Design and Implementation of Communicating Fixed and Variable Instruction Set Processors

Valery Sklyarov
DETIUA/IEETA
University of Aveiro
Aveiro, Portugal
e-mail: skl@ua.pt

Iouliia Skliarova
DETIUA/IEETA
University of Aveiro
Aveiro, Portugal
e-mail: iouliia@ua.pt

João F.Lima
IEETA
University of Aveiro
Aveiro, Portugal
e-mail: jflima@ua.pt

***Abstract*— the paper describes a computational system that is composed of a special-purpose processor augmented by an application-targeted coprocessor with variable instruction set. The primary objective is to form the processor architecture in such a way that is the most appropriate to a selected scope of applications and to optimize instructions of the coprocessor for a particular application. As an example the scope of combinatorial search algorithms was examined and experiments were carried out and analyzed with the relevant system implemented in FPGAs.**

Keywords-computational system; combinatorial search algorithms; FPGA

I. INTRODUCTION

Nowadays a special attention is paid to problem-oriented computational systems with such architectures that are the most appropriate to particular applications. A number of distributed solutions have been explored enabling the required computations to be spread among multiple processing elements connected to a net. As examples, a recently proposed paradigm of Networks-On-Chip [1] and application-specific instruction-set processors [2] can be pointed out. The latter combine general purpose processor cores and programmable fabric showing such advantages as [2]:

- Shortened development lead time and cost by reusing components of a pre-verified processor;
- Supplying optimized instructions;
- Achieving certain flexibility through reconfiguration.

It is known that the programmable fabric might be used differently and one of potential ways is an optimization of micro-programs customizing them to the requirements of a particular application. Such variable-instruction set accelerators are very promising for numerous areas [3].

The paper is dedicated to communicating fixed instruction set scope-oriented processor and an application-targeted variable instruction set co-processor with the objective to benefit from advantages [2] listed above. The main distinctive feature of the proposed technique is the rational combination of *scope-oriented* and *application-targeted instructions*.

Let us consider an example. Many practical problems can be solved through applying various algorithms of combinatorial search. Systems for solving combinatorial search problems might be implemented in field-programmable devices (FPGA, in particular). The latter have a number of advantages, which have appeared because the considered tasks possess the following specific features. Firstly, *any task involves a huge number of similar operations*. As a rule, *these operations are not the same for different combinatorial problems*. Thus, it is not easy to construct a universal combinatorial processor, i.e. *processor's instructions have to be customized for a particular problem that is going to be solved*. Secondly, different practical applications might require solving combinatorial tasks with varying complexity. However, optimal results can be achieved in case if the size of processor operands permits any required operation to be performed in one clock cycle. Thus, *the size of operands has to be properly adjusted*.

This paper suggests to customize the set of instructions through implementing variable instruction sets and to adjust the size of operands through generic statements in hardware description language used in the design flow. The following two most important distinctive features of the proposed technique can be highlighted:

- Decomposition of instructions for problems that are going to be solved into problem independent and problem dependent groups, implemented respectively in fixed instruction set processor and a variable instruction set co-processor;
- Wireless remote change of the processor's program and variation of the co-processor's instructions aimed at optimization of the designed system for a particular problem.

Note that the primary objective of this paper is to evaluate the proposed technique and to provide a case study on the basis of pre-selected examples, namely the exact algorithms for Boolean satisfiability and matrix covering.

The remainder of this paper is organized in eight sections. Section II presents the proposed general architecture of a computational system that is composed of a fixed instruction set processor and a variable instruction set co-processor. Sections III and IV describe the processor and the co-processor, respectively. Section V explains program executions. Section VI outlines particularities of a wireless interaction and remote programming. Section VII gives implementation details and demonstrates the results of experiments. The conclusion is presented in Section VIII.

II. System Architecture

At the top level, the considered system is composed of two communicating components (see Figure 1) that are a special-purpose processor and an application-specific coprocessor. The processor is entirely reused and includes components [4] augmented by a scope-oriented extensions. For the simplicity, let us assume that the bus structure and the blocks of Figure 1, marked with filled triangle at the bottom-right corner, are exactly the same as in [4]. Optimization of the system functionality for particular applications is achieved through the use of the following two suggestions:

- Extending the processor functionality through the use of scope-specific components that enable the system to take into account the particularity of the targeted group of applications. It is important that on the one hand such components are application-specific and on the other hand they are common for applications from relatively wide scope and, thus, can be implemented permanently in the fixed part of the processor;
- Variation of instructions that can be somehow optimized for particular applications from the selected scope.

