Synthesis of Zero-Aliasing Elementary-Tree Space Compactors

Bahram Pouya and Nur A. Touba

Computer Engineering Research Center
Department of Electrical and Computer Engineering
University of Texas, Austin, TX 78712-1084
E-mail: {pouya, touba}@ece.utexas.edu

Abstract

A new method is presented for designing space compactors for either deterministic testing or pseudo-random testing. A tree of elementary gates (AND, OR, NAND, NOR) is used to combine the outputs of the circuit-under-test (CUT) in a way that zero-aliasing is guaranteed with no modification of the CUT. The elementary-tree is synthesized by adding one gate at a time without introducing redundancy. The end result is a cascaded network, CUT followed by space compactor, that is irredundant and has fewer outputs than the CUT alone. All faults in the CUT and space compactor can be tested. Only the outputs of the space compactor need to be observed during testing. Experimental results are surprising; they show that very high compaction ratios can be achieved with zero-aliasing elementary-tree space compactors. Compared with parity trees and other space compactor designs that have been proposed, the method presented here requires less overhead and yet guarantees zero-aliasing.

1. Introduction

The test response of a circuit-under-test (CUT) must be checked to see if it is fault-free. In order to reduce the volume of data that must be checked, the test response is usually compacted. There are two types of compaction: space compaction and time compaction [Saluja 83]. *Space compaction* involves using combinational logic to merge n response streams into k response streams (where $k < n$). Space compaction reduces the number of bits of test response that are generated each clock cycle. The *compaction ratio* for a space compactor is defined as n / k. An example of a space compactor is a parity-tree. *Time compaction* involves using sequential logic to combine the test response during one clock cycle with the test response from previous clock cycles to form a signature. An example of a time compactor is a multiple-input signature register (MISR).

This paper focuses on space compaction. There are many applications where space compaction can be used to reduce test costs. Three major applications are described below:

1. Test point condensation - A very common design-for-test (DFT) technique is to insert observation points in the CUT to improve fault coverage [Hayes 74]. This is especially the case for non-scan or partial scan designs [Chickermane 93], [Rudnick 94], and for pseudo-random pattern testing [Eichelberger 83]. A space compactor can be used to combine (or "condense" [Fox 77]) the observation points to reduce the number of additional scan elements or chip pins that must be added to the design for testing.
2. Built-In Self-Test (BIST) - In BIST, the test response must be checked with on-chip hardware. Typically, MISR's are used to compact the test response into signatures which are then compared with fault-free reference signatures stored on-chip. A space compactor can be used to reduce the size of the MISR's that are required (as illustrated in Fig. 1). This also reduces the storage requirements for the fault-free reference signatures. This is especially helpful when checking multiple signatures.
3. Intellectual Property Core Testing - Consider the case where some user-defined logic (UDL) is driving the inputs of an intellectual property core (as illustrated in Fig. 2). One common DFT approach for testing embedded cores is to multiplex the core I/O's to the chip pins (this provides parallel access to the core) [Immaneni 90]. However, one problem that arises is how to test the UDL driving the core if the internal structure of the core is not known (i.e., it is a black block). Some means for observing the outputs of the UDL is required; a space compactor can be used to reduce the cost for this. Space compactors also support partial isolation ring [Touba 97] and UDL output space modification [Pouya 97] DFT techniques.

