LOW-OVERHEAD BUILT-IN BIST RESEEDING

Ahmad A. Al-Yamani and Edward J. McCluskey
Center for Reliable Computing, Stanford University
{alyamani, ejm@crc.stanford.edu}

Abstract

Reseeding is used to improve fault coverage in pseudo-random testing. Most of the work done on reseeding is based on storing the seeds in an external tester. Besides its high cost, testing using automatic test equipments (ATEs) makes it hard to test the circuit while in the system. In this paper, we present a technique for built-in reseeding. Our technique requires no storage for the seeds. The seeds are encoded in hardware. The seeds we use are deterministic so 100% fault coverage can be achieved. Our technique causes no performance overhead and does not change the original circuit under test. Compared to previous work, our technique takes less area overhead. Also, the technique we present is applicable for single-stuck-at faults as well as transition faults. In our technique, we expand every seed to as many ATPG patterns as possible. This is different from many existing reseeding techniques that expand every seed into a single ATPG pattern. This paper presents the built-in reseeding algorithm together with the hardware synthesis algorithm and implementation for both single-stuck-at faults as well as transition faults.

1. Introduction

An advantage of built-in self-test (BIST) is its low cost compared to external testing using automatic test equipment (ATE). In BIST, on-chip circuitry is included to provide test vectors and to analyze output responses. One possible approach for BIST is pseudo-random testing using a linear feedback shift register (LFSR) [McCluskey 85]. Among the advantages of BIST is its applicability while the circuit is in the system.

Many digital circuits contain random-pattern-resistant (r.p.r.) faults that limit the coverage of pseudo-random testing [Eichelberger 83].The r.p.r. faults are faults with low detectability (Few patterns detect them). Several techniques have been suggested for enhancing the fault coverage achieved with BIST. These techniques can be classified as: (1) Modifying the circuit under test (CUT) by test point insertion [Eichelberger 83, Cheng 95, Touba 96a] or by redesigning the CUT, (2) *Weighted pseudo-random patterns*, where the random patterns are biased using extra logic to increase the probability of detecting r.p.r. faults [Brglez 89, Muradali 90, Wunderlich 90] and (3) *Mixed-mode testing* where the circuit is tested in two phases. In the first phase, pseudo-random patterns are applied. In the second phase, deterministic patterns are applied to target the undetected faults [Koenemann 91, Hellebrand 92, Venkataraman 93, Touba 96b]. The technique we present is a mixed mode technique based on inserting deterministic patterns between the pseudo-random patterns.

Modifying the CUT is often not desirable because of performance issues or intellectual property reasons. Weighted pseudo-random sequences require multiple weight sets that are typically stored on chip. Mixed mode testing is done in several ways; one way is to apply the deterministic patterns from an external tester. Although this technique reduces the tester time and storage required, it still doesn't eliminate the need for external testing. Another technique is to store the deterministic patterns on an on-chip ROM. This ROM requires high area overhead on the chip, needs to be tested, and there needs to be additional circuitry to apply the patterns in the ROM to the circuit under test. Instead of storing patterns, seeds can be stored on the tester or on the on-chip ROM. These seeds are loaded into the LFSR and then expanded into the scan chain of the circuit. This technique reduces the storage needed at the tester or the ROM but it does not eliminate it. It also does not eliminate the need for the circuitry that applies the seeds from the ROM to the LFSR.

Another technique for mixed-mode testing is *mapping logic* [Touba 95]. The strategy is to identify patterns in the original set that don't detect any new faults and map them by hardware into deterministic patterns for random-pattern-resistant faults. This mapping is designed to use as few gates as possible. While this mapping eliminates the need for the on-chip ROM, it may require a large area because it performs the mapping for all the ATPG patterns generated for the faults that are not detected by pseudo-random patterns. It also requires hardware for detecting the patterns that don't cause fault dropping and more hardware for applying the new values.

In this paper, we present a technique that combines mapping logic and *reseeding* (loading the PRPG with a new seed). Our technique uses a simple circuit to identify the cycles at which the LFSR is to be reseeded and it uses the same circuit to choose the new seed. A distinct feature of our technique is that it utilizes the LFSR flip-flops for storing the seeds. All that is required is the circuit that detects the *end of sequence* (*EOS*) pattern, which is the last pattern in a sequence of pseudo-random patterns. This same circuit is used to pick the new seed only by inverting the flip-flops in which the seed is different from the current contents of the LFSR.

Touba's mapping logic alters the outputs of the pseudo-random pattern generator (PRPG) to include deterministic patterns in the test set. On the other hand, our technique alters the contents of the PRPG to include the deterministic patterns.

Our technique can be tuned for the desired fault coverage and completely eliminates the need for external testing or for a ROM to store the seeds. Furthermore, it needs to encode less seeds compared to previous work. Actually, in its best case, our technique causes an order of magnitude reduction in the number of seeds to be encoded compared to previous work and even in its worst case it needs less seeds than previous work. With a small modification, our technique can be applied for transition faults rather than single stuck faults. Even if it's more desirable to apply the deterministic patterns externally than to add the mapping logic, our technique is still applicable and in this case it outperforms the other reseeding schemes in the amount of storage required and allows trading off tester storage and test length.

Our contributions in this paper are summarized as follows: (1) A reseeding technique that eliminates completely the need for external pattern storage and application or an on-chip ROM. It's based on encoding the seeds in hardware and using special hardware for the LFSR. The technique is applied for both single-stuck-at as well as transition faults. (2) The hardware implementation for the given technique has much smaller overhead than that of similar techniques as shown by simulation results. (3) A reseeding algorithm based on the given hardware implementation. The algorithm allows the user to trade off test length for hardware overhead. The output of the reseeding algorithm is the reseeding circuit in a format ready for optimization and synthesis.

In Sec. 2 of this paper, we review the related literature. In Sec. 3, we explain the circuitry required for implementing the presented scheme. Section 4 discusses the reseeding algorithm. Section 5 shows the simulation results. Section 6 concludes the paper.

