Seed Encoding with LFSRs and Cellular Automata

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

Abstract

Reseeding is used to improve fault coverage of pseudo-random testing. The seed corresponds to the initial state of the PRPG before filling the scan chain. In this paper, we present a technique for encoding a given seed by the number of clock cycles that the PRPG needs to run to reach it. This encoding requires many fewer bits than the bits of the seed itself. The cost is the time to reach the intended seed. We reduce this cost using the degrees of freedom (due to don't cares in test patterns) in solving the equations for the seeds. We show results for implementing our technique completely in on-chip hardware and for applying it from a tester. Simulations show that with low hardware overhead, the technique provides 100% single-stuck fault coverage. Also, when compared with conventional reseeding from an external tester or on-chip ROM, the technique reduces seed storage by up to 85%. We show how to apply the technique for both LFSRs and CA.

Categories and Subject Descriptors
B.8.1 [Integrated Circuits]: Reliability, Testing, and Fault Tolerance.
General Terms
Algorithms, Performance, Design, Reliability.
Keywords
VLSI Test, Built-In Self Test, Reseeding.

1. Introduction

Among the advantages of built-in self-test (BIST) are low cost compared to external testing using automatic test equipment (ATE), and applicability while the circuit is in the field. 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, Bardell 87].

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 improving BIST fault coverage. They are: (1) Modifying the circuit by test point insertion or by redesigning the circuit [Eichelberger 83, Touba 96], (2) *Weighted pseudorandom patterns*, where the random patterns are biased using extra logic to increase the probability of detecting r.p.r. faults [Eichelberger 89, 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 that target the undetected faults [Koenemann 91, Hellebrand 95, Touba 00]. Our technique is a mixed mode technique based on encoding the seeds in terms of the number of clock cycles required for the PRPG to reach them.

Modifying the CUT is often not possible because of performance issues or intellectual property rights. Weighted pseudo-random sequences require multiple weight sets. Mixed mode testing is done in several ways; one is to apply deterministic test patterns from a tester. Another technique is to store the deterministic patterns (or the seeds) in an on-chip ROM. There needs to be additional circuitry to apply the patterns in the ROM to the circuit under test.

Another technique for mixed-mode testing is *mapping logic* [Touba 00] where non-fault dropping patterns in the original set are mapped by hardware into deterministic patterns.

Reseeding refers to loading the PRPG with a seed that expands into a precomputed test pattern. We presented a technique for *built-in reseeding* (encoding the seeds in hardware) in [Alyamani 03a]. The technique combines mapping logic and reseeding and applies pseudorandom patterns between the deterministic seeds because there is a chance more faults will be detected.

A seed is a PRPG *initial state*. When the PRPG is loaded with this initial state, it loads the scan chains with the desired pattern after *m* clock cycles, where *m* is the length of the scan chains. We call the state of the PRPG after loading the scan chains the *final state*.

In [Alyamani 03b], we presented a technique for minimizing the number of seeds to be loaded by ordering the seeds and by exploiting the degrees of freedom in solving for the seeds. In this paper, our contributions are: (1) A seed encoding technique that encodes the seeds in a much smaller vector that corresponds to the number of cycles to reach it. The technique also exploits the degrees of freedom in solving for the seed. (2) An architecture that implements the encoding technique. The technique is applicable for SSF and transition faults. (3) An improvement over the built-in reseeding and seed ordering techniques to make them valid for any linear machine i.e., for LFSRs or cellular automata with or without phase shifters.

In Sec. 2 of this paper, we review the related literature. In Sec. 3, we present the seed encoding scheme. In Sec. 4, we present the architecture for built-in seed encoding and reseeding. Section 5 shows the simulation results and Sec. 6 concludes the paper.

2. Related Work

Konemann presented a technique for coding test patterns into PRPGs of size S_{max}+20, where S_{max} is the maximum number of specified bits in the ATPG patterns. By adding 20 to S_{max} as the size of the PRPG, the probability that test patterns cannot be coded into seeds drops to 1 in a million [Koenemann 91].

