Pseudo Random Patterns Using Markov Sources for Scan BIST

Nadir Z. Basturkmen [*] Sudhakar M. Reddy [*]
ECE Department
University of Iowa
Iowa City, IA 52242

Irith Pomeranz [†]
School of ECE
Purdue University
West Lafayette, IN 47907

Abstract

We propose a new pseudo-random pattern generator for scan circuits. The proposed generator uses Markov sources to capture spatial correlations between consecutive bits inside a scan chain as defined by weight sets generated using a weighted random pattern testing method. The weight set generation is based on the analysis of deterministic test sets. The BIST scheme that uses the proposed pattern generator iteratively modifies the generator behavior to obtain a full fault coverage. Experiments conducted on large benchmark circuits demonstrate that the proposed BIST methodology can achieve full fault coverage with a small number of tests and a small hardware overhead.

1 Introduction

The ever increasing complexity of today's integrated circuits highly complicates the testing process. Built-in Self Test (BIST) offers solutions to these problems. Test-per-scan BIST, where internal storage elements are reconfigured as scan chains in test mode and tests are applied by shifting them through these chains, is particularly attractive due to its effectiveness and ease of implementation. Pseudo-random patterns are often used as test patterns because they are easily generated in a BIST environment in a cost effective way by employing linear feedback shift registers (LFSRs). However, due to the existence of *random pattern resistant* faults, additional measures such as inserting test points [9, 10] and/or "topping of" deterministic tests are usually needed to achieve satisfactory fault coverages.

Weighted random pattern techniques [1, 2, 3] apply pseudo-random patterns which have biased 0 and 1 probabilities at various circuit inputs to facilitate the detection of otherwise random pattern resistant faults. However, the hardware implementation cost of weighted random pattern testing can be high. For ease of implementation, some techniques employ only 3 special weights, 0, 1, and 0.5 [4, 5]. In [15], in addition to these 3 weights, weights of 2^{-k} and $1.0 - 2^{-k}$ are also used, for $k > 1$.

It is also possible to encode deterministic tests in certain ways to achieve high fault coverage. In [6] Koenemann used LFSR seeds to ensure that certain deterministic test vectors are reproduced during test application. Similarly, in [7] and [8] multiple-polynomial LFSRs are used to encode deterministic test vectors.

In [16], by adding logic, Touba and McCluskey alter the serial output of an LFSR that feeds a scan chain, in order to modify the patterns that do not detect any faults into deterministic vectors that detect hard to detect faults. Wang uses a perturbation technique in [17] to reproduce deterministic test vectors from stored test cubes using 3 weights. Brglez et. al. use additional hardware at the input of a scan chain to provide a specific weight for every bit position in the chain [13].

In this paper, we describe a new BIST methodology for full scan circuits. To achieve 100% fault efficiency we design a pseudo-random pattern generator which generates bit sequences that satisfy input weight assignments as determined by a weighted random testing methodology [3]. The unique feature of the proposed methodology is that it uses a Markov source for pseudo-random patterns. The outputs of the source have temporal dependencies. The temporal dependencies translate into spatial correlations in the scanned test vectors. This is useful since it allows us to reproduce bit patterns that exist in deterministic tests. Previously, Markov models are used for sequential BIST [18]. However, unlike our method, Markov models described in

[*]Research supported in part by NSF Grant No. CCR-0097905 and in part by SRC Grant No. 2001-TJ-949.

[†]Research supported in part by NSF Grant No. CCR-0098091 and in part by SRC Grant No. 2001-TJ-950.

this work ensures the traversal of test cubes in a certain order to regenerate test subsequences in a reasonable amount of time.

Unlike several other methods, our method does not encode or reproduce any deterministic test vector in full. Rather, it is guided by deterministic test vectors to capture the statistics of the pseudo-random bit stream that will detect most of the faults in the circuit. The test generation process does not require any special consideration. The only requirement is that unnecessary bit positions should be left unspecified by the ATPG tool. Our method involves an iterative approach that modifies the behavior of the pseudo-random pattern generator (PRPG). At every step, statistics of the generated pseudo-random (PR) sequence are matched to the requirements of the remaining faults. Thus, the method is capable of achieving 100% fault efficiency.

The rest of the paper is organized as follows: Section 2 outlines the proposed BIST methodology. Section 3 gives a brief background and discusses the design details of the proposed pseudo-random pattern generator. Experimental results are presented in Section 4. Section 5 concludes the paper.

2 Proposed BIST Scheme

In this section we describe the BIST methodology based on the generation of weighted random patterns. Figure 1 shows a CUT with m pseudo inputs. We assume that the CUT is a full-scan circuit. For convenience of discussion, we also assume that the original primary inputs of the CUT are driven by the scan elements during testing.

