Transactions Brief

Circular BIST With State Skipping

Nur A. Touba

***Abstract*—Circular built-in self-test (BIST) is a "test per clock" scheme that offers many advantages compared with conventional BIST approaches in terms of low area overhead, simple control logic, and easy insertion. However, it has seen limited use because it does not reliably provide high fault coverage. This paper presents a systematic approach for achieving high fault coverage with circular BIST. The basic idea is to add a small amount of logic that causes the circular chain to skip to particular states. This "state skipping" logic can be used to break out of limit cycles, break correlations in the test patterns, and jump to states that detect random-pattern-resistant faults. The state skipping logic is added in the chain interconnect and not in the functional logic, so no delay is added on system paths. Results indicate that in many cases, this approach can boost the fault coverage of circular BIST to match that of conventional parallel BIST approaches while still maintaining a significant advantage in terms of hardware overhead and control complexity. Results are also shown for combining "state skipping" logic with observation point insertion to further reduce hardware overhead.**

***Index Terms*—Built-in self-test (BIST), built-in testing, circuit testing, design for testability, limit cycles, pseudorandom pattern generation, scan chains.**

I. Introduction

There are two main functions that must be performed on-chip in order to implement built-in self-test (BIST): test pattern generation and output response analysis. The most common BIST schemes are based on pseudorandom test pattern generation using linear feedback shift registers (LFSRs) and output response compaction using signature analyzers [1]. Three architectures that are used for performing BIST are the following.

1) *Scan BIST:* An external LFSR is used to shift pseudorandom test vectors into the scan chain, and an external signature register is used to compact the response. This is a "test per scan" approach where one test vector is applied to the circuit each time the complete scan chain is loaded.
2) *BILBO Registers:* The flip-flops in a design are formed into BILBO registers [2] that can operate in scan mode, test pattern generator mode, or signature analyzer mode. The complete circuit is tested by scheduling the operation of the various BILBO registers so that some operate in test pattern generation mode and others in signature analyzer mode in different test sessions. BILBO registers provide a test-per-clock scheme, where a test vector is applied to the circuit each clock cycle.
3) *Circular BIST:* The flip-flops in a design are replaced with special BIST cells, which are connected to form one long circular chain [3]–[5]. During BIST operation, each flip-flop is fed by the exclusive-OR of its normal functional input and the output of the flip-flop that precedes it in the chain. Hence the response of the circuit in each clock cycle during BIST is compacted in the circular chain and then used as the test pattern in the next clock cycle. This is a test-per-clock scheme that requires only one test session.

Circular BIST provides a number of attractive features. It has a much shorter test time than scan BIST because a test pattern is applied each clock cycle. It has less overhead than using BILBO registers because there is less register interconnection, less control complexity, and fewer modes of operation. The BIST control logic for circular BIST is very simple because there is only a single test session. Moreover, circuit BIST is very easy to insert into a design because it is just like scan insertion except using BIST cells instead of scan cells.

Despite all of the advantages that circular BIST offers compared with conventional BIST approaches, it has seen limited use. The reason is that it does not reliably provide high fault coverage. The test patterns that are generated in circular BIST are not truly pseudorandom, as they are with scan BIST or BILBO registers, and there can be significant aliasing due to register adjacency [4], [6].

Several solutions to the register adjacency problem have been described in [4] and [7]–[9]. However, the problem of reliably generating test patterns that provide high fault coverage during circular BIST has not been adequately dealt with. This problem is the major factor that limits the effectiveness of circular BIST and is the problem that is addressed in this paper.

The test patterns generated by an LFSR have a guaranteed pseudorandom property that can be used to reliably predict fault coverage given the detection probabilities of the faults in the circuit [10]. This is not the case for the test patterns that are generated during circular BIST. Some probabilistic models of circuit behavior were used in [5] and [7] to try to estimate the expected number of different test patterns that are generated in a k-bit section of the circular chain during circular BIST. This analysis assumes that the inputs to the k-bit section of the circular chain are completely independent of the outputs of the k-bit section. This assumption is really an approximation because obviously the circular nature of the chain means that the outputs of the k-bit section will eventually influence the inputs of the k-bit section after some number of clock cycles. Consequently, the number of different test patterns and the probability distribution of the patterns that are predicted by probabilistic analysis of circular BIST are not reliable and cannot be depended on. As has been shown in [11] and [12], in many real circuits, the fault coverage provided by circular BIST can be surprisingly low.

