Open access • Book Chapter • DOI:10.1007/978-90-481-9304-2_14

SMT-based stimuli generation in the SystemC Verification library — Source link

Robert Wille, Daniel Grosse, Finn Haedicke, Rolf Drechsler

Institutions: University of Bremen

Published on: 01 Sep 2009 - Forum on specification and Design Languages

Topics: Electronic system-level design and verification, SystemC, Satisfiability modulo theories, Hardware description language and Electronic design automation

SMT-based Stimuli Generation in the SystemC Verification Library

Robert Wille Daniel Große Finn Haedicke Rolf Drechsler
Institute of Computer Science, University of Bremen
28359 Bremen, Germany
{rwille,grosse,finn,drechsle}@informatik.uni-bremen.de

***Abstract*—Modelling at the Electronic System Level (ESL) is the established approach of the major System-on-chip (SoC) companies. While in the past ESL design covered design methodologies only, today also verification and debugging is included. To improve the verification process, testbench automation has been introduced highlighted as constraint-based random simulation. In SystemC – the de facto standard modelling language for ESL – constraint-based random simulation is available through the SystemC Verification (SCV) library. However, the underlying constraint-solver is based on Binary Decision Diagrams (BDDs) and hence suffers from memory problems.**

In this paper, we propose the integration of new techniques for stimuli generation based on Satisfiability Modulo Theories (SMT). Since SMT solvers are designed to determine a single satisfying solution only, several strategies are proposed forcing the solver to generate more than one stimuli from different parts of the search space. Experiments demonstrate the advantage of the proposed approach and the developed strategies in comparison to the original BDD-based method.

***Keywords*-Constraint-based random simulation, SAT Modulo Theories, SystemC Verification Library**

I. Introduction

The *Electronic System Level* (ESL) emerged to deal with both, the (ever increasing) design complexity and the resulting verification crisis. Hence, the languages for ESL design, e.g. SystemVerilog or SystemC, have been enhanced to support inevitable verification methodologies like assertions and testbench automation. Especially the second methodology, highlighted as *constraint-based random simulation*, is a key technique to cope with current and future designs [1], [2], [3]. Thereby, stimuli from specified constraints are generated by means of a constraint solver so that corner case scenarios are triggered. In doing so, design bugs will be found that might otherwise remain undetected.

In the context of SystemC – the de facto standard modelling language for ESL – constraint-based random simulation is available in the *SystemC Verification* (SCV) library. Here, the underlying constraint-solver uses *Binary Decision Diagrams* (BDDs) [4] for constraint representation and subsequent stimuli generation. Due to the well-known limits of BDDs, complex arithmetic or non-trivial implications can not efficiently be handled by SCV constraints.

In this work, we introduce an alternative to the BDD-based stimuli generation. Motivated by the solving paradigms introduced in the recent past, the BDD-based constraint solver is replaced by a *Satisfiability Modulo Theory* (SMT) approach. SMT extends *Boolean Satisfiabily* (SAT), a solving technique on the pure Boolean level that has been intensely studied over the last two decades. SMT maintains all benefits of SAT but furthermore allows the exploitation of additional information due to the representation of the instances on the word level. This often gives significant speed-ups compared to pure SAT solving (see e.g. [5], [6], [7]).

While, at a first glance the substitution of BDDs by SMT techniques seems to be simple, two major tasks have to be tackled:

1) BDDs represent all solutions of a constraint at once. Hence, if the BDD can be built, stimuli generation is nothing else than a linear traversal from the root node to a 1-terminal respecting a uniform distribution. In contrast, SMT solvers return a single solution only. Thus, strategies have to be developed to determine more than one stimuli with good distributions.
2) For SAT solving, MiniSat [8] is the state-of-the-art solver today. However, in the domain of SMT the results are not so clear as e.g. the recent SMT competition shows [9]. Moreover, most of the SMT solvers (in particular the most efficient ones) are closed source. Thus, the proposed strategies for determining good distributions can only use the provided interface of an SMT solver restricting the strategies to a certain extend.

To the best of our knowledge SMT-based stimuli generation has not been considered so far. In [10], an SMT-like static analysis is used to reduce the search space before applying SAT- or BDD-based stimuli generation. Besides that, only approaches exploiting SAT techniques have been proposed. Thereby, similar issues must be addressed. In [11], a specialized SAT algorithm in combination with so called randomized XOR constraints are used to achieve even distributed solutions. Instead, the authors in [12] presented a sampling algorithm that combines concepts from the Metropolis-Hastings algorithm, Gibbs sampling, and WalkSAT to efficiently generate solutions to the constraints with an approximately uniform distribution. In our work, we propose a variety of strategies (including special blocking constraints, solution selection, and handling overconstraining) considering the particular needs of currently available SMT solvers.

