SAT-Based Semiformal Verification of Hardware

Sabih Agbaria, Dan Carmi, Orly Cohen, Dmitry Korchemny, Michael Lifshits and Alexander Nadel
Intel Corporation, P.O. Box 1659, Haifa 31015 Israel
Email: (sabih.agbaria,dan.carmi,orly.cohen,dmitry.korchemny,michael.lifshits,alexander.nadel)@intel.com

***Abstract*—Semiformal, or hybrid, verification techniques are extensively used in pre-silicon hardware verification. Most approaches combine simulation and formal verification (FV) algorithms to achieve better design coverage than conventional simulation and scale better than FV. In this paper we introduce a purely SAT-based semiformal verification (SFV) method that is based on new algorithms for generating multiple heterogeneous models for a propositional formula. An additional novelty of our paper is the extension of the SFV algorithm to liveness properties. The experimental data presented in this paper clearly shows that the proposed method can effectively find bugs in complex industrial designs that neither simulation nor FV reveal.**

I. Introduction

Traditionally, Register Transfer Logic (RTL) level design validation is carried out by applying simulation techniques throughout the design and formal verification in certain high risk areas. In simulation, design behavior is checked with a large number of mostly random tests which cover just a small fraction of the design space. FV resolves the coverage issue by exhaustively checking all possible scenarios. It usually requires building a restricted environment and reduced model, as it cannot be directly applied on typical industrial-size designs. One of today's most efficient FV methods, SAT-based bounded model checking (BMC) [1], verifies the lack of bugs in scenarios of bounded length. The maximal reachable BMC bound is not sufficient in many cases to address structures with long latency, such as deep queues or counters.

Semiformal verification approaches developed throughout the last decade trade the completeness of FV for effectiveness. They aim to detect bugs in larger designs rather than to prove their correctness. Being incomplete, these approaches are sound — all reported violations of the properties are true bugs. SFV approaches that simultaneously apply multiple verification techniques in a complementary fashion are referred as *hybrid* approaches. Bhadra et al. [2] provide a comprehensive survey of recent advances in hybrid approaches to functional verification. A major challenge for hybrid tools is their practical applicability to a wide range of industrial designs and the soundness of the integration of the individual technique. As opposed to hybrid approaches, our SFV method is based on a single FV algorithm – SAT-based BMC.

Previous SFV approaches using a single FV algorithm suggested heuristics to search in a fraction of the original state space. This allowed reducing the binary decision diagrams (BDD) [3] to a manageable size in semiformal symbolic reachability analysis [4]–[6]. BDD-based algorithms, whose capacity is limited to hundreds of variables, are unsuitable for verifying properties in today's industrial designs, which often comprise tens of thousands of state elements. Cabodi et al [7] restrict the BMC SAT engine during the search based on dynamically computed simplified BDD-based image and preimage computations. The work in [8] suggests a rarity-based metric to identify states of a particular depth, searching from which leads to better coverage.

We use a different approach that utilizes user guidance to restrict the search within the state space - an idea extending the "lighthouses" used in the SIVA tool [9]. The user guides the search by providing a series of waypoints – describing design behavior throughout the desired scenario. The idea is similar to [10], but is used in the context of property verification rather than post-silicon debugging. Some works have suggested ways to automate the guiding algorithm, as they consider user guidance as a major drawback. For example, see probabilistic state ranking in [11] and lighthouse generation automation in [12], [13]. However, our experience shows that because verification engineers are well versed in the design, they can easily specify the required waypoints. Moreover, they usually prefer to encounter events they are familiar with when analyzing the resulting counterexamples.

There are other hybrid techniques that augment simulation with formal searches, as is done in KETCHUM [14], SIVA [9] and other systems [15], [16]. The biggest challenge for these tools is the synchronization of the simulation and FV environments. Random simulation needs to take into account the FV environment, which is usually modeled with complex sequential assumptions. Although this problem was partially addressed in [17], eventuality assumptions, assumptions involving internal or output signals, and assumptions requiring a *lookahead* (e.g. $G(a \rightarrow past(b))$) are very difficult or impossible to account for, thus resulting in false negative results. Another approach applies multiple shallow FV searches starting from selected cycles in simulation, a technique known as dynamic FV. Dynamic FV approaches suffer from an inherent drawback – they require tight coordination between the FV and simulation environments, which is extremely difficult to achieve, since in most cases FV is applied at a lower level of hierarchy than simulation. Moreover, the FV environment is usually restricted, allowing only a subset of functionalities, a fact which makes many simulation tests unusable.

Our SFV technique uses user guidance to compose several applications of purely SAT-based model checking, and explores the system state space in parts. It can be applied to all LTL properties, including liveness properties. We address the known problem that some waypoint states may not be extendable to the next waypoint. We introduce two new highly configurable SAT-based algorithms for model sampling to generate different traces towards waypoints – necessary for achieving sufficient coverage and detecting corner-case bugs. This differs from previously suggested approaches, e.g. periodically tunneling or backtracking between shallow and deeper waypoints [13]. Our experimental results show the superior bug-finding ability of our approach, which detected critical bugs in industrial-scale designs that were "clean" from FV and simulation perspectives.

The rest of the paper is organized as follows. Section II describes the proposed BMC-based SFV algorithm. Section III introduces SAT-based algorithms for model sampling. Section IV is dedicated to semiformal verification of liveness properties. Our experiments are described in Sections V and VI, the first reviewing the test cases and the second summarizing the results. Conclusions and future work directions follow in Section VII.

We use a standard LTL notation for temporal properties: X for *next*, U for *until*, G for *always*, and F for *eventually* (see [18]). Instead of repeating X n times we use a shortcut notation X^n.

