Synthesis of Efficient Linear Test Pattern Generators

Avijit Dutta and Nur A. Touba

Computer Engineering Research Center
Department of Electrical and Computer Engineering
University of Texas, Austin, TX 78712

Abstract

*This paper presents a procedure for **S**ynthesis of **LIN**ear test pattern **G**enerators called **SLING**. SLING can synthesize linear test pattern generators that satisfy constraints on area, speed, internal fanout, and randomness properties and outperform existing linear test pattern generator designs including linear feedback shift registers (LFSRs) and cellular automatons (CAs). SLING is a constraint-driven synthesis procedure that takes as input a set of constraints and then synthesizes a test pattern generator that satisfies those constraints. SLING uses a set of linear transformations that it applies iteratively to evolve a linear test pattern generator. Because of the way the transformations are chosen and constraints are set, a high degree of phase shift is maintained between every pair of linear sequences generated at different bit positions of the generator and cross and auto correlations are highly minimized. Hardware overhead in terms of XOR gates is also minimized. Comparative analysis and experimental results show the effectiveness of the proposed synthesis scheme.*

1. Introduction

An efficient test pattern generator (TPG) is an essential component for built-in self-test (BIST) of digital integrated circuits. As suggested in [Mrugalski 03], to be effective, TPGs should fulfill the requirements of modular design, low area, fewer logic levels on the critical path, and reduced internal fan-out. In a multiple scan chain environment, in order to minimize test application time, different scan chains are fed by different stages of the TPG. If the outputs of successive stages of the TPG are highly correlated, then neighboring scan chains will contain correlated data which might result in reduced fault coverage. To alleviate this problem, the output sequences need to be de-correlated by a carefully designed phase shift network (PSN) [Rajski 98]. Linear feedback shift registers (LFSRs) [Bardell 87] are extensively used for random pattern generation in a BIST environment. However, while functioning in two dimensional (2D) mode where individual scan chains are driven by individual flip-flop of the generator, LFSRs display poor randomness properties. The output sequences generated at individual bit positions are highly correlated and a large PSN is required to achieve desired phase shift and reduce cross-correlation. Moreover, for external LFSRs (ELFSR) the depth of the linear logic could be large depending on the density (number of non-zero terms) of the polynomial being implemented. For internal LFSRs (ILFSR), the linear logic depth is 1, but the internal fan-out could be large depending on the polynomial being implemented. Another alternative is cellular automata (CA) [Chaudhuri 97] which display good randomness properties. Linear hybrid cellular automata (LHCA), which implement a combination of simple automata rules, have even better randomness due to reduced correlation [Rajski 99] between successive output values. However CA based TPGs suffer from high hardware overhead. Several variations of the basic LFSR based generators have been proposed in the literature to either improve speed or randomness. These include windmill machines [Warlick 80] which consist of multiple generators sharing a common stage, T-flipflop based LFSRs [Arvillias 79], hybrid designs [Wang 88], and more recently ring generators [Mrugalski 03], [Mrugalski 04] which apply *m*-sequence preserving transformations to attain a fast and highly modular implementation. In [Pradhan 99], generalized linear feedback shift registers (GLFSRs) were

Proceedings of the 21st IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT'06)
0-7695-2706-X/06 $20.00 © 2006 IEEE

proposed which use a Galois field GF(2^{δ}) where $\delta > 1$. However, GLFSRs and their variants suffer from high hardware overhead.

In [Kagaris 03], a built-in test pattern generation mechanism was proposed that can enforce an exact set of phase-shifts on the bit sequences produced at the successive stages while still maintaining low hardware overhead. The reduction in hardware was achieved by merging the original cellular automata (CA) logic with the required PSN. The synthesis procedure does not target all randomness properties and does not maintain any bound on the delay of the critical paths as well as the number of internal fan-outs.