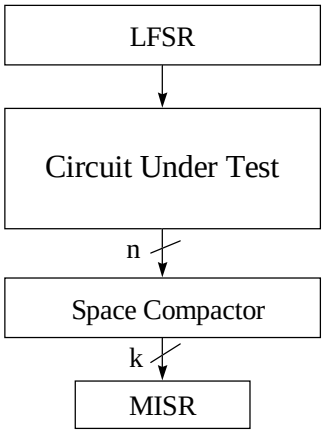


Figure 1. Using Space Compactor for BIST

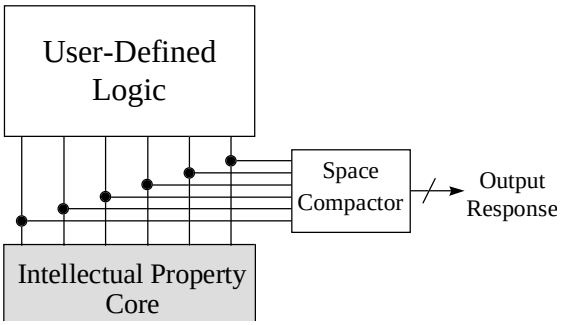


Figure 2. Using Space Compactor for Observing Logic Driving an IP Core

An important issue when compacting test response is aliasing. Aliasing occurs when the compacted test response for a faulty circuit is the same as the compacted test response for the fault-free circuit (despite the fact that the uncompacted responses were different). Aliasing results in reduced fault coverage. Most of the existing techniques for designing space compactors are based on probabilistic error models and thus do not guarantee aliasing-free compression [Li 87], [Karpovsky 87], [Reddy 88], [Jone 91], [Das 95], [Ivanov 96]. There are some recent techniques proposed by Chakrabarty, *et al.*, that do guarantee zero-aliasing [Chakrabarty 94, 95, 96].

In [Chakrabarty 94, 96], techniques are proposed for achieving zero-aliasing using a multiplexed parity-tree. In order for a fault to be detected with no aliasing through a parity tree, it must be sensitized to an odd number of outputs. However, some circuits contain faults that cannot be odd-sensitized. For those circuits, a procedure for adding additional fanout points or selective multiplexing is used to achieve zero-aliasing.

In [Chakrabarty 95], very interesting theoretical results are shown for constructing response graphs and designing optimal space compactors for a particular test set. However, if it is possible to choose a different test set for the CUT, then there are many possible minimal response graphs. The overhead for an optimal space compactor depends on the test set chosen. This degree of freedom is not utilized by the techniques in [Chakrabarty 95].

In this paper, a new method for designing zero-aliasing space compactors for either deterministic testing *or* pseudo-random testing is presented. Similar to the technique in [Li 87], a tree of elementary gates is constructed, however, only AND, OR, NAND, and NOR gates are used (no XOR gates). To compact *n* outputs into *k* outputs, *n* - *k* elementary gates are used (as illustrated in Fig. 3). A very efficient hill-climbing search strategy is used to construct the "elementary-tree" space compactor. All faults in both the CUT and the space compactor itself are guaranteed to be detected.

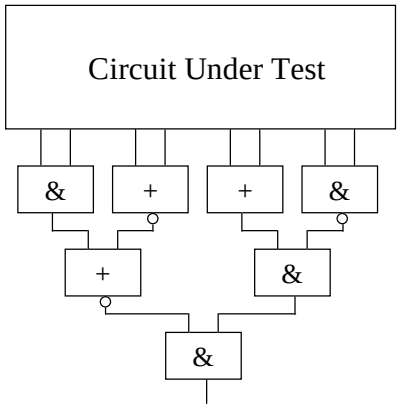


Figure 3. Example of Elementary-Tree Space Compactor

The advantages of a zero-aliasing elementary-tree space compactor over previous techniques are clear: zero-aliasing is achieved with no modifications to the CUT, the area and propagation delay of an elementary-tree are much less than a parity-tree, and the technique works for either deterministic or pseudo-random testing. The big question is what compaction ratio can be achieved with a zero-aliasing elementary-tree space compactor. The experimental results presented here are very surprising. For many circuits, the proposed synthesis procedure constructs a zero-aliasing elementary-tree space compactor that achieves maximum or near maximum compaction ratio for deterministic testing. The results for pseudo-random testing are also very good.

The paper is organized as follows: Section 2 describes the conditions for zero-aliasing. Section 3 presents a method for synthesizing zero-aliasing elementary-tree space compactors for deterministic testing. Section 4 describes how to modify the procedure in Sec. 3 for pseudo-random testing. Experimental results are shown in Sec. 5. Section 6 is a conclusion.

