# A Shift Register based Clause Evaluator for Reconfigurable SAT Solver

Mona Safar, Mohamed Shalan, M.Watheq El-Kharashi
Computer and Systems Engineering Department
Ain Shams University
Cairo, Egypt

Ashraf Salem
Mentor Graphics Egypt
Cairo, Egypt

## Abstract

*Several approaches have been proposed to accelerate the NP-complete Boolean Satisfiability problem (SAT) using reconfigurable computing. We present an FPGA based clause evaluator, where each clause is modeled as a shift register that is either right shifted, left shifted, or standstill according to whether the current assigned variable value satisfy, unsatisfy, or does not effect the clause, respectively. For a given problem instance, the effect of the value of each of its variables on its SAT formula is loaded in the FPGA on-chip memory. This results in less configuration effort and fewer hardware resources than other available SAT solvers. Also, we present a new approach for implementing conflict analysis based on a conflicting variables accumulator and priority encoder to determine backtrack level. Using these two new ideas, we implement an FPGA based SAT solver performing depth-first search with non-chronological conflict directed backtracking. We compare our SAT solver with other solvers through instances from DIMACS benchmarks suite.*

## 1. Introduction

The Boolean Satisfiability problem (SAT) is a central problem in AI, mathematical logic, and computing theory with wide range of CAD applications. SAT tests whether a given Boolean formula, usually represented in Conjunctive Normal Form (CNF), is satisfiable by searching for an assignment of truth values to variables that makes the formula evaluates to '1'. SAT solving algorithms involve compute-intensive, logic bit-level, highly parallelizable operations, which makes reconfigurable computing appealing [1]. Various approaches have been proposed to accelerate SAT solving using reconfigurable computing by either migrating the whole problem to hardware or partitioning the problem into hardware and software parts [2].

We present a reconfigurable SAT solver architecture that exploits the fine granularity and massive parallelism of FPGAs to evaluate the SAT formula. The solver performs a depth-first search with non-chronological conflict directed backtracking. The primary contribution of our work is proposing a novel way for evaluating clauses in hardware. Each clause is modeled as a shift register that is either right shifted, left shifted, or standstill according to whether current assigned variable value satisfy, unsatisfy, or does not effect the clause, respectively. The presence of '1' at the leftmost bit of a shift register indicates a conflict. Mapping different SAT problem instances onto our reconfigurable SAT solver is achieved by loading the effect of the value of each of the given problem instance variables on the evaluation of its CNF clauses in the FPGA on-chip memory. This results in lower hardware cost and less configuration effort compared to existing SAT solvers where functional units need to be configured with the given problem data, mainly literals associated with each clause, on which they operate. The second contribution is presenting a new approach for implementing conflict analysis without reverse traversal of implication graph as implemented by other hardware SAT solvers. Conflicting variables are accumulated and fed to a priority encoder that determines the backtrack level.

The rest of the paper is organized as follows. Section 2 reviews the problem in brief and discusses instance-specific and application-specific approaches for mapping a SAT formula to hardware. Section 3 reviews the previous work. Section 4 presents a shift register based clause evaluator and a proposed idea for implementing conflict analysis. Section 5 presents our hardware SAT solver architecture. Section 6 shows the experimental results in comparison with three other hardware SAT solvers. Finally, Section 7 presents concluding remarks.

## 2. Boolean Satisfiability

The SAT problem is a Constraint Satisfaction Problem (CSP) in which constraints are represented in a Boolean formula usually expressed in CNF. A K-SAT problem is one where clauses have at most K literals. If an assignment of variables that satisfies the formula exists, the formula is said to be satisfiable, otherwise it is unsatisfiable. A clause is satisfied if one of its literals is bound to '1', unsatisfied if all its literals are bound to '0', unit if one of its literals is not bound to a logic value whereas all others are bound to '0', and unresolved otherwise [3].