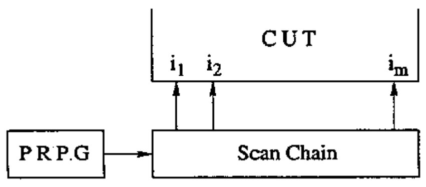

Figure 1. PRPG Driving a scan chain

Let the inputs of the CUT be $I = i_1, i_2, ..., i_m$. Suppose that a corresponding weight set calculated for pseudo-random testing of the CUT is $W = w_1, w_2, ..., w_m$. Our goal is to design a PRPG such that the weights required by weight set W at inputs I are satisfied. The flow of the proposed BIST methodology is given in Figure 2. The flow starts with the generation of deterministic test vectors for the remaining faults in the circuit. Then, a weight set that will detect these faults in a short period of time is calculated. These calculations are used to guide the PRPG design, that will produce PR bit sequences to match input weight requirements. Finally, detected faults are dropped by fault simulation.

It is a well known fact that many circuits require more than one weight set to achieve reasonable coverage [2, 11, 3]. Therefore, the BIST methodology we use involves an iterative approach. In each iteration, a new weight set is calculated for the remaining faults in the CUT and the PRPG parameters are set accordingly.

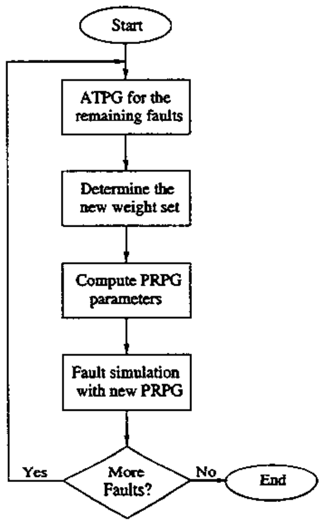
Start
ATPG for the remaining faults
Determine the new weight set
Compute PRPG parameters
Fault simulation with new PRPG
More Faults?
Yes
No
End

Figure 2. Flow of the proposed BIST scheme

A block diagram of the proposed scheme is given in Figure 3. Each iteration in the BIST flow corresponds to a different phase, and each phase is terminated according to a stopping criterion during fault simulation. As shown in the block diagram, phases are decoded by a phase decoder which follows the BIST pattern counter and changes phases at the right pattern counts. In each phase, the correct parameters for that phase are selected and loaded to a finite state machine (FSM) that produces the pseudo-random bit sequence. These phase parameters control weighted random pattern generator. Each phase parameter corresponds to a different probability of 1's in the output space of a weighted random pattern generator which determines the FSM behavior. The output of the FSM is fed into the scan-chain.

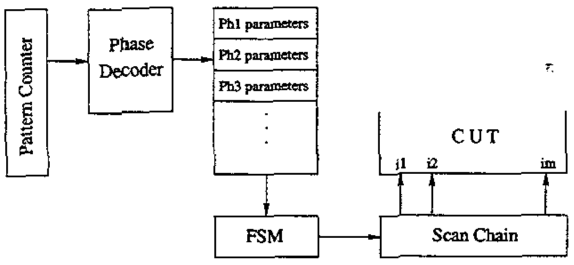

Figure 3. Block diagram of the proposed BIST scheme

In the next section we will discuss the design of the FSM and the selection of the parameters.

3 PRPG Design

We use time-homogeneous discrete-time Markov chain models to produce the bit sequences fed into the scan chain. We include an overview of the subject.

3.1 Background

A stochastic process $X(t)$ is a *Markov process* if for any set of $n+1$ values $\{t_1 < t_2 < ... < t_n < t_{n+1}\}$ in the index set and any set $\{x_1, x_2, ..., x_{n+1}\}$ of $n+1$ states

$$\begin{aligned}&P[X(t_{n+1}) = x_{n+1}| \\ &\quad X(t_1) = x_1, X(t_2) = x_2, ..., X(t_n) = x_n] = \\ &P[X(t_{n+1}) = x_{n+1}|X(t_n) = x_n] \qquad (1)\end{aligned}$$

The above equation gives the state transition probability from state n to state $n+1$, and indicates that the future of the process depends only on the present state. A Markov process is called a *Markov chain* if the state space is discrete. In discrete-time notation, the variable t can be omitted and one-step state transition probability from state i to state j can be written as:

$$P[X_{n+1} = j|X_n = i] \qquad n, i, j = 0, 1, 2, ... \qquad (2)$$

which is dependent on n. If the one-step transition probabilities are independent of n the Markov chain is said to be homogeneous in time and have stationary transition probabilities. For such a chain the one-step transition probability can be denoted by p_{ij}. The transition probabilities of a Markov chain can be written as a square matrix:

$$\mathbf{P} = \begin{bmatrix} p_{00} & p_{01} & p_{02} & \cdots \\ p_{10} & p_{11} & p_{12} & \cdots \\ p_{20} & p_{21} & p_{22} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \qquad (3)$$

