Using Partial Isolation Rings to Test Core-Based Designs

NUR A. TOUBA
BAHRAM POUYA
University of Texas

A partial isolation ring provides the same fault coverage as a full isolation ring but avoids adding multiplexers on critical timing paths and reduces area overhead. The authors examine several partial isolation ring selection strategies that vary in computational complexity.

CORE-BASED DESIGNS pose a significant test challenge. Vendors of intellectual property cores usually give no information about a core's internal logic (in other words, it is a black box). As a result, system designers cannot perform traditional test generation processes such as automatic test pattern generation (ATPG) and fault simulation. Instead, the core vendor specifies a set of test vectors that must be applied to the core to guarantee a certain fault coverage. The problem is how to apply the specified test vectors to the core and how to test the logic surrounding the core.

One simple approach for testing embedded cores is to use multiplexing to make the core's inputs and outputs accessible to the chip pins.[1] But this approach does not help with testing the logic surrounding the core and thus results in degraded fault coverage. Another approach is to place an isolation ring around the core, as illustrated in Figure 1. An isolation ring is essentially a boundary scan that provides full controllability of the core's inputs and full observability of the core's outputs. It also provides full observability of the logic driving the core and full controllability of the logic driven by the core. The drawback of a full isolation ring is the large area and performance overhead it adds. It requires a boundary-scan element and associated routing for each input and output of the core and a multiplexer delay for every path to and from the core. As a result, a full isolation ring may not be an acceptable solution in many high-performance applications. Moreover, for a compact core with a large number of I/Os, adding a full isolation ring can more than double the area.

We have developed a design-for-testability (DFT) method that reduces the area and performance overhead of using an isolation ring. This systematic procedure for designing a partial isolation ring provides the same fault coverage as a full isolation ring but avoids adding muxes on critical timing paths. Treating the core as a black box, the procedure assumes no information about the core other than the set of test vectors specified by the core supplier. The procedure uses ATPG techniques to analyze the user-defined logic (UDL) surrounding the core. This analysis identifies a maximal set of core inputs and outputs (including the critical timing paths) that need not be included in the partial isolation ring. If one core is driving another core, the procedure identifies a maximal set of isolation ring elements that can be removed from the interface between the cores.

Selecting core inputs

To start, we focus on reducing the number of isolation ring elements at the core in-

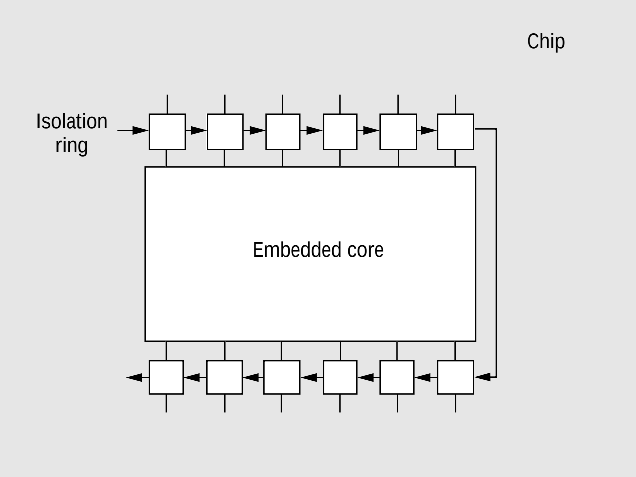


Figure 1. Isolation ring for testing an embedded core.

