Exact Stimuli Minimization for Simulation-based Verification

Sebastian Pointner Andreas Grimmer Robert Wille
Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
Email: {sebastian.pointner, andreas.grimmer, robert.wille}@jku.at

***Abstract*—Due to the ever increasing complexity of modern circuits and systems, verification represents one of the most time-consuming tasks in the entire design process for embedded systems. For this purpose, simulation-based techniques are widely applied in industrial contexts. Here, stimuli are determined which are used as input for the *Design under Verification* (DUV) and are supposed to trigger different aspects of the new design. However, usually much more stimuli are generated than actually needed to comprehensively cover all aspects. This obviously increases the run time of the verification significantly. Consequently, verification engineers aim for minimizing the number of stimuli after their generation – without loosing their coverage. Existing solutions, however, usually generate results which are far from being optimal. Besides that, their scalability is severely limited. In this work, we propose a solution for an *exact* minimization of stimuli. To this end, we utilize the computational power of modern reasoning engines such as MAX-SAT solvers which can efficiently minimize a given set of stimuli. Experimental evaluations confirm that, compared to previous work, up to 63% further reduction can be obtained and scalability significantly increases.**

I. Introduction

Embedded systems have become an integral part of our daily life. They can be found everywhere in our environment and are especially very important in their usage for safety critical applications. Since errors in safety critical applications (e.g. the control unit to trigger the airbag in a car) may have catastrophic consequences, guaranteeing their correctness is of uttermost importance. Hence, verification is essential when designing such systems.

To this end, methods for simulation-based verification [1] are widely applied in industry. Here, previously generated *stimuli* are applied to the system and, afterwards, it is checked whether the actually intended output is obtained. In this context, the system is treated as a black box where it is possible to only interact with the inputs and the outputs. Every input, i.e. stimulus, is supposed to trigger a certain functionality, i.e. *aspect*, of the system. Since modern embedded systems are highly complex systems, simulation-based verification frequently requires a large amount of stimuli to completely cover every aspect of the system.

The quality of the resulting set of stimuli is then measured by their *coverage* of aspects. Usually, verification engineers are interested to cover all given aspects of a system. Since the complexity of modern embedded systems is very high, it is usually not possible to generate a set of stimuli covering all aspects by hand within acceptable time. Because of this, several approaches for automatic stimuli generation have been presented (see e.g. [2]–[6]). Since embedded systems are frequently described in C++-based system description languages (e.g. in SystemC [7]), these approaches are not limited to the verification of embedded system only.

However, although automatic methods for stimuli generation are capable to generate a set of stimuli with high aspect coverage [8], they usually yield much more stimuli than actually needed. In fact, most of the aspects frequently get covered not only by a single stimulus, but multiple ones. While this does not have a negative impact on the verification result, this may increase the run time needed for verification as more stimuli have to be simulated. Considering that the verification run time of a highly complex industrial system can easily last multiple days, reducing the amount of unnecessary stimuli may yield substantial accelerations – in particular, since a set of stimuli usually includes a significant amount of redundancies.

Motivated by that, verification engineers aim for keeping the number of stimuli as small as possible. Accordingly, methods which optimize a given set with respect to their aspect coverage have been introduced. To the best of our knowledge, the method proposed in [9] constitutes the state-of-the-art. Here, the authors mapped the problem to a logic minimization problem which, afterwards, has been solved using Quine-McCluskey's algorithm. But since Quine-McCluskey's algorithm had been designed for the minimization of Boolean logic functions (i.e. not for stimuli minimization), they cannot guarantee that a minimal subset out of the given set of stimuli is generated (in fact, even the optimized set frequently is far from being optimal). Moreover, this approach is severely limited with respect to scalability – often only some few dozens stimuli can be optimized using the approach from [9].