2. Zero-Aliasing Space Compaction

In order for a space compactor to have zero-aliasing for a particular fault class, all faults that are detectable at the outputs of the CUT must also be detectable at the outputs of the space compactor. For deterministic testing, the problem of constructing a zero-aliasing space compactor can be thought of as adding gates to combine the outputs of the CUT without introducing redundancy in the overall circuit (CUT + space compactor). The cascaded network, CUT followed by space compactor, should be irredundant with fewer outputs than the CUT alone. If it is irredundant, then all faults in either the CUT or space compactor can be detected at the outputs of the space compactor.

For pseudo-random testing (commonly used in BIST), the pseudo-random generator can be simulated to determine the pseudo-random test set. For zero-aliasing, any fault in the CUT that is detected at the outputs of the CUT by the pseudo-random test set should also be detected at the outputs of the space compactor. For pseudo-random testing, the problem of constructing a zero-aliasing space compactor can be thought of as adding gates to combine the outputs of the CUT without introducing redundancy *with respect to (w.r.t.) the pseudo-random test set*. In other words, the space compactor should not cause any fault in the CUT or in the space compactor itself to be left undetected by the pseudo-random test set. Of course, any faults in the CUT before adding the space compactor that are not detected (i.e., redundant w.r.t. the pseudo-random test set) are not considered when adding the space compactor -- there is no issue of aliasing because they are not detected to start with.

3. Synthesizing Space Compactors For Deterministic Testing

As shown in the previous section, the conditions for a zero-aliasing space compactor can be defined in terms of redundancy for either deterministic testing or pseudo-random testing. In this section, a procedure for constructing a zero-aliasing elementary-tree space compactor is described for deterministic testing. The same basic procedure applies for pseudo-random testing with a few differences which will be explained in Sec. 4.

Constructing a zero-aliasing elementary-tree space compactor involves combining the outputs of the CUT with elementary gates without introducing redundancy. There are two degrees of freedom in constructing an elementary-tree space compactor. One is choosing which pairs of outputs to combine in a particular gate, and the other is choosing which type of gate to combine them with. The number of possible elementary-tree space compactors is exponential in the number of outputs in the CUT. Thus, some heuristic strategy is needed in constructing the space compactor. A hill climbing strategy is used here where outputs continue to be combined until a point is reached where no more outputs can be combined without introducing redundancy (no backtracking is done). This simple and fast heuristic strategy yields very good results as will be shown in Sec. 5.

3.1 Overview of Procedure

The basic steps for synthesizing an elementary-tree space compactor are the following:

1. Choose a pair of outputs to combine (explained in Sec. 3.4).
2. Choose which type of gate to combine them with (explained in Sec. 3.3).
3. Check if any redundancy has been introduced (explained in Sec. 3.2).
4. If no redundancy, then stop if only one output remains, otherwise loop back to step 1.
5. If redundancy introduced, then choose another type of gate if possible and loop back to step 3.
6. If all gate types have been tried, then choose a different pair of outputs if possible and loop back to step 3.
7. If all pair of outputs have been tried, then stop.

This hill climbing procedure keeps combining outputs until there is only one output left or until it is no longer possible to do so without introducing redundancy. At that point, the procedure stops and the gates that are added for combining the outputs form the elementary-tree space compactor. Note that each time a pair of outputs are combined with a gate, the total number of outputs is reduced by one (the two original outputs are replaced by the output of the gate in which they are combined).

