Improving Coverage of Simulation-based Verification by Dedicated Stimuli Generation

Shuo Yang* Robert Wille*† Rolf Drechsler*†
*Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
†Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
{shuo,rwille,drechsle}@informatik.uni-bremen.de

***Abstract*—Simulation-based verification is still the most frequently used technique when complex designs are to be verified. Stimuli are thereby generated and applied in order to sufficiently trigger and, by this, verify a set of considered scenarios. In general, a scenario can be triggered in various fashions. To ensure a high verification quality, each of these fashions should adequately be covered. However, to the best of our knowledge, this has not appropriately been addressed thus far, i.e. existing stimuli generation is mainly performed without an explicit consideration of the possible fashions in which a scenario might be triggered. To improve this, three approaches are proposed in this work. While examples illustrate their advantages, a case study confirms that, using the proposed approaches, very compact sets of stimuli satisfying this coverage requirement can efficiently be generated.**

I. Introduction

Functional verification, i.e. validating the functional correctness of a *Design Under Verification* (DUV), is continuously playing an important role in the design of highly integrated *Systems on Chips* (SoCs) or *Networks on Chips* (NoCs) in the semiconductor industry. Formal methods (see e.g. [1], [2]) consider a design's functionality exhaustively. However, they are limited by high computational costs. Hence, simulation-based approaches (see e.g. [3]) are still the most frequently used techniques when complex designs are to be verified. Stimuli are thereby generated and applied to the DUV. Then, the responses of the DUV are compared to the expected results.

To efficiently obtain a high verification quality, simulation-based approaches aim for adequately verifying specific, e.g. hard-to-reach, behaviours, while randomly considering the global functionality of the DUV. For this purpose, scenarios which specify these behaviours are usually provided (see e.g. [4]). Then, the goal of verification is to sufficiently trigger these scenarios with as few stimuli as possible.

To achieve this goal, advanced coverage analysis (see e.g. [5], [6], [7]) has been presented in the past. Besides unveiling verification gaps, e.g. insufficiently covered scenarios, they analyse and provide hints on how to trigger those scenarios. Then, in cooperation with modern stimuli generators (see e.g. [8], [9], [10]), this information can be exploited to generate particular stimuli covering these gaps and, by this, guide the subsequent verification in order to improve the coverage.

Although these methods have been well acknowledged, they inherit certain disadvantages, e.g. scenarios are considered as sufficiently covered when they have been triggered a certain amount of times. This is a weak criterion. Since a scenario can be triggered in various fashions, fashions should also be considered to improve the coverage. Nevertheless, to the best of our knowledge, this has not properly been addressed thus far. In particular, dedicated stimuli generations have not yet been introduced and, meanwhile, existing methods have been proven to be incapable to deliver a satisfied result (see e.g. [11]). This is discussed in more detail in Section II.

Motivated by this, three approaches are proposed in this work, which generate dedicated stimuli explicitly addressing all fashions in which a scenario might be triggered:

- Naive Approach: It is first tried to sufficiently cover all cases with one stimulus only. If no such a stimulus exists, the number of stimuli to be generated is increased by 1. This is iterated until a number of stimuli satisfying the coverage requirement can be derived. The problems are encoded as instances of Boolean satisfiability and, then, solved using off-the-shelf solvers.
- Advanced Approach: From a number of stimuli satisfying the coverage requirement, it is first tried to cover all cases with one stimulus *less*. If such a set of stimuli still exists, the number of stimuli to be generated is further decreased by 1. This is repeated, e.g. until the proposed number of stimuli always violate the coverage requirement. This allows for a simplified encoding and, thus, accelerates the solving process.
- Problem Partitioning: A set of stimuli is searched aiming for sufficiently covering a part of cases first. After it has been derived, a next set of stimuli is searched with the same goal but considering the remaining cases only. This continues until all cases have been covered. To ensure the compactness, each set of stimuli is searched using the advanced approach.

A case study illustrates the advantages and disadvantages of these approaches. While the naive approach suffers from high computational costs, a *minimal* set of stimuli satisfying the coverage requirement is obtained. In contrast, the other approaches do not ensure minimality, but still generate a compact set of stimuli in reasonable run-time.

The remainder of this paper is structured as follows: First, the addressed problem is motivated and defined in the next section. Then, the general ideas of the proposed approaches are sketched in Section III followed by a detailed description of their implementations in Section IV. Finally, a summary of the conducted case study is provided in Section V and conclusions are given in Section VI.

