MODIFYING USER-DEFINED LOGIC FOR TEST ACCESS TO EMBEDDED CORES

Bahram Pouya and Nur A. Touba

Computer Engineering Research Center
Department of Electrical and Computer Engineering
University of Texas, Austin, TX 78712-1084

Abstract

Testing embedded cores is a challenge because access to core I/Os is limited. The user-defined logic (UDL) surrounding the core may restrict the set of test vectors that can be applied to the core. Consequently, some of the core test vectors specified by the core supplier may not be contained in the output space of the UDL that drives the core and hence cannot be justified at the core inputs. Conventional solutions to this problem involve placing multiplexers or boundary scan elements at the inputs of the core to provide test access. This can be very costly in terms of area and performance. This paper presents a new approach for providing test access to an embedded core. A procedure is described for inserting control points in the UDL to modify its output space so that it contains the specified core test vectors. The flexibility in selecting the location of the control points is used to avoid performance degradation by keeping test logic off the critical timing paths. Experimental results are shown comparing the control point insertion procedure with other approaches.

1. Introduction

In order to shorten product development cycles for integrated circuits and systems, a growing trend is to make use of pre-designed blocks, called *cores*. Testing cores embedded within a larger design can be a significant challenge because there is limited access to the core I/Os. The set of vectors that can be justified at the inputs of a core is restricted to those that exist in the output space of the user-defined logic (UDL) driving the core. There are two important cases where the particular set of test vectors that are to be used to test a core are "fixed" independent of the design in which the core is embedded (i.e., the vectors are not selected using an ATPG procedure that considers the overall circuit, core plus UDL). The first case is where the core supplier considers the core to be intellectual property and thus is not willing to give any information about the internal logic of the core (i.e., it is a black box). In that case, the traditional test generation process such as ATPG and fault simulation cannot be performed. The core vendor specifies the set of test vectors that must be applied to the core to guarantee a sufficient fault coverage. The second case where the core test vectors may be "fixed" is if the core is a *legacy design* (i.e., an existing design that is being re-used) for which it is desirable to use existing test vectors rather than expending new effort in test generation. In both of these cases, some of the core test vectors may not be contained in the *output space* of the UDL driving the core, and hence some design-for-testability (DFT) is required in order to apply the vectors to the core.

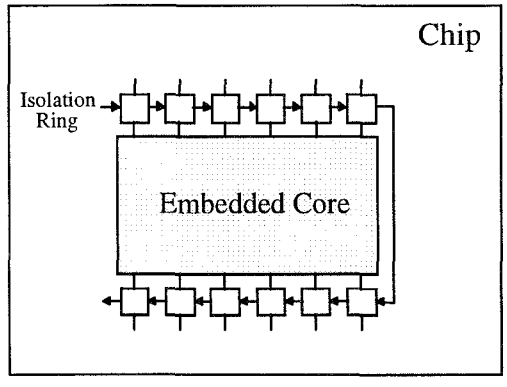

Figure 1. Isolation Ring for Testing Embedded Core

One solution to this problem is to use multiplexing to make the inputs of the core accessible to the chip pins [Immaneni 90]. However, this approach adds a MUX delay on all paths into the core and does not provide any observability to the UDL driving the core thereby resulting in degraded fault coverage. Another solution is to put what is called an *isolation ring* around the core (illustrated in Fig. 1). An isolation ring is essentially a boundary scan that provides full controllability of the inputs of the core as well as providing full observability of the UDL driving the core. The drawback of using an isolation ring is the large area and performance overhead that it adds. A boundary scan element and associated routing is required for each input of the core, and a MUX delay is added to every path into the core (as illustrated in

0-7803-4209-7/97 $10.00 © 1997 IEEE

Fig. 2). As a result, using a full isolation ring is undesirable, especially in high performance applications. Recently a partial isolation ring approach was presented in [Touba 97]. It reduces the number of isolation ring elements while still providing the same fault coverage as a full isolation ring. This is accomplished by justifying part of each core test vector through the UDL.