The worst case complexity of the procedure is $O(n^3)$ where n is the number of outputs in the CUT. However,

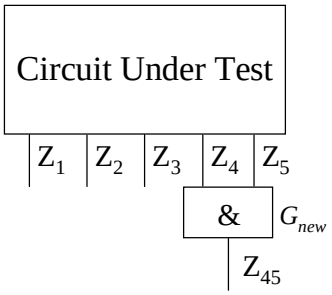


Figure 4. Example: Adding Gate G_{new} to Combine Outputs Z_4 and Z_5

	Z_1	Z_2	Z_3	Z_4	Z_5		Z_1	Z_2	Z_3	Z_{45}
Fault 1:	1	D'	0	D	1		1	D'	0	D
Fault 2:	0	1	1	0	D	G_{new} →	0	1	1	0
Fault 3:	D	D'	0	0	1		D	D'	0	0
Fault 4:	1	1	D	0	D'		1	1	D	0
Fault 5:	0	1	0	D	D		0	1	0	D

Figure 5. Example: 5-Valued Output Vectors Change When Gate G_{new} is Added

if the order in which the outputs are combined and the order in which the gate types are tried is chosen intelligently, then redundancy will rarely be found in step 3 except when the number of outputs gets very small. Thus, the procedure will rapidly combine outputs until the number of outputs gets small.

The three things that need to be discussed are how to choose which pair of outputs to combine, how to choose which type of gate to combine them with, and how to efficiently check for the introduction of redundancy. These are discussed in reverse order in the next three subsections. The issues related to checking for redundancy influence the decisions about which pair of outputs to combine and which gate type to use.

3.2 Efficiently Checking for the Introduction of Redundancy

To make the procedure practical for large circuits, an efficient means for checking for the introduction of redundancy is needed. The proposed technique for accomplishing this involves using 5-valued logic. In 5-valued logic, each logic value is in the following set {0, 1, X, D, D'}. D (D') represents the case where the fault-free value is 1 (0) and the faulty value is 0 (1). A test pattern is initially found for each fault in the CUT. Then 5-valued simulation is used to determine the 5-valued output vector for each test pattern. Each 5-valued output vector will have at least one D or D' since the fault is detected. When a pair of outputs is combined with a gate G_{new} (as illustrated in Fig. 4) then the 5-valued output vector for each fault is recalculated by simply performing the logic operation that corresponds to the gate type for G_{new} (as shown in Fig. 5). The example in Figs. 4 and 5 shows how the 5-valued output vector for each fault changes when outputs Z_4 and Z_5 are combined with an AND gate. The two columns corresponding to outputs Z_4 and Z_5 are combined into one column labeled Z_{45} which is computed by taking the logical AND of the values for Z_4 and Z_5.

After recalculating the output vectors, any fault whose new 5-valued output vector still contains a D or D' is guaranteed to not be redundant because it will still be detected through the space compactor by the same test vector. However, any fault whose new 5-valued output vector no longer contains a D or D' may potentially be redundant. In the example in Fig. 5, *Fault 2* is the only fault whose 5-valued output vector no longer contains a D or D', so *fault 2* may potentially be redundant.

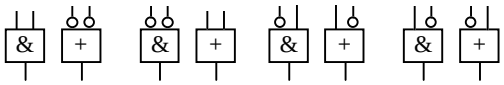
ATPG must be done to find a test pattern for any potentially redundant faults in the circuit with gate G_{new} included. If ATPG is successful in finding a test pattern, then 5-valued simulation is done for the test pattern to obtain the new 5-valued output vector for the fault (it will contain at least one D or D'). If ATPG is unsuccessful, then that means the fault is redundant and therefore the gate G_{new} cannot be used in the space compactor because it introduces redundancy.

If none of the faults in the fault list are made redundant by adding gate G_{new}, and the faults associated with G_{new} itself are irredundant, then G_{new} can be added to the zero-aliasing space compactor. The faults associated with G_{new} are added to the fault list for consideration in subsequent redundancy checks.

This procedure for checking for the introduction of redundancy minimizes the amount of ATPG that is performed. ATPG is only done for faults whose new 5-valued output vector doesn't contain a D or D'. As soon as any fault is found redundant, then no further ATPG is done. So the number of times that ATPG is targeting a redundant fault is limited to no more than once per redundancy check.

