Solving SAT with a Context-Switching Virtual Clause Pipeline and an FPGA Embedded Processor

C.J. Tavares, C. Bungardean, G.M. Matos, and J.T. de Sousa

INESC-ID/IST-Technical University of Lisbon/Coreworks, Lda
R. Alves Redol, 9, 1000-029 Lisboa, Portugal[1]
jose.desousa@inesc-id.pt

Abstract. *This paper proposes an architecture that combines a context-switching virtual configware/software SAT solver with an embedded processor to promote a tighter coupling between configware and software. The virtual circuit is an arbitrarily large clause pipeline, partitioned into sections of a number of stages (hardware pages), which can fit in the configware. The hardware performs logical implications, grades and select decision variables. The software monitors the data and takes care of the high-level algorithmic flow. Experimental results show speed-ups that reach up to two orders of magnitude in one case. Future improvements for addressing scalability and performance issues are also discussed.*

1 Introduction

Definitions and motivation. The satisfiability (SAT) problem — given a Boolean formula $F(x_1,x_2, ..., x_n)$, find an assignment of binary values to (a subset of the) variables, so that F is set to 1, or prove that no such assignment exists — is a central, NP-complete computer science problem [1], with many applications in a variety of fields. Typically F is expressed as a product-of-sums, which is also called conjunctive normal form (CNF). The terminology is reviewed via an example: in the formula $F=(x_1+x_2)(\neg x_1+ x_2)(x_1+\neg x_2)$, we have two variables, x_1 and x_2, and three clauses, each with two literals; the literals in the third clause are x_1 and $\neg x_2$, where x_1 is a non-inverted literal and $\neg x_2$ is an inverted literal. The assignment (x_1=1, x_2=1) is a satisfying assignment, as it sets F=1. Hence F is satisfiable. The formula $G=(x_1+x_2)(\neg x_1+ x_2)(x_1+\neg x_2)(\neg x_1+ \neg x_2)$ is unsatisfiable. The number of variables in a formula is denoted n and the number of clauses m. A k-clause is a clause that has k literals. A k-SAT problem is one where clauses have at most k literals.

Previous work. In recent years, solving SAT using reconfigurable hardware (configware) has become a major challenge for Reconfigurable Computing (RC) experts. It is well known that, to become a generally accepted computing paradigm, RC has to prove itself able to tackle important computer science problems such as

[1] This work has been partially funded by FCT (Portugal), Project POSI/CHS/34562/2000.

J. Becker, M. Platzner, S. Vernalde (Eds.): FPL 2004, LNCS 3203, pp. 344–353, 2004.

SAT, competing with advanced software SAT solvers such as GRASP [4], CHAFF [5] and BERKMIN [6].

Several research groups have recently explored different approaches to implement SAT on configurable hardware [8-17], as an alternative to software SAT solvers. The satisfiers implement variations of the classical full search Davis-Putnam (DP) SAT, algorithm [18]. More recently, incomplete, local search SAT algorithms like WSAT or GSAT have also been contemplated with configware implementations [14,19]. An interesting survey comparing the various approaches that have been proposed in the literature is given in [20].

The most important problems addressed by the various proposals are the following: (1) the method used to select the next decision variable and its value to be tried [8,16,17]; (2) the compilation time spent in preparing the FPGA-based circuit to be emulated [14,16,17]; (3) the ability to solve problems of an arbitrary large size [12,16,17]; software-hardware partitioning [13,14,16,17].

Main contributions and organization of the paper. This paper presents a hard evidence analysis of our approach to configware-software SAT solving. The main contributions are the following:

1. Proposes the use of an embedded processor (Microblaze from Xilinx [21]) to create a tighter coupling between configware and software, eliminating expensive communications between the two.
2. Proposes the use of a decision variable pipelined comparator tree, to select the next decision variables.
3. Publishes the first experimental results obtained with the actual (non simulated) configware/software SAT solver system proposed in [16] and refined in [22], which is also used to indirectly derive results for the architecture proposed in this paper.

