Coverage-driven Stimuli Generation

Shuo Yang* Robert Wille* Daniel Große* Rolf Drechsler*†

*Institute of Computer Science
University of Bremen, 28359 Bremen, Germany

†Cyber-Physical Systems, DFKI GmbH
28359 Bremen, Germany

{shuo,rwille,grosse,drechsle}@informatik.uni-bremen.de

***Abstract*—Simulation-based verification is still one of the most important methods to validate the correctness of System-on-Chips. Here, explicitly specified stimuli need to be generated which trigger certain scenarios of the design. However, so far stimuli generation is mainly performed independently of the desired coverage. In this work, we propose approaches for coverage-driven stimuli generation. Despite a naïve method, we introduce and discuss automatic and interactive methods for an improved stimuli generation. We show that explicitly considering coverage metrics leads to smaller and complete sets of stimuli.**

I. Introduction

The continuing improvements in fabrication technology that persisted over the last 30 years enabled the integration of more than 1 billion transistors in *System-on-Chip* (SoC) designs. The development of SoCs of such complexity leads to enormous challenges in *Computer-Aided Design* (CAD). In this context, particularly verification, i.e. ensuring the functional correctness of a design, is crucial.

Formal methods e.g. for equivalence checking (see e.g. [1], [2]) and property checking (see e.g. [3], [4]) suffer thereby from high computational costs and, thus, are not applicable to complex designs yet. Consequently, simulation-based verification (see e.g. [5], [6]) is still the most frequently used verification technique in industry.

However, using simulation-based methods, the whole functionality of a *Design Under Verification* (DUV) can usually not exhaustively be verified. Hence, explicitly specified stimuli patterns are applied to the design. They are e.g. manually provided by the verification engineers, randomly generated, or determined by techniques like constraint-based random simulation (see e.g. [6], [7], [8]). The purpose of these patterns is to explicitly stimulate dedicated scenarios (in particular corner cases) and, afterwards, compare the resulting responses with the expected results.

To evaluate the applied stimuli and to ensure whether the desired scenarios have sufficiently been triggered, coverage metrics are applied (see e.g. [9]). A coverage metric can be associated to a design's functionality, structure, or source code. It can serve as a termination criterion for a simulation-based verification process and indicates whether a DUV has sufficiently been triggered or not. Adequate verification results are obtained when the respectively applied coverage metric is satisfied.

Today, stimuli generation methods in combination with proper coverage metrics are established concepts in the validation of complex SoCs. But so far, stimuli generation is mainly performed independently of the desired coverage. That is, stimuli are generated without taking the coverage status of previously generated stimuli into consideration. Exceptions of this include e.g. the work presented in [10], [11]. However, both considered very specific applications scenarios (embedded software in case of [10] and a pipelined processor in case of [11]) and do not provide a generic coverage-driven stimuli generation scheme. Somewhat closer to that comes the work presented in [12]. But here, not the coverage of particular scenarios is considered. Instead, stimuli are generated using correlations in the toggling activity of signals.

In this work, we consider coverage-driven stimuli generation with respect to dedicated scenarios. We propose a methodology that focuses on user-defined scenarios and aims for the generation of a *small* set of stimuli by exploiting the respective coverage metric provided by the verification engineer. For this purpose three approaches are proposed:

- Naïve: Iteratively generate stimuli triggering at least one scenario until all scenarios are sufficiently covered
- Iterative: Additionally consider only those scenarios which are not sufficiently covered
- Interactive: Additionally exploit design knowledge (in combination with visualization methods) to refine the constraints applied during the constraint-based random simulation

In a case study, the proposed approaches are evaluated. This evaluation shows that the consideration of coverage metrics during stimuli generation significantly affects the resulting size of the set of stimuli. While the naïve approach leads to a very large number of stimuli, a much smaller set can be obtained when the iterative approach is applied. Moreover, interaction of the verification engineers enables further reductions.