In [Mrugalski 04], a planar high performance ring generator was proposed that uses an optimized structure called a dense ring generator to alleviate the problem of routing congestion and large logic depth associated with the linear logic used to construct phase-shifters. Ring generators implementing dense polynomials with good phase-shift properties are used for this purpose. However to drive *n* scan chains, this requires approximately *1.33n* flip-flops. The number of flip-flops can be reduced at the cost of relaxing some of the other bounds on the design properties.

In this paper, we propose a novel procedure for **S**ynthesis of **LIN**ear test pattern **G**enerators called (**SLING**) to design test pattern generators while satisfying several design constraints and randomness properties. The primary contribution in this work is that we describe a new class of linear generators that are formed by partitioning a TPG into equi-sized blocks and applying different transformations to all or some of the blocks. Depending on which set of transformations are applied to each block, many different linear generators can be obtained. Next a search procedure for obtaining a linear generator from this new class of linear generators that satisfies specified design constraints while optimizing area, delay, and/or randomness is described. The proposed synthesis procedure, SLING, provides the user a lot of flexibility to synthesize linear test pattern generators that meet the needs for a particular application. The highly equi-distributed patterns generated by SLING-synthesized generators ensure less correlated output sequences and large phase-shift while functioning as a 2D generator. In addition, an *n*-bit SLING synthesized generator can drive *n* or more than *n* scan chains in parallel depending on the transformations chosen. SLING is also amenable for applications requiring large-sized generators. A comprehensive analysis of different randomness properties is given, and we compare several different TPGs with respect to those properties.

2. Description of the SLING Procedure

In this section, we describe the SLING procedure in detail. We first provide an overview and then describe the different steps in a top-down fashion.

2.1 Overview

Any linear generator can be described by the following general linear recurrence (modulo 2):

$$X_{i+1} = \boldsymbol{A}X_i \qquad (1)$$

In the above equation, $X_i = (x_{i,0},\ldots,x_{i,k-1})^T$ is *k*-bit state. $\boldsymbol{A}$ is the $k \times k$ state transition matrix. All the operations are performed in Galois field modulo 2, GF(2). The period length of the recurrence has an upper bound 2^k-1 which is achieved when the characteristic polynomial corresponding to the state transition matrix $\boldsymbol{A}$ is a primitive polynomial over GF(2). If this condition is satisfied, then the generator is said to have the maximal period property. The proposed scheme involves partitioning the linear generator into blocks. If the size of the generator is *k*, then *k* flip-flops are required to represent the state of the generator. We partition *k* into *m*-bit blocks such that there are *n* such blocks, i.e., the state vector denoted by X_i is partitioned such that $X_i = (W_{i,0},\ldots,W_{i,n-1})^T$ and $(|W_{i,j}| = m)$. Then different transformations are applied on the *m*-bit blocks to construct the generator. The transformations that are used involve inexpensive binary operations like bit-shift, xor, bit-mixing, etc. The transformations are

described in detail in Sec. 2.4. Many different generators can be constructed depending on which set of transformations is used for each block.

SLING selects the particular generator based on the criteria the user specifies. The input to SLING consists of a set of design and randomness constraints that must be satisfied. The generator is constructed from a selected set of transformations that are combined using the proposed state transition matrix (STM). The design constraints include a bound on the number of 2-input XOR gates, a bound on the internal fan-out, and a bound on the depth of linear logic. The randomness properties of linear generators functioning in 2D mode also depend on the STM. The maximum logic depth in terms of 2-input XOR gates for the generator is $\log_2$(maximum number of 1's in any row) rounded to the next integer. The maximum internal fan-out is the maximum number of 1's in any column. The total number of 2-input XOR gates is $\sum_{\#rows}$(number of 1's in each row minus 1). By efficiently designing the STM, the user provided constraints on design properties are met (if possible). If the constraints are not satisfied within a specified search limit then no generator is returned. The randomness properties are checked by simulating the generator and analyzing the sequences generated at every bit position. For all the experiments, it is assumed that the generator functions in 2D mode where individual scan chains are driven by individual state-bit of the generator. The targeted randomness properties are cross-correlation, auto-correlation, equidistribution, phase-shift between pair of sequences generated at different bit positions, and dispersion capacity (i.e. average hamming distance between successive states).