The remainder of this paper is organized as follows. Section 2 presents an overview of the predecessor of the SAT solver system being proposed. Section 3 presents the new system. Section 4 presents experimental results and their analyses. Finally, Section 5 outlines our conclusions and describes our current and future work.

2 Overview of the Predecessor System

Our current system evolved from a previous architecture already published in [16,22,23], which is summarized in Figure 1. The SAT solver runs partly in software and partly in configware. The software runs on a host computer and communicates with the configware via the PCI bus. The configware resides in a board containing an FPGA, a control CPLD and two single port RAMs, M1 and M2. After the configware architecture is loaded in the FPGA, it may be used to process any SAT problem instance, totally avoiding hardware instance-specific computation. The architecture can be outlined as a virtual pipeline of clause data processors. Each clause data

processor represents a clause of the SAT instance. The virtual circuit is partitioned in several hardware pages. Each page implements a pipeline section of *D* stages and *M* clauses per stage. The board memories store the programming of the hardware pages (context), which specify the clauses in each page and their data, and the latest state of the SAT variables. Each memory position has *N* variables, and each variable is represented by a *K*-bit word (*K* is even), having 2 fields, P_0 and P_1, of *K*/2 bits. *L* memory addresses are used to store a total of *LN* variables. Therefore, the configware architecture is perfectly characterized by the parameters *(D,K,L,M,N)*. The processing consists of streaming the variables through the pipeline, back and forth between M1 and M2.

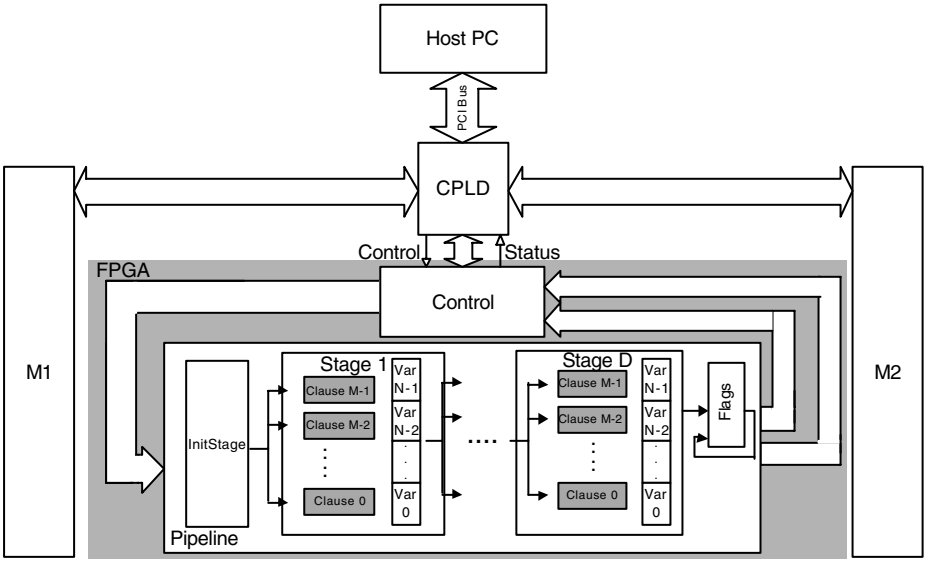

Fig. 1. High-level view of the previous system.

The algorithm starts when the software reads the SAT problem instance from a file in the CNF format, and maps it to the configware structure as a 3-SAT problem. If the original problem is *k*-SAT (*k*>3), it is converted into a 3-SAT problem. The whole formula compilation process runs in polynomial time, in a matter of seconds, much faster than any FPGA compilation flow. If the resulting circuit model is larger than the available configware capacity, it is partitioned in *P* virtual hardware pages able to fit in the configware. Thus the number of stages of the virtual clause pipeline is *PD*.

