A novel design with Cellular Automata for System-Under-Test in Distributed Computing

Arnab Mitra¹, Anirban Kundu², Matangini Chattopadhyay³, Samiran Chattopadhyay⁴

¹Department of CSE, Adamas Institute of Technology, Kolkata-700126, India, mitra.arnab@gmail.com
²Information Technology department, Netaji Subhash Engineering College, Kolkata-700152, India, anik76in@gmail.com
³School of Education Technology, Jadavpur University, Kolkata-700032, India, matanginic@gmail.com
⁴Department of Information Technology, Jadavpur University, Kolkata-700098, India, samiranc@it.just.ac.in

Abstract
Fault tolerant computing system is a vital criterion for reliable computation in distributed computing environment. This prerequisite has initiated system-under-test (SUT) as a testing approach to investigate the possible failure in any components of distributed computing. Validation of fault tolerant procedure is often performed by injecting faults. Fault injection causes for the actual faults in the system and thus is responsible for high stresses in the functionality of that system. An efficient methodology in cellular automata (CA) for validation of faults in SUT has been proposed in our paper for designing of fault tolerant reliable distributed computing. In this research, built-in self-test (BIST) has been considered for SUT in distributed environment targeting faster fault response. Generation of pseudo-random BIST pattern using Equal Length Cellular Automata (ELCA) for fault validation has been emphasized here. High degree of randomness, cost effectiveness in generated pseudo-random patterns and high fault coverage in circuit-under-test (CUT) have been reported for ELCA based SUT design. Enhanced efficiencies have been achieved in proposed ELCA based SUT in distributed computing.

Keywords: Distributed Computing, System-Under-Test (SUT), Fault Injection (FI), Built-In Self-Test (BIST), Prohibited Pattern Set (PPS), Pseudo-Random Number Generator (PRNG), Cellular Automata (CA), Equal Length CA (ELCA)

1. Introduction
Distributed computing [1-2] is a refined computation practice towards network based consistent computation over geographically secluded computing units targeting to work collectively. Distributed computing refers such a computing method [1-2] where a single task is decomposed into modules and all the modules are completed on several computing devices over an established network. All the partial solutions of modularized tasks are then summed up to form a single solution to get the concluding consequence. Hence it is of great concern that all the computing components work properly in that distributed computing system, i.e., fault tolerance is required in that particular distributed computing system. Therefore testing algorithms for such systems often requires fault injections (FI) and experiments to assess if the specified fault supposition holds for the accomplishment. Faults are considered as elements of the applicable input data of a fault-tolerant distributed computing system. FIs are categorized into Simulation based FI [3], hardware FI [4] and software FI [5].

Hardware FI is preferable as timing is alarmed with referenced to software FI. Hardware FI is primarily used on the chip-level. On the other hand fault injection is allowed in algorithmic level for software FI. Timing behavior of the complete system-under-test (SUT) is adversely affected by injection of faults. Therefore it is not suitable for typical distributed computing systems with real time constraints. In design-for-testability (DFT) technique for any circuit-under-test (CUT) for SUT in
distributed computing environment, Built-In Self-Test (BIST) is used for testing purpose. A section of 
CUT is verified and tested to understand the behavior of the total circuit design.

Cellular automaton (CA) [6] has been successfully used in BIST applications. A CA is a dynamic 
mathematical model to represent the dynamic behavior of any system. CA consists of a regular frame 
of cells. Each of the cells remains in a state having a value which is either in “On or high” or “Off or 
low”. For each cell surrounded with its neighborhood is defined with respect to the particular cell. Next 
generation is obtained with predetermined rule(s) in a timely manner. In a three-neighborhood 
dependency, the next state \( S_{i}^{t+1} \) of a cell of 1-dimensional CA is denoted by Equation 1[6].