II. Problem Formulation

This section briefly formalizes the context and provides a definition of the problem to be addressed.

Since in general not all possible stimuli can be simulated on the DUV, dedicated stimuli are generated to trigger certain (user-defined) scenarios of the DUV. Such a scenario is defined as follows:

Definition 1. *A* scenario S_i *($0 \leq i < n$) is a Boolean function over variables from the set of DUV signals. For the specification of a scenario, a constraint is formulated by using the typical HDL operators such as e.g. logic AND, logic OR, arithmetic operators, and relational operators.*

In the following, the terms scenario and constraint are used interchangeably. The set of scenarios is denoted by $S = \{S_0, \ldots, S_{n-1}\}$.

Generated stimuli are supposed to trigger at least one scenario, which formally means:

Definition 2. *A stimulus satisfies at least one scenario if the following formula evaluates to true:*

$$\bigvee_{i=0}^{n-1} S_i \tag{1}$$

Example 1. *Consider an ALU example, where the ALU has the typical select input s to define the operation to be performed on the data inputs a and b. Possible scenarios are for instance $S_0 = (s = 0)$ (triggering an addition) and $S_1 = (s = 1)$ (triggering a subtraction). Then, the stimulus $a = 23, b = 12, s = 1$ satisfies a scenario (here S_1), while $a = 23, b = 12, s = 3$ does not.*

Since the number of triggered scenarios is important, we define:

Definition 3. *Given the DUV and a set S of scenarios. For each scenario $S_i \in S$, the* coverage status c_{S_i} *denotes the total number of stimuli which have triggered the scenario S_i.*

The goal in simulation-based verification is to generate stimuli (and also check their response) so that all scenarios S are sufficiently covered. The term "sufficiently" is thereby user-defined via a threshold value. This is captured as:

Definition 4. *For each scenario S_i ($0 \leq i < n$), a* threshold t_{S_i} *is defined by the verification engineer. A scenario is considered "sufficiently" covered iff S_i is triggered by at least t_{S_i} different stimuli, i.e. iff $c_{S_i} \geq t_{S_i}$.*

Simulation-based verification terminates, when a sufficient number of stimuli has been generated:

Definition 5. *A set of stimuli is considered* sufficient *iff it sufficiently covers all scenarios of a given set S. In the following, sufficient sets of stimuli are denoted by S_{suff}.*

Due to various methods used for simulation-based verification, the nature and, particularly, the size of a sufficient set S_{suff} of stimuli varies significantly. In general, the size of S_{suff} should be as small as possible since this reduces the time for both, stimuli generation and subsequent simulation. However, how to efficiently generate compact sufficient sets of stimuli has hardly been considered in the past. Motivated by this, we address the following research question in this paper:

> *How can we efficiently determine a set of stimuli S_{suff}, so that each scenario S_i is triggered by at least t_{S_i} different stimuli and, at the same time, keeps the size of S_{suff} small?*

In the next section, solutions for this problem are proposed.

III. General Idea

To efficiently determine a set S_{suff} of stimuli, constraint-based stimuli generation [6] is exploited. Here, stimuli are generated from a given set of constraints by means of a constraint solver, i.e. stimuli are determined by the solver which satisfy the provided constraints. However, the nature of the provided constraints will affect the resulting size of the set of stimuli.

In this work, we present and evaluate different approaches of constraint-based stimuli generation in order to solve the problem stated above. The general idea of each approach is briefly outlined in the following. Afterwards, details on the respective implementations are provided in the next section. In total, we distinguish between three different approaches:

1) *Naïve Approach:*
   The naïve approach simply generates stimuli using the constraint introduced above to satisfy at least one scenario, i.e.
   $$C = \bigvee_{i=0}^{n-1} S_i. \tag{2}$$
   That is, all scenarios are always considered until all of them have sufficiently been covered. While this method is simple and can easily be applied, it is not well placed to reduce the size of S_{suff}. In fact, since the constraint can already be satisfied by a single sufficiently covered scenario, no effective "guidance" of the solver takes place, i.e. scenarios poorly covered so far are not explicitly addressed.
2) *Iterative Approach:*
   An improvement of the "naïve approach" can be gained by additionally considering the threshold t_{S_i} of each scenario S_i. For this purpose, an iterative approach is proposed. At the beginning, the complete constraint is considered as done in the naïve approach via Equation 2. In contrast to the naïve approach, after certain iterations an analysis on the sufficiency of the determined set of stimuli with respect to the thresholds of the scenarios is performed. All scenarios which already have been sufficiently covered by the stimuli generated so far are removed. Overall, the constraint is modified iteratively to
   $$C = \bigvee_{i=0}^{n-1} (S_i \wedge \overline{(c_{S_i} \geq t_{S_i})}). \tag{3}$$
   By this, only those scenarios are considered any further for which still stimuli need to be generated. This procedure continues until no insufficiently covered scenario remains, i.e. until Eq. (3) evaluates to 0.
3) *Interactive Approach*
   Finally, we also propose to make use of the design knowledge of the verification engineer. In fact, manual interaction and manual revisions of the constraint can help to further guide the solver through the constraint-based stimuli generation process. For this purpose, we propose new monitoring and analysis methods which aid the designer in understanding the scenarios still left to be triggered. Based on this, revisions and adjustments on the constraint can be performed which, eventually, lead to a better search for more appropriate stimuli directly addressing the still not sufficiently covered scenarios. This works as follows: For the set of all scenarios $S = \{S_0, ..., S_{n-1}\}$ a subset of insufficiently covered scenarios is selected and replaced by a new one such that these scenarios can be triggered simultaneously. Formally, the scenario subset $R = \{S_{i_0}, ..., S_{i_m}\} \subset S$ is substituted by a new scenario R' and the constraint for stimuli generation is
   $$C = (R' \wedge \overline{(\bigwedge_{0 \leq j \leq m} c_{s_{i_j} \geq t_{s_{i_j}}})}) \vee \bigvee_{\substack{i=0 \\ i \neq i_0, ..., i \neq i_m}}^{n-1} (S_i \wedge \overline{(c_{S_i} \geq t_{S_i})}). \tag{4}$$
   For a correct substitution we thereby need to guarantee that, if the new constraint R' evaluates to true, each substituted scenario S_{i_j} $(0 \leq j \leq m)$ also evaluates to true, i.e.
   $$R' \Rightarrow S_{i_j} \text{ for each } S_{i_j} \in R. \tag{5}$$
   For instance, a new scenario R' may combine two scenarios which is possible if they are independent. This allows to simultaneously check both scenarios.
   Overall, now the generated set of stimuli guarantee to drive the coverage of $\{S_{i_0}, ..., S_{i_m}\}$ closer to their respective thresholds simultaneously. As a consequence, less stimuli are needed to produce a set S_{suff}. This procedure continues until no insufficiently covered scenario remains.

All approaches have their respective advantages and disadvantages as illustrated later in Section V.

IV. Implementation

The implementation of the proposed approaches are detailed in this section. First, the automatic approaches, i.e. the naïve approach and the iterative approach, are described. Afterwards, the interactive one is detailed.

A. Automatic Approaches

The implementation of the naïve approach is straightforward. Simply the complete constraint (Equation 2) is passed to the solver which generates stimuli until all scenarios have sufficiently been triggered. Except for the threshold checks, this requires no further analysis or reformulation.

In contrast, the iterative approach includes a periodical analysis of the threshold values t_{S_i} and a corresponding refinement of the complete constraint. Any scenario $S_i \in S$ that has been triggered by at least t_{S_i} different stimuli is removed from further consideration. In order to satisfy this goal, the *coverage status* c_{S_i} for each scenario S_i must be monitored.