II. Problem Formulation

A. Motivation

In simulation-based verification, a set of scenarios is usually provided which specifies certain behaviours, e.g. hard-to-reach behaviours, to be verified. In the paper, the term *scenario* is formally 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 logic AND, logic OR, arithmetic operators, or relational operators. In the following, scenarios and constraints are used interchangeably. The set of scenarios is denoted by $S = \{S_0, \ldots, S_{n-1}\}$.*

Example 1. *Consider a simplified Memory Management Unit (MMU) with primary outputs mem_req (memory request) and mem_rw (memory read or write). Possible scenarios are for instances $S_0 = ((mem\_req = 1) \wedge (mem\_rw = 0))$ (specifying a memory read access) and $S_1 = ((mem\_req = 1) \wedge (mem\_rw = 1))$ (specifying a memory write access).*

Then, stimuli are subsequently generated and applied to the DUV. Scenarios are expected to be triggered by them, so that, the respective behaviours can be verified. To efficiently obtain a high verification quality, scenarios should sufficiently be triggered by a compact set of stimuli. Obviously, this goal depends on (1) an appropriate definition of what is meant by "sufficient" and (2) a proper stimuli generation dedicated to such a kind of sufficiency. While this has been considered in simulation-based verification, existing methods inherit some disadvantages and, in certain cases, are incapable to deliver a satisfying result. This is discussed next.

B. Related Work

In general, scenarios are considered sufficiently covered, e.g. when they have been triggered a certain amount of times. Beyond that, stimuli generators (see e.g. [8], [9], [10]) have been presented aiming to keep the number of stimuli as small as possible. The approach presented in [10] even allows for deriving a minimal set of stimuli.

Nevertheless, a scenario can be triggered in several fashions. Generating a set of stimuli, which triggers a scenario several times but always in the same fashion, does not significantly improve the coverage. Instead, all scenarios should be triggered in various fashions. For this purpose, the work [11] has proposed to explicitly determine all possible fashions in which a scenario can be triggered and, afterwards, use this information for stimuli generation. The term *fashion* is thereby formally defined as *case*.

Definition 2. *A case $c_l^{S_i}$ $(0 \leq l < m)$ of the scenario S_i is a Boolean function over a (minimal) set of primary inputs and flip flops including their assignments which propagate through the DUV and trigger S_i. In the following, the set of cases of a scenario S_i is denoted by $C^{S_i} = \{c_0^{S_i}, \ldots, c_{m-1}^{S_i}\}$.*

Example 2. *Consider again the MMU from Example 1 with additional primary inputs mem_ack (memory acknowledge), re_req (read request), we_req (write request), and the flip flop $state$. According to the specification, a memory read access is performed, i.e. the scenario S_0 is triggered, if*

- *the MMU is in state "idle" and a read request is pending,*
- *the MMU is in state "read" and a read request as well as a memory acknowledge are pending, or*
- *the MMU is in state "write" and a read request as well as a memory acknowledge are pending.*

Hence, there exist three cases for S_0:

1) $c_0^{S_0}$: $(state = idle) \wedge (re\_req = 1)$
2) $c_1^{S_0}$: $(state = read) \wedge (re\_req = 1) \wedge (mem\_ack = 1)$
3) $c_2^{S_0}$: $(state = write) \wedge (re\_req = 1) \wedge (mem\_ack = 1)$

Similarly, the following cases for S_1 can be derived:

1) $c_0^{S_1}$: $(state = idle) \wedge (we\_req = 1) \wedge (re\_req = 0)$
2) $c_1^{S_1}$: $(state = read) \wedge (we\_req = 1) \wedge (re\_req = 0) \wedge (mem\_ack = 1)$
3) $c_2^{S_1}$: $(state = write) \wedge (we\_req = 1) \wedge (re\_req = 0) \wedge (mem\_ack = 1)$

In [11], an automatic approach to determine those cases has been proposed. Explicitly considering cases rather than scenarios enables an improved coverage analysis. Corresponding results summarized in [11] showed that existing approaches for stimuli generation are quite ineffective to deliver a satisfying result in this regard. In fact, sets of stimuli are generated which might trigger certain scenarios a couple of times but not in all possible fashions.

C. Problem Formulation

Motivated by the discussions from above, the following research question is addressed in this work:

> *How can we efficiently determine a compact set of stimuli, which ensures to adequately cover all cases of the considered scenarios?*

In the next section, the general ideas of the proposed solutions are presented.

III. General Ideas

In order to solve the problem stated above, *minimal stimuli generation* as introduced in [10] is exploited. Different approaches based on this method are proposed and described in this section.

In the following, a naive method is proposed first. This builds the foundation for an advanced approach, which is provided next and aims for improving the applicability. Finally, to gain further improvements, a method based on partitioning the problem is proposed. Together with the advanced approach, this method composes the major contribution of this work.

A. Naive Approach

The naive approach is an adaptation of the method introduced in [10] for minimal stimuli generation. In order to describe the idea in a self-contained manner, minimal stimuli generation is described first.

Given a DUV and a set S of scenarios, minimal stimuli generation aims for determining a minimal set of stimuli which triggers all considered scenarios a certain amount of times. For this purpose, a sequence of decision problems is formulated which asks whether a proposed number c of stimuli can achieve this goal. When the decision problem is satisfiable, a set of c stimuli can be derived. Otherwise, it has been proven that no result with c stimuli exists. The general idea is then 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 problem turns out to be unsatisfiable.

To realize this method, an instance of Boolean satisfiability is created for each problem, which is eventually a conjunction of constraints representing the DUV, the scenarios, and the proposed number c. Then, this instance is solved using SAT solvers. This is described in more detail in Section IV.

