Minimal Stimuli Generation in Simulation-based Verification

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
‡solvertec GmbH, 28359 Bremen, Germany
{shuo,rwille,drechsle}@informatik.uni-bremen.de grosse@solvertec.de

***Abstract*—Simulation-based verification is still the state-of-the-art when checking the correctness of complex Systems-on-Chips. In particular, constraint-based simulation is popular, since here dedicated stimuli are generated which trigger certain corner-case behavior. However, to the best of our knowledge, only heuristic methods have been introduced so far. In this paper, we propose an approach that determines a minimal set of stimuli for the desired set of scenarios to be simulated. For this purpose, we are making use of solving techniques from Boolean satisfiability. Experimental evaluations demonstrate that the proposed approach can be applied to generate very compact stimuli sets. Furthermore, the proposed approach can be used to evaluate the quality of results obtained by heuristic methods.**

***Keywords*-Constraint-based Simulation; Simulation-based Verification; Coverage**

I. Introduction

Verification continues to be a major bottleneck in the current design flow for embedded systems. In particular, functional verification, i.e. ensuring the functional correctness of a design, still dominates the overall costs in the development of modern complex *Systems-on-Chips* (SoCs).

Different approaches have thereby been developed to improve functional verification. Formal methods, for instance property checking (see e.g. [9], [4]) and completeness checking (see e.g. [8], [12], [7]), exploit the design's behavior exhaustively. However, they suffer inevitably from high computational costs. Consequently, the much faster simulation-based verification (see e.g. [3]) is still the dominating verification method in industry.

Here, a number of stimuli is either determined randomly considering the global functionality of a *Design Under Verification* (DUV) or directed to verify certain behaviors, e.g. corner cases. Afterwards, the stimuli are applied to the DUV, and their responses are checked, i.e. compared to the expected results.

In order to evaluate the verification quality, coverage metrics have been introduced (see e.g. [10], [17]). They abstract certain functionality of the DUV (e.g. in terms of *scenarios*) which is supposed to be checked.

With them, coverage analysis methods can efficiently evaluate whether the DUV has sufficiently been checked or not (see e.g. [14], [15], [1], [2]). In the former case, the simulation-based verification process terminates. Otherwise, further dedicated stimuli are generated aiming for covering the insufficiently considered cases. Commonly, the determination of such stimuli is performed using constraint-based stimuli generation (see e.g. [21], [16], [18]). Considering the insufficiently triggered cases only, constraints for stimuli generation are formulated and, subsequently, solved by a constraint solver. By this, stimuli result that directly trigger scenarios which have not sufficiently been covered yet.

Overall, this practice is a coverage-driven verification process. The successful adoption of this methodology continues encouraging new research in this field. One trend intends to improve constraint-based stimuli generation by boosting the ability of constraint-stimuli generation (see e.g. [13], [20], [19], [6]). In [13], [20], constraints are automatically generated either (1) for improving controllability of internal signals and thereby achieving a high coverage [13] or (2) for reducing the time-consuming and error-prone manual constraint composition process [20]. In contrast, other work focus on manipulating existing constraints either to obtain a compact set of stimuli (see e.g. [19]) or to increase the efficiency of the respective solving process (see e.g. [6], [18]).

However, to the best of our knowledge exact constraint-based stimuli generation, i.e. the determination of a minimal number of stimuli satisfying the coverage requirement, has not been considered so far. This is done in the paper at hand. For this purpose, we propose:

- An iterative work-flow: First, it is tried to sufficiently cover the DUV with one stimulus only. If no such stimulus exists, the number of stimuli to be generated is increased by 1. This is iterated until a set of stimuli satisfying the coverage requirement can be determined.
- A dedicated SAT-based generation: In order to determine the desired stimuli or to prove that the coverage requirements cannot be satisfied with the currently considered number of stimuli, we encode the respective problem as an instance of Boolean satisfiability. Then, off-the-shelf solvers are applied to solve the problem

Experimental evaluations show the applicability of the proposed approach. On the one hand side, it is shown that very compact sets of stimuli can be generated. On the other hand, the proposed approach allows for an evaluation of the results obtained by heuristic methods.

II. Problem Formulation

This section formally introduces the problem addressed in this paper.

