Real-time Reconfigurable Hardware WSAT Variants

Roland Yap, Stella Wang, and Martin Henz
School of Computing, National University of Singapore
Singapore
{ryap,wangzhan,henz}@comp.nus.edu.sg

Abstract. Local search methods such as WSAT have proven to be successful for solving SAT problems. In this paper, we propose two real-time host-FPGA (Field Programmable Gate Array) co-implementations, which use modified WSAT algorithms to solve SAT problems. Our implementations are reconfigurable in real-time for different problem instances. On an XCV1000 FPGA chip, SAT problems up to 100 variables and 220 clauses can be solved. The first implementation is based on a random strategy and achieves one flip per clock cycle through the use of pipelining. The second uses a greedy heuristic at the expense of FPGA space consumption, which precludes pipelining. Both of the two implementations avoid re-synthesis, placement, routing for different SAT problems, and show improved performance over previously published reconfigurable SAT implementations on FPGAs.

1 Introduction

Stochastic local search (SLS) algorithms have been successful for solving propositional satisfiability problems (SAT). The WalkSAT family (WSAT) of algorithms [1, 2] contains some of the best performing SLS algorithms. SLS algorithms like WSAT have a very simple structure and are composed of essentially three steps which are iterated until a satisfiable solution is found: (i) evaluate clauses; (ii) choose a variable; and (iii) flip the variable's boolean value.

Since each of the steps is simple, and since the SAT clauses can be directly represented in hardware, it is tempting to build a hardware-based SLS solver. There are a number of such hardware designs and implementations [3, 4, 5, 6] using reconfigurable FPGA hardware. Hardware approaches to systematic search procedures for SAT problems are beyond the scope of this paper; see [7] for an overview.

The use of hardware SAT solver only makes sense if there is significant performance advantage compared to software. Software can make use of state of the art processors built with the latest processor technology. A hardware SAT solver, on the other hand, is less likely to have the same level of process technology, and hence longer cycle times. Earlier hardware implementations like [3, 4] did not outperform optimized software. For example, reimplementation of the design in Hamadi and Merceron [3] which was done in Henz et al. [6] had flip rates between 98 – 962 K flips/s. In some problems, this was a bit faster than software and in other cases slower. In [6], it was shown that GSAT SLS solvers running at one flip per clock cycle was achievable with performance gains of about two orders of magnitude over software. That implementation makes use of the reconfigurable nature of FPGAs to build a custom design specific to a particular SAT problem instance. While [6] show that very large speedups are feasible, this approach is not practical as a general SAT problem solver, because of the time needed to re-synthesize, place and route the new design for an FPGA is likely to significantly exceed the runtime improvement from the faster solver.

This paper explores hardware designs for WSAT, which are not instance-specific and thus do not require re-synthesis. In addition to this requirement, a hardware implementation faces interesting design tradeoffs due to the inherently limited logic resources on the chip. We propose two versions of WSAT, which allow real-time reconfiguration. The differences of the WSAT versions lead to different design choices for maximal performance. The first design emphasizes fast cycle times (one flip per clock cycle), employing random variable selection to allow for a pipelined design. The second uses a greedy variable selection heuristic, which precludes pipelining, exemplifying a tradeoff between flip rate and effectiveness of variable selection. Both designs have improved performance over other published non-re-synthesis SLS FPGA implementations.

2 Hardware Implementation Issues

2.1 Cost of Re-Synthesis FPGA Implementations

SLS SAT algorithms exhibit large amounts of parallelism and hence are a good match for a hardware solver, which can use the large amounts of parallelism available in the hardware. We focus here on WSAT implementations using Field Programmable Gate Arrays (FPGAs), which provide the benefits of customized hardware but avoid fabrication cost, and thus allow for convenient prototyping of the hardware design. Unlike software, a hardware implementation has to deal with the inherent resource limitations for combinatory logic, memory and routing on an FPGA.