The remainder of this paper is structured as follows: The next section briefly describes the SCV library, BDD-based constraint solving, and the main concepts of SMT. After discussing the limits of BDD-based constraint solving our SMT-based approach is introduced in Section III. Section IV describes strategies to generate distributed solutions. Finally, experimental results as well as conclusions with outlooks on future work are given in Section V and Section VI, respectively.

II. Preliminaries

To keep the remainder of this paper self-contained, this section briefly introduces the SystemC Verification Library which enables constraint-based random simulation for SystemC. Afterwards, SMT and the respective solve engines are reviewed. Readers wishing more in-depth treatment are referred to the references.

```
struct cstr : public scv_constraint_base {
  scv_smart_ptr<sc_uint<64> > a,b,addr;
  SCV_CONSTRAINT_CTOR(cstr) {
    SCV_CONSTRAINT( a() > 100 );
    SCV_CONSTRAINT( b() == 0 );
    SCV_CONSTRAINT(addr()>=0 && addr()<=0x400);
  }
};
```

Fig. 1. Example constraint

A. SystemC Verification Library

The SCV library was introduced in 2002 as an open source C++ class library [13], [14], [15] on top of SystemC [16], [17]. In the following we focus only on the basic features of the SCV library for constraint-based randomization.

Using the SCV library, constraints are modelled in terms of C++ classes. That way constraints can be hierarchically layered using C++ class inheritance. In detail a constraint is derived from the `scv_constraint_base` class. The data to be randomized is specified as `scv_smart_ptr` variables.

Example 1: Fig. 1 shows an example of an SCV constraint with the name *cstr*. Here, the three unsigned integer variables a, b, and $addr$ are randomized. The conditions on the variables a, b, and $addr$ are defined by expressions in the respective *SCV_CONSTRAINT()* macro.

Internally, a constraint in the SCV library is given by the corresponding characteristic function, i.e. the function is true for all solutions of the constraint. This characteristic function of a constraint is represented as a BDD, a canonical and compact data structure for Boolean functions [4]. For stimuli generation a weighting algorithm is applied to the constraint BDD to guarantee a uniform distribution of all constraint solutions and hence maximizing the possibility for entering unexplored regions of the design state space (see [13] and for improvements see [18], [19]). As BDD package CUDD [20] is used in the SCV library.

B. SAT Modulo Theory

SAT Modulo Theory (SMT) is an extension of the *Boolean satisfiabiliy* problem (SAT problem) which is defined as follows:

Definition 1: Let f be a Boolean function. Then, the SAT problem is to determine an assignment α to all variables of f such that $f(\alpha) = 1$ or to prove that no such assignment exists. In the former case f is *satisfiable*; otherwise f is *unsatisfiable*.

Usually, the Boolean function f is given in *Conjunctive Normal Form* (CNF), i.e. a product-of-sum representation. For this representation, several (backtracking) algorithms (so called *SAT solvers*) have been proposed in the past (e.g. [21], [22], [23], [8]). Most of them are based on three essential procedures: (1) The decision heuristic assigns values to free variables, (2) the propagation procedure determines implications due to the last assignment(s), and (3) the conflict analysis tries to resolve conflicts by backtracking that occur during the search. Thereby, advanced techniques as e.g. *efficient Boolean constraint propagation* [23] or *conflict analysis* [22] are exploited. These enable the consideration of problems with more than hundreds of thousands of variables and clauses. Thus, today SAT solvers are core solve engines for many application domains. Thereby, the real world problem is transformed into CNF and then solved by using a SAT solver as black box.

However, due to the increasing complexity of many problems, researchers investigated the use of higher levels of abstractions than CNF – by still exploiting the established SAT techniques. This leads to the development of SMT solvers. Here, the satisfiability problem is expressed e.g. by *Quantifier-Free Bit-Vector* (QF_BV) logic which is defined as follows:

Definition 2: A *bit-vector* is an element $\vec{b} = (b_{n-1}, \ldots, b_0) \in \mathbb{B}^n$. The *index* $[\,]: \mathbb{B}^n \times [0, n) \to \mathbb{B}$ maps a bit-vector $\vec{b}$ and an index i to the i^{th} component of the vector, i.e. $\vec{b}[i] = b_i$. The conversion from (to) a natural number is defined by $\mathrm{nat} : \mathbb{B}^n \to N$ ($\mathrm{bv} : N \to \mathbb{B}^n$) with $N = [0, 2^n) \subset \mathbb{N}$ and $\mathrm{nat}(\vec{b}) := \Sigma_{i=0}^{n-1} b_i \cdot 2^i$ ($\mathrm{bv} := \mathrm{nat}^{-1}$).