2. Previous Work

Konemann presented a technique for coding test patterns into LFSRs of size S_{max}+20, where S_{max} is the maximum number of specified bits in the ATPG patterns. The method is based on solving a linear system of equations for every pattern. The linear system of equations is based on the linear expansion of the contents of the LFSR into the scan chain. If there is any linear dependence between the equations, solving this system won't be possible. By adding 20 bits to S_{max} as the size of the LFSR, the probability of linear dependence drops to 1 in a million [Konemann 91].

In [Hellebrand 95], a scheme was presented for using Multiple-polynomial LFSRs (MP-LFSRs) in BIST. Reseeding was used to target r.p.r. faults. The authors used MP-LFSRs to reduce the probability of linear dependence. The main contribution of the work was in using a smaller LFSR of size S_{max}+4, where S_{max} is the maximum number of specified bits in the ATPG patterns, to achieve the same probability of finding an encoding as in the case of S_{max}+19 with a single-polynomial LFSR.

[Zacharia 95] and [Rajski 98] presented a reseeding-based technique that improves the encoding efficiency by using variable-length seeds together with an MP-LFSR. The authors presented a technique that reuses part of the scan chain flip-flops in expanding the seeds.

In [Huang 97], programmable LFSRs (PLFSRs) are used to enable multiple polynomials for the PRPG. The seed/polynomial selection is performed using Gauss-elimination. They try to embed as many ATPG patterns as possible in the LFSR sequence by solving a linear system of equations. Multiple seeds might be required for this technique to achieve high coverage. The authors assume that the seeds are available from an external tester or a ROM.

In [Kagaris 99], a synthesis technique for counter-based test set embedding was presented. They use counters instead of LFSRs. They present a multi-seed counter and they assume that the seeds are stored on the tester or on an on-chip ROM.

[Chakrabarty 00] presented an approach for BIST pattern generation using twisted-ring counters. Their counters use existing circuit flip-flops to make the counter. In order to achieve good fault coverage they rely on reseeding the counter. The seeds are stored in a ROM.

Hellebrand presented a reseeding technique based on *folding counters*, which resemble programmable Johnson counters. When combined with an input reduction technique, the presented scheme reduced the seed storage required [Hellebrand 00].

In [Krishna 01], a new form of reseeding was described for high encoding efficiency. The authors presented *partial dynamic reseeding*. In the presented scheme, the contents of the LFSR are incrementally modified instead of modifying them all at once. By making use of the degrees of freedom in solving the linear system of equations the paper achieves higher encoding efficiency than static reseeding.

All of the above schemes assume that seeds are either applied from an external tester or stored in an on-chip ROM. Although applying the seeds externally means much less tester storage and bandwidth than if the patterns were to be stored, it still means that the chip has to be put on the tester. Also, although the ROM eliminates the need for the tester, there must be some circuitry to choose when to load seeds from the ROM and other circuitry to actually load them onto the LFSR.

The technique presented in this paper embeds the seeds on the chip without requiring a ROM. Other than the circuit needed to detect when to reseed, minimal hardware is needed to load the desired seeds. The technique presented in this paper is orthogonal to all of the above techniques; i.e., it can be combined with Hellebrand's technique for reducing the LFSR size. It can also be combined with partial dynamic reseeding for a high encoding efficiency. It can also be combined with Rajski's technique that utilizes part of the scan chain or Chakrabarty's technique for reusing the scan chain flip-flops as an LFSR.

In [Lempel 95], an analytical method was presented for computing a single seed for random pattern resistant circuits using an LFSR with a given polynomial. The complexity of the given procedure quickly increases with the number of inputs. In [Fagot 99], a simulation scheme for calculating initial seeds for LFSRs was presented. The scheme is based on simulating several sequences that include a set of ATPG vectors.

The above schemes assume a single seed. Having several seeds should increase the fault coverage with the same test length if the seeds are chosen appropriately; the main problem is where to store the seeds.

[Savir 90] presented a reseeding scheme that requires having shadow flip-flops for the LFSR flips-flops. The shadow flip-flops contain the next seed. These shadow flip-flops are loaded with the XOR of the old shadow contents and the original LFSR flip-flops contents. This way, the new seed is expected to be far in the sequence from the current contents of the LFSR. This scheme has the advantage of diversity of the sequences from which the patterns are drawn. It also has the advantage of not requiring a ROM for storing the seeds. However, it does not necessarily solve the random pattern resistant faults problem since the seeds are random. Kim presented a method for generating non-successive pseudo-random test patterns by cascading the LFSR with the scan chain and including a feedback from the scan-out signal into the LFSR. Non-successive pseudo-random patterns don't necessarily solve the r.p.r. faults problem [Kim 96]. In [Crouch 95], a self re-seeding LFSR was presented. Again the LFSR is loaded with a random seed every time a pattern is repeated in part of the LFSR.

Touba and McCluskey came up with an innovative approach for applying deterministic patterns through mapping logic [Touba 95]. In their technique, random patterns that don't detect undetected faults are mapped to ATPG generated cubes through combinational logic. The mapping is performed in two phases, the source patterns are identified in the first phase, and the ATPG cubes are loaded in the second phase. Several iterative minimization heuristics are applied to reduce the area overhead of the mapping logic.

Our technique is a generalization of mapping logic. It has the following advantages: (1) The area required is less than that needed for logic mapping because after loading every seed, the LFSR runs in autonomous mode for sometime detecting more r.p.r. faults without having to perform more mappings. (2) In mapping logic, we need logic for detecting the pattern to be mapped and more logic to perform the mapping. On the other hand, in our technique we only need the logic that detects the patterns that need to be mapped. Enforcing the new values in the LFSR is done utilizing the current contents of the flip-flops of the LFSR. (3) Our technique is applicable to both transition faults as well as single-stuck-at faults. Also, the increase in the required area overhead for our technique is negligible.

In [Pradhan 99], a new LFSR based on Galois fields (GLFSR) was presented. Their results show that the new GLFSR achieves higher fault coverage than conventional pseudo-random pattern generators. The experiments show that using GLFSRs, the test length is reduced for SSF and transition fault coverage of 90% and 95%. The results are for combinational circuits.

Unlike the previous work on BIST for transition faults, in this paper we don't rely only on randomness for transitions faults. We perform mixed mode testing. We apply both pseudo-random as well as deterministic patterns. Also, we apply our technique on sequential circuits rather than combinational circuits.