\[
S_{i}^{t+1} = f_{i}(S_{i-1}^{t}, S_{i}^{t}, S_{i+1}^{t}) 
\]

where, \( f_{i} \) is the next state function; 
\( S_{i-1}^{t}, S_{i}^{t}, S_{i+1}^{t} \) are the present (local) states of the left neighbor, self and right neighbor of the \( i^{th} \) 
CA cell;

The collection of states \( S_{i}^{t}(S_{1}^{t}, S_{2}^{t}, \ldots, S_{n}^{t}) \) at time ‘t’ is the present state of CA having ‘n’ 
cells. If leftmost and rightmost cells of the CA are ground, the CA is called null boundary CA. We are 
only concerned about null boundary CA in this paper. Typical structure of a 3 cell Null Boundary CA 
is represented in Figure 1 [6].

![Figure 1. Typical structure of 3-cell Null Boundary CA](image)

Next state function of ith cell of CA is expressed as a truth table (refer Table 1). Decimal equivalent 
of the eight outputs is called ‘Rule’ \( R_i \) as described in [7]. In any 3-neighborhood CA (refer Figure 1), 
total \( 2^{26} \) (256) rules are possible in space. For example, rules ‘100’, ‘135’, and ‘247’ have been 
mentioned in Table 1. The first row of the table lists the possible \( 2^3 \) (8) combinations of the present 
states of ‘\( (i-1)^{th} \)’, ‘\( i^{th} \)’, and ‘\( (i+1)^{th} \)’ cells at time ‘t’.

<table>
<thead>
<tr>
<th>Present State (RMT)</th>
<th>111 (7)</th>
<th>110 (6)</th>
<th>101 (5)</th>
<th>100 (4)</th>
<th>011 (3)</th>
<th>010 (2)</th>
<th>001 (1)</th>
<th>000 (0)</th>
<th>Rule (Decimal Equivalent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) Next State</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>(i) Next State</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>135</td>
</tr>
<tr>
<td>(i) Next State</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>247</td>
</tr>
</tbody>
</table>

Few fundamental CA terminologies as discussed in [8] are reported as follows: 
The set of rules \( R = < R_1, R_2, \ldots, R_n > \) responsible for the configuration of the cells of a CA is called 
the rule vector. If same rule is followed in all the CA cells, then the CA is said to be a uniform CA; 
otherwise it is a non-uniform/hybrid CA. If the next-state logic is employed with only XOR/XNOR 
logic, then it is called a linear/additive rule. CA with a combination of XOR and XNOR rules is called 
additive CA (ACA) and associated rules are defined as additive rules. CA with a combination of 
additive rule(s) and non-linear rule(s) are referred as non-linear CA. 
Example 1: 
\( R = < 10, 90, 150, 255 > \) is a rule vector consisting of four rules.
Rest of the paper is organized as follows: related work is described Section 2; background is reported in Section 3; proposed work is discussed in Section 4; Experimental observations and result analysis are shown in Section 5 and Conclusion is reflected in Section 6.

2. Related Works

Past researches have been carried out on virtualization to validate fault tolerance in distributed system [9-23]. Fault management has been executed with fault injectors (FIs) in virtualized distributed system. Fault injections have been accomplished to validate system operation under various well-defined fault models. Usage of fault injections have been previously suggested to test distributed systems [9-23]. An infrastructure has been presented by I. Hsu et al. [9] to support the analysis of the behavior of distributed SUT. Software-implemented FI and virtualization have been combined in the proposed design [9] for an automated validation and analysis of distributed SUT. High precision has been reported for the validation of a Byzantine-fault-tolerant cluster manager [9]. Hybrid fault injections for distributed SUT have been introduced by C. Trödhandl et al. [10]. As a part of valid input data of a fault-tolerant system, faults [10] are injected to assess the specified fault hypothesis for SUT. A conceptual FI framework based on hybrid hardware-software method of fault injection for distributed system has been presented in [10].