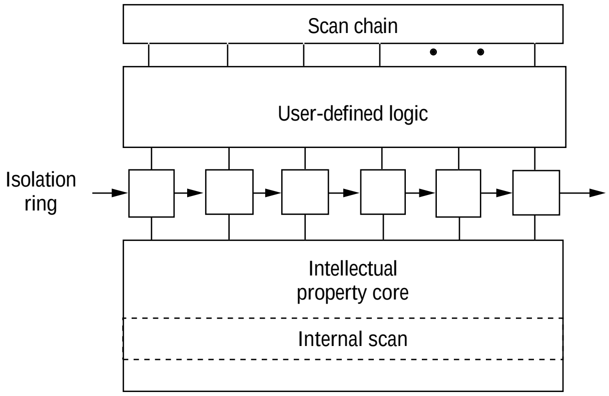


Figure 2. Architecture for testing the core and its driving UDL.

puts. The isolation ring elements at the core inputs serve two purposes. First, they provide controllability of the core inputs. We can shift the core test vectors into the isolation ring and apply them directly to the core. The core may have an internal scan chain for controlling internal flip-flops. In that case, in addition to shifting a test vector into the isolation ring, we would also shift a test vector into the internal scan chain. The second purpose of the isolation ring elements at the core inputs is to provide observability of the UDL driving the core. Figure 2 illustrates an isolation ring architecture using a scan methodology. Testing the UDL involves shifting a test pattern into the scan chain and loading the response into the isolation ring.

For a partial isolation ring, we partition the core inputs into two sets: IR, the set of core inputs included in the ring; and NIR, the set not included. For the NIR core inputs, we need an alternate means of applying the specified core test vectors and of observing the outputs of the UDL that drives them. Our method of applying the test vectors to the NIR core inputs is to justify the vectors through the UDL that drives them. To observe the outputs of that UDL, we perform space compaction by exclusive-ORing each output driving an NIR core input with an output driving an IR core input. Then we feed the combined output into the partial isolation ring.

Selection procedure. The output space of the UDL driving the core may not contain all the core test vectors specified by the core supplier. This does not mean that faults in the core are necessarily redundant. A large set of test vectors may exist for a fault in the core, but the particular test vector specified by the core supplier may happen to be one that is not contained in the UDL's output space. Because other test vectors for the fault may exist in the UDL's output space, the fault may not be redundant. But we cannot identify those test vectors without knowledge of the core's internal logic. Thus, the problem is to select a set of core inputs that will enable us to apply all the specified test vectors to the core. We have devised a selection procedure that minimizes the number of core inputs included in the partial isolation ring.

The first step is to see which of the specified test vectors can be justified through the driving UDL and which cannot. To check each vector, we append an AND gate to the UDL outputs (each UDL output is an AND gate input). Next we add inverters on AND gate inputs corresponding to each 0 bit in the vector. We then use an ATPG tool to target a stuck-at-0 fault at the AND gate output. If the fault is detectable, we know it is possible to justify the specified test vector through the UDL to the core inputs. The test vectors that cannot be justified through the UDL are the ones we must consider in designing the partial isolation ring. If all the test vectors can be justified through the UDL, no isolation ring elements are needed at the core inputs.

If there are n core inputs, there are 2^n possible partial isolation rings, because each core input can be either included or not included in the ring. A particular ring enables a test vector to be applied if the subset of bits in the test vector corresponding to excluded core inputs can be justified through the UDL. A ring is a solution if it enables all the specified test vectors to be applied to the core. We determine whether a ring is a solution by appending AND gates (AND gate inputs correspond to excluded core inputs) to the UDL outputs and performing ATPG as described earlier.

An exhaustive approach for selecting a partial isolation ring would be to check all 2^n possibilities to see which are solutions. Then one would choose the solution containing the fewest isolation ring elements and including none of the core inputs on critical timing paths. However, if the number of core inputs is large (that is, if n is large), this approach is not computationally feasible. Thus, we need some other strategies for searching the exponential space of possible

