Hardware/Software Co-modeling of SAT Solver Based on Distributed Computing Elements using SystemC

Jinwen Xi Peixin Zhong
Dept. of Electrical and Computer Engineering, Michigan State Univeristy
East Lansing, MI 48824, U.S.A.
{xijinwen, pzhong} @ egr.msu.edu

Abstract

We propose the architecture of a novel distributed SAT solver, which is composed of a control unit (CU) and multiple implication units (IU). In this model, CU handles the control-intensive tasks such as clause partitioning, decision and backtracking, and IUs process implications, which are computation-intensive. This model has been modeled with SystemC successfully and simulation results show that it has the potential to get >35 speedup compared to software solvers, and moreover, it doesn't need to re-compile implication circuits for different instances, in contrast to other hardware SAT solvers.

1. Introduction

The Boolean Satisfiability Problem (SAT) is one of the most studied NP-Complete problems because of its significance in both theoretical research and practical applications. For a given SAT problem that is normally expressed in CNF formula, solving it means finding an assignment that satisfies the formula or proving that there doesn't exist such an assignment. SAT has been extensively employed in many VLSI CAD tools such as circuit verification, ATPG and logic synthesis.

Many algorithms have been proposed to solve SAT problems efficiently. The basic algorithm for most of them is based on the backtracking search that Davis and Putnam proposed in [1]. Several software-based SAT solvers (e.g. GRASP[2], zChaff[3]) are designed to run on general-purpose CPU. With the increasing capacity and speed of FPGA, using reconfigurable computing techniques to accelerate SAT solving has become another research focus. Several efficient hardware SAT solvers were proposed [4][5]. Although hardware solvers execute faster than software ones, they have a significant overhead of re-compiling each instance into customized circuits for each problem.

To make a trade-off between software's flexibility and hardware's high performance, a novel architecture of SAT solver is proposed in this paper. The main idea comes from partitioning different stages in solving SAT problems into separate processing modules. There are two main function blocks, implication unit (IU) and control unit (CU) in this architecture. IU deals with implication tasks, which are computation-intensive, while CU handles control-intensive tasks such as decision and backtracking. One feature of this architecture is that it is not instance-specific compared to its predecessors because it need not map each instance to implication circuit before computation. This architecture is scalable since the number of IUs can change according to the size of problem and availability of hardware resources. This architecture has been modeled successfully with SystemC, and it is easy to port it into microprocessor-embedded FPGA with some commercial synthesis tools.

The paper is organized in the following way. Section 2 discusses the architecture and introduces the implementation with SystemC. Section 3 analyzes simulation results and section 4 concludes the research.

2. Architecture and Implementation

The most crucial part of DP-based SAT algorithm is determining implications to prune the search space. From statistic data, a major portion (greater than 90% in most cases) of the solvers' run time is spent in the implication process for most SAT problems [3]. For software implementation, the bottleneck is to detect implications because this process is very slow. Each clause containing the newly assigned or implied variable is accessed and updated sequentially, until there are no new implications.

Efficient implication engine is the key to any SAT solver and parallelizing implication operations can accelerate the speed greatly. The backtracking search part, on the contrary, runs sequentially no matter it lies in hardware or in software because decision and backtracking must be processed variable by variable according to its previous assignment. In our proposed

SAT solver, a novel implication architecture with local memory can process implications without generating implication circuits for different instances.

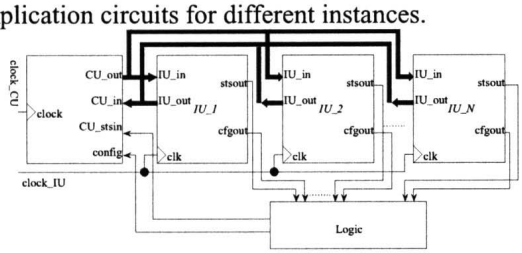

Figure 1. Architecture of the SAT solver

In this architecture, as shown in Figure 1, one CU is the central controller, which is programmed to process control-intensive tasks, such as instance partitioning, clause distribution, making assignments, conflict detection and backtrack search. Multiple IUs are connected with CU via two unidirectional data buses. IUs are designed to be finite state machine to compute implications in parallel with the synchronization of clock_IU and they communicate with CU by handshaking signals and command data. Each IU has a unique address for CU to access. This architecture is scalable since the number of IUs can be adjusted according to the size of instance and the resource availability of target FPGA.