In this work, we propose a solution for the *exact* minimization of a given set of stimuli, i.e. a solution which determines the minimal subset still covering all aspects. To deal with the complexity of this optimization problem, we utilize the computational power of reasoning engines, i.e. so called solvers for the *Maximum Satisfiability* (MAX-SAT) problem [10], [11]. More precisely, we formulate the problem in terms of a MAX-SAT instance which, afterwards, is passed to a MAX-SAT solver. From the resulting solution, the minimal subset can eventually be derived.

Experimental evaluations confirm the applicability and efficiency of the proposed solution. In fact, reductions in the number of needed stimuli amounting up to 63% compared to the solution proposed in [9] can be observed. Moreover, while the solution proposed in [9] hardly scales (already considering e.g. 15 stimuli and 15 aspects leads to timeouts of 3000 CPU seconds), the proposed approach can minimize given sets including thousands of stimuli in negligible run time (i.e. few seconds at most).

The remainder of this paper is structured as follows: The next section briefly reviews stimuli generation for simulation-based verification and motivates the considered work (including a discussion of related work). Afterwards, the proposed solution is described in Section III. Finally, the results obtained by our experimental evaluations are summarized in Section IV before the paper is concluded in Section V.

II. Background

In order to keep this work self-contained, this section briefly reviews the stimuli generation for the simulation-based verification of embedded systems. Based on this, we discuss the problem of minimizing the resulting sets of stimuli (which are very likely to be non-optimal) as well as the corresponding related work.

A. Stimuli Generation

The verification of embedded systems remains one of the most time consuming tasks in the design of circuits and systems. For the development of embedded systems at the *Electronic System Level* (ESL), SystemC [7], a C++-based modeling language used for embedded systems is frequently used within the industry. In this work, we assume that the system to verify, i.e. the *Design under Verification* (DUV), has been realized at the ESL level using SystemC. The verification of new embedded systems at the ESL abstraction layer is mostly performed by simulation-based verification [1]. The term simulation-based verification in this sense means that the verification is based on directly simulating or executing

```
int const BITWIDTH              = 8;
int const BITWIDTH_OPCODE       = 2;

sc_in<sc_uint<BITWIDTH> >                side_a;
sc_in<sc_uint<BITWIDTH> >                side_b;
sc_in<sc_uint<BITWIDTH_OPCODE> >         opcode;
sc_in<bool>                              carry_i;

sc_out<sc_uint<BITWIDTH> >      result;
sc_out<bool>                    carry_o;
sc_out<bool>                    zero;

void alu_core::calc(){

    sc_uint<BITWIDTH+1> out;
    switch(opcode){
        case alu_add:
            out = side_a + side_b + carry_i;
            break;
        case  alu_sub:
            out = side_a - side_b - carry_i;
            break;
        case alu_inc:
            out = side_a + 1 + carry_i;
            break;
        case alu_dec:
            out = side_a - 1 - carry_i;
            break;
    }

    result.write(out.range(BITWIDTH-1,0));
    carry_o.write(out[BITWIDTH]);
    zero.write(out == 0 ? 1 : 0);
}
```

Fig. 1: ESL model of an ALU.

the new system. By applying simulation techniques, the new system can be verified by using so called stimuli. The idea of simulation-based verification is to apply stimuli to the DUV and to observe its response. For stimuli-based verification of systems, the system is treated like a black box. The verification itself is performed by observing the behavior (i.e. outputs) of the DUV for certain known inputs (i.e. stimuli).

One of the most time consuming tasks in the simulation-based verification process is the generation of stimuli. Those stimuli can be generated by the SystemC designer himself/herself (which knows the design best), or they can be generated by the corresponding verification engineer. In both cases, the generation of stimuli causes a high effort and costs an enormous amount of human resources. To keep the needed effort for test stimuli generation as low as possible, automatic approaches have been proposed for this task (see e.g. [2]–[6]). The idea of those approaches is to exploit the knowledge of the underlying DUV for stimuli generation.