The method introduced in [10] does not consider cases of scenarios. Hence, an improved coverage as discussed above is not ensured. To address this, the naive approach simply replaces scenarios with cases while maintains the general idea of minimal stimuli generation. In particular, a decision problem now asks whether a proposed number c of stimuli can trigger all considered cases a certain amount of times. Similarly, to solve this problem using SAT solvers, an instance is created which however is composed of constraints for the DUV, *cases*, and the proposed number c. Obviously, this adaptation leads to a set of c stimuli which adequately covers all cases of the considered scenarios.

The resulting method is of exact nature, i.e. the naive approach determines a *minimal* solution. Nevertheless, ensuring minimality often causes tremendous computational costs and, thus, decreases the scalability of the approach. Hence, heuristic methods, which can *efficiently* determine a *compact* set of stimuli satisfying a coverage requirement, are more preferable when complex tasks are to be handled. For this purpose, an advanced approach is proposed next.

B. Advanced Approach

Keeping efficiency in mind, an advanced approach has been developed which benefits from improvements gained by adopting the following revisions on the naive approach:

1) Reducing the Complexity of the Instances: The naive approach takes both the DUV and cases into account. However as shown in Definition 2, a case is composed of a *minimal* set of *primary inputs* and *flip flops*. These signals are independent with each other and with the DUV[1]. Hence, the DUV in fact does not have to be considered as long as cases are directly taken into account. In particular, constraints for the DUV are actually not necessary and, hence, can be omitted to encode instances for decision problems. Obviously, this significantly reduces the complexity of the instances and, by this, considerably decreases the computational costs caused by solving them.

2) Utilizing Advantages of Modern SAT Solvers: The naive approach follows an incremental procedure. Hence, a sequence of *unsatisfying* instances is solved until a satisfying one yields the desired stimuli. This is not very suited for modern SAT solvers such as [12], [13], [14]. In fact, proving the non-existence of a solution requires to consider the entire search space, while this can be avoided as soon as a satisfying solution has been determined. That is, solving satisfying instances is often easier than solving unsatisfiable instances.

In order to exploit this, the procedure is revised to a *decremental* process. Again, a sequence of decision problems is formulated asking whether a set of c stimuli can adequately cover all considered cases. But in contrast to the naive approach, c is not initially set to 1 but a significantly large upper bound (e.g. the total number of cases). Then, c is iteratively decremented until

1) $c = 1$ is reached and, thus, it has been shown that all cases can be triggered by one stimulus only,
2) an instance has been proven unsatisfiable, i.e. it has been shown that all cases can be triggered by $c + 1$ stimuli but not c stimuli (then the number $c + 1$ of stimuli has been proven minimal), or
3) a previously determined time limit has been reached (then minimality is not guaranteed but still a set of c stimuli would be available).

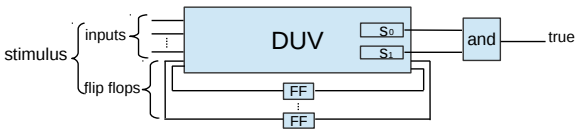
The decremental procedure effectively reduces the number of instances which are generically hard to solve. Furthermore, it iteratively approximates the resulting set of stimuli until it is either stopped or the minimum is determined. By setting the value of the run-time to be spent, designers can define the precision of approximation and, by this, ensure a reasonable compactness vs. efficiency trade-off. Hence, the advanced approach significantly improves the applicability. Nevertheless, it is still limited to the number of cases to be considered since cases are all handled simultaneously at once. Hence, to gain further improvements, an approach considering cases separately is proposed next.

C. Partitioning Approach

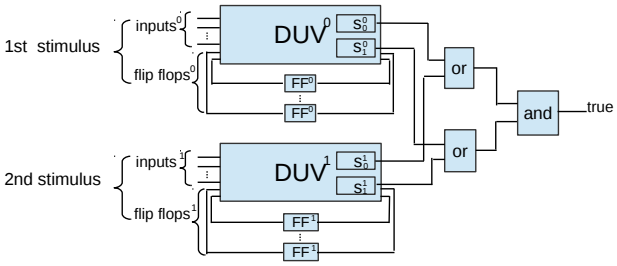
The general idea is to partition a global problem into several smaller ones and, then, solve them separately. In particular, a set of stimuli is now searched targeting on adequately covering at least one of each scenario's cases first. After it has been derived, a next set of stimuli is searched aiming for the same goal but considering the remaining cases of each scenario only. This procedure continues until no inadequately covered cases is left. Then, the final set of stimuli can be obtained by unifying all previously determined ones.

To ensure the compactness, each set of stimuli is searched using the advanced approach. Although this goal can also be achieved using the naive approach, by this, we lose benefits gained by reducing the complexity of instances and utilizing advantages of modern SAT solvers. Overall, this approach

[1]The flip flops, whose values are propagated from primary inputs and/or another flip flops, are not components of a case, since a case consists of a *minimal* set of signals.