978-3-9810801-2-4/DATE07 © 2007 EDAA

Solving SAT is an intensive search operation with backtracking. State-of-art SAT solvers employ the Davis-Putnam (DP) method with improved strategies for decision, deduction, conflict analysis, and conflict driven learning [4]. Decision selects the next free variable and a logic value to assign to it. This selection can be static, where a fixed preorder is defined, or dynamic. There are varieties of heuristics acting as a base for the decision. Deduction infers the direct and transitive implications of an assignment through a process known as Boolean Constraint propagation (BCP). BCP sequentially applies the unit clause rule, which implies a free variable to be bound to '1' ('0') if its positive literal (negative literal) appears in a unit clause. The sequence of implications generated is captured by a directed implication graph [5]. If a variable is implied to both '1' and '0' by two or more clauses, a conflict arises. Instead of backtracking chronologically to the last decided variable, conflict analysis identifies the set of predecessor variables of this conflict by traversing the implication graph in reverse direction. It allows backtracking nonchronologically to the most recently assigned variable from this set. This effectively prunes the search space. Conflict-driven learning records a new clause constructed from the conflict set to prevent re-exploring same space.

Existing SAT solvers using reconfigurable hardware implement some or all of the above strategies. They not only differ in which strategies they implement and the way they are implemented, but also, in the approach used for mapping different problem instances to the implemented reconfigurable hardware. There are two basic approaches: instance-specific and application-specific [2]. In the instance-specific approach, a specialized hardware circuit is generated for each problem instance. The advantage of the instance-specific approach is that the hardware is tailored to the instance to be solved achieving higher performance and better utilization of hardware resources. On the other hand, the total problem solving time is greatly affected by the time overhead for hardware circuit compilation. In the application-specific approach, the hardware circuit is designed and compiled into hardware just once, then for each specific instance the hardware configuration is directly customized with problem instance data avoiding the hardware compilation step. Implementing application-specific hardware SAT solvers requires a modular design, where the functional elements are not built out from the instance data. Instead, they are designed to perform some processing on it. The design must address the problem of how to reflect the specific SAT instance parameters: number of variables (N), number of clauses (M), and for each clause, the number of literals in it (K). Another important issue is the design of a modular communication network that sends the variables' updated values to the respective clauses and, for architecture employing implications, feedbacks implied variables' values to the respective variables.

## 3. Previous Work

Platzner et al. [6][7] implemented an instance-specific reconfigurable accelerator for SAT. Their implemented architecture performs simple depth-first search with chronological backtracking based on evaluating the output of a combinational circuit to which the Boolean formula is mapped. To reduce the compilation time, a domain specific compiler is built instead of using conventional configurable hardware design tools [7].

Skliarova and Ferrari [8] formulated the SAT problem over a matrix, where rows and columns correspond to clauses and variables, respectively. Searching for a satisfying assignment of variables is formulated as searching for a vector that is orthogonal to each row of the matrix by the sequential application of various reduction and splitting methods on matrices that correspond to the decision, implication, and chronological backtracking operations in a traditional SAT search. The underlying FPGA is configured to support the maximum number of variables ($N_{max}$) and clauses ($M_{max}$) that can fit into it. Two special registers are used to store specific instance parameters N and M. Two $M_{max}$ x $N_{max}$ and two $N_{max}$ x $M_{max}$ block RAMs from the FPGA store matrices corresponding to the SAT instance.

Zhong et al. [9] introduced a modular architecture based on a regular ring structure. Clauses are mapped into similar clause modules grouped in a series of Processing Elements (PEs) on the ring. For modularity, the number of literals in a clause K is limited to 3. A time multiplexed pipelined bus circulates the variables' values in a fixed sequence with a synchronizing signal at the beginning. The clause module identifies the variable by a counter that is incremented in each clock cycle and a comparator that compares the counter value with a pre-stored value. The clause cell monitors values of the variables relevant to it. If the clause becomes unit, the clause cell forces a satisfying value for the free literal on the bus. Conflict analysis is carried out in a number of cycles. It ends when all the true values on the bus are determined by branch decisions. The main controller generates the conflict set and decides the backtrack level. The modified architecture supports dynamic clause addition to support conflict-driven learning. The connections of clause modules to the propagating global signals as well as the comparators are configured for each problem instance.

Dandalis et al. [10] proposed a reconfigurable architecture that evaluates clauses in parallel during the implication deduction phase. Variables associated with a given clause are stored in the local memory of the respective module. They proposed to use partial reconfiguration to update the contents of local memories for a specific problem instance.

Sousa et al. [11] presented a Configware/Software SAT solver where conflict diagnosis, backtrack control, and clause database management operations are left to a coupled software running on the host computer. A 3-clause cell has a configuration register for each literal. Configuration

data is organized in pages that are successively loaded in the board memory, where data corresponding to the variables also reside. Each memory word contains a slice of variables. Instead of the ring structure, variable slices are accessed sequentially from one memory block, processed in the clause pipeline and stored in another memory block.

