Development tools for problems of combinatorial optimization and their use for the design of embedded computational devices*

I.Skliarova, A.B.Ferrari

iouliia@ua.pt, ferrari@ieeta.pt
Department of Electronics and Telecommunications,
University of Aveiro,
3810 Aveiro, Portugal
Fax: +351 234 370 545

Abstract
It is known that the majority of combinatorial tasks can be formulated on logic (Boolean, ternary or some other) matrices, which further might be used to solve many problems of discrete optimization, such as finding the shortest and longest path in a graph, map coloring, set partitioning, etc. These problems appear in particular in embedded control systems that deal with technological processes (such as transfer line, assembly line, etc.), traffic management and other application areas. The paper presents a formal description of matrix combinatorial problems and suggests dynamically reconfigurable (FPGA-based) parallel computational devices that allow to solve them. Synthesis of these devices can be performed with the aid of software tools developed for PC computers in Visual C++.

Key Words
Discrete Vectors, Logic Matrices, Combinatorial Processor, Reconfigurable Computations, FPGA.

1. INTRODUCTION

Various problems of combinatorial optimization appear in different application areas, such as: synthesis and optimization of digital circuits [1], mapping, placement and routing for different kinds of microchips [2], topology and cartography (map-making) [3], artificial intelligence [4], etc. Examples of such problems can be found in [1] and we will refer to just a few of them such as searching for the shortest and longest path, graph coloring, Boolean functions optimization, covering and planarity problems, encoding problems, etc. Because of the universality and wide scope of applications, the problems of combinatorial optimization are well studied and specified through strong mathematical models that can be applied to different optimization tasks. Mainly the problems of combinatorial optimization can be formulated on such mathematical objects as graphs [1], matrices [4], sets [5], Boolean functions and equations [4]. All these models have many different varieties, such as widely used hypercubes [6], decision diagrams [7] etc., and they are often convertible in such a way that one model can be formally transformed into another. It is important to be able to formulate some universal problems that are common to many models and might be considered as some kind of basis for many tasks of combinatorial optimization [4].
Since most of the combinatorial problems are NP-hard [1,4], they usually require essential time of computations. Note that for certain level of complexity these problems can be solved on only very powerful and very expensive computers and it is reasonable to consider different accelerators based on hardware, such as co-processors. Fortunately, many combinatorial problems are well suited for parallel and pipelined implementations. Many of them can be also solved on special hardware structures that model the problem. Thus we can formulate some approaches that might be used in practice:
1. Exploiting architectures targeted to combinatorial computations. The respective devices are intended to be utilized as a core for combinatorial processors.
2. Exploiting specialized (hardwired) circuits that might be used in order to solve particular combinatorial problems (such as Boolean and ternary matrix based optimization). This approach was considered, for example, in [8].
3. Exploiting special hardware structures that enable us to solve the problem by means of natural simulation. We will say that such structures directly model the problem and allow to find out the decision as a result of hardware modeling. For example, we can emulate graph by a set of connected flip-flops. Any flip-flop represents a node of the graph. Any connection corresponds to the respective arc in the graph. Such a model might be useful when we want to find out the shortest path between some nodes of the graph.
Note that the problems of combinatorial optimization are very important and they have to be solved in numerous tasks of practical applications. There are

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

many powerful algorithms and software tools that enable us to solve this problems on general-purpose computers. Moreover there is even a special language LYaPAS that is aimed at the description of combinatorial algorithms [9]. However the computational complexity of combinatorial algorithms [1,4,9] makes it difficult (and sometimes even impossible) for many application areas to solve problems, that are based on such algorithms, in appropriate time or with available computational resources. The latter is especially important for heterogeneous real-time (reactive) systems, such as embedded controllers. For example, we may want to calculate the shortest path from one point to the other point and affect the behavior of a movable object with the aid of an embedded controller. Assuming that external conditions may be changed during motion we have to provide very fast run-time recalculation. The other domain where the performance of combinatorial computations is very important is CAD systems for different application areas, such as logic synthesis, topological design, etc. Independently of area, the target requirements to essentially increase the performance of combinatorial computations and to provide capabilities to solve combinatorial problems on equipment that currently cannot be used for such purposes because of its limited resources, are very important and timely.

