Adrian SADOWSKI[1], Artur JAKUBSKI[1], Grzegorz MICHALSKI[1]

Czestochowa University of Technology (1)

doi:10.15199/48.2015.11.38

A parallel pipelined naive method for testing satisfiability

Abstract. *Field Programmable Gate Array (FPGA) systems are highly suitable for solving satisfiability problems SAT. The paper will present the possibilities in programmable FPGA chips to test satisfiability by use of parallelism and pipelining. There will be presented various options to approach this problem by use of VHDL language. For this purpose, authors created a dedicated architecture, combined with a PC, by use of the UART protocol. To build the architecture authors used a Xilinx Spartan-3AN plate, the synthesis was performed in the ISE 11.3. Xilinx software.*

Streszczenie. *Układy FPGA ze względu na swoją architekturę bardzo dobrze pasują do rozwiązywania zagadnień z zakresu rozwiązywania problemów spełnialności SAT. W artykule przedstawiono współbierzne rozwiązanie problemu spełnialności z zastosowaniem programowalnych układów FPGA. Dla potrzeb realizacji zadania opracowno dedykowaną architekturę, opartą o układ FPGA (Xilinx Spartan-3AN) kominukującię się za pomocą protokołu UART.* ***(Równoległa realizacja naiwnej potokowej metody testowania spełnialności)***

Keywords: satisfiability, parallel programming, FPGA
Słowa kluczowe: spełnialność, programowanie równoległe, FPGA

The problem of satisfiability testing (SAT) has been used in various types of applications related to the study of algorithms correctness including cryptography, automatic generation of test patterns and the test of logical formulas. There are solutions to these problems of software, hardware or software-hardware. Currently, the most commonly used for this purpose is SAT-solver, which uses recording a logical formula as a logical formula in conjunctive normal form (CNF) [1]. These SAT-solvers test if for a logical formula transformed to (k-SAT), recorded in CNF there exists such a valuation, for which the formula is true. The basic algorithm for SAT-solver is an DPLL algorithm that uses 'Basic Backtrack Algorithm' and other heuristic. However, SAT-solvers not always provide all results for all logical formulas in reasonable time. Some of them only provide information whether the formula is satisfiable or not (if satisfiable - these SAT-solvers do not provide all valuations). Therefore, in the literature we can find information about the complete SAT-solver, which returns information on whether a logical formula is satisfiable and for what valuations, and the SAT-solver not providing complete information (about satisfiability only).

To solve this problem, we used the naive method ('brutal' method), so check all the valuations of variables occurring in the logical formula. The only limitation here is time to perform another valuation in a single system clock cycle, which in our case is 20 ns (for 50 MHz frequency). Estimated results for the number of variables of interval 32-64 are shown Table 1. However, the disadvantage of this approach is small number of propositional variables for which this algorithm finds the valuation in a reasonable time. The advantage of this approach is finding all the valuations, which give the result 'true' for a given logical formula. In this paper, we focus only on solving to the satisfiability problem by use of naive algorithm.

Satisfiability problem

Satisfiability problem is one of the most important problems in the propositional calculus. Some problems can be reduced to a logical formula, and then we analyze the valuations with taking into account satisfiability or tautological character. The example of such an application may be cryptanalysis of certain cryptographic algorithms. The problem of satisfiability of logical formulas is equivalent to many problems of graph theory[3]. You could say that satisfiable formulas describe the world of situations that may arise. This note makes that modeling many systems and examining the formulas expressing their behavior, sometimes we can find the errors of the tested system (through satisfiability checking).

Satisfiability problem is a propositional calculus question - whether for a given logical formula, there exists such a substitution (valuation) of propositional variables that the formula is true. Equivalent is that it is not true that the negation of this formula is a tautology. Satisfiability problem is decidable - you can try out all the substitutions, which are 2^N, where N a number of variables of the formula. This method is called a naive algorithm. The naive algorithm for checking whether a formula is satisfiable or not, works in exponential time. For the formula of N variables, 2^N valuations need to be checked. For practical purposes this algorithm, already for several dozens of variables, using this method often is not achievable. And the problem if you can do it much faster to determine whether the formula is satisfiable, is considered one of the important problems in mathematics. ($P = NP$). The aim of our work is, however, to examine the impact of parallelization and pipelining on the architecture dedicated to this purpose for satisfiability problem for $N = 40$.

