

Combinatorial Optimization for Intelligent Control Systems

I. Skliarova

Department of Electronics and Telecommunications, IEETA
University of Aveiro, 3810-193 Aveiro, Portugal
iouliia@det.ua.pt

ABSTRACT

Combinatorial problems that have to be solved in modern intelligent control systems frequently possess exponential worst-case complexity. This fact precludes the solution of many practical problems with conventional processors. Specialized hardware based implementations are also not viable, in particular because of the inherent heterogeneity of combinatorial problems. Reconfigurable devices offer an alternative solution, which can be customized to the requirements of a specific algorithm and reutilized for other algorithms via a simple reprogramming of their internal structure. In this paper different ways of designing reconfigurable combinatorial accelerators are explored.

1. INTRODUCTION

Nowadays, intelligent control systems are used in many applications covering every aspect of human life. Examples of these applications are automotive industry, aerospace electronics, biomedicine, household appliances, etc. In intelligent control systems, real-world problems are formulated over simplified mathematical models, such as graphs, matrices, sets, logic functions and equations, etc. Then, by applying mathematical manipulations to the respective model, a solution to the original problem is obtained. The involved mathematical manipulations differ according to the system type, but frequently they recur to combinatorics and require the solution of various combinatorial problems. Typical examples of these problems are finding the shortest path in a graph, graph-coloring, logic function optimization, set covering, etc.

Many of the involved combinatorial optimization problems belong to the classes *NP-hard* and *NP-complete*, which implies that the relevant algorithms have an exponential worst-case complexity, imposing consequently very high computational requirements on the underlying implementation platform. This fact precludes the solution of many practical problems with conventional processors. This is because conventional processors are programmed with instructions selected from a predefined set, which are combined to encode a given algorithm. The use of conventional processors is justified for problems where their performance is adequate since the design cost is very low. Besides, any required change in the algorithm can easily be incorporated in the respective implementation. However, since the conventional processors are not optimized for solving the combinatorial problems, the resulting performance is very scant.

As opposed to the previous approach, a hardware-based solution can be tailored to the requirements of a given algorithm guaranteeing in this way an optimal performance. However, a specialized hardware circuit is only capable of executing a task, for which it has been designed, whereas a conventional processor might be reutilized for different tasks via a simple modification of instruction sequence. This software/hardware compromise obligates designers to trade off between performance and flexibility.

The development of dedicated hardware systems for specific problems and domains involves considerable cost and design time. The experience shows that the resulting benefits are often scant and even non-existent, because of the current rate of evolution of conventional processor technology, which enforces the supplantation of specialized and optimized computing structures by those that are less efficient for a given application domain. Besides, the proper heterogeneity of combinatorial problems discourages from developing specialized hardware accelerators.

The invention of high capacity field-programmable logic devices, such as *field-programmable gate arrays* (FPGA), set up an alternative method of computing. The typical modern FPGA architecture is presented in Fig. 1. An FPGA is composed of an array of programmable logic blocks interlinked by

programmable routing resources and surrounded by programmable input/output blocks. The logic blocks include combinational and sequential elements allowing both logic functions and sequential circuits to be implemented. The routing resources are composed of predefined routing channels interconnected by programmable routing switches. A logic circuit is implemented in an FPGA by distributing a logic among individual blocks and interconnecting them subsequently by programmable switches. Recent FPGA incorporate also various heterogeneous structures, such as dedicated memory blocks, embedded processor cores, multipliers, transceivers, etc., which allow for the implementation of systems-on-chip.

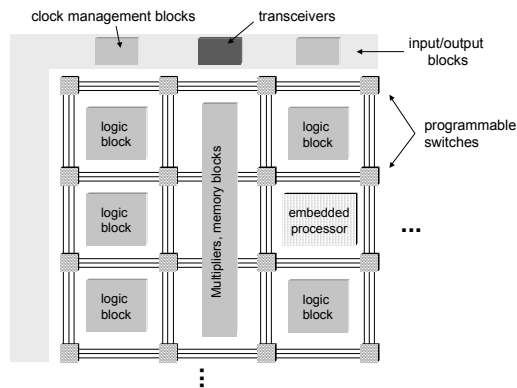


Fig. 1. Typical FPGA architecture.

