A Probabilistic Method for the Computation of Testability of RTL Constructs

José M. Fernandes, Marcelino B. Santos, Arlindo L. Oliveira, João C. Teixeira
IST / INESC-ID, R. Alves Redol, 9, 1000-029 Lisboa, Portugal

Abstract

Validation of RTL descriptions remains one of the principal bottlenecks in the circuit design process. Random simulation based methods for functional validation suffer from fundamental limitations and may be inappropriate or too expensive. In fact, for some circuits, a large number of vectors is required in order to make the circuit reach hard to test constructs and obtain accurate values for their testability. In this work, we present a static, non-simulation based, method for the determination of the controllability of RTL constructs that is efficient and gives accurate feedback to the designers in what regards the presence of hard to control constructs in their RTL code. The method takes as input a Verilog RTL description, solves the Chapman-Kolmogorov equations that describe the steady-state of the circuit and outputs the computed values for the controllability of the RTL constructs. To avoid the exponential blow-up that results from writing one equation for each circuit state and solving the resulting system of equations, an approximation method is used. We present results showing that the approximation is effective and describe how the method can be used to bias a random test generator in order to achieve higher coverage using a smaller number of vectors.

1. Introduction

The steady growth in complexity of integrated circuits and the need to reduce the time to market of products has contributed to increase the percentage of time spent in circuit verification. This verification is important both in the design phase (functional verification) and in the post- manufacturing test phase (defect testing).

In both cases, the existence of appropriate test vectors is critical to ensure defect free circuits and to avoid the need for costly re-design cycles.

There are essentially two approaches for the verification of RTL descriptions: simulation based methods and formal verification methods. Simulation based methods try to exercise all parts of the circuits by using a high number of vectors, obtained either by using knowledge of the design or by using some pseudo-random test vector generator. Formal based methods can be used to verify RTL descriptions against original specifications of the circuit, sometimes obtained from behavioral descriptions.

Simulation based method require the existence of appropriate test vectors. Regrettably, automatic generation of test vectors at higher abstraction levels [2] for complex designs remains an open problem, although significant advances have been made in this field [5][6]. The key problem is that random vectors don't exercise adequately the hard to reach conditions that lead to the execution of the *dark* spots in the design, and efficient algorithms for sequential test pattern generation are unlikely to exist since the problem is known to be PSPACE-complete [7].

Verification based approaches, on the other hand, require the existence of formal higher level specifications that are not always available. Furthermore, algorithms for formal verification of sequential circuits are also inherently complex, although advances in heuristics have made them applicable in a wider range of designs [10].

The present work addresses this problem by proposing an approximate statistic modeling approach that obtains accurate estimates of the controllability (and, in future phases, of the observability) of RTL constructs. These estimates can be used to improve the design and test processes in a number of ways.

In the design phase, they can be used to inform the designer that a given construct is not being adequately tested and that it may require changes or some other manual intervention. Our approach is specially appropriate for this type of intervention, since there is a very close connection between the RTL constructs and the internal model signals whose testability is being evaluated. At this level, it can also be used to bias the generation of randomized tests, in order to achieve adequate functional coverage of hard to test constructs.

At the defect testing phase, these testability metrics can be used in a number of ways to improve the design and test processes. Test preparation cost can be reduced if designs for testability (DFT) techniques are used before the test generation phase. These DFT techniques, like built-in self-test (BIST), test point insertion (TPI) or scan can have their impact on the circuit minimized in terms of area overhead and performance degradation if testability metrics are used to guide them.

1530-1591/04 $20.00 (c) 2004 IEEE

The use of testability measures to direct the generation of randomized vectors, either by biasing the random number generators [13] or by considering the existence of bit masks [11] has also been proposed. This represents another significant application of the techniques proposed in this work.

Our approach is based on the statistical modeling proposed by Fallah et al. [3][4] but addresses the controllability of structures directly at the RTL level, while the original work accepts a post-synthesis logic level description. In this way, the feedback given to the designer is easier to use, since testability results are given in terms of RTL constructs and not in terms of post-synthesis logic nodes.