(a) First Decision Problem



(b) Second Decision Problem

Fig. 1: Problem Structure of Minimal Stimuli Generation

eases the restriction on the number of cases to be considered and, thus, it can be applied to more complex tasks.

To demonstrate the applicability of the proposed approaches, a case study has been conducted which is summarized in Section V. But before, details on their implementations are provided next.

IV. IMPLEMENTATION

The implementations of the proposed approaches are elaborated in this section.

A. Naive Approach

Since the naive approach is simply adapted from the method of minimal stimuli generation, the implementation of minimal stimuli generation is described first.

As mentioned in III-A, in order to determine a minimal set of stimuli sufficiently covering all scenarios, a sequence of decision problems is formulated and solved using SAT solvers. For this purpose, the work [10] has proposed a general structure for the respective decision problems (see Fig. 1), which can directly be encoded as an instance of Boolean satisfiability. To simplify the description, an example is given next which illustrates step by step, how this structure helps to achieve the goal.

Example 3. *Consider again the MMU and the scenarios S_0 and S_1 from Example 1. First, assuming that each scenario is expected to be triggered at least once.*

The first decision problem asks whether one stimulus can already achieve this goal. The structure of this problem is shown in Fig. 1(a). Besides the constraints for the DUV and the scenarios, an AND gate is used to connect the outputs of the scenarios. Afterwards, the output of this AND gate is fixed to Boolean $true$ such that all scenarios are enforced to be triggered once (a $true$ at the output of a scenario means that the scenario is triggered on the DUV). With this structure, the instance to be solved can directly be encoded as:

$$DUV \wedge S_0 \wedge S_1$$

Apparently, if this instance is satisfiable, a set of stimuli with one stimulus can be derived from the satisfying solution. However, since the S_0 and S_1 can not evaluate to $true$ at the same time (see Example 1), the instance is proven to be unsatisfiable in this case.

Hence, a second decision problem is formulated. It asks whether two stimuli can achieve the goal. The respective structure is shown in Fig. 1(b). Here, a new copy of the DUV and the scenarios[2] (denoted by DUV^1, S_0^1 and S_1^1, respectively) as well as extra OR gates are added. As can be seen, different copies of a scenario are first connected to an OR gate. Then, all OR gates are connected to the AND gate[3]. Similarly, the output of the AND gate is fixed to $true$. By this, at least one copy of each scenario must evaluate to $true$ such that all scenarios must be triggered at least once. This structure results in the following instance:

$$DUV^0 \wedge DUV^1 \wedge (S_0^0 \vee S_0^1) \wedge (S_1^0 \vee S_1^1)$$

Obviously, solving this instance yields a satisfying solution, e.g. S_0^0=1 (S_1^0=0) and S_1^1=1 (S_0^1=0). Hence, a result with two stimuli (each is from one copy of the DUV) is derived. Since the proposed number of stimuli is iteratively incremented by one, minimality is ensured.

Thus far, the resulting set of stimuli triggers each scenario once. In order to determine further (new) stimuli and, by this, trigger scenarios more times, as proposed in [10] the satisfying instance is repeatedly solved while always blocking the already determined solutions until the expected number of coverings for each scenario has been reached.

Overall, an algorithm has been presented in [10]. A simplified version of it is shown in Algorithm 1. Given a DUV and a number n of scenarios as inputs, the algorithm first starts with initializing the proposed number of stimuli (denoted by c) to 1 and the set of all stimuli to be determined (denoted by S_{stim}) to $\emptyset$. Then, an instance Φ with

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

is created and solved by SAT solvers (Line 2-3). If this instance is satisfiable, a set of stimuli is extracted and stored in S_{stim} (Line 4-5). Then, an analysis is performed to check whether the respective scenarios S_i have been triggered the desired amount of times (denoted by t_{S_i}) – see Line 6. If this is the case, the algorithm terminates. Otherwise, S_{stim} is blocked

[2]Copies of the DUV are functionally identical, but independent with each other because of renamed signals in all copies of the DUV. This applies to all copies of scenarios too. Hence, when a copy of scenario is triggered on the respective copy of the DUV, the scenario is triggered on the DUV.

[3]These connection rules apply to further copies of the DUV and scenarios too, which are added in the structure in case that, the current number of copies does not allow for creating a satisfying instance.

Algorithm 1: Minimal Stimuli Generation

Input: DUV, Scenarios S

1 $c = 1$; $S_{stim} = \emptyset$;

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

3 res = solve (Φ) ;

4 **if** $res = true$ **then**

5 $S_{stim} = S_{stim} \cup$ extract () ;

6 **if** analyse (S_{stim}) $\geq t_{S_i}$ foreach S_i **then**

7 return ;

8 **else**

9 block (S_{stim}) ;

10 go to 2 ;

11 **else**

12 ++c ;

13 go to 2 ;

and the process is repeated (Line 9-10). Here, the variable c is not incremented. Further stimuli are still generated based on the current instance. This iterates until no more stimuli can be generated with the current value of c (Line 11). Then, c is incremented and, by this, an additional copy of DUV and scenarios are added in (Line 12-13).