This is called the transition probability matrix of the chain. For this matrix, the following equations must hold:

$$p_{ij} \geq 0 \qquad i, j = 0, 1, 2, ... \qquad (4)$$

$$\sum_{j=0}^{\infty} p_{ij} = 1 \qquad i = 0, 1, 2, ... \qquad (5)$$

For a Markov chain with stationary transition probabilities the matrix equation $\pi = \pi\mathbf{P}$ is satisfied. That is,

$$\pi_j = \sum_i \pi_i p_{ij} \qquad j = 0, 1, 2, ... \qquad (6)$$

where $\pi = (\pi_1, \pi_2, ...)$ is the state probability distribution and $\sum_i \pi_i = 1$.

The statistical properties of a Markov chain can be determined by specifying the entries of the matrix $\mathbf{P}$, that is, by specifying the stationary transition probabilities. In our analysis, we assign certain bit sequences to Markov chain states. The first bit in the sequence associated with a state is also the output value produced by the state. We determine the necessary transition probabilities so as to produce a PR bit sequence with desired statistical properties, i.e., a PR bit sequence that satisfies weight requirements at the circuit inputs as calculated by a weighted random pattern generation technique. We note that the complexity of the design increases as the number of states used in the Markov chain is increased. The ability of the Markov chain to generate the desired PR bit sequences also increases as the number of states is increased.

Here, we discuss the design of PRPGs using Markov chains with 2 and 4 states. A finite state machine (FSM) is used to realize the Markov chains.

3.2 2-state FSM

Figure 4 shows a 2-state Markov chain model we have used to generate PR bit sequences. The states give the current bit included in the PR sequence. State probabilities π_0 and π_1 correspond to the 0 and 1 probabilities (P_0 and P_1) of the generated bit sequence. Since $p_{00} = 1.0 - p_{01}$ and $p_{11} = 1.0 - p_{10}$, the output sequence can be entirely characterized by specifying p_{01} and p_{10}. At the steady state, the transition

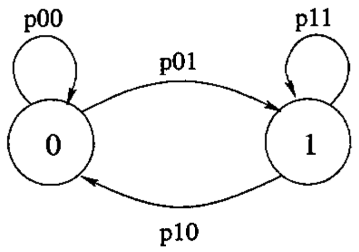

Figure 4. 2-state Markov Chain Model

probabilities must be equal, therefore we have:

$$\pi_0 p_{01} = \pi_1 p_{10} \tag{7}$$

or

$$\frac{\pi_0}{\pi_1} = \frac{p_{10}}{p_{01}} \tag{8}$$

Hence, the 0 and 1 probabilities of the output sequence are determined by the ratio of the transition probabilities p_{01} and p_{10}. However, it is possible to generate bit sequences that look quite different by selecting different state transition probabilities, although the sequences contain similar numbers of 0s and 1s. Suppose that we select $p_{01} = p_{10} = 1.0$. Then the generated sequence will be: 10101010101010101010. The sequence has $P_0 = P_1 = 0.5$. If $p_{01} = p_{01} = 0.1$, the generated sequence may look like: 00000000001111111111. Again, the sequence has $P_0 = P_1 = 0.5$. Therefore, it is essential to select the absolute values of the state transition probabilities correctly, as well as their ratio.

We determine the required one probability P_1 of the output sequence by ensuring that π_1 minimizes the following error function:

$$\sum_{i=1}^{m} (w_i - \pi_1)^2 \tag{9}$$

where $\{w_1, w_2, ..., w_m\}$ is the calculated weight set. The value of π_1 that minimizes this error function is equal to:

$$\frac{\sum_{i=1}^{m} w_i}{m} \tag{10}$$

Then, $P_1 = \pi_1$ and $P_0 = 1.0 - P_1$.

Similarly, we determine the required transition probability $\pi_1 p_{10}$ of the output sequence by selecting its value to minimize the following error function:

$$\sum_{i=m-1}^{1} ((1.0 - w_i)w_{i+1} - \pi_1 p_{10})^2 \tag{11}$$

$(1.0 - w_i)w_{i+1}$ is the $1 \rightarrow 0$ transition probability from bit $i+1$ to bit i required by the weight set. The value of $\pi_1 p_{10}$ that minimizes this error function is equal to:

$$\frac{\sum_{i=m-1}^{1} (1.0 - w_i)w_{i+1}}{m-1} \tag{12}$$

Then, the state transition probability p_{10} is calculated as:

$$p_{10} = \frac{\pi_1 p_{10}}{\pi_1} \tag{13}$$

p_{01} is calculated similarly. Specifying p_{01} and p_{10} completely specifies the 2-state Markov chain, and hence the PR sequence generated by the FSM. The FSM can be realized using a T flip-flop and a 2x1 multiplexer as shown in Figure 5. The bit streams with probabilities p_{01} and p_{10} produced by the PR source do not have to be independent of each other since only one of them will be multiplexed to the input of the T flip flop at any given time.