The FPGA enable attaining both the hardware performance and the flexibility of software, since they can be optimized for executing a specific algorithm and reutilized for other algorithms via a simple reprogramming of their internal structure. As a result, control engines can be constructed that are optimized for a given application via *reprogramming* the functionality of basic FPGA logic blocks, i.e. without introducing any changes on the “hardware” level. Implementations based on reconfigurable hardware permit the execution of the relevant algorithms to be optimized with the aid of such techniques as parallel processing, personalized functional units, optimized memory interface, etc.

The applications, for which FPGA-based reconfigurable systems have been constructed, cover diversified domains such as image processing, search engines in genetic databases, pattern recognition, neural networks, high-energy physics, etc. The best performance was achieved for such applications that exhibited a high level of parallelism, had large amounts of data to process, and used a non-standard format of information representation. The combinatorial problems possess all these characteristics and are therefore eminently suitable to FPGA-based implementations. Such implementations, tailored to combinatorial problems, would allow the exponential growth in the computation time to be delayed, thus enabling more complex problem instances to be solved.

2. RECONFIGURABLE COMBINATORIAL ACCELERATORS

In general, it should be noted that it is not feasible to realize practical combinatorial algorithms entirely in an FPGA. This is because the reconfigurable logic is not so well suited for some computations, for instance, for complex sequential operations, which might be better realized in conventional processors. Besides some fragments of combinatorial algorithms are rarely activated and we can predict for such fragments low effectiveness of FPGA-based solutions. So conventional processors are more appropriate for realizing non-regular computations. On the other hand FPGAs are more suitable for regular (repeated) treatments of large volume of data. Thus, the best result might be achieved by rational combination of processor and FPGA resources. Subroutines that can benefit from hardware implementation are mapped to the FPGA, while others are computed in software.

2.1 Specification Methods

Three different specification methods have been employed for the description of the respective hardware problem solvers: a schematic entry, a hardware description language, and a general-purpose

programming language. The schematic-based approach is probably not appropriate because instead of thinking in terms of algorithms and data structures it forces the designer to deal directly with the hardware components and their interconnections. Contrariwise, the *hardware description languages* - HDLs (such as VHDL and Verilog) are widely used for specification of combinatorial algorithms [1] since they typically include facilities for describing structure and functionality at a number of levels, from the more abstract algorithmic level down to the gate level. The general-purpose programming languages, such as C and C++, have also been employed, being the respective descriptions transformed (by specially developed software tools [2]) to an HDL, which was used for synthesis. The higher portability and the higher level of abstraction of language-based specifications have determined their popularity and widespread acceptance.

Recently, commercial tools that allow digital circuits to be synthesized from *system-level specification languages* (SLSLs) such as Handel-C [3] and SystemC [4] have appeared on the market. In this area, C and C++ with application-specific class libraries and with the addition of inherent parallelism are emerging as the dominant languages in which system descriptions are provided. This fact allows the designer to work at a very high level of abstraction, virtually without worrying about how the underlying computations are executed. Consequently, even computer engineers with a limited knowledge of the target FPGA architecture are capable of producing rapidly functional, algorithmically optimized designs.

Obviously, the higher level of abstraction leads to some performance degradation and not very efficient resource usage, as evidenced by a number of examples [5]. On the other hand SLSLs have many advantages such as portability, ease to learn (any one familiar with C/C++ will recognize nearly all features of SLSLs), ease of change and maintenance, and a very short development time. Therefore, we can expect that as the tools responsible for generating hardware (more specifically, either an EDIF – *electronic design interchange format* file or an HDL file) from system-level source code advance, the SLSLs may become the predominant hardware description methodology, in the same way as general-purpose high-level programming languages have already supplanted microprocessor assembly languages.

