Acceleration of SAT-based Iterative Property Checking

Daniel Große and Rolf Drechsler

Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
{grosse,drechsle}@informatik.uni-bremen.de

Abstract. Formal property checking is used to check whether a circuit satisfies a temporal property or not. An important goal during the development of properties is the formulation of general proofs. Since assumptions of properties define the situations under which the commitments are checked, in order to obtain general proofs assumptions should be made as general as possible. In practice this is accomplished iteratively by generalizing the assumptions step by step. Thus, the verification engineer may start with strong assumptions and weakens them gradually. In this paper we propose a new approach to speed up SAT-based iterative property checking. This process can be exploited by reusing conflict clauses in the corresponding SAT instances of consecutive property checking problems. By this the search space is pruned, since re-computations of identical conflicts are avoided.

1 Introduction

Nowadays, for successful circuit designs *Property Checking* (PC) is very important. Typically such a property consists of two parts: an *assume part* which should imply the *proof part*. In the last years tools based on *Satisfiability* (SAT) performed better than classical BDD-based approaches since SAT procedures do not suffer from the potential "size explosion" of BDDs. In SAT-based PC the initial SAT instance is generated from the circuit description together with the property to be proven. Usually, the largest part will result from the unrolled circuit description. In comparison, the parts for the commitments, assumptions, and the extra logic are much smaller. From a practical perspective, during PC as long as no design bug is found the circuit design remains unchanged, but the verification engineer modifies and adds new properties. Thus, the PC tool is used interactively. For the verification engineer on the one hand, proving becomes more easy if the assumptions of a property are very strong, i.e. the property is very restrictive and argues only over a small part of the design space. On the other hand, such proofs are not very general. Hence in practice, the formulation of a property is an iterative process. E.g., the engineer starts writing a property with strong assumptions. Then, the engineer stepwise weakens some of the assumptions to obtain a more general proof.

The basic idea is to exploit the iterative process of PC. As can be seen only a very small part of the verification problem changes in consecutive PC runs if the assumptions are weakened. Re-computations can be avoided if learned information is reused for consecutive SAT problems. *Bounded Model Checking* (BMC) as introduced in [1] reduces the verification problem to a SAT problem and then searches for counter-examples in executions whose length is bounded by k time steps. For BMC, it has been suggested to reuse constraints on the search space deduced in instance k for solving the consecutive instance $k+1$ faster [4]. However, in [4] this concept is only used during the proof of a single or more fixed properties.

In this paper we use BMC as described in [5], thus, a property only argues over a finite time interval and during the proof there is no restriction to reachable states. In contrast to [4], here two SAT instances for slightly different PC

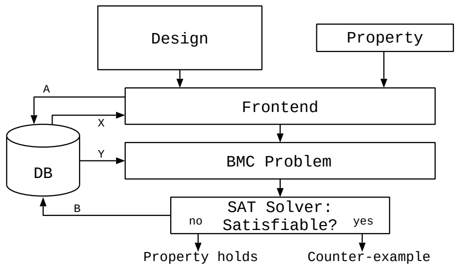

Fig. 1. Property Checking Flow

problems are considered and information from the two properties with respect to the underlying circuit is utilized. This enables to reuse learned conflict clauses in the SAT instance of the consecutive PC problem.

2 Acceleration of Iterative Property Checking

In this section the approach for reusing conflict clauses during iterative PC is presented. Before the details are given, the work flow is illustrated in Figure 1.

At first the design and the property are compiled into an internal representation. In this step information to allow for a syntactic comparison between properties is stored in the data base (A). Then the internal representation is converted into a BMC problem expressed as a CNF formula. While solving this SAT instance the references to the clauses that lead to a new conflict clause are stored in a data structure. After termination of the SAT solver this conflict clause information can be related to the single assumptions and commitments of the checked property. Finally this information is minimized and added to the data base (B). Now assume that PC is repeated but the property has been weakened. Then, this is detected (X) and before the BMC problem is given to the SAT solver conflict clauses are read from the data base, analyzed and reused (Y), if possible.

Let M be the set of clauses resulting from the translation of the design D, let P be the set of clauses resulting from the property p. Then P can be partitioned into $P = A \cup C \cup R$, where A are the clauses from the assumptions, C from the commitments and R the clauses to "glue" the assumptions and the commitments of the property together. Now consider two consecutive runs of the property checker for the unchanged design D and for two properties p^F(first) and p^S(second). Assume that the property p^S has been derived from the property p^F by weakening some of the assumptions. Let $P^F = A^F \cup C^F \cup R^F$ be the resulting clauses of the property of the first run and $P^S = A^S \cup C^S \cup R^S$ the clauses for the second run, respectively. Further assume that the variables in P^S are renamed with a variable mapping function which maps a variable from the second set of variables V_S to the according variables of the variable set V_F from the first run. Then the following holds:

1. $C^S = C^F$, since the commitments of properties p^S and p^F are equal.
2. $R^S = R^F$ since the variables to combine the assumptions and commitments can be identified.
3. $A^S \subset A^F$ because the assumptions of p^S are weaker than the assumptions of p^F.

