

Reconfigurable Hardware SAT Solvers: A Survey of Systems

Iouliia Skliarova and António B. Ferrari

University of Aveiro, Department of Electronics and Telecommunications, IEETA
3810-193 Aveiro, Portugal
{iouliia, ferrari}@det.ua.pt

Abstract. By adapting to computations that are not so well supported by general-purpose processors, reconfigurable systems achieve significant increases in performance. Such computational systems use high-capacity programmable logic devices and are based on processing units customized to the requirements of a particular application. A great deal of research effort in this area is aimed at accelerating the solution of combinatorial optimization problems. Special attention was given to the Boolean satisfiability (SAT) problem resulting in a considerable number of different architectures being proposed. This paper presents the state-of-the-art in reconfigurable hardware SAT solvers. The analysis of existing systems has been performed according to such criteria as reconfiguration modes, the execution model, the programming model, etc.

1 Introduction

Although the concept of reconfigurable computing has been known since the early 1960s [1], it is only recently that technologies that allow it to be put into practice became available. The interest started at the beginning of the 1990s as FPGA densities broke the 10K logic gate barrier. Since then, reconfigurable computing became a subject of intensive research. For some classes of applications reconfigurable systems allow very good performance to be achieved compared to general-purpose computers. Other types of applications were mapped to reconfigurable hardware because it offers innovative opportunities to explore. According to the primary objective to be achieved, all these applications can be broadly divided into three categories: hardware emulation and rapid prototyping, evolvable hardware, and the acceleration of computationally intensive tasks. The last category is without doubt the prevalent one.

Recently, a series of attempts have been made to accelerate applications that involve rather complex control flow. In this context special attention was given to problems in the area of combinatorial optimization. Among them, the Boolean satisfiability (SAT) problem stands out. This may be partially explained by the extremely wide range of practical applications in a variety of engineering areas, including the testing of electronic circuits, pattern recognition, logic synthesis, etc. [2]. In addition, SAT has the honor of being the first problem shown to be NP-

complete [3]. This means that existing algorithms have an exponential worst-case complexity. Implementations based on reconfigurable hardware enable the primary operations of the respective algorithms to be executed in parallel. Consequently, the effect of exponential growth in the computation time can be delayed, thus allowing larger size instances of SAT to be solved [2].

SAT is a very well known combinatorial problem that consists of determining whether a given Boolean formula can be satisfied by some truth assignment. The search variant of this problem requires at least one satisfying assignment to be found. Usually, the formula is presented in conjunctive normal form, which is composed of a conjunction of a number of clauses, where a clause is a disjunction of a number of literals. Each literal represents either a Boolean variable or its negation. A survey of algorithmic methods of solving the SAT problem can be found in [2].

In this paper we present the current status of reconfigurable hardware SAT solvers and give an overview of the existing approaches and their tradeoffs. The remaining part of the paper is organized as follows. Section 2 is devoted to the description of the most well known architectures of reconfigurable hardware SAT satisfiers. Analysis and classification of these architectures according to different criteria is performed in section 3. Finally, concluding remarks are given in section 4.

2 Architectures of SAT Solvers

Recently, several research groups have explored different approaches to solve the SAT problem with the aid of reconfigurable hardware [4-5], [9-12], [14-22]. Since names have not typically been given to hardware SAT satisfiers, we will refer to them according to the first author's names of the respective publications.

Suyama et al. [4-5] suggested an architecture of an *instance-specific* SAT solver capable of finding all the solutions (or a fixed number of them) of a given problem instance. The employed algorithm is characterized by the fact that at any moment a full variable assignment is evaluated. A *dynamic decision strategy* based on both experimental unit propagation and a maximum-occurrence-in-clauses-of-minimum-size heuristic has been adopted. A number of circuits have been implemented on an Altera FLEX10K250 FPGA clocked at 10 MHz. *Suyama et al.* were able to achieve a small acceleration compared to the POSIT algorithm [6] executed on an UltraSPARC-II/296 MHz over some instances from DIMACS benchmark suite [7]. However, the time spent in hardware compilation and configuration was not taken into account.

