DESIGN METHODS FOR FPGA-BASED IMPLEMENTATION OF COMBINATORIAL SEARCH ALGORITHMS

Iouliia Skliarova, Valery Sklyarov [1])

Abstract

Many combinatorial search problems can be efficiently formulated and solved over Boolean and ternary matrices that keep the initial and all the required intermediate data. Three of such problems, namely, the Boolean satisfiability, covering and graph coloring, have been examined and analyzed. The results of analysis make it possible suggesting a number of reusable functional blocks for the considered search algorithms, which provide significant support for the design of FPGA-based digital systems allowing miscellaneous combinatorial problems to be solved. The proposed technique enables the designers to concentrate their efforts on the algorithms and to minimize the design time required for less significant details of hardware implementation.

1. Introduction

There are many practical problems that can be formulated over such mathematical models as graphs, discrete matrices, sets, Boolean equations, etc. Majority of these models are mutually convertible, i.e. any of them might be selected for similar purposes. Many practical problems can be solved by applying various algorithms of combinatorial search over a chosen mathematical model. Such algorithms have two distinctive features. Firstly, they require considering a huge number of different variants. Secondly, these variants can be ordered and examined with the aid of a decision (search) tree that provides an efficient way for handling intermediate solutions.

Examples of combinatorial problems that can be solved with the aid of the mentioned above algorithms are Boolean satisfiability [5,9]; covering of Boolean matrices [7]; graph coloring [1], etc. Their results have been used for a number of practical applications such as electronic design automation [6], mobile computing [18], multimedia processors [16], embedded [4] and distributed [3] systems, formal verification [17] and many others [5,9]. The detailed analysis of combinatorial algorithms, their practical applications and publications in this area was done in [5,6,9,19].

[1] Department of Electronics, Telecommunications and Informatics, IEETA, University of Aveiro, 3810-193 Aveiro, Portugal, iouliia@det.ua.pt, skl@det.ua.pt

Systems for solving combinatorial problems might be implemented in field-programmable devices (FPGA, in particular). The latter have a number of advantages, which have appeared because the considered tasks possess a set of specific features. Firstly, any task involves a huge number of similar operations. As a rule, these operations are not the same for different combinatorial problems. Thus, it is not easy to construct a universal combinatorial processor, i.e. processor's instructions have to be customized for a particular problem that is going to be solved. This can easily be done with the aid of FPGA technology. Secondly, different practical applications might require solving combinatorial tasks with varying complexity. However, optimal results can be achieved in case if the size of processor operands permits any required operation to be performed in one clock cycle. Parameterizable circuits that provide such an opportunity can easily be implemented in FPGA. Thirdly, FPGA enable us to build on the same microchip any desired (customized) interface between a combinatorial accelerator and a general-purpose computational system (or any specialized system that requires an accelerator). Fourthly, the complexity of recent FPGA allows for the construction of a complete system-on-chip and a combinatorial accelerator can be implemented as an application-specific co-processor within this system.

Note that hardware permitting to carry out combinatorial search over various (alternative) models requires different FPGA resources, i.e. the complexity of the respective circuits depends essentially on the adopted mathematical model. Our experience has shown that a discrete (Boolean and ternary) matrix can be seen as a very adequate model for the considered computations [8,10].

This paper suggests a number of reusable blocks that can be employed for constructing application-specific hardware processors that permit combinatorial problems formulated over Boolean and ternary matrices to be solved. The blocks have been selected on the basis of analysis of different search algorithms and their primary operations. It is shown that such algorithms require recursive procedures that repeat the following basic steps, which are 1) reduction of a given or an intermediate matrix permitting to decrease the problem complexity; 2) selection of matrix elements (rows or columns) that incrementally compose a solution (or enable to conclude that the solution does not exist); 3) branching in case if more than one matrix element has to be selected; 4) termination of a branch in case if it is proved that the selected path does not permit neither to find a solution nor to improve a previously discovered solution. The selected blocks have been implemented both as Handel-C macros [2] and VHDL library modules.

2. Template for Combinatorial Search Algorithms

Figure 1 depicts the basic skeletal algorithm (template) of combinatorial search, which will be detailed below on examples of 1) Boolean satisfiability problem; 2) matrix covering problem; and 3) graph coloring problem.