Naive method of testing satisfiability

Here is an example of the algorithm checking satisfiability of formula by naive method.The input is a formula ϕ and N – the number of variables that occur in formula ϕ. Loop 2 is repeated for all valuations, from the valuation consisting of N values equal 0 to N values equal 1 (run like the binary numbers from 0 to $2^N - 1$). $v_i(\phi)$ is the valuation of formula ϕ for the valuation of i.

Algorythm 1: Naive method of testing satisfiability

Input: ϕ, N

Output: 'satisfiability','unsatisfiability'

1. $A \leftarrow$ 'unsatisfiability'.
2. For i from $\overbrace{00\ldots0}^{N}$ to $\overbrace{11\ldots1}^{N}$ do:
   - if $(v_i(\phi) = 1)$
     $A \leftarrow$ 'satisfiability'.
3. return A.

Architecture dedicated to the satisfiability testing based on the naive algorithm

In this section we discuss the evolution of architecture dedicated to the satisfiability testing (by use of naive algorithm), from sequential approach to pipe-parallel approach to this problem. We will also present how changing the architecture influenced (improved) the time of computing performance.

Dedicated sequential architecture

To investigate the satisfiability problem by naive algorithm we have designed a sequential architecture dedicated to this purpose. In the first version (sequential) of designing

Number of variables	Number of valuations	time [seconds]	time [minutes]	time [hours]	time [days]	time [years]
32	4294967296	85.90	1.43E+00	2.39E-02	9.94E-04	2.76E-06
40	1099511627776	2.20E+04	3.67E+02	6.11E+00	2.55E-01	7.07E-04
48	281474976710656	5.63E+06	9.38E+04	1.56E+03	6.25E+01	1.81E-01
56	72057594037927900	1.44E+09	2.40E+07	4.00E+05	1.67E+04	4.63E+01
64	18446744073709600000	3.67E+11	6.15E+09	1.02E+08	4.27E+06	1.19E+04

Table 1. Estimated teoretical time for simulation performed on the dedicated architecture for a f = 50 MHz.

$$\phi = ((not\ temp(3)\ or\ temp(4)\ or\ temp(7))\ and\ (not\ temp(3)\ or\ not\ temp(10)\ or\ not\ temp(16))\ and\ (not\ temp(17)\ or\ not\ temp(15)\ or\ not\ temp(9))\ and\ (temp(8)\ or\ not\ temp(18)\ or\ temp(17))\ and\ (nottemp(21)\ or\ not\ temp(31)\ or\ not\ temp(27))\ and\ (temp(14)\ or\ not\ temp(22)\ or\ temp(11))\ and\ (temp(1)\ or\ temp(6)\ or\ temp(14))\ and\ (temp(15)\ or\ not\ temp(18)\ or\ temp(2))\ and\ (temp(16)\ or\ temp(14)\ or\ not\ temp(28))\ and\ (temp(12)\ or\ not\ temp(21)\ or\ temp(23))\ and\ (temp(2)\ or\ temp(14)\ or\ not\ temp(13))\ and\ (temp(2)\ or\ temp(13)\ or\ not\ temp(10))\ and\ (not\ temp(1)\ or\ not\ temp(5)\ or\ temp(3))\ and\ (temp(12)\ or\ temp(19)\ or\ temp(21))\ and\ (temp(18)\ or\ temp(4)\ or\ temp(31))\ and\ (not\ temp(4)\ or\ not\ temp(0)\ or\ not\ temp(5))\ and\ (temp(26)\ or\ not\ temp(29)\ or\ temp(11))\ and\ (temp(15)\ or\ temp(29)\ or\ not\ temp(13))\ and\ (temp(24)\ or\ not\ temp(11)\ or\ temp(18))\ and\ (not\ temp(15)\ or\ not\ temp(26)\ or\ temp(6))\ and\ (not\ temp(20)\ or\ not\ temp(25)\ or\ not\ temp(30))\ and\ (temp(0)\ or\ temp(6)\ or\ temp(16))\ and\ temp(31)\ and\ temp(0))$$

Fig. 1. An example logical formula of $N = 32$ propositional variables caption: ϕ denotes the tested formula, $temp(31)$ – variable of number 31, $not\ temp(19)$ – negation of the variable of number 19).

the architecture (Figure 2) – our research is focused on finding satisfiability of the logical formula, for a given number of propositional variables. Without the use of parallelization and pipelining our architecture is able to check all valuations for N = 40 bits in a reasonable time (estimated time = 6.11 h, Table 1).