[Rajski 98] presented a reseeding-based technique that improves the encoding efficiency by using variable-length seeds. In [Krishna 01], the authors presented partial dynamic reseeding to incrementally modify the LFSR contents instead of modifying them all at once. This technique achieves higher encoding efficiency than static reseeding. The technique in [Alyamani 03a] encodes the seeds in hardware.

The technique presented in this paper tries to exploit the degrees of freedom in solving the linear system of equations for the seed to encode a seed by the number of additional clocks needed to reach it.

In [Lempel 95], an analytical method was presented for computing seeds for random pattern resistant circuits based on discrete logarithms. In [Fagot 99], a simulation scheme for calculating seeds for LFSRs was presented. The scheme is based on simulating several sequences that include a set of ATPG vectors.

In [Koenemann 00], a technique for skipping useless patterns is presented. The technique is based on having a Seed Skip Data Storage (SSDS) inside the tester. Fault simulation is performed to identify the *useful* (fault dropping) and *useless* (non fault dropping) sequences of patterns. Using additional control logic, the useless patterns are not loaded from the PRPG to the scan chains.

3. Seed Encoding

The BIST architecture we assume is shown in Figure 1. Our technique is applicable with any number of scan chains and any phase shifter. The results shown in Sec. 5 are for a single scan chain per circuit. However, the only difference if multiple scan chains were used is in the seed calculation. The way we calculate the seeds is explained in the appendix. The seeds are then either loaded from the tester or encoded in hardware on chip as explained in Sec. 4.

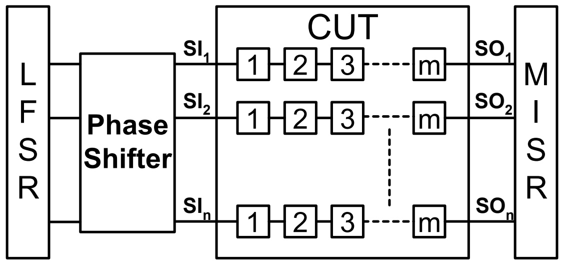

Figure 1: Multiple scan chains with a phase shifter

If the final state of the PRPG after loading the scan chain matches another seed, then that seed doesn't have to be loaded into the PRPG. If the final state of the PRPG doesn't match another seed, we can clock the PRPG a few times until we reach a match with one of the seeds. Also, instead of relying completely on randomness, we can exploit the degrees of freedom in solving the equations to generate the seed so that we increase the chances of matching a seed with the final state of the PRPG.

Let the state of the PRPG at time t be given by $s(t)$, then the PRPG state at time t+1 is given by $s(t+1)=s(t)H$, where H is called the transition matrix, and $s(t+1)=s(0)H^{t+1}$. If the longest scan chain is of length m, and seed i is $s_i(0)$, then the contents of the PRPG after the scan chains are loaded is given by $s_i(m+1)$. The PRPG can be a cellular automaton or an LFSR.

The encoding algorithm we present is based on looking ahead in the sequence of the PRPG by finding $s_i(m+1)$ for seed i and trying to find whether it matches with any of the other seeds $s_j(0)$, where $j \neq i$. If they match, then $s_j(0)$ doesn't need to be loaded into the PRPG. If a match is not found, we can search for a match with $s_i(m+d)$, where $1 \leq d \leq d_{max}$. The parameter d_{max} corresponds to the number of clock cycles we are willing to continue running the PRPG before loading the next seed. Choosing d_{max} = 1 means that if $s_i(m+1)$ doesn't match any of the remaining seeds, we will load a new seed.

The technique explained above requires changes to the BIST architecture to allow capturing after a different number of clock cycles. The architecture is presented in Sec. 4.

Given a set of seeds and a user-specified d_{max}, we order the seeds to minimize the number of seeds that need to be loaded as explained in [Alyamani 03b].

4. Seed Encoding Architecture

A fundamental issue in applying our seed encoding technique is how to make the PRPG run for a variable number of cycles for different seeds. Normally, the PRPG, runs for a number of cycles equal to the length of the longest scan chain before a capture cycle. To use our encoding technique, which represents the seed by the number of clock cycles required to reach it, we need to have the PRPG run for a variable number of cycles.

