X-TOLERANT COMPRESSION AND APPLICATION OF SCAN-ATPG PATTERNS IN A BIST ARCHITECTURE

Peter Wohl
Synopsys Inc.
Williston VT 05495
wohl@synopsys.com

John A. Waicukauski
Synopsys Inc.
Tualatin OR 97062
johnwaic@synopsys.com

Sanjay Patel
Synopsys Inc.
Beaverton OR 97006
sanjayp@synopsys.com

Minesh B. Amin
Synopsys Inc.
Mountain View CA 94043
mamin@synopsys.com

Abstract

We present X-tolerant deterministic BIST (XDBIST), a novel method to efficiently compress and apply scan patterns generated by automatic test pattern generation (ATPG) in a logic built-in self-test architecture. Our method allows test patterns to have any number of unknown values with no degradation in compression and application efficiency. XDBIST does not require changing the core logic of the device under test (DUT); no test points or X-blockage logic need be inserted. The proposed solution guarantees the same high test coverage and diagnosis ability as deterministic scan-ATPG and uses the same tester flow, while reducing test data volume and tester cycles by more than 10 times.

1. Introduction

Testing digital circuits represents a significant portion of the design, manufacture and service costs. Larger and more complex designs require more expensive test equipment that can meet demands on accuracy and pin count [1]. Scan has long been the fundamental design-for-test (DFT) method to control test cost [2], [3]. However, for large designs, even the most efficient deterministic ATPG patterns require testers to store vast amounts of scan data and apply a large number of test cycles, so the cost of automatic test equipment (ATE) and the overall test cost remain high.

Self-testing, in the form of logic BIST, was initially used to perform in-field testing of complex products [4]. BIST has been widely investigated as a method to reduce test cost by drastically lowering test data [5]. Part of the cost-savings achieved by data reduction may be offset by increased test time resulting from the need to apply a larger number of BIST test patterns. However, one of the main obstacles to wide-spread adoption of logic BIST is the requirement to modify the DUT to eliminate or block unknown (X) values from propagating to the signature analyzer, often implemented as a multiple-input signature register (MISR) [6]. A single X would compromise the self-test by making the fault-free MISR signature unpredictable [7]. In BIST-based solutions, it is also desirable to insert test points to facilitate testing the DUT. Unfortunately, any modification to the core logic may lengthen timing-critical paths and therefore reduce design performance.

2. Summary of Proposed Solution

We present an efficient solution that combines deterministic ATPG and a logic BIST architecture, has full X-tolerance and achieves significant reductions in both test data and tester cycles, while ensuring the same high test coverage and diagnosability as deterministic scan-ATPG (where all scan cells can be controlled and observed). We hereafter refer to our method as X-tolerant deterministic BIST, or XDBIST. The DUT scan chains are reconfigured into a large number of short chains and connected to on-chip structures that interface with the tester. Test patterns are generated by a modified deterministic ATPG. Load values are compressed into LFSR seeds as part of test generation; all patterns can be controlled by LFSR seeds so that all “care bits” (scan cells that must be set to a certain value) are set to the desired value, while all other scan cells are set to pseudo-random values from the LFSR [8]. A small percentage of scan chains are selected to be unloaded during each pattern and the tester directly compares unloaded values. Selecting the scan chains to observe during each pattern and compressing selection control data are integrated with test generation.

Section 3 presents the overall XDBIST architecture; section 4 details the novel scanout selector and the method to select scan chains and compress selection data; section 5 compares scan-ATPG and XDBIST results on industrial designs and section 6 concludes this paper.

3. XDBIST Architecture

The proposed solution builds on previous work that applies deterministic patterns in a logic BIST architecture [8] and re-uses the input-side pseudo-random pattern generator (PRPG), the PRPG shadow that allows re-seeding with 0-cycle overhead and the configuration of DUT scan cells into a large number (e.g., 512) of short scan chains. The method of compressing deterministic ATPG patterns into LFSR seeds is also reused. However, XDBIST employs a novel approach that fully tolerates, rather than eliminates, unknown values in the design.

The design under test typically contains many sources of unknown values (X), such as buses, non-scan state elements, embedded memories, timing-sensitive paths, etc. If all X-sources in the DUT can be detected through pre-silicon anal-

0-7803-8106-8/03 $17.00 Copyright 2003 IEEE

ysis and the design can be modified to avoid X's from propagating to the scan chains, then a MISR-based signature analyzer can be successfully used [8]. However, in some cases, X sources may not be known before silicon (due, for example, to race conditions). In other cases DUT modifications to avoid X propagation are undesired for timing reasons. Previous solutions have included circuitry to mask selected unload values so that X's do not reach the MISR [9], [10], [11]. However, if the number of X's is too large or their distribution is unfavorable, masking may have a high area overhead or it may not be selective enough, so non-X values would also be masked out (for example, by masking out entire scan chains), resulting in a potential loss of coverage or increased number of patterns. Also, if unexpected X's are detected (due, for example, to race conditions found on the tester), then the test patterns and the masking hardware may need to be regenerated to provide new X-masks.

XDBIST can tolerate any number of X's propagating to the scan chains, with no degradation in compression or application efficiency. As with traditional scan-ATPG, unexpected X's can be masked off at the tester on a per-cell basis using the same pattern set. Masking X measures never results in unwanted masking of non-X values.

To achieve complete X-tolerance with significant tester data and cycle reduction, XDBIST must simultaneously reduce scan-in data, scan-out data and the number of tester cycles. Also, XDBIST patterns must fit in a scan-test flow for failure diagnosis. These goals are achieved in the XDBIST architecture as follows:

- Scan-in data is reduced by compressing deterministic ATPG patterns into LFSR seeds, as described in [8]. All pattern loads are controlled by LFSR seeds so that all care bits are set to desired values while all other scan cells are set to pseudo-random values from the LFSR, as originally described in [12].
- Scan-out data is reduced through a novel mechanism of selectively observing only the desired scan chains (section 4). Chain selection is controlled by a separate selector register (Figure 2). When unloading the selected scan chains, the tester can measure some values and mask others, as with deterministic patterns.
- Multiple internal parallel scan chains are used to lower the number of load/unload cycles per pattern, which reduces the overall test-application cycles even though the number of patterns may increase over scan-ATPG. To avoid adding cycles for initializing the PRPG, the PRPG-LFSR is coupled with a PRPG shadow that allows re-seeding with 0-cycle overhead [8]. A similar shadow is used for loading the selector control bits (Figure 2), so no extra cycles are added.