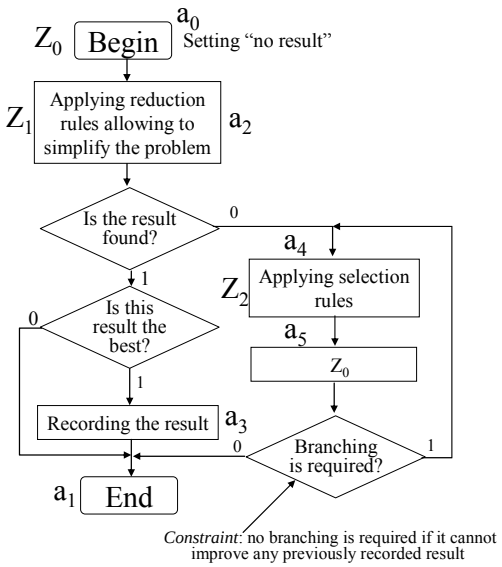

Figure 1. Template for combinatorial search algorithms.

The algorithm is presented in form of hierarchical graph-schemes (HGS) [12]. The module Z_0 in figure 1 describes the top-level recursive algorithm. The module Z_1 implements a problem specific reduction technique, which possesses an important problem-independent feature: Z_1 operates over the same instance of a Boolean/ternary matrix and the required reduction is provided through masking some of the matrix rows and columns. The module Z_2 implements a problem specific selection technique and operates also over the considered above matrix. Problem specific operations are executed with the aid of a dynamically reconfigurable unit.

2.1. Boolean Satisfiability

Let us assume that a matrix **U** shown in figure 2 is given. Conversion between Boolean formulae and ternary matrices is shown in [10]. To solve the Boolean satisfiability problem it is necessary to find out a ternary vector **w** that is orthogonal to all rows of the matrix **U**. Orthogonality of two ternary vectors is defined as follows [19]: ($\mathbf{m}_i$ *ort* $\mathbf{m}_j$) $\Rightarrow$ $\{\mathbf{m}_i\} \cap \{\mathbf{m}_j\} = \varnothing$, where $\{\mathbf{m}_i\}$ ($\{\mathbf{m}_j\}$) is a set of Boolean vectors that correspond to the ternary vector $\mathbf{m}_i$ ($\mathbf{m}_j$) by replacing the don't care values ('-') with all possible combinations of 0s and 1s. If and only if two ternary vectors are not orthogonal (let us designate this as $\mathbf{m}_i$ ~~*ort*~~ $\mathbf{m}_j$) they do intersect in the Boolean space: ($\mathbf{m}_i$ ~~*ort*~~ $\mathbf{m}_j$) $\Leftrightarrow$ ($\mathbf{m}_i$ *ins* $\mathbf{m}_j$).

Basic operations of the algorithm in figure 1 realize so-called *reduction* and *selection* rules. For the Boolean satisfiability problem the following reduction rules can be applied, which permit an initial matrix to be simplified [11]:

1. If a column contains just don't care values it must be deleted from the matrix.

2. All rows that are orthogonal to an intermediate vector **w** (that incrementally forms a solution) must be removed from the matrix. All columns that correspond to the components of the vector **w** with values '1' and '0' must be deleted from the matrix.
3. If the matrix contains a row with just one component '0' ('1') with an index *i* then the element *i* of the vector **w** must be assigned the value '1' ('0'), i.e. the inverted value.
4. If there is a column *j* in the matrix without values '1' ('0') then the element *j* of **w** can be assigned the value '1' ('0').

The *selection* rules make possible to split the problem into sub-problems and to examine them in turn. For the Boolean satisfiability problem the following selection rules can be applied [11]. A column $\mathbf{c}_{max}$, which contains the maximum number $N_1 + N_0$ of values '1' (N_1) and '0' (N_0) is selected. If $N_1 \geq N_0$ then the '0' is assigned to the element *max* of **w**, otherwise (if $N_0 > N_1$) the value '1' is assigned to the same element if **w**. It permits to construct a minor (sub-matrix) which will be examined at the next step.

Figure 2 illustrates how these rules can be applied to a practical example (the numbers of rules are shown in circles). The sequence of steps is marked by letters *a*, *b*, …, *g*. The vector **w** is incrementally constructed in an order indicated by the numbers 1-5 enclosed in squares. The last number 5 detects a solution.