Problems can be specified by using bit-vector operations as well as arithmetic operations. Let $\vec{a}, \vec{b} \in \mathbb{B}^n$ be two bit-vectors. Then, the *bit-vector operation* $\circ \in \{\wedge, \vee, \ldots\}$ is defined by $\vec{a} \circ \vec{b} := (\vec{a}[n-1] \circ \vec{b}[n-1], \ldots, \vec{a}[0] \circ \vec{b}[0])$. An *arithmetic operation* $\bullet \in \{*, +, \ldots\}$ is defined by $\vec{a} \bullet \vec{b} := \mathrm{nat}(\vec{a}) \bullet \mathrm{nat}(\vec{b})$. For brevity, in the following $\mathrm{nat}(\vec{a})$ is also written as a.

Satisfiability problems in QF_BV logic are solved either by (1) using a combination of a traditional SAT solver and a specialized (bit-vector) theory solver (see e.g. [24], [25]), (2) pre-processing the instance exploiting the higher level of abstraction before bitblasting it to a traditional SAT solver (see e.g. [26], [27]), or (3) using specialized solvers that directly work on the bit-level of the problem (e.g. [28], [29]).

III. SMT-based Stimuli Generation

This section introduces the SMT-based approach for constraint-based stimuli generation. As described in Section II-A, the current constraint-solver of the SCV library uses BDDs. Limits of this method are discussed in the first part of this section. Then, we introduce the SMT approach which replaces the BDD-based constraint-solver. Using this as a basis, strategies to determine good distributions are given in the next section.

A. Limits of BDDs

In the SCV library, constraints are represented by means of BDDs. This allows a compact representation of the respective characteristic function. Furthermore, a uniform distribution among all possible stimuli is ensured.

The flow of the BDD-based stimuli generation is depicted in the upper part of Fig. 2. By instantiating the SCV constraint, an expression tree of the constraint is generated. Afterwards, this tree is traversed to build the respective BDD. Stimuli are obtained by randomly selecting a weighted 1-path of the BDD which represents a satisfying assignment to the constraint [30]. Using the improvements for the SCV library as described in [18] a uniform distribution results, even if constraint variables are fixed to certain values.

However, even though BDDs are very successful as a compact data-structure for large Boolean functions, their capabilities are limited. More precisely, for many (practical relevant) functions no compact BDD can be built. As an example, it has been proven, that BDDs representing multiplication always have exponential size with respect to the number of input variables regardless of the variable ordering [31]. Since arithmetic operations like multiplication often occur in SCV constraints, this becomes the bottleneck of stimuli generation for many systems.

Besides that, also the above mentioned uniform distribution of stimuli is often sub-optimal. In many cases a uniform distribution is desirable not with respect to all solutions, but with respect to the control parts of the device under verification. For example, if stimuli for an *Arithmetic Logic Unit* (ALU) are generated, each ALU-operation should be

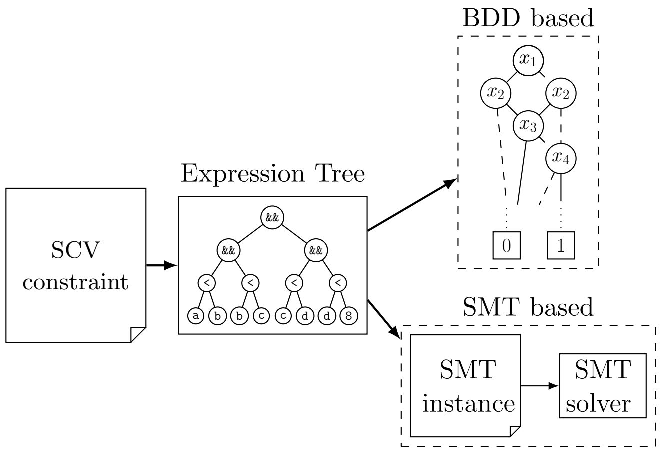

Fig. 2. Flows in the SCV library

TABLE I
SCV TO SMT (QF_BV) MAPPING

SCV	SMT
EQUAL	=
NOT_EQUAL	not (=)
GREATER_THAN	bvugt,bvsgt
LESS_THAN	bvult,bvslt
GREATER_OR_EQUAL	bvuge,bvsge
LESS_OR_EQUAL	bvule,bvsle
AND	and
OR	or
NOT	not
PLUS	bvadd
MINUS	bvsub
MULTIPLY	bvmul
BITNOT*	bvnot
BITAND*	bvand
BITOR*	bvor
BITXOR*	bvxor
BITSELECT*	extract
BITSLICE*	extract

*Extensions from [18].

tested by the same number of stimuli. However, using the current BDD-based SCV implementation this is not possible (see also Section V).