## 4. Shift Register based Clause Evaluator and Conflict Analyzer

Compared with software implementations, the main advantage of implementing a SAT solver in FPGAs is the use of the massive parallelism in evaluating the SAT CNF formula [2]. Our proposed clause evaluator consists of M shift registers, one for each clause. For modularity, the number of literals in each clause K is limited to 3. Each register is first initialized to "0001000". It is either shifted left or right according to the effect of current assigned variable. An $N_{max}$ x $2*M_{max}$ RAM (Mem_VEOC) stores the effect of variables on clauses. Each word represents how the assignment of a '0' value to the associative variable changes the evaluation of each clause as follows:

- If Mem_VEOC[i][j:j+1] = "00" then variable i does not occur in clause j/2.
- If Mem_VEOC[i][j:j+1] = "01" then assigning '0' to variable i satisfy clause j/2.
- If Mem_VEOC[i][j:j+1] = "10" then assigning '0' to variable i tends to unsatisfy clause j/2, the clause is unsatisfied if this is the same for the other 2 variables in the clause.

There is no need to store the effect of assigning '1' to variables as it is just the opposite. Variables are stored in an array where the value of each variable is encoded in two bits. One bit, F, denotes whether the variable is free or not. In the latter case, the second bit, V, represents the value of the variable. Each register has 3 controlling signals: Mode, SL, and SR. Mode is the current assigned variable's encoding value bit V. SL and SR are the 2 bits stored in the memory word associated with the current variable representing the effect of its '0' value on the clause. Table 1 illustrates how these signals control the operation of each shift register. For binary clauses, corresponding shift registers are initially shifted left. This is achieved through the first memory word in Mem_VEOC representing a dummy '0' valued variable which contains "10" for each binary clause. Since unary clauses imply a unique satisfying value for their associated variable, they are easily handled in software on parsing the CNF formula of the given SAT instance. Fig.1 sketches a trace for a clause evaluation on exploring a simple search space. A clause is unsatisfied, and hence a conflict is detected, when the leftmost bit of a shift register is '1', otherwise, it is either unresolved or satisfied. A clause is unit when the bit second to the leftmost bit is '1'. Identifying unit clauses allows performing BCP.

Leftmost bits of all clauses, where a '1' indicates a conflict, are fed to an M-input priority encoder. In case of a conflict, it determines the index of the first unsatisfied clause. This represents the first stage in conflict analysis [12]. Aided by this index, the indices of variables in the unsatisfied clause, conflict predecessors, can be read from a clause database. A $M_{max}$ x $3*\log_2 N_{max}$ RAM (Mem_ClauseDB) acts as a clause database, each word represents a clause with its associated variables. Corresponding bits of the conflict predecessors' variables are set in an accumulator of conflicting variables which is fed to an N-input priority encoder. This priority encoder outputs the highest index of the activated conflicting variables; backtracking level, hence allowing non-chronological conflict directed backtracking. It also outputs a NoBck signal that is active in case there is no conflicting variable to backtrack to.

**Table 1. Clause evaluation shift operations**

| Mode (V(i)) | Mem_VEOC[i][j:j+1] (SL & SR) | Operation |
|---|---|---|
| 0 | 01 | Shift right |
| 0 | 10 | Shift left |
| 1 | 01 | Shift left twice |
| 1 | 10 | Shift right twice |

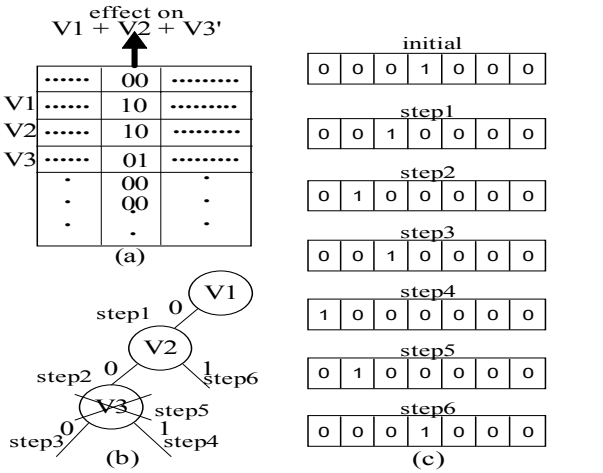


**Fig. 1. Example on clause evaluation: (a) Part of Mem_VEOC showing effect of variables on clause (V1 + V2 +V3'). (b) Explored search space. (c) Clause (V1 + V2 +V3') corresponding shift register evaluation.**