One approach is to maximize performance by making full use of parallelism, exemplified in [6], where clause evaluation and variable selection is parallelized for a GSAT SLS implementation. However, such a high degree parallelism is expensive in terms of hardware resources. That implementation optimizes the hardware design specifically for a given SAT problem instance, taking advantage of the reconfigurability of FPGAs. This *instance-specific* approach enabled a performance of one flip per clock cycle, more than two orders of magnitude faster than software. The drawback, however, is that a new solver has to be re-synthesized for each SAT problem instance. With current CAD tools, the synthesis, placement and routing for SAT instances with 200 variables can take several hours, while the resulting SAT solver may only take seconds or minutes to find a solution to the instance. Thus, while instance-specific hardware implementations demonstrate the feasibility of very high performance hardware approaches, they are impractical as general-purpose SAT solvers.

A general-purpose hardware SAT solver should instead not require re-synthesis, and be able to handle different SAT instances with only small overheads. One non-re-synthesis approach is given in [4], which takes advantage of the fact that the FPGA configuration file can be altered directly to modify the design. This provides a shortcut to re-synthesis since only small modifications to the definitions of the SAT clauses are necessary. However, the implementation is mostly sequential and does not outperform optimized software. The more serious problem is that current FPGA chips do not have an open architecture. The configuration file for these chips is a black box, which renders this approach unfeasible.

Leong et al [5] achieved a bitstream reconfigurable FPGA implementation for a WSAT variant. Their implementation stores clauses for a SAT problem in the 16x1-bit ROM available in the Logic Cells (LC) of the Xilinx FPGA. A different SAT instance requires various ROM definitions to be modified. Normally, this would require re-synthesis of the FPGA to generate a new bitstream configuration for downloading. Leong et al were able to achieve a non re-synthesis implementation, using a tool to extract the locations of the relevant LCs in the bitstream, and then directly modify the corresponding data for the ROM values in the bitstream file. This approach requires analysis of the bitstream file to figure out how to rebuild the configuration without re-synthesis.

Both of these implementations [4, 5] simulate re-synthesis in a very efficient fashion. However, they are dependent on the ability to modify the FPGA configuration.

The aim of this paper is to obtain a more portable reconfigurable implementation which nevertheless is capable of providing good performance. We also want the reconfiguration to be quite fast.

2.2 A Non-Re-synthesis Clause Evaluator

The key to avoid re-synthesis is to be able to handle any SAT instance, hence the clause evaluator in WSAT must be general rather than instance-specific. Our goal is a general clause evaluator, which fits well within an FPGA architecture and can be reconfigured quickly in a portable fashion.

We will focus on the Xilinx Virtex FPGA chips. The basic building block of Virtex FPGA [8] is an LC, which includes a 4-input function generator, carry logic and a storage element. The 4-input function generator is implemented as 4-input look-up table (LUT). Each Virtex CLB (Configurable Logic Block) contains four LCs, organized in two slices. Two LUTs in a slice can be combined to create a 16x1-bit dual port RAM. Our clause evaluator represents the clauses in the SAT instance in a 16x1-bit dual port RAM array, which can be generated from the Xilinx RAM16x1D primitive. The Xilinx RAM16x1D primitive is a 16-word by 1-bit static dual port random access memory with synchronous write capability. The device has two separate address ports; the read address port (DPRA3-DPRA0) and the write address port (A3-A0).

We describe the clause evaluator by example. Consider a SAT clause, c_3, of the form, $x_1 \vee x_2 \vee \overline{x_5}$, and let us assume that c_3 is a clause of a SAT problem over 8 variables. The clause can be written as $f_{3,1}(x_1, x_2, x_3, x_4) \vee f_{3,2}(x_5, x_6, x_7, x_8)$, where $f_{3,1}(x_1, x_2, x_3, x_4) = x_1 \vee x_2$ and $f_{3,2}(x_5, x_6, x_7, x_8) = \overline{x_5}$. Thus each SAT clause, c_i, can be decomposed into a disjunction of boolean functions on a smaller number of variables. We map each $f_{i,j}$ arising from the *j-th* part of clause *i* to a RAM16x1D primitive, treating the four variables as the address to the read port (DPRA3-DPRA0). The function $f_{i,j}$ is configured by using the write port (A3-A0) to define its truth table.