B. Integrating SAT Modulo Theories

To overcome the mentioned limitations (in particular the capability problems), SMT-based constraint solving is proposed in this section. To this end, the BDD engine is replaced by an off-the-shelf SMT solver as depicted in the lower part of Fig. 2. This requires the SCV constraint to be encoded as an SMT instance first.

Table I shows the operations available in the SCV library as well as their counterparts in QF_BV logic. As can be seen, most of the SCV operations can be represented directly by QF_BV operations. The same holds for most of the variable definitions that can be converted to bit-vector variables as defined in Definition 2. In doing so, the SCV constraint can be easily encoded as an SMT instance which is afterwards passed to the respective solver. In contrast to BDDs, this lifts the solve process from the Boolean level to a higher abstraction, where the original operations (in particular arithmetic) are preserved and thus the solve engine can exploit more information.

As a drawback, SMT solvers are designed to determine a single satisfying solution, i.e. only one stimulus is generated for each instance. To generate more than one stimulus, solutions must be blocked, otherwise the same solution is found again due to the deterministic behavior of the SMT solver. Therefore, new constraints excluding already found solutions from the solution space are added. Then, the solver is restarted which leads to a new solution and stimulus, respectively. This is repeated until either the desired number of stimuli have been generated or the instance becomes unsatisfiable (i.e. all solutions have been determined).

Example 2: Consider the SCV constraint given in Fig. 1. Encoding this into an SMT instance and solving it with an SMT solver may lead to the first stimulus $a = 101$, $b = 0$, and $addr = 1$. To block this solution, an additional SCV constraint stating $a \neq 101 \vee b \neq 0 \vee addr \neq 1$ is added. Thus, a new valid solution could be $a = 102$, $b = 0$, and $addr = 1$.

IV. Distribution of Generated Stimuli

Performing SMT-based stimuli generation as described in Section III-B leads to sub-optimal distributions of generated stimuli. More precisely, due to the blocking of previously found solutions two consecutive stimuli may only slightly differ, e.g. in one variable (see Example 2). In contrast, solutions from a significant different part of the search space or triggering a new operation of the device under verification are desired. In this section, we introduce strategies for SMT-based stimuli generation resulting in well-distributed solutions. Thereby, specialized types of blocking constraints, random pre-assignments, as well as utilization of control variables are applied. Furthermore, strategies for selecting the obtained solutions as stimuli are presented. Finally, we show how the case of overconstraining (caused by an iterative use of the techniques) can be handled.

A. Determine Distributed Solutions

To achieve good distributions of stimuli, the solution space of the respective SMT instance must be restricted, so that the solver is forced to determine solutions from different areas of the search space. For this task, three strategies are introduced in the following.

The first one uses blocking constraints as already introduced in Section III-B. In total, four variations of blocking constraints are proposed:

- *Total Blocking*
  The complete assignments to previously found solutions are blocked. This is equal to the blocking constraints introduced in the last section and ensures, that the same assignments will not be determined by the solver again. As a result, a very small part of the solution space is blocked and further solutions might be quite similar to the already found ones (i.e. the overall distribution might become sub-optimal).
- *Partial Blocking*
  Only partial assignments of the previously found solutions are considered, i.e. some variable assignments are blocked while others remain unrestricted. As a result, a significantly larger part of the search space (the 'surroundings' of the already found solutions) is excluded. This forces the solver to find more different stimuli but may lead to an early overconstraining (i.e. an exclusion of *all* remaining solutions).
- *Interval Blocking*
  Interval blocking is a special case of partial blocking, where – based on the already found solutions – intervals are excluded from the search space.
- *Bitmask Blocking*
  Some bits of the assignments to previously found solutions are blocked. To this end, a random bitmask is generated and compared to the variable assignments.

```
struct ALU : public scv_constraint_base {
  scv_smart_ptr<sc_bv<2> >  op; // control
  scv_smart_ptr<sc_uint<12> > a, b;

  SCV_CONSTRAINT_CTOR(ALU12) {
    SCV_CONSTRAINT ( (op() != 0x0) ||
        ( 4095 >= a() + b() ) );
    SCV_CONSTRAINT ( (op() != 0x1) ||
        ((4095 >= a() - b())
        && (b() <= a()) ) );
    SCV_CONSTRAINT ( (op() != 0x2) ||
        ( 4095 >= a() * b() ) );
    SCV_CONSTRAINT ( (op() != 0x3) ||
        ( b() != 0 ) );
  }
};
```

Fig. 3. Example constraint

Where the bitmask is one the respective assignment is blocked.