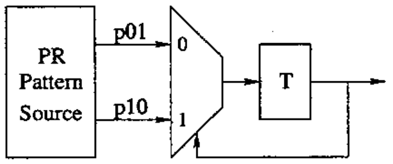

Figure 5. Realization of 2-state FSM

3.3 4-state FSM

It may be advantageous to use a 4-state machine in order to be able to capture the correlation between two consecutive bits in the deterministic test set. Figure 6 shows a 4-state Markov chain model we have used to generate PR bit sequences. The states give the last 2 bits generated by the FSM. State probabilities π_0, π_1, π_2, and π_3 correspond to 00, 01, 10 and 11 sequence probabilities within the generated bit sequence (P_0, P_1, P_2, and P_3). As seen from the figure it is not possible to have transitions between all of the states. Every state has exactly 2 incoming and 2 outgoing transitions. Since $p_{00} = 1.0 - p_{02}$, $p_{10} = 1.0 - p_{12}$, $p_{21} = 1.0 - p_{23}$, $p_{31} = 1.0 - p_{33}$, the output sequence can be entirely characterized by specifying p_{02}, p_{12}, p_{23}, and p_{33}. These are the transitions when the next generated bit is a 1. Now, we need to determine the state probabilities in the chain. The required values of the state probabilities are determined similar to the case of a 2-state FSM, but extending the calculation to 2

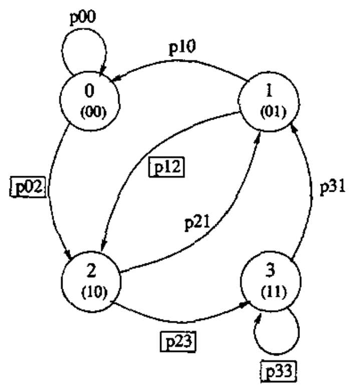

Figure 6. 4-state Markov Chain Model

consecutive bits. For example, the required probability P_3 of the output sequence is found by selecting π_3 such that it minimizes the following error function:

$$\sum_{i=m-1}^{1} (w_i w_{i+1} - \pi_3)^2 \tag{14}$$

where $\{w_1, w_2, ..., w_m\}$ is the calculated weight set. Similarly, P_2 and P_1 of the output sequence are determined by minimizing:

$$\sum_{i=m-1}^{1} (w_i(1.0 - w_{i+1}) - \pi_2)^2 \tag{15}$$

$$\sum_{i=m-1}^{1} ((1.0 - w_i)w_{i+1} - \pi_1)^2 \tag{16}$$

Then, $\pi_0 = 1.0 - \pi_1 - \pi_2 - \pi_3$ and $P_0 = 1.0 - P_1 - P_2 - P_3$.

The transition probability $\pi_0 p_{02}$ of the output sequence is determined by selecting its value so as to minimize the following error function:

$$\sum_{i=m-2}^{1} (w_i(1.0 - w_{i+1})(1.0 - w_{i+2}) - \pi_0 p_{02})^2 \tag{17}$$

$w_i(1.0 - w_{i+1})(1.0 - w_{i+2})$ is the $00 \rightarrow 10$ transition probability required by the weight set. Then, the state transition probability p_{02} is calculated as:

$$p_{02} = \frac{\pi_0 p_{02}}{\pi_0} \tag{18}$$

p_{12}, p_{23} and p_{33} are calculated similarly. Specifying p_{02}, p_{12}, p_{23} and p_{33} completely specifies the 4-state Markov chain, and hence the PR sequence generated by the FSM. The FSM can be realized using a two D flip-flops and a 4x2 multiplexer as shown in Figure 7. The bit streams with probabilities p_{02}, p_{12}, p_{23} and p_{33} produced by the PR source do not have to be independent of each other since only one of them will be multiplexed to the input of the D flip flop at any given time.

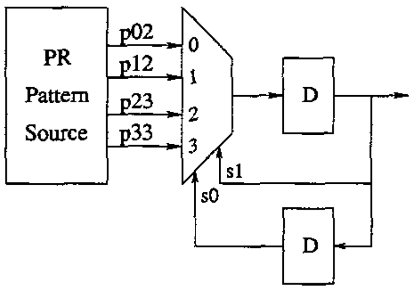

Figure 7. Realization of 4-state FSM

3.4 Calculation of weights

The weight set used in the above calculations is obtained with an ATPG based input weight calculation algorithm. The algorithm is the same as described in [3] with some minor changes.