One way to view circular BIST is that it converts the circuit into an autonomous finite-state machine (FSM) (i.e., with no primary inputs) during testing. In the state transition diagram for circular BIST, there is exactly one outgoing edge from each state. A state may not necessarily have any incoming edges, in which case it can only be visited if it is the initial state. The state transition diagram contains one or more cycles that partition it into *state transition subgraphs;* this is illustrated in Fig. 1. One problem that can arise with circular BIST is *limit cycling*, which occurs when the circuit gets stuck in a state cycle and repeatedly generates the same test patterns. It should be noted that for large circuits, as circular BIST was originally proposed for in [5], limit cycling is very unlikely to occur. However, for control circuits and other smaller circuits, identifying an initial state for circular BIST that will not result in limit cycling is a problem. Another problem is that the

Manuscript received February 13, 2002; revised November 29, 2002. This work was supported in part by the National Science Foundation under Grant MIP-9702236, in part by the Texas Advanced Research Program under Grant 1997-003658-369, and in part by the Advanced Research Projects Agency under Prime Contract DABT63-94C-0045.

The author is with the Computer Engineering Research Center, Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78712-1084 USA (e-mail: touba@ece.utexas.edu).

Digital Object Identifier 10.1109/TVLSI.2002.801564

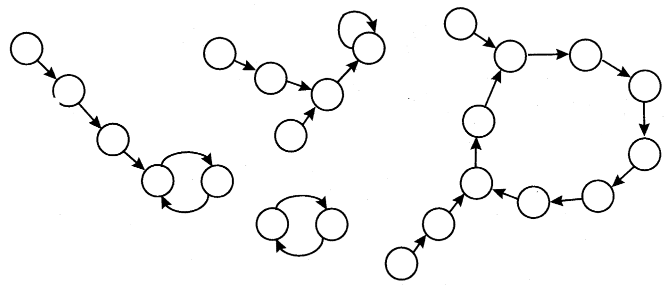


Fig. 1. Example of state transition diagram with four subgraphs.

set of test patterns that detect fault F_1 may be in a disjoint set of state transition subgraphs from the set of test patterns that detect fault F_2. In that case, no initial seed can be found that will allow both faults to be detected. These problems do not arise with an LFSR because there is only one cycle (besides the degenerate all-zero state), and the distance between states that detect different faults can be reliably predicted with probabilistic analysis.

There are three things that can limit the fault coverage achieved by circular BIST.

1) *Limit Cycling.* If the circular chain gets stuck in a cycle before a sufficient set of test patterns is generated, than the fault coverage can be low. Simulation can be used to check whether limit cycling occurs. If so, then a different initial seed can be tried, or the chain can be reordered. However, results were shown in [12] that while these techniques can help, they do not always work. A method for finding the longest acyclic path in the state transition diagram was described in [12]. This approach works in some cases but is computationally intensive for large circuits.
2) *Correlations in Test Patterns.* It may not be possible to generate test patterns for some faults regardless of the test length because of correlations due to the circuit structure. Avoiding register adjacency helps reduce this problem considerably, but there are many other sources of correlation that can occur due to reconvergence after multiple clock cycles. Some examples of a few different types of correlation that can occur in circular BIST structures were shown in [9]. Techniques for analyzing word-level correlation are described in [13].
3) *Random-Pattern-Resistant (RPR) Faults.* RPR faults can only be detected by a relatively small number of test patterns, and thus are hard to detect in any pseudorandom BIST scheme. Inserting test points to increase the detection probability for RPR faults is complicated in circular BIST. Inserting a control point completely changes the sequence of test patterns that are generated, hence simulation-based approaches for test point insertion are not effective. Moreover, inserting control points can introduce register adjacency. However, observation-point insertion can be still be used.

This paper proposes a unified approach for solving all three of the problems listed above. The idea is to add a small amount of logic that causes the circular chain to skip to particular states. This "state skipping" logic alters the state transition diagram. If simulation indicates that the circular chain gets stuck in a limit cycle, then state-skipping logic can be used to jump out of the cycle. State-skipping logic can also be used to break correlations in the test patterns, and it can be used to jump to states that detect random-pattern-resistant faults.