Efficiencies for FI usage in SUT have been examined by D. T. Scott et al. [11], M. Cukier et al. [12] and S. Han et al. [23]. Besides, K. Buchacker et al. [13-14], V. Sieh et al. [15], S. Potyra et al. [16], T. Herault [17] and M. Le et al. [18] have presented the leveraging virtualization for FI. Software-implemented fault injection (SWIFI) has been practiced S. Han et al. [15] in designing of fault tolerant distributed system. J. Arlat et al. have concluded in their research that designing of fault tolerance is the most crucial component in SUT [19]. Fault injection in Loki architecture introduced by R. Chandra et al. has focused concerned fault injections for global fault model which is dependent on internal states of all components (nodes) of the tested distributed system [20].

Validation of fault tolerant distributed systems using fault injection is still tough since erroneous symptoms are observed as soon as a rare ordering of asynchronous events are found in that specific distributed system. Thus, a flexible infrastructure is required for the execution of injected faults. Three key requirements specified by I. Hsu et al. [9] for infrastructures as follows:

i) To permit boundary conditions across multiple system components (nodes) at randomized test values;

ii) To ensure that the initial system state for each test must be at “error free” state;

iii) Availability of log for off-line analysis of fault injections, system responses, and resource usage;

ELCA based infrastructure for SUT has been projected in this paper. Detailed design has been described in Section 4. Remarkable cost optimized performance in BIST applications [26] have initiated the potential usage of ELCA based SUT design. Significant reduction for requirement of the hardware resources in SUT has been established for proposed design. A detailed experimental section using cutting edge benchmark circuits (components) using proposed infrastructure has been included in Section 5 to demonstrate the critical significance, flexibility and autonomous operation provided by proposed ELCA based infrastructure.

Past researches with ELCA have reported that it is possible to produce all the states of an n-cell CA using some equal length cycles [24-26]. It is further illustrated as below.
In a scenario of an n-cell CA, the total number of states of that n-cell CA is \(2^n\), are decomposed into two ELCA of cycle length \(2^{n-1}\). This methodology is described in Equation 2, Equation 3 and Equation 4 [24-25].

\[
2^n = 2^1 \times (2^{n-1})
\]  
\[
2^n = 2^2 \times (2^{n-2})
\]  
\[
2^n = 2^m \times (2^{n-m}) \quad \text{for } n \geq 1 \text{ and } m=1,2,3\ldots(n-1)
\]

Two ELCA of length \((n-1)\) have been described in Equation 1. ELCA are further decomposed into four smaller cycles of length \((n-2)\) as shown in Equation 2. Mathematical formulation for generation of ELCA has been described in Equation 3; where, ‘m’ is always less than ‘n’.

Generated ELCA cycles have been reported as a source of pseudo-random sequences [24-26]. High degree of randomness in generated pattern and complete fault coverage for BIST applications has been reported for ELCA PRNG as compared to Linear Feedback Shift Registers (LFSR) and maximum length CA (MaxCA) PRNG [26]. PPS has been completely removed from random pattern generating cycles [24-26]. Therefore an efficient method for SUT in distributed computing environment using ELCA should facilitate to reduce fault overhead in cost optimized way.

3. Background

Past researches with ELCA have emphasized that randomness is found from patterns generated by ELCA [24-26]. Degree of randomness for ELCA based PRNG is shown in Figure 2.

![Degree of Randomness](image)

Figure 2. Degree of randomness in generated pattern for <153, 153, 153, 153, 153>

Degree of randomness found in Diehard Test Suit for ELCA based PRNG has been reported in Table 2 [24-26]. Degree of randomness, as reported in Table 2, is based on the total number of tests passed in Diehard Test Suit. Higher degree of randomness has been reported for ELCA patterns with increased CA size.

<table>
<thead>
<tr>
<th>Diehard Tests</th>
<th>ELCA n=23</th>
<th>ELCA n=64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Number of Test Passes</td>
<td>10</td>
<td>14</td>
</tr>
</tbody>
</table>
Diehard test results as reported in Table 1 have been further graphically discussed in Figure 3. Enhanced degree of randomness is shown in case of higher CA size.