Example 3: Again, consider the SCV constraint given in Fig. 1. The following constraints can be added to generate distributed solutions:
Total Blocking: $a! = 101 \vee b! = 0 \vee addr! = 1$
Partial Blocking: $a! = 101$
Interval Blocking:
$a < 50 \vee a > 150 \vee b < 0 \vee b > 50 \vee addr < 0 \vee addr > 50$
Bitmask Blocking:
$\vec{a}[57] \neq 0 \wedge \vec{a}[52] \neq 0 \wedge \cdots \wedge \vec{a}[0] \neq 1 \wedge \vec{b}[61] \neq 0 \wedge \ldots$

As second strategy, variables are (partially) pre-assigned to direct the SMT solver into other parts of the search space. Thereby, random values are used and applied to randomly chosen bits of the SCV variables. In doing so, widespreaded solutions may result, but the possibility of overconstraining increases since a pre-assignment can be applied for which no satisfying solution exists any longer.

Finally, the third strategy exploits information of control and data paths of the device under verification. Here, in addition to the SCV constraint, the verification engineer also provides whether an SCV variable belongs to the control path (e.g. the variable controls the operation of an ALU). Then, to obtain well-distributed solutions with respect to all possible operations of the device under verification, the following strategies can be applied:

- *Force a Change*
  Two consecutive solutions include different assignments to the control variables. This ensures, that consecutive stimuli are generated, triggering the simulation of two different operations.
- *Assign Control Signals*
  Pre-assigning the control variables to (random) values. This ensures, that different (random) operations are simulated.

Example 4: Consider the SCV constraint in Fig. 3, which is used as basis for stimuli generation of an ALU. The variable *op* is marked as control variable that selects the respective operation to be performed.

Force a Change: Let $op = 1$, $a = 411$, $b = 31$ be a determined solution. Then, the constraint $op \neq 1$ is added for the next stimulus generation.

Assign Control Signals: Since in total there are four possible assignments to *op*, for each stimulus generation run, one of the constraints $op = 0$, $op = 1$, $op = 2$, or $op = 3$ is added, respectively, while the previous constraint is removed.

B. Choosing Solutions

The strategies introduced so far help to obtain different solutions. Besides that, how to select solutions as stimuli has an effect on the resulting distributions. To this end, the following methods are proposed:

- *Skip Solutions*
  Instead of using each determined solution a number of solutions can be skipped. Because the skipped stimuli are still generated – but not used – the applied strategy still generates constraints for these. This can help traversing different parts of the solution space.
- *Consider All Solutions*
  The best results regarding the distribution can be achieved by determining all solutions. Then, each solution can be selected with equal probability. On the other hand, this obviously requires a large amount of run-time and memory, respectively. Thus, this is only possible for small solution spaces.
- *Apply FIFO*
  A trade-off to the consideration of all solutions is the application of a FIFO structure. In this structure, a certain number of solutions is stored. Each time a stimulus is selected, older solutions are replaced by new ones.

C. Handling Overconstraining

The proposed strategies constrain the search space in different ways. Some strategies are very restrictive so that an overconstraining may occur very fast, i.e. constraints are added such that the resulting SMT instance becomes unsatisfiable. To handle this, some of the additional constraints must be removed. Therefore, the following strategies are proposed:

- *Remove All Random Constraints*
  Remove all constraints including random elements (e.g. random pre-assignments or bitmasks). This may lead to duplications.
- *Remove Old Constraints (FIFO)*
  Remove the oldest constraint that have been added. If then the instance is still unsatisfiable, remove the next-oldest element until the overconstraining is solved. For this purpose, a FIFO structure is used to store the constraints.

V. EXPERIMENTAL EVALUATION

The SMT-based stimuli generator as well as the strategies for determining distributed solutions as introduced in Section III and Section IV, respectively, have been implemented in C++ and integrated into the SCV library. In the following we give a comparison of the proposed techniques (using Boolector [27] as SMT solver) and the original BDD-based SCV library.

The following SCV constraints have been used to demonstrate the effects[1]:

- *bv256_neq*: A comparison between two 256-bit variables
- *add*: An addition of two 32-bit variables where the sum is forced to be equal to 999999
- *mult32*: A multiplication of two 32-bit variables where no further restrictions are applied (*_abc*) and where the product is forced to be within the interval $[1027, 1072693248]$ (*_uplow*), respectively

[1]Due to page limitation only a small set of benchmarks is presented here. However, the benchmarks include representatives of arithmetic, control logic, as well as a mixture of both and thus are sufficient to show the effects.