3.3 Choosing Gate Type for Combining Outputs

The proposed procedure for constructing the space compactor selects a pair of outputs to combine and then tries all the possible gate types for combining them. The order in which the gate types are tried is chosen to



0D or 0D’	----	Propagate	Propagate	----
1D or 1D’	Propagate	----	----	Propagate
D0 or D’0	----	Propagate	----	Propagate
D1 or D’1	Propagate	----	Propagate	----
DD or D’D’	Propagate	Propagate	----	----
DD’ or D’D	----	----	Propagate	Propagate

Figure 6. Error Propagation Properties for the Four Different Classes of Gates

minimize the amount of ATPG that is required when checking for the introduction of redundancy.

There are four different classes of elementary gates as far as error propagation properties are concerned. They are shown in Fig. 6. The different classes are shown as AND gates with inverters and OR gates with inverters. NAND and NOR gate equivalents can be found by straightforward use of De Morgan transformations. When constructing the elementary-tree space compactor, there is no concern with optimizing inverters since that can be done at the end using simple transformations.

In Fig. 6, each row corresponds to input combinations and each column corresponds to one of the four classes of elementary gates. Each entry indicates whether or not the error(s) at the input of the gate propagate to the output of the gate.

Using the error propagation information in Fig. 6, a quick check can be made to see how many faults will require ATPG when performing the redundancy check (i.e., how many 5-valued output vectors will not have a D or D’ bit). The gate types are then tried in order of increasing ATPG requirements. Generally the gate type that requires the least ATPG during the redundancy check is the one that is most likely to not introduce redundancy.

3.4 Choosing Pair of Outputs to Combine

The decision about which pair of outputs to combine each time determines the structure of the elementary-tree space compactor. It will not necessarily be a balanced tree. There are two proposed strategies for choosing the outputs to combine depending on what objective is more important.

If the delay through the space compactor is an issue, then a strategy for minimizing the delay is to always try to combine the pair of outputs at the lowest combinational depth first. The combinational depth of each output in the CUT is initially calculated. As gates are added to construct the space compactor, the depth of each output is updated. The output pairs are chosen in increasing order of depth. By so doing, the maximum delay through the CUT plus space compactor is minimized. The delay through the space compactor limits how fast the CUT can be clocked during testing. In some applications it may be important to minimize this delay.

If maximizing the compaction ratio is the most important objective, then the strategy is to choose the pair of outputs to combine that will result in the smallest decrease in the total number of D and D’ bits in the 5-valued output vectors for all the faults. Minimizing the number of D and D’ bits that are lost in compaction is a heuristic that attempts to minimize the chances that subsequent merging of outputs will introduce redundancy. Intuitively, the more outputs that each fault can be detected through, the less likely combining any pair of outputs will cause the fault to become redundant. If a fault can be detected through three or more outputs, then combining any pair of outputs is guaranteed not to cause the fault to become redundant. This is because even if the fault is masked at the two outputs that are combined, it will still be detected through the third output.

3.5 Summary of Procedure

The procedure outlined in Sec. 3.1 can be used to construct a zero-aliasing elementary-tree space compactor. The order in which the outputs are combined is chosen depending on whether minimizing delay through the compactor is the primary objective or whether maximizing the compression ratio is the primary objective. The gate type that is used to combine a particular pair of outputs is chosen based on minimizing APTG. Checking whether combining a pair of outputs through a particular gate type will introduce redundancy is done with an efficient procedure that minimizes the number of faults that need to be considered. Outputs are combined until there is only one output left or until a point is reached where no more outputs can be combined without introducing redundancy.

The resulting elementary-tree space compactor can be transformed and technology mapped. ATPG can be performed to find test vectors for every fault in both the CUT and space compactor to provide 100% fault coverage. Only the outputs of the space compactor needs to be observed during testing.