For the aspect extraction of the underlying DUV, i.e. which and how many aspects are there, either the design itself can be used, e.g. in form of source code, or formal documentation files can be used as input. Since the main language for ESL modeling is SystemC [7], software engineering based approaches can be also be applied here. Approaches which rely on the source code representation are either based on applying symbolic execution of the design [3], [12]–[14] or are working on top of instrumentation and *Control Flow Graph* (CFG) generation of the design [2]. Besides source code based stimuli generation approaches, there are also approaches which are based on documentations (e.g. the available datasheets) of the designs. Such approaches try to formulate the stimuli generation problem as an instance of the *Boolean satisfiability problem* (SAT problem) which can then be solved by invoking SAT-solvers to determine a solution for the problem [4], [6].

The major objective of automatic stimuli generation is to generate stimuli which cover as much as possible of the design. Almost all of the above mentioned approaches are capable to deliver a high coverage for given designs.

Example 1. *In this work, we are using an ESL model (i.e. a SystemC implementation) of an* Arithmetic Logic Unit *(ALU) as running example. The SystemC code of this ALU is shown in Figure 1. The ALU has four inputs (side_a, side_b, carry_i, opcode) and three outputs (result, zero, carry_o). To trigger e.g. the execution of the add instruction, the corresponding opcode has to be set to add. After that, the sum of the inputs side_a and side_b is written to the output result. Depending on the result of this instruction, the zero and carry outputs are going to be set.*

For the verification of the ALU design as shown in Figure 1, we have to provide a set of stimuli (in the following denoted S) to cover every aspect (i.e. feature) of the design (in the following, the set of all aspects to be considered as denoted A). To identify all aspects of the design, we can consider all branches of the design. If we can cover all branches of the design, we can achieve a coverage of 100%. For the identification of all aspects, we can employ the CFG which graphically represents all branches of the design. Applying the CFG helps us to achieve the maximum branch coverage possible for the design [4], [15].

TABLE I: Generated Stimuli for the ALU.

Stimuli $s_j \in S$	Aspects $a_i \in A$							
	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
s_1	1	0	0	0	1	0	0	1
s_2	0	1	0	0	1	0	1	0
s_3	0	1	0	0	1	0	0	1
s_4	0	1	0	0	1	0	1	0
s_5	0	0	1	0	1	0	0	1
s_6	0	0	0	1	1	0	1	1
s_7	0	0	1	0	0	1	0	1
s_8	1	0	0	0	0	1	0	1

However, often a single stimulus is capable of covering more than one aspect. Hence, the number $|S|$ of actually required stimuli is usually significantly smaller than the total number $|A|$ of aspects. Unfortunately, methods for stimuli generation usually determine sets of stimuli which are far from being minimal. Since having fewer stimuli directly translates into a much faster execution and, hence, verification of the considered system, this motivates the minimization of the number of stimuli.

Example 2. *Consider again the example from above and assume that eight stimuli* $S = \{s_1, \ldots, s_8\}$ *have been generated to cover all eight aspects* $A = a_1, \ldots, a_8$*. Since the CFG (which provides the basis for the aspects) is not composed of eight independent paths, it must be possible to trigger all aspects with less than eight stimuli. To evaluate that, Table I shows what aspects are covered by what stimuli (more precisely, a "1" denotes that a stimuli* $s_j \in S$ *covers an aspect* $a_i \in A$*, whereby a "0" denotes that this is not the case). From that, it can be concluded that e.g. an aspect coverage of 100% can be achieved by only using the stimuli* $S' = \{s_2, s_6, s_7, s_8\}$*, i.e. with just half of the originally generated stimuli.*

B. *Stimuli Minimization*

Observations such as conducted above motivated to consider minimizing the number of stimuli after their initial generation. However, minimizing the number of stimuli is a non-trivial task. Thus far, related work such as proposed in [9] addressed this by mapping the stimuli generation problem to a logic minimization problem. Here, methods like the Karnaugh maps or the Quine-McCluskey algorithms are available [16].

