Empirical Evaluation of IC3-Based Model Checking Techniques on Verilog RTL Designs

Aman Goel  
University of Michigan, Ann Arbor  
amangoel@umich.edu

Karem Sakallah  
University of Michigan, Ann Arbor  
karem@umich.edu

Abstract—IC3-based algorithms have emerged as effective scalable approaches for hardware model checking. In this paper we evaluate six implementations of IC3-based model checkers on a diverse set of publicly-available and proprietary industrial Verilog RTL designs. Four of the six verifiers we examined operate at the bit level and two employ abstraction to take advantage of word-level RTL semantics. Overall, the word-level verifier employing data abstraction outperformed the others, especially on the large industrial designs. The analysis helped us identify several key insights on the techniques underlying these tools, their strengths and weaknesses, differences and commonalities, and opportunities for improvement.

I. INTRODUCTION
IC3 [1] (also referred to as PDR [2]) has emerged as an effective model checking (MC) algorithm that uses incremental SAT solving to perform property-directed approximate reachability analysis. Model checkers using IC3-based techniques at the bit level have shown exceptional performance in hardware model checking competitions [3]. However, bit-level analysis can still face scalability issues, particularly when applied to large word-level designs involving wide and complex data operations. Several approaches have been proposed to extend IC3-style algorithms for better scalability [4]–[9]. These approaches suggest different abstraction refinement strategies to reduce the burden on the reasoning engines. Some of these approaches (like [4], [5]) exploit high-level information to perform IC3 at the word level, and have shown impressive results. These techniques replace bit-level reasoning using SAT solvers with word-level reasoning using SMT solvers [10].

In contrast to earlier studies where only bit-level problems were analyzed (e.g. [3], [11]), this paper compares the behavior of different IC3-based techniques on Verilog RTL designs. We performed a rigorous evaluation on 535 Verilog RTL instances including a variety of open-source designs and real-world industrial problems. Our experiments resulted in several key findings, most notably that word-level IC3 solvers can handle designs that are beyond the reach of bit-level solvers.

II. BACKGROUND
Model checking can be defined as, given a transition system $T$ (defined by a transition relation $T$ and a set of initial states $I$), check whether it meets a given property $P$, and, if not, produce a counterexample demonstrating how $T$ violates $P$. Our focus in this paper will be on safety properties defined on finite transition systems.

The reader is referred to [1], [2] for a detailed description on ideas underlying IC3. Let $R_k$ represent the set of states reachable from the initial states within $k$ steps ($k \geq 0$). The property holds if $R_\infty \subseteq P$. Finding the exact set of reachable states is intractable in practical systems. Instead, IC3 iteratively derives overapproximations of $R_k$ (called frames $F_k$) such that $F_0 = I$ and $R_k \subseteq F_k$. In its simplest form, the procedure requires two major steps to prove a property:

- **Initiation** - Prove that the property is not trivially violated i.e. $I \subseteq P$ and $I \land T \land \neg P'$ is unsatisfiable (using primes to denote next states).
- **Consecution** - Derive $F_1 \ldots F_n$ such that $F_i \subseteq P$ and $F_{i-1} \subseteq F_i$ ($i \in \{1, \ldots, n\}$) till two frames converge (i.e. $F_j = F_{j+1}$ for some $j \in \{1, \ldots, n\}$). This is done by iteratively deriving restrictions on frames using 1-step backward reachability queries “SAT? [ $F_k \land T \land c$ ]” that check if a state in cube $c$ is reachable from $F_k$ in one step.

III. REVIEW OF IC3-BASED TECHNIQUES
Several techniques have emerged that extend bit-level IC3 engines to offer better performance. The authors in [7] use lazy abstraction, the authors of [8] suggest using uninterpreted functions (UFs) to abstract away expensive data operations, while the authors in [9] use unconstrained new primary inputs to abstract away parts of the system. All these approaches use bit-level IC3 as the core reachability engine.

Certain approaches suggest deploying IC3 at the word level using SMT solvers. The authors in [4] use implicit predicate abstraction, perform word-level IC3, and refine the abstraction by adding more predicates. The Averroes system [5] demonstrated the effectiveness of applying data abstraction for word-level verification. Averroes 2 [12] is a major upgrade that introduces numerous enhancements in the abstraction, generalization, and refinement stages, is completely incremental in its word-level reasoning, and produces compact word-level inductive invariants when the property holds. These approaches exploit the power of SMT solvers in different ways to perform word-level clause learning, and offer better scalability than techniques that rely on bit-level IC3.

