BUILT-IN RESEEDING FOR SERIAL BIST
Ahmad A. Al-Yamani* and Edward J. McCluskey
*Stanford Center for Reliable Computing, Room 236, Gates Building 2A
353 Serra Mall, Stanford, California 94305-9020
Voice: (650) 723-1258, Fax: (650) 723-1451
{alyamani, ejm@crc.stanford.edu}
* - Contact author

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 equipment (ATE) 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. Also, the technique we present is applicable for transition faults as well as single-stuck-at faults. Built-in reseeding is based on expanding 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 a hardware synthesis algorithm and implementation.

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 [Eichelberger 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. Together with this ROM additional circuitry is required 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 uses mapping logic [Touba 95]. The strategy is to identify patterns in the pseudorandom sequence 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. The mapping eliminates the need for the on-chip ROM but it requires circuitry to perform the mapping for all the ATPG patterns generated for the faults that are not detected by pseudo-random patterns. Also, it requires hardware for detecting the patterns that don’t cause fault dropping and for applying the new values.

In this paper, we present a technique that combines mapping logic and reseeding (loading the LFSR 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. Our technique
utilizes the LFSR flip-flops for storing the seeds. The output of the circuit that detects when to reseed is used to pick the new seed by only 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 (LFSR) to insert deterministic patterns into the test set. On the other hand, our technique alters the contents of the LFSR to insert seeds, which produce the deterministic patterns.

Our technique can achieve perfect fault coverage and completely eliminates the need for external testing or for a ROM to store the seeds. Furthermore, it requires fewer seeds to be encoded compared to previous work. Actually, at best, 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 fewer 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 it reduces the seed storage because of the pseudorandom patterns applied between the seeds. The price for this storage reduction is paid in test length.

Our contributions in this paper are summarized as follows: (1) A reseeding technique that eliminates completely the need for external pattern storage 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 transition faults as well as single-stuck-at faults. (2) A hardware implementation for the given technique. (3) A reseeding algorithm based on the hardware implementation explained in Sec. 4. 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 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 discusses how the technique works with serial and parallel BIST architectures. Section 6 shows the simulation results. Section 7 concludes the paper.

2. Related Work

There are two ways to apply BIST for digital circuits; namely, test per clock (TPC) and test per scan (TPS). In TPC, the scan chain is reconfigured into an LFSR using additional XOR
gates. If pseudo-random patterns don’t give satisfactory coverage and additional patterns are to be applied, then every pattern needs to be as large as the scan chain. In TPS, a smaller LFSR is used. So, the additional patterns are encoded into smaller vectors (aka seeds) that are loaded into the LFSR. Among the advantages of TPS, is smaller storage for the seeds, not having additional fan-outs in the scan chain, and avoiding the routing problems for the feedback of the scan chains.

In TPS, the patterns are encoded into seeds by solving a linear system of equations, which is an algebraic representation of the linear expansion of the LFSR into the scan chain flip-flops. There are some linear dependencies between some flip-flops of the scan chain. Due to these dependencies, solving the linear system of equations may not be always possible.

2.1 Seed Calculation

Based on the statistical treatment of the linear dependencies in [Bardell 87], Konemann presented a technique for coding test patterns into LFSRs of size $S_{\text{max}}+20$, where $S_{\text{max}}$ is the maximum number of specified bits in the ATPG patterns. By adding 20 bits to $S_{\text{max}}$ as the size of the LFSR, the probability of linear dependence drops to 1 in a million [Koenemann 91].

The fact that many of the bits in the ATPG-patterns are don’t cares makes Koenemann’s technique very useful in reducing the storage needed per pattern to $S_{\text{max}}+20$. $S_{\text{max}}$ depends on the circuit under test and on the undetected faults that are targeted by the deterministic patterns.

In [Hellebrand 95], a scheme was presented for using Multiple-polynomial LFSRs (MP-LFSRs). 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 $S_{\text{max}}+4$ to achieve the same probability of finding an encoding as in the case of $S_{\text{max}}+19$ 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.

2.2 Seed Storage

In [Huang 97], programmable LFSRs (PLFSRs) are used to enable multiple polynomials for the PRPG. 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 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. 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. 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. By making use of the degrees of freedom in solving the linear system of equations the paper achieves higher encoding efficiency than static reseeding.

The above schemes assume that seeds are either applied from an external tester or stored in an on-chip ROM. This paper presents a technique for avoiding this requirement.

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.

2.3 Hardware-based reseeding

[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. 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 [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.

The above schemes have the advantage of diversity of the sequences from which the patterns are drawn. They also have the advantage of not requiring seed storage. However, They are good
for pseudorandom seeds. Our technique is based on deterministic seeds so 100% fault coverage can be achieved. We pay the price in hardware overhead compared to the above schemes.