The tester views the DUT as a traditional scan-based design (Figure 1a). However, when compared to scan-ATPG, the design with XDBIST requires fewer shift cycles per load and can have fewer scan in and scan out pins (Figure 1b), resulting in significant test data and tester cycle reduction. With XDBIST, the chains loaded from the tester are not the internal chains of the DUT, but rather the registers of the decompressor and observe selector.

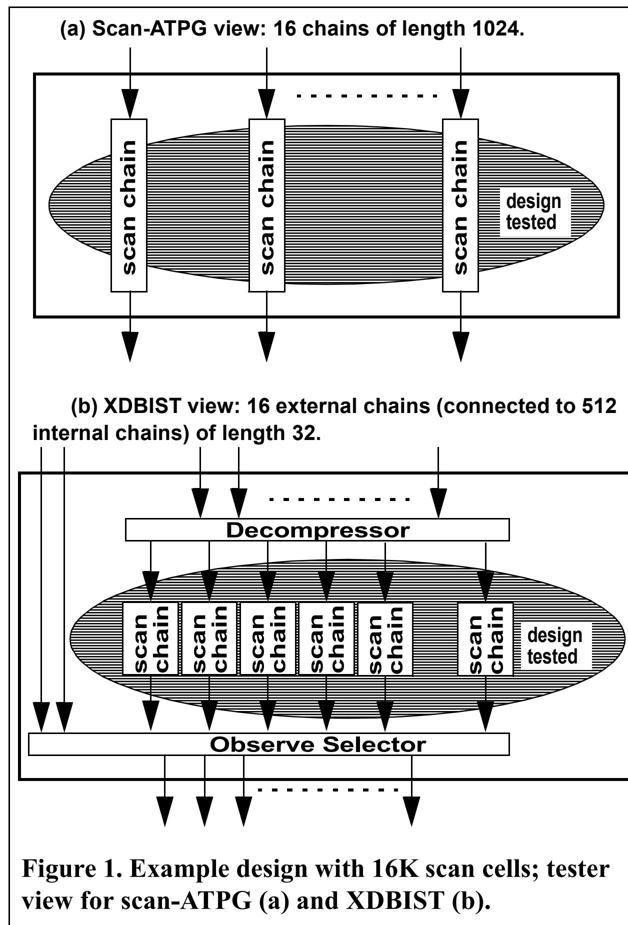

Figure 1. Example design with 16K scan cells; tester view for scan-ATPG (a) and XDBIST (b).

Unlike traditional logic BIST implementations [5], [8] primary inputs and outputs need not be isolated by scan cells. In effect, the XDBIST architecture maps only the loads and unloads of test patterns to reduce data volume and test time, while all other pattern events remain unaltered. This implies that the conditions necessary for detection of any fault can be mapped to XDBIST, not just single capture-cycle patterns. XDBIST pattern diagnosis is similar to that of deterministic ATPG patterns because all relevant scan data is directly unloaded for each pattern; a different diagnosis mode [13] is not required to unload scan data.

The decompressor and the observe selector of Figure 1b are detailed in Figure 2, which shows the overall XDBIST architecture. The decompressor comprises the PRPG shadow, the PRPG LFSR and the phase shifter. The PRPG

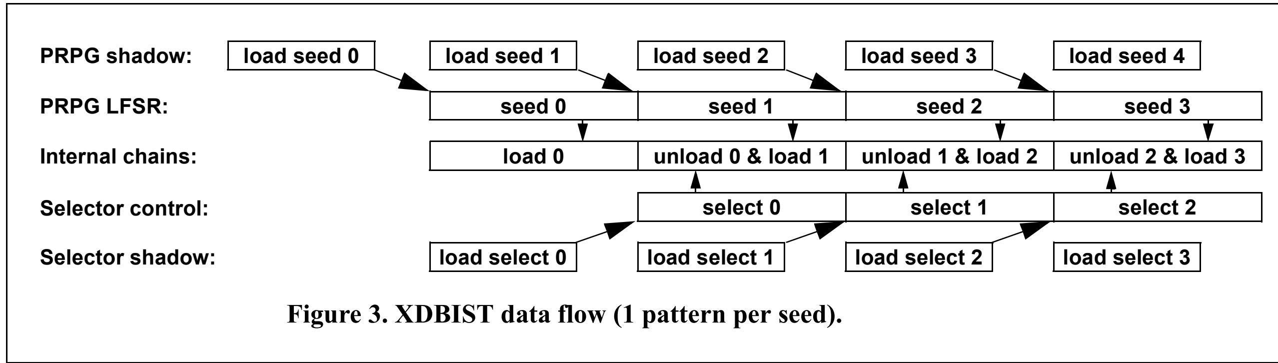

Figure 3. XDBIST data flow (1 pattern per seed).

shadow register, of the same length as the PRPG LFSR, is loaded through several parallel scan chains (shadow_si0 through shadow_si11). The internal chains of the tested design must be at least as long as the shadow chains so that the PRPG shadow can be fully loaded in the number of cycles it takes to load the internal chains. After a new PRPG seed has been loaded into the PRPG shadow, it is transferred to the PRPG LFSR during the functional capture cycle. The PRPG LFSR is re-seeded with 0-cycle overhead; it can be continuously clocked and the scan chains loaded with a new value every cycle.

The observe selector comprises the scanout selector, the selector control and the selector shadow (Figure 2). The selector shadow register, of the same length as the selector control register but shorter than the PRPG shadow, is loaded through several parallel scan chains (observe_si0 through observe_si3) at the same time the PRPG shadow is loaded, adding no extra cycles.The selector shadow data is transferred to the selector control at the same time the PRPG shadow data is transferred to the PRPG LFSR. Every pattern uses its own selector control data. The selector control determines which internal scan chains are selected by the scanout selector to be routed and observed on scanout pins so0 ...so15.

Figure 3 shows the overlap of operations in the XDBIST flow. After the initial load of seed 0, all operations are fully overlapped so there are no cycles overhead to load PRPG seeds or selector data. The load of pattern n is controlled by seed n in the PRPG LFSR, and is overlapped with the unload of pattern n-1 which is controlled by data n-1 in the selector control register. At the same time, seed n+1 is loaded into the PRPG shadow and control data n is loaded into the selector shadow. When load n is completed, seed n+1 is transferred from the PRPG shadow to the PRPG LFSR during the functional capture cycle. At the same time, control data n is transferred from the selector shadow to the selector control register, to prepare for the next unload. Thus, XDBIST pattern application does not require additional cycles over scan-ATPG patterns. The overlap of load/unload operations in scan-ATPG patterns has been extended in XDBIST to include the overlap of loading the PRPG and selector shadows.