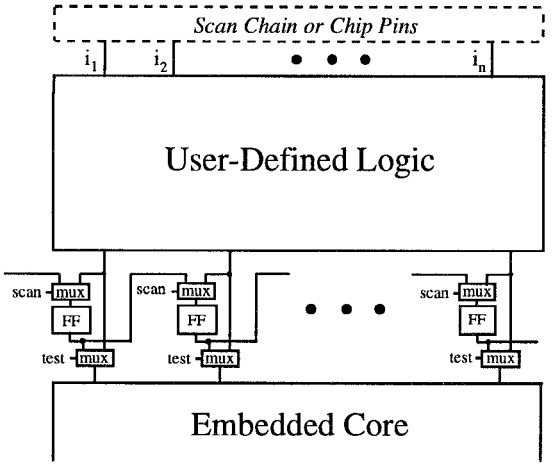

Figure 2. Conventional Approach with Isolation Ring at Inputs of Embedded Core

Bhatia, et. al, at CrossCheck [Bhatia 96] introduced a grid-based direct access methodology for testing core-based designs. This method differs from the embedded grid test method described in [Gheewala 89] and [Chandra 91], in that it uses a "soft" netlist level grid as opposed to a "hard" grid embedded at the base of gate arrays. The "soft" netlist level grid described in [Bhatia 96] provides direct access to storage elements, observation test points, and bi-directional test points via the chip pins. The basic approach is to place bi-directional test points at the inputs and outputs of the embedded core, and matrix accessible storage elements and observation test points in the UDL to provide sufficient fault coverage. This approach requires new library cells to implement the matrix accessible storage elements, bi-directional test points, and I/O pads, and it requires global routing of the test grid.

This paper presents a new approach for providing test access to an embedded core. The idea is to modify the output space of the UDL so that it contains all of the specified core test vectors (as illustrated in Fig. 3). By so doing, the core test vectors can be applied through the UDL thereby completely eliminating the need for an isolation ring and its associated MUX delays. Instead of scanning core test vectors into an isolation ring, the core test vectors are justified at the core inputs. The core test vectors are justified by controlling the inputs of the UDL. If the UDL contains an internal scan chain, then the contents of the scan chain are also controlled. If there are multiple cores where core *A* is driving core *B* with UDL in between them, then this approach can be used to eliminate the need for an isolation ring at the inputs of core *B*. The core test vectors for core *B* can be justified at the inputs of core *B* by controlling the isolation ring at the output of core *A* (which corresponds to the inputs of the UDL between core *A* and core *B*).

Note that in order to test the UDL driving the core, some means for observing the outputs of the UDL is required. This can be provided by using a space compactor. A space compactor combines outputs through combinational logic in order to simplify output response analysis. Instead of having one flip-flop per output, as is the case with an isolation ring, a space compactor can be used to greatly reduce the number of flip-flops that are needed while still providing the same fault coverage [Chakrabarty 94]. Using a space compactor reduces area overhead and speeds up output response analysis without adding logic on any of the system paths.

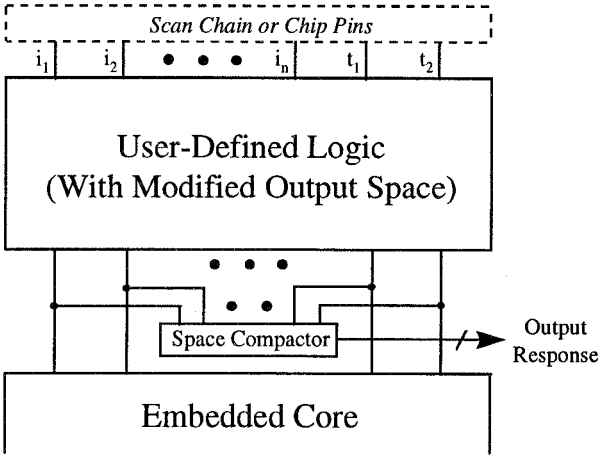

Figure 3. Proposed Approach with Output Space Modification via Control Point Insertion and Space Compaction for Observing UDL Outputs

Modifying the output space of the UDL is accomplished by adding extra inputs that are used only during testing. The general idea of adding extra inputs to a circuit to aid in testing is known as *test point insertion.* Test point insertion was originally proposed for simplifying the task of automated test pattern generation (ATPG) in [Williams 73]. Later it was proposed for improving the random pattern testability of a circuit in [Eichelberger 83]. Here it is proposed for modifying the output space of a circuit. In particular, modifying the output space of the UDL in a core-based design to enable specified sets of test vectors to be applied to each core.