To adapt this implementation to support the problem considered in this work, scenarios are replaced by cases as mentioned earlier. This is illustrated in the following example.

Example 4. *Reconsider the procedure in Example 3 and cases from Example 2. The first decision problem now asks whether one stimulus can cover all cases. The respective instance is:*

$$DUV \wedge c_0^{S_0} \wedge c_1^{S_0} \wedge c_2^{S_0} \wedge c_0^{S_1} \wedge c_1^{S_1} \wedge c_2^{S_1}$$

Since this instance is not satisfiable, i.e. not all cases can evaluate to $true$ simultaneously (see Example 2), the second decision problem with $c = 2$ is formulated. The respective instance is:

$$DUV^0 \wedge DUV^1 \wedge (c_0^{S_0{}^0} \vee c_0^{S_0{}^1}) \wedge (c_1^{S_0{}^0} \vee c_1^{S_0{}^1}) \wedge$$
$$(c_2^{S_0{}^0} \vee c_2^{S_0{}^1}) \wedge (c_0^{S_1{}^0} \vee c_0^{S_1{}^1}) \wedge (c_1^{S_1{}^0} \vee c_1^{S_1{}^1}) \wedge (c_2^{S_1{}^0} \vee c_2^{S_1{}^1})$$

Similarly, no satisfying solution exists for the above instance. Hence, c is further incremented. The first satisfying instance and, by this, the first set of stimuli is obtained when $c = 6$, i.e. each case needs one stimulus. Then, by repeatedly solving this instance, more stimuli can be generated.

Overall, the instance to be solved now can be generalized as (assuming that each of n scenarios has m cases):

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

Then, by simply replacing the instance (Line 2) in Algorithm 1 with this new instance while keeping the rest of the algorithm (the inputs now are the DUV and all considered cases), the naive approach is realized.

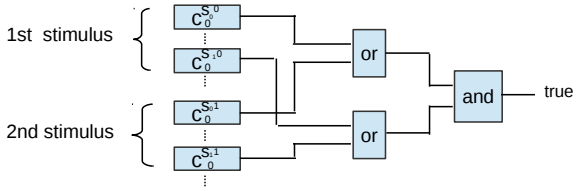


Fig. 2: Problem Structure of Advanced Approach

B. Advanced Approach

While the naive approach is realized straightforwardly, the advanced approach requires major changes.

First of all, the DUV is not considered in the approach. This leads to a simplified problem structure as depicted in Fig. 2, i.e. the DUV in Fig. 1 is omitted while the other elements and connections remain (scenarios are replaced by cases). Hence, the respective instance is encoded as:

$$\bigwedge_{i=0}^{n-1} \bigwedge_{l=0}^{m-1} \bigvee_{d=0}^{c-1} c_l^{S_i{}^d} \tag{3}$$

Then, considering this new instance a decremental solving procedure is applied. This is illustrated by means of the following example.

Example 5. *Reconsider the cases from Example 2. With the decremental procedure, the first decision problem asks whether 6 stimuli can cover all cases at least once (assuming that each case needs one stimulus). The number 6 is chosen since 6 cases are considered in total and, hence, 6 constitutes an obvious upper bound . The respective instance is:*

$$(c_0^{S_0{}^0} \vee c_0^{S_0{}^1} \vee \cdots \vee c_0^{S_0{}^5}) \wedge (c_1^{S_0{}^0} \vee c_1^{S_0{}^1} \vee \cdots \vee c_1^{S_0{}^5}) \wedge$$
$$(c_2^{S_0{}^0} \vee c_2^{S_0{}^1} \vee \cdots \vee c_2^{S_0{}^5}) \wedge (c_0^{S_1{}^0} \vee c_0^{S_1{}^1} \vee \cdots \vee c_0^{S_1{}^5}) \wedge$$
$$(c_1^{S_1{}^0} \vee c_1^{S_1{}^1} \vee \cdots \vee c_1^{S_1{}^5}) \wedge (c_2^{S_1{}^0} \vee c_2^{S_1{}^1} \vee \cdots \vee c_2^{S_1{}^5})$$

Although this instance includes 6 copies of each case, modern SAT solvers can efficiently find a solution since it is satisfiable.

Then, the proposed number is decremented by one. That is, the second instance asks whether 5 stimuli can still achieve the goal. This directly leads to an unsatisfying result or a solving time-out. In both cases, the process terminates and a set of stimuli is derived from the last satisfying solution.

In contrast to Example 4 where the incremental procedure is used, four unsatisfying instances have been avoided and, by this, obviously lots of solving time has been saved.

Similarly, to obtain further stimuli and, by this, trigger cases more times, a satisfying instance is repeatedly solved until the expected coverings for each case has been reached.