We have already mentioned that many combinatorial problems can be solved on special hardware structures. The paper presents an approach to the design of such hardware on the basis of reconfigurable circuits such as FPGAs. The following two conclusions show an effectiveness of such an approach:

1. Heterogeneity of combinatorial problems does not make sense to construct some kind of universal device, which on the one hand will be rather complex and expensive and on the other hand obviously will not be used in its full power.
2. Reconfigurable hardware has relatively short development time and can be reused. Thus we can consider a device based on such hardware as an accelerator [8] (co-processor) for general-purpose processor with statically (or even dynamically) modifiable functionality.

The paper is organized in 6 sections. Section 1 is this introductory section. Section 2 describes basic operations on discrete vectors, i.e. such vectors whose elements have limited number of possible values. Section 3 shows different approaches to the design of reconfigurable hardware, which realize a given set of operations on vectors. Section 4 presents an architecture for a reconfigurable combinatorial processor, which can solve a number of problems on discrete matrices. Section 5 describes the software tools that have been developed in Visual C++ in order to describe, test and implement in hardware basic components of the combinatorial processor. The conclusion is in section 6.

2. BASIC OPERATIONS ON DISCRETE VECTORS

Most combinatorial problems have a discrete character. That is why they can be specified by such mathematical models as graphs, sets, discrete matrices, etc. We have already mentioned that these models are often convertible in such a way that one model can be formally transformed into another. For example, in [4] it was shown that they might be converted to a universal matrix representation. Each matrix can be considered as a set of discrete vectors, i.e. such vectors whose elements have a limited number of values. In practice two kinds of such vectors are often used: Boolean and ternary. Note that for many practical applications it is reasonable to utilize such vectors whose elements have more than three values. If a vector V is composed of elements with n possible values we will call it V^n. If a matrix M is composed of vectors V^n we will call it M^n.

A particular combinatorial problem specified in matrix form can be described by a discrete algorithm that must be applied to the matrices in order to solve the problem. Matrices themselves are considered to be input operands. The result might also be presented in matrix form, but on the other hand it could be a vector or even a numeric value. Many examples of such algorithms are given in [10]. Since any algorithm is based on operations on vectors V^n we would like to consider these operations in more detail.

Analysis of different combinatorial algorithms has shown that practically all operations on vectors can be seen as either unary or binary. Unary operations have just one operand, they are applied to individual vectors and their examples are the following:

complement all the bits of the vector;

counting the number of 1s in the respective vector;

shifting left/right;

rotating left/right;

swapping the two halves of the vector (for example 10011⇒11001), etc.

Binary operations are applied to pairs of vectors. Some examples are the following:

bitwise logic operations, such as AND, OR, XOR, etc.;

absorption operation (for instance the vector 10011 absorbs the vector 00010 with 1s);

orthogonality;

intersection, etc.

Each operation can be executed on a computational core, which is similar to general-purpose arithmetic and logic unit (ALU). The difference has appeared in a uniqueness of considered operations. However there exist another important issue. Note that on the one hand we can consider practically infinite number of unary and binary operations on vectors, but on the other hand any real task requires just a very limited number of them. Indeed the majority of combinatorial algorithms is based on multiple repetition of similar operations that have to be

applied to different vectors of a matrix either sequentially or in parallel [10]. This defines the importance of dynamic modifications of the computational core. The idea is the following. On a given step of an algorithm the core is configured for the first set S_1 of operations. The number of such operations is usually very limited. Then the set S_1 will be repeated many times (i.e. it will be periodically applied to many vectors included in the respective matrix). After completing the step and jumping to the other step the core will be reconfigured for the new set S_2 of operations. This process will be repeated until the end of the algorithm. Since the number of repeated operations is usually very large, the time of reconfiguration is negligible compared with the execution time.

3. IMPLEMENTATION OF BASIC OPERATIONS

We have implemented a dynamically (run-time) reprogrammable computational core on the basis of statically reconfigurable FPGAs (such as XC4010XL of Xilinx) with RAM-based cells. The total problem has been divided into three sub-problems that will be considered below.

