A Memory-Upscaled Boolean Satisfiability Solver for Complex On-Chip Self-Verification Tasks

Buse Ustaoglu* Sebastian Huhn*† Rolf Drechsler*†

*Cyber-Physical Systems, DFKI GmbH
28359 Bremen, Germany
{buse.ustaoglu}@dfki.de

†University of Bremen, Germany
{huhn,drechsler}@uni-bremen.de

***Abstract*—The verification gap is ever-widening with the increased complexity of electronic circuit designs. A novel approach to tackle this arising challenge concerns the *self-verification*, which paves the way for continuing the systems' verification tasks after their deployment. For realizing such a self-verification capability, a verification package is required on-chip, which merges verification techniques – like Bounded-Model Checking – and application-specific co-processors – like hardware-based *Boolean Satisfiability* (SAT)-solvers – within the circuit-under-verification together. Previous works have proposed compact *Hardware Boolean Satisfiability Solvers* (HW SAT-solver) to solve a-priori given arbitrary SAT-instances; however, the maximum processable instance size is bounded and, hence, their applicability on complex verification tasks is limited. This work proposes a novel memory-upscaling scheme for lightweight HW SAT-solvers to cope with even large and industrial-relevant instance sizes. More precisely, external memory resources are orchestrated, which are seamlessly integrated into the HW SAT-solver by implementing a suitable interface module and protocol. Experiments clearly demonstrate the effectiveness of this work.**

I. Introduction

Electronic system designs occur in almost all parts of our daily life, including safety-critical applications. Consequently, it is of utmost importance that the resulting systems operate without any failure. To ensure the correctness of these systems, verification forms a crucial step checking whether the functionality of the design meets its specification or not. Due to the challenging time-to-market, state-of-the-art designs are deployed without being fully verified [1]. The concept of self-verification has been proposed to tackle the ever-widening verification gap [2], [3]. It is meant to be performed in addition to existing techniques like simulation- and emulation-based functional verification and formal verification [4]. A lightweight verification package is required in order to apply self-verification within the later in-field application.

Previous works have been proposed that develop verification packages by applying verification techniques in hardware, which utilize the *Boolean Satisfiability* (SAT)-problem. Consequently, HW SAT-solvers must be realized in hardware, e.g., by using the rapid prototyping *Field Programmable Gate Array* (FPGA) technology. In work [5], [6], a memory model has been proposed that stores a SAT-instance entirely in the fast on-chip memory *Block Random Access Memory* (BRAM) resources of an FPGA and further BRAM resources are utilized for internal algorithmic purposes. Since the available number of BRAMs is bounded, the maximum processable SAT-instances is limited and, hence, restricts the applicable self-verification tasks.

This work significantly improves available HW SAT-solvers to, in the end, take advantage of self-verification for even large and industrial-relevant instance sizes. More precisely, large external memory resources are clever orchestrated, which allows for storing even large SAT-instance while preserving the lightweight characteristic of the HW SAT-solvers and avoid any large adverse impact on the solvers' latency.

II. HW SAT-solvers for self-verification

The SAT-problem asks the question whether an assignment to all Boolean variables of a given Boolean function exists such that the function evaluates to true. If this is the case, the function is said to be *satisfiable* (sat) and, otherwise, *unsatisfiable* (unsat). This evaluation is automatically conducted by a SAT-solver, which typically processes Boolean functions, given in a *Conjunctive Normal Form* (CNF). A CNF consists of a conjunction of clauses, whereby a clause is a disjunction of literals and each literal is a Boolean variable or its negation. One difficult challenge when designing HW SAT-solvers concerns the effective storage of the SAT-instance in the available hardware. Specific criteria have to be considered like the maximum number of variables/clauses and more general aspects like the fact whether arbitrary instances can be processed without enforcing a specific format like 3-CNF. To unleash the full potential of self-verification, large arbitrary SAT-instances must be processed within a reasonable run-time.

The previously presented HW SAT-solvers [5], [6] implement a finite state machine to control the operation, which have been inspired by the *Davis-Putnam-Logemann-Loveland* (DPLL) algorithm [7]. In particular, work [6] enhances the HW SAT-solver by introducing clause learning techniques, which follows a *Conflict-Driven Clause Learning* (CDCL) scheme [8].