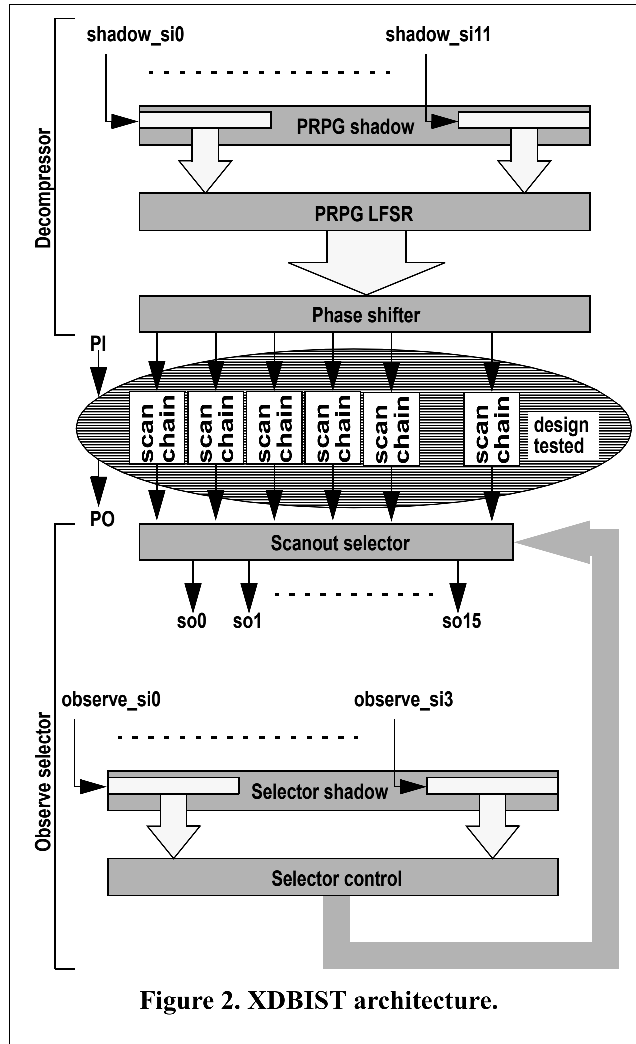

Figure 2. XDBIST architecture.

The pseudo-random values generated by the PRPG LFSR are first phase-shifted to eliminate structural dependencies ([5]) and then scanned into the internal chains of the tested design (Figure 2). Values shifted out from selected internal chains can be scanned out. The internal chain selection is controlled by the observe selector, described in section 4.

4. The Scanout Selector

The concept of selectively observing scan chains in XDBIST is practical because detection of a fault requires a single observe cell. Thus, it should be feasible to observe only a small percentage of the scan chains in a given pattern without significantly increasing XDBIST pattern count to achieve the same test coverage as scan-ATPG. Further, ATPG is modified to target as many faults as possible that can be observed in only a few scan chains, allowing the scanout selector to observe the same scan chains during the entire unload of a pattern. If the scanout selector had been required to switch scan-chain selection at every shift cycle to observe selected cells, the complexity and hardware overhead of the observe selector would increase significantly.

The scan cells of the DUT are configured into 512 internal chains. The number of scanout pins is chosen to balance data per pattern and pattern count: fewer scanout pins reduces unload data per pattern but too few pins limits observability, increasing pattern count. Therefore, the scanout selector is designed for 512 internal scan chains and 16 external scanout pins; $\frac{16}{512} = \frac{1}{32} \approx 3\,\%$ of the scan chains can be observed every pattern. The number of scan inputs was chosen to be equal to the number of scan outputs, 16, to optimize tester memory utilization; 12 scan inputs are dedicated to loading PRPG seeds and the other 4 inputs to loading selector data. When only about 3% of the unload data is observed, the number of XDBIST patterns increases by approximately 67% (section 5) compared to the number of tightly compressed scan-ATPG patterns for the same test coverage. However, data volume per pattern is reduced to about 3%. The number of tester cycles per pattern is reduced about 32 times because internal scan chains can be 32 times shorter than for scan ATPG with the same number of pins (Figure 1). Therefore, XDBIST achieves at least a 10x reduction in both tester data and cycles compared to scan-ATPG, as detailed in section 5. At the same time, XDBIST is fully compatible with scan-test flows: X's can be masked at the tester and diagnosis is similar to scan-diagnosis because unaltered scan-chain data is unloaded every pattern.

To minimize XDBIST data and tester cycles, the number of patterns must be minimal. The ability to select any desired set of chains will affect the pattern count. Therefore, design of the scanout selector must maximize the probability of being able to observe any desired subset of scan chains.

The most efficient scanout selector is fully connected (Figure 4): every scan chain connects to every scanout pin, so any subset of 16 chains can be observed. Unfortunately, the fully connected selector is impractical because of its high overhead:

- Gate count: Sixteen 512-to-1 multiplexors, equivalent to 8176 2-to-1 multiplexors.
- Delay: Each path from a scan chain to a scanout pin passes through nine 2-to-1 multiplexors, likely exceeding the clock cycle time.
- The wiring adds significantly to area and timing: each scan chain has fanout 16 and each MUX has fanin 521 (512 data inputs and 9 select inputs).
- Selector control requires 16 x 9 = 144 bits.

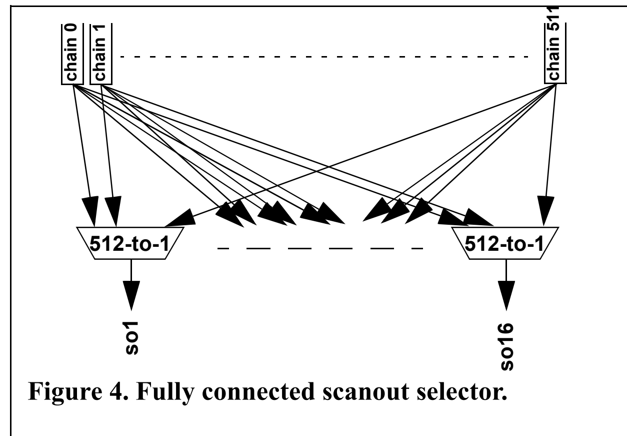

Figure 4. Fully connected scanout selector.