II. SAT-Based Semiformal Verification

A. Basic Algorithm

The verification time in BMC grows exponentially with the bound, and as a result it cannot explore scenarios that require many clock cycles to execute. The proposed semiformal verification algorithm applies multiple shallow BMC runs, trading the exhaustiveness of a search for speed. The user provides an ordered set of *waypoints* which direct the search engine towards the desired deep design state. The algorithm searches for a path from one waypoint to the next starting from the initial state, the BMC engine being restarted at each waypoint. Being familiar with the design behavior, users naturally direct the search towards the desired area by encoding the waypoints with cover points. For example, consider a queue that requires 200 clock cycles to be filled. To verify the design in a risky *"full queue"* state, possible waypoints could be *"1/4 full queue"*, *"1/2 full queue"*, *"3/4 full queue"*, each waypoint being easily reached and the overall verification time being but a fraction of the original BMC verification time.

The high-level SFV algorithm below is based on the fact that the properties may be represented with finite automata [18]. Another possibility for handling properties is to generate the satisfiability formula directly by the syntactic structure of the temporal assertion [19]. However, this algorithm is much less efficient than semantic translation based on automata [18], as shown in [20]; therefore we do not consider syntactic translation here.

Given a series of cover points $\xi_1, \xi_2, \ldots, \xi_n$ and the property φ, the algorithm performs the following steps:

1) Calculate the set of relevant assumptions for $\xi_1, \xi_2, \ldots, \xi_n$ and run BMC targeting ξ_1 from the set of initial states W_0.
2) If a witness has been found, the property automata are simulated along this witness. BMC and simulation are repeated each time using the end point of the last simulation as the new initial state, targeting consequent cover points $\xi_2, \ldots, \xi_n$. If a witness is not found for some ξ_i, an indeterminate result is reported.
3) Run BMC to determine whether φ holds. If there is a failure, append the counterexample to the concatenation of witnesses $\xi_1, \ldots, \xi_n$. If a timeout or required BMC bound is reached, report a lack of failure.

$\mathbf{init}\ q_1, \ldots, q_4 \leftarrow 0$
$\mathbf{next}(q_1) \leftarrow a;\ \mathbf{next}(q_2) \leftarrow q_1;\ \ldots; \mathbf{next}(q_4) \leftarrow q_3$
$fail \leftarrow \neg b \wedge q_4$

Fig. 1: RTL for assumption $G(a \rightarrow X^4 b)$

B. Calculation of New Initial States for Safety Properties

Since a safety property automaton can be synthesized into RTL [21], it may be simulated on the waypoint witness using a conventional RTL simulator. As an example, consider an assumption $G(a \rightarrow X^4 b)$. Its automaton may be synthesized as shown in Fig. 1.

If $a = 1$ in the witness appears in the next to last step, the initial state of the next BMC run should have $q_2 = 1$. Simulating the property automaton is important: blindly reusing the initial property condition $\mathbf{init}\ q_1, \ldots, q_4 \leftarrow 0$ would have led to the discontinuity of the adjacent BMC runs, and potentially to false negatives and bogus witnesses and counterexamples.

III. Using Multiple SAT Models to Enhance Coverage

A. Motivation and Related Work

The experiments conducted, described in Section VI, show that the proposed basic algorithm will likely miss corner-case bugs. The reason for this is that a randomly chosen path, constructed from a series of witnesses each of which satisfies the corresponding intermediate waypoint, does not exhibit sufficient coverage of the design space. Greater coverage may be achieved by advancing towards the desired deep state along multiple paths in parallel. For each intermediate waypoint, a heterogeneous set of witnesses is generated instead of a single witness, and for each such witness a separate verification process towards the next waypoint is launched. Consider Fig. 2 which illustrates a scenario where using two witnesses for the waypoints resulted in bug detection, whereas the chances of detecting the bug would have been much smaller otherwise.

A number of approaches to generating random witnesses (or solutions, or models) exist in literature. BDD-based, local-search-based, and arithmetic-based approaches such as [22], [23], and [24], respectively, are not applicable for our domain, since our test-cases are too complex for BDD-based and local-search-based algorithms, and they contain more bit-vector operations than arithmetical operations.

Modern efficient SAT solvers are able to solve complex formulas that arise in FV. SAT-based methods can also be used to sample the solutions of a given formula. One such method,

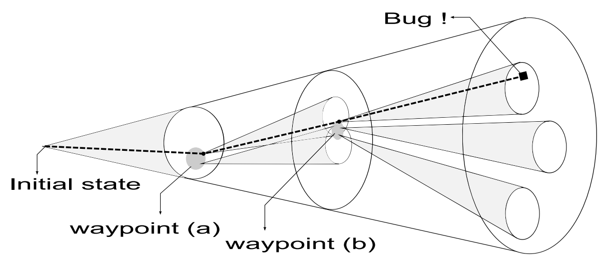

Fig. 2: Multiple witnesses

called XORSample, was proposed in [25]. XORSample invokes the SAT solver at least k times to generate k models. For each invocation, the initial formula is augmented with random XOR constraints. A sampling is not rejected only if the augmented formula has one and only one model. This requirement was relaxed in [26], whose version of XORSample does not reject samplings. Another SAT-based method, called DPLL-based sampling, was mentioned in [24] (we did not find any reference to a work introducing it). DPLL-based sampling invokes a SAT solver k times to generate k models on the same input formula. Model diversification is achieved by making the first boolean value assignment to a variable random for each invocation of the SAT solver.

Literature on the AllSAT problem (that is, the problem of finding all the models for a formula) is also relevant for our purposes. Most AllSAT engines are built on top of a SAT solver. When a model is found, a typical AllSAT solver [27], [28] adds a blocking clause which prevents the solver from rediscovering the same model in a subsequent search and restarts the search. Unlike DPLL-based sampling, AllSAT invokes a SAT solver only once.