![Degree of randomness in Diehard Tests](image)

**Figure 3.** Randomness Quality for ELCA PRNG in Diehard tests

Different complexities for concerned pseudo-random pattern generators have been enlisted in Table 2. Let us consider that space required to generate an n-cell CA is equal to ‘2n’. Therefore the space consumed for ELCA, is ‘O (n)’. Time required for m-length ELCA as illustrated in Equation 3, is ‘O (m)’. We get, ‘m’ is smaller than ‘n’ by Equation 3. Major improvements have been achieved in case of design and searching complexities for ELCA based design [24-26]. ELCA based PRNG is allowed to generate random patterns from smaller cycles that exclude prohibited pattern set (PPS). Cost effectiveness for ELCA based design has been reported in Table 3 [24-26].

**Table 3.** Cost Complexities for ELCA PRNG

<table>
<thead>
<tr>
<th>Name of Complexity</th>
<th>Measurement</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>O(n)</td>
<td>It depends on number CA cells</td>
</tr>
<tr>
<td>Time</td>
<td>O(m)</td>
<td>It depends on length of equal length cycle</td>
</tr>
<tr>
<td>Design</td>
<td>-</td>
<td>Completely removes presence of PPS</td>
</tr>
<tr>
<td>Searching</td>
<td>-</td>
<td>Improved as less overhead for presence of PPS</td>
</tr>
</tbody>
</table>

As referred in Table 3, complexities have further illustrated as follows:

Requisite number of ‘D’ flip-flops for physical realization of concerned ELCA system has been referred to as space complexity. Time required for generation of one complete equal length cycle by ELCA has been referred to as time complexity. It has been discussed in Equation 4 that length of equal length cycle is always less than the length of maximum length cycle. Therefore, O(m) within Table 3 should be always smaller value [24-26]. Inherent design methodology for dealing with problems of
PPS has been considered as design complexity. The complexity related to the searching for PPS free patterns is referred to as searching complexity.

Efficiency of ELCA based pattern generator for BIST application has been reported [26]. Fault coverage for “ISCAS 85” and “ISCAS 89” benchmark circuits have been reported in Table 4 [26]. ELCA PRNG based BIST test patterns have been used for a simulation environment in BISTAD simulation software [27-28].

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>Coverage under LFSR</th>
<th>Coverage under MaxCA</th>
<th>Coverage under ELCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>s386</td>
<td>93.17%</td>
<td>96.67%</td>
<td>100.00%</td>
</tr>
<tr>
<td>s298</td>
<td>93.30%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>s1488</td>
<td>98.92%</td>
<td>97.72%</td>
<td>98.68%</td>
</tr>
<tr>
<td>s1494</td>
<td>98.11%</td>
<td>96.87%</td>
<td>97.87%</td>
</tr>
<tr>
<td>s208_1</td>
<td>95.18%</td>
<td>96.93%</td>
<td>99.56%</td>
</tr>
<tr>
<td>s27</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>cl7</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

It is found from Table 4 that ELCA based BIST applications have produced competitive results for fault coverage in specified benchmark circuits compared to Linear Feedback Shift Register (LFSR) and maximum length CA (MaxCA) based BIST applications.

4. Proposed Methodology

We have emphasized on the efficient usage of ELCA for testing of hardware components over a distributed computing environment in system-under-test (SUT). Faults injected by FI in a target component of a SUT are tested with ELCA based pseudo-random test patterns. Cost effective pseudo-random pattern generation capability for generated equal length cycles by ELCA has been efficiently used in SUT applications. We have proposed ELCA based design of SUT architecture as described in Figure 4.
Proposed SUT architecture has been incorporated over existing Client-Server based FI hardware [3], which is represented in Figure 5[3].

**Figure 4.** Proposed SUT architecture in distributed computing environment

**Figure 5.** Proposed Fault Injection Client-Server Architecture [3]
Flowchart of proposed hardware testing for target components in SUT has been described in Figure 6.