In a usual logic BIST architecture, a *bit counter* is used to choose when to disable the Scan Enable (SE) signal for capturing. One way to implement this is to have the bit counter loaded with the value that corresponds to the length of the longest scan chain for every pattern. The bit counter is then decremented by 1 at each clock cycle. When the bit counter reaches zero, it means that the test pattern is loaded into the scan chains, so SE is disabled for one clock cycle, and so on. The length of the scan chains is stored in a register and loaded into the bit counter with every pattern. Our technique is based on running the PRPG for a number of cycles to reach the desired seed. To implement this, we need to load the bit counter register with different values corresponding to the number of cycles before the next capture. Unloading the scan chains starts right after the capture cycle. So, for encoded seeds, there are extra cycles after unloading the response to pattern *i* and before capturing the response for pattern *i*+1.

4.1 Running the technique from an ATE

To run our technique from an external tester, we have two types of seeds, seeds that need to be loaded into the scan chain, *loaded seeds*, and seeds that can be reached by continuing to run the PRPG for additional cycles after loading the scan chains, *encoded seeds*. We use the name encoded seeds because these seeds are encoded into the number of cycles the PRPG needs to run to reach them.

<u>Seed size:</u> How efficient is this encoding? Why not just load all seeds? This question can be answered by a simple example, take a circuit of 10,000 flip flops that has 10 scan chains of length 1000 each. If the maximum number of care bits in the test patterns is 500 (5%), we need a PRPG of size 520 [Koenemann 91]. Since the length of the scan chain is 1000, the bit counter needs to have only 10 bits. So, by encoding the seed into the number of cycles to reach it we get a 98% (52×) reduction in seed storage. Even if we decide to run the PRPG for up to 1000 additional cycles before reaching the next desired seed, then impact on the size of the bit counter is a single bit.

<u>Test time:</u> In terms of test length, for the example above, loading the PRPG with a new seed takes 520 cycles. Loading the bit counter register takes 11 clock cycles. This means that we can search for a match with another seed in up to 508 cycles while saving on the test time and at the same time saving on tester storage.

If we only rely on luck in finding a match by clocking the PRPG, then we may not have a very good chance. That's why we exploit the degrees of freedom in solving the linear system of equations to force such a match as explained in the appendix.

4.2 Full BIST Implementation

Our technique can be applied for full on-chip BIST with 100% SSF fault coverage. For that, we need to have a reseeding circuit for loaded seeds and a seed encoding circuit for encoded seeds.

<u>Loaded seeds:</u> We use the built-in reseeding architecture presented in [Alyamani 03a]. The operation of the reseeding circuit is as follows: the PRPG starts running in autonomous mode according to the reseeding algorithm [Alyamani 03a]. Once it is time for reseeding, a seed is loaded into the PRPG, 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 PRPG in the state that precedes the seed value, so that at the next clock pulse, the new seed is in the PRPG.

Figure 2(b) shows the structure of the PRPG and its interaction with the reseeding circuit. For our technique, we use muxed flip-flops. By activating the select line of a given mux, the logic value in the corresponding stage is inverted. The muxed flip-flops are similar to those used for scan chains. The output of the reseeding circuit activates the select lines of the muxes to invert certain stages of the LFSR such that the desired seed is loaded in the next cycle.

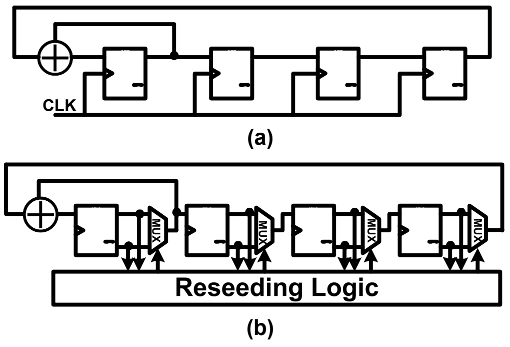

Figure 2: Reseeding circuit connection to LFSR: (a) A standard LFSR (b) LFSR with reseeding ckt.

