Improving test quality for systems on chips (SOCs) requires fully applying the test sequences that core providers typically supply. However, the increasing scale of state-of-the-art SOCs requires designers to embed cores deeply in the chip, making such high-quality tests challenging. Ensuring such high quality requires delivering test data to and from all cores, but making such tests cost-effective requires minimizing test application time. Simultaneous controllability and observability of all a core’s pins minimizes test time but imposes an onerous requirement in terms of test access mechanism (TAM) bus width. This need for a wide bus limits the number of cores that engineers can test in parallel, as does the finite number of chip I/Os. Both of these effects increase test application time—and cost.

Reducing the number of pins for core test requires narrowing the necessary test bandwidth associated with the core’s TAM. A straightforward approach is to serialize core input control (or output observation), enabling the allocation of fewer chip pins for the transportation of serialized test data. However, serial testing prolongs individual core test times, negating the benefit of increased core test parallelism.

Space compaction circuitry, on the other hand, maps the core’s outputs to fewer chip outputs, reducing test bandwidth at the core outputs without increasing test time. Data compression schemes at the core inputs could complement such an approach.1

Here, we focus only on test bandwidth reduction at the outputs, as we address the structure, design, and construction of TAMs for SOCs. Our proposed space and time compaction methodology guarantees a single-bit bandwidth, enabling the test of cores through the allocation of fewer chip pin-outs. In this way, our scheme maximizes parallelism among core tests. (The “Related work” sidebar discusses other work in this area.)

Compaction approaches

Judicious use of time compaction (compressing test responses into a signature), along with space compaction, can optimize bandwidth use. Allocating fewer chip pin-outs during core tests significantly increases the number of cores you can test simultaneously. Figure 1 (on p. ??) shows how three different compaction methodologies affect test application time. Core tests in Figure 1a must be in series, because the chip’s four outputs can map to the...
Related work

Recently, engineers have expended considerable effort testing embedded cores in systems on chips. Researchers have extensively discussed the necessity of a test access mechanism along with its effect on various parameters, such as area, test application time, and test bandwidth.

Researchers have aimed a significant amount of work at test bandwidth reduction on the core output side by designing space compactors based on probabilistic error models unable to guarantee aliasing-free compression. Several techniques implement aliasing-free compaction circuitries. Chakrabarty, Murray, and Hayes built compaction circuitry based on test response values, implementing a combinational network to guarantee that distinct test responses map to distinct space compactor output values. This approach detects any fault effect that yields an invalid test response at the circuit outputs, because such a fault also produces an invalid response at the compactor outputs. This scheme prevents aliasing, but it cannot guarantee a prespecified bandwidth.

Pouya and Touba employed a hill-climbing methodology to build aliasing-free space compaction circuitry. At every step, this method inserts a logic gate to connect a selected pair of primary outputs (POs), provided this insertion doesn’t introduce a functional redundancy. However, the extensive amount of fault simulation for the redundancy checking magnifies this approach’s complexity.

Other schemes used fault sensitization to build the compaction circuitry. Chakrabarty used fault sensitization information to partition the circuit’s POs. His method mapped POs that exhibit equivalent behavior under the complete test set to distinct partitions. A distinct parity tree connects every partition’s outputs, thus guaranteeing the detectability of every fault by at least one parity tree. However, the consequent equivalence between the number of partitions and the number of parity trees implies forgoing the possibility of one-bit space compaction. Furthermore, because the aliasing analysis must consider every single fault in a circuit, this scheme imposes unacceptable complexity requirements.

Chakrabarty and Hayes presented a space compaction mechanism that used multiplexed parity trees (MPT). The test application process involves several phases, each attacking a fault subset. In every phase except the first one, the algorithm disconnects a single PO from the parity tree; in the first phase, no disconnection occurs. This scheme selects the outputs to be disconnected from the parity tree to detect all the faults in the minimum number of phases. However, the approach significantly increases test application time because every phase necessitates applying the entire test set.