First of all, we do not use an initial pseudo-random test generation phase. Instead, we initially generate a complete test set for all of the detectable faults in the circuit. For test generation, an ATPG tool which is capable of generating test cubes with unspecified values is used. The random pattern test generation phase of the test generator is turned off since such tests have unnecessary bit assignments and such assignments incorrectly bias the weight generation process. No other special ATPG capability is required for the test generation process.

In [3], once the probabilities p_{01} and p_{10} (or p_{02}, p_{12}, p_{23} and p_{33}) are calculated, the exact values are quantized to values that are easily generated within a BIST environment. In our calculations we use exact calculated weights instead, since the implementation does not require the generation of individual input weights.

In [3], the weight calculation algorithm processes the ATPG generated test set for each bit position. A test can have 0, 1, or an unspecified value in a bit position. If a bit position does not have any specified value in any test, then the weight of that bit is assumed to be 0.5. In contrast, we keep weight values unspecified for bit positions that have no specified value in any test. When calculating Equations 9 through 17, the contribution of the i^{th} term to the summation is omitted if none of the weights in the i^{th} term has a specified value. If at least one of the weights has a specified value in the i^{th} term, then the unspecified weight values are assumed to be 0.5.

Once the weights are calculated, they are quantized to values that are easily generated in a BIST environment.

3.5 FSM output inversion

The proposed FSM generates a bit sequence that averages out the weight distribution at the circuit inputs. However, for some circuit inputs, the weight of the generated sequence and the weight required at the circuit input can be considerably different. For example, the weight requirement of input i can be 0.2, although the probability of a 1 in the generated sequence is 0.7. Such a situation can seriously reduce the effectiveness of the generated pseudo-random sequence. We note that in this example, if the output of the FSM is inverted for bit position i, then the PRP weight at this input will be 0.3 which is much closer to the desired value of 0.2.

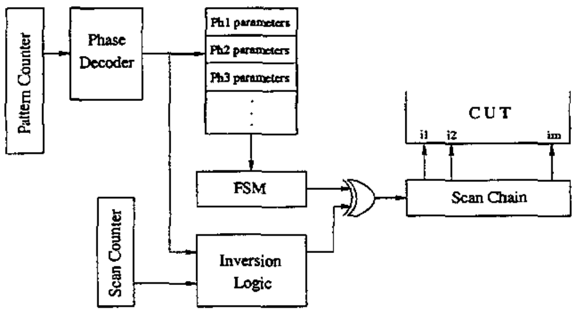

Figure 8. FSM output inversion

To be able to do such corrections, we add an XOR gate at the output of the PRPG and some additional combinational logic, which determines the bit positions to be inverted. This is shown in Figure 8.

To keep the size of the inversion logic reasonable, we require a significant change in the signal probability at bit position i as a result of inversion. If the signal probability of the sequence generated by the FSM is S and the signal probability of the inverted output is S' (which is $1.0 - S$), then the inversion logic is activated only if $|S - S|' > DeltaTh$, where $DeltaTh$ is a user specified threshold. If the weight value at a certain bit position is unspecified, the output of the inversion function is set to don't care.

To be able to carry out these calculations, we have to recalculate the steady state signal probability of the output bit stream since the transition probabilities are quantized after calculation. For a 2-state machine the following set of equations are used to calculate the signal probability of the output bit stream:

$$\frac{P_0}{P_1} = \frac{p_{10}}{p_{01}} \tag{19}$$

$$P_0 + P_1 = 1.0 \tag{20}$$

The output signal probability S is equal to P_1. For a 4-state machine, the following equations can be derived for a Markov chain at the steady state:

$$\frac{P_0}{P_1} = \frac{1.0 - p_{12}}{p_{02}},\ \frac{P_1}{P_3} = \frac{1.0 - p_{33}}{p_{23}},\ P_1 = P_2 \tag{21}$$

$$P_0 + P_1 + P_2 + P_3 = 1.0 \tag{22}$$

and the output signal probability is calculated as:

$$S = P_0p_{02} + P_1p_{12} + P_2p_{23} + P_3p_{33} \tag{23}$$

3.6 Virtual multiple scan chains

One consideration in the proposed scheme is that, if the scan chain driven by the PRPG contains a large number of scan elements, it becomes harder to satisfy weight requirements at every bit position. One solution to overcome this difficulty is to use multiple scan chains and a separate PRPG for each chain.

