FPGA-Based Hardware Acceleration for Boolean Satisfiability

KANUPRIYA GULATI
Texas A & M University
SUGANTH PAUL
Intel, Inc.
SUNIL P. KHATRI
Texas A & M University
and
SRINIVAS PATIL and ABHIJIT JAS
Intel, Inc.

We present an FPGA-based hardware solution to the Boolean satisfiability (SAT) problem, with the main goals of scalability and speedup. In our approach the traversal of the implication graph as well as conflict clause generation are performed in hardware, in parallel. The experimental results and their analysis, along with the performance models are discussed. We show that an order of magnitude improvement in runtime can be obtained over MiniSAT (the best-in-class software based approach) by using a Virtex-4 (XC4VFX140) FPGA device. The resulting system can handle instances with as many as 10K variables and 280K clauses.

Categories and Subject Descriptors: B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Algorithms implemented in hardware*

General Terms: Performance, Algorithms, Design

Additional Key Words and Phrases: Boolean satisfiabilty (SAT), boolean constant propagation (BCP), conflict induced clauses, non-chronological backtrack, FPGA

ACM Reference Format:

Gulati, K., Paul, S., Khatri, S. P., Patil, S., and Jas, A. 2009. FPGA-based hardware acceleration for boolean satisfiability. ACM Trans. Des. Autom. Elect. Syst., 14, 2, Article 33 (March 2009), 11 pages, DOI = 10.1145/1497561.1497576 http://doi.acm.org/10.1145/1497561.1497576

1. INTRODUCTION

Boolean Satisfiability (SAT) [Cook 1971] is a classic NP-complete problem, which has been widely studied in the past. Given a set V of variables, and

Authors' address: S. P. Khatri; email: sunil@ece.tamu.edu.

DOI 10.1145/1497561.1497576 http://doi.acm.org/10.1145/1497561.1497576

a collection C of Conjunctive Normal Form (CNF) clauses over V, the SAT problem consists of determining if there is a satisfying truth assignment for C. Given the broad applicability of the problem to several diverse application domains [Gu et al. 1997], there has been much effort devoted to devising efficient heuristics to solve SAT. Some of the more well-known software approaches include Silva and Sakallah [1996], Moskewicz et al. [2001], and Goldberg and Novikov [2002]. There has been much interest in the hardware implementation of SAT solvers as well. An excellent survey of existing hardware approaches to solve the SAT problem is found in Skliarova and Ferrari [2004a].

In this article, we propose an FPGA based approach to accelerate the SAT solution process, with the goal of speedily solving large instances in a *scalable* fashion. By scalable, we mean that the same platform can be easily made to work on larger SAT instances. The hardware implements the GRASP [Silva and Sakallah 1996] strategy of nonchronological backtracking. In our approach, clauses of fixed width are implemented in the FPGA. The SAT problem is mapped to this architecture in an initial partitioning step which helps maximize the hardware utilization. The FPGA can accommodate a fixed number of clauses on a fixed number of variables. This requires that the original SAT instance be partitioned into *bins* each of which can fit into the FPGA. Further, inter-bin (as well as intra-bin) nonchronological backtrack is implemented in our approach. Experimental results are obtained using LUT utilization and performance figures derived from an actual implementation using an XC2VP30 based FPGA platform. Our hardware approach performs, in parallel, both the tasks of implicit traversal of the implication graph, as well as conflict clause generation. The contribution of this work is to come up with a high capacity, fast, scalable hardware SAT approach. We do not claim to propose any new SAT solution heuristics in this paper. Note that although we used the BCP engine of GRASP in our hardware SAT solver, the hardware approach can be modified to implement other BCP engines as well. The BCP logic of any BCP based SAT solver can be ported to an Hardware Description Language (HDL) and directly synthesized in our approach.

Our approach is implemented and tested on a Xilinx Virtex II Pro evaluation board. The results from these experiments are projected to an industrial strength FPGA system, and indicate a $17\times$ speedup over the best-in-class software approach. The resulting system can handle instances with as many as 10K variables and 280K clauses.

2. PREVIOUS WORK

Several hardware-based SAT solvers have been reported in the past. We classify them into *instance specific* and *application specific* approaches. In instance specific approaches the hardware is recompiled for every instance. This is a key limitation, since compilation times for an FPGA can take several hours. Approaches that are not instance specific are application specific.