4. Synthesizing Space Compactors For Pseudo-Random Testing

Space compaction is often used in pseudo-random BIST environments. The test patterns that are applied to the CUT are determined by the pseudo-random test pattern generator circuit and thus cannot be selected or altered as they can with deterministic testing. However, the same procedure that was used for synthesizing a zero-aliasing elementary-tree space compactor for deterministic testing can also be used for pseudo-random testing. The only thing that changes is the procedure used for checking for the introduction of redundancy (i.e., step 3 in Sec. 3.1).

As was explained in Sec. 2, for pseudo-random testing, redundancy is defined w.r.t. the pseudo-random test set. So instead of using ATPG to check for the introduction of redundancy, fault simulation of the pseudo-random test set is used. Initially, fault simulation is done to determine which faults in the CUT are detected and to record one pattern in the test set that detects each fault. 5-valued simulation is then used to find the 5-valued output vector for each detected fault. As before, when checking for the introduction of redundancy, new 5-valued output vectors are computed for each fault based on the outputs that are being combined and the logical operation of the gate G_{new} that is being added. If the new 5-valued output vector for a fault doesn't have any D or D' bits in it, then the fault is potentially redundant w.r.t. the pseudo-random test set. Fault simulation of the potentially redundant faults for the pseudo-random test set is performed on the circuit that results from adding the gate G_{new}. Any fault that is not detected is redundant w.r.t. the pseudo-random test set. If all the faults are detected, then 5-valued simulation can be done for the pattern that detected each fault to find a new 5-valued output vector for the fault. As before, the faults associated with the gate G_{new} are added to the fault list.

The procedure for pseudo-random testing is basically the same as for deterministic testing except that fault simulation replaces ATPG. The time required for fault simulation is short due to the fact that only a small set of the faults (i.e., the potentially redundant faults) are simulated each time and fault dropping can be done. The compaction ratio that can be achieved for pseudo-random testing will be the same or less than that for deterministic testing because the set of test patterns is restricted.

When the procedure completes, the resulting elementary-tree space compactor will guarantee detection of all faults that are detected at the outputs of the CUT by the pseudo-random test set. Moreover, all faults in the space compactor itself will be detected by the pseudo-random test set. Only the outputs of the space compactor need to be observed during testing.

5. Experimental Results

The synthesis procedures described here were used to design zero-aliasing elementary-tree space compactors for the ISCAS 85 benchmark circuits [Brglez 85]. Table 1 shows the results for deterministic testing. The number of primary inputs, primary outputs, and the maximum number of logic levels for each of the benchmark circuits are shown followed by the results for adding a space compactor to the circuit. For the CUT cascaded with the space compactor, the table shows the number of primary outputs, the total levels of logic, the area overhead, the space compaction ratio, and the CPU times. The CPU times are shown for achieving a space compaction ratio of 2, 5, and the final space compaction ratio. The area overhead is computed by comparing the weighted gate count (gate count multiplied by the average fanin) of the CUT alone and the CUT with the space compactor. As can be seen, the area overhead is very small since no more than one elementary gate is added per CUT output. Note also that routing of the elementary-tree is very simple since it is a fanout-free circuit. The compaction ratio is quite substantial. Three of the circuits got down to just a single output. For comparison of the overhead, the zero-aliasing single output space compactor for *C880* reported in [Chakrabarty 95] required 19% area overhead compared with only 7.1% here. Note also that the number of logic levels added to the circuit by the space compactor is very small due to the technique of combining the outputs at the lowest logic levels first.

Table 2 shows results for pseudo-random testing with 10,000 patterns. The compaction ratios were not as high as for deterministic testing as is to be expected. However, the results are still very good with all circuits getting down to 13 or fewer outputs with zero-aliasing.

As can be seen from the CPU times that are shown, the procedure runs very fast initially and then progressively slows down as the compaction ratio gets higher since there are more potentially redundant faults in each iteration. Note that it is easy to tradeoff computation time with the compaction ratio. Since it is a hill climbing procedure, it can be stopped at any time and the space compactor that has been constructed up to that point can be used. Another approach for reducing the computation time for very large circuits would be to partition the outputs of the circuit and run the procedure on each partition separately.