- *ALUXX*: An ALU constraint as the one depicted in Fig. 3 where *XX* denotes the bit-width of the data variables

For measuring the distribution of the solutions the normalized Hamming distance (*ham*) has been used. More precisely, the sum of the hamming distances for all generated stimuli divided by the number of all distances is applied (i.e. the higher the value of the normalized hamming distance, the better the distribution). Besides that, also the number of duplicates (*dupl*) is considered to evaluate the quality of the distribution.

In our experiments, for each SCV constraint 1000 stimuli are generated using the following methods:

- BDD: The original SCV environment based on BDDs
- SMT-TB: The naive SMT-based stimuli generation using total blocking (TB) of previous solutions
- SMT-BB_OR: The SMT-based stimuli generation using bitmask blocking (BB) to avoid previously found solutions; in case of overconstraining, the oldest blocking constraint is removed (OR)
- SMT-BB_OR_F64,4: The SMT-BB_OR strategy (see above) combined with a FIFO storing 64 solutions from which 4 elements are replaced each time (F64,4)
- SMT-INT: The SMT-based stimuli generation using interval constraints (INT) to avoid previously found solutions
- SMT-INT_OR: The SMT-based stimuli generation using interval constraints (INT) to block previously found solutions; in case of overconstraining, the oldest blocking constraint is removed (OR)
- SMT-PB_OR: The SMT-based stimuli generation using partial blocking (PB) to avoid previously found solutions; in case of overconstraining, the oldest blocking constraint is removed (OR)
- SMT-TB_PAV: The SMT-based stimuli generation using total blocking (TB) and (partially) pre-assigned variables (PAV) to avoid previously found solutions
- SMT-TB_PAV_CC: The SMT-TB_PAV strategy (see above) combined with forcing changes on control variables (CC)
- SMT-TB_PAV_PAC: The SMT-TB_PAV strategy (see above) combined with pre-assigning of control variables (PAC)

In the experiments, for each benchmark a timeout of 900 CPU seconds has been set using an AMD Athlon 3500+ with 1 GB of main memory.

The resulting run-times (in CPU-seconds) and the information for the distribution (normalized hamming distance and number of duplications) are given in Table II and Table III, respectively. In case of timeout (denoted by *) or overconstraining (denoted by †) only the number of stimuli generated so far are shown in Table II. Note that, to compare the quality of the approaches, both tables (i.e. run-time *and* distribution) must be considered together.

As can be seen, the original SCV environment (BDD) easily handles *bv256_neq* and *uaubc999999* but suffers from time outs as soon as multipliers occur in the respective SCV constraint. In contrast, the naive SMT-based approach (SMT-TB) is able to generate the desired 1000 stimuli for all benchmarks in reasonable time (see first two rows of Table II). However, the distribution obtained by the SMT-based approach is poor (see first two rows of Table III). This emphasizes the need for advanced strategies as proposed in Section IV.

But not all developed strategies lead to the desired results (i.e. fast stimuli generation *and* good distribution). More precisely, SMT-BB_OR_F64,4, SMT-INT, SMT-BB_OR, and SMT-INT_OR are not applicable since they cannot complete all benchmarks within the timeout. Furthermore, SMT-PB_OR is not suitable due to the high number of generated duplicates (e.g. for *ALU12* 696 of the 1000 generated stimuli are duplicates). Thus, all approaches listed in rows 3-7 of Table II and Table III, respectively, are not applicable due to the high run-time or low quality of the distribution.

In contrast, the remaining strategies SMT-TB_PAV, SMT-TB_PAV_CC, and SMT-TB_PAV_PAC (lower rows of Table II and Table III) offer the best trade-off between run-time and distribution. All of them directly exclude previous found solutions and thus avoid duplicates. Furthermore, the hamming distance is significantly higher than for the naive SMT approach (SMT-TB) while the run-time remains moderate.

Finally, the effect of the strategies that incorporate control path information are considered in detail. This is done by considering the example of *ALU12*. Fig. 4 shows the distribution (with respect to the stimulated operations) for the respective approaches (i.e. SMT-TB_PAV, SMT-TB_PAV_CC and SMT-TB_PAV_PAC) as well as for BDD and SMT-TB. Here, the poor distribution of SMT-TB is not surprising. But also the BDD-based approach does not lead to a uniform distribution. Thus, to achieve a uniform distribution over all operations of the device under verification, other methods must be used. In this case, the strategy SMT-TB_PAV_PAC, which incorporates the control variables, provides the best distribution of the stimuli.

Altogether, for the general case good results are obtained by applying the strategies SMT-TB_PAV, SMT-TB_PAV_CC, and SMT-TB_PAV_PAC. If additionally control path information is available, a uniform distribution over all operations can be achieved using strategy SMT-TB_PAV_PAC.