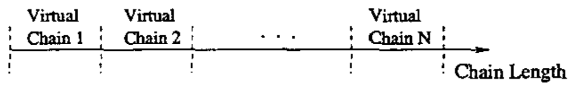

Figure 9. Virtual multiple chains

It is also possible to divide a single chain into multiple *virtual chains*, as shown in Figure 9. The term "virtual scan chain" was previously used in [19], where a scan chain is physically divided into multiple parallel chains to reduce test application time, and overall chain structure is referred to as virtual chain. Here, we divide a chain into multiple chains only conceptually. The test application time in our case will remain unchanged.

For each virtual chain, individual calculations are carried out as described previously. During scan shifts, proper transition probability sets are loaded for each chain through a virtual chain decoder which monitors the scan counter and identifies individual virtual chains. The block diagram of the scheme that uses multiple virtual chains is given in Figure 10.

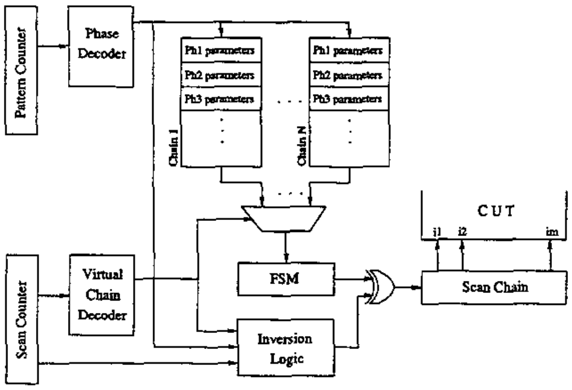

Figure 10. BIST scheme that uses virtual multiple chains

4 Experimental results

We conducted experiments on the larger ISCAS89 circuits using a Linux machine with 1400 MHz Intel Pentium IV processor and 512 MB memory. We assumed a single scan chain in all the circuits. However, we divided the single chain into multiple virtual chains of fixed length. A virtual chain length of 48 was used for s9234, s13207 and s15850. For s38417 and s38584, the virtual chain length is 64. If the virtual chain length is l_{vc}, then all the virtual chains in the circuit contain this many scan elements except possibly the last chain. The last chain contains $n_{se} - l_{vc} * \lfloor n_{se}/l_{vc} \rfloor$ elements where n_{se} is the total number of scan elements.

During the experiments, we used the following stopping criterion for each iteration. If the last 2048 patterns did not detect any new faults then the application of pseudo-random patterns is ended. A new phase is started from the point where the last fault is detected in the previous phase.

We used 5 quantization levels for the transition probabilities that determine the FSM behavior. The levels we used are 0.125, 0.25, 0.5, 0.75 and 0.875. However, not all of these levels are allowed in all the phases. For a 2-state FSM, all 5 levels are allowed only in the first phase. After the first phase, only two quantization levels are allowed, 0.25 and 0.75 or 0.125 and 0.875. The reason for restricting the transition probabilities is the following: After the first phase, an already high fault coverage is reached. Usually faults that require longer strings of 0s and 1s tend to remain undetected after the first phase. Therefore, PR sequences with small $0 \rightarrow 1$ or $1 \rightarrow 0$ transition probabilities are required in later phases. However, if a transition probability is quantized to 0.5, it may disturb the generation of such low transition probability sequences and increase the number of required tests. Therefore, we do not allow 0.5 transition probability after the first phase. We use levels 0.25 and 0.75 and bias the output signal probability toward 0 or 1, whichever is required more in the tests. Iteration continues with these quantization levels until no more faults are detected. This allows elimination of faults that require longer strings of 0s and 1s. Then the iteration continues with levels 0.125 and 0.875 to detect faults that require even longer strings of 0s and 1s. In our experiments, none of the circuits required quantization levels beyond 0.125 and 0.875 to achieve full coverage.

For a 4-state FSM, we use all 5 quantization levels for the first two phases. Here we allow one more phase with 5 quantization levels because of the 4-state FSM's higher ability to capture correlations between consecutive bits. We use levels 0.125 and 0.875 for all other phases and bias the output signal probability toward 0 or 1, whichever is required more in the tests.

Results are given in Table 1. The first two columns in the table list the circuit names and the number of virtual chains used in each circuit. Columns 3 and 4 list the test lengths to achieve 100% fault coverage by the proposed method using 2-state and 4-state FSM's respectively. We compare our results with the work in [17] that modifies pseudo-random patterns at the input of a scan chain. The test lengths to achieve 100% fault coverage using the method of [17] are listed in column 5. Column RPL lists the necessary test lengths to test the circuits using pseudo-random patterns with a uniform bit probability of 0.5. The test lengths listed in Table 1 are the number of test vectors applied to the circuit under test.