The first sub-problem is the following. For given vectors $A=\{a_1,...,a_m\}$ and $B=\{b_1,...,b_m\}$ we need to calculate the new vector $R=A\#B=\{r_1,...,r_m\}$ (# is some binary operation). Any Boolean operation on m-element vectors V^n_m, such as AND, OR, XOR, can be considered as a set of m similar operations on the respective elements. For n=2 (for Boolean vectors) any operation can be implemented on 4x1 RAM cell (with 2 inputs and 1 output). For n=3 and n=4 (for ternary and four-value vectors) any operation can be implemented on a 16x2 RAM cell (with 4 inputs and 2 output). In any case for each individual operation we will need just one RAM-based cell of an FPGA, such as XC4010XL of Xilinx (see fig. 1).

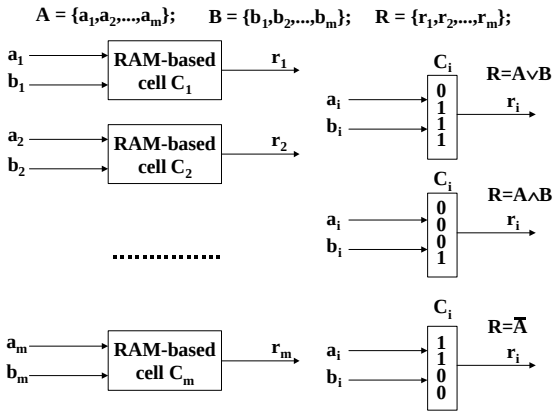

Figure 1. Implementation of operations such as R=A#B

The reprogrammability is achieved through dual-port RAMs (they are available in commercial FPGAs). Hence the first port is used for the normal functionality and the second port allows to reload the RAM contents, i.e. to change the logic function. The same structure can be used for unary operations that are similar for all the elements. Examples of such operations are depicted in fig. 1.

The second sub-problem assumes an answer in form of yes/no. For example for two given vectors A and B we have to check if they are in relationship of orthogonality. In this case the primary operations on the elements are performed on RAM-based cells and outputs of such cells are connected to elementary logic block, which carries out very limited number of elementary logic operations, such as AND and OR which can be selected by the respective code of the operation. In fact the logic block performs the following computation $f(R=A\#B)\in\{0,1\}$, "0" represents "no" and "1" represents "yes", $f\in\{O_1,O_2,...\}$, where $O_1,O_2,...$ - acceptable operations of the logic block (see fig. 2). Note that the number of such operations is very limited and they are indeed trivial. The scheme in fig. 2 permits to realize such operations as checking orthogonality (A_ort_B), absorption (A_abs_B), etc.

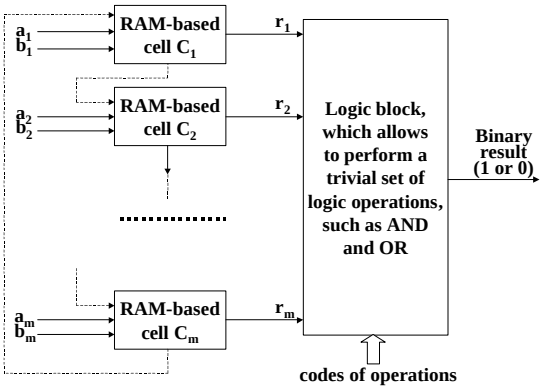

Figure 2. Reconfigurable circuit for testing binary decisions

The last sub-problem produces a result in the form of a binary number, which in general has more than one bit (for example we have to calculate the number of 1s in Boolean vector 10010110, which is equal to 100 (i.e. 4 digits with the value 1) and has 3 bits). One possible approach is the following. The vector is divided in segments of a pre-defined size (see fig. 3). Each segment is analyzed in a RAM-based group of cells G. The results from all the segments go to an elementary adder that calculates the final number. Dynamic reprogrammability is achieved through reloading the RAM-based cells and altering the control sequence. The latter can be done with the aid of a control circuit with dynamically modifiable functionality [11]. The other approach is based on one-bit (elementary) shifting (routing) and counting of what we need.

Let us consider the scheme in fig. 3 in a bit more detail. It is based on the following primary operations:

1) Shifting the group G up or down, which allows to connect inputs of G to any outputs of the registers that keep operands A and B.