2.4 Mapping logic

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 r.p.r. 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) Mapping logic needs hardware for all patterns. In built-in reseeding the LFSR runs in autonomous mode after loading every seed detecting more r.p.r. faults without having to perform more mappings. Because of this, some patterns may not be required. (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.

2.5 BIST for transition faults

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 faults coverage of 90% and 95%. The results are for combinational circuits.

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. The new seed is loaded by putting the LFSR in the state that precedes the seed value, so that at the next clock pulse, the new seed is in the LFSR.

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. By activating the select line of a given mux, the logic value in the corresponding LFSR stage is inverted.

The reseeding circuit needs to activate the select lines of the muxes in the locations where the current contents of the LFSR differ from the value of the seed. **Definition:** The end of sequence (EOS) contents are the values of the LFSR registers before reseeding. 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.

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 (LFSR with reseeding logic) 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 6th 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 $\bar{Q}_1\bar{Q}_2\bar{Q}_3\bar{Q}_4$. 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 occur in our desired sequence to generate the maximum possible number of don’t cares.

As an example, 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 c11 is different from c6 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 Q₁ and Q₂ only. The truth table for the reseeding circuit is shown in Table 1, where $Q_i$ comes from the output of the $i^{th}$ stage and $S_j$ goes into the select lines of the mux of $j^{th}$ stage. The table shows that when the LFSR has the contents 0101 (c6), $S_1$ and $S_2$ will be activated to load 1001 (c11) in the LFSR. The patterns between c6 and c11 will not occur so they are don’t cares. All the other patterns don’t activate any muxes.

![Figure 2: Example reseeding circuit (a) Select lines computation (b) Hardware implementation.](image-url)
### Table 1: Table of Combinations for the Reseeding Circuit Example

<table>
<thead>
<tr>
<th>$Q_1Q_2Q_3Q_4$</th>
<th>$S_1S_2S_3S_4$</th>
<th>$Q_1Q_2Q_3Q_4$</th>
<th>$S_1S_2S_3S_4$</th>
<th>$Q_1Q_2Q_3Q_4$</th>
<th>$S_1S_2S_3S_4$</th>
<th>$Q_1Q_2Q_3Q_4$</th>
<th>$S_1S_2S_3S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0</td>
<td>0 0 0 0</td>
<td>0 1 1 1</td>
<td>0 0 0 0</td>
<td>1 1 0 1</td>
<td>d d d d</td>
<td>0 1 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>0 0 0 0</td>
<td>1 0 1 1</td>
<td>0 0 0 0</td>
<td>0 1 1 0</td>
<td>d d d d</td>
<td>0 0 1 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
<td>0 1 0 1</td>
<td>1 1 0 0</td>
<td>0 0 1 1</td>
<td>d d d d</td>
<td>0 0 0 1</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>0 0 0 0</td>
<td>1 0 1 0</td>
<td>d d d d</td>
<td>1 0 0 1</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>d d d d</td>
</tr>
</tbody>
</table>

The resulting circuit for the example in Figure 2 is a 3-input AND as 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 complement 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 in one experiment, 58% of the defective microprocessors returned to intel had open defects. Our experimental results from the 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% SSF test sets was using transition faults propagated to all reachable outputs (TARO) [Tseng 01]. In this paper we present our built-in reseeding technique and we show how to apply it for both transition faults and SSF test sets.

There are two ways to apply transition fault test sets to circuits with scan chains. One way is to use pairs of 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 then 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 captured in the scan chain to be shifted out.
We use the first technique (launch on capture) for transition faults. The reseeding circuit only needs to load the LFSR with the 1st pattern because the 2nd pattern is the response of the logic to the 1st pattern. 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. This means that our technique requires no extra hardware to apply it to transition faults. Although pairs of 2 vectors are required for transition faults, only the first pattern needs to be encoded in hardware because the 2nd pattern is the circuit’s response to the 1st pattern.

Figure 3 shows where the reseeding circuit fits in a system level view of a circuit with an LBIST controller, which includes the additional control circuitry added for logic BIST. 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, where the LFSR is known in advance and the initial seed and test length are also known, the reseeding circuit may take its inputs from the pattern counter instead of the LFSR contents. The dashed line in Figure 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 $\lceil \log_2(TL) \rceil$. If the size of the pattern counter is much smaller than the LFSR, 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.](image-url)
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 drop more faults 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 applying the pseudorandom patterns after every deterministic pattern takes extra time on the tester where test time may be expensive. 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 pseudorandom 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.