An example of state-skipping logic is shown in Fig. 2. When the chain reaches state 1011, if the next state in the sequence would normally be 1000, then the state-skipping logic would cause it to skip to state 1100 instead. Note that the state-skipping logic is added in the

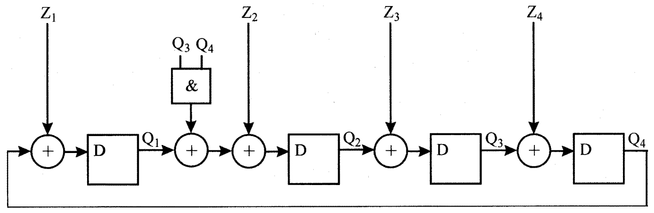


Fig. 2. Example of state-skipping logic.

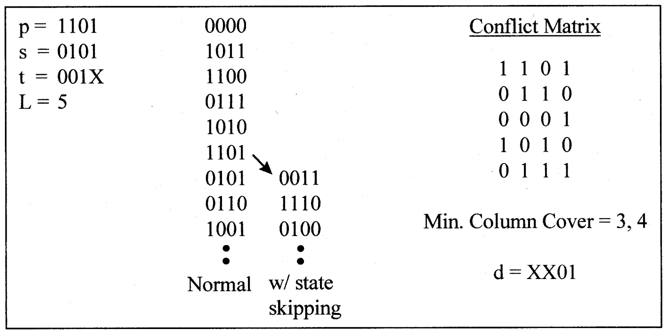


Fig. 3. Example of finding decoding cube d.

chain interconnect and not in the functional logic, so no delay is added on system paths.

A systematic procedure is described for designing state-skipping logic that provides a desired fault coverage. With this procedure, high fault coverage for circular BIST can *reliably* be achieved. This is something that chain reordering and initial seed selection alone cannot guarantee.

This paper is organized as follows. In Section II, an overview of the procedure for adding state-skipping logic to achieve a desired fault coverage is given. In Section III, the process of designing the state-skipping logic is described in detail. In Section IV, combining state skipping with observation-point insertion is discussed for improving coverage of RPR faults. In Section V, experimental results are shown comparing parallel BIST, normal circular BIST, and circular BIST with state-skipping logic. Section VI presents conclusions.

II. Overview of Procedure

Given a circular BIST structure and the initial state, this section describes a systematic procedure for adding state-skipping logic that will provide a desired fault coverage. The basic idea is to do fault simulation for the sequence of states that is generated in the circular chain until a point is reached where no new faults are being detected. At that point, state-skipping logic is added to jump to a new state that detects a fault that is currently undetected and hopefully gets the circular chain in a new state transition subgraph that will allow additional faults to be detected. The decision on when to give up on the current state sequence and add state-skipping logic is governed by a parameter m. If no new faults have been detected by the last m states, then state-skipping logic is added. The parameter m can be used to trade off between hardware area and test time. Smaller values of m will result in more state-skipping logic and a shorter overall test length. Larger values of m will result in a longer overall test length, but less state-skipping logic will be needed.

The procedure is described step by step below.

1) *Do fault simulation until no new faults are detected by the last m states.* Fault simulation is done for the sequence of states that is generated in the circular chain. If no new faults are detected by the last m states, then state-skipping logic is added.

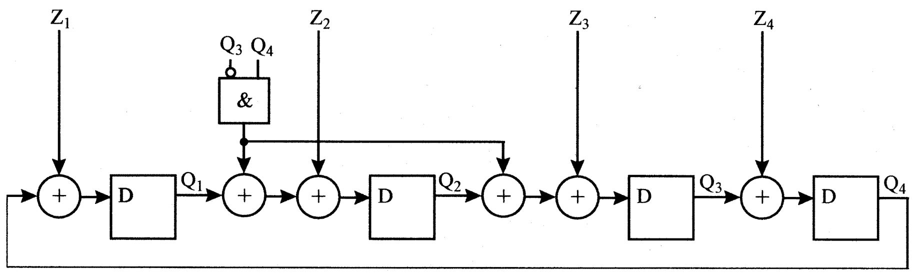


Fig. 4. Circular chain with state-skipping logic for the example in Fig. 3.