For different SAT instances, the two memories, Mem_VEOC and Mem_ClauseDB only need to be initialized. Compared to Zhong et al. [9], Dandalis et al. [10], and Sousa et al. [11], where each clause is modeled as a complex unit that performs one or more computation tasks of: parallel BCP, dynamic decision variable selection heuristics, and Conflict-driven learning, the simplicity of modeling each clause as a shift register obstructs handling these tasks but on the account of much lower hardware cost and

higher clock rates. Moreover, we avoid the need for configuring data in each clause to identify its associated variables and time spent to adapt it for a particular instance either using JBits, partial reconfiguration, or wasting hardware clock cycles to load this data from configuration data stored in memory. During solving a problem, we avoid spending cycles until the clause picks up values of its relevant variables from the circulating bus.

## 5. SAT Solver Architecture

We implemented an FPGA based SAT solver utilizing the clause evaluator and conflict analyzer described in the previous section. It performs depth first search with non-chronological conflict directed backtracking. The overall architecture is presented in Fig. 2.

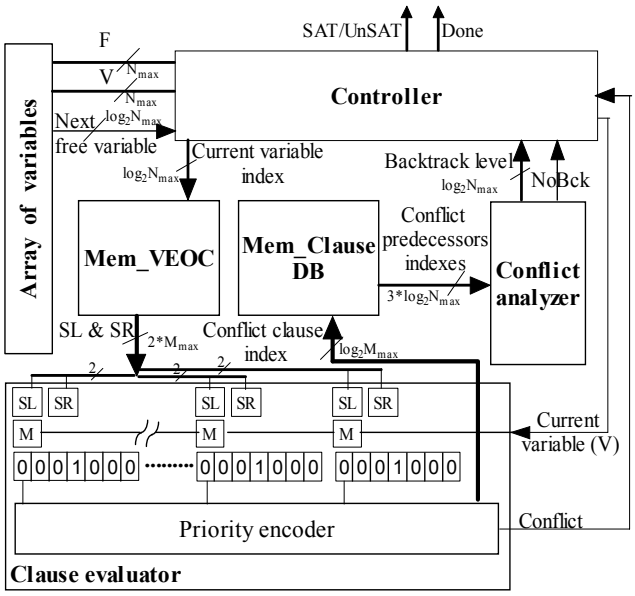


**Fig. 2. Our FPGA based SAT solver architecture.**

The control over computation is centralized. The controller has three states: branch, evaluate/analyze, and backtrack. In the branch state, the controller picks a free variable and assigns '0' to it. A static order for variables decision is applied by utilizing an N-input priority encoder, whose inputs are the variables' encoding value free bits. When there are no more free variables, the SAT problem is satisfiable. When a conflict signal is raised, the controller initiates backtracking. In the backtrack state, the controller backtracks nonchronologically to the variable whose index is determined by the backtracking level. It tries its complementary variable, assigning '1' to it. If it has been already tried then it frees the variable. Fig. 3 sketches an example on a simple SAT CNF formula. The way we accumulate conflicting variables using a simple register may result in backtracking to a variable that is not related to the latest conflict leading to re-exploring some search space. A more complex structure can be used but on the account of more hardware resources and lower clock frequency. The formula is unsatisfiable when both the controller tries to backtrack and the NoBck signal is active.

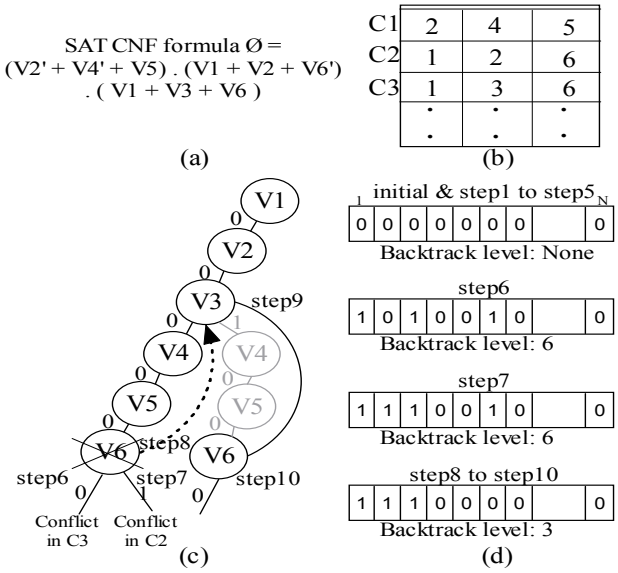


**Fig. 3. Example on conflict analysis: (a) SAT CNF formula. (b) Part of Mem_ClauseDB. (c) Explored search space. (d) Trace of conflicting variables accumulator and backtrack level determined by N-input priority encoder.**