2. Definitions and Basic Concepts

During the design process several abstraction levels are normally used to achieve rapid development of digital circuits. From the algorithmic to the physical level, design testability assessment is an important issue. Accurate assessment of the testability of a given part of a design is important not only because it avoids problems in the production testing phase, but also because it makes sure that the design is being properly tested from a functional point of view. In particular, the existence of *dark spots* in the design, i.e., blocks or constructs that are not being properly exercised is to be avoided in an agile development process.

In this work, we model the behavior or the RTL description of a circuit using a discrete time Markov chain. For the most generic case, that of sequential circuits, we are interested in the circuit behavior in the steady-state. This is important because, for sequential circuits (e.g., counters, shift-registers, control systems), many constructs only become active and exercisable when a given set of conditions is met. By computing the steady-state behavior of a circuit, we make sure that the probabilities of occurrence of events are calculated with precision, even if the events are rare or depend on very specific input conditions.

Simulation based methods need to use a very high (sometimes prohibitively high) number of vectors to make sure that even rare events occur a sufficient number of times to obtain accurate measures of the testability of the constructs. By using a statistical approach, we avoid the need to use a very high number of vectors, while still obtaining accurate estimates, even for rare events. The method described here solves the Chapman-Kolmogorov equations that describe the steady state behavior of the circuit, and calculates the probability associated with each state of the Markov chain. To reduce the computation effort and cope with large designs we use symbolic representation methods that describe the circuit function using Binary Decision Diagrams (BDD).

Figure 1 represents the general scheme of a generic synchronous sequential circuit. Its behavior can be represented by a transition graph, modeled by a tuple $(\Sigma, \Delta, Q, q_0, \delta, \lambda)$ where $\Sigma \neq 0$ is a finite set of input symbols, $\Delta \neq 0$ is a finite set of output symbols, $Q \neq 0$ is a finite set of states, $q_0 \in Q$ is the initial reset state, $\delta : Q \times \Sigma \rightarrow Q$ is the transition function, and $\lambda : Q \times \Sigma \rightarrow \Delta$ is the output function.

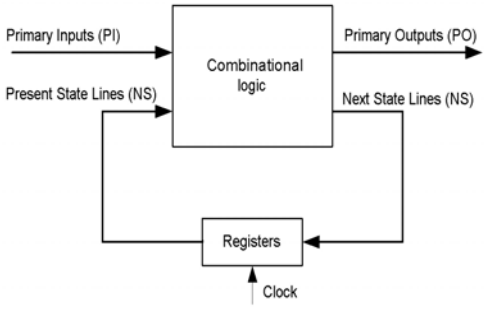

Fig. 1: Scheme of a generic synchronous sequential circuit

Under rather general assumptions, the system can be modeled by a Markov chain and the equations that describe the steady state probabilities of each state in the transition graph (the Chapman-Kolmogorov equations) have a unique solution.

Consider a binary variable $y_j = f_j(x_1, x_2, ..., x_n)$, represented in terms in its inputs. The *positive* and *negative cofactors* of f_j with respect to variable x_i, are defined by:

$$f_j(x_i = 1) = f(x_1, ..., x_{i-1}, 1, x_{i+1}, ..., x_n)$$
$$f_j(x_i = 0) = f(x_1, ..., x_{i-1}, 0, x_{i+1}, ..., x_n)$$

The probability of the y_k variable being one is equal to its conditional probability $P(y_k)$ given the values of its inputs. It can be computed using the Shannon decomposition:

$$P(y_j) = \sum_{k \in \{0,1\}} P(x_i = k) \cdot P(f_j(x_i = k)) \quad (1)$$

By using a dynamic programming approach and traversing the BDD in a bottom-up fashion, the static probabilities can be computed in time linear on the size of the corresponding BDD. We will use this property of BDD representations to efficiently compute the probability of a node in the circuit being in a given state. From this probability, the computation of the controlability of that node is straightforward.