Table 1. Overhead for arbiter

Cells	Property	Clauses	Literals	Result	Time (sec)	
					std	reuse
100	mutualexclusion	240,776	541,742	holds	9.15	9.57
100	lowestWins_50	161,399	363,193	holds	14.15	14.49
200	mutualexclusion	961,576	2,163,542	holds	176.65	177.78
200	lowestWins_50	642,799	1,446,393	holds	588.30	590.45

Table 2. Acceleration for arbiter

Cells	Property	Clauses	Literals	Result	Time (sec)		Reused	Speed-up
					std	reuse	Cl. (%)	
100	mutualexclusion	161,076	362,442	holds	13.26	13.01	20.23	1.0
100	lowestWins_50	161,247	362,839	holds	8.71	4.54	100.00	1.9
200	mutualexclusion	642,176	1,444,942	holds	1078.80	343.77	6.23	3.1
200	lowestWins_50	642,347	1,445,339	holds	656.35	22.70	100.00	28.9

Since the clauses M of the design do not change only the clauses resulting from the two properties p^F and p^S have to be compared. Under the assumptions and conclusions from above the following holds:

$$P^F - P^S = (A^F \cup C^F \cup R^F) - (A^S \cup C^S \cup R^S) = A^F - A^S$$

With this result it can be concluded that all conflict clauses can be reused which are *not* a result of an implication caused by a clause of $A^F - A^S$. In other words we have to identify the conflict clauses which have been deduced exclusively from the intersection of the two consecutive PC problems. This intersection is given by $(M \cup P^F) \cap (M \cup P^S) = M \cup A^S \cup C^F \cup R^F$. Thus, for each conflict clause of the first run the sequence of clauses which produced that conflict clause have to be determined, since with this information we can exactly identify the source of the conflict in terms of the two properties p^F and p^S. This becomes possible, if we further know which clauses have been produced by the design, the individual expressions in the assume part and the individual expressions of the proof part of both properties. Finally for a conflict clause cl the minimal source information is stored which allows to check if cl was produced by a clause of the design or by an assume expression or a proof expression. Altogether it can be decided which conflict clauses of the first run can be reused to speed up the current proof.

3 Experimental Results

To allow for access of necessary information during PC we have implemented a SAT-based property checker on top of zChaff [3]. All experiments have been carried out in the same system environment on an Athlon XP 2800 with 1 GByte main memory. The following experiments always consist of two steps. First, for a circuit a property with "overly" strong assumptions is proved. This is done with and without our approach to measure the time overhead. Next, we prove the same property but in a more general version, i.e. some of the assumptions of the property have been weakened. In this case we measure the speed-up that can be achieved by reusing conflict clauses.

In a first series of experiments we considered a scalable bus arbiter that has been studied frequently in formal hardware verification (see e.g. [2]). The considered properties for the arbiter circuit are mutual exclusion of the outputs of the arbiter and `lowestWins`. The second property states that if exactly one token is set and no cell is waiting and exactly the request i is high then the corresponding acknowledgement i will be set in the same clock cycle. In Table

Table 3. Overhead for FIFO

Size	Property	Clauses	Literals	Result	Time (sec)	
					std	reuse
64	nochange	68,077	156,723	holds	14.82	14.92
128	nochange	156,595	361,173	holds	101.83	102.03

Table 4. Acceleration for FIFO

Size	Property	Clauses	Literals	Result	Time (sec)		Reused	Speed-up
					std	reuse	Cl. (%)	
64	nochange	68,072	156,712	holds	14.80	2.16	100.00	6.9
128	nochange	156,590	361,162	holds	101.72	6.42	100.00	15.8

1 the overhead for our approach is given for different arbiter instances (column *Cells*). In the second column the name of the considered property is shown. The next two columns provide information on the corresponding SAT instance. In column *Result* it is shown whether the property holds or not. Next, the run time needed without and with our approach is given in column *std* and column *reuse*, respectively. The difference between the two given run times is the time needed to store learned information into the data base. As can be seen the overhead is negligible, i.e. less than 1% of the run time for the larger examples.

The achieved improvement of the proposed approach for the arbiter is shown in Table 2. E.g. in the weakened variant of the property `mutualexclusion` the assumption that no arbiter cell is waiting is no longer assumed. The first seven columns give similar information as in Table 1. Because the considered properties have been weakened the resulting number of clauses and literals decreases. However, since for each property learned information can be found in the data base, conflict clauses can be reused. Thus, column *Reused Cl.* gives the percentage of reused clauses. In the last column the achieved speed-up is shown. As can be seen for the 100 cell arbiter in case of the property `mutualexclusion` no speed-up results. But for the three remaining examples a significant speed-up was obtained, i.e. up to nearly a factor of 30.

In a second series of experiments we studied FIFOs of different depth. As a property we prove that the content of a FIFO does not change under the assumption that no write operation is performed. In the initial version of this property it has also been assumed that no read operation is performed. Similar information as for the arbiter examples is provided in Tables 3 and 4, respectively. Also in this case for larger examples a speed-up of more than a factor of 10 can be observed.

References

1. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *LNCS*, pages 193–207. Springer Verlag, 1999.
2. D. Große and R. Drechsler. *CheckSyC*: An efficient property checker for RTL SystemC designs. pages 4167–4170, 2005.
3. M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf.*, pages 530–535, 2001.
4. Ofer Shtrichman. Pruning techniques for the SAT-based bounded model checking problem. pages 58–70, 2001.
5. K. Winkelmann, H.-J. Trylus, D. Stoffel, and G. Fey. A cost-efficient block verification for a UMTS up-link chip-rate coprocessor. In *Design, Automation and Test in Europe*, volume 1, pages 162–167, 2004.