

# Architecture of a Reconfigurable Processor for Implementing Search Algorithms over Discrete Matrices

*Valery Sklyarov*

University of Aveiro, Department of  
Electronics and Telecommunications, IEETA  
3810-193 Aveiro, Portugal

*Iouliia Skliarova*

University of Aveiro, Department of  
Electronics and Telecommunications, IEETA  
3810-193 Aveiro, Portugal

**Abstract.** *The paper suggests architecture of a reconfigurable processor, which can be customized for implementing different search algorithms over discrete matrices. Such algorithms might be used for solving various problems of combinatorial optimization, such as covering, Boolean satisfiability, etc. The proposed architecture contains memory blocks for a binary or a ternary matrix, general-purpose registers, five stacks, that make possible to carry out recursive search procedures based on a decision tree, and a reprogrammable functional unit that allows to perform the required operations over binary and ternary vectors. Two levels of control circuits have been suggested. The first (top) level permits to realize the search algorithm. The second (bottom) level allows operations that are required for the algorithm to be implemented.*

**Keywords** – problems of combinatorial optimization, reconfigurable architecture, search algorithm, FPGA, Boolean and ternary matrices

## 1. Introduction

There are many practical applications, which require the solution of some combinatorial problems, such as Boolean satisfiability [1], covering of binary (Boolean) matrices [2,3], graph coloring [2,4], etc. It is known that the majority of these problems are NP-hard and as a result they are time and resource consuming. Because of that it is very important to design and to implement the respective accelerators such as coprocessors for general-purpose computers [3,5,6].

Combinatorial computations have two distinctive features [7]. Firstly, as a rule they require considering a huge number of different variants (feasible solutions). Secondly, these variants can be ordered and examined with the aid of a decision tree that provides an efficient way for handling intermediate solutions. The decision tree is constructed throughout the search process and it is traversed starting from the root. During the search special reduction methods are applied allowing both to simplify intermediate situations and to reduce the number of possible variants to analyze.

Most combinatorial problems have discrete character. That is why they can be formulated on such mathematical models as graphs, sets, Boolean functions, discrete matrices, etc. These models are convertible, i.e. one model can be formally transformed into another. For example, in [2] it was shown that they might be converted to a universal matrix representation. Logic matrices are very well suited for processing them in hardware (in FPGAs in particular [7,8]). This property has played an important role for the use of matrices as a basic mathematical model for architecture proposed in this paper. In [9] it is presented 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 operations that are unique and not directly supported by general-purpose processors. On the other hand very

often various combinatorial problems involve different sub-sets of such operations. It allows to conclude the following 1) reconfigurable devices should be more profitable than ASICs and general-purpose systems because the same device might be *efficiently* employed for solving *different problems*; 2) the time of reconfiguration should be negligible comparing with the time of computations.

## 2. An example of combinatorial search

Fig. 1 depicts the basic algorithm of combinatorial search, which can be used to solve combinatorial problems over Boolean and ternary matrices.

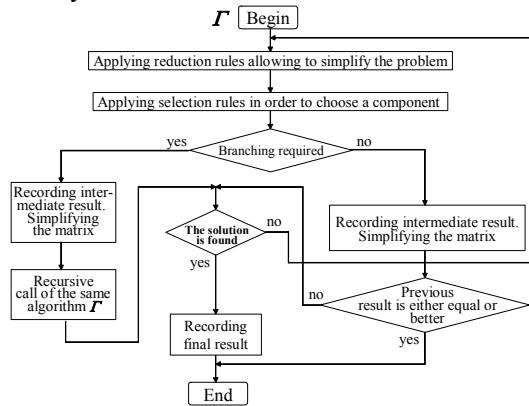


Figure 1. Basic search algorithm

Different steps of the algorithm will be demonstrated on an example of the exact method [2] that permits to find out a minimal column cover of a Boolean matrix. Suppose that it is given the following matrix:

$$\begin{array}{cccccccccccc}
 & a & b & c & d & e & f & g & h & i & j & k & l \\
 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
 2 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
 3 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
 4 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
 5 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
 6 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
 7 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
 8 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 9 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
 \end{array} \quad (1)$$

Columns  $c$  and  $g$  represent a minimal column cover, i.e. a minimal subset of columns that

have at least one value “1” in each row of the matrix. The following set of rules that permit to simplify the matrix will be used:

- If for  $i \neq j$ ,  $row_i \& row_j = row_j$  then the  $row_i$  can be removed from the matrix, for example,  $row_7 \& row_8 = row_7$  and  $row_8$  has to be removed from the matrix;
- If for  $i \neq j$ ,  $column_i \& column_j = column_i$  then the  $column_j$  can be removed from the matrix, for example,  $column_d \& column_c = column_d$  and  $column_c$  has to be removed from the matrix.
- If there is a row, which does not have values “1” then covering cannot be found.