The characteristics of this selector are shown in Table 1, column "Fully connected selector".

At the opposite extreme, the smallest and simplest scanout selector connects groups of 32 scan chains through a multiplexor to a scanout pin (Figure 5). The area and timing impact of this selector are minimal:

- Gate count: Sixteen 32-to-1 multiplexors, equivalent to 496 2-to-1 multiplexors.
- Delay: Each path from a scan chain to a scanout pin passes through only five 2-to-1 multiplexors.
- Wiring: Each scan chain has fanout one and each MUX has fanin 37 (32 data inputs and 5 select inputs).
- Selector control requires 16 x 5 = 80 bits.

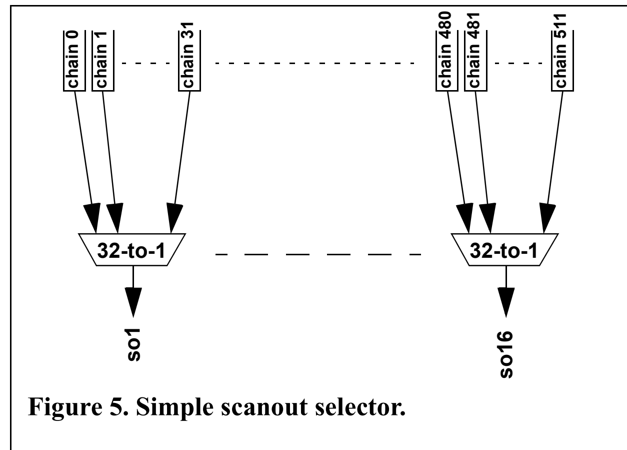

Figure 5. Simple scanout selector.

The simple selector restricts which subsets of scan chains can be simultaneously observed: for example, selecting chain 0 implies that chains 1, 2, .., 31 cannot be observed in the same pattern. To evaluate the selector performance (and thus the efficiency of XDBIST patterns), we determine the probability of successfully observing all chains in a randomly selected subset of up to 16 chains. This probability can be derived analytically or experimentally. For the latter, we generated 1000 random selections for each subset from 1 to 16 and determined whether the selected chains can all be routed to scanout pins. As shown in Table 1, column "Simple selector", the probability to observe only 8 out of 16 chains is less than 30% and decreases rapidly as more chains are added. To reduce pattern count and CPU time, ATPG must be able to efficiently generate patterns that maximize fault detection. Therefore, ATPG must be able to select almost any combination of scan chains for observation in every pattern; unfortunately, the simple selector does not meet this requirement.

4.1. Scanout selector architecture

An optimal scanout selector needs to offer a reasonable trade-off between multiple requirements:

(1) Minimize total area, including selector gates and control logic.

(2) Minimize delay from the scan chains to the scanout pins.

(3) Minimize wire routing (reduce area and power); we use the MUX fanin and scan chain fanout numbers to estimate routing requirements.

(4) Minimize data stored to ensure desired MUX selection.

(5) Maximize the probability of being able to observe all chains of a randomly selected set of up to 16 chains.

(6) Minimize the number of selector control bits.

The proposed scanout selector is shown in Figure 6; it is a two-stage selector, with each stage having a fanout of two. The first stage achieves a compression of 8 to 1 (from 512 to 64) and the second stage a compression of 4 to 1 (from 64 to 16). The area and timing impact of this selector are:

- Gate count: 64 16-to-1 multiplexors and 16 8-to-1 multiplexors, equivalent to a total of 1072 2-to-1 multiplexors.
- Delay: Each path from a scan chain to a scanout pin passes through seven 2-to-1 multiplexors.
- Wiring: Each level has fanout two and the highest MUX fanin is 20 (16 data inputs and 4 select inputs).
- Selector control requires 160 bits (section 4.2).

At each of the two levels, each input is connected to two different outputs and every two outputs share only a single input. The connections are chosen to maximize the probability that randomly selected inputs can be routed to outputs at each level. Each input (chain0, chain1, etc.) fans out to two first-stage outputs (the 16-to-1 MUXes in Figure 6). This configuration can be represented as a graph, with inputs as edges and outputs as vertices [14]. An edge (input) connects the two vertices (outputs) that the input fans out to. For example, chain0 fans out to MUXes (0) and (1); this is represented as an edge "chain0" connecting vertices "(0)" and "(1)." Vertex "(0)" also connects through edge "chain1" to some other vertex, etc. Similarly, the second stage of the selector (from the 16-to-1 MUXes as inputs to the outputs **so0**, ..., **so15**) can also be represented as a graph.

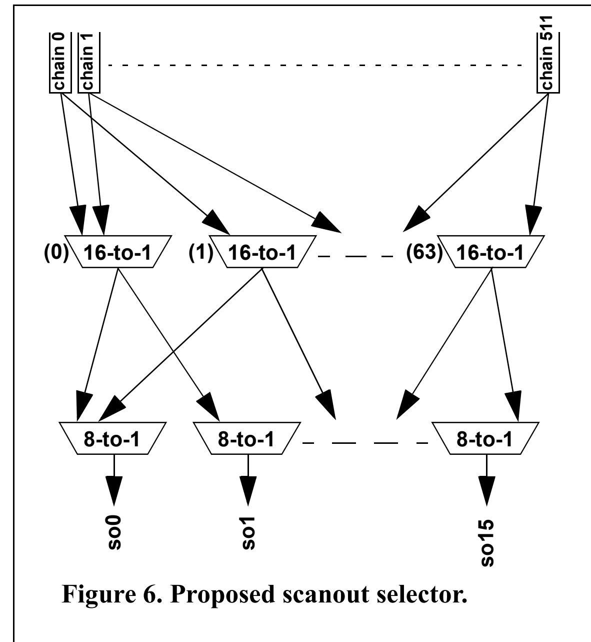

Figure 6. Proposed scanout selector.

Routing selected inputs to available outputs can be analyzed for each of the two levels of the selector in Figure 6 as uniquely assigning vertices to selected edges. If a set of p edges span at least p vertices, then such an assignment is possible; otherwise, it is not. A worst case; i.e., a set of p edges that span the minimal number of vertices, occurs when the p edges form a cycle, spanning only p vertices. If the smallest number of edges on any circle of the graph is g, then g is called the girth of the graph [15]. It can be shown that any number of edges up to $1.5g$ can be assigned unique vertices. Graphs of maximal g (given the number of edges and vertices) are called maximal-girth (MG) graphs [14]. Building each selector stage on a MG-graph ensures that any number of inputs up to at least $1.5g$ can be routed to outputs. It is still possible that many more than $1.5g$ inputs can be routed to outputs.

