Enhancing SAT-based Bounded Model Checking using Sequential Logic Implications*

Rajat Arora and Michael S. Hsiao (raarora, hsiao) @vt.edu
Electrical and Computer Engineering Department, Virginia Tech, Blacksburg, VA.

Abstract

We present a novel technique of improving the SAT-based Bounded Model Checking, by inducing powerful sequential signal correlations (crossing time-frame boundaries) into the original CNF formula of the unrolled circuit. A quick preprocessing on the circuit-under-verification, builds a large set of direct and indirect sequential implications. The non-trivial implications (spanning multiple time-frames) are converted into two-literal clauses. These clauses are quickly replicated throughout the unrolled sequential circuit, and appended to the existing CNF database. The added clauses prune the overall search space of SAT-solver engine and provide correlation among the different variables, which enhances the Boolean Constraint Propagation (BCP). Experimental Results for checking difficult instances of random safety properties on ISCAS'89 benchmark circuits show that more than 148x speedup can be achieved over the conventional approach.

1. Introduction

With the advances in VLSI and system-on-a-chip (SOC) technology, the complexity of digital systems has increased manifold. Verification of these intricate systems has become one of the foremost concerns for the validation and verification engineers. In the last decade, Model Checking based verification [15, 16] has gained much attention. It determines if the implemented design satisfies a given set of properties. Binary Decision Diagram (BDD)-based Symbolic Model Checking [5, 6] has shown to hold promise. However, BDDs are known to suffer from the memory explosion problem, and hence fail for bigger circuits with large numbers of flip-flops/state variables. Automatic Test Pattern Generation (ATPG)-based Unbounded Model Checking [14] on the other hand can suffer from temporal explosion. With recent advances in Satisfiability (SAT) solvers [1-4], SAT-based Bounded Model Checking (BMC) [7] is gaining significant importance. In this technique, the sequential circuit is unrolled into k time-frames, and counterexamples (or bugs) are searched in this bounded length k. Comparisons of SAT-based approach with Sequential ATPG and BDD-based approaches can be found in [12] and [13].

The state-of-the-art SAT solvers [1-4] are usually based on the Conjunctive Normal Form (CNF). While trying to satisfy the given CNF formula, the SAT solver makes decisions based on a given set of variable selection heuristics [1-4]. It learns dynamically from the conflicts encountered during the search, and generates conflict-induced clauses [1, 2] that can subsequently constrain the search. However, the conflict clauses learnt dynamically have the following disadvantages:

- Not all learned clauses are useful, especially the long clauses.
- Set of all learned clauses can grow very large.
- The clauses are learned gradually over the entire SAT search, which may take a long time.

Recently, efforts have been made to improve the SAT-based BMC by inducing useful information into the original CNF formula before the SAT solver starts. This overcomes the above disadvantages to some extent. In [11], the authors perform BDD-based approximate reachability analysis to gather information on the state space. This state space related information is converted to clauses and appended to the original CNF formula, which in turn restricts the search space of the SAT-solver. In [8], the authors induce signal correlation into the original CNF by locally building up BDDs around the seed node (which is selected statically or dynamically). Every path leading to '0' in such a BDD denotes a conflict, and is added as multi-literal clause to the existing CNF. However, in general, the locally built BDDs are not helpful in extracting global relations among signals across time frames.

In our proposed technique, we efficiently identify non-trivial relations among signals over the sequential circuit, especially those crossing time-frame boundaries. This is unlike the previous work [8], where only relationships among signals in the combinational portion of the circuit are learnt. Moreover, our sequential relations are learned without unrolling the circuit; thus, the static learning is fast. The preprocessing phase involves building the sequential implication graph for the circuit-under-verification (CUV), converting the non-trivial implications into two-literal clauses, and finally replicating these clauses over the entire unrolled circuit. Experimental results show that more than 148x speedup can be achieved over conventional approaches.

The rest of the paper is organized as follows. In Section 2 we give a brief overview of SAT-based Bounded Model Checking and Sequential Logic Implications. In Section 3 we present our observations when the non-trivial sequential implications are applied to the problem at hand, and show how these implications are converted into two-literal-clause form and replicated across time-frame boundaries. Experimental results are given in Section 4, and Section 5 concludes the paper.

2. Preliminaries

2.1 SAT-based Bounded Model Checking

In SAT-based BMC, given a sequential Circuit Under Verification (CUV) and the associated safety/liveness properties, the SAT-solver tries to determine their satisfiability/unsatisfiability in the bounded length 'k'. The sequential circuit is firstly unrolled into 'k' time-frames and then Bounded Model Checking (BMC) Circuitry called as Monitor Circuit [14, 17] is constructed corresponding to the property to be verified. A CNF database is built for this transformed circuit and the SAT solver is asked to satisfy the Monitor Circuit output to logic '1'. The efficiency of the BMC formulation depends on the complexity of the CUV as well as the underlying SAT solver. In order to speed up the search, relations within the circuit can be very useful because they can induce meaningful information into the original CNF formula and hence help to constrain the search space. Global relations across time frames can be extremely useful. We next discuss how these global relations are obtained.