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. In case of cellular automata, the same muxes structure can be used. The muxes should be placed right at the outputs of the flip flops before any XOR gates that are fed by the scan chain flip-flops. This way both polarities are available at the inputs of the muxes. Since XORs are linear gates, their outputs will be complemented by complementing any of the inputs, which satisfies the requirement for the above architecture to work. The connection of the reseeding logic to CA is shown in Figure 3.

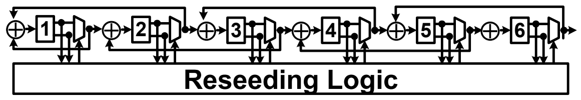

Figure 3: Reseeding circuit connection to CA

Encoded seeds: For the encoded seeds, we need a combinational circuit that takes as its input the value of the PRPG; the output of this circuit should be loaded into the bit counter register. In logic BIST architectures, a pattern counter is used to count the patterns applied to the circuit. Instead of reading the values of the PRPG stages as input to the seed encoding circuit, the value of the pattern counter can be used as input. The majority of the input combinations for the seed encoding circuit will load the bit counter register with the length of the scan chains. The pattern counter input combinations that correspond to seeds to be encoded will load a different value in the bit counter register. That value corresponds to the length of the scan chains plus the number of clock pulses that need to be applied before reaching the desired seed.

We can synthesize the seed encoding circuit from a table of combinations. The input combinations that correspond to normally loaded seeds should have the scan chains length as the output value. The input combinations that correspond to encoded seeds should have the output as the scan chains length in addition to the number of the clock cycles needed to reach to the seed. Just as in the circuit for built-in reseeding, all input combinations that won't occur in the desired test sequence can be treated as don't cares to help minimize the seed encoding circuit.

Figure 4 shows where the reseeding and seed encoding circuits fit in a system level view of a circuit with an LBIST controller, which includes the additional control circuitry added for logic BIST.

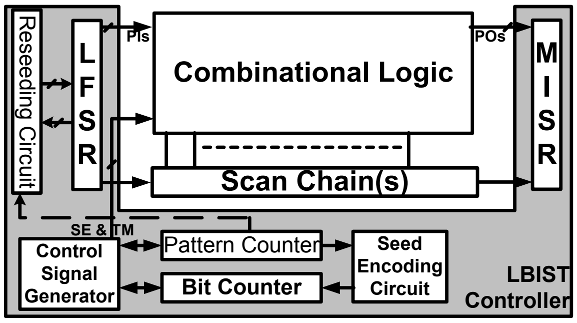

Figure 4: Reseeding and seed encoding circuits in a system view of BIST environment

If the CUT or the scan chain is changed, then the reseeding circuit and the seed encoding circuit need to be re-synthesized based on the new design and the new test patterns. However, if the seed encoding technique is applied from a tester, then changing the design only results in changes in the test patterns and accordingly the seeds that are stored. If it's preferable to apply the technique with full BIST, then it may be better to apply it after the design is stable and no more changes are applied to the circuit or the scan chain.

5. Simulation Results

In this section we present the results of some simulation experiments to evaluate our seed encoding technique. We performed our experiments on some of the 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, flip-flops in the scan chain, and in the LFSR. The BC column lists the sizes of the bit counters. We took into account the maximum number of additional clock cycles to calculate the size of the bit counter. The table 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.

The experiment was designed such that pseudo random patterns are applied first. Then, test patterns are generated for the undetected faults and the seeds are calculated from the test patterns.

Table 1: ISCAS 89 Circuits Used in Experiments.

CUT Name	PIs	POs	FFs	LFSR	BC	Cell Area
s953	16	23	29	21	7	2,286
s1196	14	14	18	15	6	2,722
s1238	14	14	18	15	6	2,740
s1423	17	5	74	50	8	4,531
s1488	8	19	6	6	6	3,555
s1494	8	19	6	6	6	3,563
s5378	35	49	179	61	9	14,377
s9234	36	39	211	80	10	25,840
s13207	62	152	638	45	10	44,255
s35932	35	320	1728	60	11	106,198

In [Koenemann 91, Hellebrand 95, Touba 00, Rajski 98, and Krishna 01] a single seed per pattern is assumed. In our technique, we encode as many of the seeds as we can by the number of additional clock cycles to reach them. The remaining seeds have to be loaded from the tester or encoded on-chip. Table 2 shows the number of test patterns generated for the undetected faults. The seed per pattern column shows the number of seeds that must be stored if a single seed per pattern is assumed as it is the case in most of the previous work. The table shows the number of seeds that must be stored and the seeds that need to be encoded with our technique.