2.2 State Transition Matrix

The state transition matrix (STM) is the most important design aspect for a 2D linear test pattern generator. While functioning in the 2D mode, the various randomness properties of the generated sequence depend on the STM. Moreover, the design properties are also determined by the STM. For an $n \times n$ STM, there are $2^{n\hat{}2}$ possible choices of matrices. Not all of them are useful. Firstly we are only interested in the primitive matrices, i.e., for which the corresponding characteristic polynomial is primitive. Also we want the STM to be as sparse as possible since a denser STM leads to more hardware overhead. Since there are competing choices of matrices (one matrix does not satisfy all the requirements), we should leave some scope for search while designing a STM so that we can search a limited but useful space for matrices and choose the best STM in terms of several randomness and design goals. The proposed STM template that is used to seed the search is shown in equation (2). The state vector is partitioned into n-blocks (n=8) where each block is m-bits. The M_i's in the STM are $m \times m$ sub-matrices corresponding to the different transformations chosen from the set of transformations T_0 through T_7 (described later). 0 is the all zero $m \times m$ sub-matrix, and I is the $m \times m$ identity matrix. From the STM template, it can be seen that if the sub-matrices M_4, M_5, M_{24} are full-rank (full row-rank and full column-rank), then the STM itself is full-rank irrespective of the other sub-matrices (M_i's). The easiest way to observe this is to set each M_i to all-zero matrices except M_4, M_5 and M_{24} which should be full-ranked. For a generator to be primitive, the corresponding STM must have full-rank (necessary but not sufficient). Our design of the STM satisfies this necessary condition by appropriately choosing only 3 sub-matrices. Also the Number of 0 sub-matrices is maximized to make the STM as sparse as possible. However there is a limit as to how sparse the STM can be. The sparseness impacts the randomness properties of the generator. A very sparse STM implies very few bit operations from one state to the next and hence there will very less bit-mixing and high degree of correlation between the generated sequences. Considering all these conflicting requirements, we propose the STM template as shown in the equation (2). Our experiments show that this particular STM template facilitates search in a limited but very effective space out of the space of all possible matrices and that targets a sufficiently large number of primitive matrices that can meet design and randomness goals. For all our

experiments, we used $n = 8$. Note that shift-induced cross correlation can be eliminated by appropriately choosing the linear transformations. This further provides an opportunity to use some of the intermediate transformed outputs to increase the number of scan chains that can be driven by the SLING synthesized generators. Next we discuss the search strategy that we employed to find primitive generators which satisfy several design and randomness properties.

$$\begin{bmatrix} M_0 & 0 & M_1 & 0 & M_2 & 0 & M_3 & M_4 \\ M_5 & 0 & M_6 & 0 & M_7 & 0 & M_8 & 0 \\ 0 & I & M_9 & M_{10} & M_{11} & M_{12} & M_{13} & 0 \\ 0 & 0 & I & M_{14} & M_{15} & M_{16} & M_{17} & 0 \\ 0 & 0 & 0 & I & M_{18} & M_{19} & M_{20} & 0 \\ 0 & 0 & 0 & 0 & I & M_{21} & M_{22} & 0 \\ 0 & 0 & 0 & 0 & 0 & I & M_{23} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & M_{24} & 0 \end{bmatrix} \quad (2)$$