Note that one advantage of this representation is that negated variables are handled automatically inside the $f_{i,j}$ block. Figure 1(a) shows an overall block diagram of the reconfigurable clause evaluator for 100 variables and 220 clauses. Figure 1(b) shows each $f_{i,j}$ block, which is configured using the controller in Figure 1(c). The result of each RAM primitive is ORed and stored in the array *all_clause[]*.The clause evaluator evaluates all clauses in parallel in one cycle.

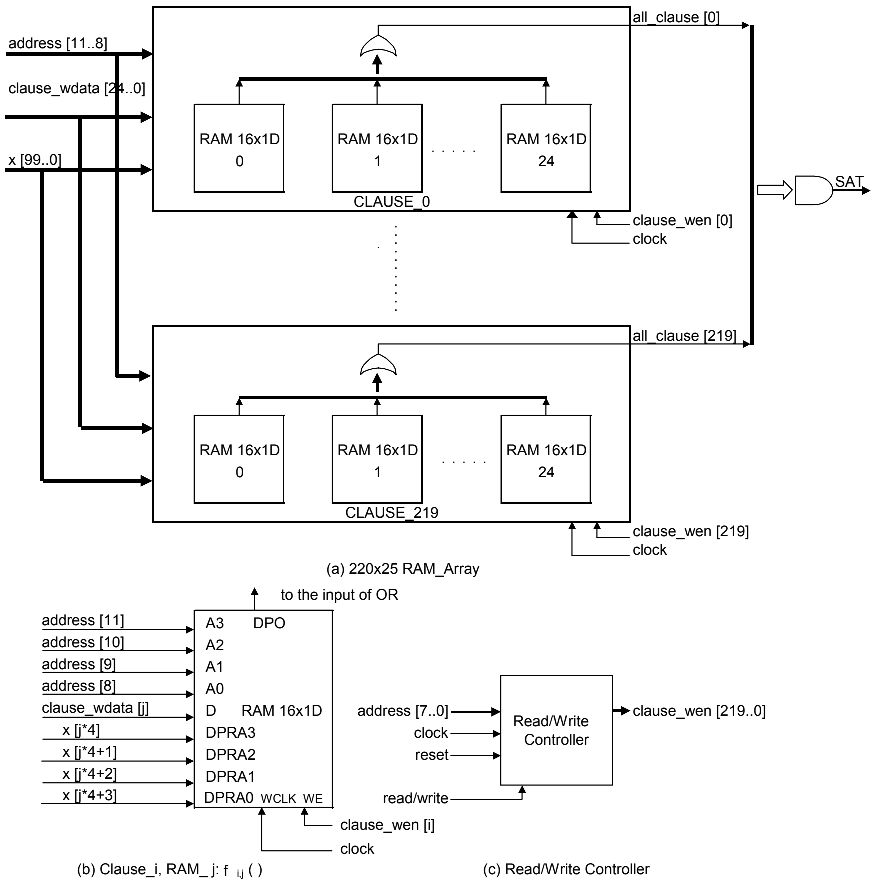

Fig. 1

3 Two Non-Re-Synthesis FPGA Implementations

The reconfigurable clause evaluator requires *O(mn)* CLBs for an implementation with *m* clauses and *n* variables. This component consumes a significant fraction of the available CLBs (as much as 80%). As we would like to be able to handle as large a problem as feasible within the constraints of the FPGA, it is impractical to consider implementations that require multiple clause evaluators. This would consume too much of the chip real estate, even if there is considerable parallelism gain. We present two implementations of WSAT for 3-SAT problems, which represent different tradeoffs in using a single reconfigurable clause evaluator.

3.1 A Pipelined FPGA Implementation Using a Random Selection Heuristic

One strategy is to produce an implementation with a fast cycle time. Given that we are constrained to a single clause evaluator, we are left with pipelining as the only option for increasing the flip rate. For maximal reuse of the clause evaluator, it is important that the pipeline be well balanced with simple pipeline stages. Given that we already have a fully parallel clause evaluator, the most expensive step in WSAT is variable selection. A particularly simple WSAT variant chooses the variable randomly in a selected unsatisfied clause. This strategy is also used in the WSAT implementation of Leong et al [5].