3. Parsing and Analysis of RTL Description

The proposed probabilistic analysis tool, named ASCOPA – **A**utomatic Static **C**ontrolability/**O**bservability **P**robabilistic **A**nalysis tool, accepts as entry an HDL Verilog description.

The Verilog subset is based on the IEEE 1364 1995 and IEEE 1364.1 1999 norms [16][17]. The system can process a commonly accepted verifiable subset of Verilog keywords that includes the most frequently used control structures and operators. The parsing process was implemented using a customized version of a research tool [1], based on widely available parsing tools. In the sequence we describe briefly the steps that are performed to process a Verilog description into an internal implicit representation of the state transition graph of the circuit.

From the Verilog parsing, the Control and Data Flow Graphs are generated. Each node of the Control Flow Graph (CFG) is associated with a Data Flow Graph (DFG). The description can have a hierarchy of several module instantiation statements. Figure 2 represents an example of a CFG for a Verilog description for one module instance with two concurrent statements.

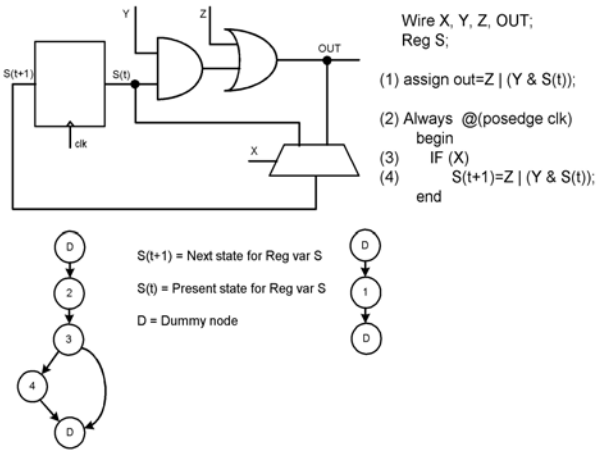

Fig. 2: Behavioral Verilog Description and CFG's

By traversing this data representation, a module instantiation graph tree is build. The same module can be instantiated more than one time. For each module instance, a new context is created for the variables of that instance. Next, and only once for each description module, the CFG´s for the concurrent statements are built, preserving the hierarchical structure of the behavioral description. In Figure 2 two CFG's are generated: one for the concurrent *assign* statement, and another for the *always* statement. For each module instance variable (Net or Register) the variable propagation and influence lists are created by traversing the module instantiation graph tree in a hierarchical way. The variables are then associated with an hierarchical name.

Each node of the CFG is associated with a Verilog statement. Two types of nodes are considered: arithmetic expression nodes and Boolean expression nodes. Each node corresponds to a particular Verilog construct in the code, such as an IF, CASE or an Always control structure.

By traversing the statements of the Verilog description, every CFG is traversed in breadth first order and a BDD representation is built for each bit of each variable. In this way, every variable (Net, Register or port) of every module instance is represented by an array of Boolean functions, one for each bit of the variable, represented as BDDs in terms of the primary input and state variables. Ports are considered Nets, in what concerns expression evaluation crossing the borders of an instance. A variable dependence graph (at the bit level) is created concurrently while traversing the Verilog description.

During the generation of the Boolean function representation, each CFG node of a specific instance becomes associated with an array pointing to all the variables of that instance that have been evaluated on that path. For the moment, two possible situations that involves expression evaluation are considered, depending on the CFG node type:

Assignment nodes have a CFG associated with the expression in the right-hand side (RHS) of the statement that depends on the individual functions associated with every variable (Net or Register) of the RHS expression, After node evaluation, the array associated with every LHS variable bit is updated with the new values.
Condition nodes have a CFG associated with the condition expression that depends on the individual functions associated with every variable (Net or Register) bit of the condition expression, that were calculated in the predecessor nodes. Two types of condition nodes are considered:
IF nodes, in which the condition that affects every assignment inside the "then" path is the complement of the "else" path condition, if it exists.
CASETAG nodes in which the condition is computed by the bit-wise XNOR between the CASE condition and the CASETEG condition.