For example, Figure 7 shows an MG-graph of girth 3 and Figure 8 shows the 10-input, 5-output selector based on the graph in Figure 7; each of the 5 output nodes is implemented as a 4-to-1 MUX.

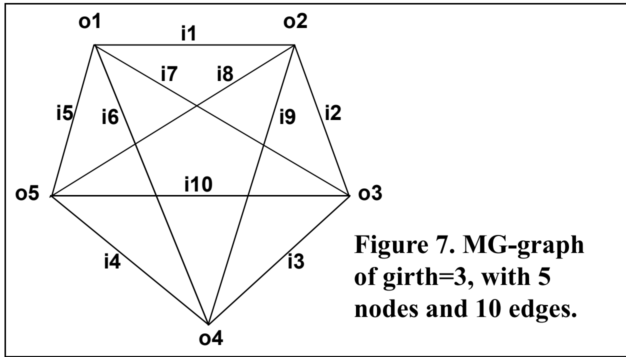

Figure 7. MG-graph of girth=3, with 5 nodes and 10 edges.

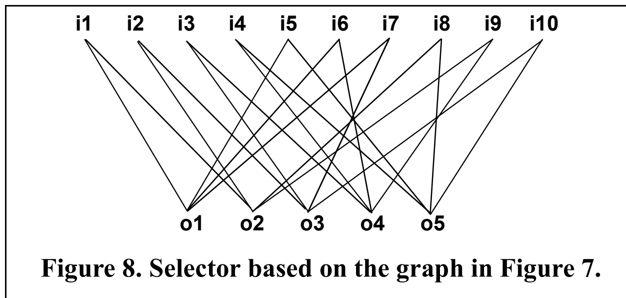

Figure 8. Selector based on the graph in Figure 7.

The first stage of the selector in Figure 6 is implemented based on an MG-graph of 64 vertices and 512 edges, with girth = 4; the second stage is based on an MG-graph of 16 vertices and 64 edges, with girth = 4.

Unlike the straightforward (but impractical) selectors of Figure 4 and Figure 5, routing selected scan chains to scanout pins is not trivial for the selector in Figure 6: at each level, there are multiple choices of routing an input and a “bad” choice may block another input from being routed. For example, routing chain 0 to MUX (0) implies that chain 1 (if also needed) cannot also be routed through MUX (0) but must pass through some other MUX (x) to which it is connected. If MUX (x) is already allocated to another input, then chain 1 is blocked. If, instead, chain 0 were routed through MUX (1), then chain 1 could use MUX (0) and avoid the blockage. The routing algorithm would be too slow if it searched all possible routing options to determine if a solution exists for the selected subset of scan chains. Instead, each level is routed independently based on a heuristic algorithm. First, active inputs are routed to the intermediate level MUXes of the scanout selector (Figure 6). The remaining unused MUXes, if any, are also assigned to active inputs. Next, intermediate MUXes are routed to outputs, exploiting multiple input paths where needed. If all inputs have been routed to outputs, then the overall routing operation has succeeded.

4.2. Scanout selector control

In the scanout selector of Figure 6, the 16 output MUXes are all utilized if 16 chains are routed, so 16 x 3 = 48 select bits are needed for the output MUXes. However, only 16 of the 64 intermediate MUXes route scanout data, while the other 48 MUXes can have arbitrary controls, so 16 x 4 = 64 “care bits” are needed out of a total of 64 x 4 = 256 select lines. To avoid any additional cycles, the loading of selector control data must overlap with the loading of the PRPG seed. Further, to optimize utilization of tester memory, the number of scan inputs is chosen equal to the number of scan outputs (16), as in Figure 1, leaving 4 scan inputs to supply the observe selection data (Figure 2). Setting the number of cycles to load PRPG seeds and selector data to 40, it follows that 160 bits can be supplied to the observe selector. These 160 bits must supply the 48 + 64 = 112 care bits and a total of 48 + 256 = 304 select lines.

The detailed architecture of the observe selector is shown in Figure 9. The selector shadow is divided into four parallel scan chains, each 40 bits long; thus, it takes 40 cycles to load the selector shadow. (The PRPG shadow is

Table 1. Selector Analysis

	Simple selector	Proposed selector	Fully connected selector
Gates:	496 MUXes	**1072 MUXes**	8176 MUXes
Delay:	5 MUXes	**7 MUXes**	9 MUXes
Wiring:	fanin = 37 fanout = 1	**fanin = 20** **fanout = 2**	fanin = 521 fanout = 16
Selector control:	80 bits	**160 bits**	144 bits
#chains selected	**Probability to observe all selected chains**		
1	100.00%	**100.00%**	100.00%
2	93.10%	**100.00%**	100.00%
3	88.10%	**100.00%**	100.00%
4	77.00%	**100.00%**	100.00%
5	65.20%	**99.90%**	100.00%
6	51.70%	**99.80%**	100.00%
7	41.60%	**99.50%**	100.00%
8	29.30%	**98.70%**	100.00%
9	18.70%	**96.40%**	100.00%
10	10.70%	**95.50%**	100.00%
11	5.60%	**92.30%**	100.00%
12	3.10%	**86.80%**	100.00%
13	1.70%	**81.60%**	100.00%
14	0.90%	**74.40%**	100.00%
15	0.00%	**61.40%**	100.00%
16	0.00%	**43.70%**	100.00%