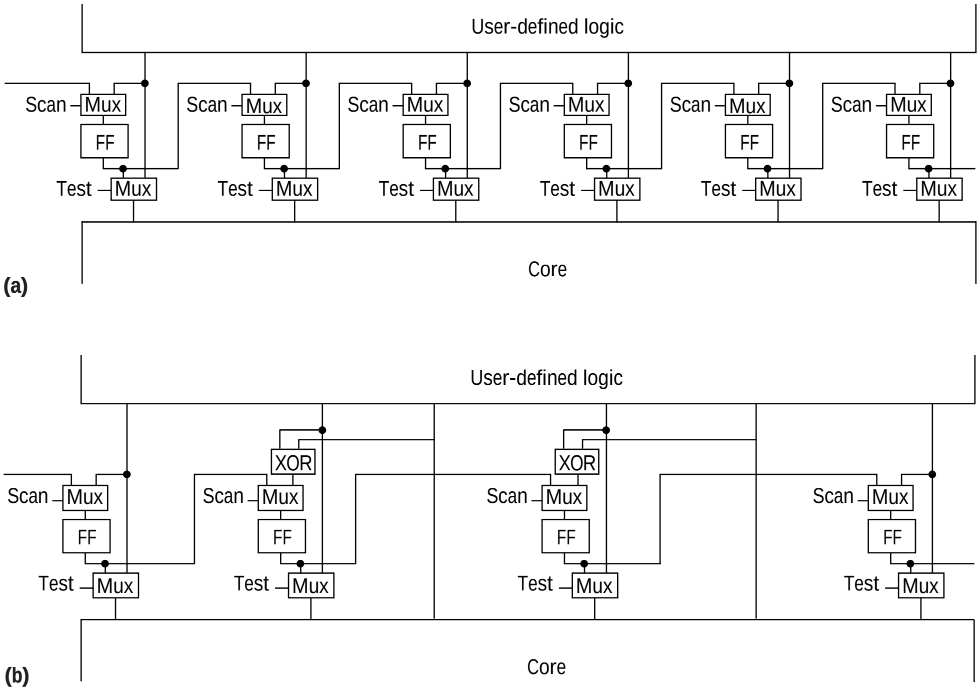


Figure 3. *Full isolation ring* ***(a)*** *and partial isolation ring* ***(b)****.*

partial isolation rings. The following paragraphs describe several strategies, varying in computational complexity. The first step is always to remove core inputs on critical timing paths from the isolation ring, since they are the ones that impact performance. After that step, the remaining task is to remove as many additional core inputs as possible.

Hill climbing—O(n). This procedure removes each core input from the isolation ring one at a time. After each removal, it checks whether the resulting partial isolation ring is still a solution. If not, it adds the core input back to the ring. If the ring is a solution, the core input remains off the ring. After the procedure loops through all *n* core inputs, it stops and the resulting partial isolation ring is the solution. This search strategy is not very clever, but it is very fast.

Clique hill climbing—O(n^2). Let *A*, *B*, and *C* be core inputs, and let *Ring* – *{A,B}* be the partial isolation ring that results from removing the set of core inputs *A* and *B*. If *Ring* – *{A,B}*, *Ring* – *{B,C}*, and *Ring* – *{A,C}* are all solutions, that does not imply that *Ring* – *{A,B,C}* is a solution. Consider the case where 111 is a core test vector, and vectors 110, 011, and 101 can all be justified through the UDL. Removing any two core inputs from the partial isolation ring is a solution, but removing all three is not. If *Ring* – *{A,B}* is not a solution, however, that does imply that *Ring* – *{A,B,C}* is not a solution. In other words, for *Ring* – *{A,B,C}* to be a solution, it is necessary but not sufficient that *Ring* – *{A,B}*, *Ring* – *{B,C}*, and *Ring* – *{A,C}* are all solutions.

From this fact, we can compute a bound on the total number of core inputs that can be removed from an isolation ring. We do this by forming a compatibility graph in which each node corresponds to a core input. We place an edge between two nodes if the ring that results from removing both corresponding core inputs is a solution. The largest clique (complete subgraph) in the compatibility graph corresponds to the largest set of core inputs that potentially can be removed from the isolation ring.