Table 1. Results for Deterministic Testing

Circuit Under Test (CUT)				CUT + Space Compactor						
Name	PI	PO	Levels	PO	Levels	Area Ovrhd (%)	Compaction Ratio	Time (Min.) CR = 2	Time (Min.) CR = 5	Time (Min.) Final CR
C432	36	7	24	2	26	2.7	3.5	< 1	NA	< 1
C499	41	32	19	1	24	10.1	32.0	< 1	2	4
C880	60	26	25	1	26	7.1	26.0	< 1	< 1	< 1
C1355	41	32	25	1	30	6.0	32.0	1	1	1
C1908	33	25	27	3	28	2.9	8.3	1	2	10
C2670	233	140	25	2	26	13.5	70.0	10	10	60
C3540	50	22	38	2	39	1.4	11.0	6	20	30
C5315	178	123	37	4	38	5.4	30.8	30	60	300
C6288	32	32	120	2	120	1.3	16.0	80	200	500
C7552	207	108	28	5	34	3.4	21.6	20	60	600

Table 2. Results for Pseudo-Random Testing with 10,000 Patterns

Circuit Under Test (CUT)				CUT + Space Compactor						
Name	PI	PO	Levels	PO	Levels	Area Ovrhd (%)	Compaction Ratio	Time (Min.) CR = 2	Time (Min.) CR = 5	Time (Min.) Final CR
C432	36	7	24	2	25	2.7	3.5	< 1	NA	< 1
C499	41	32	19	3	23	9.4	10.7	1	2	10
C880	60	26	25	3	26	6.5	8.7	< 1	5	6
C1355	41	32	25	3	29	5.6	10.7	2	5	20
C1908	33	25	27	5	28	2.7	5.0	9	60	60
C2670	233	140	25	2	26	13.5	70.0	10	10	30
C3540	50	22	38	2	39	1.4	11.0	3	10	10
C5315	178	123	37	13	39	5.0	9.5	100	600	1000
C6288	32	32	120	3	121	1.2	10.7	70	200	300
C7552	207	108	28	8	32	3.3	13.5	10	100	500

6. Conclusions

The space compactors synthesized with the procedure presented in this paper offer a number of advantages:

- Zero-Aliasing - No loss of fault coverage in the CUT results from using the space compactor. Moreover, all the faults in the space compactor itself are fully tested.
- Minimal Overhead - An elementary-tree is a very efficient structure for combining outputs. It is fanout-free so the routing is simple.
- High Compaction Ratio - Results indicate that very substantial compression ratios can be achieved.
- Small Propagation Delay - The technique of combining the outputs at the lowest levels first keeps the additional delay added to CUT during testing very small.

As was found in [Ivanov 96], the results here also indicate the power of customizing a space compactor for a particular CUT as opposed to using a CUT independent space compactor such as a parity tree. Not only can loss of fault coverage be prevented, but area and delay overhead can be significantly improved as well. The techniques described in this paper can be automated to provide a push-button solution for designing efficient zero-aliasing space compactors.

Acknowledgements

This material is based on work supported in part by the Defense Advanced Research Projects Agency (DARPA) under Contract No. DABT63-94-C-0045, in part by the National Science Foundation under Grant No. MIP-9702236, and in part by the Texas Advanced Research Program under Grant No. 1997-003658-369.

References

[Brglez 85] Brglez, F., and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan," *Proc. of Int. Symposium on Circuits. and Systems*, pp. 663-698, 1985.

[Chakrabarty 94] Chakrabarty, K., and J.P. Hayes, "Efficient Test Response Compression for Multiple-Output Circuits," *Proc. of Int. Test Conference*, pp. 501-510, 1994.