VI. Conclusions and Future Work

In this work, SMT-based stimuli generation for the SCV library has been introduced. We showed, that using an SMT solver instead of BDDs leads to a more robust stimuli generation (in particular if complex arithmetic occurs in an SCV constraint as this is the case for many practical relevant problems). As a drawback, simply replacing the BDD engine by an SMT solver leads to poor results with respect to the distribution of constraint solutions. Thus, we proposed several strategies forcing the solver to generate stimuli from different parts of the search space. Our experiments show the effectiveness of these strategies.

In future work, we plan a tight integration of the proposed concepts into the SCV library, i.e. without generating a separate SMT instance for each stimulus. For this purpose, e.g. the recently published C-interface of Boolector [27] can be used. Furthermore, strategies directly integrated in the solve engine should be developed. More precisely, instead of excluding unwanted solutions by means of additional constraints, the search process of the solver should be influenced e.g. by adjusting the decision heuristics or by modifying propagation strategies so that parts of the search space are pruned depending on the current (partial) assignment.

References

[1] J. Yuan, A. Aziz, C. Pixley, and K. Albin, “Simplifying boolean constraint solving for random simulation-vector generation,” *IEEE Trans. on CAD*, vol. 23, no. 3, pp. 412–420, 2004.

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

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

TABLE II
RUNTIMES (CPU SECONDS)

	bv256_neq	add	mult32_abc	mult32_uplow	ALU12	ALU16	ALU32
BDD	**0,00**	**0,00**	(*0)	(*0)	33,96	(*0)	(*0)
SMT-TB	720,40	90,13	437,15	**271,12**	5,57	5,91	5,94
SMT-BB_OR_F64,4	(*227)	(*630)	(*161)	(*219)	97,52	(*882)	(*579)
SMT-INT	(*559)	(†3)	(*562)	(†36)	(†523)	(†268)	(*973)
SMT-BB_OR	(*950)	110,43	(*719)	(*830)	6,80	6,72	7,64
SMT-INT_OR	(*580)	10,75	(*618)	362,74	8,24	6,37	7,20
SMT-PB_OR	312,83	54,76	419,85	320,53	**3,49**	**3,35**	**2,61**
SMT-TB_PAV	356,96	282,12	**121,84**	288,33	6,35	6,10	6,08
SMT-TB_PAV_CC	351,66	280,50	**122,68**	292,40	7,07	6,89	7,41
SMT-TB_PAV_PAC	345,59	270,89	**121,50**	288,64	7,71	7,31	7,16

In case of overconstraining (denoted by †) or timeout (denoted by *), respectively, the number of generated stimuli so far is given instead.

TABLE III
DISTRIBUTION (NORMALIZED HAMMING DISTANCE, NUMBER OF DUPLICATES)

	bv256_neq		add		mult32_abc		mult32_uplow		ALU12		ALU16		ALU32	
	ham	dupl	ham	dupl	ham	dupl	ham	dupl	ham	dupl	ham	dupl	ham	dupl
BDD	**127.70**	0	10.01	0	–	–	–	–	**6.42**	0	–	–	–	–
SMT-TB	2.50	0	5.05	0	2.50	0	3.69	0	2.50	0	2.50	0	2.50	0
SMT-BB_OR_F64,4	6.17	39	8.79	132	12.83	33	6.00	46	5.79	246	**6.62**	169	6.41	92
SMT-INT	6.35	0	6.50	0	**14.61**	0	3.32	0	4.96	0	4.18	0	**10.19**	0
SMT-BB_OR	5.96	0	**18.29**	0	10.13	0	5.14	0	4.99	0	5.75	0	5.64	0
SMT-INT_OR	5.98	0	7.46	755	7.06	0	3.69	366	5.05	6	4.85	76	7.51	0
SMT-PB_OR	3.97	220	6.66	193	9.69	73	**6.74**	204	2.70	696	2.77	693	2.65	742
SMT-TB_PAV	48.29	0	7.29	0	6.07	0	5.04	0	3.05	0	3.20	0	3.06	0
SMT-TB_PAV_CC	49.10	0	7.47	0	6.21	0	5.08	0	3.23	0	3.80	0	6.54	0
SMT-TB_PAV_PAC	47.85	0	7.38	0	6.23	0	5.17	0	3.38	0	3.97	0	6.79	0

The normalized hamming distance (ham) and the number of duplicates (dupl) for the generated stimuli (up to 1000)

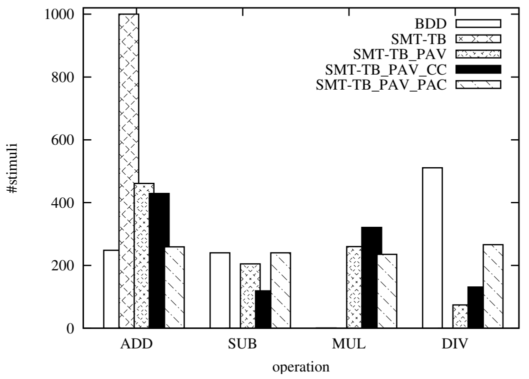

Fig. 4. Distribution with respect to operations

[4] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 677–691, 1986.