2.2 Accelerator Types

Recently, a few reconfigurable engines for combinatorial problems have been developed. Many of such problems are well suited to parallel and pipelined processing. Therefore, FPGA-based implementations can potentially lead to drastic performance improvements over traditional processors. In this context, it is important to make a distinction between the following three accelerator types (see Fig. 2):

- *instance-specific accelerators* require generation of a special configuration of FPGA for each individual problem instance. In this case, a typical design flow is used to describe, synthesize, and implement either a whole instance-specific circuit or a number of precompiled primary modules, which are further customized by specially developed software tools to match the respective problem instance. Such an approach permits performance to be increased and provides a good utilization of the resources. However the required time of hardware compilation is significant and frequently it is even higher than the time for the execution of combinatorial algorithms in hardware.
- *application-specific accelerators* can potentially be used for solving any instance of a given problem. In this case the circuit is designed and optimized only once, after which it is adapted for solving different problem instances. This can be achieved with the aid of a *template* circuit, which is downloaded to an FPGA and customized directly there with data for a particular problem instance. It should be emphasized that in this case a hardware compilation step is completely avoided and, consequently, greater acceleration over software can potentially be achieved.
- *domain-specific accelerators* enable a variety of problems and their instances in the area of combinatorial computation to be solved. This can in particular be achieved with the aid of a RAM-based template circuit, whose functionality is determined by reprogramming the contents of RAM.

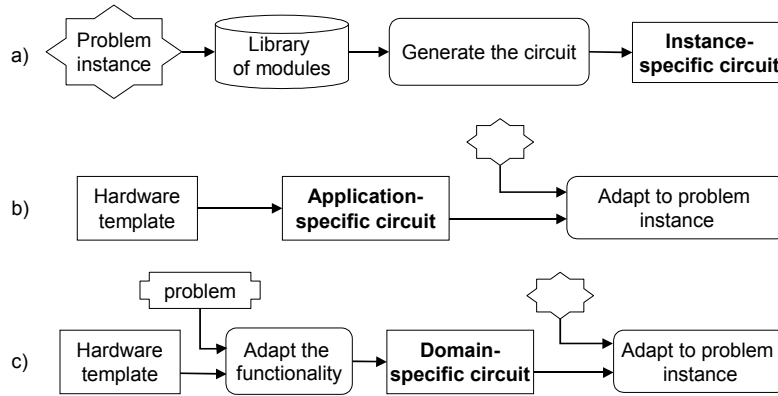


Fig. 2. Instance-specific (a), application-specific (b) and domain-specific (c) reconfigurable combinatorial accelerators.

Although the idea of using accelerators based on reconfigurable hardware seems to be obvious, there exist a number of significant obstacles. For example, in designing instance-specific accelerators the primary objective is to reduce the configuration generation time; in application-specific accelerators the major questions are how to perform fast swapping between different instances and how to process problem instances whose dimensions exceed the available hardware resources; in domain-specific accelerators the important challenge is how to accommodate efficiently a variety of different problems so as the required resources do not expand in an uncontrolled way.

The majority of researches in the area of accelerating combinatorial problems with the aid of reconfigurable hardware apply an instance-specific approach. In this paper two alternative approaches are explored. The first, *domain-specific*, approach enables a variety of problems in the area of combinatorial computation to be addressed. For this purpose, a reconfigurable combinatorial processor has been designed and implemented and a number of methods and tools that support its partial dynamic reconfiguration have been developed. The second, *application-specific*, approach is oriented towards solving individual combinatorial problems. In particular, a novel architecture is described for solving the Boolean satisfiability problem with the aid of software and reconfigurable hardware. The adopted technique avoids instance-specific hardware compilation and permits problems that exceed the available logic resources to be solved.