Similarly, in previous work, we decomposed the test application process into several phases. Because these phases overlap, the increase in test application time is not as severe as that in the MPT scheme. Performing time compaction along with space compaction allows accumulation of the test responses in several signatures in parallel. This approach simultaneously disconnects several POs from the parity tree at every phase through their superposition with the parity tree output. This scheme selects these POs to detect the largest number of remaining faults in every phase, and the test set breaks down into two subsets, each customized for the target fault subset. Test application includes application of both test subsets, which typically overlap, slightly prolonging test application time.

The methodology we propose here is the only aliasing-free scheme that guarantees an output test bandwidth of a single bit with no prolongation in test application time. The “Experimental results” section includes detailed quantitative comparisons with some of these previously published techniques.

References

outputs of only one core at a time. In Figure 1b, space compaction allows parallel testing of all four cores by enabling observation of one compacted output bit per core. Figure 1c shows the best solution, in which space and time compaction mechanisms allow parallel testing of all four cores through a single bit line.

The use of any compaction circuitry introduces the possibility of aliasing (fault coverage degradation). Test bandwidth reduction, while preserving test quality, requires judicious construction of compaction circuitry to avoid any loss in fault detection. We propose the use of parity trees to capture all the core’s detectable faults. A complete parity tree captures most of the faults; observing the signature in the flip-flops attached to incomplete, supplemental parity trees detects the rest. Continuously observing only the complete parity tree output helps minimize the bandwidth to a single bit while eliminating the vast majority of faults from any further aliasing consideration. Constructing the time-compacted supplemental parity trees involves using computationally efficient heuristics aimed at minimizing area cost.

The scheme we propose ensures that the test remains applicable to sequential cores because it doesn’t modify the test application process and retains the original test sequence. Furthermore, our compaction techniques work with tests for any fault that can be simulated, thus guaranteeing the continued applicability of these tests after compaction. These techniques therefore enable the construction of compaction circuitry to capture the effects of important fault model types, such as bridging and stuck-at faults. This space and time compaction methodology can help designers make SOC tests cost-effective. The guaranteed single-bit output bandwidth maximizes parallelism among core tests because testing each core requires only a single SOC pin-out. Not only does our approach significantly reduce overall SOC test time and preserve overall test quality, but it also eliminates the necessity for a wide TAM bus, offsetting the compaction circuitry’s associated area cost.

Choosing compaction hardware

Implementing the most appropriate compaction circuitry requires assessing the fault effect propagation behavior of the basic logic gates. The lack of a controlling value makes XOR gates useful for building compaction circuitry. Whenever a fault effect appears at only one of an XOR gate’s inputs regardless of the other input values, the fault effect always propagates to the XOR output. The only possible problem with using XOR gates concerns the manifestation of the fault effect in multiple inputs, which could cause masking. As long as a fault manifests itself in an odd number of
XOR inputs, the fault effect propagates through this gate, but the manifestation of a fault in an even number of inputs masks the fault. Because of this propagation behavior, we classify faults into two groups: odd- and even-sensitized. Odd-sensitized faults manifest themselves in an odd number of core outputs for at least one test vector; even-sensitized faults affect an even number of core outputs for every test vector. If space compaction circuitry uses a parity tree consisting solely of XOR gates, aliasing affects only the even-sensitized faults. Because a parity tree captures the effect of all the odd-sensitized faults—the vast majority of faults (typically around 97 to 99 percent)—this approach is computationally efficient. To build aliasing-free compaction circuitry, we need to consider only even-sensitized faults, which are typically few in number.

To capture the effects of even-sensitized faults, we use additional parity trees such that every supplementary parity tree captures a subset of the even-sensitized faults. This scheme detects an even-sensitized fault if it manifests itself in an odd number of inputs in at least one supplementary parity tree. However, although this approach mitigates the aliasing problem, test bandwidth remains wide because the number of outputs to be observed equals the number of parity trees comprising the compaction circuitry.

Circuits typically contain many faults that are sensitized to only a single primary output (PO), and such faults occur for almost every PO. Therefore, the complete parity tree will capture more faults than any incomplete supplementary parity tree. Thus, continuous observation of the complete parity tree’s output will detect the most faults. So despite the test bandwidth consumed by continuous observation, it is worthwhile to do for the complete parity tree.