After the hardware pages are generated, the processing of variables in the clause pipeline can start. While moving the variables through the clause pipeline back and forth between M1 and M2, the values of their fields P_0 and P_1 are updated, until they reflect the number of clauses that would be satisfied if the variable had the value '0' or '1', respectively. This is because each field P_b is incremented whenever the variable is processed by one of its unresolved clauses that is satisfied for value *b*. The incrementing saturates when P_b reaches the value

$$P_b|_{MAX\_SCORE} = 2^{K/2} - 2$$

The maximum possible value, $P_b|_{\text{ASSIGNED}}$, is reserved to represent an implied or assigned variable:

$$P_b|_{ASSIGNED} = 2^{K/2} - 1$$

Our SAT solver implements a variation of the DP algorithm, and can be thought of as having three engines: the deduction, diagnosis and decision engines. The deduction engine assigns variables in unit clauses (computes logical implications) and grades unassigned variables according to the heuristic described above. When one or more clauses are falsified, the deduction engine reports the existence of conflicting assignments (conflicts). The diagnosis engine checks if a solution has been found, or if the last decision variable assignment has resulted in a conflict. If a conflict happened, the decision engine returns to the last decision variable (chronological backtracking), and sets it to its untried value. If no conflict is found the decision engine chooses the variable with the best heuristic score to be the next decision variable. If after a conflict there is no backtrack point then the formula is unsatisfiable. A flowchart summarizing the operation of the system is shown in Figure 2, where the filled areas represent tasks implemented in configware and the unfilled areas represent tasks implemented in software.

The operation of the virtual hardware scheme is as follows. Suppose the variables rest initially in memory M1. The first virtual hardware page is programmed, and all variables are streamed from M1, processed through the virtual hardware page, and stored in M2. If no conflict is detected, the next hardware page is loaded, and the processing of variables continues now from M2 back to M1. This process goes on for all virtual hardware pages sequentially, so that all sections of the virtual clause pipeline get to process all variables. Running all hardware pages on all variables is denoted a *pass*. During a pass new implications are generated and new clauses are satisfied - these are called *clause events*. For as long as new clause events are generated in a pass, another one is started, until no more events arise (this situation is denoted *stasis*) or a conflict is found. For the variables not yet assigned, P_0 and P_1 are recomputed during each pass so that their values are up to date when stasis is reached. After stasis, the *configware* informs the software on the latest location of the variables, either memory block M1 or M2. Then the software runs the diagnosis and decision engines.

3 The New System

After implementing and evaluating the system described in the previous section, we were not surprised to find out that its performance was far from what the simulation results in [23] had predicted — this is usually the case with a first prototype. Hence, we proceeded to analyse the causes of the discrepancies between simulated and actual results, and two major causes have been identified:

1. The communication between software and hardware was slow due to the latency of the PCI bus.
2. The software processing time was high, since the decision engine required all variables to be read, to find the one with the highest heuristic score.