In order to use them, the original coverage problem is translated to a logic minimization problem by treating stimuli like prime-implicants and aspects like minterms. Since logic minimization aims for minimizing the number of prime-implicants while, at the same time, covering all minterms, this treatment can accordingly be applied to minimize stimuli while, at the same time, covering all aspects. The underlying method does not change but is simply applied to different entities.

	Aspects $a_i \in A$		
Stimuli $s_j \in S$	a_1	a_2	a_3
s_1	0	1	1
s_2	1	0	1
s_3	1	1	0

$$f(a,b,c) = \overline{a}bc + a\overline{b}c + ab\overline{c}$$

Fig. 2: Quine-McCluskey fails here.

Example 3. *Consider again the ALU from above and the eight stimuli $S = \{s_1, \ldots s_8\}$ together with their aspect coverage as provided in Table I. Treating aspects $a_i \in A$ as minters leads to the following Boolean functions in terms of a disjunction of prime-implicants:*

$$f(a_1 \ldots a_8) = a_1a_5a_8 + \cdots + a_1a_6a_8$$

Minimizing this function yields a reduced set of prime-implicants which still cover all aspects.

Unfortunately, approaches like Karnaugh maps or the Quine-McCluskey algorithm come with significant drawbacks. First, Karnaugh maps and Quine-McCluskey's algorithm have an exponential complexity and, hence, hardly scale. In fact, as discussed later in more detail in Section V, already for rather small instances considering e.g. 15 stimuli and 15 aspects only, these solutions run into a timeout of 3000 CPU seconds. Besides that, minimality is not guaranteed for the problem considered in this work. This is because the purpose of the Quine-McCluskey algorithm is the minimization of logic functions, not the minimization of stimuli. An example illustrates this problem.

Example 4. *Consider the coverage table shown in Figure 2. We can extract the minterms and the prime-implicants out of the table and apply Quine-McCluskey's algorithm. Since we know that we can transform the coverage table into a logic optimization problem that can be applied using Quine-McCluskey's algorithm, the table can be used for two different applications. A table in terms of coverage, i.e. a coverage table or in terms of a logic circuit, i.e. a truth table. If we apply the Quine-McCluskey algorithm for this minimization problem, the algorithm can not show up with any possible optimization. This is of course true, as this formula can not be more optimized without changing the formula itself. However, by referring to the coverage table shown in Figure 2, we can detect that we can cover all aspects by applying two stimuli only. By selecting s_1 and s_2, we could cover all aspects while only applying two of the three stimuli.*

Overall, no solution exists yet, which determines a minimal subset out of a given set of stimuli. In the next section, we are introducing an approach which, for the first time, is able to accomplish that.

III. Proposed Solution

This section presents the proposed solution for exact stimuli minimization. For this minimization problem, we propose to exploit the computational power of *MAX-SAT solvers*. Therefore, in this section we first review the MAX-SAT problem. Afterwards, we provide details how the stimuli minimization problem is encoded as a MAX-SAT instance.

A. MAX-SAT Problem

The *MAX-SAT* problem [10], [11] is an extension of the *Boolean satisfiability* (SAT) problem. Both problems are defined as follows:

Definition 1. *The* Boolean satisfiability problem *determines an assignment to the variables of a Boolean function $\Phi : \{0,1\}^n \to \{0,1\}$ such that Φ evaluates to 1 or proves that no such assignment exists. The function Φ is thereby given in* Conjunctive Normal Form *(CNF). Each CNF is a conjunction of clauses where each clause is a disjunction of literals and each literal is a propositional variable or its negation. A CNF is satisfied if all clauses are satisfied, a clause is satisfied if at least one literal is satisfied, and a positive (negative) literal is satisfied if the corresponding variable is set to 1 (0).*

Definition 2. *The* MAX-SAT problem *determines an assignment to the variables of a set of clauses Ψ, such that a maximum number of clauses is satisfied. Not necessary all clauses have to be satisfied – hence, they are called* soft constraints. *In addition to soft constraints, it is possible to define a set of clauses that are mandatory to be satisfied as in the pure satisfiability problem, so-called* hard constraints.