3. Reseeding Circuitry Implementation

The operation of the reseeding circuit is as follows: the LFSR starts running in autonomous mode for sometime according to the algorithm described in Sec. 4. Once it is time for reseeding, a seed is loaded into the LFSR, which then goes back to the autonomous mode and so on and so forth until the desired coverage is achieved.

Figure 1(b) shows the structure of the LFSR and its interaction with the reseeding circuit. For our technique we use muxed flip-flops as shown in the figure. It's these muxes that enable our technique to reload the LFSR with the new seed without additional logic. All that is needed from the reseeding circuit is to detect the end of sequence (EOS) contents in the LFSR. The output of the reseeding circuit generates a logic value 1 at the select lines of the muxes only at the locations where the new seed is different from the current contents of the LFSR flip-flops.

As seen in the figure, the only modification to the LFSR compared to a regular LFSR are the muxes. The LFSR flip-flops are replaced by muxed flip-flops just like the scan chain flip-flops. The mux can be implemented using transmission gates using only 6 minimum size transistors.

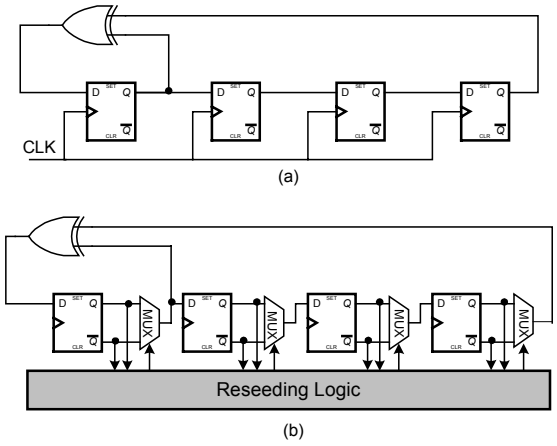


Figure 1: Reseeding circuit connection to the LFSR: (a) A standard 4-stage LFSR (b) 4-stage LFSR with reseeding circuit circuit

Let's turn our attention to the reseeding circuit itself by looking at the following example. Figure 2 is an example using a 4-stage self-reseeding LFSR with a primitive polynomial. The table in part (a) shows the full sequence of the regular LFSR. Assume that we want to reseed after the 6^{th} cycle (c6). The reseeding circuit needs to be an AND gate that takes as inputs the contents of the LFSR at c6. So in the example the input to the reseeding AND is $Q_1'Q_2Q_3'Q_4$. All the cycles that are not part of the desired sequence can be used to minimize the reseeding circuit. In other words, we can combine the patterns that will activate the reseeding circuit together with all the patterns that won't happen in our desired sequence to generate the maximum possible number of don't cares. An important advantage of the given structure is that the reseeding circuit inputs are readily available in both polarities, which gives a lot of room for more optimizations. For example, we can replace the AND gate in the figure with a NOR gate, which is less expensive in CMOS. All that is needed is to use $Q_2'Q_3Q_4'$ instead of $Q_2Q_3'Q_4$.

Let the seed be 0100 (c12); we can easily calculate c11 given the polynomial of the LFSR (c11 = 1001). The reason we calculate c11 and not c12 is because we want the seed to be loaded into the LFSR in the next clock cycle. To find the location where c6 is different from c11 we XOR them, XORing c6 with c11 yields 1100 which means that the output of the reseeding AND should generate a logic value 1 at the select lines of the MUXes of Q_1 and Q_2 only. The resulting circuit which is a 3-input AND is shown in the figure.

As more seeds are required, every select line of the MUXes will be a function of the end-of-sequence patterns that will activate it to flip the contents of its corresponding flip-flop. We can then optimize the circuit for that select line by combining all the patterns that will activate it together with all the patterns that won't occur in the desired sequence. This is like minimizing a function given all of its on-set patterns as well as all of its impossible patterns (don't care-set). Furthermore, multiple-output optimization can be done for the select lines.

[Needham 98] shows that 58% of the defective microprocessors returned to intel have open defects. Our experimental results from Murphy and ELF 35 experiments show that transition fault test sets are much more effective than SSF fault test sets in detecting open defects [Li 01]. Furthermore, the only way we could detect all the defective chips that escaped all 100% test sets was using transition faults propagated to all reachable outputs (TARO) [Tseng 01].

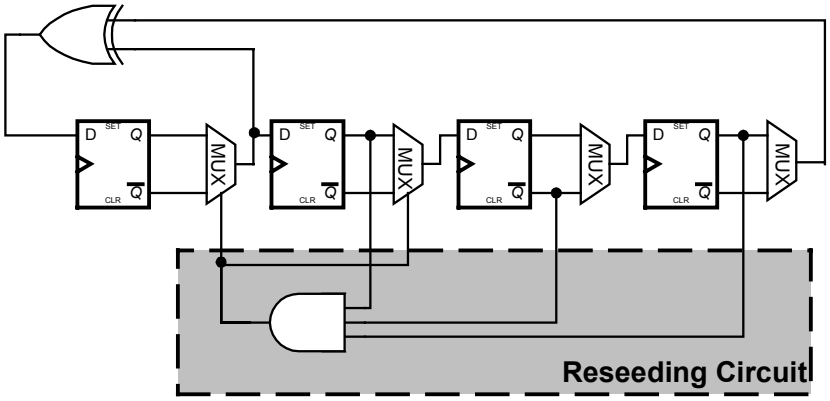
None of the previous publications on BIST considered mixed mode testing for transition faults. In this paper we present our built-in reseeding technique and we apply it using both SSF test sets and transition fault test sets.