B. SAT-Based Algorithms for Generating Multiple Witnesses

In this section we describe two new algorithms for generating heterogeneous models (witnesses) to a given formula: Rand-k-SAT and Guide-k-SAT. Both our algorithms surpass existing approaches in terms of both diversification quality (formally defined below) and performance. We also present two modifications to Rand-k-SAT and Guide-k-SAT, called AllSAT-sampling and BCP-aware Guide-k-SAT, which allow the user to trade diversification quality for performance.

Given a propositional formula F in conjunctive normal form (CNF) over variables $V = \{v_1, \ldots, v_n\}$, a SAT solver either finds a complete satisfying assignment (model) for F or proves that no model for F exists. We define the *distance* $D(\mu_1, \mu_2)$ between two partial assignments μ_1 and μ_2 to be the number of variables that are assigned in both μ_1 and μ_2 and have different values in μ_1 and μ_2. Note that our definition yields that the distance between two models is the Hamming distance. We define the *diversification quality* of k models $\mu_1 \ldots \mu_k$ $Q(\mu_1 \ldots \mu_k)$ to be the average distance between each pair of models, normalized by the number of variables:
$Q(\mu_1 \ldots \mu_k) = (\sum_{i=1}^{k} \sum_{j=i+1}^{k} D(\mu_i, \mu_j))/(n(k^2 - k)/2)$.

For example, consider a formula $F = (a \vee b \vee c) \wedge (\neg a \vee b)$ and three models $\mu_1 = \{a = 1,\ b = 1,\ c = 0\}$, $\mu_2 = \{a = 1,\ b = 1,\ c = 1\}$, and $\mu_3 = \{a = 0,\ b = 0,\ c = 1\}$. Then, $D(\mu_1, \mu_2) = 1$, $D(\mu_1, \mu_3) = 3$, $D(\mu_2, \mu_3) = 2$, and $Q(\mu_1, \mu_2, \mu_3) = (1{+}2{+}3)/(3{\times}((3^2{-}3)/2)) = 2/3$. Note that since the diversification quality is normalized by the number of variables, it must lie between 0 and 1.

Given a propositional formula F in CNF and an integer number $k > 0$, we are interested in finding k models for F with the optimization goal of increasing the diversification quality of the models. We do not intend to guarantee a certain quality in a theoretical sense, but rather to combine solid performance with a good model quality for the practical needs of efficient semiformal verification.

Both our approaches, Rand-k-SAT and Guide-k-SAT, invoke the SAT solver only once, like AllSAT solvers do. However, we do not add blocking clauses when models are discovered. Instead, the solver restarts the search after a model is discovered. Diversification is achieved solely by changing the phase selection heuristic for variables.

The decision stage of a modern SAT solver chooses a variable and its phase at each decision point during the search. The variable decision heuristic selects a variable. The phase selection heuristic selects a boolean value for the selected variable. Most modern SAT solvers use RSAT solver's phase selection heuristic [29], which tries to refocus the search on subspaces that the solver has knowledge about. This heuristic keeps a saved-phase array, indexed by variables. The array contains boolean values and is initialized with 0's. The solver stores the last assignment given to a variable in the saved-phase array. The phase selection heuristic for variable v always chooses the value of v from the saved-phase array.

Both Rand-k-SAT and Guide-k-SAT override the traditional phase selection heuristics. However, they differ from one another conceptually in their phase selection strategies. Rand-k-SAT selects the phase randomly on all occasions. Guide-k-SAT selects the polarity in a non-random manner: explicitly guides the solver to extend its partial assignment σ so that the distance between σ and previous models $\mu_1, \ldots, \mu_{n-1}$ will be as large as possible. We designed this strategy keeping in mind the goal of making the distance between the next model μ_n and the previous models as large as possible. More specifically, Guide-k-SAT uses the following greedy approach. Suppose a variable v is selected by the variable decision heuristic. Let $p(v)/n(v)$ be the number of times v was assigned 1/0 in previous models. If $p(v) > n(v)$, v is assigned 0; if $p(v) < n(v)$, v is assigned 1; if $p(v) = n(v)$ (including the case where no models have yet been identified), v is assigned a random value.

The ideas behind Rand-k-SAT and Guide-k-SAT are very simple and straightforward to implement, yet they turn out to be powerful and efficient for finding heterogeneously distributed models on well-structured problems, with an acceptable performance overhead compared to a modern SAT solver. On the one hand, we continue using all the modern SAT strategies, whose goal is to achieve solid performance on structured instances. On the other, we achieve sufficient diversification quality, either by selecting the phase randomly

TABLE I: Comparing Approaches to Generating Heterogeneous Models.

	DbS	Rand-k-SAT	Guide-k-SAT
Mean Quality	0.215	0.313	0.339
Overall Run-Time	47456	30307	28450

TABLE II: Trading Quality for Run-Time in Heterogeneous Model Generation.

	AllSAT-sampling	BaG; T=100	BaG; T=100000
Mean Quality	0.124	0.342	0.353
Overall Run-Time	8211	33392	183857

or by explicitly guiding the solver away from previous models.

DPLL-based sampling (DbS) is the best previous SAT-based approach to finding heterogeneous models. We implemented DPLL-based sampling as well as our algorithms Rand-k-SAT and Guide-k-SAT, and compared them experimentally on 66 benchmarks. The number of propositional clauses in the benchmarks varies from eight thousand to more than three million. In all the experiments, the required number of models was 10. All experiments were carried out on a machine with 4Gb of memory and two Intel Xeon CPU 3.60 processors. All the algorithms were implemented in the latest version of Intel's Eureka SAT solver. Eureka's default phase selection heuristic is RSAT's heuristic.