In general, it is infeasible to exhaustively simulate a DUV with all its possible input assignments. Therefore, simulation-based verification is applied in order to explicitly cover certain corner-case functionalities of the DUV. For this purpose, user-defined scenarios are formulated for which dedicated stimuli shall be generated. 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, scenario and constraint is used interchangeably. The set of scenarios is denoted by* $S = \{S_0, \ldots, S_{n-1}\}$.

The goal of methods for constraint-based random simulation is to generate a stimulus which triggers at least one scenario. For this purpose, a proper constraint is constructed as input for the respective engines.

Definition 2. *A stimulus satisfying at least one scenario can be derived from a solution to the following constraint:*

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

Example 1. *Consider a program counter (PC), where the inputs* $reset$ *and* $load$ *define the renewal strategy on the current program address* pc*. Possible scenarios are for instance* $S_0 = ((reset = 0) \wedge (load = 0))$ *(the* pc *is incremented by 1) and* $S_1 = ((reset = 0) \wedge (load = 1))$ *(the* pc *is renewed with an arbitrary value according to the address input of the program counter). Then, the stimulus* $\{pc = 0x00FF,\ reset = 0,\ load = 0\}$ *triggers a scenario (here* S_0*), while* $\{pc = 0x00FF,\ reset = 1,\ load = 0\}$ *does not.*

The amount of stimuli triggering a scenario is important. So 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 amount of stimuli which have triggered the scenario* S_i*.*

In simulation-based verification, stimuli are generated (and their responses are checked) so that all scenarios S are sufficiently covered. The term "sufficiently" is thereby user-defined via a threshold value such 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* $c_{S_i} \geq t_{S_i}$*.*

The simulation-based verification terminates, when a sufficient set of stimuli has been generated.

Definition 5. *A set of stimuli is considered* sufficient *iff it contains stimuli sufficiently covering all scenarios, i.e. each scenario* S_i *is triggered by at least* t_{S_i} *different stimuli. In the following, sufficient sets of stimuli are denoted by* S_{suff}*.*

Due to various methods used for the stimuli generation, the size of the sufficient stimuli S_{suff} may vary significantly.

Example 2. *Consider the DUV composed of a PC and an ALU, where scenarios* S_0 *and* S_1 *for the PC are defined similar to the one of Example 1. For the ALU, a scenario* $S_2 = (op = add)$ *forcing an ALU addition is introduced (the* op *is the select input to define the operation to be performed on the data inputs of the ALU). Finally, assume that each scenario is supposed to be triggered at least 10 times, i.e.* $\bigwedge_{i=0}^{2}(t_{S_i} = 10)$*. A naive approach would individually generate stimuli for each scenario eventually leading to a total of 30 stimuli. However, since an ALU addition (*$op = add$*) may be followed by an PC incrementation (*$load = 0$*), in fact, significant less stimuli are required in order to satisfy the coverage criteria.*

In general, a sufficient stimuli set of minimal size is interesting. The term "minimal" is thereby formalized as follows:

Definition 6. *Given a verification task and a sufficient set* S_{suff} *of stimuli. The set* S_{suff} *of stimuli is called* minimal*, iff no other sufficient set* S'_{suff} *of stimuli with* $|S_{suff}| > |S'_{suff}|$ *exists.*

A minimal set of sufficient stimuli provides a lower-bound to the amount of stimuli that have to be applied to a DUV. Commonly, in order to achieve such minimal stimuli set, one stimulus must trigger multiple scenarios at the same time. This fact is important since bugs often occur when multiple behaviors of the DUV occur simultaneously [1]. Furthermore, a minimal set of stimuli is compact and, thus, suitable to form a regression suit which can be run periodically or after major changes to the DUV.

However, determining such a minimal S_{suff} often is not trivial. Several methods that can reduce the size of a S_{suff} have been published in the past. In [20], a set of constraints was generated for multiple coverage holes. As a consequence, the resulting stimuli could affect multiple coverage holes simultaneously. However, no mechanism was used there to guarantee that the resulting stimuli set is minimal. In [19], a parallel simulation for multiple scenarios has been exploited. However, this has been conducted in a heuristic manner so that also here minimality was not guaranteed. Motivated by this, we address the following research question in this paper:

> *How can we efficiently determine a* minimal *set of stimuli* S_{suff}*, so that each scenario* S_i *is triggered by at least* t_{S_i} *different stimuli.*