On the other hand, it would save test bandwidth if we didn’t have to continuously monitor the supplementary parity trees; time compaction can eliminate the need to continuously observe them. Time compaction XORs, over time, the one-bit responses at each supplementary parity tree output through a single flip-flop, as Figure 2 shows. Detecting the remaining faults—the even-sensitized ones—depends entirely on observing the resulting one-bit signature associated with every supplementary parity tree. Capturing fault effects, therefore, is challenging because testers must observe all even-sensitized faults on the basis of the signature at the end of the test session.

Several supplementary parity trees might be necessary to cover all even-sensitized faults. An even-sensitized fault can be detected in a one-bit signature if and only if the test set sensitizes it to the outputs included in the corresponding supplementary parity tree an odd number of times in total. An odd number of core outputs, each of which the test set sensitizes the fault to an odd number of times, should therefore be included in a supplementary parity tree.

One-bit signatures formed at the end of a test session cannot capture some even-sensitized faults regardless of how you form the supplementary parity trees. These even-time faults are detectable only if a tester observes the one-bit signature at some point before the test session’s end. If the fault manifests itself in the signature an odd number of times between the test session’s beginning and this point, observing the one-bit signature will detect this fault.

### Constructing compaction circuitry

Here we present our strategy for cost-effectively generating several supplementary parity trees that capture all even-sensitized faults. Keeping the compaction circuitry’s area cost low requires minimizing the number of supplementary parity trees.

Constructing the supplementary parity

![Figure 2. Circuitry for space and time compaction.](image-url)
trees should rely on the sensitization information for even-sensitized faults. Tables can represent the sensitization information, indicating the correspondence of even-sensitized faults and the outputs to which they are sensitized. Our sensitization tables have rows denoting the even-sensitized faults, and columns denoting the core outputs. Marked entries indicate that the test set sensitizes this fault to the output an odd number of times. Guaranteeing detection of a fault requires that at least one supplementary parity tree include an odd number of outputs with marked entries in the row corresponding to the fault. We solve the problem of constructing such supplementary parity trees by mapping to the well-known problem of sequential, automatic test-pattern generation. We construct the virtual circuit for the sequential-ATPG algorithm on the basis of the sensitization table. The core’s POs constitute the virtual circuit’s primary inputs (PIs), and a single sticky-1 flip-flop corresponds to each even-sensitized fault with a non-null row. Figure 3 shows pseudocode for the generation of this single-output virtual circuit. We must execute the sequential-ATPG algorithm only on the output line’s single stuck-at-0 fault, producing a list of test vectors. Every ATPG-algorithm-generated test vector maps to a distinct supplementary parity tree. The corresponding supplementary parity tree includes the PIs (core outputs) of the virtual circuit that are set to 1 in a test vector. (A don’t-care value implies that the inclusion or exclusion of this core output makes no functional difference in the particular supplementary parity tree; nonetheless, cost-effectiveness reasons argue for its exclusion in this case.)

Figure 4 shows the pseudocode for mapping the ATPG solution to the construction of supplementary parity trees. Because the AND gate’s inputs should all be simultaneously justified to the value of 1 for the detection of the output line stuck-at-0 fault, all the flip-flops should eventually be set to 1. Consequently, the ATPG algorithm must select an odd number of core outputs with marked entries for every even-sensitized fault. The number of ATPG-algorithm-generated test vectors equals the number of supplementary parity trees formed. Breadth-first sequential-ATPG tools, which help minimize the number of test vectors, can help form the minimum number of supplementary parity trees.

The sequential-ATPG problem is computationally more difficult than a nondeterministic-polynomial-complete problem (the combinational ATPG problem). However, the regularity of the virtual circuits on which we execute the sequential-ATPG heuristic offsets the computational overhead. Furthermore, the virtual circuit’s size, which affects ATPG execution time, depends directly on the number of even-sensitized faults to be handled, because a single sticky-1 flip-flop corresponds to each even-sensitized fault. Consequently, the number of even-sensitized faults affects our heuristic’s computational cost. Fortunately, in typical state-of-the-art SOCs, only a few faults (1 to 3 percent) are even-sensitized, so our scheme is computationally efficient.