Table I compares DPLL-based sampling (DbS), Rand-k-SAT, and Guide-k-SAT in terms of mean diversification quality and overall run-time. Two scatter plots, comparing our best algorithm, Guide-k-SAT, and DPLL-based sampling in terms of run-time and quality are provided in Fig. 3. Similar scatter plots, comparing Guide-k-SAT and Rand-k-SAT, appear in Fig. 4. Our experiments yield two main conclusions.

First, both our algorithms are clearly preferable to DPLL-based sampling in terms of both quality and run-time. Table I confirms the overall advantage. Consider now the the right-hand scatter plot of Fig. 3 comparing the quality of Guide-k-SAT and DPLL-based sampling. A significant number of dots appear near the x-axis, far away from the diagonal, hinting that the gap is significant for some of the benchmarks. Now consider the run-time comparison scatter plot to the left. Guide-k-SAT outperforms DPLL-based sampling on most of the most difficult instances.

Second, Guide-k-SAT outperforms Rand-k-SAT in terms of both quality and run-time. The gap in run-time is not so significant: it stands at 6.5% overall. Also, the run-time comparison scatter plot in Fig. 4 shows that Guide-k-SAT is not always preferable to Rand-k-SAT. Now consider diversification quality. While the gap between average quality is not large, the quality comparison scatter plot clearly shows that Guide-k-SAT yields better diversification quality on every one of the benchmarks. Hence, for our examples, to achieve better performance and model diversification it is preferable to explicitly guide the SAT solver away from previous models (using Guide-k-SAT) than to use randomness (using Rand-k-SAT).

It is also possible to modify our algorithms to trade quality for run-time. Consider a variation of Rand-k-SAT, called *AllSAT-sampling*, that invokes the SAT solver once, but assigns random values only to variables selected for the first time or for the first time after a restart. Note that the solver is expected to keep assigning the same values to the variables for some restricted time after the beginning of the search or a restart due to RSAT's phase selection heuristic. A comparison of Table I and Table II shows that AllSAT-sampling is much faster than both Guide-k-SAT and Rand-k-SAT; however, the distribution quality is significantly worse. Accordingly, AllSAT-sampling can be recommended when the problem is computationally very complex.

Consider now a variation of Guide-k-SAT, called *BCP-aware Guide-k-SAT*. BCP-aware Guide-k-SAT tries to take into consideration the impact of Boolean Constraint Propagation (BCP) on the distance between the current partial assignment and the previous models. It performs BCP for both polarities, and measures the distance between the resulting partial assignments σ and previous models. Eventually, it picks the polarity that yielded the larger distance.

Specifically, the algorithm operates as follows. Suppose a variable v is selected by the variable decision heuristic. Let $p(v)$/$n(v)$ be the number of times v was assigned 1/0 in previous models. The variable v is assigned a value p as follows: if $p(v) > n(v)$, p is 1; otherwise p is 0. Then, BCP is carried out. Suppose that the set of variables V_p is assigned as a result of BCP. The algorithm saves the distance D_p between the partial assignment, induced by $\{v\} \cup V_p$, and the previous models. Afterwards, the algorithm unassigns $\{v\} \cup V_p$, assigns v the value $\neg p$, and propagates it using BCP. Suppose now that the set of variables $V_{\neg_p}$ is assigned as a result of BCP. The algorithm calculates the distance $D_{\neg p}$ between the partial assignment, induced by $\{v\} \cup V_{\neg p}$, and the previous models. If $D_{\neg p} > D_p$, the algorithm continues to the next decision. Otherwise, it unassigns $\{v\} \cup V_{\neg p}$, assigns v the value p, propagates using BCP, and continues to the next decision. Note that the algorithm first tries the polarity p that is less likely to result in better distance. The reasons is that if $\neg p$ is preferable, BCP is performed only twice; otherwise it is performed three times.

BCP-aware Guide-k-SAT is a costly algorithm, since it has to perform BCP two or three times per decision. Hence we limit its usage as follows. BCP-aware Guide-k-SAT is used until a certain number of conflicts T is encountered by the SAT solver. In addition, BCP-aware Guide-k-SAT is reinvoked after each model is discovered until T conflicts are encountered. The algorithm then uses plain Guide-k-SAT until the next model is encountered. Table II shows that BCP-aware Guide-k-SAT (BaG) improves distribution quality, but deteriorates run-time. Observe that it is possible to trade quality for run-time by changing T.

We also implemented XORSample [25] as well as the modified XORSample of [26]. We tried a variety of distribution quality values $(0.1, 0.01, \ldots, 0.0000001)$ and the number of generated XOR constraints $(1000, 10000, \ldots)$. Our results show that, depending on the configuration, XORSample is either slower by an order of magnitude compared to Rand-k-SAT and Guide-k-SAT (it timed-out on most of the instances), or its distribution quality is worse by approximately 10 times compared to Rand-k-SAT and Guide-k-SAT. Hence, although XORSample is useful on randomly generated instances and on small real-world formulas when a large number of models needs to be generated, it is inferior to other methods on difficult benchmarks when a small number of models needs to be generated.

Our experience shows that the best approach for generating multiple counterexamples in the framework of semiformal verification is to allow the user some control over the algorithm used within the tool. As our experimental results demonstrate, Guide-k-SAT is preferable as the default algorithm, since it exhibits the most attractive trade-off between run-time and solution diversification quality (which translates to efficient verification). However, we encountered a number of especially difficult cases where AllSAT-sampling was mandatory in order to satisfy performance requirements. In those cases, AllSAT-sampling was 25X faster than Guide-k-SAT (1 hour versus 25 hours to generate 10 models), although the diversification quality was 1.7X worse (0.181 versus 0.307). For easy test cases we recommend using BCP-aware Guide-k-SAT, where the trade-off between run-time and solution diversification quality is controlled by the threshold T.