The only compiled hardware circuit for our proposed SAT solver utilizes the resources of the used FPGA to support the maximum number of variables and clauses allowed by the FPGA capacity. One extra memory word in Mem_ClauseDB is used to carry problem instance number of variables N. There is no need to specify problem instance number of clauses M, since all clauses are evaluated simultaneously and extra clauses keep their values intact.

We developed a configuration generator in C++ to map different problem SAT instances to our solver. For different instances, only the data to be stored in FPGA block RAMs (BRAMs), constituting Mem_ClauseDB and Mem_VEOC, need to be changed. For this purpose, we use Data2MEM [13], a Xilinx utility that can be executed to update the FPGA configuration bitstream file with the BRAM initialization data. Besides a .bit bitstream file, Data2MEM requires a .bmm BRAM definition file detailing the placement information of used BRAMs and a .mem memory file (simple text file) containing BRAM initialization data. The total time required to solve a problem instance is:

$$T_{total} = t_{hw} + t_{config},$$

where the hardware time $t_{hw}$ is the time for solving the problem. The configuration time $t_{config}$ is equal to:

$$t_{config} = t_{parsing} + t_{Data2Mem} + t_{download},$$

where $t_{parsing}$ is the time of reading the SAT CNF file, converting it to 3-SAT by introducing new variables and clauses, and generating memory initialization data file, $t_{Data2Mem}$ is the time for updating the bitstream file with memory initialization data using Data2MEM, and $t_{download}$ is the time to download modified bitstream to hardware. The configuration process takes few seconds. During the gen-

eration of the BRAM initialization file, we use a greedy algorithm as a static preordered branching heuristic. Aiming at satisfying the largest number of clauses, variables are ordered according to their number of occurrences in clauses in a descending order. A variable whose positive literal appears more than its negative literal is substituted by its negation since our solver tries the '0' value first. Problem instances larger than the available FPGA capability can be decomposed into independent sub-formulas and then solved one sub-formula at a time [14]. Sub-formulas are solved in any order. If one turns out to be unsatisfiable then the whole problem is unsatisfiable, stopping any further processing.

## 6. Experimental Results

For the implementation of our SAT Solver, we used Memec Design Virtex-II Pro™ development board connected to host PC via parallel cable running at 5 MHz. The implemented hardware circuit supports up to 100 variables and 200 clauses, runs at 65 MHz, and occupies 2574 slices of XC2VP4, i.e., 85% of available slices. The SAT solver configuration generator is executed on Intel Pentium M/1.86 GHz/1 GB running Windows XP.

Fig. 4 shows the total solving time for hole6 (42 variables – 133 clauses) problem from DIMACS benchmarks suite [15] in a comparison between Platzner et al. [6][7], Skliarova and Ferrari software/hardware solver soft/c256 [8], and our SAT Solver. Unlike other SAT solvers, ours accepts only 3-SAT formula. The configuration generator transforms hole6 original CNF formula into 3-CNF of 63 variables and 154 clauses. Table 2 presents hardware cost in terms of occupied 4 input LUTs, execution clock frequency, and raw hardware execution time. As a reconfigurable resource, [6][7] used Digital PCI Pamette board equipped with 4 Xilinx XC4020 FPGAs. The design environment, based on PAM-Blox, used for generating instance-specific circuit was executed on Pentium-II/300MHz/128MB running Windows NT 4.0. The hardware of [8] was implemented on an ADM-XRC board equipped with one XCV812E FPGA connected to the host computer via PCI bus. The software part was executed on an AMD Athlon/1 GHz/256 MB running Windows 2000.