The pseudo-code of the iterative approach is given in Algorithm 1. The inputs to this algorithm are the DUV denoted by D, the set S of scenarios to be considered, the set T of thresholds (each t_{S_i} associates to S_i), and a number k controlling the number of stimuli to be generated in each iteration.

The algorithm starts with the initialization of two sets: First, the set S_{uncov} contains all scenarios S_i that have not been sufficiently triggered. At the beginning, it is initialized with S (Line 1). Hence, all scenarios of D are under consideration. Second, the set S_{log} records all stimuli generated so far. It

Algorithm 1: Iterative Approach

Input: Design D, Scenarios S, Thresholds T, k

1 $S_{uncov} = S$;
2 $S_{log} = \emptyset$;
3 $Instance = \bigvee_{S_i \in S_{uncov}} S_i \wedge D \wedge \overline{\bigvee_{S_{stim_i} \in S_{log}} S_{stim_i}}$;
4 Generate k stimuli S_{stim} ;
5 $S_{log} = S_{log} \cup S_{stim}$;
6 Simulate D with S_{stim} ;
7 Update coverage status c_{S_i} for each $S_i \in S_{uncov}$;
8 **foreach** $S_i \in S_{uncov}$ **do**
9 | $S_{uncov} = S_{uncov} \setminus S_i \quad (c_{S_i} \geq t_{S_i})$;
10 **if** $S_{uncov} \neq \emptyset$ **then**
11 | goto 3 ;
12 **else**
13 | **return** ;
14

serves to produce different stimuli in each iteration during the simulation-based verification process. At the beginning, S_{log} is empty (Line 2).

Then, k stimuli S_{stim} are generated (Line 3 - Line 4). For this purpose, we form an instance by ANDing up the insufficiently covered scenarios ($\vee S_i$ where $S_i \in S_{uncov}$), the negation of the previously applied stimuli S_{log}, and the design D. Next, we run the constraint-solver for this instance. If we get a solution, satisfying assignments are extracted for the primary inputs and internal states of the DUV D. They form the set of new stimuli S_{stim} (Line 4). For blocking them in subsequent iterations, they are stored in S_{log} (Line 5).

Next, the determined stimuli S_{stim} are simulated on the DUV (Line 6). This enables to update the coverage status c_{S_i} for each insufficiently triggered scenario S_i (Line 7). Based on this information, we possibly reformulate the complete constraint for stimuli generation. The coverage status c_{S_i} of each scenario S_i ($S_i \in S_{uncov}$) is compared with its corresponding threshold t_{S_i}. In case $c_{S_i} \geq t_{s_i}$, the scenario S_i has been sufficiently triggered in the previous iteration and, hence, it is removed from further considerations (Line 9). The algorithm continues on Line 3 if there are still insufficiently covered scenarios.

B. Semi-automatic approach

There are similarities of the iterative and interactive approach. However, revisions and adjustments on constraints are performed manually, which makes the approach semi-automatic.

As motivated in Section III, based on design knowledge the verification engineer selects a subset of insufficiently covered scenarios and replaces the scenarios by a new one. We demonstrate this in the following example. The example exploits that clever stimuli can trigger two scenarios at the same time (which allows parallel simulation and checking).

Example 2. *Consider a RISC processor example, where three scenarios $S = \{S_0, S_1, S_2\}$ are under consideration. S_0 and S_1 ($s = 0/1$) specify scenarios of an ALU as in Example 1. S_2 is a scenario triggering the program counter (PC) with typical control inputs en (PC enable) and le (PC load), i.e. S_2 is specified as $S_2 = (en = 1 \wedge le = 0)$ (triggering a PC incrementation). Instead of constraint $C = S_0 \vee S_1 \vee S_2$, i.e. instead of $(s = 0) \vee (s = 1) \vee (en = 1 \wedge le = 0)$, the scenario subsets $R = \{S_0, S_2\}$ or $R = \{S_1, S_2\}$ can be substituted by $R' = S_0 \wedge S_2 = (s = 0 \wedge en = 1 \wedge le = 0)$ or $R' = S_1 \wedge S_2 = (s = 1 \wedge en = 1 \wedge le = 0)$, respectively. In the case where an ALU addition or subtraction is executed, also the PC is incremented to point to the next instruction. Hence, the generated stimuli with $C = R'$ can trigger the scenario S_2 and, at the same time, additionally either S_0 or S_1. By modifying the constraint as described, this is enforced.*