Table 3 shows the seed storage needed for the seed per pattern scheme and the seed storage needed for our scheme. The storage is calculated by multiplying the number of loaded seeds by the PRPG size and the number of encoded seeds by the bit counter size. The table also shows the reduction gained by using our scheme. The reduction varies from 25% to 85%.

The area overhead required of our technique implemented completely on chip with reseeding and seed encoding circuits is comparable to the areas shown in [Alyamani 03b] where the overhead ranged between 0.3% and 10% and mostly less than 3%.

Table 2: Number of Seeds for Our Technique Compared to One Seed per Pattern.

Circuit	Seed per Pattern	Our Technique		
		Loaded Seeds	Encoded Seeds	Total
s953	47	9	16	25
s1196	92	13	16	29
s1238	97	12	12	24
s1423	17	9	0	9
s1488	89	4	9	13
s1494	85	3	11	14
s5378	35	26	1	27
s9234	117	82	6	88
s13207	182	30	16	46
s35932	7	1	0	1

6. Conclusion

In this paper, we presented a seed encoding technique based on running a variable number of clock cycles before loading the seed.

The simulation experiments showed that the storage needed is reduced by 25%-85% when our encoding technique is used compared to storing a single seed per pattern. The main characteristics that make our technique effective are: (1) Running pseudorandom patterns after loading the seeds (2) Ordering the seeds to load the minimum number of seeds, (3) Encoding the seeds by the number of cycles needed to reach them, and (4) Exploiting the degrees of freedom in solving for the seeds to match them with the current contents of the LFSR.

References

[Alyamani 03a] Al-Yamani A., and E. J. McCluskey, "Built-In Reseeding for Serial BIST", *VLSI Test Symposium*, Apr., 2003.

[Alyamani 03b] Al-Yamani A., S. Mitra, and E.J. McCluskey, "BIST Reseeding with Very Few Seeds", *VLSI Test Symposium*, Apr., 2003.

[Bardell 87] Bardell, P.H., W. McAnney, and J. Savir, "Built-In Test for VLSI", John Wiley, New York, 1987.

[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.

[Eichelberger 89] Eichelberger, E., E. Lindbloom, F. Motica, and J. Waicukauski, "Weigted Random Pattern Testing Apparatus and Method," US Patent 4,801,870, Jan. 1989.

[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 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.

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

[Koenemann 00] Koenemann, B., "System for Test Data Storage Reduction," US Patent 6,041,429, Mar. 2000.

[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.

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

[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.

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

[Touba 00] Touba, N. and E.J. McCluskey, "Altering Bit Sequence to Contain Predetermined Patterns," US Patent 6,061,818, May, 2000.

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

Table 3: Seed Storage Needed by our Technique Compared to Seed per Pattern.

Circuit	Circuit Cell Area	Seed per Pattern storage	Storage for Our Technique			Storage Rduction %
			Loaded Seeds	Encoded Seeds	Total	
s953	2,286	987	189	112	301	69.50
s1196	2,722	1380	195	96	291	78.91
s1238	2,740	1455	180	72	252	82.68
s1423	4,531	850	450	0	450	47.06
s1488	3,555	534	24	54	78	85.39
s1494	3,563	510	18	66	84	83.53
s5378	14,377	2135	1586	9	1595	25.29
s9234	25,840	9360	6560	60	6620	29.27
s13207	44,255	14560	2400	160	2560	82.42
s35932	106,198	420	60	0	60	85.71

Acknowledgement

This work was supported by King Fahd University of Petroleum and Minerals and by LSI Logic under contract No. 16517.