2.3 Search Strategy

First the shift registers representing the state vector of the generator are partitioned into equi-sized blocks. Next linear transformations are chosen randomly for each M_i from the set of predefined hardware efficient linear transformations (T_0 through T_7 described in Sec. 2.4) and applied on the block outputs. Next they are combined using the STM template shown in Eqn. 2. It is ensured that the transformations M_4, M_5 and M_{24} are full-ranked. This helps in reducing the search space for primitive generators. The characteristic polynomial for the state transition matrix (A) of the generator is given by the following general equation:

$$P(u)=det(A-uI)=u^k-\beta_1 u^{k-1}-\ldots-\beta_{k-1}u-\beta_k \quad (3)$$

Where I is the identity matrix and $\beta_l \in$ GF(2). Maximal period is achieved if the characteristic polynomial corresponding to Eqn. 3 is primitive over GF(2). Once the STM is computed based on the chosen transformations, the characteristic polynomial can be computed based on Eqn. 3. Next the polynomial is tested for primitivity. We used the algorithm proposed in [Bardell 82] and [Lidi 86] to verify whether the computed characteristic polynomial in GF(2) is primitive or not. In this paper, we consider only primitive generators. The non-primitive generators are discarded during the search procedure. For each primitive generator found, we check whether the user defined design and randomness constraints are met or not. The design constraints include an upper bound on the number of 2-input XOR gates (C_1), an upper limit on the logic depth on any critical path (C_2), and an upper bound on internal fanout (C_3) for any of the memory elements. The randomness requirements include a lower bound on the measure of equidistribution (R_1), a lower bound on uniformity (R_2), a lower bound on dispersion capacity (R_3), a lower bound on phase-shift between every pair of sequences (R_4), an upper bound on average magnitude of auto (R_5) and cross (R_6) correlation coefficients for a chosen set of sequence pairs generated at different bit positions. For every generator found during search the cost function shown in Eqn. 4 is computed.

$$cost = \sum_{i=1..3} f_i(d_i-C_i) + \sum_{i=1..4} g_i(R_i-r_i) + \sum_{i=5..6} g_i(r_i-R_i) \quad (4)$$

where f_i corresponds to a linear function that is a normalized representation of its arguments for the i-th design constraint and g_i corresponds to a linear function that is a normalized representation of its arguments for the i-th randomness constraint. For every generator, the cost function is evaluated. For the randomness properties the generator is simulated and the bit sequences are analyzed. Generators for which the cost function evaluates to a non-zero positive value are removed from consideration as for those one or more of the constraints are not met. Only primitive generator with cost ≤ 0, are chosen.

2.4 Transformations

Here we present the various transformations used to construct the generators. Several fast bitwise transformations involving minimal hardware are used. The list of transformations that we apply on the different *m*-bit blocks is given below. Figure 1 shows some of the implementations of the transformations in hardware and the corresponding transformation matrices. All the transformations are linear and can be represented by $m \times m$. matrices. The transformations were chosen keeping in mind the randomness and design constraints. All the transformations are highly area efficient and at the same time are designed to meet several randomness requirements. The matrices corresponding to the transformations can be analyzed (beyond the scope of this paper) to characterize their contribution to the overall randomness of the generated sequences. Their rank, eigenvalue, and other properties affect the period, phase-shift between different sequences generated at different bit positions, bit-mixing, uniformity, equidistribution, and correlation of the generated sequences.

T_0: Y=0
T_1: Y=X
$T_2(t)$ $(t\geq 0)$: Y=(X<<t)
$T_2(t)$ $(t<0)$: Y=(X>>-t)
$T_3(t)$ $(t\geq 0)$: Y=X⊕(X<<t)
$T_3(t)$ $(t<0)$: Y=X⊕ (X>>-t)
T_4: Y=(X<<1)
T_5: Y=(X>>1)
$T_6(t)$ $(t\geq 0)$: Y=(X>>t)⊕(X<<(m-t))
$T_7(t_1,t_2)$ $(t_1\geq 0, t_2\geq 0)$:Y=(X<<t_1)⊕(X<<t_2)
$(t_1\geq 0, t_2<0)$:Y=(X<<t_1)⊕(X>>-t_2)
$(t_1<0, t_2\geq 0)$:Y=(X>>-t_1)⊕(X<<t_2)
$(t_1<0, t_2<0)$:Y=(X>>-t_1)⊕(X>>-t_2)

Table 1. 2-input XOR gate requirement for each transformation

Transformations	T_0	T_1	$T_2(t)$	$T_3(t)$	T_4	T_5	T_6	$T_7(t_1,t_2)$
2-input XOR count	0	0	0	m-|t|	0	0	0	m-(t_1-t_2)