3. DOMAIN-SPECIFIC ACCELERATORS

The domain-specific approach is based on a *reconfigurable combinatorial processor* (RCP) that can be used for solving combinatorial tasks formulated over discrete matrices [6]. Due to the heterogeneity of combinatorial tasks, the RCP must be dynamically reconfigurable. In other words, we must be able to modify the processor's functionality at run time. In order to reduce the reconfiguration time, the RCP has to be based on a *hardware template* (HT). In this case only the basic computational operations and the corresponding control algorithms can be altered. All the other components and connections between them will not be changed. In order to do this we propose using a RAM-based HT in such a way that customizing the functionality of the RCP is achieved by reloading RAM-based blocks. Consequently, the RCP has to be built on the basis of an FPGA with distributed memory cells (e.g. LUTs) or embedded memory blocks, such as the Spartan and Virtex families from Xilinx [7]. The distributed cells grouped in blocks of required sizes can be used to store matrices, much like the way the general-purpose registers of a conventional processor store operands and the results of intermediate computations. The embedded memory blocks can also be used to implement the components of a HT with modifiable functionality. The reconfiguration is supported by auxiliary circuits that control the reloading of the RAM-based blocks.

Fig. 3 depicts the structure of RCP, which consists of two major parts: a *reconfigurable control unit* (RCU), and a *reconfigurable functional unit* (RFU). The shaded blocks in Fig. 3 can be

customized for a particular application. The remaining blocks possess fixed functionality and perform common tasks that are shared by many different combinatorial problems.

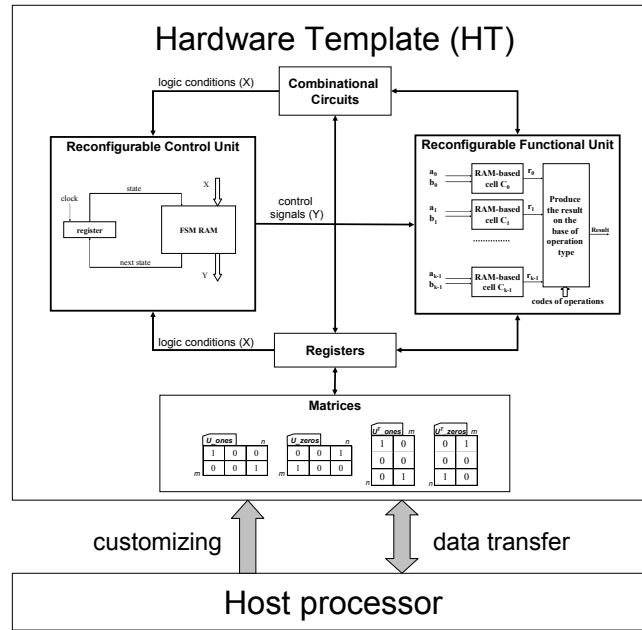


Fig. 3. Architecture of RCP.

Let us describe each reconfigurable block from Fig. 3 in more detail.

3.1 Reprogrammable matrices

The *Matrices* block can store up to three logical matrices with dimensions $m \times n$. For each logical matrix, \mathbf{U} , two physical copies are constructed, the first of which stores the original matrix, \mathbf{U} , and the second stores its transpose, \mathbf{U}^T . Of course, such an approach requires double the resources needed to store just the matrix data. However, the RCP performance is improved significantly because each row and column can be read in just one clock cycle. Each physical matrix is composed of two blocks of equal dimensions: U_ones and U_zeros . The blocks U_ones (U^T_ones) contain 1s in the positions where the original logical matrix \mathbf{U} (\mathbf{U}^T) has 1s and contain 0s in all the other positions. Before execution the relevant matrix data are transferred to the *Matrices* block. When some other problem instance is to be solved, the block can easily be loaded with different data.

3.2 Reconfigurable Control Unit

The RCU implements the control algorithms that are required. The unit is modeled by a *finite state machine* (FSM) with dynamically modifiable behavior that generates the sequence of operations for the combinatorial algorithm being executed. An FSM might be presented at the structural level as a composition of a combinational circuit that calculates the next states and outputs, and a register that stores the current state (see the block RCU in Fig. 3).