The hardware compilation time dominates the total problem solving time in [6][7] instance-specific architecture, on the other hand being tailored to this specific problem, low hardware cost and high frequency were achieved. As reported in [7], hole6 utilizes 230 CLBs (a CLB is a slice containing 2 4-LUT and 1 3-LUT). Our 100 variables - 200 clauses SAT solver runs on a comparable frequency. The 1.4x speedup in raw hardware execution time is due to conflict directed backtracking implemented by our solver. For Skliarova and Ferrari architecture, the FPGA configuration time takes about 0.37 s achieving the least total execution time. However, the comparison of configuration time is not exact. The same problem exists in the comparison presented in [8] when comparing their soft/c256 with Platzner et al. The main hurdle is that the configuration time reported is for different software platforms. Moreover, the interface used for sending configuration data to FPGA differs. The hardware cost and running clock frequency are constant for all different instances mapped to either Skliarova and Ferrari solver or ours. Skliarova and Ferrari c256 supports 128 variables and 256 clauses and occupies 5158 slices of XCV812E, as reported in [8], where a slice contains 2 LUT4. A smaller implemented circuit c128 supports 64 variables and 128 clauses and occupies 2848 slice that is 5696 CLBs and can run at 32.858 MHz. Compared to [8], we can support more variables and clauses at a lower hardware cost and higher frequency. We acknowledge that this comparison is intended to reflect hardware resources utilization rather than providing an exact comparison due to differences in used FPGAs internal structures.

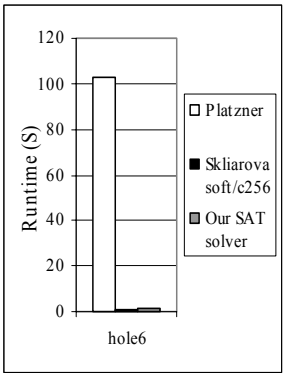


**Fig. 4. Comparing total execution time of the hole6 problem.**

**Table 2. Comparing SAT solvers implemented circuits' parameters and the raw hardware execution time for the hole6 problem. Shaded cells represent constant hardware circuit parameters**

| Architecture | Hardware cost [LUTs] | Clock frequency [Mhz] | Raw HW execution time [S] |
|---|---|---|---|
| Platzner et al. [7] | 460 | 64.935 | 0.005 |
| Skliarova and Ferrari soft/c256 [8] | 10,316 | 30.516 | 0.0131 |
| Our SAT solver | 5,152 | 65.617 | 0.00346 |

Table 3 shows a comparison between our SAT solver and that of Zhong et al. [9] over instances from DIMACS benchmarks suite in terms of number of clock cycles needed to solve the problem. For [9], we consider both cases with and without dynamic clause recording, hardware time, compile time, and total time for solving the problem instance. Their implementation used an array of Xilinx XC4036EX FPGAs. The host computer is an Intel Pentium Pro/200 MHz/128 MB running Windows 2000. Zhong et al. [9] proposed architecture supports clock rates in the range of 30 MHz. Though problem-solving number of cy-

cles of our solver exceeds that of [9], we obtained a speedup since our architecture works at a 2.17X higher clock frequency. On the other hand, our compilation time is smaller, as we just update the FPGA bitstream with memory initialization data directly extracted from SAT formula. As reported in [9], the implemented clause cell occupies 4 X 16 CLBs (a CLB is 1 slice, containing 2 4-LUT and 1 3-LUT). Considering only clauses hardware utilization and neglecting hardware resources occupied by controller and global signals, an implemented circuit handling 200 clauses occupies 25,600 4-LUTs requiring 10 FPGAs of the used Xilinx XC4036EX FPGA. Whereas, the whole circuit of our architecture occupies only 5,152 4-LUTs of the used Xilinx XC2VP4 FPGA. Fig. 5 reflects the difference in hardware cost in terms of 4-input LUTs usage.

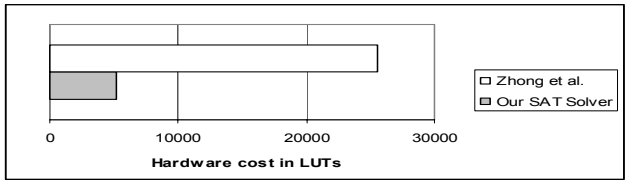


**Fig. 5. Comparing hardware cost in 4-LUTs for 200 clauses SAT solver circuits.**

## 7. Conclusions

We presented an FPGA based clause evaluator, where each SAT CNF clause is modeled as a shift register whose operation is determined by the current assigned variable's value. For different problem instances, effect of its variables' values on the evaluation of its Boolean formula is loaded in the FPGA on-chip memory eliminating configuration overhead. Instead of designing complex functional units to operate on a given problem data mainly represented as literals associated with each clause, stored data controls the operation of simple functional units, resulting in lower hardware cost and higher clock rates. We used a new approach for realizing conflict analysis based on a conflicting variables accumulator and a priority encoder that determines the backtrack level. We compared our SAT solver with other solvers through instances from DIMACS benchmarks suite. The simplicity of our architecture enables achieving higher clock rates and less resources utilization.