To search for the best partial isolation ring, we use the compatibility graph to guide the order in which hill climbing is performed. The best possible solution obtainable by removing a particular core input from the isolation ring is bound by the size of the largest clique the core input is in. Therefore, we guide the hill-climbing procedure so that it first considers the core inputs contained in the largest cliques, since they are most likely to be in the best solution. We identify the largest cliques, and order the core inputs for the hill-climbing procedure. Identifying the largest cliques in a graph is an NP-complete problem, but good heuristics exist for it.

Clique greedy—O(n^3). For this strategy, we construct a compatibility graph as previously described. We identify the largest clique. Then we remove from the isolation ring the core input corresponding to the node that is in the largest clique and has the largest number of edges. We repeat this procedure recursively for the resulting partial isolation ring. Each time we remove a core input from the ring, we add more constraints for further removing core inputs. As a result, we remove edges from the compatibility graph. This repeated updating of the compatibility graph provides more accurate information to guide the search.

Branch and bound—O(2^n). Since the largest clique in the compatibility graph provides a bound on the best possible solution, we can use this information to avoid unproductive searching. When we are exploring a branch of the search tree, the largest clique in the graph places a bound on the best possible solution in the branch. If no solution in the branch can be better than the best ring solution found so far, we can cut off the branch. In the worst case, this procedure is exponential, but generally it greatly reduces the search space. We can run the branch-and-bound procedure as long as we wish. Whenever it stops, it will provide the best solution it has found so far. If it is allowed to run to completion, it is guaranteed to find the optimum solution.

Observing the UDL. Our next task is to design a space compactor that will combine the outputs of the UDL driving the core, making their test response observable in the partial isolation ring. If n is the number of outputs and k is the size of the partial isolation ring, we can use a space compactor with $n - k$ exclusive-OR gates. By using the techniques described by Chakrabarty and Hayes,[2] we can ensure that the UDL can be tested for single stuck-at faults without any aliasing in the space compactor. These techniques involve either modifying the test set or making minor circuit modifications that sensitize each fault to an odd number of outputs. As a result, the fault's effect will not be masked in the exclusive-OR gates but rather will be captured in the partial isolation ring.

Figure 3 illustrates the results of the core input selection and space compaction techniques we have described. Figure 3a shows an example of a full isolation ring. Figure 3b shows the same ring with the third and fifth core inputs removed and exclusive-OR gates compacting the outputs of the combinational logic driving those inputs.

Selecting core outputs

Now, our focus shifts to the outputs of the core. The isolation ring elements at the core outputs serve two purposes. They provide observability of the core outputs and controllability of the inputs of the UDL driven by those outputs. If it were possible to justify a sufficient set of test vectors for the core-driven UDL through the core itself, we would need only a shift register for observing the core outputs (or we could multiplex the outputs to chip pins). A shift register is much better than an isolation ring because a shift register does not add any logic on the system paths. An isolation ring, on the other hand, adds a mux delay on every path and requires that a test mode line be routed to control the muxes. The problem is that if the core contains sequential logic, placing the core in the states needed to justify test vectors for the UDL at the core outputs may be difficult. The core may have an internal scan path, but that won't help if we know nothing about the core's internal logic.

A full isolation ring solves this problem by enabling the test vectors to be shifted in and directly applied to the UDL.

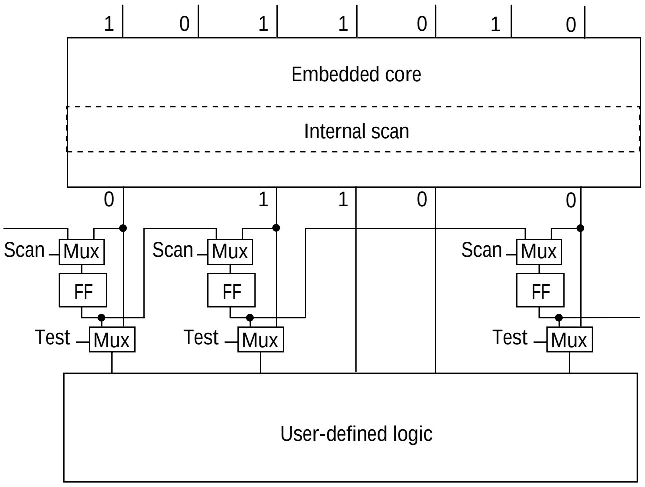


Figure 4. Using a partial isolation ring at core outputs.