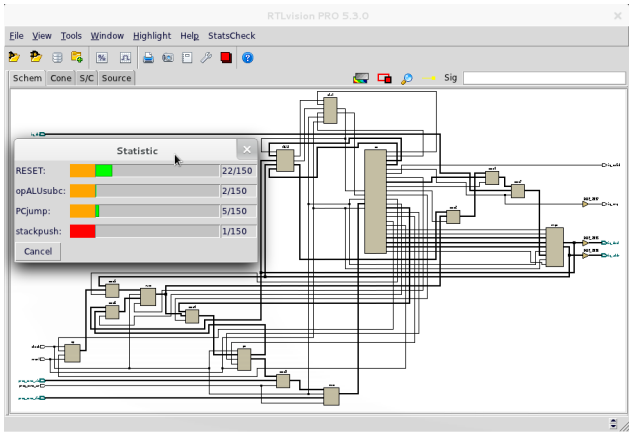

Fig. 1: Visualization aiding the verification engineers

Following this idea, the interactive approach works by (1) manually formulating a new (and better) scenario R' substituting scenarios $R \subset S$, (2) generating and applying k stimuli, (3) removing sufficiently covered scenarios, and (4) repeating this procedure until S_{suff} is achieved.

Since the formulation of a new scenario (Step (1)) is a manual step, we thereby aid the responsible verification engineers by newly developed visualization techniques. More precisely, in addition to a full and hierarchical display of the DUV (using the visualization tool *RTLvision* by Concept Engineering), the engineers can explicitly highlight scenarios and dependent signals as well as their coverage status value c_{S_i} and their corresponding threshold value t_{S_i}. Fig. 1 shows a screenshot of the developed visualization. This intuitive interface helps to locate potential candidates for substitution.

In this sense, the interactive approach is based on the iterative approach. However, instead of $R \in S_{uncov}$, the newly generated scenario R' is applied to generate the stimuli. More formally, the respective instance (Line 3 in Algorithm 1) is modified as follows:

$$Instance = R' \wedge D \wedge \overline{\bigvee_{S_{stim_i} \in S_{log}} S_{stim_i}} \tag{6}$$

Accordingly, a new threshold value $t_{R'}$ with

$$t_{R'} = max\{t_{S_{i_0}}, ..., t_{S_{i_m}}\}(\{S_{i_0}, ..., S_{i_m}\} = R) \tag{7}$$

is added guaranteeing that all scenarios $\{S_{i_0}, ..., S_{i_m}\}$ implied by R' become sufficiently covered when $c_{R'} \geq t_{R'}$. Afterwards, the algorithm continues with stimuli generation, simulation, and removal of sufficiently triggered scenarios analogously to Algorithm 1.

V. Case Study

The proposed approaches have been implemented in C++ and evaluated by means of a case study. As case study, we used the design of a simple RISC processor described in [13]. This processor employs a Harvard architecture and, thus, is composed of typical components like a control unit (CU), a program counter (PC), and an ALU. Operands of the ALU are fetched from an external data memory (RAM) to which the result are also written back. The stack pointer (SP) represents a special register that enables standard stack operations targeting the memory.

For the simulation-based verification, seven scenarios have been considered:

S_0 Reset: Initialize the PC and SP.
S_1 Alu_Not: Invoke a NOT operation in the ALU.
S_2 Alu_Sub: Invoke a subtraction operation in the ALU.
S_3 Pc_Incr: Increment the PC by 1.

