Design for Verification at the Register Transfer Level

Indradeep Ghosh
Fujitsu Labs. of America, Inc.
Sunnyvale, CA 94085
USA

Krishna Sekar
Departmen t of ECE
Univ. of California, San Diego
La Jolla, CA 92093

Vamsi Boppana
Zenasis Tech. Inc.
Campbell, CA 95008
USA

Abstract

In this paper we introduce a novel concept that can be used for augmenting simulation based verification at the Register Transfer Level (RTL). In this technique the designer of an RTL circuit introduces some well understood extra behavior (through some extra circuitry) into the circuit under verification. This can be termed as design for verification. During RTL simulation this extra behavior is utilized in conjunction with the original behavior to exercise the design more thoroughly thus making it easier to detect errors in the original design. Once the circuit is throughly verified for functionality the extra behavioral constructs can be removed to produce the original verified design. Extensive experiments on a number of industrial circuits demonstrate that the method is promising.

1. Introduction

As VLSI circuits become larger, smaller and more complex the problem of design verification is becoming increasingly intractable. There are two types of design verification methods currently used - simulation based and formal verification. Currently, the most prevalent and widely used one is simulation based verification though formal verification in certain areas is gaining acceptance. The main problem of formal v erification is its inability to tackle large designs within a reasonable amount of time and computing resources. Though a lot of research is going on in this field, experts agree that simulation based verification will remain a major verification method in the future.

In simulation based verification the design to be verified is simulated at various levels of abstraction (RTL, gate or circuit) with a suite of test vectors and the output responses or some intermediate responses are compared for correctness with that of a golden model or an executable specification (refer to Figure 1). The test vectors used are usually hand generated to target functionality or just random vectors. The main problem in this method is to obtain a test suite that will exercise the design completely. In particular the test suite should be able to expose design errors (bugs) in corner cases or difficult to reach states. This is still a very difficult problem and the fact remains that it is impossible to guarantee that all errors have been discovered in this kind or scenario even for intermediate size circuits. However, with the absence of a viable alternative, simulation based verification is still the primary verification method used in the industry.

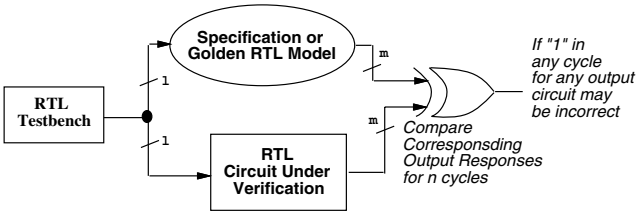


Figure 1: Typical simulation based verification for a l-input m-output RTL circuit

In this paper, we present a technique that can be used to augment simulation based verification. In this method the designer of an RTL circuit embeds a small amount of well understood extra functionality or behavior into the circuit under verification. This extra behavior is inserted into both the golden model/executable specification of the circuit and the also circuit under verification. Note that we assume the existence of such a model without which this technique will not succeed. During simulation based verification this extra behavior is used along with the existing behavior of the circuit to exercise the design more thoroughly. In contrast to traditional formal verification techniques where behavior is reduced by abstraction this method works by slight augmentation of existing behavior. Due to the extra behavior, state space exploration becomes easier and difficult to reach states and corner cases become more easily accessible. W e present three different types of extra behavioral modifications that the designer may use. However, other efficient structures may also be used. Extensive experimental results demonstrate that this technique leads to cutting down the simulation time by more that 50% on an average for a wide range of errors in a number of large industrial RTL circuits. It also helps in exposing more design errors than those detected by simulating on the original behavior alone. W e should emphasize here that some examples used in this paper have thousands of latches and hundred thousand gates which are beyond the scope of current formal verification techniques. There is no problem of scalability for this method for even larger designs. Also unlike traditional formal verification techniques that work on BDD representation of the logic implem entation of the circuit this method can work at the RTL before synthesis is done. Thus it can save a lot of time and effort used to generate the logic level design before bugs can be found. The extra circuitry used can simply be remo ved once the functional verification is complete. More con ve-

Proceedings of the 15th International Conference on VLSI Design (VLSID'02)
0-7695-1441-3/02 $17.00 © 2002 IEEE

niently they can be encased in "synthesis off" pragma directives that synthesis tools allow so that they will never be synthesized.

