as init. seeds		Proposed Fault chaining		
				ILP Sol.	#Tests	ILP Sol.	# Iterations	Runtime(s)	Sol.
s420f	35	430	1k	-	70	8	4	12	**3**
s641f	54	467	2k	8	85	7	4	27	**3**
s713f	54	543	2k	6	85	9	4	23	**3**
s838f	67	857	3k	-	129	34	7	434	**7**
s1196f	32	1242	1k	7	198	7	5	157	**5**
s1238f	32	1286	1k	6	222	8	3	164	**6**
c2670	233	2630	10k	18	150	29	3	646	**4**
s9234f	247	6475	10k	-	592	31	11	4734	**17**
c7552	207	7419	10k	-	269	37	5	4207	**10**
s15850f	611	1273	10k	-	388	66	12	25417	**19**
b07f	50	1100	1k	5	73	7	5	61	**3**
b12f	126	2766	3k	13	199	13	4	825	**6**
b14f	275	1970	10k	-	497	219	18	43534	**74**
b15f	485	7982	10k	-	523	82	9	48230	**33**
spi	277	1375	10k	25	685	24	6	7296	**19**
tv80	372	7109	10k	-	1111	93	8	34954	**43**
systemcaes	929	5614	5k	12	251	13	9	7670	**9**

'-': the seeds were insufficient to detect all faults

that [12] relies on expensive calls to the SMT solver.

We have discussed stuck-faults in a test-per-clock scheme. For circuits with a large number of scan elements, a test-per-scan or STUMPS architecture with a smaller LFSR is more suitable. Given the flexibility of expressing bitvector constructs in SMT, the formulation may be modified to consider the relevant test-vectors in any architecture. Note that although fault-simulation takes a large portion of the runtime, it can be easily and effectively parallelized. The benefits of clique-based independent-fault identification may be utilized by any other seed-computation method. Similarly, the SMT formulation may aid seed-computation in other seed-encoding and reduction schemes - the benefit is that our formulation is more likely to find a seed for a group of hard-to-chain faults.

TABLE II: Experiments: Comparison with [12]

Circuit	# Faults	# Inj. (n)	M	[12]		Fault chaining	
				Runtime	Sol.	Runtime	Sol.
c880	942	21	41	6071	**3**	4	**3**
c1355	1566	28	48	30917	**4**	22	**4**
c1908	1870	84	104	73002	12	117	**7**
s953f	1079	117	137	183809	15	891	**7**
s1423f	1501	38	58	92346	9	78	**6**
s1488f	1486	40	60	159219	13	45	**8**

V. Conclusion

We have presented a new technique for LFSR seed computation to cover all detectable stuck-faults in a circuit. The method works by iteratively identifying groups of faults to be chained into a common seed, and then using SMT to compute a new seed candidate. The proposed method computes seeds by chaining faults instead of chaining pre-computed test vectors. This helps us overcome many problems of vector ordering, grouping and linear-equation solving that traditional seed computation methods have faced. The method is general that it can be used for any LFSR structure or configuration, coupled with one of many different BIST architectures. Experimental results show that the method is able to produce a very small set of seeds with low computational cost.

References

[1] J. A. Waicukauski, E. Lindbloom, E. B. Eichelberger, and O. P. Forlenza, "A method for generating weighted random test pattern," *IBM J. Res. Dev.*, vol. 33, pp. 149–161, Mar. 1989.

[2] H.-J. Wunderlich, "Multiple distributions for biased random test patterns," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, pp. 584 –593, jun 1990.

[3] B. Koenemann, "LFSR-coded test patterns for scan designs," in *Proc. IEEE Euro. Test Conf.*, pp. 237 –242, 1991.

[4] S. Hellebrand, S. Tarnick, J. Rajski, and B. Courtois, "Generation of vector pattterns through reseeding of multiple-polynomial linear feedback shift registers," in *Proc. Intl Test Conf.*, p. 120, sep 1992.

[5] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers," *IEEE Trans. Computers*, vol. 44, pp. 223 –233, feb 1995.

[6] S. Hellebrand, B. Reeb, S. Tarnick, and H.-J. Wunderlich, "Pattern generation for a deterministic bist scheme," in *Proc. IEEE Intl Conf. Computer-Aided Design*, pp. 88 –94, nov 1995.

[7] A. Al-Yamani and E. McCluskey, "BIST-guided ATPG," in *Proc. Intl Symp. Quality of Electronic Design*, pp. 244 – 249, march 2005.

[8] A. Al-Yamani, S. Mitra, and E. McCluskey, "BIST reseeding with very few seeds," in *Proc. VLSI Test Symp.*, pp. 69 – 74, april-1 may 2003.

[9] C. Fagot, O. Gascuel, P. Girard, and C. Landrault, "On calculating efficient LFSR seeds for built-in self test," in *Proc. European Test Workshop*, pp. 7 –14, may 1999.

[10] C. Krishna, A. Jas, and N. Touba, "Test vector encoding using partial LFSR reseeding," in *Proc. Intl Test Conf.*, pp. 885 –893, 2001.

[11] M. Lempel, S. Gupta, and M. Breuer, "Test embedding with discrete logarithms," *IEEE Trans. CAD*, vol. 14, pp. 554 –566, may 1995.

[12] S. Prabhu, M. Hsiao, L. Lingappan, and V. Gangaram, "A Novel SMT-Based Technique for LFSR Reseeding," in *Proc. VLSI Design Conf.*, pp. 394 –399, jan. 2012.

[13] J. Silva and K. Sakallah, "Robust search algorithms for test pattern generation," in *Proc. Intl Sympo. Fault-Tolerant Computing*, pp. 152 –161, jun 1997.

[14] S. B. Akers, C. Joseph, and B. Krishnamurthy, "On the Role of Independent Fault Sets in the Generation of Minimal Test Sets," in *Proc. International Test Conf.*, pp. 1100 –1107, 1987.

[15] P. Flores, H. Neto, and J. Marques Silva, "An exact solution to the minimum size test pattern problem," in *Proc. Intl Conf. Computer Design*, pp. 510 –515, oct 1998.

[16] K. R. Kantipudi and V. D. Agrawal, "On the size and generation of minimal N-detection tests," in *in Proc. 19th International Conf. VLSI Design*, pp. 425–430, 2006.

[17] M. Živkovic, "A table of primitive binary polynomials," *Math. Comput.*, vol. 62, pp. 385–386, Jan. 1994.

[18] L. De Moura and N. Bjørner, "Z3: an efficient SMT solver," in *Proc. TACAS*, TACAS'08/ETAPS'08, pp. 337–340, Springer-Verlag, 2008.

[19] Gurobi Optimization, Inc., "Gurobi Optimizer Reference Manual," 2012.