The same bit variable may be assigned in one or more condition paths. If a bit is not assigned in any of the condition paths, or if the condition path is inexistent, it must keep the value as it was before the IF or CASE clause for the non assigned or inexistent paths. This is the case of the CFG for the IF statement in Figure 2. If a bit of a Net variable includes an expression that requires evaluation (either RHS of an assignment or a condition) and it has not been assigned yet, it must be subject to a future composition in every evaluated left-hand side (LHS) bit expression or condition that depends on it. In this way, every Net or Register variable, after the BDD generation process, becomes a function only of primary inputs and Register variables. It is important to stress that the data structures are generated without loosing the hierarchical information, keeping all the time a direct connection to the lines of the RTL description.

This makes it possible to relate directly the controllability of internal nodes with the original RTL constructs.

4. Probabilistic Analysis of Symbolic Networks

After processing the Verilog description as described in the previous section, it is possible to proceed to a symbolic probabilistic analysis of the circuit.

Many circuits of interest described at the register transfer level exhibit structures that lead to very deep state transition graphs. For instance, a 16 bit counter with a reset signal cannot be reasonably tested by random patterns, since the reset signal needs to be held inactive for a long period in order to let the count proceed. Structures like this lead to hard to test parts of the RTL description, and can only be adequately tested if the system automatically identifies the hard to test constructs and identifies the reasons for their lack of controllability or observability. For instance, in this case, the terminal count signal, and any parts of the RTL code that depend on that signal, will be not be well tested unless the reset signal is actuated with very low probability and/or appropriate test points are inserted in the circuit.

The probabilistic method can be used not only to compute the controllability and observability of internal signals but also to identify which signals (or combinations of signals) should be held preferentially at specific values to achieve higher levels of testability of the hard to test points.

Consider a circuit with N registers. The state space $Q = \{q_0, q_1, ..., q_{M-1}\}$ to explore has, in the worst case, $M = 2^N$ possible states. Assume that the system is in state q_i. Under the Markov assumption, the transition between state q_i and q_j occurs with probability π_{ij}, under some specific, time invariant, input distribution. Then, in steady state, the probability of state q_i is given by:

$$P(q_j) = \sum_i \pi_{ij} P(q_i) \qquad (2)$$

The direct solution of this system of M equations (the Chapman-Kolmogorov equations) is not feasible for the majority of circuits of interest, that have in general more than 20 state variables, although special purpose techniques can be applied to solve them exactly for some circuits with more than 2^{30} states [8].

Since exact solution of this system of equations is impracticable for N greater than 20, we have followed an approximation method that assumes independence of the state lines, ignoring correlations between them. Although this is known to be an approximation that, in some cases, shows significant deviations from the exact values it reduces exponentially the computation requirements, since only N values need to be computed instead of 2^N. Consider that state q_i is encoded with values $s_i^1, s_i^2, ... s_i^N$ in the state lines. Under the state line independence assumption [9][14], the probability of the system being in state q_i is given by $P(q_i) = \prod_{j=1}^{N} P(s_i^j)$, where $P(s_i^j)$ is the static probability of line state j in state i.

In this way, the resolution of the Chapman-Kolmogorov equations (2) is reduced to the problem of finding the solution of a non-linear system of N equations, one for each state line. The state lines s^j are themselves Boolean functions of the primary inputs, x^l, and the previous values of the state lines, r^j:

$$\begin{aligned} s^1 &= f_1(x^1, x^2, ..., x^L, r^1, r^2, ..., r^N) \\ &\dots \\ s^N &= f_N(x^1, x^2, ..., x^L, r^1, r^2, ..., r^N) \end{aligned} \qquad (3)$$