2. Previous W ork

Though we could not find an exact parallel to the concept that we are proposing here there has been some ideas put forward in similar lines. The need for design for verification techniques to augment current verification methods has been extensively discussed in [1]. From the testability domain the IEEE 1149.1 test bus has been used for testing as well as debug and can be termed as a kind of design for verification [2]. There has been a lot of research on effective formulation and placement of assertions and checkers during simulation [3]-[7]. In fact some verification companies already have products in the market that utilize these ideas [4]. This can also be thought of as a type of design for verification that increases the observability of the system. A technique for automatic placement of assertions has been discussed in context of a particular design environment [6]. In [5] the authors propose an abstraction technique that separates the control-flow of a circuit from the data flow and does validation on this abstracted machine where new state transitions are added. The test vectors generated are tested for validity on the original design. This method requires lot of designer intervention to separate control flow from data flow and this is not a trivial task. Also the examples provided in the paper are small unlike our work. Finally, there has been a lot of research and well established industrial standards in the design for testability (DFT) domain [8]. Though, these techniques deal with manufacturing level stuck-at faults they provide some helpful insights into the design for verification problem.

3. Design for V erification

This section describes the concept behind our verification technique. This technique does not supplant existing verification methods. It supplements them. The existence of a correct golden model or executable specification is assumed so that the simulation results can be verified and errors can be caught.

Suppose we need to check the correctness of a complex, difficult to verify circuit of uncertain functionality. A circuit of known functionality is embedded within this complex circuitry as shown in Figure 2. During verification using simulation, the functionality of the known embedded circuit is used along with that of the circuit under verification to aid in design error detection. W e will demonstrate that many design errors, which would otherwise be very difficult to find, can easily and quickly be detected by using the extra behavior of the embedded circuit.

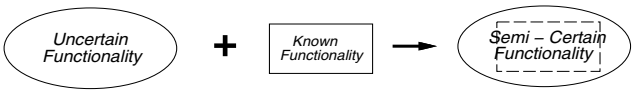
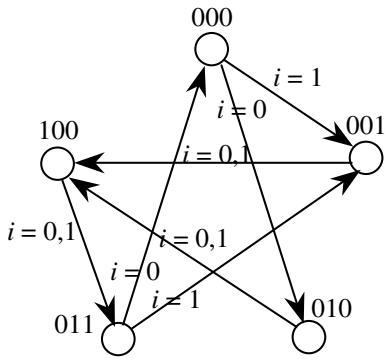
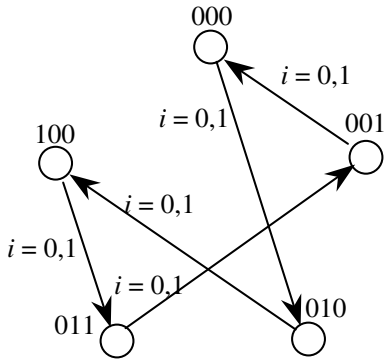


Figure 2: Adding a known ckt. in an uncertain ckt.

Consider Figure 3 which represents the state transition graphs (STG) of the original circuit and the embedded circuit. They have only one input *i*. The state



(a) STG of the original circuit



(b) STG of embedded circuit

Figure 3: STGs of the circuits

transition table of the final embedded system is shown in Figure 4. An extra mode signal, *M*, is added to control which transitions occur. When *M=0* then the original circuit transitions are taken and when *M=1* then the embedded circuit transitions are executed. As can be seen from the table, some of the transitions are common for both the original and the em bedded circuit. For example, *state 000 => state 010* when *i=0* irrespective of the mode signal. Hence by just verifying the embedded circuit transitions we can verify many of the original circuit transitions as well. Specifically for the example shown, 6 out of the 10 original transitions can be verified using this method.

Moreover the state transitions allowed by the extra circuit can result in easier reachability of hard to reach states in the original circuit. If corner case bugs exist which only gets activated in these states then they can be exposed with less difficulty. It is interesting to note that unreachable states may be used as steps to reach the hard to reach states while using the extra behavior. Thus the method might detect an error but may not be able to provide a valid counter example. This is discussed in detail in Section 5.