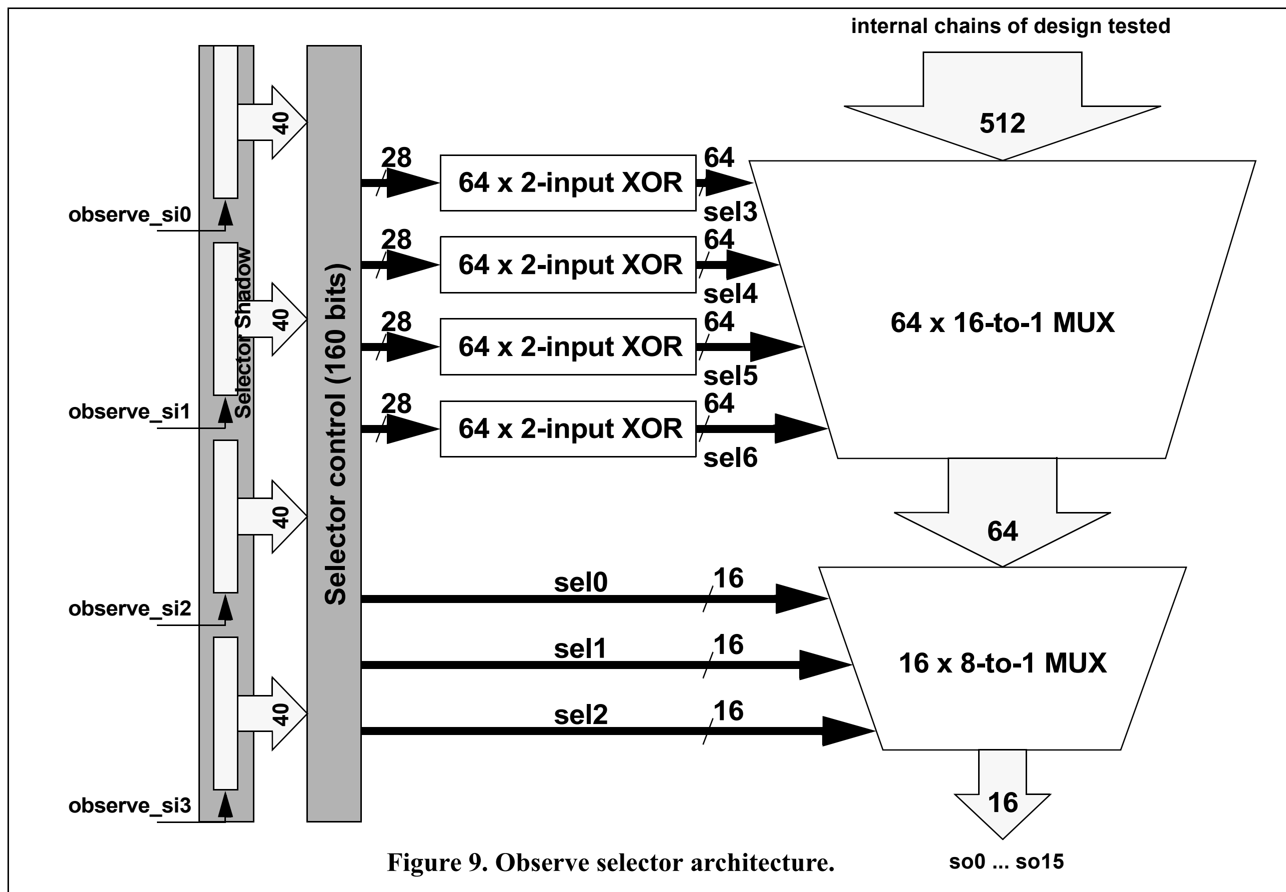

Figure 9. Observe selector architecture.

loaded at the same time as the selector shadow and uses 12 parallel scan chains, so the PRPG LFSR can be up to 480 bits long without requiring additional cycles or scan inputs to load.) The 160 bits loaded from the selector shadow to the selector control register are utilized as follows: 48 bits directly select the 16 8-to-1 output MUXes that are active in every pattern; **sel0**, **sel1** and **sel2** represent, respectively, select line 0, 1 and 2 of each of the 16 MUXes. The remaining 112 bits are decoded as four sets of 28 bits; sel3, sel4, sel5 and sel6 represent, respectively, select line 0, 1, 2 and 3 of each of the 64 MUXes in the first stage of the scanout selector. Because at most 16 first-stage MUXes route data in any one pattern, at most 16 of each set of 64 lines need to have set values (are care bits).

During test generation, ATPG selects a set of scan chains to be observed in a pattern by starting with a single chain used in detection of the primary target fault and then adding chains used in detection of secondary target faults, so that as many as possible chains can be simultaneously observed (section 4.3). To determine if a set of selected chains can be observed, the routing algorithm (section 4.1) is first run. If successful, a path through the first and second stages of MUXes has been found which routes each of the selected chains to at least one output **so0**, ..., **so15**. Next, a system of linear equations is created for each of the four 28-to-64 decoders (Figure 9): each first-stage MUX that routes a selected chain requires four select inputs, thus contributes an equation to each decoder's system of equations, where the variables are the values of the 28 inputs of each decoder. If all four systems of equations have solutions, all select lines can be set to required values and the chosen set of chains can be routed. Therefore, the four decoders must be designed to maximize the number of equations (outputs) that can be solved for the 28 variables (inputs). However, to minimize the hardware overhead of the four decoders (Figure 9), only a single 2-input XOR gate is used for each decoded output.

Each of the four 28-to-64 decoders can be represented as a graph; but, unlike the representation used in 4.1, inputs are represented as vertices and outputs as edges. Every edge (output) connects two vertices (the inputs that are XORed by the output). Figure 10 shows a decoder based on the graph of Figure 7; each output node is implemented as an XOR of its two inputs.

The set of linear equations has a solution if the selected outputs are linearly independent. In the graph representation, each output corresponds to an edge and the set of equations has a solution if the selected edges do not form a cycle. Therefore, to maximize the probability that an input combination exists which sets all outputs to desired values, each

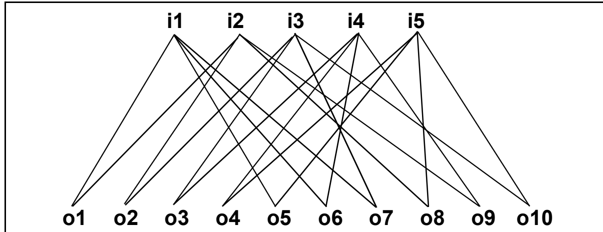

Figure 10. Decoder based on the graph in Figure 7.

28-to-64 decoder is based on an MG graph [14]; the decoders in Figure 9 are based on a graph with 28 vertices, 64 edges and girth = 4. The ability to route an arbitrary set of input scan chains and find the needed selector control bits was tested by generating 1000 random sets for each subset from 1 to 16 and determining if the selected chains can all be routed to outputs, and the selector control lines can be set by solving the four linear systems for the 28-to-64 decoders. The results are shown in Table 1, column "Proposed selector." At the beginning of test generation, every pattern can detect a very large number of faults by observing almost any full set of up to 16 scan chains; therefore, the probabilities of observing arbitrary selections of up to 16 chains must be sufficiently high (Table 1, column "Proposed selector"). The latter part of test generation focuses on hard-to-detect faults, which account for most patterns. During this stage, merging as many faults as possible into few patterns requires that a small set of chains be observed; not being able to route the selected chains to outputs would increase pattern count because hard-to-detect faults typically offer few observation opportunities. The observe selector presented supports this requirement by ensuring very high observation probabilities for few chains (Table 1, column "Proposed selector.")