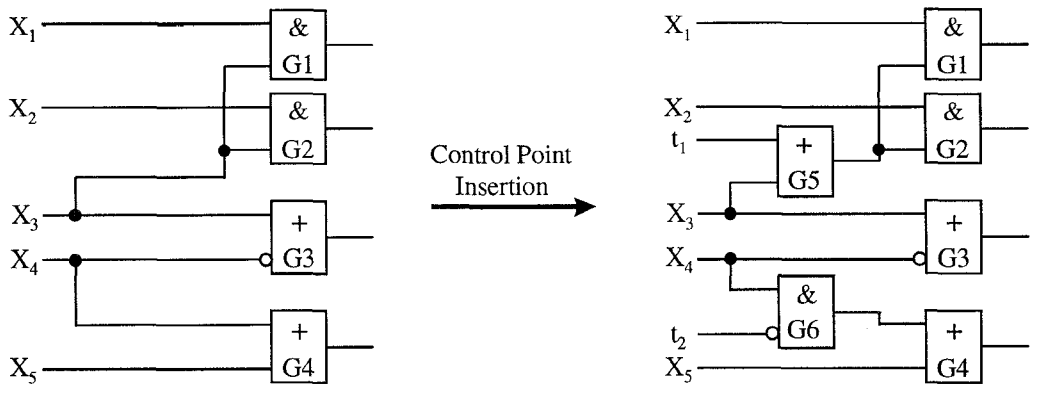

Figure 4. Example of Modifying Output Space by Inserting Control Points

Whereas using an isolation ring or using MUXed inputs to apply test vectors to the core adds a MUX delay to every path, inserting control points in the UDL only adds delay to some of the paths. By carefully selecting the location of the control points, the critical timing paths can be avoided so that system performance is not degraded. Putting multiplexers on every output of the UDL is a simple brute force way to modify the output space, but it is very costly in terms of area and performance. The technique described here is a more efficient and low-cost way to modify the output space of the UDL to provide test access to an embedded core.

The paper is organized as follows: In Sec. 2, the use of control points for modifying the output space of the UDL is described. In Sec. 3, the concepts and strategies that form the basis for the proposed control point insertion procedure are explained. In Sec. 4, a step by step description of the control point insertion procedure is given. Each step is illustrated on an example circuit. In Section 5, experimental results are shown for the control point insertion procedure and the overhead is compared with that for using full and partial isolation rings. Section 6 is a summary and conclusion.

2. Modifying Output Space

The technique described in this paper involves modifying the output space of the UDL while still allowing it to perform its intended function during system mode. The means for accomplishing this is to insert *control points* in the UDL which can be used during test mode to control the value at certain nodes in order to justify particular output vectors. In the example in Fig. 4, the output space of the circuit contains only 9 of the 16 possible patterns. A control point is inserted to fix the logic value at the input of gates *G1* and *G2* to a 1 when the control point is activated (this is called a *control-1 point*), and a control point is also added to fix the logic value at the input of gate G4 to a 0 when the control point is activated (this is called a *control-0 point*). These two control points allow all 16 possible patterns to be justified during test mode. During system mode, the t_1 and t_2 inputs are set to 0, so the control points are not activated and thus don't affect the system function. However, control points do add an extra level of logic to some paths in the circuit. If a control point is placed on a critical timing path, it can increase the delay through the circuit, so care must be taken in selecting the location of the control points.

3. Strategy For Inserting Control Points

This section describes several important concepts and strategies that form the basis for the proposed control point insertion procedure for modifying the output space of the UDL. Given a specified core test vector that cannot be justified through the UDL, there are two basic steps: the first is to identify conflicts that arise when trying to justify the vector, and the second is to insert control points to remove those conflicts so that the vector can be justified.

3.1 Identifying Conflicts

For each core test vector that is to be justified through the UDL, the values of the primary outputs are set to the corresponding values and backtracing towards the primary inputs is performed. If a fanout stem is reached where the values that are assigned on its branches are not all the same, a *conflict* occurs. If backtracing can be completed all the way to the primary inputs with no conflicts, then the output vector can be justified through the UDL using the input vector corresponding to the final values assigned to the primary inputs. Notice that if there is no fanout in

the circuit, no conflicts can occur. Hence, the output space of a fanout-free circuit contains all possible output vectors.