The scope of this verification technique lies mostly in RTL to RTL comparison or while comparing an RTL implementation with that of a cycle accurate executable specification. The two circuits that are being compared needs to have the same state encoding or similar state encoding so that the extra circuitry that

Present State	Next State			
	$(i,M) = 00$	$(i,M) = 01$	$(i,M) = 10$	$(i,M) = 11$
000	010	010	001	010
001	100	000	100	000
010	100	100	100	100
011	000	001	001	001
100	011	011	011	011

Figure 4: State transition table of embedded system

is introduced does not behave differently in the two circuits. In case of RTL to logic-level comparison of a particular design the scheme may be used only if the state encoding of the circuit remains the same for both the levels and if the flip-flop correspondences between the gate level and RTL circuit is known and also if it is possible to embed the extra circuitry at the gate level. The extra embedded circuit may also be synthesized if the gate-level circuit is derived from an RTL description. However, we believe that in that case it would be best to use boolean equivalence checking after manual mapping of the state elements have been done. This scheme is more suitable for RTL to RTL comparison or a RTL to specification comparison if there exist a cycle accurate executable specification that can be simulated. This method can lead to early detection of bugs even before a full synthesis is done of the design. Also since it is simulation based technique it can scale well to any size designs.

For any given circuit, the problem then is to find the appropriate circuit to embed in it. This circuit should have as many transitions as possible which coincide with the original circuit but with the constraint that it should be easily verifiable. This is a non-trivial problem and we provide some solutions in the next section

4. Some example embedded circuits

In this section some examples of em bedded circuits that we have used in our experiments are provided. This is by no means a complete set of possible circuits that can be used. In fact more efficient circuits may be devised that exercise the design better and is better able to seamlessly integrate into the original design. The goal over here for the new embedded circuit is to share as much as possible of the original behavior and not to perturb the original design too much. In that way there will be less chances of errors being introduced while the extra circuit is being added. Also a large part of the design can be verified while simulating the behavior of the extra circuit which should be correct by construction. It should be possible to verify this extra circuit just by visual inspection for correctness as it should be simple and small.

After the functional verification is complete these extra circuitry may be removed from the RTL description before synthesis. Alternatively these modifications may be encased in a "synthesis off" pargma directive that all RTL synthesis tools allow. Then they will never

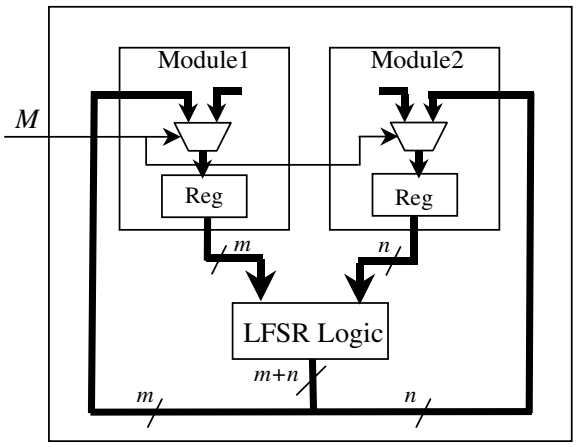


Figure 5: RTL embedding of an LFSR

be synthesized.

Note that some of these behavioral modifications can be done by modifying the test bench to directly load state elements with predetermined values in the middle of the simulation. This is definitely an alternative method of implemen ting this technique. However, this will usually result in complicated and much larger test benches. The simulation will also become slower due to the complex behavior of the test bench. We believe that slight modification of the original RTL is much easier. Also note that in case of designs with tri-state buses, bus contention prevention logic has to inserted to accommodate the extra behavior. This is very similar to what is done for scan design in the testability domain.

4.1 A linear feedback shift register

The first approach taken was to embed a *Linear Feedback Shift Register* in the original circuit. The LFSR is configured with a primitive polynomial. A mode signal, M, is added to the circuit to take either LFSR transitions ($M=1$) or original transitions ($M=0$). A multiplexer at the input of each state element or register is used to choose between the original circuit values and the LFSR values depending upon whether the mode signal M is 0 or 1 respectively. During simulation M is suitably set or reset. Hence the circuit either makes original transitions or LFSR transitions. Since at the RTL, the state elements/registers can be distributed across different modules, the inputs/outputs of the modules have to be configured appropriately so that all the flip-flops of the circuit can be chained together to implement a LFSR. This is illustrated by Figure 5.

Due to the embedded LFSR, the state transition graph now becomes much more dense and the diameter of the graph (longest distance between any two nodes) is also likely to decrease. Hence any given state is much more likely to be reached during random vector simulation.