S_4 Pc_Jump: Update the PC with a target address.
S_5 Stack_Push: Stack push, triggered by a function call.
S_6 Stack_Pop: Stack pop, triggered by return of a function.

The threshold t_{s_i} for each scenario S_i has been set to 40. The maximum number of stimuli to be generated by the naïve approach has been set to $k = 600$. The number of stimuli to be generated by the iterative approach and by the interactive approach has been set to $k = 50$ for each iteration. All evaluations have been conducted on an AMD Athlon 3700 machine with 4 GB of memory running linux.

Table I shows the results obtained by applying the naïve approach. The second row denotes the respective scenarios, while the third row denotes the coverage status for each scenario, i.e. the number of stimuli triggering the respective scenarios. Bold indicates insufficiently covered scenarios in each iteration. As can be seen, after 600 generated stimuli still not all scenarios have been sufficiently covered. In fact, scenarios S_0 and S_2 are below the threshold. This emphasizes the need for an elaborated support of coverage-driven stimuli generation. Just generating stimuli without an explicit consideration of the respective coverage metric does not lead to satisfactory results.

In contrast, the iterative approach shows considerable improvements. Results are provided in Table II. The first column denotes the respective scenarios, while the remaining columns denote the respective coverage status for each iteration. Here, a sufficient set of stimuli can indeed be generated. This set eventually is composed of 350 stimuli, i.e. is even smaller than the non-sufficient set of stimuli which has been generated by the naïve approach.

In Table II, the evolvement of this set can be retraced: At the beginning, all scenarios are considered simultaneously. After the first iteration, S_1 is already sufficiently triggered. Hence, this scenario is removed from further consideration. This continues until the sixth iteration, where only scenario S_6 remained left. That is, in the last iteration, only stimuli satisfying S_6 are generated eventually leading to a sufficient set of stimuli for all scenarios.

Furthermore, the coverage status for all scenarios except S_3 and S_4 remained unchanged after their respective threshold has been achieved. This allows the conclusion that S_3 and S_4 share concurrency with other scenarios which have not sufficiently been triggered at that time. As mentioned above, such concurrency can be exploited to generate stimuli triggering multiple scenarios and, therefore, lead to a further reduction of the number of needed stimuli. This is the case in the interactive approach which is evaluated next.

Results obtained by the interactive approach are provided in Table III. The denotation of this table is almost identical to Table II – just a further row (denoted by R') is added which lists interactively added scenarios for each iteration. Also here, a sufficient set of stimuli can be generated. With 250 stimuli, this set is even smaller than the two sets generated using the other approaches. In fact, overall an improvement of approx. 60% (compared to the naïve approach) and approx. 30% (compared to the iterative approach) is documented.

This improvement was possible because of the interactive addition of new scenarios (in the second iteration and the third iteration). The former one became obvious, after S_1 got sufficiently been covered. Here, scenario S_2 (invoke a subtraction in the ALU) and scenario S_3 (increment the PC by 1) have been ANDed up. This immediately led to stimuli sufficiently satisfying both scenarios. Similarly, a better constraint $S_4 \wedge S_5$ was determined in the third iteration exploiting the concurrency of a PC jump and a Stack push operation. As a consequence, scenarios S_4 and S_5 also became sufficiently triggered immediately.

Overall, the case study confirmed the benefits of coverage-driven stimuli generation. Without an explicit consideration of coverage metrics, it was not possible to generate a set of stimuli which sufficiently covers all scenarios. In contrast, using the iterative approach, such a set has been determined. Additionally exploiting design knowledge by the verification engineers further improves the results.

VI. Conclusions

In this work, we considered coverage-driven stimuli generation. We proposed approaches that not only generated stimuli for a given set of scenarios, but additionally incorporated respective coverage metrics during this process. A case study demonstrated the benefits of the proposed approach: Explicitly considering coverage metrics led to smaller and complete sets of stimuli. The results can further be improved by manual interaction. For this purpose, the verification engineers are supported by visualization techniques.