We are considering a RAM-based FSM, i.e. an FSM for which the combinational circuit is constructed from RAM-based blocks [8]. Any state code from the register is combined with input variables (i.e. logic conditions from the set X) and the result forms an address in the FSM RAM (see Fig. 3). Each address accesses a word in the FSM RAM that contains both the code for the corresponding next state and the outputs (i.e. the control signals from the set Y). Such an architecture is easily reprogrammable since reloading the contents of the FSM RAM changes the functionality of the RCU. In order to reduce the size requirements of the RCU, a special state encoding technique was employed that allows the functional dependency of outputs on inputs to be diminished [8]. The technique relies on combining the inputs from the set X with state codes, enabling the depth of the FSM RAM to be reduced.

To support reprogrammability, the RCU is decomposed into several RAM-based sub-blocks. As a result, the RCU is based on a parameterizable HT that has some predefined constraints. These constraints restrict the sizes of the respective RAM-based blocks, and consequently determine the maximum number of FSM states, inputs, outputs, and conditional state transitions that can be accommodated [8]. It is very easy to reprogram this device. For such purposes it is sufficient to reload the contents of the RAM-based blocks. We have developed special software tools [8] that allow a given behavioral specification of the control algorithms to be translated into the RAM contents, assuming that the RCU is based on the predefined HT.

3.3 Reconfigurable Functional Unit

The RFU is composed of memory elements and circuits needed to store and process the variables of the algorithm. Basic computations over columns and rows of discrete matrices are executed in the *reconfigurable core* shown in Fig. 3. The core is composed of a number of RAM-based blocks (the number of blocks, k , is equal to $\max(m,n)$). Each block performs an operation over one or two 2-bit values and calculates a 2-bit result. The first bit comes from one of the blocks U_ones or U^T_ones , and the second bit is from the blocks U_zeros or U^T_zeros .

In order to implement different operations, it is necessary to reload the appropriate group of RAM-based blocks. Three groups of operations have been proposed: Boolean operations, such as $\mathbf{a} \wedge \mathbf{b}$; operations that require an answer in the form YES/NO, for example “test if \mathbf{a} is orthogonal to \mathbf{b} ”; and counting operations, such as calculating the number of zeros in a Boolean vector [9].

3.4 Implementation of the RCP

Two variants of the RCP have been designed, implemented, and tested. The first variant was implemented based on the XStend board from XESS [10] containing one Xilinx XC4010XL FPGA. The reprogrammable blocks of the RCP were constructed from LUTs available in this FPGA. The reconfiguration was carried out through the parallel port. Since the FPGA employed has very restricted resources and reconfiguration via a parallel interface does not provide much flexibility, this implementation has only been used for verification purposes and for some experiments.

The second variant of RCP architecture was implemented subsequently using the ADM-XRC PCI board [11]. This board contains one XCV812E Virtex Extended Memory FPGA with approximately 254K logic gates and embedded memory blocks that provide for a total capacity of more than 1Mbits [7]. Interaction with the FPGA is carried out with the aid of the ADM-XRC API library, which provides support for initialization, loading configuration bitstreams, data transfers, interrupt processing, clock management and error handling.

For configuring the FPGA the following model has been proposed. Basic functions of combinatorial algorithms (for instance, *find-max-column*, *find-ort-row*, etc.) are included in a parameterized library. The system-level specification is prepared in the C++ programming language, i.e. a combinatorial algorithm is described in C++ using the library mentioned above. The assisting software tools extract the corresponding configuration from the library and download it to the FPGA when required. Note that the basic HT is loaded from the very beginning, so it is only necessary to reprogram the alterable components in the RCU and RFU. Finally the matrix data are passed to the FPGA for processing. When the computation has been completed, the assisting software tools will store the intermediate results and program execution will proceed, eventually reaching another hardware library function forcing a similar sequence of actions, i.e. the process considered above will be repeated.