2.1. Implication Unit (IU)

Implication unit generates implications based on the current assignments and communicates with CU. IU contains 4 main blocks: (1) address decoder and input buffer (ADIB), (2) local clause memory (LCM), (3) local variable memory (LVM), (4) finite state machine (FSM), (5) output buffer (OPB).

ADIB interfaces with input data bus and receives the commands and data from CU and directs them to LCM, LVM and FSM according to the types of input data. LCM is designed as a local memory to store the clauses including literals and their relationship. Its structure is a memory array with redundant elements to store clauses with different length. FSM accesses LCM by address pointers via local bus. LVM is another block of local memory to store variables' information such as values and status (assigned, implied or free). FSM functions as scheduler of the whole implication process. It will evaluate LCM to find possible implications when it senses that there are data changes in LVM. The implication process employs a counter for each clause structure. This counter records the number of free literals in the clause. An implication will be generated if the counter's content is 1 and the clause evaluates '0' with current assignments. Meanwhile, IU outputs its status by signals "cfgout" and "stsout". The first one will be driven from low to high if CU finishes transferring sub-instance to this IU and the other one indicates that there are implications under current assignment data.

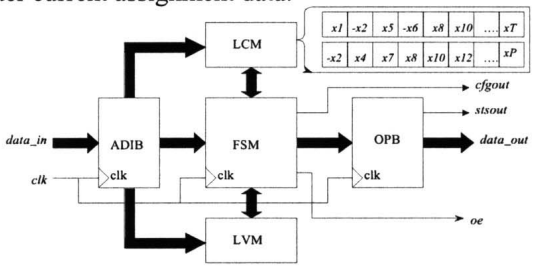

Figure 2. Architecture of IU

2.2. Control Unit (CU)

CU is designed as a firmware of microprocessor to perform different operations according to status signals from IUs. While it is possible to create a custom circuit to implement CU's functions, using microprocessor provides flexibility to implement more sophisticated algorithms.

After hardware reset, CU enters into its software initialization state, where all internal variables and data structures are reset to their defaults. In the starting stage, CU reads the SAT instance and decomposes it into sub-instances according to the number of IUs in the system. Then CU distributes sub-instances to IUs one by one. Once all IUs are configured successfully, the "config" signal will be driven to low, indicating CU to start decision and backtrack search engines. At first, CU clears the implication data in CU and IUs, and then makes an assignment and broadcasts it to all IUs. Once all IUs receive assignment data, they start their implication process simultaneously.

A variable database is maintained by CU according to the input SAT instance to trace status of all variables in making assignment and backtracking search. At first, CU chooses a free variable and assigns it a value by broadcasting this assignment to all IUs. IUs will start their state machines to perform implication algorithm simultaneously. IU's status signal "stsout" will be driven from low to high if there are valid implications. CU will choose the next free available variable from the database and make a new assignment if there are no implications ("CU_stsin" is low), or broadcast another command to read the implication data from IUs when "CU_stsin" signal is high. When all IUs send out their implication data to CU, CU will analyze the data to decide whether there are conflicts. CU will make backtracking search and flip the most recently assigned variable in the next assignment operation if a conflict

occurs, or choose a free variable to make a new assignment if there is no conflict. Finally CU will find that all variables in the database are assigned and no conflicts are generated and it means all sub-instances are satisfied and hence the original problem is satisfied. Otherwise the instance is unsatisfiable since the solver accesses all search space and doesn't find a solution.