**Definition:** *coverage improvement threshold* (CIT) is the improvement in fault coverage required by the algorithm to continue applying pseudorandom patterns. It’s a parameter to quantify the effectiveness of pseudorandom patterns. 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 before measuring the improvement in coverage. **Definition:** We define the *step size* as the number of patterns simulated before measuring the improvement in coverage. In our simulations, we used different values for the step size. Algorithm 1 shows the built-in reseeding algorithm.

The only user-specified parameters for this algorithm are the coverage improvement threshold (CIT) and the step size. 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 low hardware overhead for the reseeding circuit. 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, test length is considered very scarce. 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 as shown in Sec. 6. It’s 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 a generalization of mapping logic. If we choose
not to continue running the LFSR in pseudorandom mode after reseeding and we choose to alter
the LFSR output of the LFSR contents, then 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. We also chose to alter the
contents of the LFSR to enable a different sequence of pseudorandom patterns to drop more faults.

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. Test Per Clock vs. Test Per Scan

We performed our experiments on built-in reseeding using both test per clock (TPC) and test per scan (TPS). For TPC, the scan chain is reconfigured into an LFSR. TPC requires multiple test sessions and additional logic to ensure that the flip-flops capture their intended inputs. Among the disadvantages for TPC is the additional fanout in the scan chain. This fanout may cause additional delay on the critical path. Another disadvantage is having to route the feedback of the LFSR across the scan chain. A third disadvantage is the storage needed per pattern for additional patterns that are to be fed from the ATE or a ROM. One of the advantages of TPC is that it doesn’t require calculating the seeds; the seed is the ATPG pattern itself.

TPS overcomes the disadvantages mentioned above for TPC. The main disadvantages for TPS are the additional flip-flops for the LFSR and the requirement for calculating the seeds for the ATPG patterns. Why is calculating the seeds a problem? (1) It may not always be possible to find a seed for a given pattern due to linear dependencies. This is because some bits are linear combinations of others according to the function of the LFSR (2) By solving the linear system to calculate the seed, we give up many of the don’t care bits. There are some degrees of freedom in
choosing some bits of the seed. Once those bits are chosen, many other bits are fixed. By having less don’t cares the complexity of the reseeding circuit increases.

6. 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 2. The table shows the number of primary inputs, primary outputs and flip-flops in the circuits. It also shows the size of the LFSR used. The last column 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>
<thead>
<tr>
<th>Circuit Name</th>
<th>Primary Inputs</th>
<th>Primary Outputs</th>
<th>Scan Chain Size</th>
<th>LFSR Size for TPS</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1423</td>
<td>17</td>
<td>5</td>
<td>74</td>
<td>50</td>
<td>4,531</td>
</tr>
<tr>
<td>s1488</td>
<td>8</td>
<td>19</td>
<td>6</td>
<td>6</td>
<td>3,555</td>
</tr>
<tr>
<td>s1494</td>
<td>8</td>
<td>19</td>
<td>6</td>
<td>6</td>
<td>3,563</td>
</tr>
<tr>
<td>s5378</td>
<td>35</td>
<td>49</td>
<td>179</td>
<td>61</td>
<td>14,376</td>
</tr>
<tr>
<td>s9234</td>
<td>36</td>
<td>39</td>
<td>211</td>
<td>80</td>
<td>25,840</td>
</tr>
<tr>
<td>s13207</td>
<td>62</td>
<td>152</td>
<td>638</td>
<td>45</td>
<td>44,255</td>
</tr>
<tr>
<td>s15850</td>
<td>77</td>
<td>150</td>
<td>534</td>
<td>150</td>
<td>48,494</td>
</tr>
<tr>
<td>s35932</td>
<td>35</td>
<td>320</td>
<td>1728</td>
<td>60</td>
<td>106,198</td>
</tr>
<tr>
<td>s38417</td>
<td>28</td>
<td>106</td>
<td>1636</td>
<td>200</td>
<td>120,180</td>
</tr>
<tr>
<td>s38584</td>
<td>38</td>
<td>304</td>
<td>1426</td>
<td>200</td>
<td>115,855</td>
</tr>
</tbody>
</table>

6.1 Comparison with previous work

In order to evaluate the effectiveness of our built-in reseeding technique we performed some simulation experiments to compare it with previous work. Most of the previous work assumes one seed per pattern. In mapping logic, hardware is needed to map each pattern but no pattern or seed storage is needed. From here on, “previous work” refers to all work that assumes one seed per pattern.
We compare the number of seeds that need to be encoded if our built-in reseeding technique is used and the number of seeds that need to be stored or mapped if previous techniques are used. The number of seeds determines the area of the reseeding or mapping circuit. Since further area minimization heuristics can be applied to all techniques, it’s fair to compare them in terms of the number of seeds 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 built-in reseeding may require longer test length than 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 scarce resource 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. (4) The increase in test length we have is much smaller than that if only pseudorandom patterns were used. In fact the user has the option to minimize the test length for area overhead.