The first two operations are called subsumption operations. For the selection purposes the following rules will be used:

- If a row has just one value “1” then the respective column (i.e. the column that has this “1”) must be included into the covering;
- 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.

Fig. 2 shows all the steps that are required in order to find out the minimal column cover of the matrix (1).

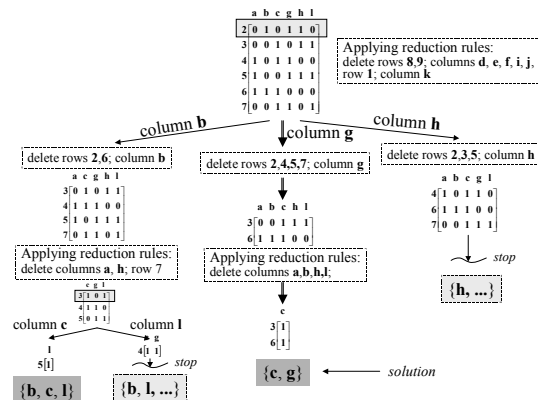


Figure 2. Using search algorithm to find out the minimal column cover of the matrix (1)

The way that leads to the minimal cover  $\{c, g\}$  is shown with the aid of double arrows. There are two branching points in fig. 2:  $b-g-h$  and  $c-l$ . After getting the first solution  $\{b, c, l\}$

we are interested just in coverings with 2 or less columns. Thus, it is not necessary to traverse all branches and forward propagation in the search process can be stopped at any point that gives a 2-component incomplete solution.

Similar search algorithms can be used for solving many combinatorial problems, such as Boolean satisfiability, graph coloring, etc. They have several common features:

1. They are recursive and as a result a recursive control circuit 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 required operations are usually not the same for different combinatorial problems (for example we can compare the considered above subset and the subset of operations that are employed for the Boolean satisfiability problem [7]).

5. In order to perform forward and backtrack propagation we can use a stack memory that stores and restores intermediate results (such as 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 very similar for different algorithms. 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. It shows in particular a necessity of reconfiguration.

All these features have been taken into account in the proposed architecture of a combinatorial processor targeted to the considered type of search algorithms.

### 3. Architecture of a reconfigurable processor

Fig. 3 depicts the proposed architecture of a reconfigurable processor. The processor is composed of five following primary units.

**3.1. Storage for a ternary (Boolean) matrix.** Any Boolean matrix is kept in two memory

(RAM) blocks. The first block addresses the columns of the matrix and the second block addresses the rows of the matrix (i.e. an initial matrix and its transpose are stored in two separate memories). It allows any row or column to be read in one clock cycle. An address is provided by the respective counter, which can also be loaded in parallel (i.e. any arbitrary address might be specified). Any ternary matrix is coded by two Boolean matrices. The first matrix contains values "1" in all positions where the initial ternary matrix has values "1" and values "0" in all other positions. The second matrix contains values "1" in all positions where the initial ternary matrix has values "0", and values "0" in all other positions. Thus zeros, ones and don't care are coded respectively by 01, 10 and 00. Note that for the considered algorithms the combination 11 is not needed, but if necessary it can also be used.

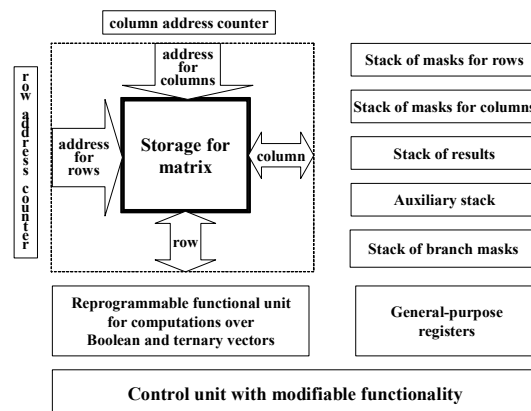


Figure 3. Architecture of a combinatorial processor for implementing search algorithms over Boolean and ternary matrices