Zhong et al. implemented a version of the well-known Davis-Putnam (DP) algorithm [8]. In their early work [9] they constructed an implication circuit and a state machine for each variable in the formula, all the state machines being connected in a serial chain. As a preprocessing step, all the variables are sorted taking into account the number of their appearances in a given formula. In [10] hardware implementation of *non-chronological backtracking* was proposed. The resulting hardware execution time was quite good but the design had two distinct drawbacks. First, the clock frequency was low (ranging from 700KHz to 2MHz for different

formulae). Second, the hardware compilation time took several hours (on a Sun 5/110MHz/64MB) thus canceling all the advantages of fast hardware execution.

In more recent work [11], [12] the basic design decisions were revised. As a result, a regular ring-based interconnecting structure was employed instead of irregular global lines, essentially reducing the compilation time in this way (to an order of seconds) and increasing the clock rate (to 20-30 MHz) [12]. In addition, a technique enabling conflict clauses to be generated and added was proposed. The experimental results are based on both hardware implementation (on an IKOS emulator containing a number of FPGA array boards) and simulation. The speedups achieved over the software satisfier GRASP [13] executing on a Sun5/110MHz/64MB (in a restricted mode), including the hardware compilation and configuration time, are of an order of magnitude [12] for a subset of the DIMACS SAT benchmarks [7].

Abramovici et al. [15] employed the technique of modeling a formula by a 2-level circuit. The SAT solver proposed in [14] is based on the PODEM algorithm. In [15] an improved architecture is suggested that employs the DP algorithm and implements an enhanced variable selection strategy. For hardware implementation *Abramovici et al.* suggest creating a library of basic modules that are to be used for any formula. The modules have predefined internal placement and routing. In this case the solver circuit will be built from modules, which allows the compilation time to be reduced (to the order of minutes). The authors implemented simple circuits on XC6264 FPGA and simulated the bigger ones. For a circuit occupying the whole area of the XC6264 FPGA the clock frequency is about 3.5 MHz. In [15] *Abramovici et al.* report speedups from 0.01 to 7000 (after time unit justification) achieved over GRASP [13] for a subset of DIMACS SAT benchmarks [7]. In [15] a virtual logic system was proposed allowing circuits to be constructed for solving SAT problems that are larger than the available hardware resources. This is achieved by decomposing a formula into independent sub-formulae that can be processed in separate FPGAs either concurrently or sequentially.

The SAT solver proposed by *Platzner et al.* [16], [17] is similar to that of *Zhong* [9]. It consists of a column of finite state machines, deduction logic and a global control unit. The deduction logic computes the result of the formula based on the current partial variable assignment. All variable assignments are tried in a fixed order. The authors implemented an accelerator prototype on the base of a Pamette board containing 4 Xilinx XC4028 FPGAs. The speedups obtained for *hole6...hole10* SAT benchmarks from DIMACS [7], including hardware compilation and configuration time, range from 0.003 to 7.408 compared to GRASP executing on a PII/300MHz/128MB [16]. The designs for the *holex* problems run at 20 MHz [17].

More recent work in this direction is targeted at avoiding instance-specific layout compilation. *Boyd et al.* [18] proposed an architecture for a SAT-specific programmable logic device that excludes instance-specific placement and routing. The suggested design consumes polynomial hardware resources (with respect to the number of variables and clauses) and requires polynomial time to configure. The authors implemented a small version of their SAT satisfier for a problem having 8 variables and 8 clauses on a Xilinx XC4005XL running at 12 MHz. However, no results on large benchmark problems were reported.

Sousa et al. [19], [20] were the first to propose partitioning the job between software and reconfigurable hardware with the most computationally intensive tasks (such as computing implications and choosing the next decision variable) assigned to hardware, while the control-oriented tasks (such as conflict analysis, backtrack control and clause database management) are performed in software. The suggested SAT solver has an *application-specific* architecture that uses configuration registers for SAT formula instantiation [20]. In order to deal with instances that exceed the available hardware capacity, a virtual hardware scheme with context switching has been proposed. The results reported in [19] are based on a software simulator of the system under an estimated clock frequency of 80 MHz, assuming that the context-switching device can swap pages in one clock cycle.