Figure 2 depicts a five-stage pipelined implementation. Stage 1 finds a random unsatisfied clause (this checks all clauses in parallel). Stage 2 generates three variable indices for the selected clause. Stage 3 implements the random selection heuristic, flipping of its input variables. Stage 5 checks for satisfiability. There are a number of storage buffers used. Buffer 1 stores the clause table which gives the mapping of clause to variables used within that clause as represented by a variable index. The SAT problem is initially loaded into buffer 2 which then is used to initialize the $f_{i,j}$ blocks in the clause evaluator. The result is a one flip per cycle implementation.

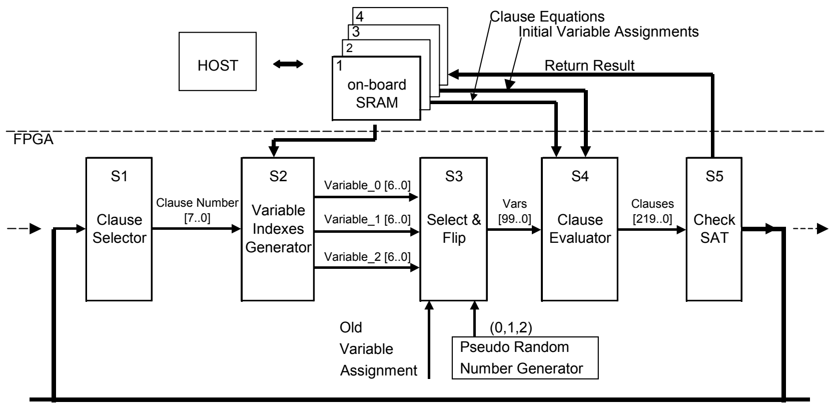

Figure 2 Pipelined Random WSAT

3.2 FPGA Implementation Using a Greedy Selection

A more typical WSAT variable selection heuristic is to select the variable, which best improves the score. In terms of the constraints of the hardware, this corresponds to a design with more complex operations. We have chosen to use a pure greedy heuristic without noise (but a noise component can be easily added).

Figure 3 shows the block diagram of a sequential implementation. Since we are dealing with 3-SAT, it is only necessary to determine at most which of the three variables in a clause to select. However, any kind of parallel implementation of this step would require computing the score of each of the three possibilities. This would require three clause evaluator units, which we deem too space consuming for the targeted SAT problem size.

Thus, we are restricted to a sequential implementation for the variable selection

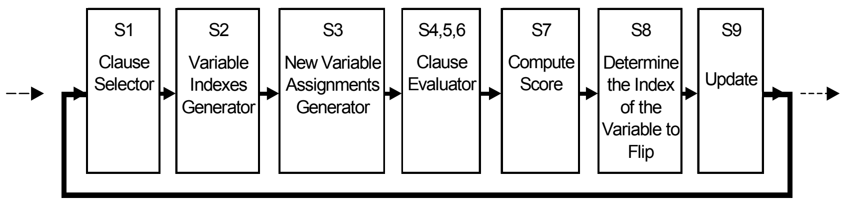
S1
Clause Selector
S2
Variable Indexes Generator
S3
New Variable Assignments Generator
S4,5,6
Clause Evaluator
S7
Compute Score
S8
Determine the Index of the Variable to Flip
S9
Update

Figure 3 Sequential Greedy WSAT

(Stages 4-6), which reduces the flip rate. Our current implementation performs one flip in nine cycles, as opposed to one cycle achieved by the design for random selection heuristic.

4 Results

Our hardware SAT solver is implemented on Celoxica's RC1000-PP standard PCI bus board, which is equipped with a Xilinx XCV1000 FPGA. This board has 8Mb of SRAM directly connected to the FPGA in four 32-bit wide memory banks. Each of the four banks may be granted to either the host CPU or the FPGA at any time. Data can therefore be shared between the FPGA and the host CPU by placing it in the SRAM. It is accessible to the host CPU by DMA transfer across the PCI bus.

