DiVer: SAT-Based Model Checking Platform for Verifying Large Scale Systems

Malay K Ganai, Aarti Gupta, and Pranav Ashar

NEC Laboratories America, Princeton, NJ USA 08540
{malay, agupta, ashar}@nec-labs.com

Abstract. We present a SAT-based model checking platform (*DiVer*) based on robust and scalable algorithms that are tightly integrated for verifying large scale industry designs. *DiVer* houses various SAT-based engines each targeting capacity and performance issues inherent in verifying large designs. The engines with their respective roles are as follows: Bounded Model Checking (BMC) and Distributed BMC over a network of workstations for falsification, Proof-based Iterative Abstraction (PBIA) for model reduction, SAT-based Unbounded Model Checking and Induction for proofs, Efficient Memory Modeling (EMM) and its combination with PBIA in BMC for verifying embedded memory systems with multiple memories (with multiple ports and arbitrary initial state). Using several industrial case studies, we describe the interplay of these engines highlighting their contribution at each step of verification. *DiVer* has matured over 3 years and is being used extensively in several industry settings. Due to an efficient and flexible infrastructure, it provides a very productive verification environment for research and development.

1 Introduction

Verifying modern designs requires robust and scalable approaches in order to meet more-demanding time-to-market requirements. Compared to symbolic model checking [1, 2] based on Binary Decision Diagrams [3], SAT-based model checking techniques [4-17] have been able to scale and perform well due to the many recent advances in DPLL-style SAT solvers [18-20]. We present a SAT-based model checking platform (*DiVer*) based on robust and scalable algorithms [5-7, 11-17, 20, 21] that are tightly integrated for verifying large scale industry designs. We present a brief overview of *DiVer* with its engines each targeting capacity and performance issues inherent in verifying large designs. Using several industrial case studies, we describe the interplay of these engines highlighting their contribution at each step of verification.

2 Tool Overview

DiVer uses an efficient circuit representation with on-the-fly simplification algorithms [18, 21], and an incremental hybrid SAT solver [20] that combines the strengths of circuit-based and CNF-based solvers seamlessly. *DiVer* houses the following

N. Halbwachs and L. Zuck (Eds.): TACAS 2005, LNCS 3440, pp. 575 – 580, 2005.

SAT-based engines, each geared towards verifying large systems: bounded model checking (BMC) [7] and distributed BMC (d-BMC) over a network of workstations (NOW) [12] for falsification, proof-based iterative abstraction (PBIA) for model reduction [13], SAT-based unbounded model checking (UMC) [15] and induction for proofs [5, 11], Efficient Memory Modeling (EMM) [14] and its combination with PBIA in BMC for verifying embedded memory systems with multiple memories (with multiple ports and arbitrary initial state) and to discover irrelevant memories and ports for proving property correctness [17].

DiVer has matured over 3 years and is being used extensively by the designers in our company. Because of an efficient and flexible infrastructure, it provides a very productive environment for research and development. In this paper we provide useful pointers to the various research efforts, and describe how they fit well together. We present the tool as a "wheel of verification engines" in Figure 1(a). We show the interplay of these engines in verification flows for designs with and without embedded memory in Figures 1(b-c). In the following, we briefly describe various engines:

Internal Representation and Hybrid SAT Solver: The verification model is represented efficiently as a circuit graph with 2-input OR/INVERTER gates, using an on-the-fly multi-level functional hashing algorithm [18, 21] that detects and removes structural and local redundancies. We use this graph to represent the transition relation, unrolled time frames, and the set of enumerated states. For Boolean reasoning, we combine [20] the strengths of circuit-based [18] and CNF-based SAT solvers [19] with incremental SAT solving capabilities [7]. The solver uses deduction and diagnostics engines efficiently on the hybrid Boolean representation, i.e., circuit graph and CNF. The decision engine also benefits from both circuit and CNF based heuristics.