Cycle	Q1	Q2	Q3	Q4	Cycle	Q1	Q2	Q3	Q4
0	1	0	0	0	8	1	1	0	1
1	1	1	0	0	9	0	1	1	0
2	1	1	1	0	10	0	0	1	1
3	1	1	1	1	11	1	0	0	1
4	0	1	1	1	12	0	1	0	0
5	1	0	1	1	13	0	0	1	0
6	0	1	0	1	14	0	0	0	1
7	1	0	1	0	15	1	0	0	0

End of Sequence (EOS) = c6 = 0101
Seed = 0100 = c12
Select Lines Activated = (c6) XOR (c11)
= 0101 XOR 1001
= 1100

=> Select lines of Q1 and Q2 activated

(a)



(b)

Figure 2: Example reseeding circuit (a) Select lines computation (b) Hardware implementation

There are two ways to apply transition fault test sets to circuits with scan chains. One way is to use double clock pulses (launch on capture). Once the 1st pattern is loaded into the scan chain, a clock pulse is applied so the response of the combinational logic to the 1st pattern is latched into the flip-flops. Another clock pulse is also applied such that the response of the combinational logic to the 1st pattern is used as the 2nd pattern in the transition fault pair. Logic and fault simulation are used to figure out the response of the 1st pattern and accordingly find the detected faults. The other way is to load the scan chain with the two successive patterns (launch on shift). Once the first pattern is applied, the contents of the scan chain are shifted, the 2nd pattern is applied and the results are latched in the scan chain to be shifted out to the compactor.

We use the first way (launch on capture) for transition faults. We also don't force a particular value on the primary inputs when applying the 2nd pattern. The primary inputs receive their values from the LFSR whatever they are. This way, the reseeding circuit only needs to load the LFSR with the 1st pattern, which results in a considerable reduction in the area overhead. The reseeding circuit is synthesized such that it changes the contents of the LFSR from the current values to the 1st pattern in the transition fault pattern pair.

Figure 3 shows where the reseeding circuit fits in a system level view of a circuit with an LBIST controller. The pattern counter is part of the LBIST controller and it is used to count the patterns applied to the circuit under test (CUT).

In a BIST environment, if the LFSR is known in advance and the initial seed and test length are also known, the reseeding circuit for a given select line may be designed to be the logical sum of the pattern counter values of those patterns that activate that particular select line. The dashed line in Fig. 3 corresponds to the reseeding circuit taking its inputs from the pattern counter. If a set of test length *TL* is applied to the circuit the pattern counter will be of size $\lceil \log_2(TL) \rceil$. This can lead to large reduction in the complexity of the reseeding circuit because the number of inputs of the reseeding circuit is reduced. Again having both polarities available at the input gives room for further minimization of the reseeding circuit.

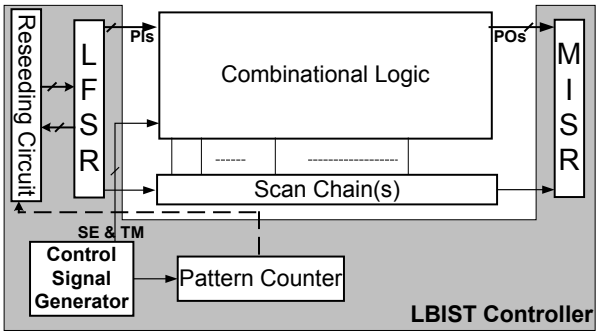


Figure 3: Reseeding circuit in a system view of BIST environment.

As has been shown in this section, the area overhead required by the reseeding circuit should be much less than that needed by the mapping logic technique because: (1) The number of seeds needed is smaller than number of ATPG patterns as will be shown in the simulation results section (2) No need for additional logic to force the desirable value in the flip-flops. Forcing the desirable value is done through the muxed flip-flops by utilizing the values already stored in them in both polarities. In other words, all that is required to be synthesized in gates is the circuitry to detect the source pattern. No circuitry for mapping. The mapping is done through activating particular select lines if the LFSR is to be reseeded. This is due to the use of the logic values in the LFSR flip-flops. We only need to know whether we should flip them or not.

Our technique requires no extra hardware to apply it to transition faults. Although pairs of 2 vectors are required for transition faults, our technique needs hardware only for the first pattern. Our technique is as shown, orthogonal to all previous optimizations, so it can be directly used with any previous technique for reducing the LFSR size or increasing the encoding efficiency.

4. Reseeding Algorithm

The algorithm we present is based on the following strategies in mixed mode testing: (1) Generate ATPG patterns for faults that were not detected with pseudo-random patterns and calculate the seeds for these patterns, (2) When a seed is loaded into the LFSR, let the LFSR run in autonomous mode for sometime because there is a chance that some pseudo-random patterns will cause fault dropping so that some of the ATPG patterns are not needed. This may not be a wise idea if the seeds are applied from a tester because it may take longer to wait for the ATPG patterns in a pseudo-random sequence than just to apply them from the tester. On the other hand, if the seeds are stored or coded on-chip, as it is the case in this work, then it is definitely worth it to minimize the number of seeds that need to be loaded for the same coverage. This will directly minimize the area overhead on the chip, (3) as long as the pseudo-random patterns are detecting more faults the LFSR should keep going on autonomous mode. When the pseudo-random patterns become ineffective, the LFSR should be loaded with a new pattern. How to quantify the effectiveness of pseudo-random patterns? The answer is in the next paragraphs.

We define a parameter called *coverage improvement threshold* (CIT). As long as applying more pseudo-random patterns improves the coverage by at least CIT%, the pseudo-random patterns will be considered effective. When the improvement falls below CIT%, the pseudo-random patterns are considered ineffective. We need to determine how many patterns are simulated at a time before measuring the improvement in coverage. Most of the fault simulation tools perform parallel fault simulation. They apply patterns in groups and measure their effect on the fault coverage. For this work, the fault simulation tool (SynTest) simulated 32 patterns at a time. In our simulations, we used different values for the number of patterns applied at a time. Algorithm 1 shows the built-in reseeding algorithm.

The only user-specified parameters for this algorithm are the coverage improvement threshold (CIT) and the number of patterns applied at a time. At one extreme, choosing a very small CIT, means that the user prefers to stick to pseudo-random patterns as long as they have any improvement in the coverage. This in turn means the user wants the minimum hardware overhead for the reseeding circuit area overhead. In return for that, the user is willing to have a long test length. At the other extreme, if the user specifies a very high CIT, it means that he has enough area on the chip for the reseeding circuit. However, there is minimal tolerance for test length. In other words, the number of seeds to be loaded is inversely proportional to the test length and directly proportional to the area of the self-reseeding circuit. Its up to the user to choose which of the two resources is scarcer in his design. The results in the next section show the impact of CIT on the test length and on the area of the reseeding circuit.