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

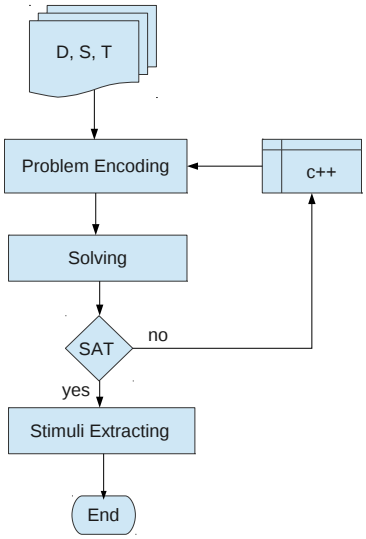

Fig. 1: Determination of Minimal Set S_{suff}

III. General Idea and Main Flow

Given a DUV, a set S of scenarios, and a threshold value t_{S_i} for each scenario, the determination of a minimal S_{suff} is formulated as a sequence of decision problems. The respective decision problems ask whether there is a set of c stimuli which sufficiently triggers all scenarios of the considered DUV. When this decision problem turns out to be satisfiable, a sufficient set of c stimuli can be derived. Otherwise, it has been proven that no S_{suff} with c stimuli exists. The general idea of our approach is to solve this kind of decision problems until a satisfying solution has been obtained. Minimality is thereby ensured by starting with $c = 1$ and iteratively incrementing c by one whenever the decision problems turns out to be unsatisfiable. The respective decision problems are solved using solvers for Boolean satisfiability.

Based on these ideas, the overall method is formulated as an iterative approach as shown in Fig. 1. The inputs are the DUV, a set S of scenarios, and a set T of corresponding thresholds. The flow starts with encoding the problem as an instance of the satisfiability problem with a fix number c of stimuli. Afterwards, the resulting instance is solved using off-the-shelf solvers. When the instance is satisfiable, c stimuli composing a minimal S_{suff} are extracted and the approach terminates. Otherwise, c is incremented by one and the last steps are repeated.

In order to realize this approach, the respective problem needs to be encoded with respect to the dedicated solving engines, i.e. the applied SAT solvers in this case. Therefore, the problem is formulated as a conjunction of the following constraints: the proposed stimuli number c, the constraint of the DUV, the constraint of all scenarios S, the constraint of the threshold t_{S_i} for each scenario, and, finally, the constraint which enforces all S to be sufficiently triggered. In the next section, the details on the encoding are presented.

IV. Endcoding and Implementation

For clarity, how to determine a minimal stimuli set covering each scenario S_i at least one time, i.e. $t_{S_i} = 1$, is first described. Afterwards, extensions are introduced so that arbitrary threshold values, i.e. $t_{S_i} \geq 1$ can be supported.

A. Minimal Stimuli Set with $t_{S_i} = 1$

As mentioned earlier, a sequence of decision problems is formulated and solved based on SAT techniques. For this purpose, an instance is created for each of the decision problems. In order to describe the respective encoding of the instance smoothly, the applied vectors of variables are defined first.

Definition 7. *Let $c \in \mathbb{N}$ with $c > 0$ which corresponds to the currently considered number of stimuli. Then, given the DUV with m primary inputs, k states (FFs), as well as n scenarios ($m, k, n \in \mathbb{N}$), we define:*

1) $\overrightarrow{DUV} = (DUV^0, DUV^1, \ldots, DUV^{c-1})$ *represents a vector of DUV copies, i.e. DUV^0 denotes the first copy, DUV^1 denotes the second one, etc. Apparently, all copies are functionally identical but allow for different assignments of their respective signals.*
2) *For each copy DUV^d with $0 \leq d < c$, the vector $\overrightarrow{x^d} = (x_0^d, \ldots, x_{m-1}^d)$ denotes the primary inputs of the currently considered DUV-copy. Similarly, the FFs of the copy are denoted by $\overrightarrow{r^d} = (r_0^d, \ldots, r_{k-1}^d)$. Overall, $\overrightarrow{x} = (\overrightarrow{x^0}, \ldots, \overrightarrow{x^{c-1}})$ and $\overrightarrow{r} = (\overrightarrow{r^0}, \ldots, \overrightarrow{r^{c-1}})$ represent the primary inputs and the FFs of the c different DUV copies, respectively. Since the value assignments on $\overrightarrow{x^d}$ and $\overrightarrow{r^d}$ together build up a stimulus on DUV^d, the assignments on $\{\overrightarrow{x}, \overrightarrow{r}\}$ compose a stimuli set.*
3) *For each copy DUV^d, $\overrightarrow{s^d} = (S_0^d, \ldots, S_{n-1}^d)$ defines one copy of all scenarios. Thereby, $\overrightarrow{s} = (\overrightarrow{s^0}, \ldots, \overrightarrow{s^{c-1}})$ represents c copies of them. Similarly, the copies of each scenario S_i are functionally identical, but independent with each other because of the renamed signals in all copies of S_i.*

