Hierarchical Specification and Implementation of Combinatorial Algorithms Based on RHS Model*

Valery Sklyarov, Iouliia Skliarova, António B. Ferrari
skl@ieeta.pt, iouliia@ua.pt, ferrari@ieeta.pt
Department of Electronics and Telecommunications
University of Aveiro
3810-193 Aveiro, Portugal

Abstract

The paper introduces RHS model for combinatorial computations that describes partitioning of the problem between reconfigurable hardware (RH) implemented in FPGA and software (S) running on host computer and specifies the respective communication mechanisms. It is shown that such model is very efficient for the considered scope. It follows from two important characteristics of combinatorial problems. Firstly, each particular problem requires very limited number of task-specific operations that have to be repeated a huge number of times. Secondly, majority of operations is unique, i.e. task-oriented. The proposed specification method is based on graphical language that extends hierarchical graph-schemes.

1. Introduction

There are many practical applications, which require the solution of some combinatorial problems [1],[2],[3]. It is known that the majority of these problems are NP-hard and as a result they are time and resource consuming [2]. Because of that it is worthwhile to consider and to implement the respective accelerators such as coprocessors for general-purpose computers. It should be noted that the heterogeneous nature of combinatorial tasks makes it difficult to construct an effective specialized device, i.e. an ASIC. On the other hand, it is possible to analyze different combinatorial problems and select subsets of unique and common operations. As a result we can design a device including a fixed part (for common operations) and a reconfigurable part (for unique operations). This device might be constructed on the basis of reconfigurable hardware, such as commercially available FPGAs.

Most of the relevant work on this is in the area of virtual application-specific circuits that offer a mixture of hardware performance and software versatility. Bibliography related to this topic with abstracts and useful links can be found in [4]. In scope of combinatorial computations, hardware accelerators have been designed mainly for solving problems of Boolean satisfiability [5],[6],[7],[8]. There are many other problems in this area. For example, the paper [9] describes 44 various combinatorial tasks. Mainly they are computationally intensive. Thus the respective hardware accelerators could be considered as a very efficient tool allowing to reduce time of required computations.

As a rule, combinatorial tasks require executing a huge number of operations even for problems with relatively small volume of input data. Processing of the data is often performed with the aid of special search tree. Dedicated methods allow traversing the tree starting from his root, eventually pruning some branches that cannot lead to the desired solution. When we traverse the search tree, the volume of handled data is decreased (during the motion from the root to the leaves) and increased (during the backtrack) periodically. Since available hardware resources are often insufficient to solve the entire combinatorial problem, it is possible to employ the hardware just for treating subtasks appeared at various levels of the search tree, i.e. such subtasks that correspond to the presented hardware capacity.

Another important characteristic of combinatorial problems is that each one usually requires a distinctive set of operations. As a result it is necessary to modify the functionality of the hardware for solving different tasks. Thus the technique of dynamic reconfiguration might be very efficient. A variety of methods for run-time reconfiguration were considered in [10]. In general they assume dynamic swapping of replaceable blocks. These operations can be controlled by state machines. The papers [11],[12] have proposed a device with dynamically modifiable functionality

* This work was partially sponsored by the grant FCT-PRAXIS XXI/BD/21353/99

that might be used for problems of combinatorial computations. One part of the device has been fixed and the other has been built on the base of reprogrammable components. This device is intended to be used as a reprogrammable combinatorial processor (RCP). The RCP was designed in order to solve different problems formulated over logic matrices [13]. The reprogrammable part is used to implement such subset of operations that are unique for a given task. Since this subset has to be changed from one application to another the required modifications can be realized with the aid of a reprogramming technique. In fact there are two components, which have to be reprogrammed. The first component modifies the operations themselves and can be considered as an alterable processor core. We will call it reprogrammable function unit (RFU). The second component changes the control algorithm that forces the required sequence of operations to be executed and it can be seen as a reprogrammable control unit (RCU).

Experiments with the proposed RCP have shown that that best results might be achieved by rational partitioning of combinatorial algorithms in such a way that one part will be implemented in software of a general purpose computer and the other part will be implemented in an attached to host computer RCP. We will call such a model RHS, which denotes collaboration between Reconfigurable Hardware implemented in RCP and Software implemented in the host computer that is also responsible for modifications of RCP functionality in order to be able to implement various combinatorial algorithms on RCP with limited hardware recourses. The paper proposes a specification technique for such model and the design method based on problem-oriented templates for RFU and RCU.