In backtracing through the circuit, two types of gates are encountered: "decision gates" and "imply gates." If the value assigned to the output of a multi-input gate can be justified by assigning a value to only one of the inputs (e.g., justifying a 0 at the output of an AND gate), then the gate is said to be a *decision gate* since there is more than one way to backtrace through the gate. If justifying the value assigned to the output of a gate requires assigning values to all of the inputs (e.g., justifying a 1 at the output of an AND gate), then the gate is said to be an *imply gate*. In the example in Fig. 5, for output vector *0110*, gates *G1*, *G2*, *G3*, and *G5* are imply gates, while gate *G6* is a decision gate.

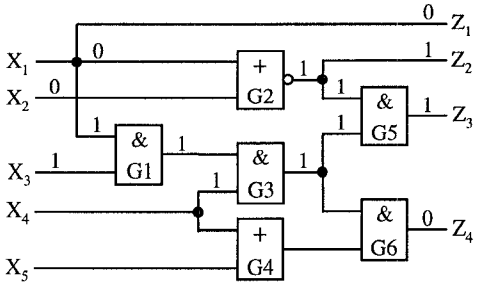

Figure 5. Example of Backward Implications from Primary Outputs

If a conflict occurs during backtracing, then it may be possible to avoid the conflict by choosing to backtrace down a different path in some decision gate. If all possible paths for backtracing result in a conflict(s), then a control point must be inserted to remove the conflict(s) to permit the core test vector to be justified through the UDL.

3.2 Imply and Decision Conflicts

Each conflict that is encountered during backtracing can be classified as either an "imply conflict" or a "decision conflict." An *imply conflict* is one that occurs due to backtracing through imply gates only (i.e., no backtracing is done through decision gates), whereas a *decision conflict* is one that occurs due to backtracing through one or more decision gates and any number of imply gates. A decision conflict can be avoided by backtracing down a different path in some decision gate, whereas an imply conflict cannot be avoided because there are no decision gates between the primary outputs and the fanout point where the conflict occurs. In Fig. 5, the conflict at primary input X_1 is an imply conflict because inconsistent values are implied on the fanout branches through imply gates only. For the decision gate *G6*, if backtracing is done down the input towards gate *G3*, then a decision conflict occurs at the output of gate *G3*, however, if backtracing is done down the other input towards gate *G4*, then a decision conflict occurs at primary input X_4.

All imply conflicts must be removed with control points in order to justify the output vector. Some decision conflicts may also need to be removed, but there is more than one option as to which decision conflict is removed for justifying a particular output vector. When inserting control points to justify a set of vectors, a good strategy is to first insert control points to remove all of the imply conflicts for all of the vectors (since they must be removed in any solution) before removing decision conflicts. Consider the case where vector v_1 can be justified by either removing decision conflict c_1 or decision conflict c_2, but for vector v_2, conflict c_2 is an imply conflict. Any solution will require that conflict c_2 be removed, so removing conflict c_1 would be superfluous. It is better to defer the selection of which decision conflicts to remove until all imply conflicts have been removed.

3.3 Removing Imply Conflicts

A conflict involves a fanout point in which some of the fanout branches have a 0 implied on them and some have a 1 implied on them. A control point must be inserted in order to change the implied values on the fanout branches so that they are all consistent (either all 1's and X's, or all 0's and X's). A conflict at a fanout point can be removed with a single control point. The set of branches with a 1 (0) can be controlled by a single control-1 (control-0) point that is placed between the fanout stem and the set of branches. Then all of the branches in the original fanout point will either have a 0 (1) or an X implied on them resulting in a consistent set of values with a 0 (1) being implied on the stem.

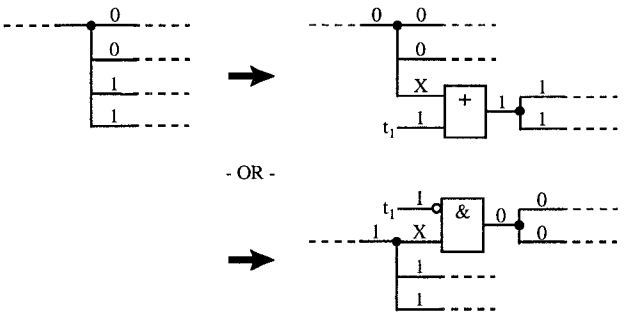

Figure 6. Removing a Conflict with Either a Control-1 Point or a Control-0 Point.