Skliarova et al. [21], [22] proposed an application-specific SAT solver realizing a DP-based algorithm. The problem was formulated over a ternary matrix by setting a correspondence between clauses and variables of a formula and rows and columns of the matrix. In order to solve various problem instances it is only necessary to download the respective matrix data. All the other components of the satisfier remain unchanged. This allows local reconfigurability to be used and reduces the configuration overhead. The problem is partitioned between software and reconfigurable hardware in such a way that an FPGA is only responsible for processing sub-problems that appear at various levels of the decision tree and satisfy the imposed hardware constraints (such as the maximum allowed number of rows and columns in the matrix). This technique permits problems to be solved that exceed the resources of the available reconfigurable hardware. The SAT satisfier was implemented on an ADM-XRC PCI board containing one XCV812E Virtex-EM FPGA (running at 40MHz). The results of experiments on some of DIMACS benchmarks [7] have shown that it is possible to achieve a significant speedup compared to GRASP (up to 111x, including the FPGA configuration time, with GRASP executed on an AMD Athlon/1GHz/256MB).

3 Analysis of Hardware SAT Solvers

In this section we attempt to analyze the reconfigurable hardware SAT solvers according to such criteria as algorithmic issues, programming model, execution model, reconfiguration modes, logic capacity and performance. Table 1 summarizes the respective characteristics of the architectures considered in the previous section.

3.1 Algorithmic Issues

The majority of the existing reconfigurable hardware SAT solvers employs some variation of the Davis-Putnam algorithm [8]. An exception to this is the SAT satisfier of *Abramovici et al.*, which implements a PODEM-based algorithm [14].

The search process in the DP algorithm is usually organized with the aid of a *decision tree*, whose nodes are characterized by the respective partial variable assignments, and arcs represent the *decisions* taken. There exist two basic approaches

to the selection of the decision variables: *static* and *dynamic*. Although dynamic selection has been considered to be a difficult task for hardware implementation, it was realized in a number of architectures [5], [19-22].

In the present-day software SAT solvers a lot of advanced techniques (such as non-chronological backtracking [13]) are employed that enable those regions of the search space that do not contain any solution to be identified and avoided. However, up to now these techniques have been largely ignored by hardware SAT solvers. The few exceptions to this rule are the SAT satisfiers of *Zhong et al.* [12] and *Sousa et al.* [19, 20] (the latter implements them in software).

3.2 Programming Model

There are two basic approaches to mapping a SAT formula to a reconfigurable system: *instance-specific* and *application-specific*. The first approach has been extensively explored by the SAT research community [4-5], [9-12], [14-17] and assumes the generation of an individual hardware configuration for each problem instance. In this case, a typical design flow is used to describe and implement either a whole instance-specific circuit or a number of primary modules, which are further customized (at compile time) by specially developed software tools to match the respective formula.

In an *application-specific approach* the circuit is designed and optimized only once, after which it can be used for different problem instances [18-22]. This can be achieved with the aid of a hardware template, which is also developed using a typical design flow but is customized with data for a particular problem at run-time (instead of compile-time). It should be noted that in this case a hardware compilation step is completely avoided.

3.3 Execution Model

A SAT problem can be either entirely mapped to reconfigurable hardware (leaving just the tasks of preprocessing and initialization to the host processor) [4-5], [9-12], [14-18] or partitioned between hardware and software [19-22]. There exist different methods of software/hardware partitioning. In the domain of SAT solvers, two are usually employed: *partitioning according to computational complexity* and *partitioning with respect to logic capacity*.

The first method assigns computationally intensive portions of an application to hardware, while the remaining portions that exhibit little parallelism are handled by the host processor [19], [20]. Reconfigurable systems of this type are based on the 90/10 rule, which states that 90% of execution time of an application is spent by 10% of its code. Thus, in order to increase performance it is attempted to accelerate this small portion of an application with the aid of programmable logic devices.

The second method performs partitioning according to the available logic capacity of hardware employed [21], [22]. In this case, if a problem instance does not “fit” to a chosen device (or a number of interconnected devices), it has first to be processed by software up to the point at which it can be transferred to hardware.

3.4 Reconfiguration modes

In the domain of reconfigurable computing it is common to distinguish between two configuration modes: *static mode* (also known as *compile-time configuration* or *design-time binding*) and *dynamic mode* (frequently referenced as *run-time configuration* or *implementation-time binding*).