The procedure described here reduces the area and performance penalty of a full isolation ring by replacing some ring elements with shift register elements (or multiplexing ring elements to chip outputs). The idea is to justify a subset of the bits of each test vector through the core and use a partial isolation ring to shift in the rest of the bits. Figure 4 shows an example in which we generate each test vector by shifting the first, second, and fifth bits into a partial isolation ring and justifying the third and fourth bits through the core. The goal is to avoid adding isolation ring elements on the critical timing paths.

The set of core outputs that we need not include in the isolation ring depends on what vectors can be justified at the core outputs. One set of vectors easily justified at the core outputs is the output response of the test vectors specified by the core supplier. Since we will test the core with these test vectors, their output response will be generated at the core outputs. Thus, we can use them in testing the UDL driven by the core outputs. Consider the example in Figure 4. One of the supplier-specified test vectors is 1011010, and the corresponding core output vector is 01100. We apply test vector 1011010 to the core inputs. If the core has an internal scan path, the appropriate specified scan vector also shifts into the core's internal scan path. The corresponding core output vector 01100 is now justified at the core outputs, so we can use it for testing the UDL driven by the core outputs. Moreover, we can apply any test vector that has 10 in the third and fourth bit positions (having the form XX10X) to the driven UDL by shifting the bits into the ring.

Based on the set of easily justified vectors, we can design a partial isolation ring that enables a sufficient set of test vectors to be applied to the driven UDL. Depending on the core's functionality, it may be possible to identify a greater number of easily justified vectors than just the output response of the supplier-specified test vectors.

Checking core output fault coverage. Let us refer to the set of vectors easily justified at the core outputs as the "core output vectors." Given the core output vectors, we must determine which core outputs we can remove from the isolation ring without reducing the UDL's fault coverage. Let *FRR* be the set of faults that require an isolation ring in order to be detected. *FRR* equals all the faults in the UDL minus redundant faults (undetectable by any set of test vectors) and minus faults detectable by the core output vectors. Computing the set of faults in *FRR* requires a redundancy check on the UDL and fault simulation for the core output vectors.

Removing a set of core outputs from the isolation ring may restrict the vectors that can be justified at those outputs. Again consider the example in Figure 4. If justifying a 11 on the third and fourth core outputs is not possible, no test vectors of the form XX11X can be applied to the UDL. The question then is whether or not all the faults in FRR can be detected without using any test vectors of the form XX11X. We determine this by performing ATPG with constraints on the allowable values of a test vector's bits. These constrained ATPG techniques[3-5] consider the constraints as early as possible in the ATPG decision-making process. We use these techniques to check whether the faults in FRR are detectable under the constraints imposed by a particular partial isolation ring. If some cannot be detected because of the restricted set of vectors that can be applied by that ring, it will degrade fault coverage and thus is not a solution.

Given the set of excluded core outputs, we determine the ATPG constraints by analyzing the core output vectors. Let's say, for example, that the second, third, and fourth core outputs are not included in the partial isolation ring and the core output vectors are 110101, 100101, 011010, and 011101. Therefore, the ATPG constraints are that all test vectors must have the form X101XX, X001XX, X110XX, or X111XX. We can simplify this set of allowable test cubes to XX01XX and X11XXX by combining adjacent cubes. The procedure for checking that a particular partial isolation ring is a solution is as follows:

1. Set bit positions in the core output vectors corresponding to core outputs included in the partial isolation ring to don't cares (X's) to form the set of allowable test cubes.
2. Use a two-level minimizer to minimize the set of allowable test cubes.
3. Perform ATPG for the faults in FRR under the constraint that test vectors must be contained in the allowable test cubes.

If all the faults in FRR are detectable under these constraints, the partial isolation ring is a solution.