Note that for any conflict, there are two ways to remove the conflict with a single control point: either a

control-1 point can be inserted to control all of the branches with a 1, or a control-0 point can be inserted to control all of the branches with a 0 (as illustrated in Fig. 6). If one or more of the branches is on a critical timing path, then the decision on which way to remove the conflict can be made based on minimizing the performance impact by keeping the control point off the critical timing paths if possible (it will always be possible if only one branch is on a critical timing path, but it may not be possible if multiple branches are on the critical timing path and have opposite logic values implied on them). If none of the branches are on the critical timing path, then the decision on which way to remove the conflict can be based on which value is better to imply on the stem. If it is easier to control the stem to a 1 (0), then a control-0 (control-1) point should be inserted to remove the conflict. Many methods for determining heuristic controllability values exist [Rutman 72], [Breuer 78], [Goldstein 79], [Brglez 84], etc. These controllability values can be used to determine whether it is easier to control a particular node in the circuit to a 1 or to a 0.

If the control point needs to be inserted on only one of the branches (i.e., only one branch has a conflicting logic value from the rest of the branches), then rather than inserting the control point right after the fanout point where the conflict occurs, it is better to insert the control point further down the circuit towards the primary outputs. The reason for this is that the control point reduces the number of backward implications that are made in the circuit as illustrated in Fig. 7. The question then is how far towards the primary outputs should the control point be placed. If the fanout branch where the conflict occurs propagates to only one primary output, then the best location to place the control point is right at the primary output. If the fanout branch where the conflict occurs (call it $FB_{conflict}$) propagates to multiple primary outputs then that means there is another fanout point (call it $FP_{no\_conflict}$) further down the circuit where the path branches out towards multiple primary outputs. In that case, the strategy is to place the control point right before $FP_{no\_conflict}$ (as illustrated in Fig. 7), because otherwise multiple control points would be needed to

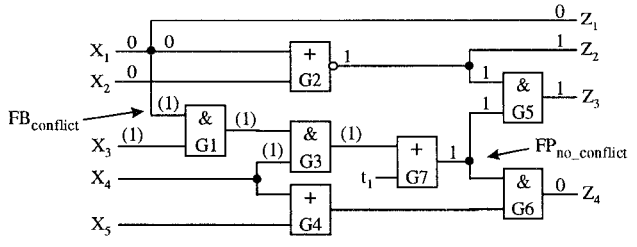

Figure 7. Backward Implications Removed by the Control Point (Gate *G7*) are Shown in Parenthesis.

change the value on each branch of $FP_{no\_conflict}$ in order to change the value that is implied on the stem at $FP_{no\_conflict}$ (which is necessary to remove the conflict at $FB_{conflict}$). So the idea is to use only one control point to remove the conflict at $FB_{conflict}$, and to place that control point as far down the circuit towards the primary outputs as possible in order to remove as many backwards implications in the circuit as possible. The fewer backwards implications there are, the less chance there is for additional conflicts.

3.4 Removing Decision Conflicts

Once the imply conflicts have been removed for all of the core test vectors, then for the remaining vectors that still cannot be justified at the output of the UDL, some decision conflicts must be removed with control points. Unlike the case with imply conflicts, there is some flexibility in choosing which decision conflicts to remove. This flexibility can be used to avoid inserting control points on critical timing paths. The first criteria in selecting which decision conflicts to remove involves checking to see which can be removed without inserting logic on critical timing paths. As was described before, a conflict can be removed from a fanout point without adding logic on a critical timing path provided the fanout point does not have multiple branches that are on critical timing paths with opposite logic values implied on them. The decision conflicts that cannot be removed without inserting logic on critical timing paths should be avoided if possible. For the remaining decision conflicts, the strategy is to remove the conflicts one at a time until all of the vectors can be justified. The heuristic that is used for selecting which decision conflict to remove is to choose the one whose fanout branches have the worst controllability (one of the many methods for determining heuristic controllability values can be used to estimate which has the worst controllability). The idea behind this selection heuristic is that inserting a control point reduces the number of backward implications, so placing the control point at the harder to control nodes is more likely to reduce the number of additional conflicts. Once the decision conflict to be removed has been selected, then the procedure for inserting a single control point to remove the conflict is the same as was previously described for imply conflicts. Decision conflicts continue to be removed one at a time until all of the core test vectors can be justified at the outputs of the UDL.