Reseeding based on CIT is one way to choose when to reseed the LFSR. Many other strategies can be used for selecting the reseeding cycles. One way is to choose a fixed length for running the LFSR in autonomous mode after the seed is loaded.

```
TL: Test Length.
PF: Patterns File.
FC (PF, TL): Fault Coverage of TL patterns from PF.
RFC: Required Fault Coverage.
CIT: Coverage Improvement Threshold.
S: Step Size = Number of patterns simulated in parallel.
EOS: End of sequence = the seed of the last pattern in PF.
PLA: Input output file used for synthesis of the reseeding circuit.

Built-In Reseeding (RFC, CIT, LFSR, S)
   1. Run LFSR in autonomous mode for a long enough length.
   2. Save the patterns → PF.
   3. Fault simulate the patterns in PF on the CUT in groups of S at a time.
   4. Let MTL = minimum TL such that: (FC (PF, MTL+S) – FC (PF, MTL)) < CIT
   5. Truncate PF at MTL.
   6. If FC (PF, MTL) ≥ RFC, Return (PF).
   7. Else
         a. Run ATPG such that coverage of (MTL + ATPG) ≥ RFC
         b. Calculate the seeds for the ATPG patterns.
         c. Save the seeds → SEEDS
         d. For i=1 to |SEEDS|
               i. Pick PAT from SEEDS such that DF=EOS(PF)⊕PAT is minimized
              ii. If PAT does not improve coverage, pick another one.
             iii. Add EOS(PF)to PLA with output DF.
              iv. Run LFSR in autonomous mode for a long enough length.
               v. Append patterns → PF.
              vi. Fault simulate PF patterns on the CUT, S at a time.
             vii. Set MTL s.t. it includes PAT && improvement < CIT.
            viii. Truncate PF at MTL.
              ix. If FC (PF, MTL) ≥ RFC, Return (PF, PLA).
         e. Return (PF, PLA).
```

Algorithm 1: Built-in reseeding algorithm

In a sense, our built-in reseeding algorithm is actually a generalization of mapping logic. For example, assume that we choose to run the LFSR for a fixed length after reseeding and we choose the length to be one. In that case, our technique becomes similar to mapping logic. We chose to expand the seeds into many patterns since this should reduce the number of seeds to be loaded and accordingly reduce the area of the reseeding circuit.

For some circuits, all we need to catch the undetected faults is to take the LFSR to another location in the pattern space. In that case, one or two seeds are enough. For other circuits, the undetected faults require many seeds to be loaded. In that case, our technique will have area overhead close to that of mapping logic. Yet, mapping logic will have a shorter test length. A major advantage for the technique we present is that it allows the user to discover what type of circuit he has and accordingly the right parameters can be chosen for an optimal test set.

In case of transition faults, the only change to the algorithm is that fault simulation and ATPG should be done for transition faults instead of SSFs. ATPG generated transitions fault patterns have only the 1st pattern specified. The 2nd pattern is the response of the circuit to the 1st pattern. So, only the first patterns of the transition fault patterns' pairs should be encoded in the hardware of the reseeding circuit.

5. Simulation Results

In this section we present the results of some simulation experiments we performed using our built-in reseeding technique. We performed our experiments on some ISCAS 89 benchmarks. The characteristics of the benchmarks we used are shown in Table 1. The table shows the number of primary inputs, primary outputs and flip-flops in the circuits. It also shows the cell-area of the circuits in LSI library cells area units. The library used for technology mapping is LSI Logic G.Flex library, which is a 0.13 μ technology library.

Table 1: ISCAS 89 CUTs Used in The Experiments.

Circuit Name	Primary Inputs	Primary Outputs	Scan Chain Size	Area
s1238	14	14	18	2,740
s1488	8	19	6	3,555
s1494	8	19	6	3,563
s5378	35	49	179	14,376
s9234	36	39	211	25,840
s13207	62	152	638	44,255
s15850	77	150	534	48,494
s35932	35	320	1728	106,198
s38417	28	106	1636	120,180
s38584	38	304	1426	115,855

5.1 Comparison with previous work

In order to evaluate the effectiveness of our technique we performed some simulation experiments to compare it to previous work that requires a seed per pattern. The previous work we compare to includes [Koenemann 91], [Hellebrand 92], [Rajski 98] and [Krishna 01] where the assumption is to have a seed per pattern. It also includes [Touba95], where hardware is needed to map each pattern.

The comparison is based on the number of seeds that need to be encoded or stored if our built-in reseeding technique is used and the number of seeds that need to be stored or patterns that need to be mapped if previous techniques are used. The number of patterns determines the area of the reseeding or mapping circuit. Since further area minimization heuristics can be applied to all techniques, its fair to compare them in terms of the number of patterns that need to be mapped or stored for the same coverage. This comparison can be done for all possible combinations of coverage improvement threshold (CIT) and step size, see Sec. 4. Since the test length increases when we use our reseeding technique compared to previous techniques, it's fair to show the test length in the comparison. Why should we tolerate this increase in test length? (1) The area is a very scarce resource compared to test length in BIST. (2) The effect of increasing the test length is not as severe with BIST as it is with external testing because BIST is much faster than external testing. (3) Increasing the test length increases the number of pseudo-random patterns that are likely to be effective in catching unmodeled defects.

Table 2 shows a comparison of previous techniques and our reseeding technique. The table is for 100% single-stuck-at fault coverage, 1.0 CIT and 1024 patterns as a step size. In its best case, built-in reseeding reduces the number of seeds required for 100% coverage by an order of magnitude and increases the test length by only a factor of 1.7. Notice that this increase in test time does not come at extra tester cost because the reseeding is built into the circuit itself. Overall, the minimum reduction in the number of patterns is 5%. Not surprisingly, the worst case improvement for the number of patterns comes with the worst case degradation for the test length. This means that the remaining faults in the circuit are very random pattern resistant. So, although we keep loading seeds and running pseudo-random patterns, no improvement happens in the coverage unless we pick another seed.