An obvious concern about this procedure is the amount of ATPG it requires. There are several approaches to reducing the amount. One very effective technique is to perform fault simulation for some random patterns to reduce the number of faults needing ATPG. We generate the random patterns by randomly specifying the unspecified bit positions in allowable test cubes. This ensures that the random patterns are contained in the allowable test cubes. Another effective technique is to record each test vector found for each fault. Since we target the same faults each time we consider a partial isolation ring, we can check whether one of the previously identified test vectors for the fault satisfies the current constraints. If so, ATPG is not necessary. These two techniques can dramatically reduce the amount of ATPG required. Note that not all faults are considered—only those in FRR. Moreover, as soon as one fault is found untestable under the constraints, we classify the partial isolation ring as not being a solution. Thus, almost all the ATPG is targeting testable faults. Time-consuming ATPG for untestable faults takes place no more than once per partial isolation ring.

Selection procedure. The procedure for selecting core outputs to remove from the partial isolation ring is almost exactly the same as that for core inputs. The only difference is the method of determining whether the resulting partial isolation ring is a solution. Thus, we can use the four search strategies described earlier.

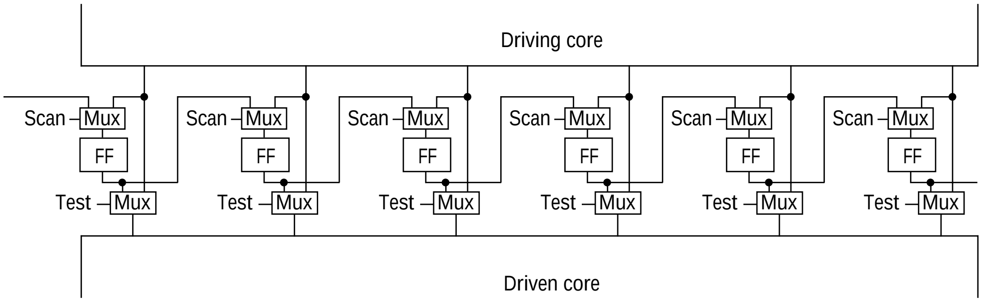


***Figure 5.** Example of full isolation ring at interface between cores.*

Selecting ring elements at core interfaces

So far we have assumed that only user-defined logic surrounds the core. Now we must extend our techniques to handle cases in which

Table 1. *Results for partial isolation rings at core inputs.*

	Core			Hill climbing $O(n)$		Clique hill climbing $O(n^2)$		Clique greedy $O(n^3)$		Branch and bound $O(2^n)$	
UDL name	**Name**	**Test vectors**	**Full ring size**	**Ring size**	**Time (min.)**	**Ring size**	**Time (min.)**	**Ring size**	**Time (min.)**	**Ring size**	**Time (min.)**
vda	s838	185	34	31	1	30	1	29	4	29	28
k2	s9234	285	36	33	1	31	3	31	9	31	70
apex7	C499	68	41	9	1	7	2	6	31	6	200
x4	s13207	831	62	9	6	9	15	NA	NA	NA	NA
apex6	s15850	738	77	8	11	8	16	NA	NA	NA	NA
C2670	dsip	63	140	29	15	28	36	NA	NA	NA	NA
C5315-a	C5315-b	71	117	74	12	73	30	NA	NA	NA	NA
C7552-a	C7552-b	92	261	69	10	63	43	NA	NA	NA	NA

one core is driving some or all the inputs of another core. With a full isolation ring, we place ring elements on all the connections in the interface between cores, as shown in Figure 5. The issue is which of these elements can be removed without losing fault coverage. As described earlier, the set of test vectors that can be justified at the driving core's output may be limited. If the driving core contains sequential logic, placing that core in the states needed to justify the test vectors for the core being driven may be very difficult. One set of vectors easily justified at the driving core's outputs is the output response of the supplier-specified test vectors. Since we will test the driving core with the specified test vectors, we can use their output response, generated at the core outputs, in testing the driven core. It may be possible to justify other vectors at the output of the driving core, depending on its functionality.