Figure 6. Proposed Flowchart of SUT using ELCA BIST

We have concentrated on PPS free pseudo-random pattern generation for producing test cases in BIST applications. These BIST applications are modular parts of component testing for SUT in a distributed computing environment. Algorithm 1 and Algorithm 2 have been used to produce PPS free pseudo-random test pattern for BIST applications and performing CUT for SUT.

Algorithm 1. PPS\textunderscore free\textunderscore pseudo\textunderscore random\textunderscore BIST pattern generation

\textbf{Input:} CA size (n), PPS Set, Balanced rule(s)
\textbf{Output:} PPS free pseudo-random BIST patterns
\begin{enumerate}[Step 1: Start]
  \item Initialize the number of n-cell CA and balanced rules to generate random patterns
  \item Decompose the cell number (n) into equal numbers (m) such that $n=2^m$
  \item Check each PPS whether it belongs to a single smaller cycle CA
  \item Repeat Step 3 and Step 4 until each PPS are separated in some smaller cycles
  \item Remove all the PPS containing smaller cycles
  \item Allow remaining m-length equal length cycles to produce pseudo-random patterns for BIST
  \item Stop
\end{enumerate}
Algorithm 2. ELCA_based_Built-In Self-Testing_for_System-Under-Testing

Input: SUT components
Output: Checked error free system components for reliable computation

Step 1: Start
Step 2: Initialize the circuit-under-test (CUT) area to be tested by server
Step 3: Inject faults by Fault Injector (FI) in target circuit of the client side
Step 4: Initialize the Built-in self-test (BIST) at client side
Step 5: Generate PPS free pseudo-random test patterns for BIST using Algorithm 1
Step 6: Perform testing on target circuit
Step 7: Analyze the signature generated in BIST
Step 8: Report circuit status for fault inspection at server side
Step 9: Report SUT tester about component status for processing of reliable computation
Step 10: Stop

Example 2:
PPS free equal length cycle generation has been illustrated in Figure 7. Synthesis of Example 2 has been achieved for null boundary CA with <51, 51, 195, 204> for an arbitrary PPS as {0, 2, 4}. Prohibited pattern in equal length cycle is indicated by dark color state.

![Equal length cycle](Image)

Figure 7. Equal length cycle structure with presence of PPS for <51, 51, 195, 204>

Generation of four equal length cycles has been illustrated in Figure 7. Presences of all prohibited patterns have been found in two generated equal length cycles. Therefore these two equal length cycles containing prohibited patterns have not been considered for pseudo-random pattern generation.

5. Experimental Observations & Result Analysis

Available functional fault models for BIST [29] are reported as follows:
i) Stuck at faults (SAF): A cell / line is stuck to logical zero/one state. SAF is thus categorized into Stuck at 1 and Stuck at 0 fault.

ii) Transition faults (TF): It is not possible to retain previous state value since the memory value has been changed once. It is similar to SAF.

iii) Coupling faults (CF): Coupling between two adjacent cells is focused with CF during a transition from “zero to one” or, “one to zero”. Neighboring cell is forced to change its value during transition of target cell.

iv) Neighborhood_pattern_sensitive faults (NPSF): Center cell of a nine neighborhood configuration is bound to change its value influenced by its neighborhood.

v) Data_retention_faults (DF): It is not possible by memory cell to retain its value over time after a memory write/read operation.

vi) Address_decoder_faults (AF): No cell or, multiple cells are accessed simultaneously with an address or, a single cell is accessed by multiple addresses.

We have considered SAF testing in BISTAD simulation setup. SAF coverage is measured by projecting each pin of the hardware model at logic ‘0’ and logic ‘1’, respectively and executing the test vectors. If at least one of the outputs differs from expected result, the fault is considered to be sensed. Hence SAF testing for hardware components in distributed system has been considered for the design of system under test [30]. BISTAD Simulation details for “t3.agm” benchmark circuit have been further illustrated in Figure 8, Figure 9, Figure 10 and Figure 11.