2) *Compare test cubes for the undetected faults with the last m states to identify the test cube c and state s that differ in the fewest number of bits (minimum Hamming distance).* Test cubes for the undetected faults are found by automatic test pattern generation (ATPG) and leaving unspecified inputs as "don't cares" (Xs). This ATPG need only be done the first time, and then the test cubes can be stored and reused in subsequent iterations. The reason for finding the test cube c and state s that differ in the fewest number of bits is to minimize the amount of state-skipping logic that is required. None of the last m states detected any faults, so it does not matter which of those states are skipped.
3) *Add state-skipping logic to cause the sequence to jump from the state directly preceding state s to a state that matches the test cube c.* The state-skipping logic alters the sequence of the circular chain so that instead of going to state s, the sequence jumps to a state that matches the test cube c. The state-skipping logic is designed in a way that preserves the same state sequence up to, but not including, state s. Thus, all of the faults that have been detected up to state s will remain detected, and there is no need to resimulate the circular chain. The procedure for designing the state-skipping logic is described in detail in Section III.
4) *If the fault coverage is still not sufficient, then loop back to step 1).* The procedure iterates and continues to add state-skipping logic until the fault coverage is sufficient.

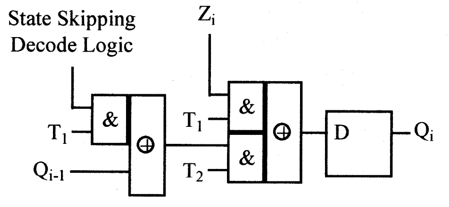
This procedure uses a hill-climbing approach to continue adding state-skipping logic until the fault coverage is sufficient. Because the state-skipping logic preserves the previous sequence, time-consuming resimulation is not necessary. The procedure is guaranteed to eventually achieve the required fault coverage. The overall test length will depend strongly on the value of m. If the test length becomes too long, then the procedure can be repeated with a smaller value of m.

III. State-Skipping Logic

Step 3) of the procedure involves adding state-skipping logic to alter the sequence of the circular chain so that instead of going to state s, the sequence jumps to a state that matches a test cube c. The process of designing this state-skipping logic is the subject of this section.

The first step is to find the largest cube d that contains the state that directly precedes state s (this state will be labeled state p) in the sequence but does not contain any of the other previous states in the sequence. Cube d is used to decode state p and initiate the state skipping. To preserve the sequence, cube d should not decode any of the other previous states in the sequence that come before state p. It is desirable to have cube d be as large as possible so that the corresponding AND will have as few inputs as possible.

Finding the largest cube d is done by finding a minimum column cover in a Boolean conflict matrix. The conflict matrix has one row for each state in the sequence that comes before state p. So if state p is the $(L+1)$th state in the sequence, then there are L rows in the conflict



T_1 T_2	Mode
0 0	Reset
0 1	Shift
1 0	Normal
1 1	BIST

Fig. 5. Example of control logic for circular BIST with state skipping.

matrix. The columns in the conflict matrix correspond to the bits in the state. So if there are n flip-flops in the circular chain, then there are n columns in the conflict matrix. For each of the L states that come before state p, the corresponding row in the conflict matrix is formed by placing a "1" in each column where the bit value of the state is different from the bit value of state p and placing a "0" in each column with the bit values are the same. A set of columns covers the matrix if every row has a "1" in at least one of the columns in the set. The literals in the largest cube d correspond to the set of columns in the minimum column cover of the conflict matrix. This set of literals is compatible with state p, but conflicts with all L states that come before state p in the sequence. Finding the minimum column cover is an NP-complete problem, but efficient heuristics and techniques exist for solving it [14].

An example of forming the conflict matrix and finding the largest cube d is shown in Fig. 3. The conflict matrix is covered by columns 3 and 4, so cube d has two literals that correspond to the last two bits in state p.

When the circular chain reaches state p, the decoding cube d is activated and the state-skipping is performed by complementing the bits in the next state (i.e., state s) that differ from the test cube c to force the next state of the circular chain to match test cube c. Complementing the bits of the next state is performed by adding exclusive-OR gates in front of each flip-flop where state s differs from test cube c. The exclusive-OR gates are added in the chain interconnect and not on the functional path. This is done so that no delay is added to system paths.

Fig. 4 shows the circular chain with state-skipping logic for the example in Fig. 3. The second and third bits in state s differ from the second and third bits in test cube d, so exclusive-OR gates are added in front of the corresponding flip-flops.