The remainder of the paper is organized as follows. Section 2 suggests a modular specification of combinatorial algorithms. Section 3 discusses design methods for the RCP. Section 4 presents implementation details and the results of experiments. The conclusion is in section 5.

2. Modular Specification of Combinatorial Algorithms

Most combinatorial problems have discrete character and they can be formulated on such mathematical models as graphs, sets, discrete matrices, etc [3]. These models are often convertible in such a way that one model can be formally transformed into another. For example, in [13] it was shown that they might be converted to a universal matrix representation. Logic matrices are very well suited for processing them in FPGA [12]. This property has played an important role for the use of matrices as a basic mathematical model for combinatorial computations in RCP. The paper [9] presents a classification of different problems that can be solved on discrete matrices. Analysis of these problems shows that each of them requires quite a limited number of different operations. Their classification was performed in [12]. On the other hand as a rule various combinatorial problems require different sets of operations. It allows to conclude the following. First, the reconfigurable devices should be more profitable. Second the reconfiguration time should be negligible comparing with the time of combinatorial computations.

In order to simplify the reconfiguration process it was suggested to specify combinatorial algorithms with the aid of slightly modified hierarchical graph schemes (HGS) [14]. HGS allows to describe control algorithms in such a way that they will be composed of modules, which can be activated sequentially, hierarchically, and in parallel. Special communication methods, proposed in [15], enable the majority of control algorithms that are required in practical applications to be realized, including pipelining, various kinds of parallelism, and so on.

A general description of a HGS is the following. A HGS [14] is a directed connected graph containing rectangular, rhomboidal and triangular nodes. Each HGS has one entry point, which is a rectangular node named *Begin*, and one exit point, which is a rectangular node named *End*. Other rectangular nodes contain either *microinstructions* or *macroinstructions*, or both. Any *microinstruction* Y_j, includes a subset of *microoperations* from the set $Y=\{y_0,...,y_{N-1}\}$. A *microoperation* is an output signal, which causes a simple action in the attached datapath. Any *macroinstruction* incorporates a subset of *macrooperations* from the set $Z=\{z_0,...,z_{Q-1}\}$. Each *macrooperation* is described by another HGS of a lower level. Each rhomboidal node is used for bi-directional branching and contains one element from the set $X\cup\Theta$, where $X=\{x_0,...,x_{L-1}\}$ is the set of *logic conditions*, and $\Theta=\{\theta_0,...,\theta_{I-1}\}$ is the set of *logic functions*. A *logic condition* is an input signal, which communicates the result of a test. Each *logic function* is calculated by performing some predefined set of sequential steps that are described by an HGS of a lower level. Triangular nodes have one input and more than two outputs. They are used like rhomboidal nodes but for alternative multidirectional (i.e. 3 directions or more) branching. Directed lines (arcs) connect the inputs and outputs of the nodes in the same manner as for an ordinary graph-scheme [16].

Fig. 1 integrates various capabilities of HGSs. The primary component of HGS is a module, which can be also seen as a HGS. Any module can be activated from another module and it permits to describe a hierarchy. For example, in fig. 1 the module Γ_3 has been activated by macrooperation z_3 from the module Γ_0 and the module Γ_4 has been activated from the module Γ_3. If several (more than one) modules are being activated at the same time they will be executed in parallel (see the modules Γ_1 and Γ_2 in fig. 1). Any macrooperation might be virtual. In this case an actual link between this macrooperation and the particular module has not been specified statically. However this link can be established dynamically, i.e. during execution time [14] (see macrooperation z^v_5 in fig. 1). Any macrooperation can activate itself (for example, in fig. 1 macrooperation z_4 can activate the module Γ_4). It allows to describe recursive procedures. It is permitted to construct HGS from just two nodes that are *Begin* and *End* (see the left HGS in fig. 1). Such HGS describes nothing. Using this technique enables us to provide incomplete specification, which is very useful for debugging purposes.

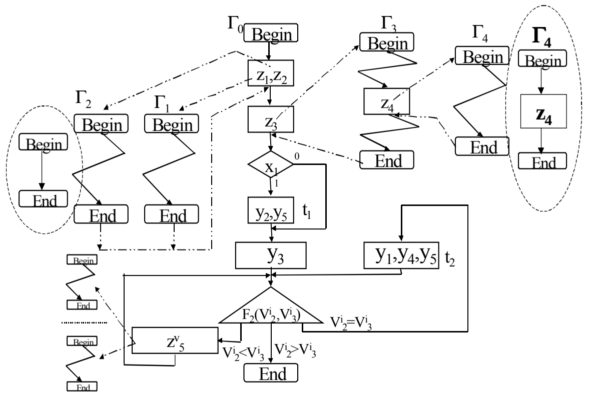

Fig. 1. Basic capabilities of HGSs.