In terms of probabilities, this can written as

$$\begin{aligned} P(s^1) &= P(f_1(x^1, x^2, ..., x^L, r^1, r^2, ..., r^N)) \\ &\dots \\ P(s^N) &= P(f_N(x^1, x^2, ..., x^L, r^1, r^2, ..., r^N)) \end{aligned} \qquad (4)$$

The value of the right hand sides in equation (4) can be readily computed from the BDD representations of the next state variables, obtained as described in section 3, given values for $P(r^j)$ (obtained from the previous step computation of $P(s^j)$) and $P(x^l)$. In steady state, the probability of the next state variables is equal to that of the previous state variables.

$$P(s^i) = P(r^i) = p_i \quad , \quad i \in [1, N] \qquad (5)$$

The set of equations (4) is solved using a fixed point method, which, in this case, corresponds to the Picard-Peano method. Starting with an initial value for the latch probabilities p_i, the process iterates the application of expressions (4) until a fixed point (5) is reached, where the differences in probability of the state lines between two consecutive iterations falls below a threshold. Alternatively, this non-linear system of equations can be solved using the Newton-Raphson method. Once the values of p_i are known, the controllability of a circuit node to 0 and to 1 is given by:

$$\begin{aligned} C(y^i = 0) &= 1 - P(y^i) \\ C(y^i = 1) &= P(y^i) \end{aligned} \qquad (6)$$

For the example of Fig. 2, Figure 3 represents the Markov chain with the transition probabilities between the states of the circuit, under a uniform distribution of the inputs.

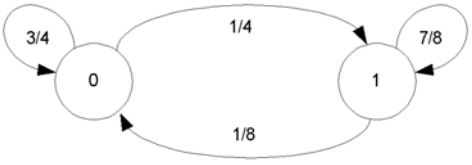

Figure 3: Markov chain for the example circuit

This chain is associated with the transition matrix

$$\prod = \begin{bmatrix} \pi_{00} & \pi_{10} \\ \pi_{01} & \pi_{11} \end{bmatrix} = \begin{bmatrix} 0.75 & 0.125 \\ 0.25 & 0.875 \end{bmatrix}$$

In this particular case, the solution gives $P(s) = 0.667$, which, in this case, gives exactly the values of the state probabilities (since there is only one register, $P(q_1) = P(s^1)$)

$$\begin{bmatrix} P(q_0) \\ P(q_1) \end{bmatrix} = \begin{bmatrix} 0.3333 \\ 0.6667 \end{bmatrix}$$

The controllability of circuit nodes under the assumptions of uniform distribution of the inputs (and the RTL constructs that they correspond to) can now be directly computed. For example, the computation of $C(Out)$ using expressions (6) and (1), in this case, is equivalent to the computation of:

$$C(Out = 1) = P(z) + P(s^1 \wedge y \wedge \bar{z}) = 0.667$$

The existence of an implicit closed form expression that gives the controllability of an RTL construct as a function of the probabilities of the input signals opens the possibility of changing the test procedure in one of the ways addressed in section 1. In this particular case, if one is interested in the somewhat harder to test case of the output stuck at 0, this result leads to the indication that the probability of variable z should be increased above its default value of 0.5.

5. Results

The ITC99 benchmark circuits [15] were used to evaluate the precision of the controllability measures obtained by the new tool. A prototype of the developed system was integrated within the SIS framework [12] and used to test the approach.

In order to obtain the simulation based testability metrics, the PLI (Programming Language Interface) of a commercial Verilog simulator was used to implement a task that evaluates controllability and detectability. This task reports the number of activations and detections of each LSA fault. Faults are injected in all bits of the RTL description. When using a large number of random vectors, the number of activations of the LSA0/1 fault is proportional to the controllability to 1/0. For the benchmark circuits used in these experiments the simulation-based controllability was calculated as the average of the controllability obtained with five sets of 5000 random vectors.

Table 1 shows the percentage of nodes for each circuit that show deviations in the controllability values, when compared with the values obtained by random simulation with 5000 vectors.