**3.2.** Five different stacks, which permit to store intermediate results at any branching point in order to provide backtracking if required. The stacks are employed for storing the following data:

- masks for rows of the matrix;
- masks for columns of the matrix;
- intermediate results in branching points (i.e. names of some columns that have already been included into the covering);
- masks for branches that have already been considered at any branching point;
- values of estimation criteria, which permit the proper component of the matrix to be





ones). The first column from left to right that has "1" in the row 3 is chosen. This is the column  $c$ . Since this is a branching point that might require to return back to evaluate another branch ( $l$  in our case) it is necessary to save in the stacks all data that are needed to restore this branching point in future (see fig. 4). The selected row 3 is deleted from the matrix.

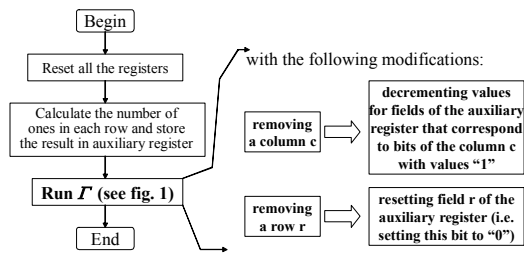


Figure 6. The basic operations of top-level control algorithm

Step 8: The row 4 (because it has the value "1" in the column  $c$ ) and the column  $c$  are removed from the matrix and this requires changes shown in fig. 4.

Step 9: According to the reduction rules (see section 2) the column  $g$  is removed from the matrix.

Step 10: The row 5 contains just one value "1" in the column  $l$ . Thus the column  $l$  has to be selected leading to the first covering  $\{b, c, l\}$  which is stored in a general-purpose register. The number 3 is also saved in a general-purpose register in order to discard in future all branches that lead to solutions with more than 2 columns. It allows the search space to be reduced significantly.

Step 11: The last branching point ( $c-l$ ) is restored from the stacks (see fig. 5). For keeping branch masks and intermediate results we can change the stack registers directly (see bottom-left and bottom-right stacks in fig. 5). The register of the stack of branch masks allows to conclude that the current branch is the last in the considered branching point. For the other stacks (i.e. stack of masks for rows, stack of masks for columns and auxiliary stack) the pop operation is executed, which permits to restore values of the respective registers (i.e. the rows mask register, the columns mask register, and the auxiliary register depicted in fig. 5) for the last branching point (i.e. the point  $c-l$ ). After that the considered above steps for the

algorithm  $F$  will be repeated. Finally it permits to get an exact solution of the covering problem. For our example the first solution has a complexity 3 and is actually not the best. As it is illustrated in fig. 2 there exist two solutions of complexity 2.

## 4. Implementation details

The effectiveness of the proposed technique and architecture that permits to implement this technique was evaluated on software models and in hardware (in an FPGA in particular). Two problems that are the covering and the Boolean satisfiability were solved in software (C++ language) that describes architecture in fig. 3. For the covering problem we used the considered above search algorithm. The time of execution was evaluated by counting the required clock cycles taking into account the clock frequency that can be used in simulated hardware implementations (this frequency was set to 100 MHz). The results of experiments have shown that the proposed architecture can be used for solving different combinatorial problems with the aid of search algorithms (see their general structure in fig. 1). The proposed in fig. 6 modifications slightly increase the complexity of the search algorithms, but on the other hand they enable us to reduce the time required for calculations at selection steps (see section 2). Recursive algorithms can easily be implemented in VHDL on the basis of a hierarchical FSM (HFSM) model [12]. However we have found some difficulties for implementation of HFSM on the basis of system-level specification languages, such as Handel-C.

The considered exact algorithm (see sections 2, 3) that solves the covering problem was entirely described in Handel-C and implemented in FPGA XC2S200 of Spartan-II family available on RC100 board of Celoxica [11]. This work was done in 2003 by two students (Pedro Almeida and Manuel Almeida) of Aveiro University within their final year project. Experiments with this circuit have shown an essential speed up (at least 3 times) comparing with a software implementation. This work will be continued within the project that is currently carried out for FPGA XC2VP7 (Xilinx Virtex-II Pro family) available on PCI

prototyping board ADM-XPL of Alpha Data [13].