Static configuration assumes fixed functionality of the device once it has been programmed [4-5], [9-10], [16-17]. Dynamic reconfiguration allows the functionality of the system to be changed during the execution of an application. Dynamic reconfiguration can in turn be *partial* or *global*. Global reconfiguration reserves all the hardware resources for each step of execution. After a step has been concluded, the device may be reprogrammed for the next step [15]. Partial reconfiguration implies the selective modification of hardware resources [19-22]. This opportunity allows the hardware to be adapted to better suit the actual needs of the application. Since only selected portions are reconfigured, the configuration overhead is less than in the previous case.

A variety of reprogrammable devices can be employed to carry out dynamic reconfiguration. *Single-context* devices require complete reprogramming in order to introduce even a small change. Although many commercially available FPGAs are single-context, there exist techniques (based on hardware templates) that allow partial reconfiguration to take place [19-22]. *Multi-context* devices possess various planes of configuration information with just one of them active at any given moment [19]. The main advantage of such devices is the ability to switch the context very fast. *Partially reconfigurable* devices permit small portions of their resources to be modified without disturbing the remaining parts. Although this kind of devices (such as the XC6200 family of Xilinx) was employed for some SAT solvers [15], the potential for partial reconfigurability has not been explored.

3.5 Logic Capacity

The logic capacity of the employed hardware device is always limited. Thus, efficient techniques are needed to deal with the situation when a problem instance exceeds the available hardware resources. The answers to this issue differ accordingly to the programming and execution models adapted. Basically, four possibilities have been explored.

The first is the expansion of the logic capacity by interconnecting a number of programmable devices and partitioning the circuit between them. It should be noted that fast and efficient multi-device partitioning and routing is quite a difficult task (of course modular design styles [12] can alleviate it). Moreover, the working frequency of such multi-device systems is usually quite limited.

The second method is to partition the problem into a series of configurations to be run either sequentially or in parallel. The partitioning is performed by decomposing an initial formula into a set of independent sub-formulae [15]. Each sub-formula must satisfy the imposed hardware constraints. The main limitation of this method is that the efficiency of the decomposition greatly depends on the characteristics of the

formula. As a result, for some problem instances the partitioning time may increase to unacceptable levels.

The third method is based on software/hardware partitioning according to the available logic capacity of hardware that is employed (see section 3.3). In this case just those sub-problems that appear at different levels of the decision tree and respect the capacity limitations are assigned to hardware, the remaining portion of the problem being processed by a software application [21-22].

The last method is based on a virtual hardware scheme proposed in [19-20], which relies on dividing the circuit into a series of hardware pages that are successively run being the intermediate results stored in external memory blocks. Since all the hardware pages have the same structure with only a number of registers being reconfigured, the page switching is performed very fast.

3.6 Performance

The total time (t_{total}) spent by a reconfigurable hardware SAT satisfier to solve a particular problem instance comprises four components: hardware compilation time (t_{comp}), hardware configuration time (t_{conf}), time required for communication between software and hardware (t_{comm}) and actual execution time (t_{ex}). If a problem solution is partitioned between software and hardware then the execution time t_{ex} is composed of software execution time (t_{ex_s}) and hardware execution time (t_{ex_h}). It should be noted that the values of these components depend on the programming and execution models employed and some of them may be zero. For example, if a problem instance is entirely mapped to hardware, usually there is no communication (except for notifying the final result) between the host processor and the programmable device. In the same manner, if an application-specific approach is followed, the hardware compilation time is zero. Actually, the compilation time may constitute a large portion of the total solving time. For easy problem instances it even dominates and cancels out all the benefits of fast hardware execution [4-5], [9-10], [16-17]. That is why a number of techniques targeted at reducing the hardware compilation time have been proposed. They are based on exploiting modular design styles and developing customized software tools instead of using commercially available ones [11-12], [15].

One characteristic inherent in reconfigurable hardware SAT solvers is that it is very difficult to analyze and compare their performance accurately. As a rule, the designers present the results achieved in the light of the software SAT satisfier GRASP [13]. However, GRASP is run on different platforms and with dissimilar parameters that heavily influence its performance. Moreover, the parameters set are frequently not published. The majority of the SAT solvers considered involve a hardware compilation step, which is sometimes ignored (or hidden) when presenting the results. It is also difficult to estimate the exact impact of compilation on the total execution time because of the variety of software platforms used. Nevertheless, in all recent designs a clear intention to reduce and even to avoid the hardware compilation step is apparent [12], [15], [18-22].