4.2 Extra read/write ports for memory elemen ts

Another approach taken was to introduce external read/write ports for all memory banks and regis-

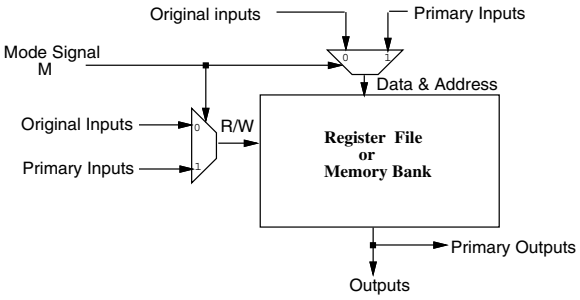


Figure 6: RTL implementation of extra memory ports

Table 1: Circuit size statistics for the example circuits

Circuit	*RTL*		*Logic level*		Modification Scheme
	HDL Lines	Design Type	#Gates	#FFs	
HRCC	837	hier	16205	70	LFSR
EXE	8075	hier	12327	939	LFSR
MCM16	28828	hier	105430	6570	Mem Ports
ALM	3504	hier	8265	1490	Counter Loads

ter files so that all memory locations are controllable/observable from the outside. Again a mode signal and a set of multiplexers are used to activate these external read/write ports. This is shown in Figure 6. This extra circuitry allows the designer to view the intermediate results inside a memory during simulation. Also it allows the configuration of the memory as needed from outside for executing a critical corner case. The extra circuitry required for the modification is quite simple as it should be.

4.3 Parallel loading of counters

Counters are very bad for verification as they are very difficult to control. To set the most significant bit of a counter a sequence of length 2^n is required where n is the bit-width of the counter. To alleviate this problem each counter can be modified with a parallel load from the primary inputs. Easy controllability of a counter can lead to better verification of all downstream logic that the counter feeds. The implementation was similar as before with a set of multiplexers and a mode signal.

5. Problem of false negative and counter examples

Though the above technique is quite promising in detecting errors through simulation, there are some potential disadvantages and some workarounds are necessary to alleviate the problems. These are discussed next.

As stated earlier the extra behavior of the circuit embedded in the original circuit will usually increase the reachable state space in the design. One potential problem is an error being detected in an unreachable state. This may or may not result in a false negative (if the error is again detected in a reachable state then the error is not a false negative). Hence once an error is flagged it needs to be checked whether it is valid or not. In order to do this, the state in which the error is flagged has to be checked for validity. First the state elements (flip-flops/latches) which feed the logic cone where the error resides need to be determined. This can be done by back-tracing from the erroneous output(s). The state residing in the other elements is a don't care and is to be ignored for this error. Once these state elements are found the ones inside them which correspond to control-state registers need to be further separated. The bit patterns residing in these registers need to be checked as valid state encodings in the RTL circuit. This is a simple check which will immediately determine an invalid state if a bit-pattern is an invalid encoding. All memory banks and register files are to be ignored as it is possible to take them to any state using a appropriate number of loads.

If all the above tests pass the registers inside a pipeline are to be checked for consistency as many states inside a pipeline are invalid states. A backward trace using the original behavior from the values present in the registers of the pipeline will catch any inconsistency quickly. Finally the most difficult problem is to verify the validity of different data path register bit-patterns with respect to the different control states and the interaction of control states in different state machines. To alleviate this problem some checkers may be inserted into the RTL circuit that checks for invalid control state combinations in different FSMs. If anyone of these checkers is asserted in the state where the error is detected then the state is invalid. Finally the embedded circuit may be constrained by adding extra circuitry to avoid known invalid states.

Using the above set of rules we believe that it will be possible to filter out errors detected in invalid states in the circuit.

Another problem is the generation of a valid counter example or input sequence that will detect the error in the original circuit. As stated earlier the extra behavior may reach hard to reach states where an error is detected by stepping through unreachable states. Hence the simulated sequence will usually not be useful to redetect the errors using the original behavior. Thus this technique will be useful in detecting errors if they exist. Once they are found, diagnosis and debugging might require extra inspection and effort.

6. Experimental results

In this section, we present the experiments done to validate our technique and the results obtained. We have done extensive simulation runs on four industrial RTL circuits written in VHDL or Verilog. The first one, *HRCC* is a cache coherence controller. *EXE* is a memory controller. *MCM16* is a multi-chip module with lot of embedded memories. *ALM* is a part of an ATM switch and has a number of counters in the design.