We constructed the example virtual circuit in Figure 5 (next page) on the basis of the sensitization table in Table 1. Running the sequential-ATPG algorithm on the virtual circuit forms two supplementary parity trees, because the ATPG-algorithm-generated test sequence includes two vectors: 01 1100 and 000001. The first parity tree includes $PO_o$, $PO_0$, and $PO_{00}$ and captures faults $f_2$, $f_3$, $f_6$, and $f_7$. The second one includes only $PO_{00}$ and captures faults $f_2$, $f_4$, $f_6$, and $f_7$. The sequential-ATPG problem is computationally more difficult than a nondeterministic-polynomial-complete problem (the combinational ATPG problem). However, the regularity of the virtual circuits on which we execute the sequential-ATPG heuristic offsets the computational overhead. Furthermore, the virtual circuit’s size, which affects ATPG execution time, depends directly on the number of even-sensitized faults to be handled, because a single sticky-1 flip-flop corresponds to each even-sensitized fault. Consequently, the number of even-sensitized faults affects our heuristic’s computational cost. Fortunately, in typical state-of-the-art SOCs, only a few faults (1 to 3 percent) are even-sensitized, so our scheme is computationally efficient.
Faults \( f_1 \) and \( f_5 \) remain undetected, because their effects do not manifest themselves on any outputs an odd number of times.

Regardless of how we use the algorithm to form the supplementary parity trees, it cannot cover even-time faults, such as \( f_1 \) and \( f_5 \) in Table 1. These faults are sensitized to every PO an even number of times. (All entries in the rows corresponding to even-time faults are unmarked.) Because we cannot detect these faults by observing the signature at the test session’s end, we must identify certain points before the end of the test session to sensitize the even-time faults to at least one core output an odd number of times. We select these points to ensure the detection of the even-time faults exactly once. Hence, the fault is sensitized to several outputs exactly once, enabling the formation of the supplementary parity trees that capture that fault’s effect.

To identify the appropriate signature observation points, we introduce another structure, a vector-fault correspondence table. The table’s rows denote the even-time faults, and the columns denote the test vectors. A marked entry means the test vector detects the fault. Using our ATPG heuristic, we can form supplementary parity trees, after the selection of the minimum number of observation points that cover all the even-time faults.

In the previous example, the rows of the sensitization table in Table 1 corresponding to even-time faults \( f_1 \) and \( f_5 \) are completely blank. Examining the vector-fault correspondence table in Table 2 shows that the test set detected these two even-time faults exactly once, right after the application of the first two test vectors. As long as the tester observes the signature right after the application of the first two vectors, a supplementary parity tree that guarantees the detection of these faults is formed. Based on the sensitization table in Table 3, we can construct the additional supplementary parity tree by solving yet another sequential-ATPG problem. A parity tree comprising only \( PO_4 \) captures both even-time faults, provided the tester observes the signature after the application of the second test vector. Figure 6 (next page) shows the complete compaction circuitry.

**Results with random circuits**

To assess our scheme’s viability on circuits approaching the size of current SOCs, we con-
ducted experiments on large, randomly generated circuits. Using a probabilistic model, we generated sensitization tables for circuits with many outputs. We then built compaction circuits based on these sensitization tables. We compared our approach with other approaches, including an independent-set selection heuristic for compaction circuitry construction.

We set the number of POs for the randomly generated circuits to 1,000 and set the number of faults to 10,000. We assumed that the test set size for these circuits was 500. Several parameters further differentiated the generated circuits—for example, fault detection probability and the number of outputs to which a detected fault was sensitized. We used a constant parameter for the detection probability, and a Gaussian distribution for the number of POs to which the fault was sensitized. After generating the number of such POs, we randomly selected the POs that we treated as if the fault had manifested itself to them. Adjusting the detection probability and the Gaussian distribution's variance produced random circuits of varying sensitization characteristics.