Example 5. *Let $\Phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_3)$. Then, $x_1 = 1, x_2 = 1$, and $x_3 = 1$ is a satisfying assignment solving the SAT problem.*

Accordingly, let $\Psi = \{\neg x_1 + \neg x_2, x_1, \neg x_1 + x_2\}$. Then, $x_1 = 0$ and $x_2 = 1$ is one possible solution to the MAX-SAT problem, allowing two clauses to evaluate to true.

Both, SAT- and MAX-SAT solvers employ intelligent decision heuristics, powerful learning schemes, and fast implication methods, which allow to efficiently traverse large search spaces [17], [18]. They have been proven to be very effective for many practically relevant design problems such as formal verification [19] (e.g. Model Checking) as well as test pattern generation for the post-silicon production test [20]. Our thesis is that this deductive power can also be utilized in order to determine the minimal set of required stimuli.

B. Problem Encoding

As already motivated in the last section, possibly only a subset of stimuli is needed in order to completely cover all aspects (e.g. by exploiting the fact that a single stimulus covers multiple aspects). In our approach, we formulate the question "How many and what stimuli have to be applied to the DUV so that all aspects are covered?" as a MAX-SAT instance. For this, we first provide a formalization of the input, second introduce the used variables, and finally formulate the applied constraints.

Input: As input to our approach, we use stimuli generated by *automatic stimuli generation engines* (e.g. [3]) or by invoking *Constrained Random Verification* (CRV) [1], [5].

Definition 3. *The input is then given as a set of stimuli $S = \{s_1 ... s_n\}$ (with $n \in \mathbb{N}$), a set of aspects $A = \{a_1 ... a_m\}$ (with $m \in \mathbb{N}$) as well as a value $c_{j,i}$ of each element of the Cartesian product of $S \times A$, which represent whether ($c_{j,i} = 1$) or not ($c_{j,i} = 0$) the stimuli s_j can cover aspect a_i.*

Variables: Next, we introduce the variables for the MAX-SAT instance. Therefore, we introduce Boolean variables which represent whether a stimulus is used or not. More formally:

Definition 4. *For each stimulus $s_j \in S$, we introduce a free Boolean variable $\mathbf{s_j}$*[1]*. These variables represent whether ($\mathbf{s_j} = 1$) or not ($\mathbf{s_j} = 0$) the stimulus s_j is used.*

Passing the $\mathbf{s_j}$-variables to a MAX-SAT solver would yield an arbitrary assignment to these variables. Hence, it is not guaranteed that all aspects are covered by at least one stimulus and that the number of used stimuli is minimal. Therefore, we have to restrict the assignments of these variables as follows.

Constraints: Since we formulate the problem as a MAX-SAT instance, in the following we have to differentiate between hard constraints (cf. all of them have to be satisfied) and soft constraints (cf. a maximum subset has to be satisfied). Furthermore, the required representation, i.e. the CNF clauses, can easily be derived from any Boolean function in linear time (see e.g. [21] and [22]). Hence, for sake of clarity we provide the following formulations in general pseudo-Boolean algebra.

The constraints have to ensure that all aspects of $a_i \in A$ are at least covered by one stimulus $s_j \in S$. Therefore, we have to ensure for each aspect $a_i \in A$ that at least one conjunction

[1] Note that variables for the MAX-SAT instance are written in bold font.