Appendix

In [Alyamani 03b], we presented seed calculation for LFSRs. In this section, we show how to apply the algorithm to cellular automata. Figure 5 shows an example for a 6-stage Cellular Automaton.

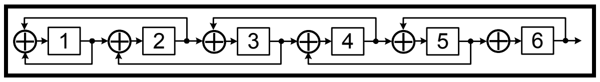

Figure 5: A 6-stage Cellular Automaton.

For every flip-flop in the scan chains, there is a corresponding equation in terms of the bits of the PRPG. Let's label the flip-flops of a given scan chain by $S_0 \rightarrow S_{m-1}$ where m is the length of the scan chain. Also, let's label the stages of the PRPG by $L_0 \rightarrow L_{n-1}$ where n is the size of the PRPG. In the example above, assume that the CA is connected to a scan chain at the output of stage 6 of the CA. Also assume that the scan chain has 9 stages $S_0 - S_8$. The equation for the deepest stage of the scan chain is $S_8 = L_5$, because after m clock cycles the most significant bit of the seed ends up in the most significant bit of the scan chain. The reader is invited to verify the remaining equations:

$S_7 = L_4$ $\qquad$ $S_6 = L_3 \oplus L_5$

$S_5 = L_2$ $\qquad$ $S_4 = L_1 \oplus L_3$

$S_3 = L_0 \oplus L_4$ $\qquad$ $S_2 = L_0 \oplus L_1 \oplus L_2 \oplus L_4$

$S_1 = L_2 \oplus L_5$ $\qquad$ $S_0 = L_1 \oplus L_3 \oplus L_4$

We can represent the above equations by an $m \times n$ matrix in which the rows correspond to the scan chain flip-flops and the columns correspond to the PRPG stages. An entry (i,j) is 1 if and only if L_j appears in the equation of S_i. According to this system, the following matrix shows the equations for all the flip-flops in the scan chain of the example above:

$$E = \begin{array}{c} \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \\ S_8 \end{matrix} \\ \begin{matrix} L_0 & L_1 & L_2 & L_3 & L_4 & L_5 \end{matrix} \end{array}$$

In the case of multiple scan chains, the outputs of the PRPG stages may need to go through a phase shifter to avoid structural dependencies that cause undesired correlation between patterns in different chains.

The phase shifter for a given scan chain is a linear sum of some stages from the PRPG. The phase shifter for chain i can be represented by a vector $p^i = \begin{bmatrix} p^i_{n-1} & p^i_{n-2} & \cdots & p^i_2 & p^i_1 & p^i_0 \end{bmatrix}$. p^i_j is one if the XOR feeding scan chain i has the output of stage j of the PRPG as one of its inputs.

The algorithm that generates the equations for a scan chain S that is fed through a phase shifter p^s starts with the vector p^s. Algorithm 1 is used to generate the equation matrix E_s.

The algorithm starts by assigning the vector p^s to the last row of E_s. The other rows are generated bottom up by multiplying the transition matrix H by the following row of E. This algorithm can be used with any number of scan chains.

The algorithm works with any PRPG and phase shifter. It depends on the transition matrix and the phase shifting vector, so it works with any linear machine.

```
1.  m: depth of the scan chains
2.  n: size of PRPG
3.  $p^s$: the phase shifter for the current scan chain
4.  $E_s$: equations matrix for the current scan chain
5.  $E_s$-Generator (m, n, h, $E_s$)
6.      $E^{(m-1)} = p^s$
7.      for i=m-2 to 0
8.              $E^i = H \times (E^{(i+1)})^T$
9.      endfor
10. end
```

Algorithm 1: Generating equations matrix E_s for scan chain s fed through a phase shifter

We exploit the degrees of freedom in solving the equations to increase the probability of finding a match between $s^{(m+1)}$ and the remaining seeds. The degrees of freedom are caused by the fact that the number of equations is less than the number of unknowns. This is a consequence of don't care bits in test patterns and the fact that the size of the PRPG depends on the maximum number of care bits.

The methodology for utilizing the degrees of freedom to increase the chances of finding a match between the final state of the PRPG and one of the remaining seeds is explained with an example in [Alyamani 03b].