As shown by the results of the 2002 software SAT competition [23], GRASP has been surpassed by more recent SAT satisfiers such as zChaff [24] and BerkMin [25]. Consequently novel algorithmic and architectural techniques need to be explored in

order to put the reconfigurable hardware SAT solvers in a more favorable light comparing to a software solution.

Table 1. Principal characteristics of the reconfigurable hardware SAT solvers

SAT solver	Algorithmic issues	Programming model	Execution model	Reconfiguration mode	Logic capacity	Performance (t_{total})
<i>Suyama et al.</i>	DP-like algorithm with dynamic selection	instance-specific	hardware only	static	multi-FPGA system	t_{comp}^+ t_{conf}^+ t_{ex_h}
<i>Zhong et al. [12]</i>	DP-based algorithm with static selection, non-chronological backtracking, conflict analysis	instance-specific	hardware only	dynamic (global)	multi-FPGA system	t_{comp}^+ t_{conf}^+ t_{comm}^+ t_{ex_h}
<i>Platzner et al.</i>	DP-based algorithm with static selection	instance-specific	hardware only	static	use larger device	t_{comp}^+ t_{conf}^+ t_{ex_h}
<i>Abramovici et al. [15]</i>	DP-based algorithm with optimized static selection	instance-specific	hardware only	dynamic (global)	logic partitioning in sub-formulae	t_{comp}^+ t_{conf}^+ t_{comm}^+ t_{ex_h}
<i>Sousa et al.</i>	DP-based algorithm with dynamic selection and conflict analysis (in software)	application-specific	software/hardware partitioning according to computational complexity	dynamic (partial)	virtual hardware scheme	t_{conf}^+ t_{comm}^+ $t_{ex_s}^+$ t_{ex_h}
<i>Skliarova et al.</i>	DP-based algorithm with dynamic selection	application-specific	software/hardware partitioning according to logic capacity	dynamic (partial)	software/hardware partitioning	t_{conf}^+ t_{comm}^+ $t_{ex_s}^+$ t_{ex_h}

4 Conclusion

This paper is dedicated to the description and comparison of reconfigurable hardware SAT solvers. The analysis leads to the following conclusions:

- The majority of designers implement complete search algorithms derived from the DP algorithm. Conflict analysis is usually not performed and just chronological backtracking is executed (with a few exceptions).
- Practically all the proposed SAT solvers are based on the instance-specific approach. However, the hardware compilation time restricts the range of problems for which a reconfigurable hardware solution is more effective than the software-

based approach. That is why all recent efforts have been focused on avoiding instance-specific placement and routing.

- All the reconfigurable SAT solvers considered are loosely coupled systems with the programmable device (usually, a commercially available FPGA) being attached to the host processor via an external interface.
- It is quite difficult to compare the results that have been achieved. First, the hardware compilation and configuration times are not always clearly exposed. Second, the results are usually compared to GRASP, executed on different platforms with dissimilar parameters, which can lead to variations in the solving time of up to an order of magnitude.
- Real-world SAT formulae are quite large, however how instances that do not fit into an available device can be handled efficiently is not always discussed. Recently, in what seems to be a promising solution, it was suggested that the problem should be partitioned between software and reconfigurable hardware. It should also be noted that due to the rapid evolution in FPGA capacity, many challenging problem instances can now either be fitted into a single FPGA, or at least partitioned more efficiently.
- The speedups achieved by reconfigurable hardware compared to a software solution are significant just for certain classes of SAT instances, for which the optimization techniques proposed and implemented by software SAT satisfiers are not very efficient. Some examples of these techniques are: different decision strategies, exploiting problem symmetry, careful conflict analysis, etc. Consequently, although many interesting and worthwhile architectures have already been proposed, innovative approaches still need to be explored in the reconfigurable hardware domain.

Acknowledgment

This work was supported by the Portuguese Foundation of Science and Technology under grant No. FCT-PRAXIS XXI/BD/21353/99.