The table shows that there are many cases where reseeding offers considerable improvement in the number of patterns required to test the circuit and at the same time, it doesn't cause a large degradation in the test length. Also, even in its worst case, built-in reseeding gives some improvement over previous techniques in terms of the number of seeds to be stored or patterns to be mapped.

We performed another experiment to compare how the coverage improvement threshold (CIT) affects previous work and our built-in reseeding technique. Figure 4 shows the test length of previous work and built-in reseeding as a function of CIT for one of the circuits. Recall that using a high CIT means not accepting random patterns unless they cause significant improvement on the fault coverage. The required coverage is 100%. For both techniques, the higher the CIT, the lower the test length. This is because if CIT is higher, then we are allowing less random patterns and using more deterministic patterns for both techniques. The less random patterns we use, the shorter test length we get.

Let's now look at the number of patterns or seeds we need for 100% coverage as a function of CIT. Figure 5 shows the number of patterns mapping logic needs to map for 100% coverage and the number of seeds built-in reseeding needs for the same coverage as a function of CIT. For previous work, the higher the CIT, the more deterministic patterns are required which means larger area overhead for the mapping logic or larger storage for the seeds. For built-in reseeding, the curve is not monotonically increasing. This simply means that using a larger CIT to get a shorter test length doesn't necessarily mean a larger area overhead for the reseeding circuit. In fact, for the given example, using a very high CIT results in much less seeds compared to using a small CIT. This is a very significant advantage of built-in reseeding compared to mapping logic because it simply means that we can get double wins both in test length and area overhead, why? Because built-in reseeding is very sequence dependent, unlike mapping logic.

Table 2: Comparison of Built-In Reseeding and Previous Work.

Circuit	Number of seeds/patterns to be mapped			Test Length		
	Seed per pattern	Built-in reseeding	%Reduction	Previous work	Built-in reseeding	Degradation factor
s1238	24	9	62.5	4,096	15,360	3.8
s1488	4	1	75.0	3,072	4,096	1.3
s1494	4	1	75.0	3,072	4,096	1.3
s5378	60	22	63.3	3,072	27,648	9.0
s9234	93	62	33.3	5,120	76,800	15.0
s13207	121	47	61.2	5,120	60,416	11.8
s15850	41	35	14.6	4,096	45,056	11.0
s35932	18	1	94.4	3,072	5,120	1.7
s38417	78	74	5.1	5,120	89,088	17.4
s38584	107	68	36.4	4,096	76,800	18.8

The algorithm we presented in this paper picks the seed that is closest to the end of sequence pattern in the LFSR. This means that by allowing different sequence lengths, the algorithm will end up picking different seeds that may result in more useful pseudo-random sequences.

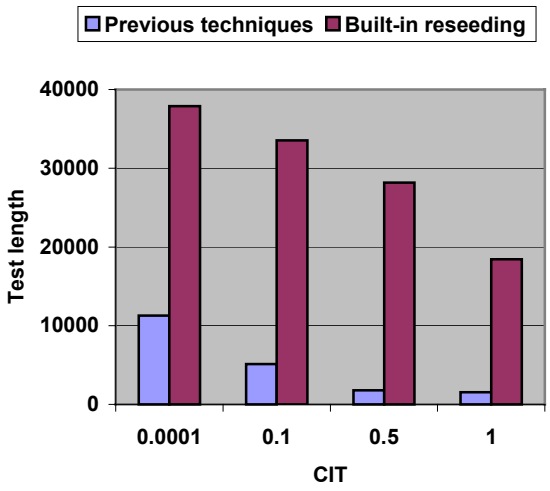


Figure 4: Test length vs. CIT for previous-techniques and built-in reseeding (s13207)

Most of the circuits exhibited similar behavior to that shown in Fig. 4 and 5. Due to space limitations, we cannot show 2 graphs per circuit. However, in order to avoid drawing conclusions from a special case, we show the same graphs for the average test length and number of patterns and seeds for all circuits.

Figure 6 shows the average test length of previous techniques compared to built-in reseeding as a function of CIT for all circuits. For both techniques, the higher the CIT, the lower the test length. Figure 7 shows the average number of patterns mapping logic needs to map for 100% coverage and the number of seeds built-in reseeding needs for the same coverage as a function of CIT. Again, For reseeding, the curve is not monotonically increasing like that for mapping logic.

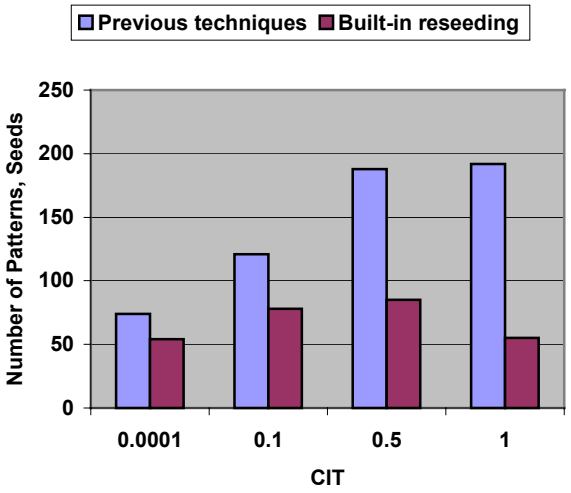


Figure 5: Number of patterns/seeds vs. coverage improvement threshold (s13207).

As has been shown in this section our built-in reseeding technique provides a significant saving in area compared to mapping logic and in storage compared to previous reseeding techniques. As explained in Sec. 3, using the pattern counter contents as inputs to the reseeding circuit will significantly reduce the number of inputs of the reseeding circuit because only $\lceil \log_2(TL) \rceil$ inputs are needed, where TL is the test length. This reduction in the number of inputs reduces the area overhead significantly.