Based on these definitions we can formulate the following theorem:

Theorem 1. *A scenario S_i is considered to be sufficiently ($t_{S_i} = 1$) covered iff at least one of its copies S_i^d is triggered on the corresponding DUV^d.*

With this theorem, the problem to determine a minimal S_{suff} with $t_{S_i} = 1$ is formalized as: *Is there an assignment to $\{\overrightarrow{x}, \overrightarrow{r}\}$ such that for all scenarios, at least one of its copies is triggered and c for both $\overrightarrow{x}$ and $\overrightarrow{r}$ is minimal?*

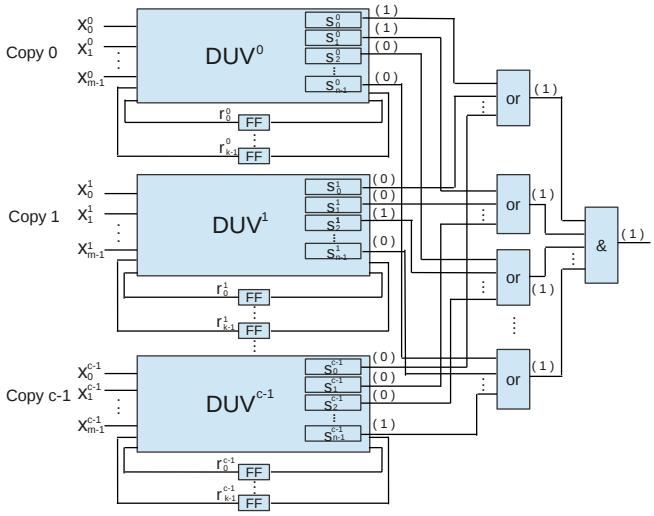

Fig. 2: Structure of the Problem Encoding

The general structure for the problem to be converted to an instance is shown in Fig. 2. Besides the elements for c copies of the DUV and scenarios S, all different copies S_i^d of the scenario S_i are connected via an OR gate as depicted. Furthermore, an AND gate constrains all the outputs of the OR gates. Apparently, the output of the AND gate must be fixed to 1 such that all scenarios are enforced to be sufficiently triggered (a 1 at the output of a scenario copy S_i^d ensures that the scenario S_i is simulated by the input vectors $\{\overrightarrow{x^d}, \overrightarrow{r^d}\}$ on DUV^d).

Example 3. *As an example, a Boolean assignment on $\overrightarrow{s}$ is shown in Fig. 2. As can be seen, the stimulus $\{\overrightarrow{x^0}, \overrightarrow{r^0}\}$ triggers S_0^0 and S_1^0, i.e. S_0 and S_1 while S_2^0 to S_{n-1}^0 evaluate to 0, i.e. S_2 and S_{n-1} are not triggered in Copy 0. When the structure contains only one copy, i.e. $c = 1$, no S_{suff} can be determined. The scenarios S_2^0 and S_{n-1}^0 evaluate to 0, which is invalid since the AND gate requires them to become 1[1]. As a consequence, the number c has to be incremented and thereby, further copies are gradually added in the structure.*

Example 4. *Consider again Example 3. Further copies 1 to $c-1$ have been added. As a consequence, the stimulus $\{\overrightarrow{x^1}, \overrightarrow{r^1}\}$ generated of Copy 1 triggers S_2^1, i.e. S_2, and $\{\overrightarrow{x^{c-1}}, \overrightarrow{r^{c-1}}\}$ of Copy $c-1$ triggers S_{n-1}^{c-1}, i.e. S_{n-1}. Since S_0 and S_1 have already been triggered by $\{\overrightarrow{x^0}, \overrightarrow{r^0}\}$ in Copy 0, the stimuli set $\{\{\overrightarrow{x^0}, \overrightarrow{r^0}\}, \{\overrightarrow{x^1}, \overrightarrow{r^1}\}, \ldots, \{\overrightarrow{x^{c-1}}, \overrightarrow{r^{c-1}}\}\}$ with minimal c sufficiently triggers all scenarios.*

With the structure as shown in Fig. 2, the instance to be solved can be obtained by directly encoding the elements into a SAT instance as follows:

[1]Please note for the case of one copy there is no OR level.