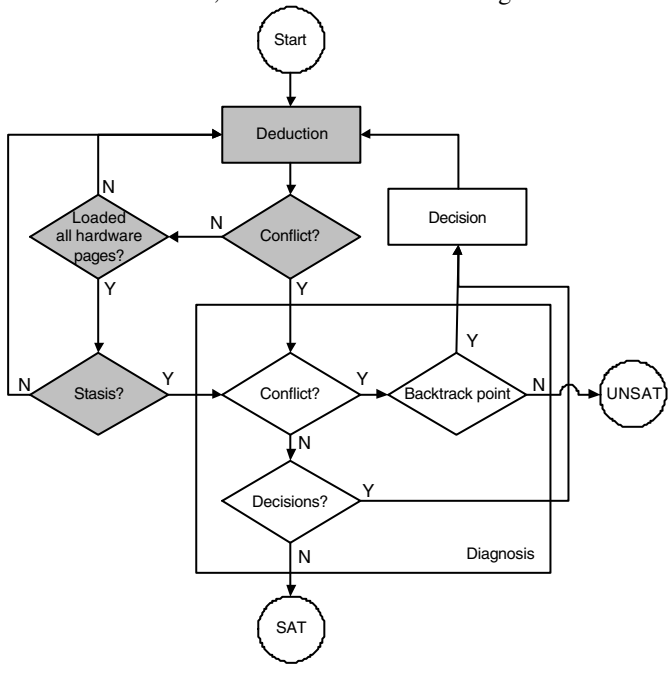

Fig. 2. Configware/Software SAT Solver Algorithm

To address these problems we came up with the improved architecture depicted in Figure 3, whose main new features are:

1. An embedded processor, MicroBlaze (MB) from Xilinx [21], was introduced to create a tight coupling between software and configware.
2. Comparator stages were introduced in the pipeline to select the variable with the best heuristic score, relieving the software of this burden.

MB uses the On-chip Peripheral Bus (OPB, inherited from IBM's CoreConnect infrastructure) to communicate with the clause pipeline, and to access the memories via the control CPLD. This way, MB and the clause pipeline share the same memory, and *there is no need to move the variables elsewhere.* In the predecessor system, where the host PC was running the software, all variables were transferred via DMA to the PC's central memory to be processed there, and then transferred back to the board memory. This had to be done for every decision variable, which, due to the high latency of the PCI bus, was still less expensive than accessing the variables one by one from the board memory.

To select the next decision variable a tree of variable comparators has been introduced. To preserve the frequency of operation, each level of the tree is placed in

a pipeline stage. The number of levels (height of the tree) is $log_2(2N)$, which creates a constraint for the minimum pipeline depth D. However, since N is not a large number, the tree is quite short anyway, and this new constraint is irrelevant.

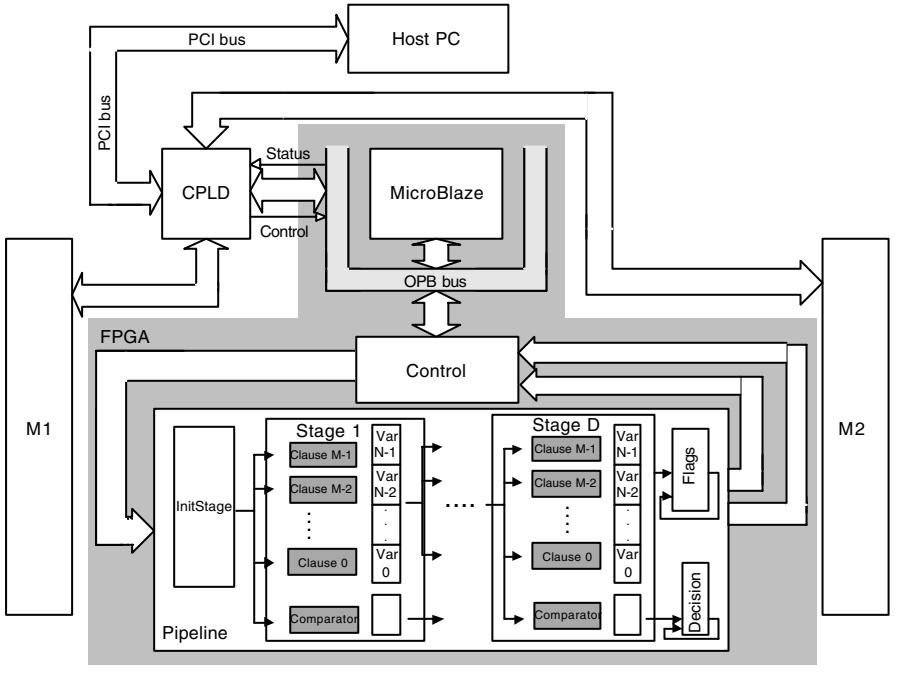

Fig. 3. High-level view of the proposed system.