TABLE II: Experiments comparing Quince-McCluskey and MAX-SAT

			Quine-McCluskey Stimuli Optimization [16]			Exact Stimuli Optimization			
System	Number of Aspects	Number of Given Stimuli	Result. Stimuli	Total Runtime	Total Reduction	Result. Stimuli	Total Runtime	Total Reduction	Improvement to Quine-McCluskey
adpcm	10	10	6	6.5 s	4 (40%)	3	<0.1 s	7 (70%)	3 (50%)
adpcm	11	11	7	19.5 s	4 (36%)	3	<0.1 s	8 (72%)	4 (43%)
adpcm	15	15	-	timeout	-	3	<0.1 s	12 (80%)	-
adpcm	15	5000	-	timeout	-	3	1.5 s	4997 (99%)	-
aes128	10	10	7	8.3 s	3 (30%)	3	<0.1 s	7 (70%)	4 (43%)
aes128	50	50	-	timeout	-	5	<0.1 s	45 (90%)	-
aes128	50	5000	-	timeout	-	5	1.2 s	4995 (99%)	-
alu	8	8	7	1.1 s	1 (13%)	4	<0.1 s	4 (50%)	4 (43%)
euclid	8	7	5	<0.1 s	2 (29%)	3	<0.1 s	4 (57%)	2 (40%)
rsa	10	10	2	<0.1 s	8 (80%)	1	<0.1 s	9 (90%)	1 (50%)
rsa	20	20	2	<0.1 s	18 (90%)	1	<0.1 s	19 (95%)	1 (50%)
rsa	27	27	2	2820 s	25 (92%)	1	<0.1 s	26 (96%)	1 (50%)
arbiter	5	5	3	<0.1 s	2 (40%)	2	<0.1 s	3 (60%)	1 (33%)
arbiter	10	10	5	<0.1 s	5 (50%)	3	<0.1 s	7 (70%)	2 (40%)
arbiter	14	14	8	1080 s	6 (41%)	3	<0.1 s	11 (79%)	5 (63%)
b01	10	10	6	0.8 s	4 (40%)	4	<0.1 s	6 (60%)	2 (33%)
b01	20	20	-	timeout	-	6	<0.1 s	14 (70%)	-
b01	25	25	-	timeout	-	6	<0.1 s	19 (76%)	-

Note that the timeout which we have used for the experiments has been set to 3000 seconds.

between the $c_{j,i}$ and the $\mathbf{s_j}$-variable evaluates to true. In this case, the stimulus s_j allows to cover aspect a_i (i.e. $c_{j,i}$ is equal 1) and it is actually applied (i.e. the Boolean variable $\mathbf{s_j}$ is equal 1). Formally, this constraint is represented as:

$$\bigwedge_{1 \le i \le m} \bigvee_{1 \le j \le n} c_{j,i} \land \mathbf{s_j}.$$

Adding these constraints, we now have ensured that each aspect is covered by at least one stimulus. When we pass this formulation to a MAX-SAT solver, it returns an assignment of the $\mathbf{s_j}$-variables. In case, a variable $\mathbf{s_j}$ is assigned equal 1, the stimulus is used.

However, the MAX-SAT solver does not yet return a minimal set of stimuli which have to be applied. Therefore, we finally add soft constraints which ensure a minimal subset of stimuli but still ensures all constraints. Therefore, we add for each stimuli a soft constraint which tries to make the stimuli unused (i.e. the stimuli is unused when $\mathbf{s_j} = 0$). Formally, we add the following soft-constraints for all $\mathbf{s_j}$-variables: $\neg \mathbf{s_j}$.

When passing this formulation to a MAX-SAT solver, we now obtain a *minimal* set of stimuli which have to be executed in order to cover all aspects.

IV. Experiments

In order to evaluate the performance of the proposed approach, we implemented the solution described in Section III in C++ and applied it to several stimuli sets obtained from multiple test cases. To solve the resulting MAX-SAT instances, we utilized Z3 [23]. In this section, we summarize the obtained results. To this end, we first describe the setup as well as the considered test cases. Afterwards, the results are provided and discussed.