1) Constraints for c copies of the DUV and the scenarios following the well known Tseitin transformation.
2) Constraints for the OR gates and the AND gate are added.
3) Since $t_{S_i} = 1$, no constraint for the threshold is needed.

Finally, the complete instance for each decision problem is:

$$\bigwedge_{d=0}^{c-1} DUV^d \wedge \bigwedge_{i=0}^{n-1} \bigvee_{d=0}^{c-1} S_i^d \tag{2}$$

As mentioned earlier, a minimal S_{suff} with $t_{S_i} = 1$ can be obtained by gradually incrementing c. The worst case is $c = n$, i.e. each scenario needs a new copy since no scenarios can be triggered simultaneously.

In the following section, we consider the case with threshold values greater than one.

B. Minimal Stimuli Sets with $t_{S_i} \geq 1$

To enable a threshold value greater than one, the problem instance as introduced in the previous section can be reformulated. Considering the problem instance as presented in Formula 2, the second term in the formula ensures that each the scenario is triggered since in at least one of the scenario's copies a 1 is required. This is expressed using the OR term. Now, a simple OR is not sufficient. Instead, we have to sum up the triggerings for each scenario S_i and ensure that the user-defined threshold value is achieved. This results in the following new instance:

$$\bigwedge_{d=0}^{c-1} DUV^d \wedge \bigwedge_{i=0}^{n-1} (\sum_{d=0}^{c-1} S_i^d \geq t_{s_i}) \tag{3}$$

Based on this instance, the general procedure for minimal stimuli generation does not change. However, a satisfying solution may require a large value for c since potentially many copies are required to meet the threshold condition of each scenario. Hence, we propose a different approach. Instead of "blowing up" the instance, we perform several calls to the constraint solver while blocking the previous solutions (hence ensuring new stimuli to be found).

Example 5. *Consider again Example 2. Assume that two stimuli S_{stim1} and S_{stim2} have been determined. S_{stim1} triggers S_0 and S_2, while S_{stim2} triggers S_1. To generate additional stimuli, we block the previous stimuli. As problem instance we get:*

$$\bigwedge_{d=0}^{1} DUV^d \wedge (\bigwedge_{i=0}^{2} \bigvee_{d=0}^{1} S_i^d) \wedge \overline{S_{stim1}} \wedge \overline{S_{stim2}} \tag{4}$$

Now, we can call the constraint solver again to determine additional stimuli meeting higher threshold values.

Algorithm 1: Minimal Stimuli Generation

Input: DUV, Scenarios S, Thresholds T

1 $c = 1$; $S_{log} = \emptyset$; $c_{S_i} = 0$ foreach $S_i \subseteq S$;

2 $Instance = \bigwedge_{d=0}^{c-1} DUV^d \wedge (\bigwedge_{i=0}^{n-1} \bigvee_{d=0}^{c-1} S_i^d) \wedge \overline{\bigvee_{S_{stim_i} \in S_{log}} S_{stim_i}}$;

3 $result = solving(Instance)$;

4 **if** $result = true$ **then**

5 extract c stimuli $\{S_{stim_0}, \ldots, S_{stim_{c-1}}\}$;

6 $S_{log} = S_{log} \cup \{S_{stim_0}, \ldots, S_{stim_{c-1}}\}$;

7 $c_{S_i} = c_{S_i} + \sum_{d=0}^{c-1} S_i^d$ foreach $S_i \in S$;

8 **if** $c_{S_i} \geq t_{S_i}$ foreach $S_i \in S$ **then**

9 return

10 **else**

11 go to 2 ;