2.2 Static logic Implications in sequential circuits

Static logic implications are obtained by asserting the logic values '0' and '1', respectively, to every gate in the circuit. They are made up of direct, indirect and extended backward implications.

* supported in part by NSF Grants CCR-0196470 and CCR-0305881.
The readers are referred to [9, 10] for an in-depth study on static logic implications. In our approach, only the indirect implications are used to induce relations into the original CNF formula; extended backward implications are computationally expensive and hence not utilized. A memory-efficient, directed-graph based technique [10] for representing the sequential implication relations has been used. For a given circuit with $N$ gates, the total number of nodes in this graph is $N^*2$, since each gate can take on a logic value of 0 or 1. Also, by representing the sequential implications as a graph, transitive closure of a node can be easily obtained using the depth-first search technique. Figure 1 shows the example sequential circuit and Figure 2 shows its partial implication graph.

A directed edge between two nodes (e.g. from node $g'=1$ to node $h=1$) represents an implication between the nodes (i.e. $(g, 1) \rightarrow (h, 1)$). The weight associated with an edge represents the relative time frame associated with the implication. It should be noted that the time-frames are bounded by D flip-flops and the current time-frame is always time-frame ‘0’. When an implication propagates across a D flip-flop, the time frame is incremented or decremented accordingly. For example, $e=1$ implies $j=1$ in time frame ‘1’, and is represented as $(e, 1) \rightarrow (j, 1, -1)$. This nice representation allows for implications to span multiple time frames without explicit unrolling of the circuit. For instance, because $(g, 1) \rightarrow (e, 1)$, and $(e, 1) \rightarrow (j, 1, -1)$ as shown in the implication graph in Figure 2, $(g, 1) \rightarrow (j, 1, -1)$ by transitive property. In general, the transitive help leads to induce implication relations with edge weights ranging from ‘-n’ to ‘+n’ (n being a whole number). However, in case of a loop, ‘n’ can be infinity. Hence, we restrict this value of ‘n’ in our implementation and make it user-specified. We call this ‘n’ as sequential implication depth.

As an illustration, the direct implications of gate ‘g’ set to value ‘1’ are $impl\{g, 1\} = \{g, 1\}, (h, 1), (i, 1), (j, 1, -1), (m, 0), (e, 1), (f, 1), (l, 1, -1)\}$. The Indirect implications of a node can be computed by simply logic simulating the transitive closure of its direct implications, time-frame by time-frame, in an event-driven fashion. The transitive closure of $impl\{g, 1\}$ is $\{(g, 1), (h, 1), (i, 1), (m, 0), (e, 1), (f, 1), (l, 1, -1), (k, 0, -1), (d, 0), (j, 1), (c, 0), (l, 1, -1)\}$. Now we see that $(h, 1)$ or $(i, 1)$ individually, do not imply anything on gate ‘j’ (in time-frame ‘0’). However, together they imply $(j, 0)$. Thus, $(g, 1) \rightarrow (j, 0)$ is an indirect implication. Now, $(j, 0)$ together with $(d, 0)$ implies $(k, 1)$ in time-frame ‘0’. Therefore, $(g, 1) \rightarrow (k, 1)$ is also an indirect implication. It should be noted that $(k, 1)$, an indirect implication of $(g, 1)$ is obtained by making use of implications of time-frame ‘-1’, since $(j, 1, -1) \rightarrow (k, 0, -1) \rightarrow (d, 0)$ and hence is a non-trivial implication. These resulting implications are added to the implication graph of the circuit along with their corresponding contrapositive implications. Thus, $impl\{g, 1\} = \{(g, 1), (h, 1), (i, 1), (m, 0), (e, 1), (f, 1), (j, 1, -1), (k, 0, -1), (d, 0), (j, 1), (c, 0), (j, 0), (k, 1), (d, 1, l), (f, 0, 1), (d, 0, 2)\}$. Note that the indirect implications can also cross time frame boundaries.

3. Application of Sequential Implications to SAT

3.1 Constrained Search Space and Enhanced Boolean Constraint Propagation (BCP)

A two-time-frame unrolled circuit, corresponding to example circuit of Figure 1 is shown below in Figure 3.