The test lengths to achieve 100% coverage in our method are reduced by orders of magnitude compared to the pseudo-random patterns with uniform probability. Our test lengths are also shorter in most cases than those obtained in [17]. For example, for circuits s15850 and s13207 we achieve 62% and 57% reductuions in

Table 1. Experimental results

Circuit Info.		Test Length				Gate Equivalent			Pct. Cost		#Phases		Run Time	
Name	VC	2St	4St	[17]	RPL	2St	4St	[17]	2St	4St	2St	4St	2St	4St
s9234	6	44.8K	37.1K	44.0K	11M	290.0	227.0	611.0	4.17	3.27	6	4	120s	87s
s13207	15	24.8K	27.0K	71.6K	264K	394.5	358.5	380.5	2.82	2.56	4	3	108s	113s
s15850	13	45.5K	37.3K	87.0K	>100M	410.5	488.0	631.5	2.90	3.45	4	4	226s	170s
s38417	26	87.1K	76.6K	86.0K	>100M	1130.0	1080.0	1685.5	3.14	3.00	6	6	1164s	1084s
s38584	23	49.7K	47.0K	49.1K	>100M	410.0	272.0	584.5	1.20	0.80	6	3	731s	656s

test length using a 4-state FSM.

If we compare the 2-state FSM design and 4-state FSM designs, we observe that the 4-state FSM design can capture the bit correlations in the weight sets better than the 2-state design, and hence it produces better PR sequences that result in shorter test lengths for all the circuits except s13207.

In our method, the overall test length can be reduced even further by reducing the stopping criterion, for example from 2048 to 1024. By doing so, the portions of the PR sequence that sparsely detect new faults will be eliminated. However, this in turn will increase the number of required phases and potentially the size of the necessary logic.

We synthesized the logic required to implement the discussed BIST architecture. We used espresso for 2-level minimization of the logic functions. Then we calculated the gate equivalent numbers of the logic realization as used in [17, 15], that is, $0.5n$ for n input NAND and NOR gates, 0.5 for inverters and $2.5(n-1)$ for an n input XOR gate. The total cost of the implementation in terms of gate equivalents are given in columns 7 and 8 of Table 1 when we use 2-state and 4-state FSMs, respectively. The given numbers include the logic necessary to implement the inversion logic, the phase decoder, the virtual chain decoder and the circuit necessary to load the FSM parameters. Gate equivalent costs reported in [17] are given in column 9. The work in [17] describes two methods for altering pseudo-random bit sequences: The bit sequence can be altered serially at the input of a scan chain or it can be altered within the scan chain through parallel access to individual scan elements. Scan chain reordering to reduce the necessary additional hardware is also discussed in [17]. Since our work describes a method only for altering a pseudo-random bit sequence at the input of a given scan chain, and does not consider scan chain reordering, we only list data from [17] that is comparable to our work. In columns 10 and 11, we also report the implementation cost of 2-state and 4-state FSMs as the percentage of total gate equivalent cost of the whole circuit. In these calculations, we have assumed flip-flops are master-slave D-flip flops composed of 8 NAND gates and an inverter.

As seen from Table 1, our hardware implementation costs are smaller than those reported in [17]. For example, the reduction is 63% for circuit s9234 and 35% for circuit s15850 when 2-state FSMs are used. For the largest circuits, the reductions are also significant. We achieve 36% reduction for s38417 and 53% reduction for s38584 by using 4-state FSM designs. Our hardware cost is higher only for s13207 when we use a 2-state FSM. In this case the hardware cost increase is about 4% compared to the cost in [17]. The results indicate that absolute value of the area cost increases with increasing circuit size, however as seen from columns 10 and 11, percentage cost of implementation in fact do not increase with increasing circuit size. It is as low as 0.8% for s38584 when 4-state FSM is used.

If we compare the 2-state and 4-state FSMs we see that the necessary hardware is smaller in 4 out of 6 circuits for the 4-state machine. The reason for this is that the number of necessary iterations, i.e., the number of phases, is reduced when a 4-state machine is used, which results in a simpler hardware. The number of phases for both 2-state and 4-state machines are reported in columns 12 and 13 of Table 1. The number of phases for a 4-state FSM is always smaller than or equal to the number of phases for a 2-state machine.

Finally we report the run times in the last two columns of Table 1 for both 2-state and 4-state designs. The run times include the times for deterministic test generation, weight set calculation and fault simulation. Time spent during logic synthesis using espresso is not included. Since the number of phases and the total test lengths are reduced for a 4-state machine, the total run times are also typically less for the 4-state machine compared to those for the 2-state machine.