We use the same procedure for this case as that described for selecting a partial isolation ring at a core's inputs, with one difference: For the previous case, we used ATPG to determine whether a partial vector can be justified at the driven core's inputs. For this case, we check to see if that vector is contained in one of the core output vectors that can be justified at the driving core's outputs. So determining whether a partial isolation ring at the core interface is a solution is a simple procedure: It involves checking whether all specified test vectors on connections not containing ring elements are contained in the set of vectors justifiable at the driving core's outputs. Thus, we can use all the search strategies described earlier.

Experimental results

We used the procedures described here to select partial isolation rings for some designs constructed from the MCNC (Microelectronics Center of North Carolina) benchmark circuits. For designs in which UDL surrounded a core, we treated one benchmark circuit as an intellectual property core and thus as a black box; we treated another benchmark circuit as the UDL at the core's input or output. For designs with two interconnected cores, we used two benchmark circuits as cores treated as black boxes. We partitioned two benchmark circuits, C5315 and C7552, into two parts, one considered a core and the other either UDL or another core.

Selecting at core inputs. Table 1 shows the results for experiments in selecting partial isolation rings at a core's inputs. For each design, the table shows the name of the UDL driving the core, followed by the core's name and the number of specified test vectors for the core. (We obtained the test vectors through ATPG on the core.) Next, the table shows the number of isolation ring elements in a full isolation ring, followed by results for each of the four search strategies described earlier. For each strategy, the table shows the number of isolation ring elements in the selected partial isolation ring, along with the CPU time (CPU times would be much smaller if we had used an industrial-quality ATPG tool). NA indicates that a procedure ran for more than five hours.

The procedures ran on a Sun UltraSparc. As the table shows, we could use the clique-greedy and branch-and-bound search strategies only when the number of core inputs (n) was small (less than 50). The branch-and-bound strategy is guaranteed to find the optimum solution. Therefore, where all four search strategies were used, the results indicate that the clique hill-climbing strategy found something very close to the optimum solution. In some cases, the simple hill-climbing strategy performed poorly because early in the procedure it selected a core input that was incompatible with many other core inputs.

Selecting at core outputs. Table 2 shows our results for experiments in selecting partial isolation rings at the outputs

Table 2. *Results for partial isolation rings at core outputs.*

Core						Hill climbing $O(n)$		Clique hill climbing $O(n^2)$	
Name	Outputs	Output vectors	UDL name	FRR faults	Full ring size	Ring size	Time (min.)	Ring size	Time (min.)
mm30a	30	210	x1	128	30	11	1	8	6
s9234	39	285	C880	56	39	7	2	7	12
s5378	49	402	apex6	54	49	7	19	7	140
sbc	56	229	i5	87	56	25	12	19	16
s15850	150	738	i4	79	150	28	22	25	180
dsip	197	63	i7	21	197	4	10	4	710
C5315-a	117	34	C5315-b	30	117	12	13	11	200
C7552-a	261	63	C7552-b	80	261	12	25	12	230

of a core. For each design, it shows the core's name, number of outputs in the core, and number of core output vectors. Then it shows the name of the UDL driven by the core and the number of faults that require an isolation ring in order to be detected (faults in FRR). Finally, for two search strategies, the table shows the number of isolation ring elements in the selected partial isolation ring and CPU time.

Selecting at core interface. Table 3 shows results for experiments in selecting partial isolation rings at the interface between two cores. For each design, we show the driving core's name, the number of output vectors generated during testing, the name of the core being driven, and the number of specified test vectors for the driven core. We show the number of isolation ring elements in a full isolation ring and results for each search strategy.

In general, the proportion of removable elements in a partial isolation ring to a full isolation ring is not as high at core interfaces as in designs whose UDL drives the core. This result is due to our conservative assumption that only the output response of the specified test vectors for the driving core can be justified at the core interface. Again, depending on the driving core's functionality, it may be possible to justify other vectors and thus reduce the number of isolation ring elements needed at the interface.