IV. Checking Liveness Properties

A. Motivation

To the best of our knowledge, no attempt at semiformal verification of liveness properties has ever been described in the literature. We do not restrict our consideration to pure liveness properties, and by "liveness" we understand everywhere general liveness. Verifying liveness properties is required when the exact timing in end-to-end properties is not specified, and to check the absence of starvation. FV of liveness properties without prior aggressive abstraction is challenging: the complexity of their BMC-based verification is significantly more expensive than the verification of safety properties. Therefore the ability to perform semiformal verification of liveness properties is important.

One possible way of handling liveness properties would to convert them to equivalent safety properties, as explained in [30]. However, this approach is problematic in the semiformal verification context for the following reasons: 1) The number of property variables doubles when transforming a liveness property into a safety property, and 2) This translation makes sense when the resulting safety property is exhaustively checked. Therefore we did not explore this option in our work.

It is well known [31] that a violated liveness property always has a lasso-shaped counterexample: a state path consisting of a linear prefix and a loop. As explained in [32], in BMC of liveness properties these lasso-shapes paths are described with Boolean formulas parameterized by the size of the prefix and of the loop. SFV may help get to a design state close to the beginning of the loop, and/or to a neighborhood of a smaller loop. For example, to check starvation, it is necessary to bring the system into a state where resources have been requested by several clients. Applying BMC directly from the initial state is useless if the greatest feasible bound is insufficient to bring the system to such a state.

To apply classical algorithms based on semantic translations to check liveness properties in semiformal verification, the main challenge is to simulate their automata along the waypoint witness. Application of the algorithm proposed below is not limited to BMC-based semiformal verification; it may also be combined with other semiformal methods such as those described in [14], [15], [33].

B. Simulation of Non-deterministic Büchi Automata

Liveness properties cannot be represented as finite automata on finite words, and for their representation a finite automaton on infinite words (a so called Büchi automaton) is needed [18]. In practice it is more convenient to represent LTL properties with a more general form of Büchi automata — alternating Büchi automata [18]. For the sake of simplicity we describe our algorithm for regular (nondeterministic) Büchi automata only, but with minimal changes the same method may be applied to alternating Büchi automata as well. Unlike safety property automata, Büchi automata representing liveness properties are simulated symbolically, as described below.

In our algorithm we use a symbolic representation of the transition relation as a Boolean function of two sets of variables, current (unprimed) and next (primed) [19]: $\delta(w, w')$. We also introduce a map $\beta : w' \mapsto w$ to convert functions of next variables to functions of current variables. For example, $\beta(a' \wedge b') = a \wedge b$.

Let U_i be a symbolic representation of the states reachable at step i (*active states*) from one of the initial states while respecting the given witness. For the witness of the first waypoint, $U_0 = Q_0$ — the set of initial states of the automaton. For other witnesses U_0 is the symbolic representation of the endpoint of the automaton simulation along the previous waypoint witness. Let V_i be the set of pairs (w, w'), where $w \in U_i$ is a current active state, and w' is the next state reachable from w according to the transition relation δ, respecting the limitations imposed by the witness a_i at step i: $V_i = U_i \wedge \delta \wedge a_i$. The next variables computed this way become current variables for the next step, and the process is repeated: $U_{i+1} = \beta(\exists w. V_i)$. In this formula the existential quantifier selects the member w' of the pair $(w, w') \in V_i$.

We will illustrate this algorithm on the Büchi automaton in Fig. 5 for a 4-cycle long witness trace shown in Table III. The symbolic transition relation $\delta = \bigwedge_{i=0}^{4} \delta_i$, where

$$\begin{aligned}
\delta_0 &= q_0 \to q_0' \vee \neg a \wedge q_1' \vee a \wedge q_2' \\
\delta_1 &= q_1 \to \neg a \wedge q_1' \vee a \wedge q_2' \\
\delta_2 &= q_2 \to q_3' \\
\delta_3 &= q_3 \to \neg b \wedge q_3' \vee b \wedge q_4' \\
\delta_4 &= (q_4 \to q_4')
\end{aligned}$$

The values of U_i and V_i are shown in Table III. As expected, the values of U_i contain symbolic representation of the active states of the automaton at each simulation step. The initial state of the next BMC run should have $q_0 \vee q_2 = 1$.

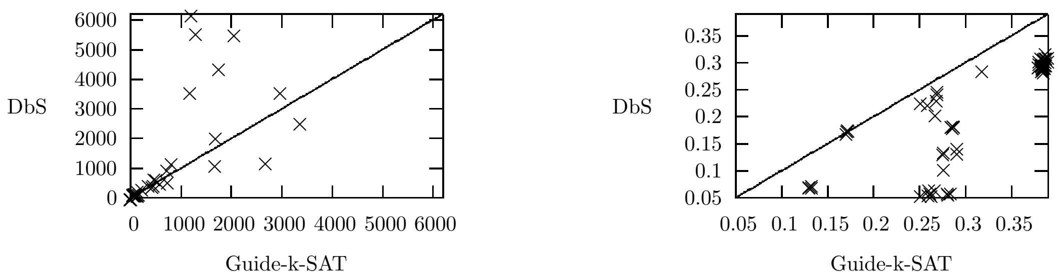

Fig. 3: Comparing the Run-Time in Seconds (on the Left) and the Quality (on the Right) of Guide-k-SAT (x-axis) vs. DPLL-based Sampling (y-axis)