As host we use a PC with an AMD Athlon 1.2GHz CPU. Our prototype generates the clause configuration for a new SAT instance in software in about 7ms (this is unoptimized and is probably dominated by file I/O). Transferring the clause configuration from the host PC to the on-board SRAM takes 0.6ms. The FPGA takes 220 x 16 clock cycles to read the SRAM. With an FPGA clock frequency of 20MHz, this corresponds to 0.176ms. Thus the configuration overhead for solving a new SAT instance is 7.776ms.

Table 1. Time/Space Cost Comparison of FPGA-based Implementation

	Random-Strategy WSAT		Greedy-Strategy WSAT	
System Size	Delay (ns)	Cost of Slices	Delay (ns)	Cost of Slices

50-var/170-c	24.097	4946 (40%)	24.842	6408 (52%)
100-var/220-c	31.005	10396 (85%)	31.639	11834 (96%)

The prototype implementations investigate the two designs on two SAT problem sizes; a 50 variable/170 clause format and a 100 variable/220 clause format, the latter chosen to such that its reconfigurable clause evaluator fits on the FPGA used. Table 1 gives the hardware costs in terms of slices for the various implementations. The minimum gate delay is as reported by the Xilinx place and route tools. There is only a small difference in gate delay between the two implementations. The larger influence is the increased delay due to larger problem sizes.

Table 2 shows the flip rate performance comparison for FPGA-based hardware implementations versus software for various 3-SAT benchmarks. The benchmarks shown are simply those, which fit within the required problem sizes. As the main purpose of the benchmarks is to measure flip rate performance, the difficulty of the benchmarks is not relevant. Our FPGA implementations were clocked at 20Mhz. The software WSAT implementation is WalkSAT35 by Bart Selman running on a Pentium2 400Mhz PC. The choice of processor is motivated by the need for a fair technology comparison; the Xilinx XCV1000 chip stems roughly from the same processor generation as this chip.

Table 2. Flip Rate Speedups: FPGA-based versus Software

SAT Problems	Software	FPGA-based Implementations Speedup		
	Flip Rate (Kfps)	Random Speedup		Greedy Speedup
		Our implementation, Pipelined	Leong et al.[5], non-pipelined	
uf20-01	169.8	121	2.14	13
uf50-01	252.4	81	-	9
aim-50-1_6-yes1-1	456.0	82	-	5
aim-50-2_0-yes1-1	516.1	40	0.69	4
aim-50-3_4-yes1-1	326.3	63	1.86	7
aim-100-1_6-yes1-1	457.1	45	-	5
aim-100-2_0-yes1-1	469.4	43	-	5

The flip rate for the random and greedy variable selection heuristics is constant throughout the problems – 20M flips for random, and 2.2M flips for the greedy heuristics, due to its 9-stage pipeline. We also measured actual timings as a reality check. The random and speedup columns represent the ratio of measured flip rate versus the software flip rate. Note that the software flip rate varies with the problem, while it is constant in our implementations.

The fourth column compares our pipelined random strategy with the WSAT reconfigurable FPGA implementation from Leong et al. [5] which also uses a random strategy. Their implementation uses a smaller FPGA with problems of up to 50 variables and hence could be clocked at a faster speed of 33Mhz. The speedup has been recomputed using the average timing results in their paper. Where timings or benchmarks are not available, this is indicated by a (-). A major difference between their implementation and the greedy pipelined one here is that our implementation is based on a constant flip rate. Their implementation, on the other hand, has a variable flip rate, because of the use of sequential clause selection and is bounded by a maximum flip rate of 364Kfps.

With the random variable selection heuristic, the preliminary results show that our reconfigurable FPGA implementation is significantly faster than software and previous hardware implementations. This implementation achieves one flip per clock cycle at 20Mhz. The greedy variable selection implementation has more modest speedups. The speedup is likely comparable to software or slightly faster, if the fastest state of art microprocessors are used, since performance scales at a lower rate with clock speed for microprocessors. However, the reduced flip rate may be offset by the increased effectiveness of the variable selection strategy. The greedy heuristic typically gives a better success rate than a random heuristic for WSAT. A detailed analysis of the effect of different variable selection heuristics is given in [9].