A significant benefit of using HGSs is that any complex control algorithm can be developed step by step, so that our efforts can be concentrated at each stage at a specified level of abstraction (that is, on a particular element of the set $E=Z\cup\Theta$). Each component of the set E is usually very simple, and can be checked and debugged independently. Modular specification provides support for either top-down or bottom-up design. Top-down design is based on extending given modules by supplying new detail. Bottom-up design enables us to use more complicated components that are defined as modules and to organize the design process with the aid of libraries of modules. Any module can be reused. Extending or modifying a module does not change the existing specification. These and many other useful characteristics of HGSs demonstrate their fundamental advantages and make them very attractive for the specification of modifiable algorithms. Note that paper [14] suggested a formal method that enables us to translate HGS specification into the respective hardware.

Let us extend the set E in such a way that $E = E_{rh} \cup E_s \cup E_C \cup E_I$, where E_{rh} and E_s are the sets of modules that describe primary operations of the considered algorithm and they have to be implemented in reconfigurable hardware and in software respectively, E_C is a set of modules that provide dynamic reconfiguration of FPGA, E_I is a set of modules, which specify an interface between software and hardware components. Note that it is not necessary to implement all the modules from E_{rh} at the same time, i.e. only currently required modules might be loaded and they can be swapped with the aid of reconfiguration handler (i.e. with the aid of modules from the set E_C). The strategy of communication between software and hardware is the following.

1. The main module is being activated. Since at a specification level there is no difference between the considered above modules, the main module might be of any type (i.e. it might be any module from the extended set E). For majority of practical applications the main module is a software procedure and most frequently this is a module from the set E_s.
2. The software is being executed and it activates hardware when required (by means of modules from the set E_{rh}). Note that software and hardware might be run in parallel (the respective description capabilities will be considered later).
3. As soon as the subtask performed by the hardware is finished the hardware activates the respective module of the set E_I. From this time the selected interface module will be responsible for data exchange between software and hardware.
4. After completing operations mentioned in point 3 the hardware will be set in waiting state. Starting from this state it might be either reconfigured by the software (by a module from the set E_C) or a new subtask might be activated (see point 2). It has been already mentioned that this model of collaboration is called RHS.

Each module from the set E_I activates a sub-algorithm that handles the desired interface function. This sub-algorithm can be implemented partially in software and partially in hardware. From the side of software the required interface operations have been provided with the aid of C++ custom library functions. Hardware part has been implemented through components of the set $Z\cup\Theta$.

Fig. 2 integrates basic new facilities of the proposed specification method (it shows only nodes that will be used below). Let us assume that the main module is Γ_0. It can activate any other module from the set E. Each module can be any of the following types:

1. Continuous module, which means that after activation it will be executed continuously in a loop and in parallel with other operations of Γ_0. The loop is repeated until it is stopped by a special signal, which might be generated in any module from the set E. The respective technique for activation/deactivation of HGS modules was considered in [15]. Module Γ^1_5 in fig 2 has this type. Any continuous module Γ_i is activated by

macrooperation z^S_i (of the type S - set) and deactivated by macrooperation z^R_i (of the type R - reset). The similar superscripts are used for microoperations. Thus y^S_j sets the output j in an active state and y^R_j sets the output j in a passive state. The superscript D denotes activity of the selected output when the respective rectangular node is also active [15].

2. Interrupting modules, which interrupt the calling algorithm. Each module is executed once. If a few (more than one) modules have been activated from the same node, the calling algorithm will be suspended until all the modules will be finished. Modules Γ_1, Γ_2 and Γ_3 in fig 2 have this type. Any interrupting module Γ_i is activated by macrooperation z^D_i (of the type D)

3. Terminating modules are the same as modules considered in point 1 but they will be executed only once. Module Γ^2_5 in fig 2 has this type.