Table 1: Percentage of nodes with controllability deviations within the indicated ranges

% Dev	b01	b02	b03	B04	B06	b09	b10	b13
<0.1	85,7	100,0	70,6	72,7	54,5	96,6	74,2	76,2
[0.1,0.3]	14,3	0,0	29,4	27,3	45,5	3,4	25,8	14,3
[0.3,0.5]	0,0	0,0	0,0	0,0	0,0	0,0	0,0	6,3
[0.5,0.7]	0,0	0,0	0,0	0,0	0,0	0,0	0,0	3,2
[0.7,0.9]	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
>0.9	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0

Figure 4 shows a graphic representation of the results in Table 1 and makes it clear that a very small number of nodes show significant discrepancies. Note that the deviations obtained may have as origin the approximations made in this approach, but may also result from imprecise results obtained from the limited length simulation runs.

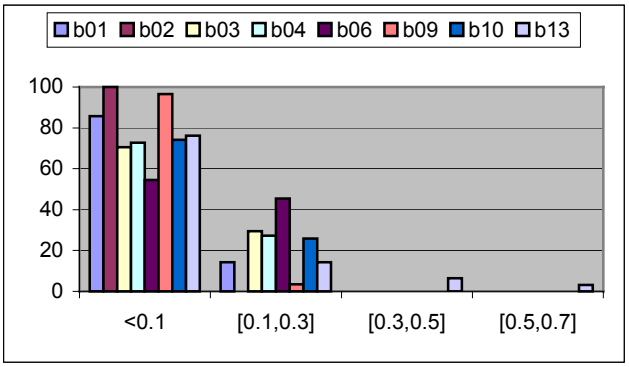

Figure 4: Percentage of nodes with deviations in range.

5.1 Automatic Mask Generation

The automatic mask generation tool [11] was used to generate test masks for all the nodes of the benchmarks with probability value higher than 0,95 (low controllability to "0") and lower than 0,05 (low controllability to "1"). The second column of Table 2 shows the number of nodes with low controllability for each circuit. This table shows also the number of registers (#regs) and the number of primary inputs (#PIs).

Table 2: Number of registers that require partial-scan.

Circuit	#bits with Contr<0,05	#regs	#Pis	#masked registers	#masked PIs
b03	15	30	6	9	1
b04	10	86	13	7	1
b06	1	9	4	3	2
b09	22	28	3	19	1
b10	9	20	13	4	3
b13	25	63	12	25	1

For each one of the targeted bits, a set of masks was generated that force specific values on the register and primary input variables. The last two columns of Table 2 show the number of registers and primary inputs that occur at least once in one of the generated masks.
Control of the targeted nodes can be achieved including a partial-scan chain with the registers forced by masks. The required length for each chain, in these examples, is in average only 30% of the required length for full-scan.

Table 3: Masks for control of uscite[1] in b06

Eql	Reset	cont_eql	uscite[1]	uscite[0]	cc_mux[1]	cc_mux[0]	ackout	state[2..0]	enable_count
x	1	x	x	x	x	x	x	Xxx	x
x	0	x	x	x	x	x	x	000	x
1	0	x	x	x	x	x	x	001	x
0	0	x	x	x	x	x	x	001	x

.
Table 3 shows four masks that were automatically generated in order to enable controllability of four possible nodes for *uscite[1]* in circuit b06. Besides the *clock* input this circuit has three additional primary inputs: *eql*, *reset* and *cont_eql*. The generated masks require control over two of them. From the 9 registers of b06 only three of them have to be included in a partial-scan chain in order to ensure controllability of *uscite[1]*.

6. Conclusions and Future Work

In this work we presented the first non-simulation based approach that efficiently computes controllability of RTL constructs as a function of the probability distribution at the inputs. The approach uses a simplifying assumption regarding the probability of states in the steady state that has been shown to be efficient and effective in the computation of the state probabilities and the testability measures of constructs in the Verilog description.

This ability to obtain a precise controllability measure of RTL constructs can be used in an effective way to improve the testability of the design. In the immediate future work, we will apply the system to improve BIST systems, by modifying the probabilities of input signals in order to improve the testability of the difficult to test parts.