Circuit design for “t3.agm” benchmark circuit has been described in Figure 8. Five input lines (X1, X2, X3, X4, X5) and two output lines (Y1, Y2) are shown in circuit diagram of “t3.agm” combinational circuit.

![Figure 8. Screen shot for “t3.agm” benchmark circuit](image)

Simulation details for “t3.agm” benchmark circuit for fault coverage using ELCA PRNG has been reported in Figure 9, Figure 10 and Figure 11. Fault Table used for “t3.agm” circuit testing in BISTAD simulation has been reported in Figure 9. ‘X’ has been used to represent “don’t care” condition in Figure 9.
A novel design with Cellular Automata for System-Under-Test in Distributed Computing
Arnab Mitra, Anirban Kundu, Matangini Chattopadhyay, Samiran Chattopadhyay

Figure 9. Fault table for “t3.agm”

Figure 10. Screen shot of SAF test coverage graph for “t3.agm”

Figure 11. Screen shot of fault coverage for “t3.agm”
Stuck-at-faults (SAF) coverage for “t3.agm” has been reported in Figure 10 and fault coverage of “t3.agm” has been described in Figure 11. 100% SAF coverage graph has been shown for “t3.agm” in Figure 10 and 100% Fault Coverage has been discussed for “t3.agm” in Figure 11.

Fault coverage in BISTAD for different independent systems has been reported in Table 5. ELCA generated random test patterns have been applied to “t.agm” combinational circuits; each circuit has five inputs and two outputs behaving as independent system. High accuracy for fault coverage has been achieved in test results for reported “t.agm” benchmark circuits. Fault coverage has been described in Equation 5 [31].

\[
\text{fault coverage} = \frac{\text{no of faults noticed}}{\text{total no of faults}}
\] (5)

Table 5. Fault coverage table for ‘t’ benchmark circuits

<table>
<thead>
<tr>
<th>Serial No.</th>
<th>Model No.</th>
<th>Fault coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>t1.agm</td>
<td>100%</td>
</tr>
<tr>
<td>2</td>
<td>t2.agm</td>
<td>100%</td>
</tr>
<tr>
<td>3</td>
<td>t3.agm</td>
<td>100%</td>
</tr>
<tr>
<td>4</td>
<td>t4.agm</td>
<td>100%</td>
</tr>
<tr>
<td>5</td>
<td>t5.agm</td>
<td>100%</td>
</tr>
<tr>
<td>6</td>
<td>t6.agm</td>
<td>100%</td>
</tr>
<tr>
<td>7</td>
<td>t7.agm</td>
<td>100%</td>
</tr>
<tr>
<td>8</td>
<td>t8.agm</td>
<td>100%</td>
</tr>
<tr>
<td>9</td>
<td>t9.agm</td>
<td>100%</td>
</tr>
<tr>
<td>10</td>
<td>t10.agm</td>
<td>100%</td>
</tr>
</tbody>
</table>

Hundred percent fault coverage for different “t.agm” benchmark circuits has been reported in Table 5.

6. Conclusion

It is found from experimental observations that ELCA PRNG has a high degree of randomness in generated pseudo-random pattern. Furthermore complete exclusion of PPS form pseudo-random sequence generating ELCA cycles is possible in cost effective approach. Efficiency of ELCA based PRNG in BIST applications have been reported for “ISCAS 85”, “ISCAS 89” and “t.agm” circuits. Efficiency of ELCA based SUT architecture has been established. Therefore, ELCA based PRNG is an efficient choice as a random test pattern generator for SUT architecture in distributed computing environment.

7. References

A novel design with Cellular Automata for System-Under-Test in Distributed Computing
Arnab Mitra, Anirban Kundu, Matangini Chattopadhyay, Samiran Chattopadhyay


A novel design with Cellular Automata for System-Under-Test in Distributed Computing
Arnab Mitra, Anirban Kundu, Matangini Chattopadhyay, Samiran Chattopadhyay