IV. EXPERIMENTAL SETUP
We analyzed 535 safety checking problems (Verilog RTL files with SystemVerilog assertions (SVA)) classified as follows:
• industry: a set of 370 problems collected from industrial collaborators1. Their code sizes range from 109 to 22065 lines and number of flip-flops from 6 to 7249.
• crafted: a set of 24 simple problems synthetically created.

The six tools we evaluated in this experiment were:
• From ABC version 1.01 [16]:
  – pdr: pdr is one of the best implementations of the IC3 algorithm at the bit level.
  – dprove: dprove employs a pre-processing stage using a portfolio of techniques (bounded MC, retiming, simulation, interpolation, etc.) with carefully-tuned heuristics to quickly solve/reduce the problem. If still unsolved, dprove invokes pdr on the reduced problem.
  – pdr-nct: the -nct flags configure pdr to use better generalization [17] and to enable localization abstraction [6].
• From nuXmv version 1.1.1 [18]:
  – nuXmv-ic3: this is the bit-level IC3 implementation in nuXmv based on simplic3 [11]. It starts with a pre-processing step (latch equivalency and temporal decomposition) followed by a state-of-the-art implementation of IC3.
  – nuXmv-ic3ia: this is the word-level IC3 implementation in nuXmv using implicit predicate abstraction [4].
• avr: Averrees 2 [12] is a word-level IC3 implementation using data abstraction and is particularly suited for verifying control-centric properties.

We set up the experiment as shown in Fig. 1. The Verilog designs and SVA are parsed by yosys [19] which removes any hierarchy and produces flat RTL. For the nuXmv tools and avr, this flat RTL is syntactically translated into the equivalent input formats used by these tools. For the ABC tools, yosys synthesizes the RTL to a bit-level implementation.

All experiments were conducted on a cluster of 163 2.5 GHz Intel Xeon E5-2680v3 processors (cores) running 64-bit Linux. Each run was given exclusive access to a single core, with a memory limit of 16 GB and a time limit of 5 hours.

V. EXPERIMENTAL ANALYSIS

Space limitations preclude inclusion of all data visualizations generated by this experiment. The raw data, including details on the benchmarks, are available in [20].

A. Aggregate results

Overall, each instance in the suite of 535 problems was successfully solved by at least one tool. A total of 483 problems were proved safe and 52 problems proved unsafe. Table I and Fig. 2 lead to the following observations:
• avr solved more problems than any other solver. It especially dominated in the industry category, solving all but 2 with 55 of them uniquely solved. dprove marginally dominated in the opensource category.
• Implicit predicate abstraction helped nuXmv-ic3ia solve 4 more problems from the industry category than its bit-level version nuXmv-ic3.
• In general, the bit-level IC3 implementations in the ABC tools performed significantly better than nuXmv-ic3.

B. Runtime comparison

The scatter plots in Fig. 3 compare avr’s runtime with the runtimes of the other tools leading to the following observations:
• For the vast majority of problems in the industry category, avr is able to solve them much faster than the other tools. These problems have wide data paths and expensive data operations that cause the other tools to run slower and time out in many cases, while the data abstraction used by avr is particularly effective.
• avr is not as effective in the opensource category, clearly under performing the ABC tools. These problems involve data-centric properties that are ill-suited for avr’s data abstraction and cause avr to perform many expensive data refinements to repair its initial abstraction.

Fig. 4 shows additional runtime comparisons that shed further light on the behavior of these tools as follows:
• Fig. 4.a shows that the extra optimizations in pdr-nct’s do not result in a discernible improvement over pdr.
• Fig. 4.b confirms our earlier observation that the ABC tools, represented by pdr, significantly outperform nuXmv-ic3 in runtime on almost all categories.
• Fig. 4.c shows that nuXmv-ic3ia is usually faster than nuXmv-ic3, especially for the industry category. This seems to confirm that abstraction is critical for large-scale problems that tend to tax the capacity of bit-level analysis.
• Fig. 4.d shows that for problems that can be solved during the pre-processing stage of dprove, the runtime of pdr without pre-processing can sometimes be faster! Still, pre-processing does help, especially for the industry problems.
• Fig. 4.e provides further evidence of the importance of pre-processing. Specifically, in most cases where dprove is faster than avr, it is because the problem is solved in the pre-processing stage. This suggests that word-level pre-processing techniques similar to the bit-level techniques used in dprove may further help scale abstraction based tools such as nuXmv-ic3ia and avr.