5 Conclusion

In this work, we described a new pseudo-random pattern generation technique for full scan circuits. The test pattern generator generates bit sequences to satisfy input weight requirements as determined by a weighted random pattern testing technique. The calculation of input weight assignments is based on the analysis of a deterministic test set and requires no special ATPG except the ability to produce test cubes with unspecified bit positions.

The core of our PRPG design is based on a Markov source, which results in a temporal dependence in the produced bit stream. Using this temporal dependence, it is possible to create the spatial correlation corresponding to the desired weight requirements inside a scan chain. PRPG's ability of capturing the spatial correlation is increased as the number of states in the Markov chain is increased, which is demonstrated by the experiments with 2-state and 4-state FSM designs. The implementation of the proposed PRGP source requires very little additional hardware even for the large benchmark circuits.

References

[1] F. Muradali, V.K. Agarwal, and B. Nadeu-Dostie, "A New Procedure for Weighted Random Built-In Self-Test", *Proc. of Int'l Test Conference*, pp. 660-669, 1990.

[2] H.J. Wunderlich, "Multiple Distributions for Biased Random Test Patterns", *Proc. of Int'l Test Conference*, pp. 236-244, 1988.

[3] H.S. Kim, J. Lee, and S. Kang, "A New Multiple Weight Set Calculation Algorithm", *Proc. of Int'l Test Conference*, pp. 878-884, 2001.

[4] S. Pateras, and J. Rajski, "Cube-Contained Random Patterns and Their Application to the Complete Testing of Synthesized Multi-Level Circuits", *Proc. of Int'l Test Conference*, pp. 473-482, 1991.

[5] I. Pomeranz, and S.M. Reddy "3-Weight Pseudo-Random Test Generation Based on a Deterministic Test Set for Combinational and Sequential Circuits", *IEEE Trans. on Computer-Aided Design of Circuits and Systems*, Vol. 12, No. 7, pp. 1050-1058, July 1993.

[6] B. Koenemann, "LFSR-Coded Test Patterns for Scan Designs", *European Design and Test Conference*, pp. 237-242, 1992.

[7] S. Hellebrand, S. Tarnick, J. Rajski, and B. Courtois, "Generation of Vector Patterns Through Reseeding of Multiple-polynomial Linear Feedback Shift Registers", *Proc. of Int'l Test Conference*, pp. 120-129, 1992.

[8] S. Venkataraman, J. Rajski, S.Hellebrand, and S. Tarnick, "An Efficient BIST Scheme Based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers", *IEEE Int'l Conf. on Computer Aided Design*, pp. 572-577, 1993.

[9] B. Seiss, P.M. Trouborst, and M.H. Schulz, "Test point Insertion for Scan-Based BIST", *Proc. of European Test Conference*, pp. 253-262, April 1991.

[10] N. Tamarapalli, and J. Rajski, "Constructive Multi-Phase Test Point Insertion for Scan-Based BIST", *Proc. of Int'l Test Conference*, pp. 649-658, 1996.

[11] J.A. Waicukauski, and F. Motika, "Testing VLSI Chips with Weighted Random Patterns", *Proc. of Symp. on VLSI Technology, Systems and Applications*, pp. 149-154, 1989.

[12] B. Reeb, and H.J. Wunderlich, "Deterministic Pattern Generation for Weighted Random Pattern Testing", *Proc. of European Design and Test Conf.*, pp. 1023-1030, 1996.

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

[14] F. Brglez, C. Gloster, and G. Kedem, "Built-In Self Test with Weighted Random Pattern Hardware", *Proc. of Int'l Conf. on Computer Design*, pp. 161-166, 1990.

[15] J. Hartmann, and G. Kemnitz, "How to do weighted random testing for BIST", *Proc. of Int'l Conf. on Computed Aided Design*, pp. 568-571, 1993

[16] N.A. Touba, and E.J. McCluskey, "Altering a Pseudo-random Bit Sequence for Scan Based BIST", *Proc. of Int'l Test Conference*, pp. 649-658, 1996.

[17] S. Wang, "Low Hardware Overhead Scan Based 3-Weight Weighted Random BIST", *Proc. of Int'l Test Conference*, pp. 868-877, 2001.

[18] L. Brehelin et al., "Hidden Markov and Independence Models with Patterns for sequential BIST", *Proc. of VLSI Test Symposium*, pp. 359-367, 2000.

[19] A. Jas, B. Pouya and N.A. Touba, "Virtual Scan Chains: A Means for reducing Scan Length in Cores", *Proc. of VLSI Test Symposium*, pp. 73-78, 2000.

[20] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits", *Proc. of Int'l Symp. on Circuits and Systems*, pp. 1929-1934, 1989.

[21] A.O. Allen, "Probability, Statistics, and Queueing Theory with Computer Science Applications", *Academic Press*, New York, San Francisco, London, 1978.