3.5 Reducing Number of Test Inputs in UDL

Extra primary inputs, called *test inputs*, must be added to the UDL to drive the control points. During test mode, the test inputs are used to justify the specified core test vectors through the UDL, and during system mode, the test inputs are simply set to 0 so that the control points

don't affect the system function. Rather than having one test input driving each control point, in many cases it is possible to have a test input drive several control points. This is obviously advantageous because it reduces the size of the test vectors that need to be applied to the inputs of the UDL during testing hence reducing the cost of test application.

Reducing the number of test inputs in the UDL can be done by simply checking whether one test input can be combined with another while still being able to justify all of the specified core test vectors through the UDL. If so, then the two test inputs are replaced by one which drives the combined set of control points. Test inputs can continue to be combined until a point is reached where no pair of test points can be combined without causing some specified core test vector to not be justifiable.

4. Control Point Insertion Procedure

The control point insertion procedure is described step by step in this section. A running example of inserting control points in the small circuit shown in Fig. 8 to justify only the one vector *0110* at its outputs will be used to illustrate each step (of course in practice a whole set of vectors would be considered for each step). The critical timing paths in the circuit in Fig. 8 are shown in bold.

Input: UDL and Specified Set of Core Test Vectors

Output: UDL with Modified Output Space and UDL Input Vectors that Justify Core Test Vectors

Step 1: Identify all imply conflicts for all vectors

This is done by setting the primary outputs of the UDL to correspond to each specified core test vector and backtracing through imply gates only. Any fanout point whose branches have inconsistent values is marked as an imply conflict.

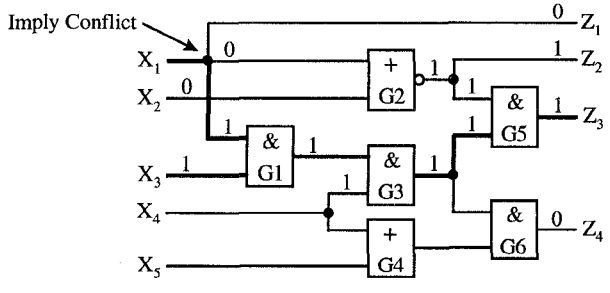

Figure 8. Example Circuit with Backtracing Through Imply Gates (Critical Timing Paths Shown in Bold)

In the example in Fig. 8, backtracing through the imply gates is done for the vector *0110*. Notice, for example, that no values are implied on the inputs of gate *G6* because it is a decision gate. There is one imply conflict which is at the fanout point at primary input X_1.

Step 2: Insert control points to remove all imply conflicts

One control point is inserted for each imply conflict. Either a control-0 point is inserted to control all of the 0 branches, or a control-1 point is inserted to control all of the 1 branches. If one or more of the conflicting branches is on a critical timing path, then this decision about which type of control point to insert is made based on not adding logic on the critical timing path. Otherwise, the decision about which type of control point to add is made based on whether it is easier to control the stem to a 1 or to a 0. If the control point needs to be inserted on only one branch, then it is placed as far towards the primary outputs as possible as described in Sec. 3.3.

In the example in Fig. 8, the 1 branch is on a critical timing path, so a control-0 point is inserted to remove the conflict. The resulting circuit with the control-0 point (gate *G7*) inserted is shown in Fig. 9.

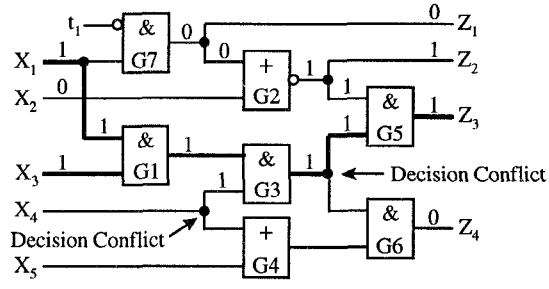

Figure 9. Example Circuit with Control Point (Gate *G7*) Inserted to Remove Imply Conflict.

Step 3: Insert control points to remove decision conflicts one at a time until all vectors can be justified

The decision conflicts are removed one at a time based on two criteria. The first is to avoid inserting control points on critical timing paths, and the second is to choose the one whose fanout branches have the worst controllability.

In the example in Fig. 9, there are two decision conflicts. One at the output of gate *G3*, and the other at primary input X_4. Both can be removed without inserting a control point on critical timing paths, so the one at the output of gate *G3* is selected because it has the worst controllability. It is removed by inserting a control-0 point. Since there is only one branch, the control point

(gate *G8*) is placed as close to the primary outputs as possible (which in this case is right at the primary output Z_4). The resulting circuit is shown in Fig. 10. Now there are no more conflicts.