One question that arises is whether the state-skipping logic that is added is itself tested. The way that it is constructed ensures that it will be tested by the patterns that are generated during circular BIST. The number of literals in the decoding cube is minimized such that if any input to the corresponding AND gate is removed due to a stuck-at fault, then the cube would decode some pattern earlier in the sequence and therefore change the sequence in the presence of the fault. If the output of the AND gate is stuck at zero, then the state skipping would not occur and the circular chain would follow the normal sequence. So any fault in the state-skipping logic is guaranteed to change the sequence of the circular chain. The only way that the fault would not be detected is if signature aliasing occurs.

TABLE I
RESULTS FOR ISCAS'89 BENCHMARK CIRCUITS

Circuit			Parallel BIST		Circular BIST		Circular BIST w/State Skipping			
Name	Lits	Chain Size	Test Len	Cov	Test Len	Cov	Test Len	Cov	Extra Lits	Overhead
s208	181	18	1248	100	(1187)	92.6	5584	100	49	27%
s298	244	17	480	100	(650)	99.2	721	100	19	8%
s344	269	24	256	100	(25)	77.4	92	100	31	12%
s382	306	24	352	100	(4735)	95.8	5923	100	29	10%
s420	383	34	50K	92.9	50K	92.0	47K	100	137	36%
s510	424	25	832	100	(2835)	98.4	5899	100	56	13%
s526	445	24	10K	100	(8559)	98.6	11K	100	63	14%
s641	539	54	50K	97.6	50K	98.5	45K	100	66	12%
s1196	1009	32	50K	99.7	50K	98.9	35K	100	53	5%
s1423	1164	91	42K	100	46K	100	46K	100	0	0
s5378	4212	199	50K	99.8	50K	86.0	47K	100	1366	32%
s9234	7971	247	50K	93.6	50K	93.7	49K	100	1148	15%
s13207	11234	700	50K	99.0	50K	99.0	44K	100	460	4%

TABLE II
RESULTS FOR OBSERVATION-POINT INSERTION

Circuit			Circular BIST w/State Skipping				Circular BIST w/State Skipping and Observation Point Insertion				
Name	Lits	Chain Size	Test Len	Cov	Extra Lits	Overhead	Test Len	Cov	O-Points	Extra Lits	Overhead
s420	383	34	47K	100	137	36%	35K	100	1	59	15%
s641	539	54	45K	100	66	12%	23K	100	2	0	0
s1196	1009	32	35K	100	53	5%	25K	100	2	0	0
s5378	4212	199	47K	100	1366	32%	50K	100	4	386	9%
s9234	7971	247	49K	100	1148	15%	31K	100	6	873	11%
s13207	11234	700	44K	100	460	4%	49K	100	5	364	3%

Fig. 5 shows an example of how the control logic can be implemented for circular BIST with state-skipping logic. The circular BIST cell that is shown is the one proposed in [4].

IV. COMBINING STATE SKIPPING AND OBSERVATION-POINT INSERTION

For circuits that contain random pattern resistant faults, state-skipping logic can be used to jump to states where the random pattern resistant faults will be detected. However, in some circuits where there is a cone of logic that is difficult to observe, a lot of state skipping may be necessary to detect all the faults in that cone. A more efficient solution would be to insert some observation points to alleviate the observability problem.

Combining observation-point insertion with state skipping is very easy. When a point is reached where very few faults are being detected by additional state skipping, a path-tracing-based observation-point insertion procedure (e.g., [15], [16]) can be used to analyze the patterns applied so far and select the optimal location for observation points to increase the fault coverage. If the response of the observation points is captured in a separate miltiinput signature register (MISR), then the sequence of states visited during circular BIST will be preserved.

Another approach would be to use a testability measure-based test point insertion procedure (e.g., [17] and [18]) right away before the state-skipping procedure is used. This would insert test points in the design at the outset to reduce the random pattern resistance of the design.