Table 3 shows a comparison of previous techniques and our reseeding technique. The table is for 100% single-stuck-at fault coverage, 1% CIT (coverage improvement threshold) 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 compared to using one seed per pattern. Overall, the minimum reduction in the number of patterns is 5%.

Table 3: Comparison of Built-In Reseeding and Previous Work Encoding One Seed per Pattern.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Number of seeds to be encoded</th>
<th>Test Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>One seed per pattern</td>
<td>Built-in reseeding</td>
</tr>
<tr>
<td>s1423</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>s1488</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>s1494</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>s5378</td>
<td>60</td>
<td>22</td>
</tr>
<tr>
<td>s9234</td>
<td>93</td>
<td>62</td>
</tr>
<tr>
<td>s13207</td>
<td>121</td>
<td>47</td>
</tr>
<tr>
<td>s15850</td>
<td>41</td>
<td>35</td>
</tr>
<tr>
<td>s35932</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>s38417</td>
<td>78</td>
<td>74</td>
</tr>
<tr>
<td>s38584</td>
<td>107</td>
<td>68</td>
</tr>
</tbody>
</table>
The table shows that there are many cases where reseeding offers a considerable improvement in the number of patterns required to test the circuit and at the same time doesn’t cause a large increase 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 encoded.

The following charts show how the coverage improvement threshold (CIT) affects previous work and our built-in reseeding technique for SSF. Fig. 4 shows the test length of previous work (top-off seeds) 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 using fewer random patterns and using more deterministic patterns for both techniques. The fewer 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 the CIT. Fig. 5 shows the number of patterns previous techniques (top-off seeds) require 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 many fewer seeds compared to using a small CIT. This is a significant advantage of built-in reseeding because it 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. The algorithm we presented in this paper picks the seed that is closest to the end of the 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.

Most of the circuits exhibited similar behavior to that shown in Fig. 4 and Fig. 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.
Fig. 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. Fig. 7 shows the average number of patterns (across all circuits) 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.
6.2 Area overhead and test length for SSFs

In this section we present the area overhead of our built-in reseeding technique using the SSF model. Table 4 shows the area overhead of the reseeding circuits for the benchmarks we used. The reseeding circuits given were designed based on 100% SSF fault coverage. The area overhead given in the table is the minimum area overhead we could achieve for 100% coverage after trying multiple CITs and step sizes.

For most of the circuits, the area overhead ranged from 0.05% to 4%. In a few cases we had a large reseeding circuit.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Area</th>
<th>Reseeding Area</th>
<th>% Area Overhead</th>
<th>TL</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1423</td>
<td>4,531</td>
<td>197</td>
<td>4.3</td>
<td>8,160</td>
</tr>
<tr>
<td>s1488</td>
<td>3,555</td>
<td>54</td>
<td>1.5</td>
<td>6,549</td>
</tr>
<tr>
<td>s1494</td>
<td>3,563</td>
<td>54</td>
<td>1.5</td>
<td>6,560</td>
</tr>
<tr>
<td>s5378</td>
<td>14,376</td>
<td>462</td>
<td>3.2</td>
<td>928</td>
</tr>
<tr>
<td>s9234</td>
<td>25,840</td>
<td>3,566</td>
<td>13.8</td>
<td>4,544</td>
</tr>
<tr>
<td>s13207</td>
<td>44,255</td>
<td>562</td>
<td>1.3</td>
<td>1,600</td>
</tr>
<tr>
<td>s15850</td>
<td>48,494</td>
<td>1,988</td>
<td>4.1</td>
<td>2,432</td>
</tr>
<tr>
<td>s35932</td>
<td>106,198</td>
<td>52</td>
<td>0.05</td>
<td>6,144</td>
</tr>
<tr>
<td>s38417</td>
<td>120,180</td>
<td>9,468</td>
<td>7.9</td>
<td>6,016</td>
</tr>
<tr>
<td>s38584</td>
<td>115,855</td>
<td>1,763</td>
<td>1.5</td>
<td>61,440</td>
</tr>
</tbody>
</table>

The minimum area overhead was achieved with different values for CIT for different circuits. 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.

An important conclusion from the above results is that we need an algorithm to efficiently find the optimal order for loading the seeds such that the area of the reseeding circuit is minimized. This is the subject for future research in the area.

6.3 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. The reseeding circuits given were designed based on the maximum
achievable transition fault coverage. The area overhead ranged from 0.1% to 12% in most cases. We will show the final table in the final version because we don’t have complete results yet.

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 it’s in pairs of patterns.

7. 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.

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 the transition fault model. The simulation experiments show that the area overhead is reasonable 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 satisfies 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.
References