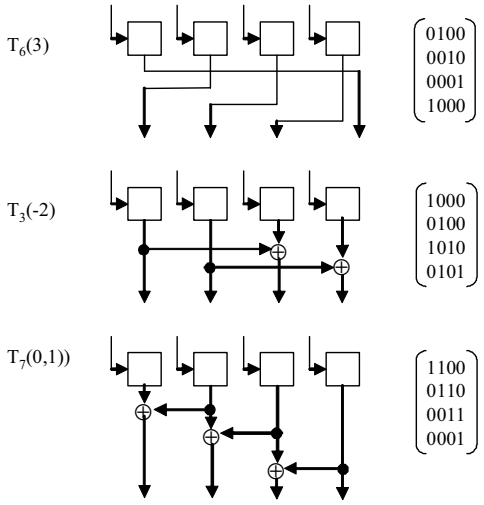

Figure 1. Hardware Implementation of Linear Transformations

Transformations T_0 sets the output to 0 irrespective of the state of the *m*-bit block. T_1 is the identity transformation. $T_2(t)$ is the left or right shift by *t* bits operation. $T_3(t)$ performs the xor of current state with a shifted version of the same. The number of XOR gates required for this transformation is $(m-|t|)$ where *m* is the block width. T_4 and T_5 are left shift by one bit position and right shift by one bit position respectively. $T_6(t)$ shuffles the current state by first shifting it right by *t*-bit positions and then shifting it left by $(m-t)$-bit positions and finally performing xor of the two shifted versions of the current state. $T_7(t_1,t_2)$ performs xor of the two shifted version

of the current state. The direction of the shift depends on the polarity of the arguments. Table 1 shows the XOR gate requirements for each of the transformations.

3. Comparing Randomness Properties

In this section, we compare the randomness properties of different types of linear TPGs. For a random sequence, any arbitrarily extracted subsequence should satisfy randomness properties. The behavior of a generator must be consistent across starting values (seeds). To ensure scalability and consistency, all the tests were performed on multiple sequences starting from randomly selected seed values.

3.1 Chaotic State Evolution

The evolution of the output sequences of finite state machines can be exemplified by means of time-state diagrams where each cell of the pattern generator is represented with a black pixel if the corresponding bit is a logical 1. Figure 2 shows the time-state diagrams for different 64-bit maximal-period generators. The diagrams correspond to an external LFSR (ELFSR) implementing a dense polynomial (29 non-zero coefficients), an internal LFSR (ILFSR) implementing the same polynomial, a sparse ILFSR, a linear hybrid cellular automaton (LHCA) with null boundary condition and implementing combinations of rule 90 and 150, and the SLING-synthesized generator corresponding to Table 3 (row 2). In each case, over 300 time steps were used with the same initial state having only a single bit set to 1. The LHCA was implemented using rule: $(2E635C255ABB8628)_H$ where 0 and 1 stand for CA rule 90 and 150 respectively. As can be seen, the state evolution of the SLING-synthesized generator is most chaotic. Dense ILFSR also have chaotic state evolution, but the internal fanout is unacceptably high.