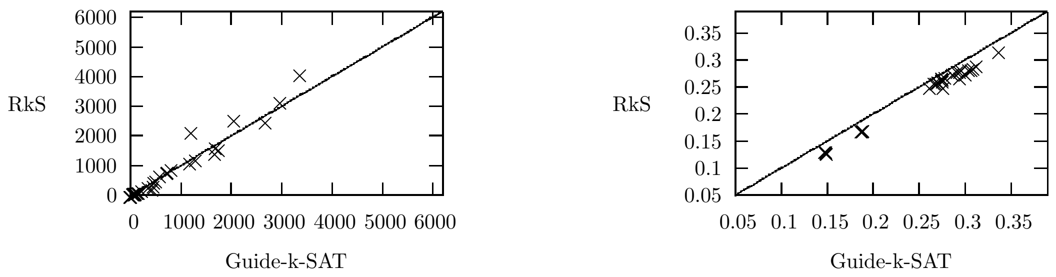

Fig. 4: Comparing the Run-Time in Seconds (on the Left) and the Quality (on the Right) of Guide-k-SAT (x-axis) vs. Rand-k-SAT (y-axis)

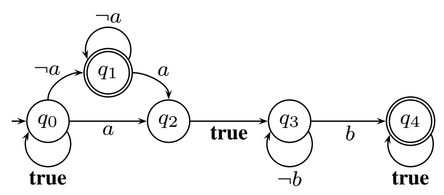

Fig. 5: Büchi automaton with accepting states q_1 and q_4

TABLE III: Simulation of Büchi automaton

Time	a	b	U_i	V_i
0	0	0	q_0	$q_0 \wedge \neg a \wedge \neg b \wedge (q_0' \vee q_1') \wedge \bigwedge_{i=1}^{4} \delta_i$
1	0	0	$q_0 \vee q_1$	$(q_0 \vee q_1) \wedge \neg a \wedge \neg b \wedge (q_0 \rightarrow q_0' \vee q_1')$ $\wedge (q_1 \rightarrow q_1') \wedge \bigwedge_{i=2}^{4} \delta_i$
2	1	0	$q_0 \vee q_1$	$(q_0 \vee q_1) \wedge a \wedge \neg b \wedge (q_0 \rightarrow q_0' \vee q_2')$ $\wedge (q_1 \rightarrow q_2') \wedge \bigwedge_{i=2}^{4} \delta_i$
3	0	0	$q_0 \vee q_2$	—

V. Test Cases

We implemented the algorithm in Intel's proprietary FV tool and chose three CPU design blocks for our experiments. These design blocks had been extensively tested in simulation and the design was believed to be mature. The blocks were modeled in SystemVerilog and included novel features carrying high risk. The properties were captured using SystemVerilog Assertions (SVA). We chose blocks of sizes that SAT-based FV engines could handle — the full cone of influence of a typical assertion comprised 1K inputs, 5K state elements, and 75K gates. As a result, the FV confidence level was not high enough in all test cases, as the BMC bound reached by the traditional BMC approach was not sufficient. In most cases, after reducing the models using both manual and automatic techniques, design scenarios requiring more than forty clock cycles could not be addressed. It is worth noting that our attempts to apply industrial semiformal verification tools yielded no tangible results. This was due to complex environments that needed to be synchronized and to the unique properties and large size of the CPU design blocks.

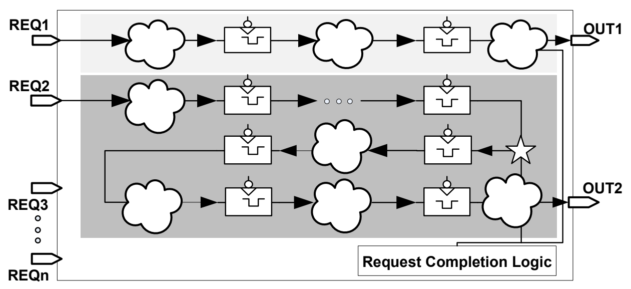

Fig. 6: Request Tracker

The first block, a Request Tracker, is responsible for managing various request types and ensuring the correct execution order of the requests, giving preference to high-priority requests while not starving low-priority requests. Requests arrive from various sources, and each is associated with a unique identifier (ID). A high-level diagram of Request Tracker is shown in Fig. 6.

The different request types vary in the time needed to process them, e.g. a $REQ1$ request (path $REQ1$ — $OUT1$) requires considerably fewer clock cycles than a $REQ2$ request (path $REQ2$ — $OUT2$). We chose to experiment with $REQ2$, which had not been properly addressed in FV due to BMC bound limitations.

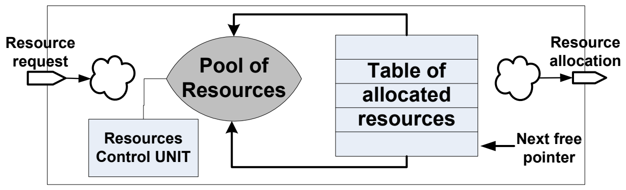

Fig. 7: Resource Manager

The second block, a Resource Manager, is responsible for controlling resources and making sure that no resource is allocated twice and that none are lost. The resources are kept in the pool and allocations/deallocations are recorded using a cyclic table. See Fig. 7 for a high-level diagram of the block.

The third block, a Flow Manager, implements a mechanism to control a complex flow involving many agents. It comprises a central FSM with additional smaller FSMs around it, each being responsible for a specific flow scenario or sending and receiving data from a certain agent. The central FSM controls the flow, supervising all the smaller FSMs around it. This block was used for algorithm experiments with liveness properties, as the main concern is that the flow will eventually finish successfully without getting stuck in live-lock due to a bug in one of the FSMs.

VI. Results

In this section we describe the results of applying the proposed SFV algorithm to the RTL blocks described in Section V. The most important result was the exposure of three real corner-case bugs described in Section VI-A. We also inserted several artificial corner-case bugs described in Section VI-B. We sought to corroborate different characteristics of the algorithm, namely its ability to adequately cover design state space using the multiple witness approach.