Instance specific approaches are reported in the literature [Zhao et al. 2001; Pagarani et al. 2000; Suyama et al. 2001; Abramovici et al. 1999; Platzner and Micheli 1998]. Among these approaches, as reported in Pagarani et al. [2000], the largest example that can be handled has about 1300 clauses with an average speedup of 10×. Our approach, in contrast, is application specific and thus the same device, once configured, can be used multiple times for different instances. Further, we show 17× speedup over the best-in-class software approach and a capacity of accommodating instances with upto 10K variables and 280K clauses.

Some existing application specific hardware or reconfigurable approaches run into the problem of an instance not fitting in a single FPGA or reconfigurable device. The approach of Mencer and Platzner [1999] and Platzner and Micheli [1998] implements the prototype on a Pamette board containing four Xilinx XC4028 FPGAs. These approaches do not propose anything for solving problem instances whose size exceeds the template dimensions. The application-specific approach in Redekopp and Dandalis [2000], like Suyama et al. [2001] and Zhong et al. [1998], employs several interlinked FPGAs, but assume that FPGA resources are sufficient for solving a SAT instance. Also, the runtimes they reported were achieved with the aid of a software simulator.

There are some application-specific approaches which can handle instances that do not fit in a single FPGA device. The approaches described in Safar et al. [2006, 2007], like our approach, are single FPGA approaches. However, in these approaches, the memory module storing the instance has to be reconfigured for different problem instances (or independent subinstances for large instances). Their authors do not clarify the procedure followed when independent subinstances of feasible sizes cannot be obtained. The consistency of assignments across subinstances is not trivial to maintain in hardware, but this is not addressed. Our approach maintains this consistency, and backtracks to a previous partition nonchronologically, in case the offending decision was not made in the current subinstance. The approach of [Skliarova and Ferrari 2004b] creates a matrix (where rows are clauses and column are variables) from the problem instance, and searches for a ternary vector orthogonal to every row, in order to satisfy the instance. For larger instances, it attempts at solving it in software, until the subinstance size is accommodable in the FPGA resources. In our approach, software is used only for the initial partitioning and clause transfer, thereafter, all steps are performed entirely in hardware. Further, the speedups reported in this paper against GRASP are nominal and only for the *holex* benchmarks. Our approach reports speed up against MiniSAT,[1] which is known to be significantly faster than GRASP, and over a variety of benchmarks.

A custom ASIC implementation of a Boolean satisfiability solver was presented in Gulati et al. [2008]. Our current approach is an FPGA version of Gulati et al. [2008]. However, the custom ASIC approach solves the entire instance in a monolithic fashion. Our FPGA-based approach, on the other hand, partitions a CNF instance into bins and is required to maintain consistency in assignments, across all bins efficiently. This requires several changes in the preprocessing step, the hardware design and the overall flow.

[1]Een and Sorensson, http://www.cs.chalmers.se/cs/research/formalmethods/MiniSAT/main.html.

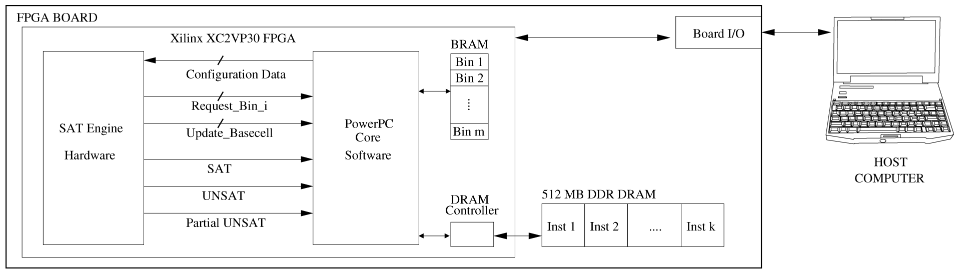

Fig. 1. Hardware architecture.