The designed and implemented RCP was used for solving two combinatorial problems, the Boolean satisfiability and the covering problems. The results of experiments on randomly generated problem instances have shown that the RCP can achieve a significant speedup compared to the solution in software (up to 20x - 400x) [6].

4. APPLICATION-SPECIFIC ACCELERATORS

As mentioned in Section 2, hardware compilation time restricts the range of problems where the instance-specific approach can be useful compared to software solution. In order to eliminate this constraint we have attempted to design a universal circuit, or hardware template, whose structure is

not changed for different problem instances. We have selected the Boolean satisfiability (SAT) problem for our experiments.

We formulated the SAT problem over a ternary matrix \mathbf{U} by setting a correspondence between clauses of a Boolean formula and rows of \mathbf{U} , and between variables and columns of \mathbf{U} . Note that the problem of satisfying the Boolean formula is equivalent to finding a ternary vector \mathbf{v} , which is orthogonal to each row of the corresponding matrix \mathbf{U} . In order to solve the SAT problem formulated in terms of a ternary matrix we applied a backtracking search algorithm, which is based on the well-known Davis-Putnam procedure. The proposed satisfier architecture is depicted in Fig. 4.

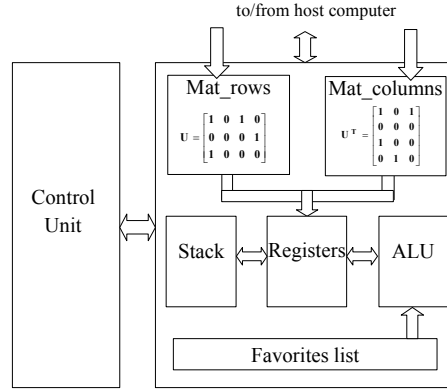


Fig. 4. The SAT solver architecture.

The central control unit executes the employed algorithm. The control unit is described in VHDL and is implemented as a finite state machine. The block *Registers* stores information about the active row and column, the current decision level, the partially computed vector \mathbf{v} , the deleted rows and columns, etc.

There are four RAM blocks storing matrix data. The first two correspond directly to the matrix \mathbf{U} and the remaining blocks represent a transpose of \mathbf{U} . The maximal dimensions of matrix \mathbf{U} have been fixed and they are equal to $m_{\max} \times n_{\max}$. If it is necessary to process a matrix of smaller dimensions then the actual parameters, m_{act} and n_{act} , will be stored in two special registers. Taking into account these values the control unit will only enforce the handling of the required area of the matrices.

The ALU is used to calculate the number of ones and zeros in rows and columns of the matrix, to determine whether any row is orthogonal to the vector \mathbf{v} , etc.

The stack memory supports the backtracking process. During the forward search, when the algorithm performs a decision, the current values from the registers are stored at the respective addresses in the RAM blocks, and the stack pointer is incremented. During the backtracking process, the stack pointer is decremented and the required values are restored, i.e. the data is moved from the RAM blocks back to the registers.

The favorites list contains permitted partial assignments, which are pre-calculated in software with the aid of a specially developed hybrid algorithm [12]. At each node of the decision tree the current partial assignment \mathbf{v} is checked for consistency with the favorites list. If it is consistent, the search process proceeds as usual. In the opposite case it becomes clear that the current branch of the decision tree will not lead to the solution, so the algorithm should backtrack. As a result, so-called *premature backtracking* is performed. Although this technique is very simple, it has a great influence on the execution time, because it allows to reduce essentially the number of visited nodes in the decision tree.

In order to solve various problem instances it is only necessary to download the respective matrix data. All the other components of the satisfier remain unchanged. It allows utilizing local reconfigurability and reduces the configuration overhead.