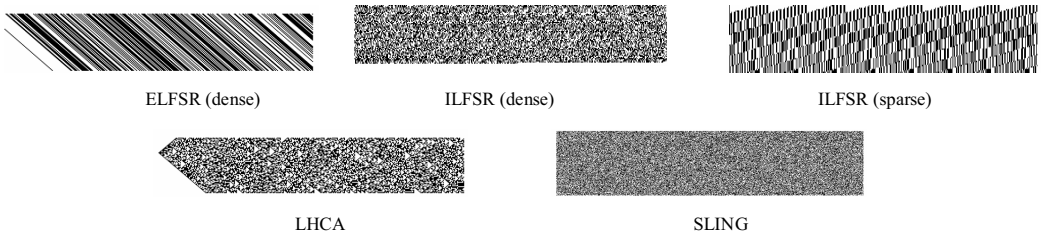

Figure 2. Evolution of States

3.2 Uniformity, Bit Mixing and Equidistribution

For a good pseudo-random TPG, the generated patterns should be equi-distributed. The average hamming distance between successive states should be close to the half of the size of the generator. A high dispersion capacity (i.e., a large hamming distance between successive states) ensures good bit-mixing. Figure 3 shows the average hamming distance for a set of 32-bit generators over 500 clock cycles starting from a skewed initial state. Clearly, the output sequence of the proposed generator is the most uniform and close to the optimal value (16 in this case). Due to high dispersion capacity (i.e., large hamming distance between successive states) the SLING-synthesized generator requires the least number of clock cycles to reach the uniform stage starting from a skewed state. This can be attributed to the fact that a large number of bit operations are performed from one state to the next when compared to other generators. Also the SLING-synthesized generators showed excellent equi-distribution properties when compared to the other generators.

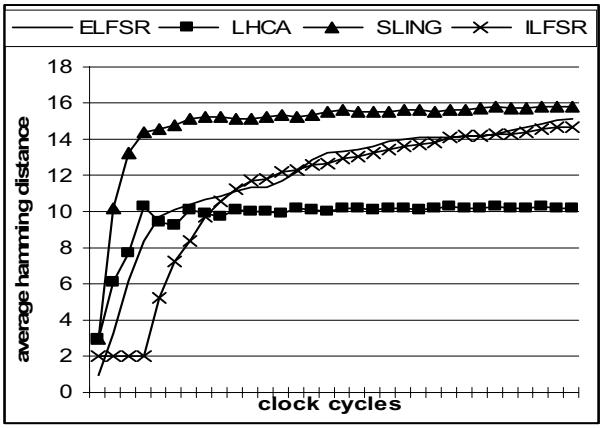

Figure 3. Average Hamming Distance

3.3 Correlation and Phase Difference

Auto-correlation and cross-correlation are two important measures of randomness. Auto-correlation is a measure of how well a sequence can differentiate between itself and a time-shifted variant of itself. Cross-correlation is defined as the correlation between two different sequences. Cross-correlation for TPGs can be measured by observing sequences generated at different bit positions of the generators. The correlation measure lies in the range [-1, 1] and the magnitude is bounded by the range [0, 1]. The cross-correlation between two sequences can be defined as (#agreements – #disagreements)/length of sequence. The more the magnitude of correlation deviates from zero, the higher the linear dependence.

Experiments were performed to compare the properties of different generators. Results are shown in Tables 2, 3, and 4 for generators of size 32, 64, and 128, respectively. Column 4 in Tables 2, 3 and 4 shows several cross-correlation measures. The average cross-correlation is measured in the following way. Every pair of sequences is time shifted by a random amount and then their cross-correlation is computed. In the first sub-column, the random amounts of time shift range from 0 to the generator size, and for the second sub-column, the random amounts of time shift ranges from 0 to 1000. This process is repeated 100 times and the average value is reported. In each case, the number of pairs across each trial for which the cross-correlation exceeded 0.04 is reported. In all cases, the SLING synthesized generators have the least number of sequence pairs exceeding the upper-bound on cross-correlation and have the minimum average cross-correlation.

4. Comparing Design Properties

In this section, we compare area in terms of 2-input XOR gates, maximum logic depth, and maximum internal fanout for several primitive generators. In [Kagaris 05], the different techniques presented in [Mrugalski 00] and [Rajski 98] are generalized and a unified approach is proposed for designing a PSN to achieve a precomputed phase-shift between every pair of sequences for any LFSM. Using this technique, a PSN can be designed for each of the generators to achieve similar randomness properties in terms of phase-shift (>=10000) and cross-correlation. The last column of Tables 2, 3, and 4 shows the number of 2-input XOR gates that are required for each generator including the required PSN. As can be seen, the SLING synthesized generators require the least amount of hardware in all cases for achieving a particular level for randomness. Moreover, the comparative hardware efficiency increases as the size of the generator increases. The operational speed of the generators is determined by the number of XOR gate delays on the feedback path and also the internal fanout of the memory elements. For ILFSRs and ring generators, the XOR gates are interspersed between the memory