Overall, we have devised Algorithm 2. Given a set C of cases, the algorithm starts with initializing c and S_{stim} to the total amount of considered cases and $\emptyset$ (Line 1). In addition, a flag variable f is set to Boolean $false$ indicating that a set of stimuli should *not* be extracted from a satisfying solution yet

Algorithm 2: Advanced Approach

Input: a set C of cases derived from scenarios S
(DUV is omitted in this approach)

1 $c = C$.size(); $S_{stim} = \emptyset$; $f = false$;
2 $\Phi = \bigwedge_{i=0}^{n-1} \bigwedge_{l=0}^{m-1} \bigvee_{d=0}^{c-1} c_l^{S_i\,d}$;
3 res = solve (Φ) ;
4 **if** $timeout$ || $(res = false)$ **then**
5 $f = true$;
6 c++ ;
7 **else**
8 **if** $f = false$ **then**
9 **if** $c = 1$ **then**
10 $f = true$;
11 **else**
12 c-- ;
13 **else**
14 $S_{stim} = S_{stim}$ $\cup$ extract () ;
15 **if** analyse $(S_{stim}) \geq t_{c^{S_i}}$ foreach c^{S_i} **then**
16 return S_{stim} ;
17 **else**
18 block (S_{stim}) ;
19 go to 2 ;

(the function of f will gradually be described in the following). Then, the instance Φ is created and solved for the first time (Line 2-3). Since this first instance assumes c to be the number of all considered cases, it is satisfiable (Line 7). Nevertheless, a set of stimuli should not be derived from it yet, because further satisfying instances composed of less copies of each case may still exist and they should be obtained (or proven to be unavailable) in order to ensure the resulting set of stimuli as compact as possible. For this purpose, f keeps being $false$ and c is decremented by one (Line 12). Then, the process continues at Line 2. This iterates until the satisfying instance made of one copy of each case has been reached (Line 9), the expected time for solving an instance has run out (denoted by $timeout$) or, an unsatisfying result has been obtained (Line 4). That is, f is switched to $true$ in order to allow for extracting a set of stimuli either from the current satisfying solution (Line 10, 19) or from the last one (Line 5-6, 19). Similarly, the overall result is analysed if it has adequately covered all cases (Line 15). Based on this analysis, either the overall process terminates (Line 16) or S_{stim} is blocked and more stimuli are derived subsequently (Line 18-19).

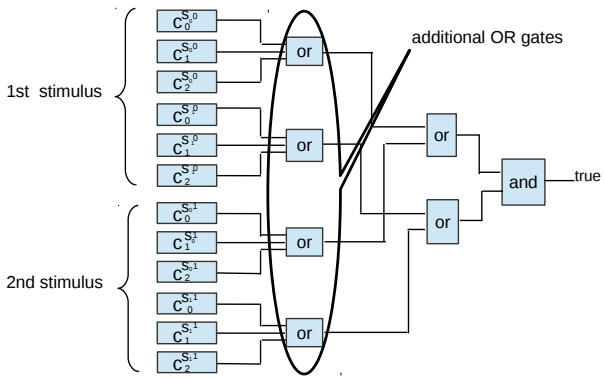


Fig. 3: Problem Structure to Partition Problem

C. Partitioning Approach

In order to realize the idea sketched in Section III-C, different implementations are possible. A trivial solution may just arbitrarily partition the cases into groups and, afterwards, solve the resulting instances separately. However, a more elaborated scheme is proposed in this work.

In fact, the respective cases to be considered are grouped by their respective scenarios. To this end, a similar approach as introduced in Section IV-B is applied. The decremental solving procedure remains, but the precise SAT encoding is revised to

$$\bigwedge_{i=0}^{n-1} \bigvee_{d=0}^{c-1} \bigvee_{l=0}^{m-1} c_l^{S_i\,d}. \tag{4}$$

That is, instead of the formulation as shown in Fig. 2 an extended formulation as shown in Fig. 3 is applied. Here, additional OR gates are introduced which respectively connect all cases of a scenario together. By this, a satisfying solution is derived when at least one case of each scenario evaluates to $true$. Hence, only a partition of all cases must be satisfied in order to determine a stimuli.

Example 6. *Consider again the cases from Example 2. Since these cases are derived from two scenarios, the first decision problem asks whether two stimuli can cover at least one case of each scenario. With respect to the new structure, this problem is encoded as ($c = 2$):*

$$((c_0^{S_0\,0} \vee c_1^{S_0\,0} \vee c_2^{S_0\,0}) \vee (c_0^{S_0\,1} \vee c_1^{S_0\,1} \vee c_2^{S_0\,1})) \;\wedge$$

$$((c_0^{S_1\,0} \vee c_1^{S_1\,0} \vee c_2^{S_1\,0}) \vee (c_0^{S_1\,1} \vee c_1^{S_1\,1} \vee c_2^{S_1\,1}))$$

Compared to the first instance as shown in Example 5, the complexity of this instance is obviously decreased. Hence, noticeable solving time can be reduced (this is confirmed by a case study presented in the next section).