4.3. XDBIST ATPG

In traditional deterministic scan-ATPG a fault is considered observed as soon as the fault effect was sensitized and propagated to a primary output or scan cell. In XDBIST however, only 3% of the scan cells are observed during a pattern, so special consideration must be given to observable scan chains. The test generator was modified to include chain observation as part of the conditions required for fault detection (Figure 11). After creating the fault list, a primary target fault is chosen and a new pattern is started; the scan chain where the fault effect was observed is added to a new chain set. Next, a secondary target fault is chosen which can be observed in a chain already in the chain set or in a new chain that can be added to the chain set. A new chain can be added to the chain set if all chains can be simultaneously routed (section 4.1) and the required MUX selection lines can be provided (section 4.2). If the test generator succeeds in expanding the pattern to test the secondary fault, then the new chain, if any, is added to the chain set. Similarly, additional secondary faults are added to the pattern. When no further secondary faults can be found, the pattern is complete; the selector control values are computed and stored as part of the pattern. Next, the fault simulator determines all the chains observed when using the stored selector control data, which may include additional chains not targeted by the test generator; all faults detected in any observed chain are removed from the fault list. The entire process is then repeated until the fault list is empty.

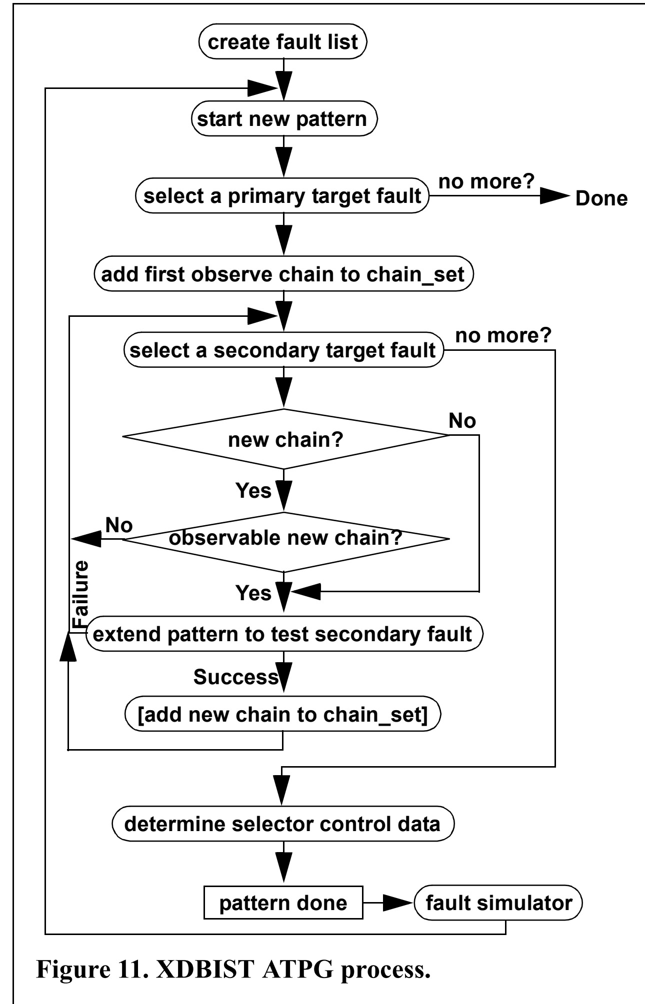

Figure 11. XDBIST ATPG process.

5. Results

The input and output components of XDBIST have been implemented as generic library elements that can be automatically instantiated and technology-mapped during test synthesis. The decompressor offers the user the choice of different-length PRPG-LFSRs, with matching PRPG-shadow and phase shifter. Each phase shifter is designed for its corresponding LFSR and supports at least 512 internal chains, with a phase shift of at least 2048 cycles between any two chains. To minimize area overhead, the phase shifters require only a single 2-input XOR gate for each internal scan chain. The scanout selector is implemented using fully

decoded tri-state driver MUXes to minimize delay and area overhead. Table 2 shows the equivalent area of the XDBIST components, including circuitry added to support testing the decompressor and observe selector. All gates are normalized to equivalent 2-input NAND gates as follows:

- AND, NAND, OR, NOR: 1 equivalent gate each.
- TSD: 1 equivalent gate each.
- XOR, MUX: 3 equivalent gates each.
- DFF: 6 equivalent gates each.

Table 2. XDBIST components equivalent area

Gate type	479 PRPG w/ shadow	257 PRPG w/ shadow	phase shifter	observe selector
DFF (x6)	958	514		320
MUX (x3)	491	265		163
XOR (x3)	93	125	512	256
[N]AND [N]OR (x1)	3	3		1668
TSD (x1)				1152
TOTAL equiv. gates	7503	4257	1536	5997

Table 3 shows the total area of two XDBIST configurations: a 257-bit LFSR configuration for smaller designs and a 479-bit LFSR configuration for larger designs. For very large designs, multiple decompressors and observe selectors may be used as needed. The total area overhead per scan chain is 23 to 29 equivalent gates.

Table 3. XDBIST area overhead

	257-bit decompressor + observe selector	479-bit decompressor + observe selector
Total	11,790 equiv. gates	15,036 equiv. gates
Per chain	**23 equiv. gates**	**29 equiv. gates**

Using multiple decompressors and observe selectors for a design may result in a small decrease in the number of patterns because more care bits are available from multiple PRPGs. The test-data volume is dominated by observe data and remains about the same because about 3% of the scan cells are observed, independent of the number of observe selectors. More importantly, multiple decompressors and observe selectors allow using more and, therefore, shorter scan chains, which directly decreases tester cycles.