elements and hence the maximum logic depth is 1. For an ELFSR with a balanced binary tree implementation of the external gate, the number of logic levels in the feedback path is $log_2 d$ where d is the number of non-zero coefficients of the characteristic polynomial. The maximum internal fanout for an ELFSR is 2. ILFSRs suffer from large internal fanout specially when implementing a dense polynomial. While operating in 2D mode, the operational speed of LFSRs is limited by the maximum of the logic depth or the depth of the PSN. Columns 2 and 3 of Table 2, 3, and 4 shows the logic depth and internal fanout of several standalone (without PSN) generators. Note that in all cases an n-bit generator drives n scan chains in parallel. The dense ring generator proposed in [Mrugalski 04] displays better design properties but to drive n scan chains in parallel on average $1.33n$ flip-flops are required. Note that the sparse ELFSR and ILFSR in Table 2 implements a primitive polynomial with 3 non-zero coefficients and the dense LFSRs implements a primitive polynomial with 27 non-zero coefficients. The LHCAs implement primitive rules: (rule1: $4609BBD5_H$) and (rule2: $6030E230_H$). The sparse ELFSR and ILFSR in Table 3 implement a primitive polynomial with 3 non-zero coefficients and the dense LFSRs implements a primitive polynomial with 45 non-zero coefficients. The LHCAs implement primitive rules: (rule1: $2E635C255ABB8628_H$) and (rule2: $1461DD5AA43AC674_H$). The dense ELFSR and ILFSR in Table 4 implements a primitive polynomial with 73 non-zero coefficients. The LHCA implement primitive rule: ($48882FBD67031A7A7A79C0E6BDE41112_H$).

Table 2. Comparing 32-bit Generators

Generator	Max Logic Depth	Max Internal Fanout	Time Shift				Num. 2-Input XOR
			0 to SIZE(32)		0 to 1000		
			Avg	Cor > 0.04	Avg.	Cor > 0.04	
SLING	2	4	.0081	6	.0079	0	60
SLING	2	4	.0080	5	.0078	1	74
ELFSR (sparse)	2	2	.0264	479	.0091	32	96
ELFSR (dense)	5	2	.0248	452	.0080	22	92
ILFSR (sparse)	1	3	.0139	147	.0083	6	90
ILFSR (dense)	1	27	.0085	11	.0080	3	72
LHCA (rule1)	2	3	.0082	7	.0080	2	78
LHCA (rule2)	2	3	.0081	6	.0080	1	64

Table 3. Comparing 64-bit Generators

Generator	Max Logic Depth	Max Internal Fanout	Time Shift				Num. 2-Input XOR
			0 to SIZE(128)		0 to 1000		
			Avg	Cor > 0.04	Avg.	Cor > 0.04	
SLING	2	5	.0082	12	.0079	3	162
SLING	2	4	.0083	15	.0078	5	176
ELFSR (sparse)	2	2	.0180	1028	.0092	88	210
ELFSR (dense)	6	2	.0179	998	.0080	72	198
ILFSR (sparse)	1	3	.0158	918	.0078	70	208
ILFSR (dense)	1	45	.0079	49	.0074	6	180
LHCA (rule1)	2	3	.0080	13	.0079	7	188
LHCA (rule2)	2	3	.0082	19	.0079	11	180

Table 4. Comparing 128-bit Generators

Generator	Max Logic Depth	Max Internal Fanout	Time Shift				Num. 2-Input XOR
			0 to SIZE(32)		0 to 1000		
			Avg	Cor > 0.04	Avg.	Cor > 0.04	
SLING	2	5	.0079	32	.0079	22	302
ELFSR	7	2	.0132	2153	.0087	409	456
ILFSR	1	73	.0094	606	.0080	390	360
LHCA	2	3	.0080	36	.0080	27	376

5. Conclusions

By choosing the design and randomness constraints appropriately, highly effective and application specific test pattern generators can be designed using SLING. The block-based approach can also be exploited for highly regular design and efficient routing.