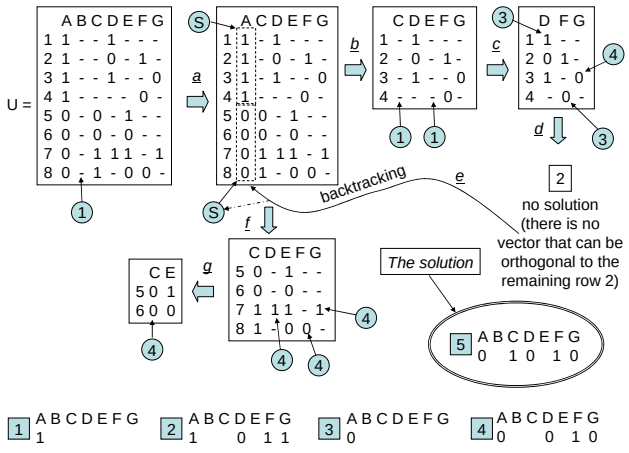

Figure 2. Using the algorithm in figure 1 for solving the Boolean satisfiability problem.

2.2. Matrix Covering

Let us consider now how to use the same template in figure 1 to solve another known combinatorial problem that permits to find out a minimal column cover of a Boolean matrix [10]. Suppose that it is given a matrix shown at the top-left corner of figure 3. Columns **a**, **b** and **d** represent one

possible solution, i.e. the minimal number of columns that have at least one value '1' in each row of the matrix. The following set of rules that permit to reduce (simplify) the matrix can be used:

1. If for $i \neq j$, $\mathbf{row}_i$ & $\mathbf{row}_j$ = $\mathbf{row}_j$ then the $\mathbf{row}_i$ can be removed from the matrix, for example, $\mathbf{row}_6$ & $\mathbf{row}_5$ = $\mathbf{row}_5$ and $\mathbf{row}_6$ has to be removed from the matrix.
2. If for $i \neq j$, $\mathbf{column}_i$ & $\mathbf{column}_j$ = $\mathbf{column}_i$ then the $\mathbf{column}_i$ can be removed from the matrix.
3. If there is a row, which does not have values '1' then covering cannot be found.

For the selection purposes the following rules can be used:

1. If a row has just one value '1' then the respective column (i.e. the column containing this '1') must be included into the covering and deleted from the matrix. All the rows that have value '1' in this column must also be deleted from the matrix.
2. If all rows have more than one value '1' then the first row from the top of the matrix that contains the minimum number of ones has to be selected. For this row it is necessary to analyze all possible branches and the number of such branches is equal to the number of values '1' in the row. Obviously, any branch has to be examined until the step where an intermediate result becomes equal to or worse than any previously discovered covering.

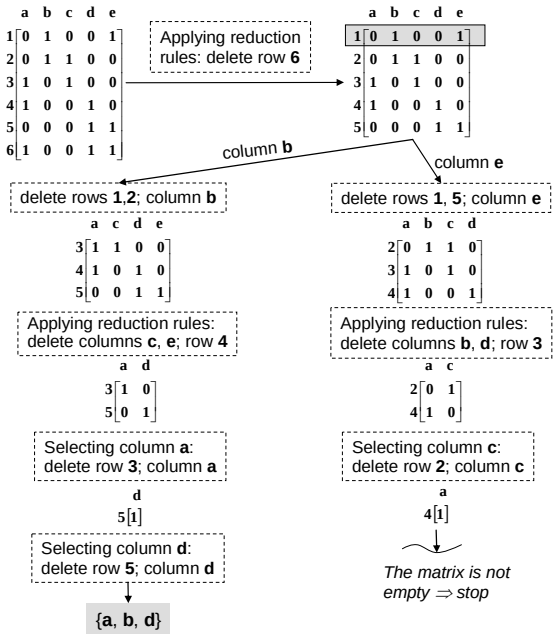

Figure 3. Using the algorithm in figure 1 to find out the minimal column cover of the matrix.

Figure 3 shows all the steps that are required in order to find out the solution for the given matrix. There is just one branching point in figure 3: **b-e**. After getting the first solution {**a**,**b**,**d**} we are interested just in coverings composed of 2 or less columns. Thus, it is not necessary to traverse all branches and forward propagation can be stopped at any point that gives a 2-component incomplete solution.