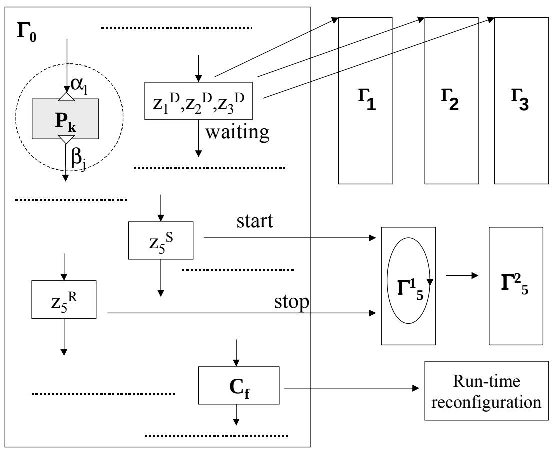

Fig. 2. Extended capabilities of HGSs.

Let us assume that Γ_0 is being executed in hardware. If necessary it can activate any module from the set E. In fig. 2 $P_k \in E_s$, $\alpha_l, \beta_j \in E_I$, $C_f \in E_C$. In case if components of the set E_I have direct link with a component P_k of the set E_s we will use triangles on input and output of the node P_k (see fig. 2). "Direct link" denotes interface functions that provide data exchange between hardware and the respective software modules (such as Γ_0 and P_k).

3. Design methods

The architecture of RCP was proposed in [12]. It is composed of RFU and RCU. The RFU is responsible for three designated groups of operations that are Boolean, composite (such as counting operation) and verification in form of yes/no. They are implemented in RAM-based core and the modifiability is achieved by reprogramming the respective sections of RAM. The model of RCU is a RAM-based finite state machine (FSM) [17]. Note that such FSM can be synthesized from the specification to be obtained in the previous section. Thus the changes in a sequence of combinatorial operations can be also provided by the reloading of RAM. RFU and RCU have been implemented on the base of hardware templates. The respective technique was discussed in [12],[18]. A distinctive feature of this paper is an integrating of previously exploited methods with RHS model of computations.

The design flow is based on the following methods and tools:

1. Specification of combinatorial algorithms with the aid of extended HGSs (see section 2). In order to evaluate the proposed specification method two well-known combinatorial problems have been considered. They are covering [2] and satisfiability [5],[6],[7],[8]. Let **M** be a discrete matrix that is Boolean for the first and ternary for the second problem, $V_0,...,V_{A-1}$ are columns of **M**, $H_0,...,H_{B-1}$ are rows of **M**. The respective combinatorial algorithms will include the following elementary operations:

- checking for orthogonality between two rows (columns), i.e. H_i ort H_j ($i \neq j$, $H_i,H_j \in H=\{H_0,...,H_{B-1}\}$) or V_i ort V_j ($i \neq j$, $V_i,V_j \in V=\{V_0,...,V_{A-1}\}$);
- counting the number of ones in rows and columns;
- testing if one row/column covers another row/column.

The modules of combinatorial algorithms have been constructed from a sequence of elementary operations that designate a macro action of the algorithms, for example

- checking if a row H_i is orthogonal to all rows from the set $H\backslash\{H_i\}$;
- selecting all columns that are covered by the selected row;
- reducing the matrix **M** by removing the selected row and all columns that are covered by this row, etc.

One module can activate another one, for example reducing **M** includes discovering all columns that are covered by the selected row. Using this technique we can construct modular and hierarchical specification.

2. Partitioning of specification in hardware and software. For exploratory purposes the following approach has been used. We have already mentioned that the design is based on hardware templates implemented in FPGA [12]. Each template has pre-defined constraints, such as the maximum value of A and B for **M**, the maximum number of states, inputs and outputs for RAM-based FSM, etc. Taking into account these parameters we assume that each combinatorial algorithm has to be implemented in software until the step in which all the constrains will be satisfied. After that step the hardware will be responsible for dealing with the problem. The most common approach to solving combinatorial problems is

based on construction of a search tree. The root of the tree represents an initial point, i.e. an initial situation that has to be considered and examined. All other nodes of the tree correspond to situations that can be reached during the search process. If we construct the complete tree we will be able to find out a solution or to conclude that it does not exist. The RHS model allows to incorporate the following strategy:

- the root of tree is the starting point;
- if **M** satisfies the pre-given constraints the problem will be completely solved in hardware;
- if **M** does not satisfy pre-given constraints the software will apply special splitting and reduction methods until an intermediate matrix, which has to be constructed at the current search step, will not satisfy the pre-given constraints. After that the hardware will be responsible for the subsequent steps;
- if hardware finds a solution the problem is considered to be solved and the result will be dispatched to the host computer;
- if the considered branch of the tree does not allow to find a solution the control will be returned to software;
- if the size of current intermediate matrix exceeds the constraints the execution will be continued in software;
- the considered steps will be repeated until we receive either negative or positive result, i.e. we will get the solution or we will conclude that the problem does not have any solution.