The compilation of the SAT problem instance is still performed on the host PC for convenience and speed, since it is only done once as a pre-process step. Work to incorporate the decision variable comparator tree and the Xilinx's MicroBlaze soft processor is currently under way.

4 Results

All experimental results have been obtained using the system described in Section 2, whose prototype has been finished recently. The results for the proposed architecture have been derived by carefully measuring the DMA communication time and the elapsed time of the decision variable selection software routine, and subtracting these two terms from the total elapsed time obtained with the predecessor system of Section 2. The results obtained in this way are valid for an FPGA 30% larger, which is the hardware overhead estimated for the added variable comparator tree and the MicroBlaze embedded processor. This is no problem since FPGA devices much larger than the ones used in our experiments are commercially available.

Experimental setup. The software runs on a *Linux Suse 8.0* host PC with a Pentium 2 processor, at 300.699 MHz and 192 Mbytes of memory. The configware architecture is implemented in a Celoxica's RC1000 board [2] with *PCI* interface, featuring a XCV2000E device with 4 SRAM banks of 2 Mbytes and 32 bits wide. The memory blocks M1 and M2 are implemented using 2 SRAM banks each, so the variables are accessed as 64-bit words. The clause pipeline programming data (hardware pages or contexts) are stored as 128-bit words in the 4 SRAM banks simultaneously. The configware architecture is characterized by the parameters D=17, K=8, L=*1024*, M=4, N=7, as described in Section 2 and optimized according to [22]. Thus the system implemented can process SAT formulae of complexity up to 7168 variables 165036 clauses. The hardware occupies 96% of the FPGA area, so it has a complexity of 1.92M system gates and works at 40 MHz.

	A0 and A1			GRASP		
Example	Variables	Clauses	Decisions	Variables	Clauses	Decisions
aim-50-1_6-no-2	50	80	10141	50	80	13390
aim-50-1_6-no-3	50	80	37363	50	80	100471
aim-50-1_6-no-4	50	80	2883	50	80	2332
aim-50-2_0-yes1-3	50	100	2022	50	100	2170
aim-50-2_0-yes1-4	50	100	135	50	100	6164
aim-100-1_6-yes1-1	100	160	1287235	100	160	14384
aim-100-1_6-yes1-2	100	160	2119121	100	160	3916671
dubois20	60	160	25165823	60	160	12582911
ssa432_3	561	1405	3911	435	1027	3115
hole6	63	196	5883	42	133	3245
hole7	96	324	49405	56	204	21420
hole8	126	459	674595	72	297	378343
hole9	150	595	7520791	90	415	4912514

Table 1. Benchmark SAT instances used.

Experimental results. Our results have been obtained using a subset of the well-known benchmark set from DIMACS [7]. The results are compared to those obtained with GRASP, a well-known and publicly available SAT solver. Its options have been set to implement the same DP search algorithm we use in our system. Our k-SAT to 3-SAT decomposition technique augments the size of the formula, which may alter the number of decisions comparatively to using the original formula; GRASP is always run on the original formula. Table 1 shows the number of variables, clauses and decisions when running our algorithms, denoted A0 and A1, and GRASP. Note that a larger formula does not necessarily mean more decisions, since a different direction of the search may change the number of decisions dramatically. In Table 2, execution time results are presented. TGRASP is the total time taken by GRASP for each instance. TA0 is the total time taken by our predecessor system, and TA1 is the

time taken by the system proposed in this paper. SUA1 is the overall speed-up, and SUA1PD is the speed-up per decision.

These results show that the predecessor system (algorithm A0) can not obtain any speed-ups compared to GRASP (see columns TGRASP and TA0), while the proposed system (algorithm A1) can in fact obtain an acceleration against GRASP (see columns TGRASP, TA1 and SUA1). For the *aim-50-2_0-yes1-4* example the overall speed-up against GRASP is almost 250, which is a 2 orders of magnitude acceleration. However, comparing the execution times without taking in consideration the number of decisions shown in Table 1 is imprecise. In fact, the *aim-50-2_0-yes1-4* benchmark has a significantly lower number of decisions (135) when using the 3-SAT formula (A0 and A1) than when using the original formula (6164 decisions with GRASP). Therefore, a more fair comparison is to use the execution time per decision rather than the total elapsed time. These results are shown in column SUA1PD, which shows more modest speed-ups reaching one order of magnitude. On the other hand, many more examples show speed-ups greater than one, when using the SUA1PD metric.