VII. Acknowledgments

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the projects SANITAS under contract no. 01M3088 and VisES under contract no. 01M3197B.

TABLE I: Naïve approach

Threshold: 40, k: 600, CPU: 51m22.204s

Constraints	S_0	S_1	S_2	S_3	S_4	S_5	S_6
c_{s_i}	***11***	50	***39***	407	138	46	67

TABLE II: Iterative approach

Threshold: 40, k: 50, CPU time: 29m9.827s

Constraints	c_{s_i} after each iteration						
	50	100	150	200	250	300	350
S_0. Reset	***0***	***3***	***3***	***3***	***3***	53	53
S_1. Alu_Not	42	42	42	42	42	42	42
S_2. Alu_Sub	***1***	***1***	***1***	***1***	51	51	51
S_3. Pc_Incr	***31***	78	78	128	128	128	128
S_4. Pc_Jump	***3***	***3***	53	53	53	53	103
S_5. Stack_Push	***10***	***0***	***0***	49	49	49	49
S_6. Stack_Pop	***2***	***2***	***3***	***4***	***4***	***4***	54

TABLE III: Interactive Approach

Threshold: 40, k: 50, CPU time: 14m50.706s

Constraints	c_{s_i} after each iteration				
	50	100	150	200	250
S_0. Reset	***0***	***0***	***0***	***0***	50
S_1. Alu_Not	42	42	42	42	42
S_2. Alu_Sub	***1***	51	51	51	51
S_3. Pc_Incr	***31***	81	81	81	81
S_4. Pc_Jump	***3***	***3***	53	103	103
S_5. Stack_Push	***0***	***0***	50	50	50
S_6. Stack_Pop	***2***	***2***	***2***	52	52
R'	none	$S_2 \wedge S_3$	$S_4 \wedge S_5$	none	none

References

[1] D. Brand, "Verification of large synthesized designs," in *Int'l Conf. on CAD*, 1993, pp. 534–537.
[2] S. Disch and C. Scholl, "Combinational equivalence checking using incremental SAT solving, output ordering, and resets," in *ASP Design Automation Conf.*, 2007, pp. 938–943.
[3] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999.
[4] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Tools and Algorithms for the Construction and Analysis of Systems*, 1999, pp. 193–207.
[5] J. Bergeron, *Writing Testbenches Using SystemVerilog*. Springer Verlag, 2006.
[6] J. Yuan, C. Pixley, and A. Aziz, *Constraint-based Verification*. Springer, 2006.
[7] N. Kitchen and A. Kuehlmann, "Stimulus generation for constrainted random simulation," in *Int'l Conf. on CAD*, 2007, pp. 258–265.
[8] R. Wille, D. Große, F. Haedicke, and R. Drechsler, "SMT-based stimuli generation in the SystemC verification library," in *Forum on Specification and Design Languages*, 2009, pp. 1–6.
[9] S. Tasiran and K. Keutzer, "Coverage metrics for functional validation of hardware designs," *IEEE Design & Test of Comp.*, vol. 18, no. 4, pp. 36–45, 2001.
[10] D. Lettnin, M. Winterholer, A. Braun, J. Gerlach, J. Ruf, T. Kropf, and W. Rosenstiel, "Coverage driven verification applied to embedded software," in *IEEE Annual Symposium on VLSI*, 2007, pp. 159 –164.
[11] P. Mishra and N. Dutt, "Functional coverage driven test generation for validation of pipelined processors," in *Design, Automation and Test in Europe*, 2005, pp. 678–683.
[12] S. M. Plaza, I. L. Markov, and V. Bertacco, "Toggle: A coverage-guided random stimulus generator," in *Int'l Workshop on Logic Synth.*, 2007, pp. 351–357.
[13] D. Große, U. Kühne, and R. Drechsler, "HW/SW Co-Verification of Embedded Systems using Bounded Model Checking," in *ACM Great Lakes Symposium on VLSI*, 2006, pp. 43–48.