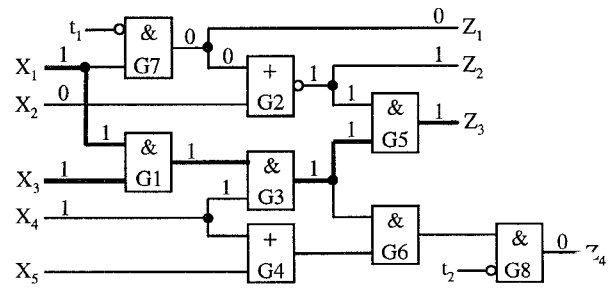

Figure 10. Example Circuit with Control Point (Gate *G8*) Inserted to Remove Decision Conflict

Step 4: Reduce the number of test inputs

A check is made to see if any two test inputs can be combined while still being able to justify all of the vectors. If so, then the two test inputs are replaced by a single test input which drives the combined set of control points. This process continues until a point is reached where no pair of test points can be combined without causing some of the vectors to not be justifiable.

In the example in Fig. 10, a check is made to see if the two test inputs, t_1 and t_2, can be combined. Since there is only one test vector that needs to be justified and both control points are activated to justify that vector, the test inputs can be combined such that there is only one test input that drives both of the control points. The final circuit is shown in Fig. 11. The output vector *0110* can be justified by applying the input vector $< t_1, X_1, X_2, X_3, X_4, X_5 >$ = *11011X* to the circuit.

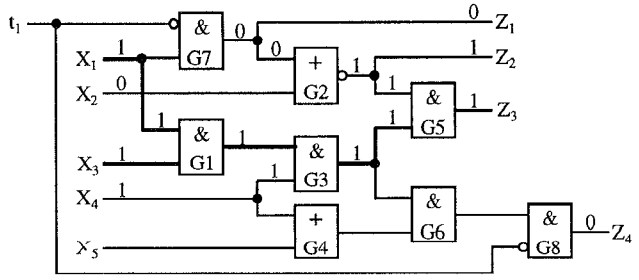

Figure 11. Example Circuit After the Control Point Insertion Procedure Completes

5. Experimental Results

The control point insertion procedure described in this paper was used to modify the UDL in some designs that were constructed from the MCNC benchmark circuits. In each design, one of the benchmark circuits was considered to be a core while another benchmark circuit was considered to be the UDL driving the core. Two of the benchmark circuits, *C5315* and *C7552*, were partitioned into two parts where one part was considered to be a core and the other part was considered to be the UDL.

Table 1 gives information about the designs that were used. For each design, the name of the UDL driving the core is shown followed by the name of the core. The number of inputs to the core, number of faults in the core, and number of specified test vectors for the core are shown (the test vectors were obtained by doing ATPG on the core alone).

Table 1. Information about Designs

#	UDL		Core			
	Name	Inputs	Name	Inputs	Faults	Vectors
1	vda	17	s838	34	820	185
2	k2	45	s9234	36	6010	285
3	apex7	49	C499	41	928	68
4	x4	94	s13207	62	8450	831
5	apex6	135	s15850	77	4055	738
6	C2670	233	dsip	140	5854	63
7	C5315-a	178	C5315-b	117	493	71
8	C7552-a	207	C7552-b	261	877	92

Table 2 shows results for each of the designs in Table 1. The area overhead for different approaches are compared in terms of gate equivalents (GE's) where a MUX is counted as 1.5 GE's and a two-input gate is counted as 1 GE. Results are shown for three cases: (1) using either a full isolation ring or MUXed inputs (both require one MUX per core input), (2) using a partial isolation ring (as described in [Touba 97]), and (3) using the control point insertion procedure described here. For each of the first two approaches, the number of MUXes and the corresponding number of gate equivalents are shown. For the control point insertion procedure, three things are shown: the number of control points that are inserted in the UDL, the number of test inputs (i.e., extra primary inputs) that are added to drive the control points, and the corresponding number of gate equivalents (there is one 2-input gate for each control point). Note that in all three cases, the fault coverage is 100%.