5 Conclusion

We demonstrate two prototype hardware solvers implemented on the Xilinx Virtex XCV1000 FPGA with significantly better performance than software and previous hardware WSAT solvers. Furthermore, the solvers are reconfigurable in real-time, with a reconfiguration time of a few milliseconds for problems with 100 variables. Our two implementations illustrate the tradeoff between time, space and effectiveness of the SLS algorithm. The random solver achieves an optimal flip rate at the cost of a simple variable selection strategy, while the greedy solver uses the more expensive and effective strategy but is not amenable to pipelining and is hence slower.

Both implementations are limited by the size of the Xilinx Vertex XCV1000 chip used, which can accommodate a reconfigurable clause checker only for problems with 100 variables and 220 clauses. This chip, dating from 1999, is fabricated using a 5-layer metal 0.22μm CMOS process. In comparison, the current Virtex-II generation uses an 8-layer 0.15μm CMOS process. The XC2V10000 has about 10 times more system gates than the XCV1000 and has significantly faster clock speeds. For example, a 100 variable/600 clause evaluator requires about 30K slices and fits in a XC2V6000 which has 6M system gates.

An FPGA implementation will have more limitations on problem sizes even when larger FPGAs are used. A fast hardware based solver can however still be useful for general SAT solving. One approach is with hybrid search + stochastic solvers. For example, Zhang et al. [10] combine Davis Putnam with stochastic search. Their approach uses Davis Putnam to generate a smaller sub-problems which are then solved with WSAT.

Another route to deal with larger problems is to use ASICs rather than FPGAs. Our implementation is not restricted to FPGAs since the reconfiguration for different SAT instances is not dependent on the reconfigurable logic of FPGAs. The prototype uses FPGAs simply because they are more cost effective for development. Given the real-time reconfiguration capability, this may be a promising candidate for direct ASIC implementation, which means higher clock speeds and much more resources for dealing with larger problems.

References

[1] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of AAAI-94*, pages 337-343, 1994.

[2] David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *Proceedings Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 1997.

[3] Youssef Hamadi and David Merceron. Reconfigurable architectures: A new vision for optimization problems. In Gert Smolka, editor, *Principles and Practice of Constraint Programming–CP97, Proceedings of the 3rd International Conference,* Lecture Notes in Computer Science 1330, pages 209-221, Springer-Verlag, Berlin, 1997.

[4] Wong Hiu Yung, Yuen Wing Seung, Kin Hong Lee, and Philip Heng Wai Leong. A runtime reconfigurable implementation of GSAT algorithm. In Patrick Lysaght, James Irvine, and Reiner W. Hartenstein, editors, *Field-Programmable Logic and Applications,* pages 526-531. Springer-Verlag, Berlin, 1999.

[5] P. H. W. Leong, C. W. Sham, W. C. Wong, H. Y. Wong, W. S. Yuen, and M. P. Leong. A bistream reconfigurable FPGA implementation of the WSAT algorithm. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* 9(1): 197-200, 2001.

[6] Martin Henz, Edgar Tan, Roland Yap. One flip per clock cycle. *In Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming, CP2001,* Cyprus, Nov/Dec 2001.

[7] M. Abramovici and A. Sousa. A SAT solver using reconfigurable hardware and virtual logic, Journal of Automated Reasoning 24(1/2): 5-36, 2000.

[8] Xilinx. Virtex 2.5 Field programmable gate arrays, 1999.

[9] H. Hoos and T. Stützle. Local search algorithms for SAT: An empirical evaluation Journal of Automated Reasoning, 24:421-481, 2000.

[10] Wenhui Zhang, Zhuo Huang, Jian Zhang. Parallel Execution of Stochastic Search Procedures on Reduced SAT Instances*. In Proceedings of the Seventh Pacific Rim International Conference on Artificial Intelligence, PRICAI 2002*, Tokyo, Japan, August 18-22, 2002, Lecture Notes in Computer Science 2417:108-117. Springer-Verlag. 2002.