2.3. Graph Coloring

For solving the graph-coloring problem it is necessary to paint graph nodes in such a way that: 1) any two nodes connected with an edge are painted in different colors; 2) the number of used colors is minimal. The details of converting a graph to a ternary matrix are given in [15]. The resulting matrix $\mu(G)$ has the same number of rows as the number N of vertices in the graph G and at maximum N-1 columns. If and only if two vertices *i* and *j* are connected with an edge in G then the matrix rows $\mathbf{m}_i$ and $\mathbf{m}_j$ must be orthogonal.

The algorithm permitting to solve the graph coloring problem consists in finding the minimal number K of such subsets of rows in the matrix $\mu(G)$ that any subset does not contain mutually orthogonal rows. In other words, all the rows belonging to the same subset must intersect in the Boolean space. The number K is the minimal number of colors for the graph G. Rows from each subset correspond to the vertices of the graph G that can be painted in the same color.

Some of the *reduction* and *selection* rules can be directly borrowed from the method of condensation, proposed in [19]. The following *reduction* rules can be used:

1. If, after selecting a new color, the matrix $\mu(G)$ contains a column without values '0' ('1') then this column can be deleted;
2. If the matrix $\mu(G)$ contains a row with just don't care values then this row can be deleted from the matrix and included in the constructed subset;
3. All the rows that have already been included in the constructed subsets are removed from the matrix $\mu(G)$.

The coloring algorithm consists in the application of the following steps [15]:

a) Choose a new color (i.e. create a new initially empty subset);
b) Apply the reduction rules;
c) Consider the topmost row $\mathbf{m}_i$ in the matrix;
d) Include the row $\mathbf{m}_i$ in the constructed subset and delete it from the matrix;
e) Find out all other rows intersecting (i.e. not orthogonal) with the vector $\mathbf{m}_i$;
f) Select the first not tried yet row $\mathbf{m}_j$ from point e), include it in the constructed subset and then delete it from the matrix;
g) Assign $\mathbf{m}_i = \mathbf{m}_i$ *ins* $\mathbf{m}_j$ and repeat the steps e)-g) if this is possible. If this is not possible go to the step h);
h) If the intermediate matrix is not empty repeat the steps a),…,g). Otherwise, store the solution found and then backtrack to the nearest branching point (set at the step f) and try to find a better solution by repeating the steps f)…h).

2.4. Common Features of Combinatorial Search Algorithms

The considered above search algorithms have similar characteristics. Their distinctive feature is the execution of problem-specific operations and traversing a decision (search) tree starting from its root by involving such procedures as forward search and backtracking. Any branching point can be considered as extracting a sub-tree with a local root. These algorithms possess several common features:

1. They can be formulated recursively, therefore a recursive control unit might be very helpful.
2. They do not change the initial data (i.e. the initial matrix) because the matrix reduction can be provided by masking some rows/columns and using just the remainder of the matrix.
3. They invoke a very limited number of operations (such as reduction and selection operations considered above), which have to be applied to a huge volume of data.
4. Subsets of the required operations are usually not the same for different combinatorial problems. Thus, a reconfigurable control unit is required which permits to customize a sequence of control steps for a particular combinatorial problem.
5. In order to perform forward and backward propagation we can use a stack memory that stores and restores intermediate results (such as the values of mask registers) in branching points.
6. The algorithms can be decomposed into two levels of control operations. The top-level (recursive) sequence is the same for different algorithms (see figure 1). The bottom level sequence permits the required operations over Boolean and ternary vectors to be executed. As a rule, these operations are not the same for different algorithms and it requires changes to the functionality of the respective circuit.

3. Specification of Reusable Functional Blocks

An analysis of combinatorial search algorithms applied to Boolean and ternary matrices allowed to select the following functional blocks (FB) for implementing these algorithms in hardware:

1. Memory permitting to store Boolean and ternary matrices and to provide direct access to both rows and columns.
2. Mask registers allowing to use the same storage for handling the initial matrix and all the sub-matrices (minors), which can be constructed by removing some rows/columns.
3. Stacks for managing forward and backward propagation steps, which permit to construct sub-matrices sequentially and to return back to any intermediate sub-matrix if required. These stacks have to store masks for the sub-matrices and the contents of general purpose registers.
4. General-purpose registers for keeping Boolean/ternary vectors.
5. A device for computations over Boolean and ternary vectors.
6. Control circuits, which make possible to execute operations at two levels: a) operations over either one or two Boolean/ternary vectors; and b) operations over the entire matrices.