2) Realizing in G any desired set of Boolean functions with the aid of run-time reprogramming technique. In fact a similar approach might be applied to vectors with n>2. In this case any value of any element has to be coded and G will operate with coded inputs.
3) Calculating a multibit binary number with the aid of a simple adder. The adder has an accumulator on its outputs. Thus it allows to execute such operations as counting and incremental addition. The actions (algorithms) of operations are pre-defined.
4) Any complex operation on vectors can be described algorithmically by a microprogram specifying the desired sequence of elementary operations mentioned in the previous three points. We have considered a device supporting the change of these microprograms without any altering of its hardware resources. It might be done if we construct the microprogramming control unit as a RAM-based finite state machine (FSM) with dynamically modifiable functionality. For these purposes we have used the technique described in [11].

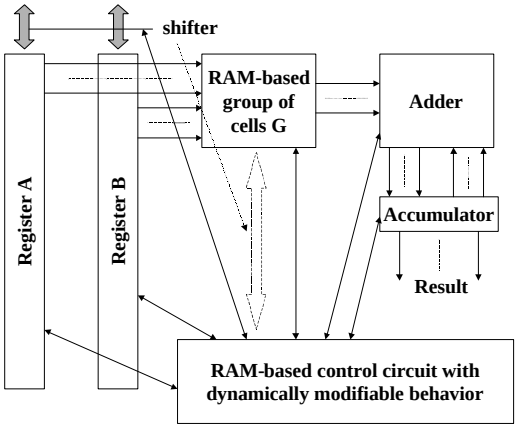

Figure 3. An implementation based on microprogramming control unit with dynamically modifiable functionality

4. COMBINATORIAL PROCESSOR

The computational unit for the problem, which we are going to solve, can be specified in the following way:

1. The most general task for this unit is calculations on a single (unary) or a pair (binary) of logic matrices. Each matrix is composed of m vectors V^n_m with n-bit primary elements. The vectors can be viewed either as a set of rows or as a set of columns. In fact we should be able to convert one form into the other (see point 3 below).

2. Most computations are homogeneous, i.e. they invoke the same operation(s) for regular data. Any complex operation on matrices requires a very large number of similar elementary operations.

3. We have to be able to perform the required operations either on rows or on columns of logic matrices.

4. The volume of data representing matrices is usually very large.

Taking into consideration all these requirements we have proposed the architecture of a combinatorial processor depicted in fig. 4 that is going to be used for future analysis and estimation.

The processor contains 3 very fast blocks that are based on RAM implemented in CLBs of an FPGA. The size of RAM-based blocks is modifiable statically. They are used in order to store 3 matrices Z, X and Y, where the matrices X and Y are considered to be operands and Z keeps the result of combinatorial computations. The matrices X and Y are loadable from outside and the matrix Z is readable from outside. This can be implemented on the base of dual-port RAM, such that are used in FPGA of XC4000XL family. On the other hand the value of Z might be utilized as an operand (either X or Y) in future computations.

Each matrix (such as Z, X and Y) is composed of discrete vectors (such as V^n_m). Any complex operation on matrices (such as finding minors [10]) can be described algorithmically in the form of a microprogram. The processor has to allow dynamic modifications of the implemented microprograms.

The RFU and RCU depicted in fig. 4 can be re-programmed during run-time. An implementation of the RFU was considered in the previous section and it allows to run unary and binary operations on discrete vectors. The RCU permits to change algorithms that have to be performed on matrices. Dynamic modifiability of RCU is achieved with the aid of the technique described in [11].

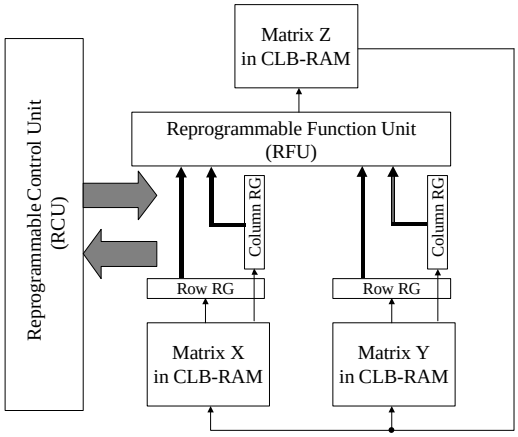

Figure 4. Primary architecture of a combinatorial processor

5. DEVELOPMENT TOOLS

We are developing software tools to be incorporated in a system for the design of a combinatorial processor that can solve different problems on discrete matrices. They are based on previously developed tools [12,13] and have been designed for PC with the aid of Visual C++ environment and MFC

library. Fig. 5 depicts the basic components of the system, which are the following:

1) Library;
2) Schematic editor (SE);
3) Compiler;
4) Block that provides support for dynamic reconfiguration;
5) Control Algorithm Editor (CAE);
6) Simulator;
7) Synthesizer.

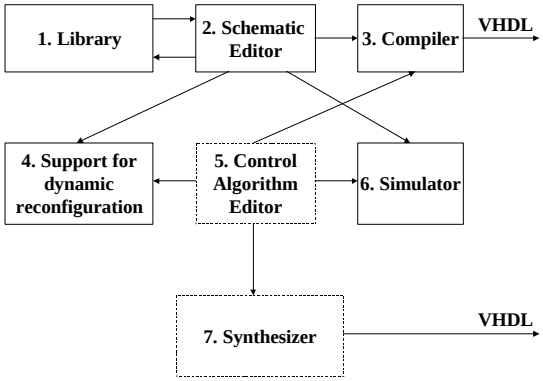

Figure 5. Basic components of the system

The first three blocks have already been developed and tested. The simulator is under development. The fifth block has been borrowed from [14]. The last block is currently used from Xilinx Foundation Software. In future we are going to replace this block with a new one, which will implement methods presented in [11]. These methods allow to synthesize control units on the base of RAM blocks, which implement the primary functionality. Since the basic model of a control circuit is a FSM, the RAM blocks enable to realize all the state transitions and to generate the required output signals. Note that commercially available FPGAs, such as Xilinx XC4010XL, contain built-in RAM-cells with different structural organization, such as 32x1 and 16x2. They might be programmed in such a way that memory can be addresses through dual port. Reloading RAM makes it possible to change the functionality of the respective FSM. It can be done either statically or dynamically, i.e. during run-time. The latter is essentially simplified with the use of dual-port address mode. Thus the first port provides the primary functionality and the second port enables to reload the RAM (previously suspended) in order to change this functionality. Finally it allows to construct control circuits with dynamically reloadable microprograms.

The Schematic Editor (SE) allows to construct a combinatorial processor and its reconfigurable components. This might be done with the aid of a pre-defined template that can be customized for a particular application. The primary user interface of this block is shown in fig. 6. Future extensions of this software will be devoted to increasing the level of the specification. In order to do this we are going to consider a higher level of abstraction for representing the basic ideas of the design using tools such as the Unified Modeling Language (UML) [15]. Reconfigurable combinatorial processor can be synthesized on the basis of library components as the ones considered in section 3. The resulting scheme can be saved for future needs. Block 3 permits to compile the graphical specification of the scheme into behavioral VHDL code that can be used as an entrance to Xilinx Foundation Series Software. The simulator will allow to verify the design. The Control Algorithm Editor (CAE) makes possible to specify the control algorithm for the RCU (see fig. 4) in the form of hierarchical graph-schemes [14] and their varieties. Software tools [14] transform this graphical description to VHDL code, which can be compiled by the Xilinx synthesis tool. Block 4 obtains all the required data for dynamic configurations from SE and CAE. This is used in order to reload during run-time the RAM-based cells affecting the functionality of the combinatorial processor.

The first three blocks of the developed software have been tested. We considered some combinatorial tasks such as covering problem for Boolean matrices. The combinatorial device that might be used for such purposes has been designed with the aid of the developed software tools and Xilinx Foundation Series Software. The device has been physically implemented on an XStend development board with Xilinx FPGA XC4010XL and tested. The results of experiments have proved that the considered approach is correct and can be used for practical applications.