3. HARDWARE ARCHITECTURE

Figure 1 shows the hardware architecture of our approach. We use an FPGA board that has a duplex communication link with the host system. The FPGA is first loaded with the configuration information for our SAT engine. No instance information is loaded at this stage. Since most practical-sized CNF instances would not readily fit on the FPGA fabric, we heuristically partition the CNF into smaller CNFs, called *bins*, such that the number of common variables across bins are reduced. Also, each of these bins are sized such that they can individually fit in the FPGA fabric. This partitioning is performed as a pre-processing step on the host system, before loading the bins to the FPGA board. In reality, multiple CNF instances (in their respective partitioned bin formats) are stored in a 512 MB DDR DRAM memory card, which is on the FPGA board. A number of partitioned CNF instances are first loaded onto the on-board DRAM from the host system using the board I/O. Next, the bins of one of these CNF instances are loaded in the on-chip BRAM. This is the instance which is being currently processed, and we refer to it as the *current instance* in the sequel. Note that bins can potentially be cached in the BRAM, enhancing scalability. The FPGA is then loaded with any one of the bins of the current instance. This is done using an embedded PowerPC processor, which transfers the bin data from the BRAM to the FPGA fabric. The on-chip PowerPC manages both the loading of the current instance from the DRAM to the BRAM, and the loading/unloading of bins from BRAM onto the FPGA (as dictated by the hardware). These transfers are performed by the Xilinx provided bus transfer protocols over the processor local bus (PLB) and on-chip peripheral bus (OPB). Now for this bin, the FPGA starts to perform implication and conflict clause generation in parallel. The next section discusses our approach of solving a CNF instance which is partitioned across several bins, in our FPGA based hardware SAT solver.

4. SOLVING A CNF INSTANCE WHICH IS PARTITIONED INTO SEVERAL BINS

We partition the original CNF instance $\mathcal{C}$ into smaller bins, b_1 b_2 .... b_n. Our hardware engine tries to satisfy each bin b_i, using the stored global assignments on the variables. The variables $\mathcal{V}$ of the CNF $\mathcal{C}$ are statically awarded a decision level once the bins have been created. Along with each bin b_i, we load the decision levels of the variables $V_i \subseteq \mathcal{V}$ it contains, along with the current *state*

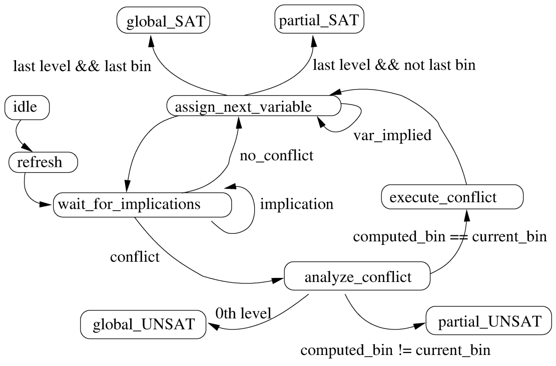

Fig. 2. State diagram of the decision engine.

of every variable $v \in V_i$. The global state of all variables $\mathcal{V}$ is stored in the on-chip BRAM. The state of a variable consists of the following information:

—Whether the variable has been decided (assigned or implied).
—The current decision on the variable.
—If the variable has been decided, the decision level it was decided at.
—If the variable has been decided, the bin it was decided in.
—If the decision on this variable is the highest decision level we backtracked on.

Our FPGA based SAT solver is based partly on our custom ASIC approach presented in Gulati et al. [2008], with significant changes in the hardware, decision engine, and overall flow. These changes are induced by our need to partition the SAT instance into bins.

Figure 2 shows the state machine of the decision engine. To begin with, the first bin of the current CNF instance is loaded onto the hardware. All signals are initialized to their *refresh* state. The decision engine assigns the variables (in the *assign_next_variable* state) in the order of their *identification tag*, which is a numerical ID for each variable, statically assigned such that most commonly occurring variables are assigned a lower tag. The assignments are forwarded to all the clauses of the bin. The decision engine then waits for the bin to compute all the implications during the *wait_for_implications* state. For bins other than the first bin, the decision engine directly enters the *wait_for_implications* state, and propagates any existing decisions on any of the variables of this bin, in the ascending order of their decision levels. All variables implied due to existing (new) assignments, store the decision level of the existing (new) assignment due to which they were implied.

All implied variables store the current bin number in their state bits. When an assignment is made, if no conflict is generated due to the assignment, the decision engine assigns the next unassigned variable in the current bin. If the next unassigned variable v does not occur in any of the clauses of the current bin, or all clauses containing v are already satisfied, the decision engine skips an assignment on this variable and proceeds to the next variable. If all the clauses of the current bin b_i are satisfied and there are no conflicts, then b_i is declared to be *partial_SAT*. A new bin, b_{i+1}, is loaded on to the FPGA along with the states its related variables. If the last bin is *partial_SAT*, the given CNF instance is declared to be *global_SAT* or *satisfiable*.