Table 4 illustrates the number of even-sensitized faults in these circuits. These numbers strongly affect the computation time requirements because they directly impact the aliasing analysis duration. The number of even-sensitized faults is inversely proportional to the probability of fault detection: The consequent increase in the number of test vectors detecting a fault increases the probability that at least one of the vectors is odd sensitizing. Variance adjustments, on the other hand, seem to have a negligible effect on the number of even-sensitized faults. Adjusting the distribution of the number of POs that a fault is sensitized to keeps intact the even-sensitization probability, given a fixed detection probability. It is interesting to note in Table 4 that a detection probability of 0.05 yielded no even-sensitized faults in almost all cases, and a 0.005 detection probability generated even-sensitized faults that constituted more than 20 percent of the faults for all cases.

Both our earlier heuristic and this new approach aimed at reducing area overhead by generating a minimum number of supplementary parity trees. When forming a supplementary parity tree, our earlier heuristic selected POs corresponding to the disjoint columns of the sensitization table to greedily cover the maximum number of faults. Using the heuristic we formed several supplementary parity trees, eventually covering all even-sensitized faults. The disjoint column selection in our earlier approach guaranteed that every fault covered was sensitized to only one input of a supplementary parity tree an odd number of times. This was a sufficient, but unnecessary, condition. The number of supplementary parity tree inputs where the fault manifested itself an odd number of times was restricted to one, whereas the necessary condition imposes the existence of an odd number of such inputs in a supplementary parity tree. The incomplete exploration of the search space often resulted in shorter run times, but might provide nonoptimal solutions. Table 5 compares the two approaches in terms of execution time and area overhead of the supplementary parity trees; we used six circuits to run the two heuristics. Even though...

![Figure 6. Complete compaction circuitry.](image-url)
The independent-set selection heuristic ran faster, it consistently generated more supplementary parity trees than the ATPG heuristic. Designers can further explore the tradeoff between execution time and area overhead for individual circuits, depending on the circuit’s size and fault sensitization characteristics.

Reducing area overhead by restructuring the parity tree

We can construct various parity trees for the same functionality; the gate cost is insensitive to the design of parity trees. Figure 7 shows two structurally different parity trees of five XOR gates each; these trees are functionally identical. On the basis of this observation, we can design the complete parity tree to use its internal lines when forming the supplementary parity trees. Consequently, we can save the area cost for a portion of the XOR gates that would otherwise be needed for the supplementary parity trees. For example, we could restructure the parity tree in Figure 6 to the design in Figure 8, reducing area.

The strategy pursued in implementing the parity trees exploits the common core outputs included in several parity trees. Rather than repeatedly sending these core outputs through an XOR gate in several parity trees in distinct sequences, the compaction circuitry sends them through an XOR gate once and uses the XOR output in implementing the parity trees.
thus saving considerable area. A table that maps the parity trees to the core outputs for inclusion in these trees helps properly construct the complete parity tree. The table’s rows denote the parity trees, and its columns denote the core outputs. A marked entry means the parity tree includes the corresponding core output. In this table, the first row corresponds to the complete parity tree, which includes the complete set of core outputs. The heuristic we propose for restructuring the complete parity tree is an iterative algorithm. At every step, the algorithm includes the core output pair in the largest number of parity trees selected. In the complete parity tree, the algorithm sends these two core outputs through an XOR gate. Subsequently, the algorithm adds an additional column to the table to represent the XOR of the two core outputs. It also updates the rows of the parity trees that include both core outputs by marking the new column and erasing the marks in the columns corresponding to the two individual core outputs. The algorithm terminates when it constructs all the parity trees; at this point, a single mark remains in each row.

The example in Figure 9 shows how the algorithm restructures a complete parity tree. To save several XOR gates in implementing the four supplementary parity trees indicated in Figure 9a, the algorithm manipulates the parity tree table to the representation in Figure 9b. The restructured complete parity tree and the supplementary parity trees indicated in Figure 9b reduce the number of XOR gates from 12 to five.

Experimental results

We applied the proposed space and time compaction scheme to the 1985 IEEE International Symposium on Circuits and Systems benchmark suite. Atalanta, an automatic test-generation tool, generated the test sets for these circuits.