2.3 Implementation with SystemC

SystemC is chosen as the platform to model this proposed architecture in system level. SystemC is a C++ library and it has the ability to effectively create cycle-accurate models of software algorithms, hardware architecture and system-level designs. For this architecture, there are several IUs that works in parallel and one CU to perform control functions, and they are corresponding to the concurrent hardware part (IU) and the sequential software part (CU). In SystemC, CU and IU can be implemented as separate modules, and simulation scheduler can make IUs works concurrently in the cycle-accurate simulation. CU is executed as a firmware in a microprocessor, which is the same as what C++ does in the simulation.

3. Simulation Results and Discussions

We use Visual C++ 6.0 and SystemC 2.0.1 library as the main implementation tool suites. ModelSim 6.0 is used to view simulation waveforms. All these tools run on a PC with Pentium-M 1.3GHz CPU and 512MB RAM. There are 3 IUs in the experimental model and the address bus width is 9-bit so it can process at most 512-variable instances. The clock is set to 10MHz and a subset of well-known DIMACS benchmark is chosen for testbench. The run time converted from number of clock cycles for each instance is recorded as Figure 3. Meanwhile, we compare our results ("sat_sc") with the hardware solver that Zhong proposed ("sat_hw") in [5]. We also create a basic DP-based solver with C++ and run the same benchmark on it to compare performances with the other two as "sat_sw" in Figure 3 shows.

For most of the experimental instances, our solver is a bit slower than the "sat_hw" by only comparing clock cycle number because of its memory storage structure for clauses and variables. Since "sat_hw" solver needs to compile each instance into customized circuit before starting computation. Our solver is faster than, at least comparable with, the hardware one if all compilation, placement and routing time are taken into consideration. From the data, we can see that this architecture is much faster than software one with the same algorithm and the speed-up ratio ranges from 35.4 to 133.9.

We just test the solver with 3 IUs and see the speed-up compared with software solvers, so we can estimate that the more the IUs in the system, the better the performance we can get. Since all IUs are in the same structure, we can implement more IUs if there are enough hardware resources in FPGA in the future.

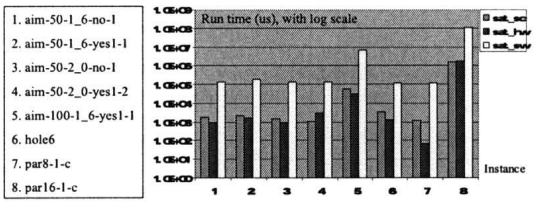

Figure 3. Simulation results comparison

4. Conclusions and Future Work

In this paper we propose the architecture of a SAT solver, which makes a trade-off between software's flexibility and hardware's concurrence. CU takes the responsibility of making decisions, detecting conflicts and performing backtrack search, and IU computes the implications according to different assignments. This architecture has been modeled and simulated with SystemC successfully. Since this architecture is not instance-specific, there is no compilation overhead and large problems can be fit into available hardware, as long as the local memory is adequate.

One of the next steps of the research is to implement this architecture in real FPGA. SystemC synthesis tool will be used to create the hardware portion. On the other side, communication overhead will be more significant with increasing size of instance and number of IUs in the system. It is necessary to consider an optimized on-chip communication protocol between CU and IUs for higher performance. The other direction is to model non-chronological backtracking algorithm, which is key to advanced software solvers, into this model, to get further performance.

References

[1] M. Davis, and H. Putnam, "A Computing Procedure for Quantification Theory ", *ACM Journal*, 1960, pp. 201-215.

[2] Marques-Silva, J. P., and Sakallah, K.A., "GRASP: A Search Algorithm for Propositional Satisfiability", *IEEE Transactions on Computers*, vol. 48, 1999, pp. 506-521.

[3] Matthew, W.,M., Conor F.,M., Zhang, L., Malik, S., "Chaff: Engineering an Efficient SAT Solver", pp. 151-153.

[4] Suyama, T., Yokoo, M.., "Solving Satisfiability Problems on FPGAs", *Proceedings International Workshop on Field Programmable Logic and Applications*, 1996.

[5] Zhong, P., Martonosi, M., Ashar, P. and Malik, S., "Using Reconfigurable Computing Techniques to Accelerate Problems in CAD Domain: A Case Study with Boolean Satisfiability", *Proceedings of DAC*, June 1998.