1We obtained these designs under non-disclosure agreements and, unfortunately, cannot make them publicly available.
E. Bit-level versus word-level verification

Figs. 6.c-e compare different IC3 statistics between pdr (bit-level IC3 engine) and avr (word-level IC3 with data abstraction). These plots indicate the following:

- In many cases, pdr makes orders of magnitude more counterexamples to induction (CTI) checks than avr and learns many more clauses before solving the problem (Fig. 6.c-d). This affirms why avr requires many fewer solver calls compared to pdr (Fig. 5.a), and shows the strengths of using a word-level IC3 procedure that can exploit the word-level semantics to learn strong clauses.
- Some crafted and opensource problems require avr to perform more CTI checks and clause learning compared to pdr due to avr’s data abstraction being ineffective when the property is data-dependent.
- Fig. 6.e compares the number of clauses in the inductive invariant produced by the two tools (for cases where the property is true). Again due to word-level clause learning, avr learns stronger word-level inductive invariants with many fewer clauses compared to pdr.

VI. CONCLUSIONS

Our goal in this preliminary study was to better understand the landscape of IC3-based MC techniques by comparing not just bit-level engines, but also word-level techniques to shed light on their suitability for various types of MC problems. Word-level IC3 engines add a layer of abstraction to lift the reasoning above the bit level, and takes advantage of the word-level information to potentially scale to much larger designs.

ACKNOWLEDGMENT

We would like to thank Alberto Griggio for providing a custom version of nuXmv with detailed statistics output.

REFERENCES

TABLE I: Number of problems solved by each tool
TO: timed out, MO: out of memory, Unique: solved uniquely (not solved by others), IN: industry, OS: opensource, CR: crafted, B: bit-level, W: word-level

<table>
<thead>
<tr>
<th>Tool</th>
<th>Solved (555)</th>
<th>TO</th>
<th>Error / MO</th>
<th>Unique</th>
<th>IN (370)</th>
<th>OS (141)</th>
<th>CR (24)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdr</td>
<td>466</td>
<td>69</td>
<td>0</td>
<td>1</td>
<td>308</td>
<td>137</td>
<td>21</td>
</tr>
<tr>
<td>dprove</td>
<td>477</td>
<td>57</td>
<td>1</td>
<td>3</td>
<td>315</td>
<td>138</td>
<td>24</td>
</tr>
<tr>
<td>pdr-nct</td>
<td>466</td>
<td>68</td>
<td>1</td>
<td>1</td>
<td>308</td>
<td>137</td>
<td>21</td>
</tr>
<tr>
<td>nuxmv-ic3</td>
<td>385</td>
<td>133</td>
<td>17</td>
<td>0</td>
<td>228</td>
<td>134</td>
<td>23</td>
</tr>
<tr>
<td>nuxmv-ic3ia</td>
<td>389</td>
<td>92</td>
<td>54</td>
<td>0</td>
<td>232</td>
<td>133</td>
<td>24</td>
</tr>
<tr>
<td>avr</td>
<td>526</td>
<td>0</td>
<td>9</td>
<td>55</td>
<td>368</td>
<td>134</td>
<td>24</td>
</tr>
</tbody>
</table>

Fig. 2: Survival plot comparing the number of problems solved versus runtime

Fig. 3: avr runtime comparisons. avr’s times are better (resp. worse) above (resp. below) the diagonal.

Fig. 4: Additional runtime comparisons. In parts d and e, instances solved (resp. unsolved) during the pre-processing stage of dprove are colored brown (resp. blue).

Fig. 5: Comparison of number of solver calls (SAT solver calls for pdr & nuxmv-ic3, SMT solver calls for nuxmv-ic3ia & avr)

Fig. 6: Comparison of number of refinement iterations (parts a and b) and IC3 statistics (parts c to e) w.r.t. avr

All plots exclude runs in which a tool reported an error or ran out of memory, and all runtimes refer to CPU time in seconds.