Similarly, solving this instance yields a satisfying solution, e.g. $c_0^{S_0\,0} = 1$ and $c_0^{S_1\,1} = 1$. Hence, c is decremented by one. This directly results to the following unsatisfiable instance:

$$(c_0^{S_0} \vee c_1^{S_0} \vee c_2^{S_0}) \wedge (c_0^{S_1} \vee c_1^{S_1} \vee c_2^{S_1})$$

Algorithm 3: Problem Partitioning

Input: a set C of cases derived from scenarios S
(DUV is omitted in this approach)

1 $C_{tmp} = C$; $C_{log} = \emptyset$; $Tmp = \emptyset$; $S_{stim} = \emptyset$;
2 **while** $C_{tmp} \neq \emptyset$ **do**
3 Tmp, C_{log} = revised_advanced_approach (C_{tmp}) ;
4 $S_{stim} = S_{stim} \cup Tmp$;
5 $C_{tmp} = C_{tmp} \setminus C_{log}$;
6 return S_{stim} ;

Thus, the first set of stimuli is derived from the last solution which triggers $c_0^{S_0}$ and $c_0^{S_1}$ Hence, four cases are left to be considered in the next iteration, i.e. the following instance is considered next:

$$((c_1^{S_0{}^0} \vee c_2^{S_0{}^0}) \vee (c_1^{S_0{}^1} \vee c_2^{S_0{}^1})) \wedge ((c_1^{S_1{}^0} \vee c_2^{S_1{}^0}) \vee (c_1^{S_1{}^1} \vee c_2^{S_1{}^1}))$$

In this manner, stimuli for all remaining cases will eventually been generated.

Overall, we have devised Algorithm 3. Similarly, given a set C of cases, the process starts with initializing variables. Besides of the known S_{stim}, three set C_{tmp}, Tmp and C_{log} are introduced (Line 1), which, in each iteration, record the cases left to be considered, the resulting set of stimuli, and the cases being newly satisfied. These sets are initially set to C, $\emptyset$, and $\emptyset$, respectively. Then, the first set of stimuli is determined (Line 3) as described above. More precisely, 1) Line 2 in Algorithm 2 is replaced by Instance 4 and, 2) Line 15 is now satisfied when only (at least) one case of each scenario has adequately been covered and, 3) besides of the resulting stimuli, Line 16 returns also information on which cases have been covered. After the result has been derived, it is stored in S_{stim} (Line 4). Meanwhile, the newly satisfied cases are excluded from C_{tmp} (Line 5) and, thus, they are not passed to the next iteration. Then, the process is repeated until C_{tmp} is eventually empty (Line 2). Then, the process terminates and returns the final set of stimuli (Line 6).

V. Case Study

The proposed approaches have been implemented in C++. As SAT solver, we utilized MiniSAT [14]. To demonstrate the advantages and disadvantages of the approaches, a case study has been conducted on a *Memory Management Unit* (MMU), i.e. an interface between a CPU and an external memory which manages the respective data transactions.

For this MMU, 93 cases have been considered which were derived from 18 scenarios using the approach introduced in [11]. The objective was to determine a compact set of stimuli covering each of these cases at least once. This case study has been conducted on a 64-bit AMD Athlon Dual Core machine with 4 GB of memory running Linux.

The results are summarized in Table I and Table II. Table I reports the results obtained by the naive and the advanced approach. More precisely, the first column gives the predefined number c of stimuli to be considered in order to trigger all cases. Columns $|PI|$ and $|V|$ specify the number of free variables and the total number of variables of the instance, respectively. Finally, column SAT denotes whether the respective instance has been classified as satisfiable ($\surd$) or unsatisfiable ($\times$), while column $Time$ provides the run-time (in CPU seconds) which was required to derive this result.

Table II summarizes the results of the partitioning approach. For the considered MMU, a total of seven iterations were necessary. For each iteration (denoted in column *Iteration*), columns # and $Time$ provide the number of generated stimuli and the required run-time, respectively. Column $|C_a^{S_i}|$ shows the number of cases which got covered in this iteration.

The results clearly confirm the discussions from above. Although the naive approach would generate a *minimal* result, it suffers from the high computational cost. For the considered MMU, no satisfying solution was obtained within a time limit of 5000 CPU seconds. It could only be proven that the considered cases cannot completely be covered by 11 or less stimuli.

In contrast, the applicability is significantly improved using the advanced approach. Although the satisfying instances may be orders of magnitudes larger (e.g. the first instance encodes 93 copies of cases), modern SAT solvers are capable to derive a solution in negligible run-time. Utilizing this advantage, the decremental procedure can efficiently approximate the proposed number c towards the minimal value. Although the minimal value is eventually not reached, a fair approximation composed of 27 stimuli was determined in just 13.328 CPU seconds.

Finally, the partitioning approach is confirmed to be the most efficient method with respect to the run-time. Since this approach partitions all cases with respect to its scenarios and the total of 93 cases to be considered have been derived from 18 scenarios, the upper bound for c is 18. Additionally exploiting the advanced approach, this significantly reduces the complexity and, hence, allows for an efficient determination of the stimuli. In contrast, several iterations have to be conducted which eventually increase the number of generated stimuli: Following this scheme a total of 30 stimuli are generated until all cases are covered. Although this is more than the 27 stimuli determined by the advanced approach, this result was generated in just a bit more than a CPU second and did not even require the application of a run-time limit.