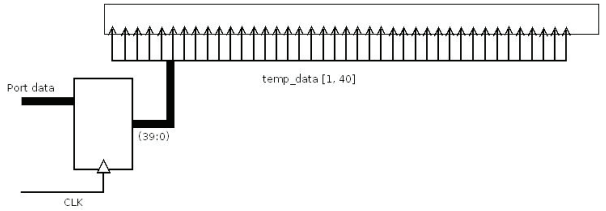

Fig. 2. Sequential architecture (FPGA) for testing satisfiability by use of naive method.

At the first attempts to create such an architecture, we wanted to send individual valuations of propositional variables from a PC to the FPGA through the UART transmission protocol using RS232, and also receive in the same way. However, due to delays that could result from communication between the PC and the FPGA, we used the counter in the FPGA architecture to valuate N propositional variables. The results – the valuations which formula is satisfiable for – are sent in blocks of 8 bits (1 Byte) to the PC. To compare our architecture with a general-purpose architecture, we used processor Intel Atom N450, which was running Ubuntu Linux. For this purpose, we realized a software written in the C++ language. This program uses the bitset container and through built-in mechanisms it runs faster than a program created on the boolean tables [4]. Obtained results are presented in the Table 2. Tested formulas are identical to the both FPGA architectures and the general-purpose processors used in the research. [5, 6]

The results of the study relate to the same logical formula, tested in individual cases for the CPU and FPGA. All FPGA times given in the Table 2 show that the result of finding valuation is less than one second. More accurate results would be estimated results only (based on the clock frequency of the FPGA and the number of pulses generated for each of the cases). These values are less than one second for given sample formulas. Obtaining more accurate results was not the goal of our research. The goal of the paper is showing that the use of programmable devices for this algorithm gives a several dozen times acceleration.

Number of variables	Number of valuations	Estimated time [seconds]	FPGA [s]	CPU - bitset [s]
24	16777216	0,34	< 1	0,26
25	33554432	0,67	< 1	0,66
26	67108864	1,34	< 1	1,36
27	134217728	2,68	< 1	2,68
28	268435456	5,37	< 1	5,44
29	536870912	10,73	< 1	10,82
30	1073741824	21,47	< 1	21,76
31	2147483648	42,95	< 1	43,58
32	4294967296	85,90	< 1	77,14

Table 2. Results for FPGA (Spartan 3AN f=50 MHz) and CPU (Atom N450)

The results given in Table 2. indicate that the architecture dedicated to the problem runs 70 times faster than the general-purpose architecture for $N = 32$. Comparing the Spartan 3AN clock frequency and, in this case, Intel Atom N450 of 1.6 GHz clock frequency, can be assumed that the Spartan 3AN is 32 times slower than the Atom N450. Summing up, we can resolutely say that dedicated architecture is able to operate up to 100 times faster than PC (for the mentioned CPU processor), for the problem with the number of propositional variables $N = 32$.

For $N = 32$, our formula is of the form shown on Figure 1.

It should also be noted that the acceleration of the calculations, to a large extent, depends on the logical formula. It may happen that the result is obtained at the start of the logical formula testing. Experimental studies were carried out by synthetizing the models of test control units, described in VHDL. Synthesis was made in the software Xilinx ISE 11.3 for Spartan-3AN FPGA (XC3S700AN). In further sections of the work there are discussed both pipelining and parallelization mechanisms, which we have used in our system.

Pipelining

Pipelining is one of the methods of parallel processing. It involves dividing up the processing unit into blocks (in programmable FPGA chips - processing element) which are connected in the 'pipeline'. Data processed by a block go to the next processing block in the pipeline, while the previous block can process the next portion of data. In hardware implementation, there are used registers to separate the blocks [7]. Individual fragments are separated by system registers, which makes each piece can process data independently. Clock frequency of the system is dependent on the slowest block. For a propositional formula with a number of variables $N = 36$ and logical operators, we obtain the following pipeline blocks (Figure 3). Pipelining effect allows you to perform in one clock cycle more operations, consequently lowering the clock frequency. However, this mechanism is not always worth while. In our case, pipelining is selected individually for each logical formula.