If there is a conflict in b_i, all the variables participating in the conflict clause are communicated by the clauses in the bin, to the decision engine. Based on this information, during the *analyze_conflict* state, the conflict induced clause is generated and stored in the FPGA fabric, just like a regular clause. Also the decision engine non-chronologically backtracks according to the GRASP algorithm. Using the information contained in the state of a variable, the engine can compute the latest assignment among the variables participating in the conflict, and the bin (*backtrack* bin) where the assignment on this variable was made. When the backtrack bin is the current bin, and the backtrack level is lower than a variable's stored decision level, then the stored decision level is cleared before further action by the decision engine during the *execute_conflict* state. When the backtrack bin is not the current bin, the decision engine goes to the *partial_UNSAT* state, causing the required bin to be loaded. After a conflict is analyzed, the backtracked decision is applied. The variable to be backtracked on is flagged with this information. At any given instance, only the flag of the lowest indexed variable is recorded. If a backtrack has been requested on every variable involved in a conflict, and a conflict exists even by backtracking on the earliest decision, the given CNF is declared as *global_UNSAT* or *unsatisfiable*. Our FPGA based SAT solver is a GRASP based algorithm with static selection of decision variables. Just like GRASP, it performs nonchronological backtracking and dynamic addition of conflict induced clauses. As a result, it retains (within as well as across bins) the completeness property of GRASP. Each time a bin is loaded onto the FPGA, we say that the bin has been *touched*, in the sequel.

5. PARTITIONING THE CNF INSTANCE

To partition a given CNF instance into multiple bins of bounded size (which can fit in the FPGA fabric) we use a 2-dimensional graph bandwidth minimization algorithm, followed by greedy bin-packing.

Algorithm 1. Pseudocode of Bandwidth Minimization

```
Best_Cost = Infinity
for i = 1; i ≤ Number of iterations; i++ do
  Compute Gravity of all clauses and rearrange clauses in increasing order of gravity
  Compute Gravity of all variables and rearrange variables in increasing order of
    gravity
  Perform greedy bin packing for creating bins and compute cost of current
    arrangement $Cost_i$
  if ($Best\_Cost \geq Cost_i$) then
    $Best\_Cost = Cost_i$ ; Store current arrangement
  end if
end for
return (Stored Arrangement)
```

Let us view the CNF instance as a matrix whose columns are the variables, and rows are clauses. The bandwidth minimization algorithm attempts to diagonalize this matrix. For each clause C_i, we assign it a *gravity* $G(C_i)$ which is computed as follows: $G(C_i) = \sum_{C_j \in R(C_i)} (P(C_j) \cdot S(C_i, C_j))$ Here, $R(C_i)$ is the set of clauses that have at least one variable common with clause C_i, $P(C_j)$ is the index of the current row of C_j and $S(C_i, C_j)$ is the number of common variables between clauses C_i and C_j. The exact dual is used for computing the gravity of every variable in the CNF instance. The pseudocode of the bandwidth minimization algorithm is shown in Algorithm 1. With every rearrangement of clauses and variables in an increasing order of gravity, we compute a new cost. The cost of the arrangement is the equally weighted sum of number of bins, the sum (for all variables v in the instance) of the number of bins in which v occurs and the sum (for all variables v in the instance) of the number of bins v *spans*. By *span* we denote the difference in the smallest and the largest bin number, in which v occurs. The greedy bin packing step simply packs the rearranged CNF instance into bins which have a predetermined maximum number of clauses C_{max} and variables V_{max} (such that any bin can fit monolithically in the FPGA fabric). We take $k \leq C_{max}$ clauses and assign them to a new bin provided the variable support of these clauses is less than or equal to V_{max}.

6. EXPERIMENTAL RESULTS

Our working system is implemented on a low-end FPGA evaluation board. In order to obtain projected performance numbers on a high-end FPGA board, we first extract detailed performance data from our system. Using this data, we develop a mathematical performance model, in Section 6.2. This model estimates the bin size, numbers of bins touched, and communication speeds as a function of FPGA size. Using this performance model, we project the system performance (using our existing performance data) for industrial strength FPGA boards, in Section 6.3.