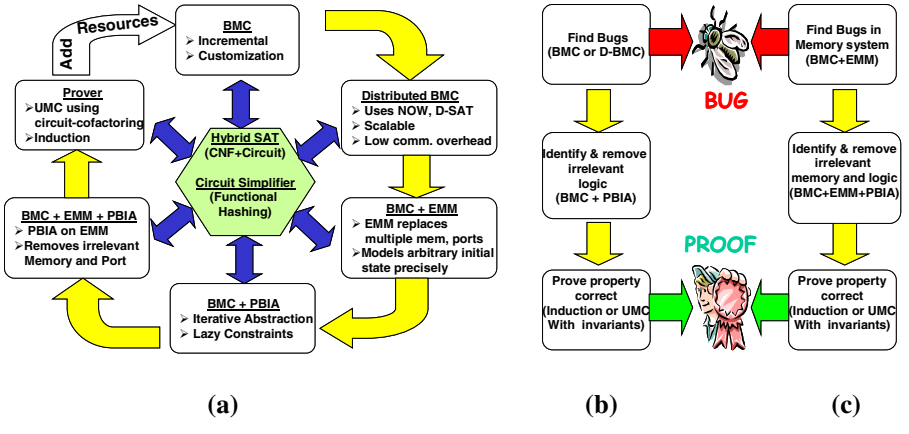

Fig. 1. *DiVer* Overview (a), Verification without (b) / with (c) embedded memory

BMC: Our SAT-based BMC engine uses the simplified circuit graph to represent unrolled time frames and the hybrid SAT solver to falsify the given the property. For commonly occurring properties, we use *customized translations* of LTL properties that involve partitioning the problem and using incremental model checking [7].

d-BMC: Our d-BMC engine over a network of workstations [12] overcomes the memory limitation of a single server to provide a scalable approach for carrying out deeper search on memory-intensive designs. We achieve a) *scalability* by not keeping the entire problem data on a single processor, and b) *low communication overhead* by making each process cognizant of the partition topology while communicating; thereby, reducing the process's receiving buffer with unwanted information.

BMC+EMM: Our EMM approach [14, 17] augments BMC to handle embedded memory systems (with multiple read, write ports) without explicitly modeling each memory bit, by capturing the memory data forwarding semantics efficiently using exclusivity constraints. An arbitrary initial state of the memory is modeled precisely using constraints on the new symbolic variables introduced [17].

BMC+PBIA: Our PBIA technique [13] generates a *property-preserving* abstract model (up to a certain depth) by a) obtaining a set of latch reasons (LR) involved in the unsatisfiability proof of a SAT problem in BMC, and b) by abstracting away all latches *not in this set* as pseudo-primary inputs. We further reduce the model size by using the abstraction *iteratively* and using *lazy constraints* [16].

BMC+EMM+PBIA: We combine the EMM and PBIA techniques [17] to identify fewer memory modules and ports that need to be modeled; thereby reducing the model size, and verification problem complexity. If no latch corresponding to the control logic for a memory module or port is in the LR set (obtained by PBIA), we do not add the EMM constraints for that memory module or port during BMC.

UMC: Our UMC approach [15] improves the SAT-based blocking clause approach [8] by several orders of magnitude, by using *circuit-based cofactoring* to capture a larger set of new states per enumeration, and representing them efficiently using a simplified circuit graph. The method is combined with inductive invariants, e.g., reachability constraints [11] for faster fixed-point computations.

3 Selected Case Studies

Using selected case studies from the industry, we demonstrate the role of various engines at each step of the verification. Note that without the interplay of the engines we could not have verified any of these designs. The first two case studies use the verification flow shown in Figure 1(b) and the next two use that shown in Figure 1(c). All experiments were performed on a server with 2.8 GHz Xeon processors with 4GB running Red Hat Linux 7.2.

Industry Design I: The design has 13K flip-flops (FFs), ~0.5M gates in the cone of influence of a safety property. Using BMC, we showed there was no witness up to depth 120 (in 1643s) before we run out of memory. Using d-BMC, we showed no witness up to depth 323 (in 8643s) using 5 workstations (configured as 1 Master and 4 Clients and connected with 1Gps Ethernet LAN), with a communication overhead of 30% and scalability factor of 0.1 (i.e, potentially we could do a 10 times deeper

analysis than that on the single server.) We hypothesized that the property is correct. We used the PBIA engine to obtain an abstract model with 71 FFs and ~1K gates in 6 iterations taking ~1200s. With UMC, we proved the property correct taking ~2400s.