References

1. Estrin, G.: Reconfigurable Computer Origins: The UCLA Fixed-Plus-Variable (F+V) Structure Computer. *IEEE Annals of the History of Computing*. Oct.-Dec. (2002) 3-9
2. Gu, J., Purdom, P.W., Franco, J., Wah, B.W.: Algorithms for the Satisfiability (SAT) Problem: A Survey. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 35 (1997) 19-151
3. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company. San Francisco (1979)
4. Yokoo, M., Suyama, T., Sawada, H.: Solving Satisfiability Problems Using Field Programmable Gate Arrays: First Results. In: *Proc. of 2nd Int. Conf. on Principles and Practice of Constraint Programming* (1996) 497-509

5. Suyama, T., Yokoo, M., Sawada, H., Nagoya, A.: Solving Satisfiability Problems Using Reconfigurable Computing. *IEEE Trans. on VLSI Systems*, vol. 9, no. 1 (2001) 109-116
6. Freeman, J.W.: Improvements to Propositional Satisfiability Search Algorithms. Ph.D. dissertation. Univ. Pennsylvania (1995)
7. DIMACS challenge benchmarks. [Online]. Available: <http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html>
8. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. *Communications of the ACM* n. 5 (1962) 394-397
9. Zhong, P., Martonosi, M., Ashar, P., Malik, S.: Using Configurable Computing to Accelerate Boolean Satisfiability. *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 18, n. 6 (1999) 861-868
10. Zhong, P., Ashar, P., Malik, S., Martonosi, M.: Using reconfigurable computing techniques to accelerate problems in the CAD domain: a case study with Boolean satisfiability. In: *Proc. Design Automation Conf.* (1998) 194-199
11. Zhong, P., Martonosi, M., Ashar, P., Malik, S.: Solving Boolean satisfiability with dynamic hardware configurations. In Hartenstein, R.W., Keevallik, A. (eds.) *Field-Programmable Logic: From FPGAs to Computing Paradigm* (1998). Springer-Verlag. 326-235
12. Zhong, P.: Using Configurable Computing to Accelerate Boolean Satisfiability. Ph.D. dissertation. Department of Electrical Engineering. Princeton University (1999)
13. Silva, L.M., Sakallah, K.A.: GRASP: a search algorithm for propositional satisfiability. *IEEE Trans. Computers*, vol. 48, n. 5 (1999) 506-521
14. Abramovici, M., Saab, D.: Satisfiability on Reconfigurable Hardware. In: *Proc. 7th Int. Workshop on Field-Programmable Logic and Applications* (1997), 448-456
15. Abramovici, M., de Sousa, J.T.: A SAT solver using reconfigurable hardware and virtual logic. *Journal of Automated Reasoning*, vol. 24, n. 1-2 (2000) 5-36
16. Platzner, M.: Reconfigurable accelerators for combinatorial problems. *IEEE Computer*. Apr. (2000) 58-60
17. Platzner, M., De Micheli, G.: Acceleration of satisfiability algorithms by reconfigurable hardware. In: Hartenstein, R.W., Keevallik, A. (eds.) *Field-Programmable Logic: From FPGAs to Computing Paradigm*. Springer-Verlag (1998) 69-78
18. Boyd, M., Larrabee, T.: 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 - FCCM* (2000)
19. de Sousa, J., Marques-Silva, J.P., Abramovici, M.: A configware/software approach to SAT solving”. In: *Proc. of 9th IEEE Int. Symp. on Field-Programmable Custom Computing Machines* (2001)
20. Reis, N.A., de Sousa, J.T.: On Implementing a Configware/Software SAT Solver. In: *Proc. of 10th IEEE Int. Symp. Field-Programmable Custom Computing Machines* (2002) 282-283
21. Skliarova, I., Ferrari, A.B.: A SAT Solver Using Software and Reconfigurable Hardware. In: *Proc. of the Design, Automation and Test in Europe Conference* (2002) 1094
22. Skliarova, I., Ferrari, A.B.: A hardware/software approach to accelerate Boolean satisfiability. In: *Proc. of IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop* (2002) 270-277
23. Simon, L., Le Berre, D., Hirsch, E.: The SAT2002 Competition. Technical Report (preliminary draft) [Online]. Available: <http://www.satlive.org/SATCompetition/onlinereport.pdf> (2002)
24. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an Efficient SAT Solver. In: *Proc. of the 38th Design Automation Conference* (2001) 530-535
25. Goldberg, E., Novikov, Y.: BerkMin: a Fast and Robust SAT-solver. In: *Proc. Design, Automation and Test in Europe Conference* (2002) 142-149