Table 2. Comparison of Different Approaches for Applying Specified Set of Core Test Vectors

#	UDL Name	Core Name	Full Isolation Ring or MUXed Inputs		Partial Isolation Ring [Touba 97]		Control Point Insertion		
			Num MUXes	Num Gate Equiv.	Num MUXes	Num Gate Equiv.	Num Cntrl Points	Num Test Inputs	Num Gate Equiv.
1	vda	s838	34	51	29	43	14	4	14
2	k2	s9234	36	54	31	46	22	9	22
3	apex7	C499	41	61	6	9	11	4	11
4	x4	s13207	62	93	9	13	15	6	15
5	apex6	s15850	77	105	8	12	5	3	5
6	C2670	dsip	140	210	28	42	31	8	31
7	C5315-a	C5315-b	117	175	73	109	68	14	68
8	C7552-a	C7552-b	261	391	63	94	57	11	57

As can be seen from the results, in many cases fewer control points are needed than the number of MUXes required in a partial isolation ring. Each control point is implemented with a single two-input gate and requires routing of one control line, whereas MUXes require routing of both a control line and a data line. Moreover, the number of test inputs needed to drive the control points is very small which translates into reduced test application costs.

6. Summary and Conclusions

A new approach for providing test access to the inputs of an embedded core was presented. Given a specified set of core test vectors, an automated procedure was presented for efficiently inserting control points to modify the output space of the UDL so that all of the vectors can be justified through the UDL. This approach of justifying the specified core test vectors through the UDL eliminates the need for placing MUXes at the inputs of the core to directly control them.

Inserting control points in the UDL is a very efficient and flexible way to provide test access to the inputs of an embedded core. This fact can be exploited to reduce the cost of DFT in core-based design in the following ways:

1) Avoiding performance degradation - The flexibility in selecting the location of control points can be used to keep test logic off of critical timing paths. Techniques for accomplishing this were described in this paper.
2) Reducing test application costs - Control points provide a means for maximizing the effectiveness of each test input. A single test input can drive multiple control points and thus have a bigger impact in justifying the specified core test vectors. This results in fewer test inputs, less test data, and faster test time.
3) Reducing DFT logic - Control points are very efficient to implement requiring only a single two-input gate.
4) Reducing DFT routing - A major concern about DFT in core-based designs is the amount of routing that it adds. The flexibility in selecting the location of the control points provides a means for reducing routing complexity. This paper did not address the issue of routing, but that is an area for further investigation.

Acknowledgments

This work is part of the TOPS project at the Center for Reliable Computing at Stanford University and was supported in part by the Advanced Research Projects Agency under prime contract No. DABT63-94-C-0045, and by the National Science Foundation under Grant No. MIP-9702236.

References

[Bhatia 96] Bhatia, S., T. Gheewala, and P. Varma, "A Unifying Methodology for Intellectual Property and Custom Logic Testing," *Proc. of International Test Conference*, pp. 639-648, 1996.

[Breuer 78] Breuer, M.A., "New Concepts in Automated Testing of Digital Circuits," *Proc. of EEC Symposium on CAD of Digital Electronic Circuits and Systems*, pp. 69-92, 1978.

[Brglez 84] Brglez, F., "On Testability Analysis of Combinational Networks," *Proc. of International Symposium on Circuits and Systems*, pp. 221-225, 1984.

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

[Chandra 91] Chandra, S., T. Ferry, T. Gheewala, and K. Pierce, "ATPG Based on a Novel Grid Addressable Latch Element," *Proc. of 28th Design Automation Conference*, pp. 282-286, 1991.

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

[Gheewala 89] Gheewala, T., "CrossCheck: A Cell Based VLSI Testability Solution," *Proc. of 26th Design Automation Conference*, pp. 706-709, 1989.

[Goldstein 79] Goldstein, L.H., "Controllability-Observability Analysis of Digital Circuits," *IEEE Trans. on Circuits and Systems*, Vol. CAS-26, No. 9, pp. 685-693, Sept. 1979.

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

[Rutman 72] Rudman, R.A., "Fault Detection Test Generation for Sequential Logic by Heuristic Tree Search," *IEEE Computer Group Repository*, Paper No. R-72-187, 1972.

[Touba 97] Touba, N.A., and B. Pouya, "Partial Isolation Rings for Testing Embedded Cores," *Proc. of VLSI Test Symposium*, pp. 10-16, 1997.

[Williams 73] Williams, M.J.Y., and J.B. Angell, "Enhancing Testability of Large-Scale Integrated Sequential Circuits Via Test Points and Additional Logic," *IEEE Trans. on Computers*, Vol. C-22, pp. 46-60, 1973.