V. EXPERIMENTAL RESULTS

Experimental results were generated for some of the ISCAS'89 [19] benchmark circuits comparing parallel BIST, normal circular BIST, and circular BIST with state-skipping logic. The procedure described in this paper was used to insert the state-skipping logic. The results are shown in Table I. For each circuit, the factored form literal count is shown along with the size of the circular chain. The circuits were simulated for up to 50 000 patterns. The same initial seed was used for each of the three BIST schemes. The test length and fault coverage are shown for each scheme. The fault coverage is for detectable faults. For the parallel BIST approach, an LFSR was used to apply pseudorandom patterns, and a separate MISR was used to compact the response. For the normal circular BIST scheme with no state-skipping logic, if the circular chain got stuck in a limit cycle, then the number of distinct test patterns that were generated are shown in parenthesis. For the circular BIST with state-skipping logic, the number of extra literals that are added for the state-skipping logic is shown, and a percentage overhead figure is computed by comparing the number of extra literals for the state-skipping logic with the number of literals in the functional circuit.

Several observations can be made about the results. The fault coverage for circular BIST in most of the smaller circuits ($s208$, $s298$, $s344$, $s382$, $s510$, $s526$) was limited by the fact that the circular chain got stuck in a cycle. Those circuits were very random pattern testable, and parallel BIST achieved complete fault coverage in a very short test length. A small amount of state-skipping logic was sufficient to allow the circular chain to jump out of the limit cycles and achieve 100% fault coverage. The area overhead of the state-skipping logic is much less than what is required for a separate MISR or a CBILBO register to perform parallel BIST. Hence, these results indicate that circular BIST with state skipping is an attractive and effective approach for BIST of small controllers.

For the circuits that contain random-pattern-resistant faults, the fault coverage for both parallel BIST and circular BIST was limited. In some cases, the added correlation in the test patterns generated in circular BIST provided slightly more fault coverage than the purely pseudo-random patterns generated in parallel BIST (e.g., $s641$ and $s9234$), and in some cases it provided less fault coverage (e.g., $s420$, $s1196$,

$s5378$). The fault coverage for $s5378$ was quite a bit lower. For the circuits with a relatively small number of random-pattern-resistant faults, the results indicate that adding state-skipping logic is an efficient way to boost the fault coverage up to 100%.

For the circuits that had a large number of random-pattern-resistant faults (e.g., $s420$ and $s5378$), adding state-skipping logic is not so efficient. As was described in Section IV, a more effective approach for RPR circuits would be to combine state skipping with observation-point insertion. Results are shown in Table II for combining state skipping with observation-point insertion for the RPR circuits. The observation-point procedure described in [16] was used. As can be seen from the results, adding a small number of observation points significantly reduces the amount of additional logic needed for state skipping. It should be noted that some means for observing the observation points is required, either with a condensation network [20] or an MISR (which may already exist in the boundary scan).

VI. Conclusion

A systematic approach for reliably achieving high fault coverage with circular BIST was presented. State-skipping logic is inserted into the circular chain to improve the test patterns that are generated during circular BIST. The state-skipping logic is used to jump out of limit cycles, break correlations in the test patterns, and jump to states that detect random-resistant faults. Result indicate that, in many cases, this approach can boost the fault coverage of circular BIST to match that of conventional parallel BIST approaches while still maintaining a significant advantage in terms of hardware overhead and control complexity.

Acknowledgment

The author would like to acknowledge the contributions of Dr. B. Pouya and Prof. T. Konstantopoulos from the University of Texas at Austin.