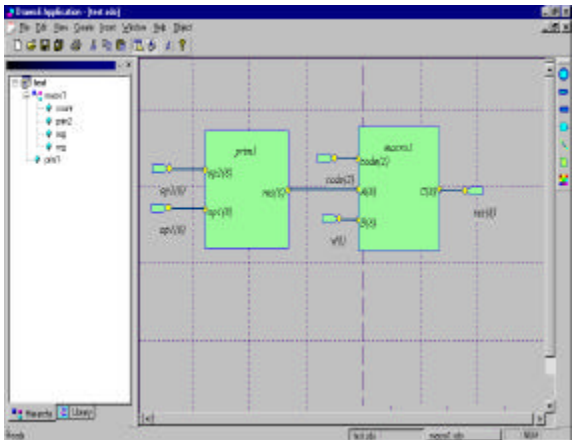

Figure 6. Primary user-interface of SE

6. CONCLUSION

We have described an approach to the design of a combinatorial processor with a dynamically reconfigurable core on the basis of statically configurable FPGAs such as Xilinx XC4010XL. The dynamic modifications required to implement discrete operations were achieved through run-time reprogramming of RAM-based FPGA cells. Dynamic changes of algorithms on discrete matrices have been provided with the aid of a microprogrammed control

unit with dynamically modifiable behavior. It was shown that such unit can be synthesized as RAM-based FSM. Finally we developed basic components of the system, which is intended to be utilized as a design environment for combinatorial computational units. These might be used as co-processors for general-purpose processors. The system has been used for the design, implementation and test of a device, which allows to find out minimal covering for Boolean matrices.

REFERENCES

1. Giovanni De Micheli, "Synthesis and optimization of digital Circuits", McGraw-Hill, Inc., 1994, 570 p.
2. Nozomu Togawa, Masao Yanagisawa, Tatsuo Ohtsuki, "Maple-opt: A Performance-Oriented Simultaneous Technology Mapping, Placement, and Global Routing Algorithm for FPGA's", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, N. 9, September 1998, pp. 803-818.
3. T.Tambouratzis, "A Consensus-Function Artificial Neural Network for Map-Coloring", IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics, Vol. 28, N. 5, October 1998, pp. 721-728.
4. A.Zakrevskij, "Combinatorial Problems over Logical Matrices in Logic Design and Artificial Intelligence", Electrónica e Telecomunicações, vol. 2, no. 2, 1998, pp. 261-268.
5. S.Baranov, "Logic Synthesis for Control Automata", Kluwer Academic Publishers, 1994.
6. A.T.M.Shafiqul Khalid, Mohammad S.Alam, A.A.S.Awwal, "KH-map: A new way of representing the hypercube structure", Journal of Systems Architecture, Vol. 44, N. 11, August 1998, pp. 873-885.
7. Zeljko Zilic, Zvonko G.Vranesic, "Using Decision Diagrams to Design ULMs for FPGAs", IEEE Transactions on Computers, Vol. 47, N. 9, September 1998, pp. 971-982.
8. Marco Platzner, "Reconfigurable Accelerators for Combinatorial Problems", Computer, April 2000, pp. 58-60.
9. M.Gavrilov, A.Zakrevskij, "LYaPAS: A programming language for logic and coding algorithms", Academic Press, New York, London, 1969.
10. A.Zakrevskij, "Logical Synthesis of Cascade Schemes", Moscow, Science, 1981, 414 p.
11. V.Sklyarov, "Synthesis and Implementation of RAM-based FSM", Proceeding of FPL'2000, Villach, Austria, 2000.
12. I.Skliarova, A.B.Ferrari, “An Architect's Workbench for Reconfigurable Computing”, SBCCI99 – XII Symposium on Integrated Circuits and Systems Design, Natal, Brazil, October 1999, pp. 154-157.
13. I.Skliarova, A.Ferrari, "Projecto e Implementação de um subconjunto da arquitectura MIPS16 com base em FPGA XC4010XL ", Electrónica e Telecomunicações, v.2, Nº6, September 1999, pp. 724-732.
14. Arnaldo Oliveira, Andreia Melo, Valery Sklyarov, "Specification, Implementation and Testing of HFSMs in Dynamically Reconfigurable FPGAs", Proceedings of the 9th International Workshop FPL'99, Glasgow, UK, 1999, pp. 314-322.
15. Grady Booch, James Rumbaugh, Ivar Jacobson, “The Unified Modeling Language User Guide”, Addison-Wesley, 1999, 470 p.