EXAMPLE	TGRASP	TA0	TA1	SUA1	SUA1PD
aim-50-1_6-no-2	1,830	3,461	0,340	5,382	4,076
aim-50-1_6-no-3	10,510	14,165	0,874	12,025	4,472
aim-50-1_6-no-4	0,250	1,264	0,248	1,008	1,246
aim-50-2_0-yes1-3	0,350	0,847	0,030	11,667	10,871
aim-50-2_0-yes1-4	1,000	0,220	0,004	249,988	5,475
aim-100-1_6-yes1-1	2,600	787,969	85,081	0,031	2,735
aim-100-1_6-yes1-2	614,310	1312,940	164,268	3,740	2,023
dubois20.cnf	1040,400	6751,870	561,789	1,852	3,704
ssa432_3.cnf	1,300	39,810	10,817	0,120	0,151
hole6.cnf	0,260	2,325	0,372	0,699	1,267
hole7.cnf	4,340	24,830	6,003	0,723	1,668
hole8.cnf	56,450	408,494	136,194	0,414	0,739
hole9.cnf	825,120	6225,630	1932,090	0,427	0,654

Table 2. Execution time results for GRASP, A0 and A1.

Comparing Tables 1 and 2 we can observe that the speed-ups drop with the size of the instance, reflected in the size of the virtual clause pipeline. The explanation for this is the still immature virtual hardware scheme that has been implemented. In our current approach for every new variable assignment all clauses are evaluated, no matter if the new variables assigned are present in the evaluated clauses or not. This is inefficient and makes the assignment evaluation time $O(mn)$. Ideally the number of evaluated hardware pages should depend only on their having clauses to update with new assignments. Also, all variables are updated in the process, when there is only need to update variables in clauses that have been updated themselves. We have current plans to optimize this aspect, which will considerably boost the performance and prevent degradation when the problem scales up.

5 Conclusion

In this paper we have proposed a novel architecture of a SAT solver that combines a configurable hardware device with a small size embedded processor. The configware device implements a section of a virtual clause pipeline circuit (hardware page). The large virtual circuit embodies the SAT instance to be solved, and is operated by context-switching, where each context is a hardware page and its data.

The configware computes logical implications, grades decision variables using a heuristic score, and selects the next decision variable based on this figure. The software manages the search process (decision tree).

Experimental results have been obtained using a host PC to implement the software, and an FPGA to implement the configware. The performance of the proposed architecture has been derived by subtracting the PCI communication time and the elapsed time of the decision variable selection routine from the total elapsed. Work to incorporate the MicroBlaze embedded processor and the proposed comparator tree to select the next decision variable is under way. Our results show that speed-ups up to 2 orders of magnitude can be obtained with the proposed system.

Future work. We now have an architecture flexible enough to implement sophisticated algorithmic improvements, such as non-chronological backtrack and clause addition, like in modern software SAT solvers.

References

1. J. Gu, P. W. Purdom, J. Franco, and B. W. Wah, "Algorithms for the Satisfiability (SAT) Problem: A Survey", DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 35, pp. 19-151, 1997.
2. http://www.celoxica.com: RC1000 configware platform.
3. M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP Completeness", W.H. Freeman and Company, San Francisco, 1979.
4. J.M. Silva and K. A. Sakallah, "GRASP: a search algorithm for propositional satisfiability", IEEE Trans. Computers, vol. 48, n. 5, pp. 506-521, 1999.
5. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver", in Proc. 38th Design Automation Conference, 2001, pp. 530-535.
6. E. Goldberg and Y. Novikov, "BerkMin: a Fast and Robust SAT-solver", in Proc. Design, Automation and Test in Europe Conference, 2002, pp. 142-149.
7. http://www.intellektik.informatik.tudarmstadt.de/SATLIB/benchm.html: DIMACS challenge benchmarks.
8. T. Suyama, M. Yokoo, H. Sawada, and A. Nagoya, "Solving Satisfiability Problems Using Reconfigurable Computing", IEEE Trans. on VLSI Systems, vol. 9, no. 1, pp. 109-116, 2001.