12 **else**

13 ++c

14 go to 2

Overall, we have devised Algorithm 1 to determine the stimuli set S_{suff} with $t_{S_i} \geq 1$. The number c is initialized to 1, and the log files recording the generated stimuli as well as coverage status for each scenario are initialized in Line 1. Then, the instance is built and solved (Line 2-3). If the instance is satisfiable (Line 4), the stimuli are extracted and recorded. Furthermore, the coverage status is updated (Line 5-7). If all scenarios are sufficiently covered (Line 8), the algorithm terminates. Otherwise, a new instance with the blocked stimuli is created and the process is repeated (Line 11). Here, the number c is not incremented. New stimuli are still generated based on the current instance. In case no more stimuli can be generated with the current value of c, c is incremented such that an additional DUV copy is included (Line 12-13).

The effectiveness of the proposed algorithm is demonstrated in the experimental evaluation in the next section.

V. Experimental Evaluation

In this section the experimental evaluation is presented. The proposed approach has been implemented in C++. As constraint solver we used Boolector [5]. To illustrate the applicability of our approach, experimental evaluations for several benchmarks have been conducted. In the following, we summarize the results.

As benchmarks, the following DUVs have been considered:

- The RISC processor from [11] has been used. Essentially, this is a Harvard architecture composed of typical components such as a *Control Unit* (CU), a *Program Counter* (PC), an ALU, an external data memory (RAM), and a stack pointer which represents a special register and enables standard stack operations on the RAM.
- An alternative CPU with a reduced ALU (denoted by *ALU_reduced*) implementing 8 ALU functions.
- A *Memory Management Unit* (MMU) which is used as an interface between a CPU and an external memory. This unit manages the corresponding data transactions between these components.

For each DUV, suitable scenarios have been considered, i.e. several scenarios for the respective components of the CPU (e.g. 3 scenarios for the PC and 16 scenarios for the CU, etc.), 8 scenarios for the reduced CPU, and 53 scenarios for the MMU. The results generated by our approach have also been compared to the results obtained by previously introduced approaches, namely the naive and the iterative stimuli generation scheme as proposed in [19]. Here, the maximal number of stimuli to be generated has been set to 1000. All experiments have been conducted using a 64-bit AMD Athlon Dual Core machine with 4 GB of memory running Linux.

Results of the evaluations assuming a threshold for each scenario of $t_{S_i} = 1$ and a threshold of $t_{S_i} = 40$ are reported in Table I and Table II, respectively. The first column denotes the name of the DUV, while the second column gives the number of the considered scenarios. The remaining columns provide both, the number of generated stimuli and the run-time to generate them (in CPU seconds) for each approach.

The results clearly show the benefits of the proposed approach. Of course, determining the minimal number of stimuli is computationally more expensive than a heuristic approach. However, particularly compared to the naive approach from [19] the proposed solution gives very good results. In fact, orders of magnitudes less stimuli are generated in a reasonable time. Compared to the iterative approach from [19], the proposed approach allows for rating the effectiveness of the heuristic: In fact, we were able to show that for many benchmarks the iterative approach already determines minimal or close to minimal solutions (cf. *PC*, *Stack*, or *ALU_reduced*). Considering all benchmarks, the iterative approach never generates more than twice the number of stimuli than necessary. Overall, the proposed approach allows (1) for an efficient stimuli generation and (2) for an analysis how far heuristics are away from the optimum.

VI. Conclusions

In this work, we have presented a solution for the determination of a minimal set of stimuli for simulation-based verification. For this purpose, we formulated the problem as a sequence of decision problems which, in turn, are solved using techniques of Boolean satisfiability. Experimental evaluations demonstrated the applicability of the proposed solution, i.e. the approach can be applied to generate a very efficient set of stimuli or to evaluate the quality of results obtained by heuristic methods. The methods presented in this paper can further be adopted targeting on other research questions such as design learning, parallel simulation, or coverage analysis.