[5] A. Armando, J. Mantovani, and L. Platania, "Bounded model checking of software using SMT solvers instead of SAT solvers," in *SPIN*, 2006, pp. 146–162.

[6] S. K. Lahiri, R. Nieuwenhuis, and A. Oliveras, "SMT techniques for fast predicate abstraction," in *Computer Aided Verification*, 2006, pp. 424–437.

[7] Z. Zeng, K. R. Talupuru, and M. J. Ciesielski, "Functional test generation based on word-level SAT," *Journal of Systems Architecture*, vol. 51, no. 8, pp. 488–511, 2005.

[8] N. Eén and N. Sörensson, "An extensible SAT solver," in *SAT 2003*, 2003, pp. 502–518.

[9] "The Satisfiability Modulo Theories Competition (SMT-COMP) ," `www.smtcomp.org`, 2008.

[10] H. Kim, H. Jin, K. Ravi, P. Spacek, J. Pierce, B. Kurshan, and F. Somenzi, "Application of formal word-level analysis to constrained random simulation," in *Computer Aided Verification*, 2008, pp. 487–490.

[11] S. Plaza, I. Markov, and V. Bertacco, "Random stimulus generation using entropy and XOR constraints," in *Design, Automation and Test in Europe*, 2008, pp. 664–669.

[12] N. Kitchen and A. Kuehlmann, "Stimulus Generation for Constrainted Random Simulation," in *Int'l Conf. on CAD*, 2007, pp. 258–265.

[13] *SystemC Verification Standard Specification Version 1.0e*, SystemC Verification Working Group, http://www.systemc.org.

[14] J. Rose and S. Swan, "SCV Randomization Version 1.0," 2003.

[15] C. N. Ip and S. Swan, "A Tutorial Introduction on the New SystemC Verification Standard," White Paper, 2003.

[16] *Functional Specification for SystemC 2.0*, Synopsys Inc., CoWare Inc., and Frontier Design Inc., http://www.systemc.org.

[17] *IEEE Standard SystemC Language Reference Manual*, IEEE Std. 1666, 2005.

[18] D. Große, R. Ebendt, and R. Drechsler, "Improvements for constraint solving in the SystemC verification library," in *ACM Great Lakes Symposium on VLSI*, 2007, pp. 493–496.

[19] D. Große, R. Wille, R. Siegmund, and R. Drechsler, "Contradiction analysis for constraint-based random simulation," in *Forum on Specification and Design Languages*, 2008, pp. 130–135.

[20] F. Somenzi, *CUDD: CU Decision Diagram Package Release 2.3.0*. University of Colorado at Boulder, 1998.

[21] M. Davis, G. Logeman, and D. Loveland, "A machine program for theorem proving," *Comm. of the ACM*, vol. 5, pp. 394–397, 1962.

[22] J. Marques-Silva and K. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Trans. on Comp.*, vol. 48, no. 5, pp. 506–521, 1999.

[23] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Design Automation Conf.*, 2001, pp. 530–535.

[24] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. Rossum, S. Schulz, and R. Sebastiani, "The MathSAT 3 System," in *Int. Conf. on Automated Deduction*, 2005, pp. 315–321.

[25] B. Dutertre and L. Moura, "A Fast Linear-Arithmetic Solver for DPLL(T)," in *Computer Aided Verification*, 2006, pp. 81–94.

[26] V. Ganesh and D. L. Dill, "A decision procedure for bit-vectors and arrays." in *Computer Aided Verification*, 2007, pp. 519–531.

[27] 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*, 2009, pp. 174–177.

[28] S. Deng, J. Bian, W. Wu, X. Yang, and Y. Zhao, "EHSAT: An efficient RTL satisfiability solver using an extended DPLL procedure." in *Design Automation Conf.*, 2007, pp. 588–593.

[29] R. Wille, G. Fey, D. Große, S. Eggersglüß, and R. Drechsler, "Sword: A SAT like prover using word level information," in *VLSI of System-on-Chip*, 2007, pp. 88–93.

[30] J. Yuan, K. Shultz, C. Pixley, H. Miller, and A. Aziz, "Modeling design constraints and biasing in simulation using BDDs," in *Int'l Conf. on CAD*, 1999, pp. 584–590.

[31] R. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," *IEEE Trans. on Comp.*, vol. 40, pp. 205–213, 1991.