References

[1] Cheng S. T., Brayton R. K., "Compiling Verilog into Automata", Department of Electrical Engineering and Computer Science, University of California, Berkeley, TR. UCB/ERL M94/37, May 1994.

[2] Cho C. H., Armstrong J. R., "B-Algorithm: A Behavioral Test Generation Algorithm", Proc. IEEE International Test Conference (ITC), pp. 968-979, 1994.

[3] Fallah F., Devadas S., Keutzer K., "OCCOM: Efficient Computation of Observability-Based Code Coverage for Functional Verification ", Proc. of the 34th Design Automation Conference (DAC), pp. 152-157, June 1998.

[4] Fallah F., Ashar P., Devadas S., "Simulation Vector Generation from HDL Descriptions for Observability-Enhanced Statement Coverage", Proc. of the 35th Design Automation Conference (DAC), pp. 666-671, 1999.

[5] Ferrandi F., Fummi F., Sciuto D., "Implicit Test Generation for Behavioral VHDL Models". Proc. IEEE International Test Conference (ITC), pp. 587-596, 1998.

[6] Ferrara G., Ferrandi F., Fin A., Fummy F., Sciuto D., "Functional Test Generation for Behaviorally Sequential Models ", Proc. of the Design Automation and Test in Europe Conference (DATE), pp. 403-410, March 2001.

[7] Freitas A. T., Neto H. C. and Oliveira A. L.. "On the complexity of power estimation problems". In International Workshop on Logic Synthesis (ILWS), pages 239-244, June 2000.

[8] Freitas A. T. and Oliveira A. L., "Implicit resolution of the Chapman-Kolmogorov equations in sequential circuits : An application in power estimation", Proc. of the Design Automation and Test in Europe Conference (DATE), pp. 764--769, March 2003.

[9] Hachtel G.D., Macii E., Pardo A., Somenzi F., "Markovian Analysis of Large Finite State Machines", IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems", Vol. 15, Num. 12, pp. 1479-1493, December 1996.

[10] Kern C., Greenstreet M.R., "Formal Verification In Hardware Design: A Survey", ACM Transactions on Design Automation of Electronic Systems, 4:2, pp. 123-193, 1999.

[11] Santos M.B., Fernandes J.M., Teixeira I.C., Teixeira J.P., "RTL Test Pattern Generation for High Quality Loosely Deterministic BIST", Proc. of the Design Automation and Test in Europe Conference (DATE), pp. 994--999, March 2003.

[12] Sentovich E. M., Singh K. J., Lavagno L. , Moon C., Murgai R., Saldanha A., Savoj H., Stephan P. R., Brayton R. K. and Sangiovanni-Vincentelli, A., "SIS: A System for Sequential Circuits Synthesis", Electronics Research Laboratory, Memorandum No. UCB/ERL M92/41, Department of EECS, University of California, Berkeley, 1992.

[13] Tasiran S., Fallah F., Chinnery D., Weber S., Keutzer K-, "A Functional Validation Technique: Biased-Random Simulation Guided by Observability-Based Coverage", Proc. of the 2001 IEEE Int. Conference on Computer Design: VLSI in Computers & Processors (ICCD), pp. 82-88, 2001.

[14] Tsui C. Y., Monteiro J., Pedram M., Devadas S., DespianS., Lin B., " Power Estimation Methods for Sequencial Logic Circuits", IEEE Transactions on VLSI Systems, 3(3):404-416, 1995.

[15] CMUDSP benchmark (I – 99 - 5, ITC 99 5), http://www.ece.cmu.edu/~lowpower/benchmarks.html

[16] IEEE Standard 1364-1995, "IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language", IEEE, Inc., New York, NY, USA, October 14, 1996.

[17] IEEE Standard P1364.1/D1.4, "Draft Standard for Verilog Register Transfer Level Synthesis", IEEE, Inc., New York, NY, USA, April 26, 1999.