9. P. Zhong, M. Martonosi, P. Ashar, and S. Malik, "Using Configurable Computing to Accelerate Boolean Satisfiability", IEEE Trans. CAD of Integrated Circuits and Systems, vol.18, n. 6, pp. 861-868, 1999.
10. M. Platzner, "Reconfigurable accelerators for combinatorial problems", IEEE Computer, Apr. 2000, pp. 58-60.
11. M. Abramovici and D. Saab, "Satisfiability on Reconfigurable Hardware", in Proc. 7th Int. Workshop on Field-Programmable Logic and Applications, 1997, pp. 448-456.
12. M. Abramovici and J. T. de Sousa, "A SAT solver using reconfigurable hardware and virtual logic", Journal of Automated Reasoning, vol. 24, n. 1-2, pp. 5-36, 2000.
13. Dandalis and V. K. Prasanna, "Run-time performance optimization of an FPGA-based deduction engine for SAT solvers", ACM Trans. on Design Automation of Electronic Systems, vol. 7, no. 4, pp. 547-562, Oct. 2002.
14. P. H. W. Leong, C. W. Sham, W. C. Wong, H. Y. Wong, W. S. Yuen, and M. P. Leong, "A Bitstream Reconfigurable FPGA Implementation of the WSAT algorithm", IEEE Trans. On VLSI Systems, vol. 9, no. 1, pp. 197-201, 2001.
15. M. Boyd and T. Larrabee, "ELVIS – a scalable, loadable custom programmable logic device for solving Boolean satisfiability problems", in Proc. 8th IEEE Int. Symp. on Field-Programmable Custom Computing Machines, 2000.
16. J. de Sousa, J. P. Marques-Silva, and M. Abramovici, "A configware/software approach to SAT solving", in Proc. 9th IEEE Int. Symp. on Field-Programmable Custom Computing Machines, 2001.
17. Skliarova and A. B. Ferrari, "A SAT Solver Using Software and Reconfigurable Hardware", in Proc. the Design, Automation and Test in Europe Conference, 2002, p. 1094.
18. M. Davis and H. Putnam, "A Computing Procedure for Quantication Theory" ACM journal, vol. 7, 1960, pp. 201-215".
19. R. H. C. Yap, S. Z. Q. Wang, and M. J. Henz, "Hardware Implementations of Real-Time Reconfigurable WSAT Variants", in Proc. 13th Int. Conf. on Field-Programmable Logic and Applications, Lecture Notes in Computer Science, vol. 2778, Springer, 2003. pp. 488-496.
20. Skliarova and A. B. Ferrari, "", in Proc. 13th Int. Conf. on Field-Programmable Logic and Applications, Lecture Notes in Computer Science, vol. 2778, Springer, 2003. pp. 468-477.
21. http://www.xilinx.com/xlnx/xil_prodcat_product.jsp?title=microblaze; MicroBlaze Soft Processor.
22. N. A. Reis and J. T. de Sousa, "On Implementing a Configware/Software SAT Solver", in Proc. 10th IEEE Int. Symp. Field-Programmable Custom Computing Machines, 2002, pp. 282-283.
23. R. C. Ripado and J. T. de Sousa, "A Simulation Tool for a Pipelined SAT Solver", in Proc. XVI Conf. on Design of Circuits and Integrated Systems, Nov. 2001, pp. 498-503.