## 8. References


[1] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM computing Surveys*, vol. 34, no. 2, pp. 171-210, June 2002.

[2] I. Skliarova and A. B. Ferrari, "Reconfigurable Hardware SAT Solvers: A Survey of Systems," *IEEE Trans. on Computers*, vol. 53, no. 11, pp. 1449-1461, November 2004.

[3] J. P. Marques-Silva and L. Guerra e Silva, "Solving Satisfiability in Combinational Circuits," *IEEE Design and Test of Computers*, pp. 16-21, July-August 2003.

[4] L. Zhang and S. Malik, "The Quest for Efficient Boolean Satisfiability Solvers," *14th Int. Conf. of Computer Aided Verification*, pp. 17-36, July 2002.

[5] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," *IEEE Trans. on Computers*, vol. 48, no. 5, pp. 506-521, May 1999.

[6] M. Platzner and G. De Micheli, "Acceleration of Satisfiability Algorithms by Reconfigurable Hardware," *Proc. of the Int. Workshop on Field-Programmable Logic and Applications, Springer-Verlag, Berlin,* pp. 69- 78, 1998.

[7] O. Mencer and M. Platzner, "Dynamic Circuit Generation for Boolean Satisfiability in an Object-Oriented Design Environment," *Proc. of the 32th Hawaiian Int. Conf. on System Sciences*, Los Alamitos, CA, 1999.

[8] I. Skliarova and A. B. Ferrari, "A Software/Reconfigurable Hardware SAT Solver," *IEEE Trans. on Very Large Scale Integration Systems,* vol. 12, no. 4, pp. 408-419, April 2004.

[9] P. Zhong, M. Martonosi, and P. Ashar. "FPGA-based SAT Solver Architecture with Near-zero Synthesis and Layout Overhead," *IEE Proc. Computer and Digital Techniques,* vol. 147, no. 3, pp. 135-141, May 2000.

[10] A. Dandalis and V.K. Prasanna, "Run-Time Performance Optimization of an FPGA-Based Deduction Engine for SAT Solvers," *ACM Trans. Design Automation of Electronic Systems*, vol. 7, no. 4, pp. 547-562, October 2002.

[11] N.A. Reis and J.T. de Sousa, "On Implementing a Configware/Software SAT Solver," *10th IEEE Symp. on Field-Programmable Custom Computing Machines*, pp. 282-283, 2002.

[12] M. Safar, M. W. El-Kharashi, A. Salem, "FPGA based Accelerator for 3-SAT Clauses Conflict Analysis in SAT Solvers," *13th Advanced Research Working Conf., Charme 2005*, pp. 384-387, Saarbrucken – Germany, October 2005.

[13] Data2Mem Xilinx memory tool, http://www.xilinx.com.

[14] M. Abramovici and J.T. de Sousa, "A SAT Solver Using Reconfigurable Hardware and Virtual Logic," *J. of Automated Reasoning*, vol. 24, no. 1-2, pp. 5-36, 2000.

[15] DIMACS challenge benchmarks, ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/benchmarks/cnf [online].


**Table 3. Comparison with Zhong et al. [9] SAT solver**

| Instance | Zhong et. al SAT solver [9] | | | | | Our SAT Solver | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Run time in cycles: No added clauses | Run time in cycles: Added clauses | HW run time [S] | Compile time [S] | Total time [S] | Run time in cycles | HW run time [S] | Compile time [S] | Total time [S] |
| aim-50-2_0-yes1-2 | 7151 | 3583 | 0.0004 | 1.9 | 1.9004 | 1351807 | 0.0208 | 1.18 | 2.4008 |
| pret60-40 | 181611844 | 341338 | 9 | 2.3 | 11.3 | 484305862 | 0.8 | 1.18 | 3.18 |