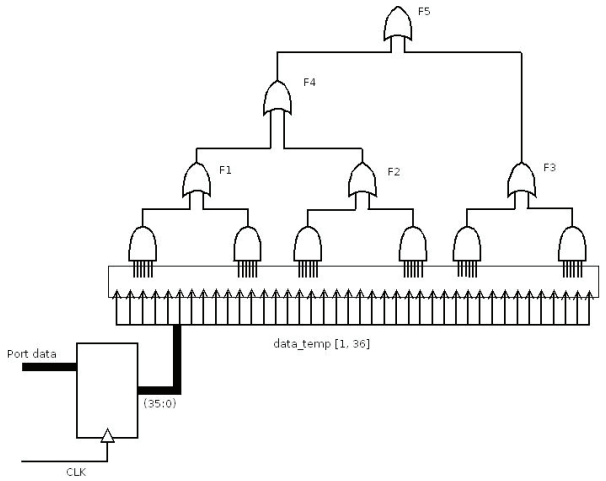

Fig. 3. Sequential architecture for testing satisfiability by use of naive method.

The advantage of this approach is often a significant boost capacity at low cost and lower power consumption. The disadvantage is the use of additional registers in FPGA. Another disadvantage of using pipelining is the extension of data flow control system, in particular, for conditional statements and it reduce the clock frequency[8]. It must be added, that pipelining is adjusted individually for each logical formula. For a logical formula consisting of operators and only - we get the full pipelining, acting with maximum clock frequency, in our case, f = 50MHz.

Parallelization

FPGAs effectively implement low parallelism, due to the operations performed on the bits. The structure of programmable circuits lets to adapt the parallelism to the tested problem. In carrying out the same functional blocks in one clock cykle for different propositional variables, we are able to get the most effective of all the hardware parallelism available in the market. We apply this feature to testing the satisfiability problem by use of naive method.

Parallelism is a mechanism that significantly contributes to the acceleration of the calculations in our architecture. For our algorithm (naive method), a reasonable solution, is dividing the counter into k smaller counters.

The simplest solution to this problem is to create two counters. The counter that runs on even numbers (last bit 0) and the counter running on odd numbers (last bit 1). However, in our experiment for $N = 40$ propositional variables we obtain a problem consisting of 2^{40} valuations. This value is divided into the factors 2^{36} and 2^4. Which means that we get $k = 16$ counters (of 4 bits) and each one will check up 2^{36} valuations. So, each of the parallelizable counters has to check $2^N/k$ valuations (in this case: 68719476736). This will give us the 16 blocks (PE - Procesing Element) running in parallel (Figure 4). The estimated checking time of frequency $f = 50$ MHz, in this case (2^{36} valuations) is $2820s$ (47 min.).

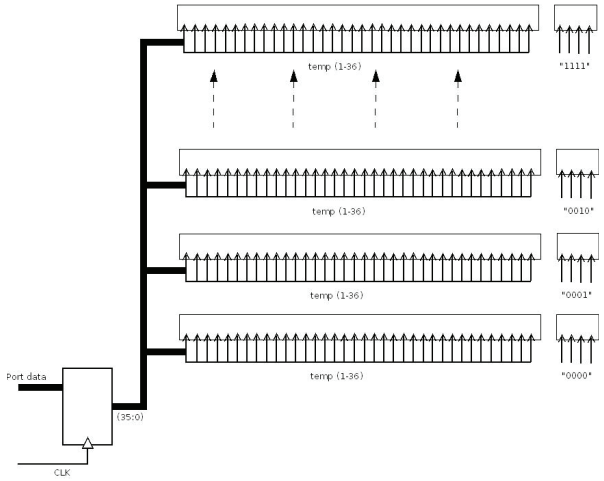

Fig. 4. Parallelisation mechanism for N=40 propositional variables.

The combination of parallelization and pipelining mechanisms

Parallelisation mechanisms with pipelining used in them are widely used for SAT problems [9, 10]. In the final stage of our work, for the tested logic formulas, we linked both parallelism and pipelining mechanisms, which has contributed significantly to improving the results. By linking pipelining and pararelism it has been created the architecture to test satysfiability by use of the naive method. We divided our parallel architecture into 16, 64 and 256 bloks (PE). The achieved results for studied architectures are placed in Table 3. For 256 PE we got over 130 times more acceleration of computing. One should here add that the code in this case consists of over 4000 lines and creating it is time-consuming.