The characteristics of the circuits are shown in Table 1. The circuits are synthesized from HDL descriptions using the Synopsys Design Compiler to gate-level netlists. The synthesized results are for information purposes only as they provide a notion of the complexity of the circuits. All the experiments are done at the RTL using an RTL HDL simulator. The last column in Table 1 shows the modification scheme used on the circuits for the design for verification experiments which we elaborate next.

Table 2: Simulation results for Circuit *HRCC*

Error Type	Error no.	# Simulation Vectors Original	Modified
Type i	1	11	15
	2	-	-
	3	-	24
	4	-	30
	5	2	2
Type ii	1	-	831
	2	-	32
	3	6	204
	4	-	277
	5	-	32
Type iii	1	-	-
	2	-	-
	3	-	274
	4	-	-
	5	-	-
Type iv	1	17	10
	2	8	15
	3	13	86
	4	13	471
	5	12	10
Type v	1	-	495
	2	-	666
	3	-	-
	4	-	123
	5	-	309
Average [a]	-	6803	2556

[a] considering all errors and using the total number of vectors simulated for an undetected error to be 10000

Table 3: Simulation results for Circuit *EXE*

Error Type	Error no.	# Simulation Vectors Original	Modified
Type i	1	-	652
	2	-	954
	3	-	321
	4	26	-
	5	-	98
Type ii	1	32	54
	2	17	782
	3	-	654
	4	-	-
	5	-	541
Type iii	1	-	-
	2	45	40
	3	-	87
	4	101	109
	5	-	56
Type iv	1	-	-
	2	56	786
	3	7	19
	4	350	64
	5	-	698
Type v	1	-	56
	2	78	54
	3	-	-
	4	12	65
	5	-	876
Average	-	6028	2276

The experimental methodology is as follows. The original circuit is modified by embedding some extra circuitry in it as described in the previous sections. A *Mode* signal is used to switch between the original behavior and the modified behavior. A random input vector set (*Vector Set1*) is generated for the original circuit. The input vector set for the modified circuit is the same as this random vector set with the *Mode* signal randomly activated (*Vector Set2*). Thus during simulation using *Vector Set2* we are simulating the original behavior and the modified behavior randomly and in an interleaved fashion. These two circuits are now simulated with these input vectors respectively and their output responses captured.

Now an identical error is introduced in both the original circuit and the modified circuit. These erroneous circuits are also simulated with the vector sets *Vector Set1* and *Vector Set2* respectively and the output responses captured. These output responses are then compared with the corresponding good circuit responses to check whether the error introduced has been detected. *i.e.* whether the output responses differ for the good and erroneous circuit in any clock cycle. The number of simulation cycles required to catch the error in either case is noted.

In the RTL circuits the following types of errors were introduced :
i) missing cases in *case* statements.
ii) missing clause in a conditional expression.
iii) missing *assign* statements.
iv) erroneous output values inside case statements.
v) incorrect state transitions inside an FSM

All experiments were done using a random pattern sequence of 1000 vectors. For the first two examples an LFSR was embedded into the circuit comprising of most of the state elements in the circuit. For the third example the memories and register files were randomly loaded from the primary inputs and observed at the primary outputs. For the last example parallel loads were introduced into the four counters present in the circuit. The results are shown in the Tables 2-5.

In Column 1 of the Tables the type of error introduced is shown and this corresponds to the errors discussed in the previous paragraph. Column 2 just provides a counter for each kind of error. In Column 3 the number of simulation vectors required to detect the error in the original circuit is shown. A "-" means the error has not been detected in the 1000 random vector simulation run. In Column 4 the corresponding number is presented for the circuit modified by the design for verification hardware. At the end of the table the average number of vectors required to catch an error in either case is shown assuming a penalty of 10000 vectors for each undetected error. Though this comparison is not very scientific it gives a notion of the overall improvement in simulation run times.