A. Setup and Benchmarks

For our experiments, we have considered multiple SystemC benchmarks taken from [9], [24], [25] and representing: nosep

- Adaptive Differential Pulse Code Modulation - (*adpcm*),
- Advanced Encryption Standard - (*aes128*),
- Arithmetic Logic Unit of a CPU, see Figure 1 - (*alu*),
- Euclidian algorithm - (*euclid*),
- Rivest-Shamir-Adleman cryptosystem - (*rsa*),
- Data processing algorithm for USB cores - (*arbiter*),
- FSM comparing serial flows - (*b01*)

For each of these systems, different sets of aspects to be covered have been considered. Then, sets of stimuli have been generated (using methods such as reviewed in Section II) which fully cover the respective aspects. With this as baseline, we afterwards applied two solutions for stimuli minimization, namely the Qunie-McCluskey algorithm (which has been used as a basis for the state-of-the-art solution [9]) and the MAX-SAT solution as described in Section III. All experiments have been conducted on an Intel i5 CPU with 8 GB of RAM running Arch Linux.

B. Results and Discussion

The results of the conducted experiments are shown in Table II. The table compares the performance of the Quine-McCluskey algorithm and the proposed MAX-SAT based approach. More precisely, the first columns provide the name of the system as well as the number of considered aspects and the originally determined number of stimuli. Afterwards, the results of the considered stimuli minimization methods are reported, including the resulting (minimized) number of stimuli, the run-time required to determined the minimized set, as well as the absolute and relative reduction. Note that, in order to evaluate the scalability of the approaches, we additionally considered a setting with a very high number of originally given stimuli (namely 5000 for the systems adpcm and aes). In order to observe the scalability with a very high number of stimuli, we have used the benchmark with the highest number of aspects, i.e. *aes128*, as well as the benchmark with an average number of aspects, i.e. *adpcm*.

First, it can clearly been observed that the previously proposed method based on Quine-McCluskey's algorithm hardly scales. Already for a rather small sets of stimuli (namely 15 e.g. for adpcm with 15 aspects), no minimization was possible within a given timeout of 3000 CPU seconds. In contrast, the proposed solution is highly scalable. Even for the actually rather artificially large amount of 5000 stimuli for the considered systems, results can be obtained in negligible run time (i.e. few seconds at most).

Moreover, in contrast to the previously proposed solution, the MAX-SAT approach even allows to determine an exact, i.e. minimal, result. That is, based on a given set of stimuli and their aspect coverage, the smallest possible set of stimuli can be obtained. This also allows to reduce the number of needed stimuli by further 63%. Overall, applying the proposed MAX-SAT based approach allows for determining optimally minimized results for every system. The execution time needed for the minimization always stayed less then 2 seconds even if a large number of stimuli has to be considered.

V. Conclusion

Since engineers for simulation-based verification are challenged with the ever increasing complexity of modern embedded systems, approaches for automatic stimuli generation have been proposed. Unfortunately those approaches do not always deliver optimal sets of stimuli in terms of their number of generated stimuli. In this paper, we proposed an approach for the minimization of stimuli for simulation-based verification. To this end, we employed solvers for the MAX-SAT problem. In contrast to previously proposed solutions for this problem, we were able to generate exact, i.e. minimal, results in a much more scalable fashion. Overall, this allows for significantly reducing the number of required stimuli even for larger systems and corresponding sets of stimuli.

Acknowledgements

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

REFERENCES

[1] J. Yuan, C. Pixley, and A. Aziz, *Constraint-based Verification*. New York, NY, USA: Springer, 2006.

[2] A. Dias Junior and D. C. da Silva Junior, "Code-coverage Based Test Vector Generation for SystemC Designs," in *IEEE Annual Symposium on VLSI*. Porto Alegre, RIG, Brazil: IEEE, 2007.

[3] B. Lin, Z. Yang, K. Cong, and F. Xie, "Generating High Coverage Tests for SystemC Designs Using Symbolic Execution," in *ASP Design Automation Conf.* Macau, China: IEEE, 2016.

[4] S. Yang, R. Wille, and R. Drechsler, "Improving Coverage of Simulation-based Verification by Dedicated Stimuli Generation," in *EUROMICRO Symp. on Digital System Design*. Verona, Italy: IEEE, 2014.