Table 6 shows the number of bits that the tester must observe and the area cost associated with using our proposed scheme for the given circuits. (The even-sensitized faults typically constitute less than 3 percent of the
faults for nine out of 10 benchmark circuits.) The reported area overhead is the percentage increase in the number of gates. The observed bits must include the one-bit responses of the complete parity tree and the signature content. For circuit c1908, the test set includes 116 patterns, and the signature length is 2 bits; the number of one-bit parity tree responses is 116, whereas the number of bits observed is 118. This circuit must use two supplementary parity trees because every signature bit corresponds to a distinct supplementary parity tree. The circuit with the largest even-sensitized faults ratio, c2670, requires five supplementary parity trees. The existence of more even-sensitized faults leads to more supplementary parity trees for capturing them.

Table 7 compares the achievable test bandwidth of three earlier approaches for building aliasing-free compaction circuits. Unlike our new methodology, which guarantees a one-bit bandwidth, these approaches cannot provide a prespecified bandwidth.

Table 7 compares the test application time and area overhead of the proposed methodologies to those of two schemes that guarantee the minimum possible test bandwidth. Although the area overheads of the multiplexed-parity-trees (MPT) approach are comparable to those of our approach, multiple applications of the entire test set lengthen the test application times. For circuits c2670, c3540, and c7552, the test application times associated with our methodology are an order of magnitude less than those for MPT.
Because the independent-set heuristic partitions the test set into two possibly overlapping subsets, the resulting test application time generally exceeds the time it takes to apply the original test set; experimental data confirms the expected increase in test application times for this scheme.

To enable cost-effective test of SOC designs, it is necessary to test cores in parallel. Our proposed space and time compaction methodology maximizes parallelism among core tests by guaranteeing the minimum test bandwidth of a single bit, thus enabling the test of cores through the allocation of fewer chip pin-outs.

### Table 8. Comparison of test application time and area overhead for our methodology against MPT and the independent-set heuristic.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Proposed scheme</th>
<th>Multiplexed parity trees</th>
<th>Independent-set heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Test application time (s)</td>
<td>Area (percentage)</td>
<td>Test application time (s)</td>
</tr>
<tr>
<td>c432</td>
<td>48</td>
<td>4.17</td>
<td>183</td>
</tr>
<tr>
<td>c499</td>
<td>54</td>
<td>16.82</td>
<td>126</td>
</tr>
<tr>
<td>c880</td>
<td>55</td>
<td>6.97</td>
<td>66</td>
</tr>
<tr>
<td>c1355</td>
<td>86</td>
<td>5.46</td>
<td>522</td>
</tr>
<tr>
<td>c1908</td>
<td>116</td>
<td>3.21</td>
<td>508</td>
</tr>
<tr>
<td>c2670</td>
<td>108</td>
<td>10.37</td>
<td>1,143</td>
</tr>
<tr>
<td>c3540</td>
<td>148</td>
<td>1.54</td>
<td>1,566</td>
</tr>
<tr>
<td>c5315</td>
<td>119</td>
<td>7.87</td>
<td>274</td>
</tr>
<tr>
<td>c6288</td>
<td>28</td>
<td>1.25</td>
<td>200</td>
</tr>
<tr>
<td>c7552</td>
<td>211</td>
<td>3.81</td>
<td>2,124</td>
</tr>
</tbody>
</table>

References


Ozgur Sinanoglu is a PhD student in the Computer Science and Engineering Department at the University of California, San Diego. His research interests include VLSI test, digital signal processors, and computer architecture. Sinanoglu has a BS in electrical and electronics engineering and a BS in computer engineering, both from Bogazici University, Turkey, and an MS in computer engineering from the University of California, San Diego. He is a member of the IEEE.
Alex Orailoglu is a professor in the Computer Science and Engineering Department at the University of California, San Diego. His research interests include digital and analog test, fault-tolerant computing, computer-aided design, and embedded processors. Orailoglu has an SB in applied mathematics from Harvard University, and an MS and PhD in computer science from the University of Illinois, Urbana-Champaign. He is a member of the IEEE.

Direct questions and comments about this article to Ozgur Sinanoglu, Computer Science and Engineering Dept., Univ. of California, San Diego, La Jolla, CA 92093-0114; ozgursin@yahoo.com.

For further information on this or any other computing topic, visit our Digital Libary at http://computer.org/publications/dlib.