6.1 Current Implementation

To validate our approach, we have implemented our hardware SAT solver on a Xilinx XC2VP30 device based evaluation board using ISE 8.2i for hardware (Verilog) and EDK 8.2i for instantiating the PowerPC, processor local bus (PLB), on-chip peripheral bus (OPB), BRAM and PLB2OPB bridge cores. Our current implementation can solve CNF instances of size 8K variables and 31K clauses. If we were to cache the bins in BRAM, then the capacity of the system increases to 77K clauses over 8K variables. The size of a single bin is 16 variables and 24 clauses. The FPGA device utilization with this configuration, including the EDK cores, is ~70%. With larger FPGAs, significantly larger CNFs can be tackled. Our current implementation correctly solves several nontrivial CNF instances. Our regression suite consists of about 100,000 satisfiable and unsatisfiable instances. Further, the regression testing of 100's of instances can be performed with a single loading of the DRAM. As mentioned previously, several CNF instances, after being partitioned into bins, are loaded onto the board DRAM. Only the current instance resides completely in the on-chip BRAM.

6.2 Performance Model

Table I reports the various experiments we performed to develop the performance model. Column 1 lists the name of the experiment. Column 2 details the experiment, while Column 3 reports the result drawn from the experiment.

For the experiment to compute Clauses/Variable Ratio, we define *bin utilization* as follows: if a single bin is viewed as a matrix, with clauses for rows and variables for columns, bin utilization is the ratio of the number of filled matrix entries to the total available matrix entries.

We also studied the distribution of the LUTs in XC2VP30 and XC4VFX140 FPGAs over portions of our design that scale with bin size (clauses of the bin) and also those portions of the design that do not scale (Xilinx Cores for DDR, clocking (DCM), PowerPC, BRAM, PLB, OPB and finite state machine for the decision engine) with bin size. The non-scaling (scaling) parts are those for which the LUT utilization does not (does) increase while increasing the bin size. In XC2VP30, out of the total 30K available LUTS, and assuming a 70% device utilization, only 14K LUTs can be used for storing the clauses of the bin. In case of the XC4VFX140 FPGA, about 91K LUTs can be used for the clauses of the bin. We know that the number of LUTs of *Device* utilized for clauses of the bin is $300{\cdot}V + 20{\cdot}V \cdot C$. Since $C = \frac{2}{3} \cdot V$ based on the golden ratio A_g, we have $300{\cdot}V + 20{\cdot}\frac{2}{3} \cdot V^2 =$ Available LUTs in *Device*. Solving this quadratic equation for V gives us the size of the bin (V, C) that can be accommodated in any FPGA device. V and C for the XC2VP30 device are 16 and 10 respectively, and for the XC4VFX140 device, they are 75 and 50 respectively. These calculated bin sizes have been verified by synthesizing the design netlist generated for these bin sizes using the Xilinx ISE 8.2i tool, for the corresponding device.

6.3 Projections

From the performance models in Section 6.2, we can project the system performance for a Xilinx XC4VFX140 device (over the current implementation on the XC2VP30 device) as follows:

—Number of bins in the design are projected to grow as $\frac{V_{XC2VP30}}{V_{Device}}$, since the number of bins required for a CNF instance is inversely proportional to the bin size.

—Number of bins touched grows as $\frac{V_{XC2VP30}}{V_{Device}}$, since the number of bins touched is inversely proportional to bin size, which is proportional to the number of variables in a bin.

—Software (PowerPC) runtimes improve as: $\frac{F_{Device}}{F_{XC2VP30}} \cdot \frac{V_{Device}}{V_{XC2VP30}} \cdot 50$. This is because software runtime is inversely proportional to the device frequency. Further, if the number of bins touched is reduced, the number of bin transfers directed by the PowerPC are reduced proportionately. The bus transfer rate using Xilinx bus transfer protocols (for PLB and OPB cores) is about 50 cycles per word in our current implementation. This transfer rate can be reduced to 1 cycle per word, by writing a custom bus transfer protocol.