Table 4 shows results obtained by applying XDBIST to four industrial designs, and compares the results to highly-optimized scan-ATPG; one or more decompressors and observe selectors were used to evaluate each design. For all designs, the fault coverage obtained from scan-ATPG and from XDBIST were about the same, ranging from 96% to 99%+. The CPU time for XDBIST was up to 20% lower than the CPU time for scan-ATPG, even though XDBIST generates more patterns and performs additional computations to encode patterns as PRPG seeds and observe selector control data. The CPU time is lower because XDBIST has limited controllability (the number of care bits obtained from the PRPG) and observability (the scan chains routed through the observe selector) and, therefore, doesn't expend as much time in merging faults into each pattern. The number of XDBIST patterns is higher than the number of scan-ATPG patterns, on the average, by 45%, 121%, 73% and 30% respectively for the four designs; on the average, XDBIST pattern count is 67% higher. All XDBIST configurations shown in Table 4 use the 479-bit PRPG; smaller designs can reduce area overhead by using a smaller PRPG. From 1 to 11 decompressors and observe selectors have been used in the data shown, keeping the total XDBIST area overhead less than 3% for the two smaller designs, and less than 2% for the two larger designs in Table 4. Assuming that scan cells are evenly distributed over all available scan chains, the data reduction averages 28x and the cycles reduction averages 16x, compared to scan-ATPG, while achieving the same test coverage.

The data reductions were computed as follows:

$$\frac{Datascan}{DataXDBIST} = \frac{3(patsscan)(scancells)}{(patsXDBIST)\left(640 + \frac{2scancells}{32}\right)} \quad (1)$$

assuming 1 bit is used for each load value and 2 bits for each unload value; each XDBIST pattern loads a 479-bit PRPG seed + 160 bits of selector control data = 639 bits (rounded up to 640). The data for primary input forces, primary output measures and capture clocks are very small compared to load/unload data and have not been included in equation (1).

The cycles reduction were computed as follows:

$$\frac{Cyclesscan}{CyclesXDBIST} = \frac{patsscan\frac{scancells}{chainsscan}}{patsXDBIST\frac{scancells}{chainsXDBIST}} \quad (2)$$

ignoring the contribution of capture cycles.

6. Conclusions

The presented XDBIST method reduces test-data volume and tester cycles by a factor of more than 10, while achieving the same high test coverage as scan-ATPG. The method tolerates any number of X's, so the core of the tested device is not changed; test points or X-blockage logic are not used. The scan cells of the DUT are configured into

Table 4. XDBIST results on industrial designs

Design size			Scan-ATPG results		XDBIST results			XDBIST savings		XDBIST cost
#gates	#scan cells	#faults	#chains	#patterns	#decompressors,selectors	#chains	#patterns	data reduction	cycles reduction	area overhead
1.1M	44K	2.1M	13	3,786	1	512	6,019	**24x**	**25x**	**1.5%**
					2	1024	4,960	**30x**	**60x**	**2.9%**
1.8M	73K	3.8M	109	2,153	1	512	6,325	**14x**	**2x**	**0.8%**
					2	1024	4,361	**21x**	**5x**	**1.7%**
					3	1536	3,638	**25x**	**8x**	**2.5%**
6.2M	237K	14.2M	96	8,445	2	1024	15,222	**25x**	**6x**	**0.5%**
					3	1536	14,884	**26x**	**9x**	**0.8%**
					4	2048	15,463	**25x**	**12x**	**1.0%**
					5	2560	12,877	**30x**	**17x**	**1.3%**
10.0M	467K	22.6M	196	23,627	5	2560	31,148	**36x**	**10x**	**0.8%**
					7	3584	30,699	**36x**	**14x**	**1.1%**
					9	4608	30,446	**36x**	**18x**	**1.4%**
					11	5632	30,708	**36x**	**22x**	**1.7%**
							Average:	**28x**	**16x**	**1.4%**

many short internal chains; a decompressor and an observe selector are added between the internal chains and the external interface. The area overhead of the added logic is about 26 equivalent gates (about 9 XOR gates) per scan chain, or 1.4% on average. XDBIST patterns are generated by a modified deterministic ATPG and compacted into decompressor seeds, observe selector controls and a small subset of unload data. Unlike standard BIST methods, XDBIST patterns can directly force primary inputs and measure primary outputs, as in scan ATPG. Diagnosis is similar to scan ATPG and can detect the failing gate(s), not just the failing scan cells, because all relevant pattern data is directly scanned out.

Acknowledgments

We are very grateful to Martin Funcell, Francisco da Silva, Rayleigh Chin, Sandeep Kaushik, Bill Lloyd, Tom Finklea, Padmashree Takkars and the entire team for their contributions to the development of this work.

References

[1] Semiconductor Industry Association (SIA), *International Technology Roadmap for Semiconductors (ITRS)*, 1999.

[2] M. Abramovici, M.A. Breuer, A.D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, 1990.

[3] E.B. Eichelberger, E. Lindbloom, J.A. Waicukauski, T.W. Williams, *Structured Logic Testing*, Prentice-Hall, 1991.

[4] H.J. Nadig, “Testing a Microprocessor Product Using a Signature Analysis”, *International Test Conf.* 1978, pp.159-169.

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

[6] P.H. Bardell, W.H. McAnney, “Self-Testing of Multichip Logic Modules”, *International Test Conf.* 1982, pp.200-204.

[7] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan, J. Rajski, “Logic BIST for Large Industrial Designs: Real Issues and Case Studies”, *International Test Conference* 1999, pp.358-367.

[8] P. Wohl, J.A. Waicukauski, S. Patel, M. Amin, “Efficient Compression and Application of Deterministic Patterns in a Logic BIST Architecture”, *Design Automation Conference* 2003, pp. 566-569.

[9] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, B. Koenemann, “OPMISR: The Foundation for Compressed ATPG Vectors”, *International Test Conference* 2001, pp. 748-757.

[10] J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee, R. Thompson, K.H. Tsai, A. Hertwig, N. Tamarapalli. G. Mrugalski, G. Eide, J. Qian, “Embedded Deterministic Test for Low Cost Manufacturing Test”, *International Test Conf.* 2002, pp. 301-310.

[11] I. Pomeranz, S. Kundu, S.M. Reddy, “On Output Response Compression in the Presence of Unknown Output Values”, *Design Automation Conference* 2002, pp. 255-258.

[12] B. Könemann, “LFSR-Coded Test Patterns for Scan Designs”, *European Test Conference*, Munich, 1991.

[13] P. Wohl, J.A. Waicukauski, S. Patel, G. Maston, “Effective Diagnostics through Interval Unloads in a BIST Environment”, *Design Automation Conference* 2002, pp. 249-254.

[14] P. Wohl, L. Huisman, “Analysis and Design of Optimal Combinational Compactors”, VLSI Test Symposium, 2003.

[15] J. A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, North Holland, 1976.