[Chakrabarty 95] Chakrabarty, K., B.T. Murray, and J.P. Hayes, "Optimal Space Compaction of Test Responses," *Proc. of Int. Test Conference*, pp. 834-843, 1995.

[Chakrabarty 96] Chakrabarty, K., and J.P. Hayes, "Test Response Compaction Using Multiplexed Parity Trees," *IEEE Trans. on Computer-Aided Design*, Vol. 15, No. 11, pp. 1399-1408, Nov. 1996.

[Chickermane 93] Chickermane, V., E.M. Rudnick, P. Banerjee, and J.H. Patel, "Non-Scan Design-For-Testability Techniques for Sequential Circuits," *Proc. of 30th Design Automation Conference*, pp. 236-241, 1993.

[Das 95] Das, S.R., H.T. Ho, W.-B. Jone, and A.R. Nayak, "An Improve Output Compaction Technique for Built-In Self-Test in VLSI Circuits," *Proc. of Int. Conference on VLSI Design*, pp. 403-407, 1995.

[Eichelberger 83] Eichelberger, E.B., and E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test," *IBM Journal of Research and Development*, Vol. 27, No. 3, pp. 265-272, May 1983.

[Fox 77] Fox, J.R., "Test-point Condensation in the Diagnosis of Digital Circuits," *Proc. of the IEE*, Vol. 124, No. 2, Feb. 1977, pp. 89-94.

[Hayes 74] Hayes, J.P., and A.D. Friedman, "Test Point Placement to Simplify Fault Detection," *IEEE Trans. on Computers*, Vol. C-23, No. 7, pp. 727-735, Jul. 1974.

[Immaneni 90] Immaneni, V., and S. Raman, "Direct Access Test Scheme - Design of Block and Core Cells for Embedded ASICS," *Proc. of Int. Test Conference*, pp. 488-492, 1990.

[Ivanov 96] Ivanov, A., B. Tsuji, and Y. Zorian, "Programmable BIST Space Compactors," *IEEE Trans. on Computers*, Vol. 45, No. 12, pp. 1393-1404, Dec. 1996.

[Jone 91] Jone, W.-B., and S.R. Das, "Space Compression Method for Built-In Self-Testing of VLSI Circuits," *Int. Journal of Computer-Aided Design*, Vol. 3, pp. 309-322, Sep. 1991.

[Karpovsky 87] Karpovsky, M., and P. Nagvajara, "Optimal Time and Space Compression of Test Responses for VLSI Devices," *Proc. of Int. Test Conference*, pp. 523-529, 1987.

[Li 87] Li, Y.K., and J.P. Robinson, "Space Compaction Methods with Output Data Modification," *IEEE Trans. on Computer-Aided Design*, Vol. 6, No. 3, pp. 290-294, Mar. 1987.

[Pouya 97] Pouya, B., and N.A. Touba, "Modifying User-Defined Logic to Provide Test Access to Embedded Cores," *Proc. of Int. Test Conference*, pp. 60-68, 1997.

[Reddy 88] Reddy, S.M., K.K. Saluja, and M.G. Karpovsky, "A Data Compression Technique for Built-In Self-Test," *IEEE Trans. on Computers*, Vol. C-37, No. 9, pp. 1151-1156, Sep. 1988.

[Rudnick 94] Rudnick, E., V. Chickermane, J.H. Patel, "An Observability Enhancement Approach for Improved Testability and At-Speed Test," *IEEE Trans. on Computer-Aided Design*, Vol. 13, No. 8, pp. 1051-1056, Aug. 1994.

[Saluja 83] Saluja, K.K., and M. Karpovsky, "Test Compression Hardware Through Data Compression in Space and Time," *Proc. of Int. Test Conference*, pp. 83-88, 1983.

[Touba 97] Touba, N.A., and B. Pouya, "Testing Core-Based Designs Using Partial Isolation Rings," *IEEE Design & Test*, pp. 52-59, Oct. 1997.