Table I. Experiments to Obtain Performance Model

Exp. Name	Details	Result
FPGA Resources	Conducted several synthesis runs, using different bin sizes to quantify the FPGA resource utilization as a function of # of variables V and # of clauses C	LUT utilization $= 20{\cdot}V \cdot C + 300{\cdot}V$
Clauses/Variable Ratio	Conducted experiments to find the golden ratio ($A_g = \frac{V}{C}$) in a bin. For 20 instances performed several binning runs, and then for a given V we varied C. Finally, plotted μ, $\mu + \sigma$ and $\mu - \sigma$ of bin utilization	For 60% bin utilization, $A_g = \sim \frac{2}{3}$
Cycles v/s Bin Size	Conducted experiments to obtain the # of hardware cycles required (I) as a function of bin size ($V \cdot C$) which is inversely proportional to the # of bins m. Varied both V and C ($C = A_g \cdot V$)	I ($\sim$125 cycles) per bin was found to be roughly constant for different (V, $A_g \cdot V$) values because if $V \cdot C$ is large, I decreases since # of bins touched decreases
Bins Touched v/s Bin Size	Ran several instances and recorded the backtracks required for completely solving them. For a given bin size, we simulated whether each of these backtracks would have resulted in a new bin being touched	# of bins touched reduces *linearly* with an increase in bin size

—Hardware (Verilog) runtimes improve as: $\frac{F_{Device}}{F_{XC2VP30}} \cdot \frac{V_{Device}}{V_{XC2VP30}} \cdot \frac{(\frac{cycles}{bin})_{XC2VP30}}{(\frac{cycles}{bin})_{Device}}$. This is since hardware runtime is inversely proportional to the device frequency. Also if the number of bins touched is reduced, the total number of hardware cycles required for solving the instance are reduced proportionately. Finally, the total number of hardware cycles required is proportional to the number of cycles required to solve a single bin.

Using the above expressions for the scaling of the hardware and software runtimes, the projected runtimes for a XC4VFX140 based system are shown in Table II. Note that the results in Table II are obtained by taking the actual hardware and software runtimes of our XC2VP30 based platform, and projecting these numbers to a industry strength XC4VFX140 based platform. Column 1 lists the instance name, and Columns 2 and 3 list the number of variables and clauses, respectively, in the instance. Column 4 lists the number of bins obtained after the CNF partitioning is performed on the host machine. Column 5 lists the number of bins touched by the XC4VFX140-based hardware for solving this instance. The runtimes (in seconds) are listed in Columns 6, 7 and 8. Column 6 reports the software runtime of our approach (i.e., the time taken by the PowerPC at 450 MHz to perform the bin transfers). Column 7 reports the hardware runtime (i.e. hardware runtime over all bins). The runtimes for the pre-processing step are not considered, since they are negligible with respect to the hardware or software runtime. Even if the pre-processing

Table II. Runtime Comparison XC4VFX140 versus MiniSAT

Instance Name	Num. Vars	Num. Cls.	Num Bins	Bins Touched	Time (Sec) PowerPC	Verilog	MiniSAT
mux_u	133	504	13	1	1.24×10^{-8}	1.43×10^{-9}	9.39×10^{-4}
cmb	62	147	4	66	6.90×10^{-6}	2.84×10^{-5}	1.01×10^{-3}
cht_u	647	2,164	48	6	9.32×10^{-7}	1.00×10^{-6}	1.97×10^{-3}
frg1_u	310	2,362	71	1	1.79×10^{-7}	5.02×10^{-7}	1.03×10^{-3}
ttt2_u	874	3,284	84	4	9.24×10^{-7}	6.77×10^{-7}	9.98×10^{-4}
term1_u	1,288	4,288	114	3	1.06×10^{-6}	3.72×10^{-7}	1.99×10^{-3}
x4_u	1,764	5,772	138	12	2.24×10^{-6}	2.67×10^{-6}	2.99×10^{-3}
x3_u	3,301	10,092	257	18	3.84×10^{-6}	3.39×10^{-6}	3.01×10^{-3}
aim-50-2_0-yes1-20	50	100	3	3	2.99×10^{-7}	1.49×10^{-6}	1.84×10^{-4}
holes6	42	133	3	4600	5.00×10^{-4}	2.23×10^{-3}	8.98×10^{-3}
holes8	72	297	5	276751	3.04×10^{-2}	1.21×10^{-1}	1.49
uuf100-0457	100	430	17	43806	4.39×10^{-3}	2.58×10^{-2}	1.90×10^{-2}
uuf125-07	125	538	21	1120471	1.35×10^{-1}	9.03×10^{-1}	2.01×10^{-2}
Geo. Mean					1.31×10^{-5}	2.27×10^{-5}	3.78×10^{-3}

runtimes were higher, the time spent in partitioning the CNF instance is amply recovered when multiple incrementally different SAT calls need to be made for the same instance, which commonly occurs in VLSI CAD based SAT instances. Finally, the last column reports the MiniSAT runtimes obtained on a 3.6 GHz, Pentium IV machine with 3 GB of RAM, running Linux.