The HW SAT-solver principle of [5] is briefly described as follows: At first, the complete SAT-instance is transferred to the *main memory* after the solver's initialization. Certain meta information – like the number of literals in each clause – are determined and stored into the *Clause Position Memory* (CPM). If one clause exists that holds only one literal (unary clause), an implication is derived and, otherwise, a decision is being made. The current decision level is incremented by 1 with every decision. Then, the assignment is propagated to all clauses. Subsequently, the regular DPLL-search procedure is conducted, as implemented on hardware, which is not described in detail due to the page limitation. More information about the underlying DPLL-search procedure and the criteria of termination is given in [7].

The conflict resolution of [6] performs clause learning, meaning that the literals of the clause in which a conflict occurs are stored in *Reason Literal Memory* (RLM). The implication clauses are accessed via the *Implication Graph Memory* (IGM) that stores the clause information of the implied literal at each implication. For more details about the implementation, consider the works [5], [6].

III. Memory-Upscaled HW SAT-solver

The fastest on-chip memory within an FPGA is BRAM, which is, in turn, very expensive. Thus, the amount of available

BRAM within state-of-the-art FPGAs is strictly limited. In contrast to BRAM, an external memory provides a large storage capacity at relatively low costs. However, accessing such an external memory requires significant more cycles than on-chip memories. With respect to the intended HW SAT-solver application, using external memory naively would introduce a latency such that the SAT-instance could not be processed within reasonable run-time. Consequently, it is of utmost importance that a suitable trade-off between the gained performance – in the sense of solving run-time – and the resulting (hardware) overhead is identified.

This work proposes a clever memory-upscaling scheme such that the SAT-instance is stored in the external memory of high capacity while introducing only a negligible latency overhead. By this, the allocation ratio of limited BRAMs resources of the FPGA can heavily be reduced such that even larger and industrial-relevant SAT-instances can be processed successfully. It can be taken advantage of remaining BRAM resources to incorporate further optimization of the SAT-solving process to improve the resulting run-time even more. More precisely, the proposed memory-upscaling scheme allows for adjusting the memory word lines by identifying the essential information based on the given constraints. The encoding of a literal in the word line of the external memory is as follows: (a) the 15 most significant bits store the decision level, (b) the 5 middle bits keep meta information of the solving process and (c) the remaining 17 bits store the index of the literal.

For the memory-upscaled DPLL-based solver, the BRAMs are utilized for CPM entirely, which bounds the maximum number of clauses, and is as follows: The 15 most significant bits store the decision level of a clause, meaning the highest decision level of its literals, 1 bit holds *Clause Sat Status* (CSS) and the 10 least significant bits store the number of literals in a clause. Analogously, for the memory-upscaled CDCL-based solver, a share of the available BRAM resources is reserved for IGM and RLM. The encoding of IGM is as follows: The data width is 28 bits, while the first 27 bits store the clause address of the implied literal. Furthermore, the most significant bit checks whether the decision level is stored, which allows to group the implications of the same decision level, as required for a potential backtracking operation during the conflict resolution. The RLM encoding is the same as the literal encoding in the main memory.

IV. Experimental Evaluation & Conclusion

The proposed HW SAT-solver has been implemented on a *Xilinx Kintex-7 KC705 Evaluation Platform* with an embedded *xc7k325tffg900* FPGA core that operates at 125 MHz system clock. This device has an external DDR3 memory that can be accessed via the *Memory Interface* (MIG) to submit (memory) read and write requests. A controller has been implemented to seamlessly integrate the MIG within the solver's search procedure while keeping the lightweight character of the solver. This is clearly proven by the low utilization ratio of the *LUTs* (*Muxes*) FPGA resources, which are about 5.50% (0.35%) for the DPLL-based and 6.09% (0.32%) for the CDCL-based HW SAT-solver. The overall system is visualized in Fig. 1 with all the semantic memory blocks, as mentioned in the previous sections.