OUR RESULTS INDICATE that partial isolation rings provide a means for significantly reducing DFT overhead in core-based designs (with no loss of fault coverage). They are an effective alternative to full isolation rings for supporting the use of highly optimized cores in timing-critical and area-limited applications.

Acknowledgments

This work was supported in part by the Advanced Research Projects Agency under prime contract DABT63-94-C-0045, and in part by the National Science Foundation under grant MIP-9702236.

References

1. V. Immaneni and S. Raman, "Direct Access Test Scheme—Design of Block and Core Cells for Embedded ASICs," *Proc. Int'l Test Conf.*, IEEE Computer Society Press, Los Alamitos, Calif., 1990, pp. 488-492.
2. K. Chakrabarty and J.P. Hayes, "Efficient Test Response Com-

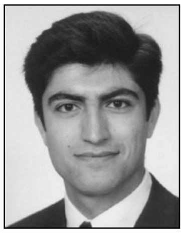
Table 3. *Results for partial isolation rings at core interfaces.*

Driving core		Driven core			Hill climbing $O(n)$		Clique hill climbing $O(n^2)$		Clique greedy $O(n^3)$		Branch and bound $O(2^n)$	
Name	Output vectors	Name	Test vectors	Full ring size	Ring size	Time (min.)	Ring size	Time (min.)	Ring size	Time (min.)	Ring size	Time (min.)
s5378	402	mm30a	210	33	28	1	28	1	26	1	26	136
s9234	285	s838	185	34	29	1	28	1	27	1	27	268
s15850	738	sbc	110	40	34	1	34	2	33	3	NA	NA
C2670	193	s13207	831	62	58	1	58	7	57	13	NA	NA
dsip	63	s15850	738	77	73	1	73	13	72	25	NA	NA
C5315-a	34	C5315-b	71	117	115	1	115	2	113	10	NA	NA
s38584	1294	des	110	256	248	1	248	198	NA	NA	NA	NA
C7552-a	63	C7552-b	92	261	257	1	257	52	NA	NA	NA	NA

pression for Multiple-Output Circuits," *Proc. Int'l Test Conf.*, IEEE CS Press, 1994, pp. 501-510.

3. M.H. Konijnenburg, J.T. van der Linden, and A.J. van de Goor, "Test Pattern Generation with Restrictors," *Proc. Int'l Test Conf.*, IEEE CS Press, 1993, pp. 598-605.
4. M.H. Konijnenburg, J.T. van der Linden, and A.J. van de Goor, "Automatic Test Pattern Generation for Industrial Circuits with Restrictors," *Microelectronics J.*, Vol. 26, No. 7, Oct. 1995, pp. 598-605.
5. P. Wohl, and J. Waicukauski, "Test Generation for Ultra-Large Circuit Using ATPG Constraints and Test-Pattern Templates," *Proc. Int'l Test Conf.*, IEEE CS Press, 1996, pp. 13-20.

Nur A. Touba is an assistant professor in the Department of Electrical and Computer Engineering at the University of Texas at Austin. His research interests are in automated design of testable and fault-tolerant circuits. He received a National Science Foundation Career Award in 1997. Touba received a BS degree from the University of Minnesota and MS and PhD degrees in electrical engineering from Stanford University. He is a member of the IEEE.



Bahram Pouya is a graduate student in the Department of Electrical and Computer Engineering at the University of Texas at Austin. His technical interests include design for testability, testing core-based designs, and circuit design. Pouya received his bachelor's degree in electrical engineering from the University of Texas at Austin. He is a member of Tau Beta Pi, Eta Kappa Nu, and the IEEE.

Send questions and comments about this article to Nur A. Touba, University of Texas, Dept. of Electrical and Computer Eng., Engineering Science Bldg., Austin, TX 78712-1084; [touba, pouya]@ece.utexas.edu.