Over all test cases, the net speedup over MiniSAT is 90×, and for benchmarks in which more than 4500 bins are touched, the speedup is about 17×. Also, for benchmarks which fit in a single bin, the speedup is 2.85×10^4. We also projected the number of clauses that can be accommodated in the BRAM for the XC4VFX140 based system and found that upto 280K clauses on 10K variables can be accommodated.

7. CONCLUSION AND FUTURE WORK

In this article, we present an FPGA-based approach for Boolean satisfiability, in which the traversal of the implication graph as well as conflict clause generation are performed in hardware, in parallel. Our entire flow has been verified for correctness on a Virtex-II Pro based evaluation platform. We project the runtimes obtained on this platform to an industry strength XC4VFX140 based system, and show that a speed up of 17× can be obtained over the best-in-class software approach. The projected system can handle instances with as many as 280K clauses on 10K variables.

REFERENCES

Abramovici, M., de Sousa, J., and Saab, D. 1999. A massively-parallel easily-scalable satisfiability solver using reconfigurable hardware. In *Design Automation Conference*. 684–690.

Cook, S. 1971. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium Theory of Computing*. 151–158.

Goldberg, E. and Novikov, Y. 2002. BerkMin: A fast and robust SAT solver. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*. 142–149.

Gu, J., Purdom, P., Franco, J., and Wah, B. 1997. Algorithms for the satisfiability (SAT) problem: A survey. In *Discrete Math. and Theoretical Computer Science*, DIMACS, Rutgers, NJ, 19–151.

Gulati, K., Waghmode, M., Khatri, S., and Shi, W. 2008. Efficient, scalable hardware engine for Boolean satisfiability and unsatisfiable core extraction. *IET Comput. Digit. Teclin. 2,* 3, 214–229.

Mencer, O. and Platzner, M. 1999. Dynamic circuit generation for Boolean satisfiability in an object-oriented design environment. In *Proceedings of the 32nd Annual Haubiiau International Conference on System Sciences*. 3044–3052.

Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference*. 530–535.

Pagarani, T., Kocan, F., Saab, D., and Abraham, J. 2000. Parallel and scalable architecture for solving Satisfiability on reconfigurable FPGA. In *Proceedings of the Custom Integrated Circuit Conference*. 147–150.

Platzner, M. and Micheli, G. D. 1998. Acceleration of satisfiability algorithms by reconfigurable hardware. In *Field-Programmable Logic and Applications*, Springer-Verlag, 69–78.

Redekopp, M. and Dandalis, A. 2000. A parallel pipelined SAT solver for FPGAs. In *Proceedings of the Field Programmable Logic*. Springer-Verlag, 462–468.

Safar, M., El-Kharashi, M., and Salem, A. 2006. FPGA-based SAT solver. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering*. 1901–1904.

Safar, M., Shalan, M., El-Kharashi, M. W., and Salem, A. 2007. Interactive presentation: A shift register based clause evaluator for reconfigurable SAT solver. In *Proceedings of the Conference and Exhibition on Design, Automation and Testing Europe (DATE)*. 153–158.

Silva, M. and Sakallah, J. 1996. GRASP-a new search algorithm for satisfiability. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. 220–7.

Skliarova, I. and Ferrari, A. 2004a. Reconfigurable hardware SAT solvers: A survey of systems. *IEEE Trans. Comput. 53,* 11, 1449–1461.

Skliarova, I. and Ferrari, A. B. 2004b. A software/reconfigurable hardware SAT solver. *IEEE Trans. VLSI Syst. 12,* 4, 408–419.

Suyama, T., Yokoo, M., Sawada, H., and Nagoya, A. 2001. Solving satisfiability problems using reconfigurable computing. *IEEE Trans. VLSI Syst. 9,* 1 (Feb), 109–116.

Zhao, Y., Malik, S., Wang, A., Moskewicz, M., and Madigan, C. 2001. Matching architecture to application via configurable processors: A case study with Boolean Satisfiability problem. In *Proceedings of the International Conference on Computer Design (ICCD)*. 447–452.

Zhong, P., Martonosi, M., Ashar, P., and Malik, S. 1998. Accelerating Boolean Satisfiability with configurable hardware. In *FPGAs for Custom Computing Machines*, 186–195.

Received September 2007; revised June 2008; accepted November 2008