The partial CNF of above circuit is as follows:

$$\begin{align*}
(h + j)(h + j)(-h + -i + j)(-j + -k)(-d + -k)(j + d + k)g + e' = 1
\end{align*}$$

We see that making the decision $k' = 0$ satisfies a total of two clauses, $(j + -k)$ and $(d' + -k)$. It does not yield any unit clauses. Considering the indirect implications of the corresponding sequential circuit (Figure 1), we know that $(g', 1) \rightarrow (j', 0)$ is an indirect implication, and from contrapositive law $(j', 1) \rightarrow (g', 0)$. The two-literal clause corresponding to these two implications is $(-g' + -j')$. Also $(g', 1) \rightarrow (k', 1)$ is an indirect implication with its contrapositive being $(k', 0) \rightarrow (g', 0)$. The two-literal clause in this case is $(-g' + k')$. In general, a two-literal clause embeds in itself both the indirect implication as well as its contrapositive. Now, if we add the clause $(-g' + k')$ to the implication $(g', 1) \rightarrow (k', 1)$, assigning $k' = 0$ and doing Boolean Constraint Propagation (BCP) will yield unit clauses, implying $g' = 0$ and then $m' = 1$. The added clause helps to satisfy a total of eight clauses instead of two. This is shown in Figure 4 below. It is also evident that without adding any clauses, the decision $k' = 0$ followed by $g' = 1$ causes a conflict, and hence results in backtracking; with the added clause this backtracking is avoided. Addition of a large number of these non-trivial two-literal clauses provides correlation among the different CNF variables, which not only prunes the overall search space of the SAT-solver engine, but also improves the BCP. These implications between intermediate points of the circuit propagate in the forward/backward direction, crossing the flip-flop boundaries, and hence help to identify global relations throughout the sequential circuit. This process of adding sequential implications as two-literal clauses is termed as Sequential Learning.

3.2 The Efficacy of Sequential Implications

Greater sequential implication depth allows more learning, but requires additional computational effort. Due to reasons cited in section 2.2, we make this sequential implication depth user-specified. In our experiments, a sequential implication depth of $2’$ (time-frames ranging from -2 to +2) was sufficient to provide a large amount of learning. In Figure 5 below, we give the representation of sequential implications in a 5 time-frame unrolled sequential circuit. These implications not only help us to identify relations throughout the combinational portion of the sequential
circuit (of type $a \rightarrow b$), but also the relations spanning multiple time frames (of type $w \rightarrow x$ and $y \rightarrow z$) which play a very significant role. We see that node $a \rightarrow node b$ is an implication of depth '0'. This is replicated as node $a_0 \rightarrow node b_0$, node $a_1 \rightarrow node b_1$, node $a_2 \rightarrow node b_2$ and so on for each of the time-frames, starting from time-frame '0' till time-frame 'k-1'. The sequential implications crossing time-frame boundaries are replicated successively from time-frame '0' to time-frame 'k-1' as per their edge weights. As seen above, node $y \rightarrow node z$ in time-frame '2' (implication of depth '2'). This is replicated as node $y_2 \rightarrow node z_0$, node $y_2 \rightarrow node z_1$ and so on in the unrolled circuit; the difference in the times-frames of nodes $z_0$ and $y_2$ and also $z_2$ and $y_2$ being '2-2'. Again, similar replication is done for the implication, node $w \rightarrow node x$ in time-frame '1'. Note that these sequential implications crossing time-frames ($w \rightarrow x$ and $y \rightarrow z$ in the figure) only need to be computed once in our approach, and the subsequent replication is applied automatically. This is different from combinational learning on the unrolled circuit, where each relation crossing time-frame boundary (each of $w_i \rightarrow x_i$ and $y_i \rightarrow z_i$) is regarded as a distinct relation and is learned individually. These sequential implication relations, especially the ones crossing time-frame boundaries, help to induce meaningful structural information throughout the unrolled sequential circuit, which in turn enhances the SAT-solver performance.

3.3 The Algorithm

1. Construct the CNF database for the transformed unrolled sequential circuit, with monitor circuit [14, 17], as per the property to be verified.
2. Build the Sequential Implication Graph for the circuit under verification for a user defined implication depth.
3. Formulate the two-literals clauses (spanning multiple time-frames) corresponding to the indirect implications learnt.
4. Replicate these two-literals clauses successively in each of the time-frames as per their edge weights (discussed in section 3.2).
5. Appended these clauses to the existing CNF database.
6. Ask the SAT solver to satisfy the monitor circuit O/P to logic '1'.