A. Real Corner-Case Bugs in Mature Designs

Our SFV algorithm revealed the following bugs in the Resource Manager, two of them critical. These bugs could be revealed neither in simulation, nor using traditional FV, nor using SFV with a single witness.

- Incorrect STALL calculation in a very specific combination of allocation requests, which causes resources to be lost.
- A bug in recovery/restart event handling which results in not all of the allocated resources being correctly sent back to the resource pool.
- Corruption, in a scenario involving extremely high allocation traffic, of a mechanism which validates resource integrity in the Resource Control Unit.

B. Testing the Ability to Adequately Cover Design State

We inserted an artificial corner-case bug in the Request Completion Logic sub-block which causes a failure when multiple $REQ2$ type requests from particular sources and ID ranges arrive in a particular order. The bug results in one of the requests being incorrectly marked as completed. This bug could not be revealed with the simulation regression.

TABLE IV: Resource Manager Verification Results

CP/Asrt	BMC		SFV, single		SFV, multiple	
	Result	Bound	Result	Bound	Result	Bound
Line 4	covered	69	covered	69	covered	69
Line 8	covered	71	covered	77	covered	77..83
Line 12	uncov.	24	covered	89	covered	85..95
Line 16	uncov.	26	covered	99	covered	93..107
Line 19	uncov.	26	covered	113	covered	99..119
Line 0	N/A	N/A	covered	129	covered	107..133
Asrt	TO	38	TO	42	failed	142

We used nine different waypoints modeling several $REQ2$ requests in various pipe stages on a path $REQ2$ — $OUT2$; for example, the one marked by a star in Fig. 6. In this and other experiments we used *general* waypoints (waypoints previously defined by validation engineers for other purposes) in order to eliminate the possibility that prior knowledge about the bugs might lead us unconsciously to craft waypoints leading directly to them. For each waypoint we calculated 5 witnesses, targeting each of the twelve assertions from 9×5=45 different initial states defined by these witnesses. The cover points occurred at bounds 64–70, and verification took 1406–3379 seconds (on a machine with 4Gb memory and two Intel Xeon CPU 3.60 processors). A failure was detected by one out of 12 assertions from only one initial state, whereas runs from the other 44 initial states missed the problematic scenario. It occurred at bound 34 (70+34=104 clock phases from the original initial state) after 14707 seconds.

We inserted an artificial corner-case bug into the Resource Manager logic which calculates the condition for next request $STALL$. This caused *Next free pointer* to wrap around early due to illegal allocation, thereby running over other resources in the table. We used general cover points as waypoints asserting that table lines were allocated, and the table was incrementally filled up until the wraparound. We ran traditional BMC and SFV with single as well as multiple witnesses. The assertion verified that resources were not being lost in the system. In all cases a timeout of 20 hours was used. Results are summarized in Table IV.

A wraparound happens after the 19th table line is allocated, as the cyclic allocation table size is 20. BMC could not get beyond the allocation of line 8, and the multiple witness approach was needed in order to come across the problematic combination of resource requests. The total number of verification runs was 3(witnesses)$^{6(\text{waypoints})}$=729. Note that the SFV algorithm does not necessarily produce the shortest counterexample — line 8 was reached with bound 71 using BMC whereas using SFV it was reached with bound 77 to 83.

We experimented with liveness properties in the Flow Manager block. The properties validate forward progress with the control FSM (dispatcher), eventually reaching predefined control points without getting stuck, e.g. due to a bug in one of the FSMs. The proof assumes the legal behavior of the surrounding agents. We used waypoints describing the state transitions of the dispatcher FSM. Although we did not find any real design bugs, we validated the correctness of the algorithm by properly detecting a known deep bug using our approach. The failure was detected faster: 1575 seconds (509 seconds towards the waypoint and 1064 seconds to get a counterexample) vs. 5470 seconds for traditional BMC (3.5X faster). This is due to the run-time reduction phenomenon described in Section II-A.

VII. Conclusion and Future Work

The method suggested in this paper for pure SAT-based semiformal verification is very simple to grasp and straightforward to implement, yet it exhibits a superior ability to achieve good design coverage and detect deep, corner-case bugs in industrial-scale designs. The experimental results confirm this by exposing both real and artificial design bugs missed by simulation (due to coverage limitations) and classic FV (due to bound limitations). These encouraging results were achieved with a relatively small amount of work on the part of the validation engineers, much less than the effort required by the traditional FV and simulation approaches applied prior to our experiments. Moreover, the suggested method can save the substantial effort usually invested in reducing designs to fit the capacity limitations of FV tools, as it can replace such activities.

As a by-product, we developed two SAT-based algorithms, Rand-k-SAT and Guide-k-SAT, that are able to efficiently find a number of heterogeneous models for a given problem. We also discuss variations of Rand-k-SAT and Guide-k-SAT that allow the user to achieve the desired balance between performance and solution diversification quality. We have also proposed an extension of the semiformal verification algorithm for liveness properties.

In our future work we intend to study how different diversification techniques affect bug detection capabilities and to collect more experimental data on semiformal verification of liveness properties to better understand the practical utility of this technique.

Acknowledgments

The authors would like to thank Vadim Ryvchin, Paul Inbar, Roy Frank, Tamir Salus, Yael Zbar, Zurab Khasidashvili, Asi Sapir, Haim Kerem, Dani Even-Haim, Amit Palti, Eli Singerman, and Alon Flaisher for their valuable suggestions, ideas, experimental results, reviews, and support of our work.

References

[1] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," *Advances in Computers*, vol. 58, 2003.