A number of circuits for individual components have been implemented in hardware. They include a parameterizable reprogrammable FSM [12] (that is used for implementing the bottom and the top levels of control) and the reprogrammable functional unit. All these experiments were performed with TE-XC2Se board of Trenz electronic [14] (FPGA XC2S300E of Spartan IIE family of Xilinx), RC100 board of Celoxica [11] (FPGA XC2S200 of Spartan-II family of Xilinx) and ADM-XRC PCI board of Alpha Data [13] (FPGA XCV812E of Virtex-EM family of Xilinx). For the design of FPGA-based circuits we used the integrated software environment of Xilinx ISE 5.2 [15] (VHDL-based flow) and the DK1 design suite of Celoxica [11] that supports the design of digital circuits with the aid of Handel-C.

## 5. Conclusion

The paper suggests architecture of an application-specific reprogrammable processor, targeted to search algorithms over discrete matrices. It is shown that these matrices can be efficiently used for specifying various combinatorial problems, such as the Boolean satisfiability, covering, etc. One of these problems (namely the covering) was examined in detail, i.e. all elementary steps of this task executed in hardware for getting the result were described. The considered algorithms enable us to find out exact solutions of combinatorial problems and they have many common features. The latter makes possible to suggest a number of reusable components for the processor, such as storage for matrices that allows an independent access to matrix rows/columns, stacks that keep intermediate results in branching points, two levels of control circuits, etc. Note that these components and the relevant operations are not supported or, at least, cannot be efficiently implemented, in general-purpose processors, which proves in particular a necessity of the proposed specialized device. The suggested architecture was modeled in software (C++ language) and implemented in FPGA-based circuits. The preliminary results of experiments have shown

that it could provide a significant speed up comparing with pure software implementations.

## Acknowledgements

This work was supported by the Portuguese Foundation of Science and Technology under grants POSI/43140/CHS/2001 and FCT-PRAXIS XXI/BD/21353/99.

## References

- [1] Gu, J.: Satisfiability Problems in VLSI Engineering. DIMACS Workshop on Satisfiability Problem, Mar. 1996.
- [2] Zakrevski, A.D.: Logical Synthesis of Cascade Networks. Moscow: Science, 1981.
- [3] Plessl, C., Platzner, M.: Instance-Specific Accelerators for Minimum Covering, in: Proc. 1<sup>st</sup> Int. Conf. ERSA'2001, Las Vegas, pp. 85-91.
- [4] Murgai, R., Brayton, R., Sangiovanni-Vincentelli, A.: Logic Synthesis for Field Programmable Gate Arrays. Kluwer Academic Publisher, 1995.
- [5] Platzner, M.: Reconfigurable accelerators for combinatorial problems. IEEE Computer, Apr. 2000, pp. 58-60.
- [6] Abramovici, M., de Sousa, J.,T.: A SAT solver using reconfigurable hardware and virtual logic. Journal of Automated Reasoning, vol. 24, nos. 1-2, February 2000, pp. 5-36.
- [7] Sklyarov, V., Skliarova, I., Ferrari, A.B.: Hierarchical Specification and Implementation of Combinatorial Algorithms Based on RHS Model. Proc. XVI Conf. on Design of Circuits and Integrated Systems - DCIS, 2001, pp. 486-491.
- [8] Skliarova, I., Ferrari, A.B.: Design and Implementation of Reconfigurable Processor for Problems of Combinatorial Computations. Proc. of EUROMICRO Symp. on Digital Systems Design, Warsaw, Sept. 4-6, 2001, pp. 112-119.
- [9] Zakrevskij, A.: Combinatorial Theory of Logical Design. Automatics and Computers. 1990, N 2, pp. 68-79.
- [10] V.Sklyarov, Reconfigurable models of finite state machines and their implementation in FPGAs. Journal of Systems Architecture, 2002, 47, pp. 1043-1064.
- [11] DK1, Handel-C: <http://www.celoxica.com>.
- [12] Sklyarov, V., Skliarova, I. Design of Digital Circuits on the Basis of Hardware Templates. Proceedings of ESA'2003, Las Vegas, 23-26 June, 2003.
- [13] PCI boards: <http://www.alpha-data.com>
- [14] Spartan IIE Development Platform, 2002. Available: [www.trenz-electronic.de](http://www.trenz-electronic.de)
- [15] ISE 5.2, Xilinx FPGA. Available: <http://www.xilinx.com/>