4. Experimental Results

The proposed concept was implemented in C++ in a preprocessing engine called SIMP2C (Sequential Implications to Clauses). The experiments were run on Pentium-4, 1.8GHz machine, with 512Mb of RAM and Linux as the operating system. Arbitrary safety properties of the form EF(s) (where s is a complete or partial state) are generated and verified. Liveness properties of the form EG(s) can also be verified using our method. However, because most liveness properties are extremely easy to check for, in the bounded length 'k' (since the SAT instance is more constrained), they are omitted in this work. ZChaff[1] and BerkMin[4] are used as the SAT solver for all instances. The results for the effectiveness of our approach are shown in Table 1. The execution times reported are the average on a set of 10 random difficult safety properties for each circuit. The easy properties are quickly solved by the SAT-solver and don’t require any preprocessing. It is the difficult instances where our novel technique yields a significant speed-up. These properties include both satisfiable and unsatisfiable instances. In Table 1, for each of the sequential circuits, we give the average execution time taken by our preprocessing engine SIMP2C; the number in the parenthesis indicates the sequential implication depth (seqImp_depth). We then give the average execution time taken by ZChaff and BerkMin without any preprocessing and the combined time taken by SIMP2C + ZChaff and SIMP2C + BerkMin. We also report the speedup obtained by using our preprocessing over the conventional SAT-based approach.

According to Table 1, the proposed method (SIMP2C + Sat-solver) achieved speedups ranging from 1.24x for s4863 to 148.98x for s9234.1, irrespective of the underlying SAT-solver. The vast range in speedup is due to the fact that the execution time is both circuit and property dependent. Some properties can be quickly solved by SAT-solver alone whereas some are computationally expensive. For instance, the random properties generated for circuit s13207.1 were all solved very quickly with ZChaff (alone) taking average time of 20.68 secs. After the non-trivial implication clauses were added using SIMP2C, the average time taken by ZChaff reduced to 7.99 secs (not shown), indicating a good amount of search space pruning. However, the total time taken by SIMP2C + ZChaff was 16.41 secs, resulting only in a small speedup of 26%. On the other hand, for circuit s9234.1, the average execution time to solve a set of 10 difficult random safety properties was reduced from 1445.2 secs (BerkMin alone) to 9.7 secs (SIMP2C+BerkMin), thereby achieving a speedup of 148.98x.

The time taken by our pre-processing engine SIMP2C is very low, ranging from 0.14 seconds to 25.12 seconds, making our method very attractive; little additional effort is sufficient to reduce SAT solution complexity.

4.1 Effect of increasing the Bounded Length ‘k’

We observed that in the conventional SAT-based BMC method, increasing the time-frame bound ‘k’ to verify the given safety property causes an exponential increase in the execution time. This is shown in Figure 6 for circuits s510 and s38417. The properties being checked here are Unsatisfiable. From Figure 6, we see that our approach (SIMP2C + ZChaff) is able to reduce this exponential execution time to almost linear time.

4.2 Effect of increasing the Sequential Implication Depth

Table 2 below shows the effect of increasing the sequential implication depth on execution time. Greater sequential implication depth allows for greater sequential learning, but at an increased cost. From Table 2, we see that for circuit s382, increasing learning from seqImp_depth '0' (combinational learning) to seqImp_depth '4' decreases the average execution time from 16.28 to 13.23 seconds. For s35932, the increase in learning from seqImp_depth '0' to seqImp_depth '1' resulted in increased speedup ratio from 4.37x to 12.25x. However, increasing sequenti-
al learning further to seqImp_depth of ‘2’ resulted in SIMP2C taking more time and hence speedup ratio reduced by a small amount from 12.25x to 10.77x. For s1512 speedup ratio increased from 43.6x to 47.96x. Note that similar results were obtained with BerkMin as well, but due to space constraints have not been shown.

Table 2. Effect of increasing implication depth on SAT-solver Performance

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>k</th>
<th>ZChaff (secs.)</th>
<th>SIMP2C (secs.)</th>
<th>ZChaff + SIMP2C (secs.)</th>
<th>Speedup</th>
<th>BerkMin (secs.)</th>
<th>SIMP2C + BerkMin (secs.)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>s298</td>
<td>500</td>
<td>29.21</td>
<td>0.11(1)</td>
<td>4.78</td>
<td>6.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s382</td>
<td>410</td>
<td>54.39</td>
<td>0.08(0)</td>
<td>26.16</td>
<td>2.97</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1512</td>
<td>190</td>
<td>857.12</td>
<td>0.6(1)</td>
<td>9.67</td>
<td>43.60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s35932</td>
<td>65</td>
<td>858.05</td>
<td>0.78(2)</td>
<td>17.87</td>
<td>47.96</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. Conclusion

A novel technique of enhancing the SAT-based Bounded Model Checking has been presented. Non-trivial implications resulting from constructing the sequential implication graph are quickly computed for the circuit-under-verification and converted into two-literal clauses. These clauses (spanning multiple time-frames) are computed without unrolling the circuit, making our method very cost effective. When the resulting clauses are added to the existing CNF, they serve as constraints and help the SAT solver in the search process. Experimental Results for difficult random safety properties on ISCAS’89 benchmark circuits showed that we achieved speedups of up to 148.98x over the conventional SAT-based Bounded Model Checking approach.

References