[2] J. Bhadra, M. S. Abadir, L.-C. Wang, and S. Ray, "A survey of hybrid techniques for functional verification," in *IEEE Design and Test of Computers*, vol. 24, 2007, pp. 112–123.

[3] K. L. McMillan, *Symbolic Model Checking*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[4] K. Ravi and F. Somenzi, "High density reachability analysis," in *ICCAD*, 1995.

[5] J. Yuan, J. Shen, J. A. Abraham, and A. Aziz, "On combining formal and informal verification," in *CAV*, 1997, pp. 376–387.

[6] R. Fraer, G. Kamhi, B. Ziv, M. Y. Vardi, and L. Fix, "Prioritized traversal: Efficient reachability analysis for verification and falsification." in *CAV*, 2000, pp. 389–402.

[7] G. Cabodi, S. Nocco, and S. Quer, "Improving sat-based bounded model checking by means of bdd-based approximate traversals," in *DATE*, 2003, pp. 10 898–10 905.

[8] M. K. Ganai and A. Aziz, "Rarity based guided state space search," in *GLSVLSI*, 2001, pp. 97–102.

[9] M. Ganai, P. Yalagandula, A. Aziz, A. Kuehlmann, and V. Singhal, "Siva: A system for coverage-directed state space search," *J. Electron. Test.*, vol. 17, no. 1, pp. 11–27, 2001.

[10] C. R. Ho, M. Theobald, B. Batson, J. Grossman, S. C. Wang, J. Gagliardo, M. M. Deneroff, R. O. Dror, and D. E. Shaw, "Post-silicon debug using formal verification waypoints," in *DVCon*, 2009.

[11] A. Kuehlmann, K. L. McMillan, and R. K. Brayton, "Probabilistic state space search," in *ICCAD*, 1999, pp. 574–579.

[12] P. Yalagandula, V. Singhal, and A. Aziz, "Automatic lighthouse generation for directed state space search," in *DATE*, 2000, pp. 237–242.

[13] P. Bjesse and J. H. Kukula, "Using counterexample guided abstraction refinement to find complex bugs," in *DATE*, 2004, pp. 156–161.

[14] P. H. Ho, T. Shiple, K. Harer, J. Kukula, R. Damiano, V. Bertacco, J. Taylor, and J. Long, "Smart simulation using collaborative formal and simulation engines," in *ICCAD*, 2000, pp. 120–126.

[15] A. Aziz, J. Kukula, and T. Shiple, "Hybrid verification using saturated simulation," in *DAC*, 1998, pp. 615–618.

[16] D. L. Dill and C. H. Yang, "Validation with guided search of the state space," in *DAC*, 1998, pp. 599–604.

[17] E. Cerny, A. Dsouza, K. Harer, P.-H. Ho, and T. Ma, "Supporting sequential assumptions in hybrid verification," in *ASP-DAC*, 2005, pp. 1035–1038.

[18] M. Y. Vardi, "An automata-theoretic approach to linear temporal logic," in *Proceedings of the VIII Banff Higher order workshop conference on Logics for concurrency: structure versus automata*, 1996, pp. 238–266.

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

[20] E. M. Clarke, D. Kroening, J. Ouaknine, and O. Strichman, "Computational challenges in bounded model checking," *Journal of Software Tools and Technology Transfer*, vol. 7, no. 2, pp. 174–183, 2005.

[21] R. Armoni, S. Egorov, R. Fraer, D. Korchemny, and M. Vardi, "Efficient LTL compilation for SAT-based model checking," in *ICCAD*, 2005.

[22] J. Yuan, K. Albin, A. Aziz, and C. Pixley, "Simplifying boolean constraint solving for random simulation-vector generation," in *ICCAD '02*, 2002, pp. 123–127.

[23] W. Wei, J. Erenrich, and B. Selman, "Towards efficient sampling: Exploiting random walk strategies," in *AAAI*, 2004, pp. 670–676.

[24] N. Kitchen and A. Kuehlmann, "Stimulus generation for constrained random simulation," in *ICCAD*, 2007, pp. 258–265.

[25] C. P. Gomes, A. Sabharwal, and B. Selman, "Near-uniform sampling of combinatorial spaces using XOR constraints," in *NIPS*, B. Schölkopf, J. C. Platt, and T. Hoffman, Eds., 2006, pp. 481–488.

[26] S. Plaza, I. L. Markov, and V. Bertacco, "Random stimulus generation using entropy and XOR constraints," in *DATE*, 2008, pp. 664–669.

[27] K. L. McMillan, "Applying SAT methods in unbounded symbolic model checking," in *CAV*, 2002, pp. 250–264.

[28] S. K. Lahiri, R. E. Bryant, and B. Cook, "A symbolic approach to predicate abstraction," in *CAV*, 2003, pp. 141–153.

[29] K. Pipatsrisawat and A. Darwiche, "A lightweight component caching scheme for satisfiability solvers," in *SAT*, 2007, pp. 294–299.

[30] A. Biere, C. Artho, and V. Schuppan, "Liveness checking as safety checking." *Electr. Notes Theor. Comput. Sci.*, vol. 66, no. 2, 2002. [Online]. Available: http://dblp.uni-trier.de/db/journals/entcs/entcs66.html#BiereAS02

[31] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, 6th ed. MIT Press, 2008.

[32] M. K. Ganai, A. Gupta, and P. Ashar, "Beyond safety: customized sat-based model checking," in *DAC '05: Proceedings of the 42nd annual Design Automation Conference*. New York, NY, USA: ACM, 2005, pp. 738–743.

[33] M. K. Ganai, A. Aziz, and A. Kuehlmann, "Enhancing simulation with BDDs and ATPG," in *DAC*, 1999, pp. 385–390.