Industry Design II: The design with environmental constraints has 3.3K FFs and ~28K gates for a safety property. Using BMC, we showed there was no witness up to depth 113 (in ~3hr, 720MB). Again, we hypothesized the correctness of the property. We used PBIA to obtain an abstract model A1 with 163 FFs and ~2K gates in 4 iterations taking 9000s. Without the environmental constraints, the abstract model A2 has only 66 FFs and ~1K gates. We computed a reachability invariant [11] on the A2 model (in ~4s) and used this with UMC on the A1 model to obtain a proof in ~60s.

Industry Design III: The design has 756 FFs (excluding the memory registers), and ~15K gates. It has two memory modules, both having address width, AW = 10 and data width, DW = 8. Each module has 1 write and 1 read port, with the memory state initialized to 0. There are 216 reachabality properties. Using BMC+EMM, we found witnesses for 206 of the 216 properties, taking ~400s and 50Mb. The maximum depth over all witnesses was 51. Using explicit modeling, we required 20540s (~6hrs) and 912Mb to find witnesses for all 206 properties. By using induction with BMC+EMM, we proved the remaining 10 properties in <1s (25 s for explicit modeling).

Quicksort: The implementation has two memory modules: an un-initialized array with AW=10, DW=32, 1 read and 1 write port and an un-initialized stack (for recursive function calls) with AW=10, DW=24, 1 read and 1 write port. The design has 167 FFs (excluding memory registers), and ~9K gates for array size 5. The property states that after return from a recursive call, the program counter should go to a recursive call on the right partition or return to the parent on the recursion stack. Using BMC+EMM+PBIA, we reduced the model to 91 FFs and ~3K gates, and also identified the array module as irrelevant for this property. On this reduced model we proved correctness using forward induction (proof diameter = 59) in 2.3Ks, 116MB. (Without the abstraction, the induction proof in BMC+EMM takes ~5Ks, 400MB. For explicit model, however, we could obtain neither a proof nor an abstract model in 3 hours).

References

[1] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*: MIT Press, 1999.
[2] K. L. McMillan, Symbolic Model Checking: An Approach to the State Explosion Problem: Kluwer Academic Publishers, 1993.
[3] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35(8), pp. 677-691, 1986.
[4] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Proceedings of DAC*, 1999.
[5] M. Sheeran, S. Singh, and G. Stalmarck, "Checking Safety Properties using Induction and a SAT Solver," in *Proceedings of FMCAD*, 2000.
[6] M. Ganai and A. Aziz, "Improved SAT-based Bounded Reachability Analysis," in *Proceedings of VLSI Design Conference*, 2002.

[7] M. Ganai, L. Zhang, A. Gupta, P. Ashar, and Z. Yang, "Efficient Approaches for Bounded Model Checking," *US Patent Application 2003-0225552*, Filed on May 30, 2002.
[8] K. McMillan, "Applying SAT methods in Unbounded Symbolic Model Checking," in *Computer-Aided Verification*, 2002.
[9] K. McMillan and N. Amla, "Automatic Abstraction without Counterexamples," in *Proceedings of TACAS*, 2003.
[10] K. McMillan, "Interpolation and SAT-based Model Checking," in *Proceedings of CAV*, 2003.
[11] A. Gupta, M. Ganai, C. Wang, Z. Yang, and P. Ashar, "Abstraction and Bdds Complement SAT-Based BMC in DiVer," in *Proceedings of CAV*, 2003.
[12] M. Ganai, A. Gupta, and P. Ashar, "Distributed SAT and Distributed Bounded Model Checking," in *Proceedings of CHARME*, 2003.
[13] A. Gupta, M. Ganai, P. Ashar, and Z. Yang, "Iterative Abstraction using SAT-based BMC with Proof Analysis," in *Proceedings of ICCAD*, 2003.
[14] M. Ganai, A. Gupta, and P. Ashar, "Efficient Modeling of Embedded Memories in Bounded Model Checking," in *Proceedings of CAV*, 2004.
[15] M. Ganai, A. Gupta, and P. Ashar, "Efficient SAT-based Unbounded Model Checking Using Circuit Cofactoring," in *Proceedings of ICCAD*, 2004.
[16] A. Gupta, M. Ganai, and P. Ashar, "Lazy Constraints and SAT Heuristics for Proof-based Abstraction," in *Proceedings of VLSI Design*, 2005.
[17] M. Ganai, A. Gupta, and P. Ashar, "Verification of Embedded Memory Systems using Efficient Memory Modeling," in *Proceedings of DATE, 2005.*
[18] A. Kuehlmann, M. Ganai, and V. Paruthi, "Circuit-based Boolean Reasoning," in *Proceedings of DAC*, 2001.
[19] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," in *Proceedings of DAC*, 2001.
[20] M. Ganai, L. Zhang, P. Ashar, and A. Gupta, "Combining Strengths of Circuit-based and CNF-based Algorithms for a High Performance SAT Solver," in *Proceedings of DAC*, 2002.
[21] M. Ganai and A. Kuehlmann, "On-the-fly Compression of Logical Circuits," in *Proceedings of IWLS*, 2000.