Sequentially	Number of the PE		
	16	64	256
22920[s]	2820[s]	680[s]	170[s]

Table 3. Obtained times of testing the formula satisfiability.

The results

Our goal in that study was to test what acceleration gives the parallelism and pipelining in the architecture exploring satisfiability by use of the naive method. The tested logical formula consisted of $N = 40$ propositional variables (2^{40} valuations). We divided the logical formula into $16, 64$, and 256 defined bloks (PE), which consequently check in one clocks cycle $2^{36}, 2^{34}$, and 2^{32} tested valuations. In the study we have received the results for sequential architecture and parallel architecture. In a sequential architecture – checking satisfiability of logical formula consisting of $N = 40$ propositional variables took 6 hours 22 minutes, on the card FPGA Spartan–3AN (of frequency of 50 MHz). After parallelization (on the same card FPGA), testing the same logical formula, we got the result 'satisfiability' in time 170 seconds (for 256 PE).

Summary

The obtained results show a significant vulnerability naive satisfiability checking method for parallelization using programmable FPGA chips. The applied in considering simplest possible method of verifying satisfiability, allowed to start further work on the parallelization of more complex algorithms. The selected method of checking satisfiability logical formula was aimed especially to verification of the ability of FPGAs to resolve these types of problems. At the same time you should consider the other available highly parallel systems like as graphics processors or Intel Xeon Phi.

REFERENCES

[1] Hauck S., DeHon A. Reconfigurable Computing The Theory and Practice of FPGA-Based Computation. Morgan Kaufmann/Elsevier, 2008.
[2] Zhang L., Malik S. The Quest for Efficient Boolean Satisfiability Solvers. Proceeding CAV '02 Proceedings of the 14th International Conference on Computer Aided Verification, Springer-Verlag London, 2002.
[3] Cichoń J., Gogolewski M., Kutyłowski M. Logika dla informatyków. Wydawnictwo Wyższej Szkoły Komunikacji i Zarządzania w Poznaniu, 2006.
[4] Sadowski A., Jakubski A. Dedykowana architektura stworzona na FPGA do badania logicznych formuł. Zeszyty Naukowe Wydziału Elektroniki, Telekomunikacji i Informatyki Politechniki Gdańskiej, 2011.
[5] Dlugosch P., Brown D., Glendenning P., Leventhal M., Noyes H. An Efficient and Scalable Semiconductor Architecture for Parallel Automata Processing IEEE Transactions on Parallel & Distributed Systems , no. 1, pp. 1, PrePrints PrePrints, doi:10.1109/TPDS.2014.8
[6] Hu, Y. Exploring Formal Verification Methodology for FPGA-Based Digital Systems. Technical Report SAND2012-7926, Sandia National Laboratories, 2012
[7] Łuba T., Rawski M., Tomaszewicz P., Zbierzchowski B. Synteza układów cyfrowych. Praca zbiorowa. Wydawnictwa Komunikacji i Lacznosci, Warszawa, 2011.
[8] Wiatr K. Dedicated Pipelined Architecture of Specialized Processors for Real-Time Image Pre-Processing. Proc. of the International Symposium on Robotics and Automation, Monterrey 2000, pp. 301-306
[9] Redekopp M., Dandalis A. A Parallel Pipelined SAT Solver for FPGA. Lecture Notes in Computer Science, Volume 1896/2000, pp. 462-468, Springer-Verlag, 2000
[10] Pagarani T., Kocan F., Saab D.G., Abraham J.A. Parallel and scalable architecture for solving SATisfiability on reconfigurable FPGA. Proceedings of the IEEE Custom Integrated Circuits Conference, CICC 2000, pp. 147-150.

***Authors**: Ph.D. Student Adrian Sadowski, Ph.D. Artur Jakubski, Ph.D. Grzegorz Michalski, Institute of Computer and Information Sciences, The Faculty of Mechanical Engineering and Computer Science, Czestochowa University of Technology, ul. Dabrowskiego 69, 42-201 Czestochowa, Poland, email: adrian.sadowski@icis.pcz.pl, artur.jakubski@icis.pcz.pl*