Fig. 3 shows an example of a search tree for discovering of minimal column cover of Boolean matrix with the aid of algorithm [2].

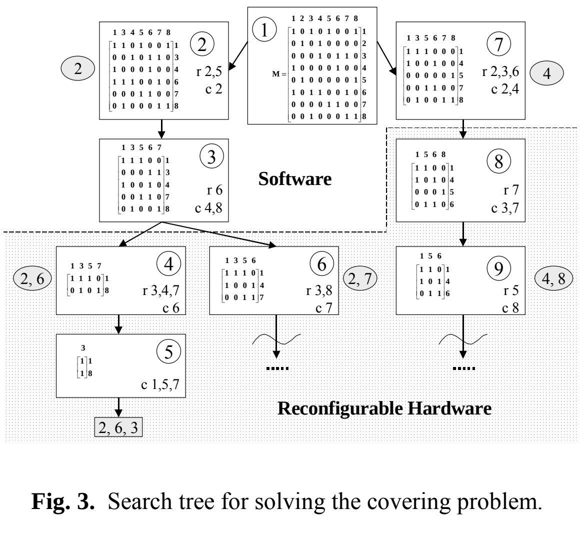

Fig. 3. Search tree for solving the covering problem.

This algorithm assumes sequential applying special reduction and splitting methods. The first methods permit to simplify the initial and intermediate matrices by means of deleting of some rows and columns, which cannot lead to the desired solution because of their specific properties. When the reduction methods cannot be applied the algorithm examines possibilities of including various columns in the constructed covering. The selection of these columns is based on the proposed in [2] heuristic criteria. The numbers of vertices in the search tree (see fig. 3) reflect the order of considered situations. Shaded circuits point to the columns that are included in the covering during the search process. After any covering is found it is considered as a constraint for the next steps of the algorithm (i.e. only coverings of smaller cardinality will be analyzed). Fig. 3 assumes that the maximum size of matrices handled in hardware template is equal to 4*4. Thus the upper part of the search tree is treated in software and the lower part is processed in reconfigurable hardware.

3. Implementing hardware components based on hardware templates that have been constructed with the aid of Xilinx Foundation Software [12],[18]. The template for RFU permits to implement operations [12] considered at the beginning of this section. The template for RCU allows to realize any desired control algorithm within pre-given constraints [18]. Note that RFU has universal structure and it is not necessary to run the synthesis tools. The software has to reload the RAM-based blocks in order to implement the desired set of elementary operations. The sequence of operations in modules is specified by the respective segments of HGS. The designed tools allow to customize the template for RCU, i.e. to load the RAM in such a way that the sub-algorithm for considered module will be implemented. Any module is included in problem-oriented library and can be reused.

4. The software part was described in Visual C++ running under Windows. It provides habitual user interface supported by the MFC library.

5. Reconfiguration of hardware can be performed when it is set in waiting state.

4. Implementation details and experiments

We have considered two different implementations of RCP. The first one was based on XS40 board (XStend V1.2) of XESS that contains the Xilinx XC4010XL FPGA. Reconfigurable circuits were constructed from FPGA CLBs configured as RAM. For dynamic modifications we used dual port RAM library components, such as RAM 16X4D. Thus the first port took part in the FSM state transitions whilst the second port was used to reprogram RAM from the PC via parallel port. Since the board XS40 communicates with the host computer through parallel interface it is obvious that we cannot gain any advantage comparing with pure software implementation of combinatorial algorithms. However this realization has allowed to verify the proposed technique of template-based design and separate hardware modules for various segments of combinatorial algorithms. This approach was used in order to solve covering problem that has been specified on discrete matrices [12]. The second implementation was based on Alpha Data ADM-XRC PCI board that contains one Virtex FPGA XCV812E. The design process was supported by the respective library [19]. The developed software and hardware for RCP includes C++ programs running under Windows and a set of hardware modules designed with the aid of Xilinx Foundation Software (release 3.1i). The communication with the board was supported by ADM-XRC library functions supplied by Alpha Data [19]. This approach was used in order to solve a satisfiability problem. Hardware implementation allowed to solve the problem on matrix **M** with the following constraints: $A \leq 32$, $B \leq 32$. We compared the results of the proposed RHS model with pure software implementation (PC computer with AMD Athlon/1GHz/256MB was used). For hole7-hole10 benchmarks from the DIMACS [20] that are all unsatisfiable and thus can be characterized by very high computational complexity, the RHS model gave better results. Dependently on particular task the speed was increased from 1,5 to 2.5 times. We consider these results as preliminary and we expect that they can be essentially improved in future. It is very important to note that the presented data include configuration time of FPGA.