Appendix

DiVer is the core of the verification system as shown in the Figure 2. The tool has the ability to handle several industry design features including multiple clocks, phase, and gated clocks with arbitrary frequency ratios, embedded memories with multiple read and write ports, environmental and fairness constraints. Current input spec is LTL, but support for other specification like PSL is on the way. *DiVer* is used extensively by the designers within the company who are not verification experts. We have often received feedbacks that tool has been able to discover hard to detect bugs that simulations could not have found, or could have found at very high cost in terms of resources. As of now, the tool is not available for free download.

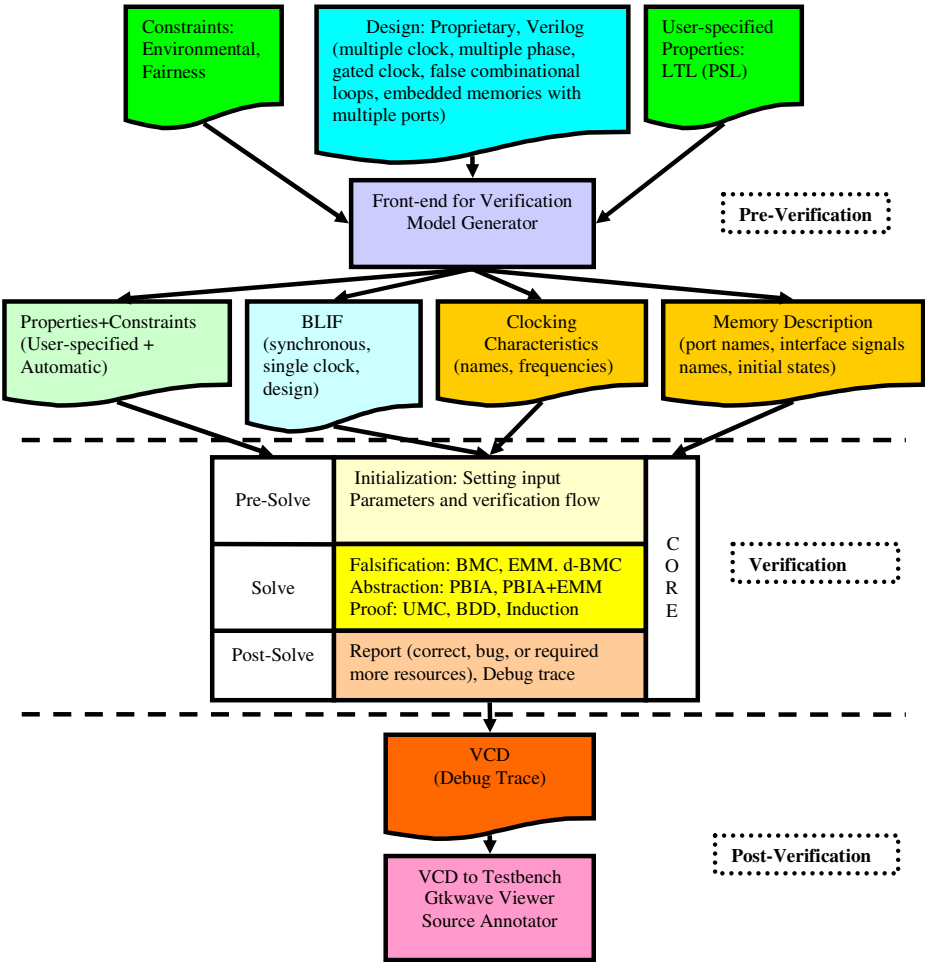

Fig. 2. Overview of Verification System