The proposed clever orchestration of the memory resources in combination with an effective memory model allows for a significant enhancement of the processible SAT-instance sizes. The resulting limits are as follows: The maximum number of clauses and literals is $524,288$ for DPLL-based solver and $262,144$ for CDCL-based solver, which can hold at maximum $131,072$ implications. Besides this, a maximum number of decision levels (literals per clause) is $65,536$ $(1,024)$ is supported, which were not exceeded by any considered benchmark. In comparison to previously published approaches, the proposed memory-upscaled DPLL-based solver allows for processing over 30 times and CDCL-based solver over 15 times larger instances.

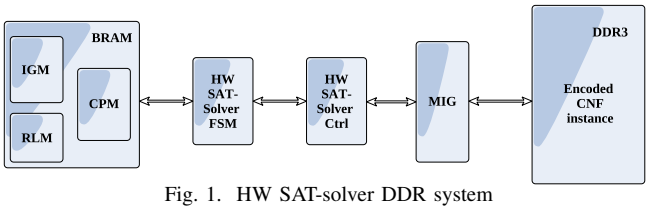

Fig. 1. HW SAT-solver DDR system

TABLE I
Run-time results

Instance	#Var	#Cls	Status	Run-time [s] DPLL-based	Run-time [s] CDCL-based
ii16b1	1,728	24,792	SAT	timeout	49.04
b12-4	2,450	20,666	UNSAT	0.13	0.05
b17-100	50,282	183,811	SAT	timeout	17,990.00
average-1	65,210	177,351	UNSAT	89.45	67.50
weighted-2	116,106	140,078	UNSAT	timeout	5,720.66

Table I presents the resulting run-time of both memory-upscaled HW SAT-solver when considering large benchmark SAT-instances. Note that the previous solvers could not handle these instances. The DPLL-based solver is not able to solve some specific instance within 5 hour, which is assumed as a timeout. The CDCL-based solver solves all considered benchmark successfully since the invoked clause learning techniques improve the search process, in particular, when processing hard-to-solve SAT-instances.

This paper proposed a new memory-upscaling scheme for hardware-based SAT-solvers, which is seamlessly integrated into both a DPLL-based and a CDCL-based solver while retaining the solvers' lightweight character. The proposed memory scheme allows for a clever orchestration of DDR3 memory resources in combination with the available BRAMs for both, the original SAT-instance and learnt clauses. The considered benchmark clearly demonstrated that the newly proposed scheme allows for the first time to process even large SAT-instances within a reasonable time. Different parameters exist to provide a trade-off between utilized resources and the maximum processible instance size. By this, it is even possible to enlarge the boundaries and, hence, to tackle the upcoming challenges in the context of self-verification successfully.

Acknowledgments: *This work was supported by the German Federal Ministry of Education and Research (BMBF) within the project SELFIE under grant no. 01IW16001.*

References

[1] W. C. Rhines, "Design verification challenges: Past, present and future," in *DVCon US*, 2016.
[2] M. Ring, F. Bornebusch, C. Lüth, R. Wille, and R. Drechsler, "Better late than never: Verification of embedded systems after deployment," in *DATE*, 2019, pp. 1–6.
[3] R. Drechsler, M. Fränzle, and R. Wille, "Envisioning self-verification of electronic systems," in *ReCoSoC*, 2015, pp. 1–6.
[4] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999.
[5] B. Ustaoglu, S. Huhn, D. Große, and R. Drechsler, "SAT-Lancer: a hardware SAT-solver for self-verification," in *GLSVLSI*, 2018, pp. 479–482.
[6] B. Ustaoglu, S. Huhn, F. Sill Torres, D. Große, and R. Drechsler, "SAT-Hard: a learning-based hardware SAT-Solver," in *DSD*, 2019, pp. 74.–81.
[7] M. Davis, G. Logeman, and D. Loveland, "A machine program for theorem proving," *Comm. of the ACM*, vol. 5, pp. 394–397, 1962.
[8] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik, "Efficient conflict driven learning in a boolean satisfiability solver," in *ICCAD*, 2001, pp. 279–285.