Overall, compact sets of 27 and 30 stimuli covering *all* cases are efficiently determined using the proposed approaches. Compared to previous work, this is a significant improvement. In fact, the evaluations in [11] showed that, using e.g. the stimuli generator proposed in [10], even a set of 120 generated stimuli did not cover all these cases. This clearly shows how an explicit consideration of cases advances the coverage in simulation-based verification.

TABLE I: Stimuli Generation Considering Cases of Scenarios of the MMU
93 cases derived from 18 scenarios are under the consideration.

Naive Approach					Advanced Approach				
c	$\|PI\|$	$\|V\|$	SAT	Time (s)	c	$\|PI\|$	$\|V\|$	SAT	Time (s)
1	14	2359	×	0.001	93	744	211203	√	0.589
2	28	4625	×	0.015	92	736	208933	√	0.307
3	42	6891	×	0.016	91	728	206663	√	1.118
4	56	9157	×	0.020	...				
5	70	11423	×	0.037	66	528	149913	√	0.184
6	84	13689	×	0.136	65	520	147643	√	0.167
7	98	15955	×	0.966	64	512	145373	√	0.208
8	112	18221	×	3.124	...				
9	126	20487	×	14.82	28	224	63653	√	0.639
10	140	22753	×	347.50	27	216	61383	√	0.167
11	154	25019	×	467.03	26	208	59113	–	$t.o.$
12	168	27285	–	$t.o.$					
Total SAT solving time until $t.o.$: 833.66					Total SAT solving time until $t.o.$: 13.328				

c: Pre-defined number of stimuli to be considered $|PI|$: Number of free variables
$|V|$: Number of total variables SAT: Result of the solving engine Time (s): Run-time in CPU seconds
$t.o.$: Time out

TABLE II: Problem Partitioning

Iteration	#	Time (s)	$\|C_a^{S_i}\|$
1.	6	0.322	28
2.	6	0.283	24
3.	6	0.187	20
4.	5	0.149	14
5.	5	0.082	5
6.	1	0.028	1
7.	1	0.014	1
Total number of stimuli: 30 Total SAT solving time: 1.065			

#: Number of determined stimuli
Time (s): Run-time in CPU seconds
$|C_a^{S_i}|$: Number of covered cases in an iteration

VI. CONCLUSION

In this work, we considered the generation of dedicated stimuli for simulation-based verification which cover scenarios to be triggered in all possible fashions. For this purpose, three approaches have been proposed including a naive approach based on minimal stimuli generation, an advanced approach with less complexity, and an approach employing a partitioning scheme. By means of examples, the application of these approaches has been demonstrated. A case study confirmed their advantages and disadvantages. Overall, it was possible to determine very compact sets of stimuli satisfying all cases of the considered scenarios. With the approaches presented in this work, a contribution towards increasing the coverage has been made.

ACKNOWLEDGMENTS

This work was supported by the German Federal Ministry of Education and Research (BMBF) within the project SPECifIC under grant no. 01IW13001 as well as the German Research Foundation (DFG) within a Reinhart Koselleck project under grant no. DR 287/23-1.

REFERENCES

[1] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Design Automation Conf.*, 1999, pp. 317–320.

[2] D. Große, U. Kühne, and R. Drechsler, "Analyzing functional coverage in bounded model checking," *IEEE Trans. on CAD*, vol. 27, no. 7, pp. 1305–1314, 2008.

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

[4] S. Tasiran and K. Keutzer, "Coverage metrics for functional validation of hardware designs," *IEEE Design & Test of Comp.*, vol. 18, no. 4, pp. 36 –45, 2001.

[5] 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.*, 1998, pp. 158–165.

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

[7] H. Azatchi, L. Fournier, E. Marcus, S. Ur, A. Ziv, and K. Zohar, "Advanced analysis techniques for cross-product coverage," *IEEE Trans. on Comp.*, vol. 55, no. 11, pp. 1367–1379, 2006.

[8] P. Mishra and N. Dutt, "Functional coverage driven test generation for validation of pipelined processors," in *Design, Automation and Test in Europe, 2005. Proceedings*, march 2005, pp. 678 – 683 Vol. 2.

[9] I. Wagner, V. Bertacco, and T. Austin, "Microprocessor verification via feedback-adjusted markov models," *IEEE Trans. on CAD*, vol. 6, no. 26, pp. 1126–1138, 2007.

[10] S. Yang, R. Wille, D. Große, and R. Drechsler, "Minimal stimuli generation in simulation-based verification," in *EUROMICRO Symp. on Digital System Design*, 2013, pp. 439–444.

[11] S. Yang, R. Wille, and R. Drechsler, "Determining cases of scenarios to improve coverage in simulation-based verification," in *Symp. on Integrated Circuits and System Design*, 2014.

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

[13] 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.

[14] N. Eén and N. Sörensson, "An extensible SAT solver," in *Conference on Theory and Applications of Satisfiability Testing*, ser. LNCS, vol. 2919, 2004, pp. 502–518.