[5] F. Haedicke, H. M. Le, D. Große, and R. Drechsler, "CRAVE: An advanced constrained random verification environment for SystemC," in *2012 International Symposium on System on Chip*. Tampere, Finland: IEEE, 2012.

[6] 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*. Sophia Antipolis, France: IEEE, 2009.

[7] "IEEE Standard for Standard SystemC Language Reference Manual," *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)*, pp. 1–638, 2012.

[8] S. Yang, R. Wille, and R. Drechsler, "Coverage-driven Stimuli Generation," in *EUROMICRO Symp. on Digital System Design*. Cesme, Turkey: IEEE, 2012.

[9] K. Yamazaki, Y. Sekihara, T. M. Aoki, E. Hosoya, and A. Onozawa, "A heuristic algorithm for reducing system-level test vectors with high branch coverage," in *IEEE International Symposium on Circuits and Systems*. Rio de Janeiro, RJ, Brazil: IEEE, 2011.

[10] D. S. Johnson, "Approximation algorithms for combinatorial problems," *Journal of Computer and System Sciences*, pp. 256–278, 1974.

[11] P. Hansen and B. Jaumard, "Algorithms for the maximum satisfiability problem," *Journal of Computing*, pp. 279–303, 1990.

[12] G. Li, I. Gosh, and S. P. Rajan, "KLOVER: A Symbolic Execution and Automatic Test Generation Tool for C++ Programs," in *Computer Aided Verification*. Snowbird, UT, USA: Springer, 2011.

[13] K. Sen, D. Marinov, and G. Agha, "CUTE: A Concolic Unit Testing Engine for C," in *European Software Engineering Conference*. Lisbon, Portugal: ACM, 2005.

[14] P. Gonzales de Aledo, N. Prizigoda, R. Wille, R. Drechsler, and P. Sanchez, "Towards a Verification Flow Across Abstraction Levels: Verifying Implementations Against Their Formal Specification," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, pp. 475–488, 2017.

[15] S. Tasiran and K. Keutzer, "Coverage metrics for functional validation of hardware designs," *IEEE Design & Test of Comp.*, pp. 36–45, 2001.

[16] T. Sasao, *Switching Theory for Logic Synthesis*. Norwell, MA, USA: Kluwer Academic Publishers, 1999.

[17] A. Biere, "Picosat essentials," *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, pp. 75–97, 2008.

[18] R. Wille, G. Fey, D. Große, S. Eggersglüß, and R. Drechsler, "Sword: A sat like prover using word level information," in *International Conference on Very Large Scale Integration*. Atlanta, GA, USA: IEEE, 2007.

[19] A. Biere, "Tutorial on Model Checking: Modelling and Verification in Computer Science," in *International Conference Algebraic Biology*. Hagenberg, Austria: ACM, 2008.

[20] S. Eggersglüß, R. Wille, and R. Drechsler, "Improved SAT-based ATPG: More Constraints, Better Compaction," in *International Conference on CAD*. San Jose, CA, USA: IEEE, 2013.

[21] G. Tseitin, "On the complexity of derivation in propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logic*. Heidelberg, Germany: Springer, 1968.

[22] N. Eén and N. Sörensson, "Translating Pseudo-Boolean Constraints into SAT," *Journal on Satisfiability, Boolean Modeling and Computation*, pp. 1–26, 2006.

[23] N. de Moura, Leonardoand Bjørner, "Z3: An efficient SMT solver," *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, 2008.

[24] B. C. Schafer and A. Mahapatra, "S2CBench: Synthesizable SystemC Benchmark Suite for High-Level Synthesis," *IEEE Embedded Systems Letters*, pp. 53–56, 2014.

[25] E. Clarke, H. Jain, and D. Kroening, "Verification of SpecC using predicate abstraction," *Formal Methods in System Design*, pp. 5–28, 2007.