From the tables we can make the following observations. Out of the 100 errors in the different circuits 22 are undetected by both schemes. The number of undetected errors becomes larger for the more complex circuits as is to be expected in a random testing scenario. Only 4 errors are detected in the original circuits that remain undetected in the modified circuits whereas as many as 34 errors are detected in the modified circuits but are undetected in the original circuits. Though a direct comparison is impossible because of so many undetected errors, by looking at the averages we can say with some degree of confidence that the

Table 4: Simulation results for Circuit *MCM16*

Error Type	Error no.	# Simulation Vectors Original	Modified
Type i	1	-	-
	2	-	675
	3	-	-
	4	-	567
	5	98	76
Type ii	1	-	-
	2	675	-
	3	-	832
	4	-	-
	5	-	560
Type iii	1	431	234
	2	-	753
	3	-	-
	4	-	239
	5	87	320
Type iv	1	-	-
	2	765	89
	3	-	-
	4	-	673
	5	-	-
Type v	1	-	340
	2	-	-
	3	-	-
	4	453	679
	5	352	981
Average		7298	4680

Table 5: Simulation results for Circuit *ALM*

Error Type	Error no.	# Simulation Vectors Original	Modified
Type i	1	-	453
	2	32	14
	3	563	474
	4	-	-
	5	19	80
Type ii	1	67	153
	2	5	42
	3	-	76
	4	-	-
	5	95	72
Type iii	1	452	32
	2	315	924
	3	-	79
	4	430	-
	5	65	-
Type iv	1	-	18
	2	211	605
	3	-	790
	4	-	310
	5	453	232
Type v	1	28	85
	2	92	156
	3	-	872
	4	67	13
	5	604	884
Average		3739	1854

simulation time can be shortened by more than 50% by using the design for verification modifications. The probability of an error being detected is also increased significantly by using this technique. Note that this reduction is achieved by random vectors only which are usually quite bad for detecting errors in sequential circuits. More savings may be obtained by directed tests that use the design for verification hardware to control and observe the internals of the circuit better.

In the experimental setup we never encountered the problem of false negatives as we introduced the errors ourselves and if the simulation outputs did not match an error was guaranteed to be present. Also we did not need to do any diagnosis or debugging as we knew the location of the error. At the present time it is therefore difficult to comment on the problem of counter example generation and the amount of effort required after an error has been detected.

7. Conclusions

In this paper we have proposed a novel design for verification technique that can be used to augment simulation based verification for RTL circuits. In this method the designer of an RTL circuit embeds some extra circuitry into the original circuit and uses this during simulation based verification to access corner cases and hard to reach states in the design. Once the functional verification is complete this extra circuitry may be removed from the design. Some examples of embedded circuits that may be used were provided. Experimental results demonstrate the efficacy of the technique where even through random simulation we were able to detect randomly introduced design errors in less than half the time on the modified circuits compared to the non-modified circuits. Some disadvantages of the method like generation of false negatives were also discussed and a possible workaround proposed. As part of future work various other embedded circuits are currently under investigation which might lead to even smaller verification times with less perturbation on the original circuit.

References

[1] D.L. Dill and S. Tasiran, "Embedded Tutorial: Formal verification meets simulation," *Int. Conf. Computer-Aided Design*, pp. 221, Nov. 1999. `http://sprout.stanford.edu/talks.html`

[2] A. Cron, "IEEE 1149.1 use in design for verification and testability at Texas Instruments," *White paper: Texas Instruments*, Nov. 1990. `http://www.edta.com/scribe/reference/appnotes/md003e9f.htm`

[3] M. Pandey, R. Raimi, R.E. Bryant, and M.S. Abadir, " Formal verification of content addressable memories using symbolic trajectory evaluation," in *Proc. Design Automation Conf.*, pp. 167-172, June 1997.

[4] 0-In Design Automation. Inc., "White-box verification for complex designs," *White Paper*, Mar. 2000. `http://www.0in.com`

[5] D. Moundanos, J.A. Abraham, and Y.V. Hoskote, " Abstraction techniques for validation coverage analysis and test generation," *IEEE Trans. on Computer*, Vol. 47-1, pp. 2-14, Jan. 1998.

[6] L.C. Wang, M.S. Abadir, and N. Krishnamurthy, "Automatic generation of assertions for formal verification of PowerPC microprocessor arrays using symbolic trajectory evaluation," in *Proc. Design Automation Conf.*, pp. 534-537, June 1998.

[7] S. Switzer and D. Landoll, "Using embedded checkers to solve verification challenges," in *Proc. DesignCon IP World Forum*, Feb. 2000.

[8] M. Abramovici, M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, New York, 1990.