TABLE I: Minimal S_{suff} with $t_{S_i} = 1$

Maximal amount of generated stimuli: 1000

method		naive approach [19]		iterative approach [19]		minimal approach	
design	# of scenarios	# of stimuli	run time(s)	# of stimuli	run time(s)	# of stimuli	run time(s)
PC	3	16	0.01	3	0.01	3	0.01
Stack	3	104	0.02	3	0.01	3	0.02
Data Memory	4	$\geqslant 1000$	-	3	0.28	2	0.34
ALU_reduced	8	42	0.04	8	0.02	8	0.16
ALU	16	47	0.03	16	0.03	-	time out
Control Unit	16	$\geqslant 1000$	-	5	0.01	3	0.02
RISC	40	$\geqslant 1000$	-	20	1.65	12	46.16
MMU	53	$\geqslant 1000$	-	20	0.06	11	6.11

TABLE II: Minimal S_{suff} with $t_{S_i} = 40$

Maximal amount of generated stimuli: 1000

method		naive approach [19]		iterative approach [19]		minimal approach	
design	# of scenarios	# of stimuli	run time(s)	# of stimuli	run time(s)	# of stimuli	run time(s)
PC	3	404	0.08	120	0.03	120	0.05
Stack	3	143	0.03	120	0.03	120	0.03
Data Memory	4	$\geqslant 1000$	-	120	3.30	80	3.00
ALU_reduced	8	428	0.08	320	0.10	320	0.21
ALU	16	989	0.36	640	0.29	-	time out
Control Unit	16	$\geqslant 1000$	-	200	0.07	120	0.10
RISC	40	$\geqslant 1000$	-	800	49.85	480	92.18
MMU	53	$\geqslant 1000$	-	800	1.33	440	7.89

REFERENCES

[1] S. Asaf, E. Marcus, and A. Ziv. Defining coverage views to improve functional coverage analysis. In *Design Automation Conf.*, pages 41–44, 2004.

[2] H. Azatchi, L. Fournier, E. Marcus, S. Ur, A. Ziv, and K. Zohar. Advanced analysis techniques for cross-product coverage. *IEEE Trans. on Comp.*, 55(11):1367–1379, 2006.

[3] J. Bergeron. *Writing Testbenches Using SystemVerilog*. Springer Verlag, 2006.

[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*, pages 193–207, 1999.

[5] R. Brummayer and A. Biere. Boolector: An efficient SMT solver for bit-vectors and arrays. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 174–177, 2009.

[6] A.-C. Cheng, C.-C. Yen, and J.-Y. Jou. A formal method to improve system verilog functional coverage. In *IEEE International High Level Design Validation and Test Workshop*, pages 56–64, 2012.

[7] H. Chockler, D. Kroening, and M. Purandare. Coverage in interpolation-based model checking. In *Design Automation Conf.*, pages 182–187, 2010.

[8] K. Claessen. A coverage analysis for safety property lists. In *Int'l Conf. on Formal Methods in CAD*, pages 139–145, 2007.

[9] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[10] R. Grinwald, E. Harel, M. Orgad, S. Ur, and A. Ziv. User defined coverage-a tool supported methodology for design verification. In *Design Automation Conf.*, pages 158–165, 1998.

[11] 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*, pages 43–48, 2006.

[12] D. Große, U. Kühne, and R. Drechsler. Analyzing functional coverage in bounded model checking. *IEEE Trans. on CAD*, 27(7):1305–1314, 2008.

[13] O. Guzey and L.-C. Wang. Coverage-directed test generation through automatic constraint extraction. In *IEEE International Workshop on High-Level Design Validation and Test*, pages 151–158, 2007.

[14] O. Lachish, E. Marcus, S. Ur, and A. Ziv. Hole analysis for functional coverage data. In *Design Automation Conf.*, pages 807–812, 2002.

[15] A. Piziali. *Functional Verification Coverage Measurement and Analysis*. Springer, 2004.

[16] S. M. Plaza, I. L. Markov, and V. Bertacco. Random stimulus generation using entropy and XOR constraints. In *Design, Automation and Test in Europe*, pages 664–669, 2008.

[17] S. Tasiran and K. Keutzer. Coverage metrics for functional validation of hardware designs. *IEEE Design & Test of Comp.*, 18(4):36 –45, 2001.

[18] 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*, pages 1–6, 2009.

[19] S. Yang, R. Wille, D. Große, and R. Drechsler. Coverage-driven stimuli generation. In *EUROMICRO Symp. on Digital System Design*, pages 525–528, 2012.

[20] H.-H. Yeh and C.-Y. Huang. Automatic constraint generation for guided random simulation. In *ASP Design Automation Conf.*, pages 613–618, 2010.

[21] J. Yuan, C. Pixley, and A. Aziz. *Constraint-based Verification*. Springer, 2006.