7. Additional auxiliary circuits for testing, debugging and interacting with the considered system targeted to combinatorial search algorithms over Boolean and ternary matrices.

To make the considered above blocks reusable we have to provide them with the properties of parameterization and reconfigurability. The first property allows for scalability in such a way that the considered blocks can be applicable to matrices of different dimensions (i.e. with different number of rows and columns). The second property permits to configure control circuits at any level in such a way that different operations over vectors can be chosen.

Figure 4 depicts the proposed architecture of a reconfigurable processing unit for executing combinatorial search algorithms over Boolean and ternary matrices.

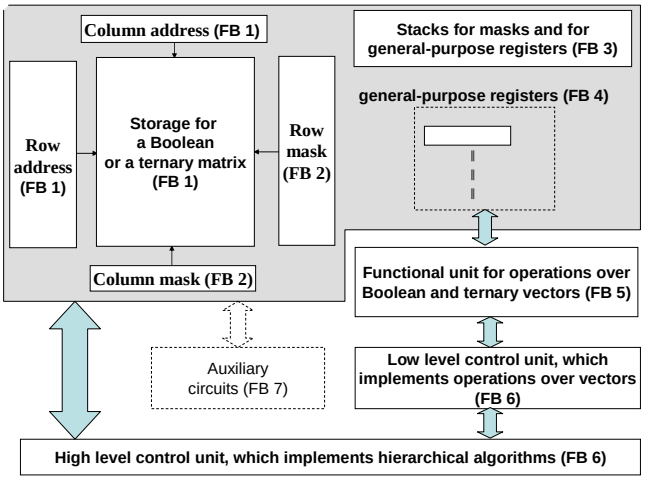

Figure 4. Processing unit constructed from the functional blocks (FB) indicated in the points 1-7.

The following assumptions have been taken into account:

1. A ternary matrix $\mathbf{M}_t$ is represented by two binary matrices $\mathbf{M}_b^0$ and $\mathbf{M}_b^1$, where $\mathbf{M}_b^0$ contains the values '1' in the positions that have the values '0' in $\mathbf{M}_t$, and $\mathbf{M}_b^1$ contains the values '1' in the positions that have the values '1' in $\mathbf{M}_t$.
2. Dual access to the matrix rows/columns is provided through keeping in memory two matrices: the original matrix and its transpose.
3. Hierarchical modular specification is used to implement control circuits in such a way that an execution of low-level operations is provided with the aid of modules (sub-algorithms), which can be activated (called) by a control circuit running a higher level algorithm.
4. Changing low-level and high-level algorithms is achieved by reconfiguring the control circuit modeled by a reconfigurable hierarchical finite state machine [12]. An example of such machine for implementing operations over Boolean and ternary vectors was considered in detail in [14].

4. Implementation Details

A number of FPGA-based circuits for implementing combinatorial search algorithms over Boolean and ternary matrices have been designed from specifications in VHDL and Handel-C and tested in hardware. Different FB described in VHDL have been examined in [8]. For example, the functional blocks 1-5 indicated in the previous section have been designed and tested in Xilinx FPGA of Virtex-EM family. Hierarchical finite state machines for VHDL and Handel-C projects have also been designed and tested in Xilinx FPGA of Spartan-II family and the respective results were reported in [13]. Complete Handel-C projects were described and tested for all the problems considered in the paper and the relevant results were reported in [13,15].

The results of experiments have shown that the proposed technique makes possible to shorten essentially the design time of combinatorial processors. The proposed FB take into account many specific features of combinatorial search algorithms and they have been optimized for the considered problems. It makes possible to provide block-based high-level design, i.e. to concentrate the efforts of the designer on the considered algorithms avoiding (or at least minimizing) the details of hardware implementation. Since the proposed reusable blocks were implemented as a set of Handel-C macros and VHDL library modules, it allows to consider either the design flow on the basis of a system-level specification language or a widely-used hardware description language. A set of additional blocks described in Handel-C and VHDL provide very effective visualization and debugging tools and they can be inserted into “debug version” and removed from the “release version”. They are also very helpful for experimental purposes.