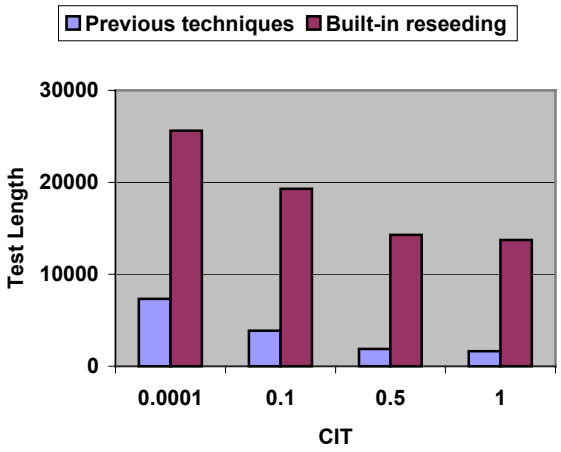


Figure 6: Test length vs. coverage improvement threshold (average).

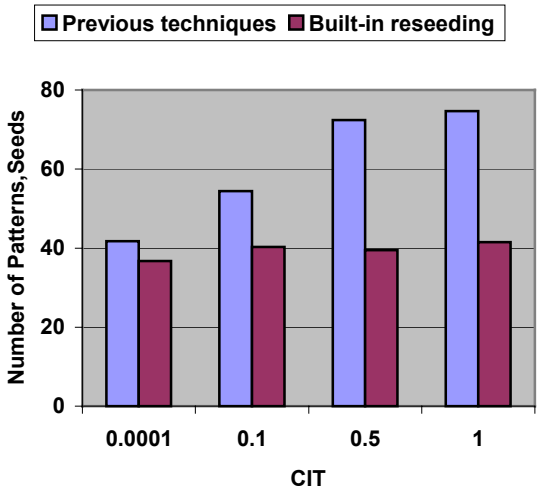


Figure 7: Number of patterns/seeds vs. coverage improvement threshold (average)

5.2 Area overhead and test length for SSFs

In this section we present the area overhead of our built-in reseeding technique using SSF model. Table 3 shows the area overhead of the reseeding circuits for the benchmarks we used. The reseeding circuits given were designed based on 100% fault coverage. The second column is the cell area of the circuit. The 3rd column is the cell area of the reseeding circuit. The 4th column is the area overhead of the reseeding circuit. As mentioned earlier, since the number of seeds required for achieving the desired coverage is dependent on the order on which seeds are picked, its not necessary that using a high CIT leads to a large reseeding circuit. The area overhead given in the table is the minimum area overhead we could achieve for 100% coverage. The 5th column shows the CIT for which the given area overhead was achieved. The 6th column is the test length given by that CIT and the 7th column is the step size used. For almost all circuits, the area overhead ranged from 0.05% to 5.5%.

As shown in the table the minimum area overhead is achieved with different CITs. This is due to the fact that the number of seeds required to be loaded is highly dependent on the order in which seeds are picked. Our algorithm picks the seeds with the objective of minimizing the area overhead so this might have different effects on the test length.

5.3 Area overhead and test length for transition faults

In this section we present the area overhead of our built-in reseeding technique using the transition fault model. Table 4 shows the area overhead of the reseeding circuits for the circuits we used. The reseeding circuits given were designed based on 100% transition fault coverage. The area overhead given in the table is the minimum area overhead we could achieve for 100% coverage. The area overhead ranged from 0.1% to 12% in most cases.

As shown in the table, in most cases, the minimum area overhead is achieved with different step sizes. This is again due to the fact that the number of seeds that need to be loaded is highly dependent on the order in which seeds are picked.

In general the area overhead of the built-in reseeding circuit for transition faults was close to that for SSFs. The reason is that we only encode the 1st patterns of the transitions patterns pairs in the reseeding circuit. We measure the test length of transition fault test sets in pairs of patterns. So although the test length looks smaller than that of single stuck faults its actually higher because its in pairs of patterns.

6. Conclusions

In this paper, we presented a built-in reseeding scheme. Our scheme is based on encoding the seeds in a reseeding circuit that is included on chip. 100% fault coverage can be achieved with our technique without any external testing. Our built-in reseeding scheme is a generalization of mapping logic. It allows the designer to trade-off area overhead for test length in a more optimized way than mapping logic.

Table 3: Area Overhead and Test Length For Built-In Reseeding (SSFs)

Circuit	Area	Reseeding Area	% Area Overhead	CIT	TL	STP
s1238	2,740	149	5.4	0.1	20,480	1024
s1488	3,555	52	1.5	0.5	5,120	1024
s1494	3,563	52	1.5	0.5	5,120	1024
s5378	14,376	229	1.6	0.5	17,408	1024
s9234	25,840	3587	13.9	0.1	99,328	1024
s13207	44,255	747	1.7	0.1	53,920	1024
s15850	48,494	2660	5.5	0.5	39,936	1024
s35932	106,198	50	0.05	0.1	6,144	1024
s38417	120,180	20734	17.3	0.0001	47,616	256
s38584	115,855	2528	2.2	0.1	63,488	1024

Table 4: Area Overhead and Test Length For Built-In Reseeding (Transition Faults)

Circuit	Area	Reseeding Area	% Area Overhead	CIT	TL	STP
S1238	2,740	70	2.6	1	6,144	1024
S1488	3,555	107	3.0	1	6,368	1024
S1494	3,563	73	2.0	1	4,320	1024
S5378	14,376	1733	12.1	1	2,592	32
S9234	25,840	2599	10.1	1	2,688	32
s13207	44,255	1048	2.4	1	2,240	32
s15850	48,494	8698	17.9	1	86,016	1024
s35932	106,198	61	0.1	1	2,560	256
s38417	120,180	43261	36.0	1	5,728	32
S38584	115,855	4656	4.0	1	30,720	256

Our technique uses special hardware for the LFSR such that the reseeding circuitry area overhead is minimized. Also, The technique we presented is directly applicable to transition fault model. The simulation experiments show that the area overhead is very small for 100% SSF as well as transition fault coverage.

We also presented a reseeding algorithm that minimizes the area overhead. The algorithm takes care of seed selection and reseeding cycle selection. Our built-in reseeding technique achieves very low area overhead while satisfying 100% fault coverage for the benchmarks used. The simulation results show that -for most of the cases- the test length doesn't have to be maximized when the area overhead is minimized, which is a double win for our technique.

Acknowledgement

This work was supported by King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.