References

[1] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques*. New York: Wiley, 1987.
[2] B. Könemann, J. Mucha, and G. Zwiehoff, “Built-in logic block observation techniques,” in *Proc. Int. Test Conf.*, 1979, pp. 37–41.
[3] P. H. Bardell and W. H. McAnney, “Self-testing multichip logic modules,” in *Proc. Int. Test Conf.*, 1982, pp. 200–204.
[4] C. E. Stroud, “Automated BIST for sequential logic synthesis,” *IEEE Design Test Comput.*, pp. 22–32, Dec. 1988.
[5] A. Krasniewski and S. Pilarski, “Circular self-test path: A low-cost BIST technique for VLSI circuits,” *IEEE Trans. Computer-Aided Design*, vol. 8, no. 1, pp. 46–55, Jan. 1989.
[6] C. L. Hudson and G. D. Peterson, “Parallel self-test with pseudo-random test patterns,” in *Proc. Int. Test Conf.*, 1987, pp. 954–963.
[7] S. Pilarski, A. Krasniewski, and T. Kameda, “Estimating testing effectiveness of the circular self-test path technique,” *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 1301–1316, Jan. 1992.
[8] L. J. Avra and E. J. McCluskey, “Synthesizing for scan dependence in built-in self-testable designs,” in *Proc. Int. Test Conf.*, 1993, pp. 734–743.
[9] J. Carletta and C. Papachristou, “Structural constraints for circular self-test paths,” in *Proc. VLSI Test Symp.*, 1994, pp. 87–92.
[10] E. J. McCluskey, S. Makar, S. Mourad, and K. Wagner, “Probability models for pseudorandom test sequences,” *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 68–74, Jan. 1988.
[11] O. Brynestad, E. J. Aas, and A. E. Vallestad, “State transition graph analysis as a key to BIST fault coverage,” in *Proc. Int. Test Conf.*, 1990, pp. 537–543.
[12] F. Corno, P. Prinetto, and M. S. Reorda, “Making the circular self-test path technique effective for real circuits,” in *Proc. Int. Test Conf.*, 1994, pp. 949–957.
[13] J. Carletta and C. Papachristou, “Testability analysis and insertion for RTL circuits based on pseudorandom BIST,” in *Proc. Int. Conf. Computer Design*, 1995, pp. 162–167.
[14] O. Coudert, “On solving covering problems,” in *Proc. 33rd Design Automation Conf.*, 1996, pp. 197–202.
[15] V. S. Iyengar and D. Brand, “Synthesis of pseudo-random pattern testable designs,” in *Proc. Int. Test Conf.*, 1987, pp. 501–508.
[16] N. A. Touba and E. J. McCluskey, “Test point insertion based on path tracing,” in *Proc. VLSI Test Symp.*, 1996, pp. 2–8.
[17] B. H. Seiss, P. M. Trouborst, and M. H. Schulz, “Test point insertion for scan-based BIST,” in *Proc. Eur. Test Conf.*, 1991, pp. 253–262.
[18] K.-T. Cheng and C. J. Lin, “Timing-driven test point insertion for full-scan and partial-scan BIST,” in *Proc. Int. Test Conf.*, 1995, pp. 506–514.
[19] F. Brglez, D. Bryan, and K. Kozminski, “Combinational profiles of sequential benchmark circuits,” in *Proc. Int. Symp. Circuits Systems*, 1989, pp. 1929–1934.
[20] J. R. Fox, “Test-point condensation in the diagnosis of digital circuits,” *Proc. Inst. Elect. Eng.*, vol. 124, no. 2, pp. 89–94, Feb. 1977.

Combined Data-Driven and Event-Driven Scheduling Technique for Fast Distributed Cosimulation

Dohyung Kim, Chan-Eun Rhee, and Soonhoi Ha

***Abstract*—Fast distributed cosimulation is a challenging problem for the embedded system design. The main theme of this paper is to increase the simulation speed by reducing the frequency of intersimulator communications, reducing the active duration of simulators, and utilizing the parallelism of component simulators. Those enhancements are accomplished by the proposed virtual synchronization technique, which combines event-driven and data-driven simulation methods. Experimental results show that the proposed technique can boost the cosimulation speed significantly compared with the previous conservative approaches.**

***Index Terms*—Cosimulation, distributed simulation, time accurate simulation, virtual synchronization.**

I. Introduction

A complex embedded system usually consists of software modules and hardware modules that are mapped to heterogeneous components such as programmable processors, customized application-specific integrated circuits, and Internet protocols (IPs). Though several efforts are invested to model all modules in a single simulation platform, a current practice of system-level simulation is likely to involve communication and synchronization between component simulators, sometimes geographically distributed, to make it a distributed cosimulation. For time-accurate simulation, component simulators are basically event-driven or time-driven so that the simulator processes events in chronological order [1].

As system complexity increases and fast design turnaround time is required, increasing the simulation performance becomes more important especially because the simulation complexity is a superlinear function of design size. The performance bottleneck of distributed event-

Manuscript received November 19, 2001. This work was supported by the National Research Laboratory under Program M1-0104-00-0015, by the Brain Korea 21 Project, and by RIACT at Seoul National University.

The authors are with the School of Computer Science and Engineering, Seoul National University, 151-742 Seoul, Korea.

Digital Object Identifier 10.1109/TVLSI.2002.801572