5. Conclusion

The paper proposes and analyses the specification method and the RHS model of computations that describes collaboration of software and reconfigurable hardware. Hardware was implemented in FPGA on the base of problem-oriented templates. Each template can be customized in order to solve tasks within pre-given constraints. FPGA-based circuit is considered to be a hardware accelerator for general-purpose computer. Software is running on host computer and it is responsible for a part of computations and for reconfiguration of hardware. Specification method is based on graphical language that extends hierarchical graph-schemes. The considered technique has been tested for covering and satisfiability problems in such a way that sub-tasks satisfying pre-given constraints were implemented in hardware and other sub-tasks - in software. The results of experiments have shown that the proposed model has advantages comparing with the other approaches.

References

[1] G.De Micheli. “Synthesis and Optimization of Digital Circuits”, McGraw-Hill, Inc., 1994.

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

[3] I.Skliarova, A.B.Ferrari, "Modelos matemáticos e problemas de optimização combinatória”, Electrónica e Telecomunicações, v.3, Nº 3, January, 2001, pp.202-208.

[4] http://www.ife.ee.ethz.ch/~enzler/rc/ bib_abstracts.html.

[5] P.Zhong, et al, “Using Reconfigurable Computing Techniques to Accelerate Problems in the CAD Domain: A Case Study with Boolean Satisfiability”, Proc. of the 34th Design Automation Conference, 1998.

[6] M.Platzner, “Reconfigurable Accelerators for Combinatorial Problems”, IEEE Computer, pp. 58-60, April 2000.

[7] J.T. de Sousa et al., “A Configware/Software Approach to SAT Solving”, Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines - FCCM’2001.

[8] M.Boyd, T.Larrabee, “ELVIS – A Scalable, Loadable Custom Programmable Logic Device for Solving Boolean Satisfiability Problems”, Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines - FCCM’2000.

[9] A.D.Zakrevski, “Combinatorial Theory of Logical Design”, Automatics and computer techniques, Nº2, 1990, pp. 68-79.

[10] N.Shirazi, W.Luk, P.Y.K.Cheung. “Run-Time Management of Dynamically Reconfigurable Designs. Programmable Logic and Applications”, 8th International Workshop FPL'98, Springer, 1998, pp. 59-68.

[11] I.Skliarova, A.Ferrari. “Exploiting FPGA-Based Architectures and Design Tools for Problems of Combinatorial Computations”. Proceeding of SBCCI'2000, Manaus, Brazil, 2000, pp. 347-352.

[12] I.Skliarova, A.B.Ferrari, "Development tools for problems of combinatorial optimization", Proceedings of the 4th Portuguese Conference on Automatic Control (CONTROLO'2000), Guimarães, Portugal, October, 2000, pp. 552-557.

[13] A.Zakrevskij, “Combinatorial Problems over Logical Matrices in Logic Design and Artificial Intelligence”, Electrónica e Telecomunicações, vol. 2, no. 2, pp. 261-268.

[14] V.Sklyarov, “Hierarchical Finite-State Machines and Their Use for Digital Control”, IEEE Transactions on VLSI Systems, 1999, Vol. 7, No 2, pp. 222-228.

[15] V.Sklyarov, “Graphical Description and Hardware Implementation of Parallel Control Algorithms”, Proceedings of PDPTA'99, June, Las Vegas, USA, 1999, pp. 1390-1396.

[16] S.Baranov, “Logic Synthesis for Control Automata”, Kluwer Academic Publishers, 1994.

[17] V.Sklyarov, "Synthesis and Implementation of RAM-based Finite State Machines in FPGAs", Proceedings of FPL’2000, Villach, Austria, August, 2000, pp. 718-728.

[18] I.Skliarova, A.B.Ferrari, "Synthesis of reprogrammable control unit for combinatorial processor”, Proceedings of the 4th IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems – IEEE DDECS’2001, Gyor, Hungary, April, 2001, pp.179-186.

[19] http://www.alphadata.co.uk

[20] http://www.intellektik.informatik.tu-darmstadt.de/ SATLIB/Benchmarks/SAT/DIMACS/