5. Conclusion

The paper analyzes different combinatorial problems and demonstrates that they can be formulated over Boolean and ternary matrices. It is shown that many problems from this area might be solved with the aid of search algorithms that have a number of common features. The paper suggests primary building blocks for these algorithms, which provide unique operations over matrices, support stacks and implement recursive control algorithms. These blocks were described in Handel-C and in VHDL and implemented in FPGA. Three combinatorial algorithms for the Boolean satisfiability, covering and graph coloring problems were examined. The results of implementations have demonstrated a significant speedup in the design process and some other advantages such as support for visualization of intermediate results and for the debugging.

6. References

[1] Bibliographies and links on graph coloring, available at: http://www.cs.ualberta.ca/~joe/Coloring/index.html.

[2] Celoxica products, available at: http://www.celoxica.com.

[3] Feldman, R., Haubelt, C., Monien, B., Teich, J., Fault Tolerance Analysis of Distributed Reconfigurable Systems Using SAT-Based Techniques, in: Proceeding of FPL'2003, Portugal, pp 478-487, 2003.

[4] Goossens, G., Praet, J.V., Lanneer, D., *et al*., Embedded Software in Real-Time Signal Processing Systems: Design Technologies, in: Proceedings of the IEEE, vol. 85, no. 3, March, 1997, pp. 436-454.

[5] Gu, J., Purdom, P.W., Franco, J., Wah, B.W., Algorithms for the Satisfiability (SAT) Problem: A Survey, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 35, pp. 19-151, 1997

[6] Marques-Silva, J.P., Sakallah, K.A., Boolean Satisfiability in Electronic Design Automation, in: Proceedings of DAC, USA, Los Angeles, 2000.

[7] Plessl, C., Platzner, M., Instance-Specific Accelerators for Minimum Covering, in: Proc. 1st Int. Conf. ERSA'2001, Las Vegas, pp. 85-91.

[8] Skliarova, I., Reconfigurable Architectures for Problems of Combinatorial Optimization, Ph.D. Thesis, University of Aveiro, Portugal, 2004.

[9] Skliarova, I., Ferrari, A.B., Reconfigurable Hardware SAT Solvers: A Survey of Systems, in: IEEE Transactions on Computers, vol. 53, issue 11, pp 1449-1461, 2004.

[10] Skliarova, I., Ferrari, A.B., The Design and Implementation of a Reconfigurable Processor for Problems of Combinatorial Computation, in: Journal of Systems Architecture, Special Issue on Reconfigurable Systems, vol. 49, nos. 4-6, pp, 211-226, 2003.

[11] Skliarova, I., Ferrari, A.B., A Software/ Reconfigurable Hardware SAT Solver, in: IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 12, n. 4, pp 408-419, 2004.

[12] Sklyarov, V., Hierarchical Finite-State Machines and Their Use for Digital Control, in: IEEE Transactions on VLSI Systems, vol. 7, no. 2, pp 222-228, 1999.

[13] Sklyarov, V., Skliarova, I., Architecture of a Reconfigurable Processor for Implementing Search Algorithms over Discrete Matrices, in: Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms - ERSA'2003, T.P. Plaks (ed.), Las Vegas, USA, June 2003, pp. 127-133.

[14] Sklyarov, V., Skliarova, I., Oliveira, A., Ferrari, A.B., A Dynamically Reconfigurable Accelerator for Operations over Boolean and Ternary Vectors, in: Euromicro Symp. on Digital System Design, Belek, Turkey, pp. 222-229, Sept. 2003.

[15] Sklyarov, V., Skliarova, I., Pimentel, B., Modeling and FPGA-based implementation of graph coloring algorithms, in: Proc. of the 3rd Int. Conf. on Autonomous Robots and Agents - ICARA 2006, New Zealand, Dec. 2006.

[16] Thorup, M., Combinatorial Power in Multimedia Processors, ACM SIGARCH Computer Architecture News, vol. 31, no. 5, December, 2003, pp.6-11.

[17] Velev, M.N., Bryant, R.E., Effective use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors, in: Proceedings of the 38th conference on Design automation, Las Vegas, USA, 2001, pp. 226-231.

[18] Vidyarthi, G., Ngom, A., Stojmenovic, I., Combinatorial Evolutionary Methods in Wireless Mobile Computing, in: Combinatorial Optimization in Communication Networks, Kluwer Academic Publishers, 2005.

[19] Zakrevski, A.D., Logical Synthesis of Cascade Networks, Moscow: Science, 1981.