References

[Brglez 89] Brglez, F., C. Gloster, and G. Kedem, "Hardware-Based Weighted Random Pattern Generation for Boundary Scan," *Proc. of International Test Conference*, pp. 264-274, 1989.

[Chakrabarty 00] Chakrabarty, K., B. Murray, and V. Iyengar, "Deterministic Built-in Test Pattern Generation for High-Performance Circuits Using Twisted-Ring Counters," *IEEE Transactions of Very Large Scale Integration Systems*, Vol. 8, No. 5, pp. 633-636, Oct. 2000.

[Cheng 95] Cheng, K.-T., and C.J. Lin, "Timing-Driven Test Point Insertion for Full-Scan and Partial-Scan BIST," *Proc. of International Test Conference* pp.506-514, 1995.

[Crouch 95] Crouch, Alfred, and M. D. Pressly, "Self Re-seeding Linear Feedback Shift Register (LFSR) Data Processing System for Generating a Pseudo-random Test Bit Stream and Method of Operation," US Patent 5,383,143, Jan. 1995.

[Eichelberger 83] Eichelberger, E. B., and E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test", *IBM Journal of Research and Development*, Vol. 27, No. 3, pp. 265-272, May 1983.

[Fagot 99] Fagot, C., O. Gascuel, P. Girard and C. Landrault, "On Calculating Efficient LFSR Seeds for Built-In Self Test," *Proc. of European Test Workshop*, pp. 7-14, 1999.

[Hellebrand 92] Hellebrand, S., S. Tranick, J. Rajski, and B. Courtois, "Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *Proc. of International Test Conference*, pp. 120-129, 1992.

[Hellebrand 95] Hellebrand, S., 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 Transactions on Computers*, Vol. 44, No. 2, pp. 223-233, Feb. 1995.

[Hellebrand 00] Hellebrand, S., H.-G. Liang and H.-J. Wunderlich, "A Mixed Mode BIST Scheme Based on Reseeding of Folding Counters," *Proc. of International Test Conference*, pp. 778-784, 2000.

[Huang 97] Huang, L.-R., J.-Y. Jou, and S.-Y. Kuo, "Gauss-Elimination-Based Generation of Multiple Seed-Polynomial Pairs for LFSR," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 16, No. 9, pp. 1015-1024, Sep. 1997.

[Kagaris 99] Kagaris, D., and S. Tragoudas, "On the Design of Optimal Counter-Based Schemes for Test Set Embedding," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 18, No. 2, pp. 219-230, Feb. 1999.

[Kim 96] Kim, Kee, "Scan-Based Built-In Self Test (BIST) with Automatic Reseeding of Pattern Generator," US Patent 5,574,733, Nov. 1996.

[Koenemann 91] Koenemann, B., "LFSR-Coded Test Patterns for Scan Designs," *Proc. of European Test Conference*, pp. 237-242, 1991.

[Krishna 01] Krishna, C. V., A. Jas, and N. Touba, "Test Vector Encoding Using Partial LFSR Reseeding" *Proc. of International Test Conference*, pp. 885-893, 2001.

[Lempel 95] Lempel, M., S. Gupta and M. Breuer, "Test Embedding with Discrete Logarithms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 14, No. 5, pp. 554-566, May 1995.

[Li 01] Li, J. C.M., C.W. Tseng, and E.J. McCluskey, "Testing for Resistive Opens and Stuck Opens" *Proc. of International Test Conference*, 2001.

[McCluskey 85] McCluskey, E.J., "Built-In Self-Test Techniques," *IEEE Design & Test of Computers,* pp. 21-28, Apr. 1985.

[Muradali 90] Muradali, F., V.K. Agrawal, B. Nadeau-Dostie, "A New Procedure for Weighted Random Built-In Self Test," *Proc. of International Test Conference*, pp. 660-668, 1990.

[Needham 98] Needham, Wayne, C. Prunty, and E. Yeoh, "High Volume Microprocessor Test Escapes, An Analysis for defects our test are missing," *Proc. of International Test Conference*, pp. 25-34, 1998.

[Pradhan 99] Pradhan, D., and M. Chatterjee, "GLFSR – A New Test Pattern Generator for Built-in-Self-Test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 18, No. 2, pp. 238-247, Feb. 1999.

[Rajski 98] Rajski, J., J. Tyszer and N. Zacharia , "Test Data Decompression for Multiple Scan Designs with Boundary Scan," *IEEE Transactions on Computers*, Vol. 47, No. 11, pp. 1188-1200, Nov. 1998.

[Savir 90] Savir, J., and W. McAnney, "A Multiple Seed Linear Feedback Shift Register," *Proc. of International Test Conference*, pp. 657-659, 1990.

[Touba 95] Touba, N.A., and E.J. McCluskey, "Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST," *Proc. of International Test Conference*, pp. 674-682, 1995.

[Touba 96a] Touba, N.A., and E.J. McCluskey, "Test Point Insertion Based on Path Tracing," *Proc. of VLSI Test Symposium*, pp. 2-8, 1996.

[Touba 96b] Touba, N.A., and E.J. McCluskey, "Altering a Pseudo-Random Bit Sequence for Scan-Based BIST," *Proc. of International Test Conference*, pp. 167-175, 1996.

[Tseng 01] Tseng, C.W., and E.J. McCluskey, "Multiple Output Propagation Transition Fault ATPG" *Proc. of International Test Conference*, 2001.

[Venkataraman 93] Venkataraman, S., J. Rajski, S. Hellebrand, and S. Tarnick, "An Efficient BIST Scheme Based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers," *Proc. of International Conference on Computer-Aided Design*, Vol. 9, No. 6, pp. 572-577, 1993.

[Wunderlich 90] Wunderlich, H.-J., "Multiple Distributions for Biased Random Test Patterns," *IEEE Transactions on Computer-Aided Design,* Vol. 9, No. 6, pp.584-593, Jun. 1990.

[Zacharia 95] Zacharia, N., J. Rajski and J. Tyszer, "Decompression of Test Data Using Variable-Length Seed LFSRs," *Proc. of VLSI Test Symposium*, pp. 426-433, 1995.