Acknowledgements

This material is based on work supported in part by the National Science Foundation under Grant No. CCR-0306238 and CCR -0426608.

References

[Arvillias 79] Arvillias, A., and D. Maritsas, "Toggle-Registers Generating in Parallel *k*th Decimations of m-sequences," in *IEEE Trans. Computers*, Vol. 28, No. 2, pp. 89-101, 1979.

[Bardell 87] Bardell, P.H., W. H. McAnney, and J. Savir, "Built-in Test for VLSI: Pseudorandom Techniques," *John Wiley & Sons,* 1987.

[Chaudhuri 97] Nandi, S., and P.P., D. Chowdhury, "Additive Cellular Automata as an on Chip Test Pattern Generator," *Proceedings of 2nd Asian Test Symposium*, pp. 166-171, 1997.

[Kagaris 03] Kagaris, Dimitri., "Built-In TPG with Designed Phase Shifts," *Proc. of VLSI Test Symposium*, pp. 365–370, 2003.

[Kagaris 05]] Kagaris, Dimitri., "A Unified Method for Phase Shifter Computation," *ACM Trans. on Design Automation of Electronic Systems*, Vol. 10, No. 1, pp. 157-167, January, 2005.

[Lempel 71] Lempel, A., and W. Eastman, "High Speed Generation of Maximal Length Sequences," *IEEE Trans. Computers,* Vol. 20, No. 2, pp. 227-229, 1971.

[Lidi 86] Lidi, R., and H. Neiderreiter, "Introduction to Finite Fields and Their Applications," *Cambridge University Press*, 1986.

[Mrugalski 00] Mrugalski, G., J. Rajski, and J. Tyszer,. "Cellular Automata Based Test Pattern Generators with Phase Shifters*," IEEE Trans. on Computer-Aided Design*, Vol. 19, No. 8, pp. 878-893, 2000.

[Mrugalski 03] Mrugalski, G., J. Rajski, and J. Tyszer, "High Speed Ring Generators and Compactors of Test Data," *Proc. of VLSI Test Symposium*, pp. 57–62, 2003.

[Mrugalski 04] Mrugalski, G., J. Rajski, and J. Tyszer, "Planar High Performance Ring," *Proc. of VLSI Test Symposium*, pp. 193–198, 2004.

[Pradhan 99] Pradhan, D., and M. Chatterjee, "GLFSR: a New Test Pattern Generator for Built-In-Self-Test," *IEEE Trans. on Computer-Aided Design*, Vol.18, No. 2, pp. 238–247, 1999.

[Rajski 98] Rajski, J., N. Tamarapalli, and J. Tyszer, "Automated Synthesis of Large Phase Shifter for Built-in Self-test," *Proc. of International Test Conference*, pp. 1047–1056, 1998.

[Rajski 99] Rajski, J., G. Mrugalski, and J. Tyszer, "Comparative Study of CA-Based PRPGs and LFSRs with Phase Shifters," *Proc. of VLSI Test Symposium*, pp. 236–245, 1999.

[Wang 88] Wang, L.-T., and E. McCluskey, "Hybrid Designs Generating Maximum-length Sequences," *IEEE Trans. on Computer-Aided Design*, Vol. 7, No. 1, pp. 91–99, 1988.

[Warlick1 80] Warlick, W., and J. Hershey, "High speed m-sequence generators," *IEEE Trans. Comput*., Vol. 29, No. 5, pp. 398–400, 1980.

[Wohl 03] Wohl, P., and J.A., Waicukauski, S. Patel, and M.B. Amin, "Efficient Compression and Application of Deterministic Patterns in a Logic BIST Architecture," *Proc. of Design Automation Conference*, pp. 566–569, 2003.