The satisfier was implemented on an ADM-XRC PCI board containing one XCV812E Virtex Extended Memory FPGA [7], which is very well suited to the proposed architecture of a SAT solver because we can use a large amount of RAM blocks to store matrices.

One very important issue affecting any algorithm implemented on reconfigurable hardware is related to the capacity of the hardware platform. Because of that we developed a special strategy that

allows to partition the problem between software and reconfigurable hardware [13]. In this case, an FPGA is only responsible for processing sub-problems that appear at various levels of the decision tree and satisfy the imposed hardware constraints (such as the maximum allowed number of rows and columns in the matrix). This technique permits problems to be solved that exceed the resources of the available reconfigurable hardware.

The results of experiments on some of DIMACS benchmarks [14] have shown that using our approach it is possible to achieve a significant speedup compared to the state-of-the-art software SAT solver GRASP (up to two orders of magnitude, including the FPGA configuration time, being GRASP executed on an AMD Athlon/1GHz/256MB).

5. CONCLUSION

Reconfigurable hardware supplies very vast opportunities for implementing effective engines targeted at accelerating intelligent control systems processes. This is because FPGA offer fast time to market, low design and manufacturing cost and risk, extremely high processing performance and easy configurability. All these features, when combined, allow the FPGA to supplant ASICs and conventional processors in a number of applications, especially in low-volume designs that use specialized data processing algorithms and impose certain power and security constraints.

In this paper we described two possible ways of constructing reconfigurable combinatorial cores. Namely, we have explored domain-specific accelerators and application-specific accelerators. The results of experiments show that the developed circuits can efficiently be used for solving moderate size real-world problems appearing in intelligent control systems.

REFERENCES

- [1] P. Zhong, Using Configurable Computing to Accelerate Boolean Satisfiability, Ph.D. dissertation, Department of Electrical Engineering, Princeton University, 1999.
- [2] O. Mencer and M. Platzner, "Dynamic Circuit Generation for Boolean Satisfiability in an Object-Oriented Design Environment", in Proc. 32nd Hawaii Int. Conf. on System Sciences, 1999.
- [3] Handel-C, <http://www.celoxica.com>.
- [4] SystemC, <http://www.systemc.org>.
- [5] E.M. Ortigosa, P.M. Ortigosa, A. Cañas, E. Ros, R. Agís, and J. Ortega, "FPGA Implementation of Multi-layer Perceptrons for Speech Recognition", in Proc. 13th Int. Conf. on Field-Programmable Logic and Applications – FPL'2003, Lisbon, Portugal, pp 1048-1052, Set. 2003.
- [6] I. Skliarova and A.B. Ferrari, "The Design and Implementation of a Reconfigurable Processor for Problems of Combinatorial Computation", Journal of Systems Architecture, Special Issue on Reconfigurable Systems, vol. 49, nos. 4-6, pp. 211-226, 2003.
- [7] Xilinx, <http://www.xilinx.com>.
- [8] I. Skliarova and A.B. Ferrari, "Synthesis of reprogrammable control unit for combinatorial processor", in Proc. 4th IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems – IEEE DDECS'2001, Gyor, Hungary, pp. 179-186, Apr. 2001.
- [9] I. Skliarova and A.B. Ferrari, "Development tools for problems of combinatorial optimization", in Proc. 4th Portuguese Conference on Automatic Control - CONTROLO'2000, Guimarães, Portugal, pp. 552-557, Oct. 2000.
- [10] XESS Corp., <http://www.xess.com/>.
- [11] Alpha Data, <http://www.alpha-data.com>.
- [12] I. Skliarova and A.B. Ferrari, "A hardware/software approach to accelerate Boolean satisfiability", in Proc. IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop - DDECS'2002, Brno, Czech Republic, pp. 270-277, Apr. 2002.
- [13] I. Skliarova, Reconfigurable Architectures for Problems of Combinatorial Optimization, Ph.D. Thesis, University of Aveiro, Portugal, Jan. 2004.
- [14] DIMACS benchmarks, <http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html>.