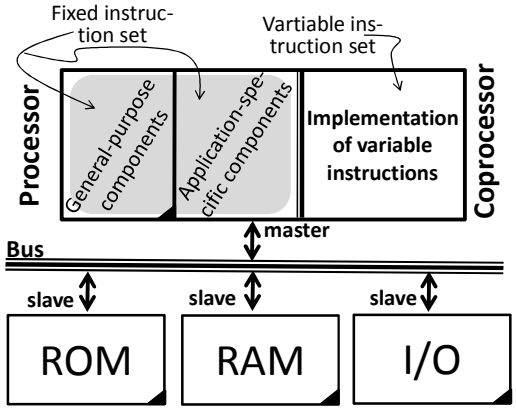

Figure 1. System architecture.

For example, solving combinatorial search problems over discrete matrices considered in [5] would require some dedicated blocks that are common to different problems and, therefore, can be implemented permanently in the fixed part of the processor. Any concrete combinatorial search problem involves dedicated operations that are not the same for different problems. Even a little reduction of the execution time for such operations speeds up the relevant algorithms dramatically due to a huge number of repetitions [5].

Since the proposed system possesses the coprocessor and scope-oriented circuits, the subsequent sections need to be more concrete. Therefore we will discuss the proposed technique on an example of backtracking combinatorial search algorithms [5].

III. Processor Architecture

Application-specific components of the processor targeted to combinatorial search problems (see Figure 2) were selected on the basis of analysis of the relevant algorithms [5] and include:

- Memory, permitting to store discrete matrices and to provide direct access to both: their rows and columns;
- Mask registers allowing the same storage to be used for processing the initial matrix and all eventual intermediate sub-matrices (minors);
- Stacks for managing forward and backward propagation steps in such a way that permits to construct sub-matrices sequentially and to return back to any intermediate sub-matrix if required;
- Application-specific registers.

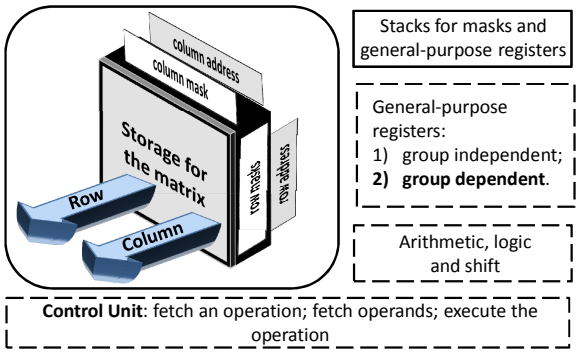

Figure 2. Processor architecture.

The considered architecture combines general-purpose and application specific components. The latter are entirely included in the rectangles surrounded by a solid line and integrated with general purpose components in the rectangles surrounded by a dashed line.

To make the considered above blocks reusable they have to be parametric and customizable (reconfigurable). The first property allows for scalability in such a way that the blocks can be applicable to matrices of different dimensions (i.e. with different number of rows and columns). It is achieved through generic statements in hardware description language used in the design flow. The second property enables different operations over vectors (rows and columns of the matrixes) to be chosen. It is done through the following four ways:

- Reloading a new program to the RAM of the processor (see Figure 1) that is composed of both fixed and variable instructions;

- Optimizing fixed instructions for the selected group of combinatorial search problems;
- Optimizing the set of variable instructions;
- Updating a virtual table that associates the names of variable instructions with micro-programs invoked to execute the instructions.

IV. Coprocessor Architecture

The developed variable instruction set coprocessor is composed of control and execution units. The execution unit is constructed in such a way that permits to implement unique instructions for combinatorial search problems that are intended to be solved. Indeed, according to [5], all necessary operations are executed over Boolean and ternary vectors and they can be divided in the following two groups:

- Executing operations over individual elements of one or two vectors, for example, sequential counting the number of '1's, finding the maximum number of successive elements with the same value, etc.;
- Executing operations over entire vectors, for example, parallel operations over all elements of one/two vectors, verification if two vectors are orthogonal, etc.

Dependently on target requirements (such as performance) the same operation can be run either sequentially or in parallel. For example, counting the number of '1's in a binary vector can be implemented through trivial shift and count or in parallel using cascaded set of adders. In the first case the operation over two vectors would require at least N clock cycles (N is the number of elements in each vector). In the last case the operation would be executed just during one clock cycle. The following assumption is used. Since all operations over vectors can be executed sequentially, we can apply the technique "shift and compute". The shift operation is needed to select an element(s) in vector(s). The second operation is needed to execute some computations over two binary or ternary elements in order to produce the results required for combinatorial search algorithms. For some algorithms the "shift" operation can be replaced with a "select" operation enabling the required computations to be applied just for certain pre-selected elements of vector(s). Obviously the considered sequential operations are time consuming but they also have a number of advantages namely:

- They can be executed in a reusable finite state machine (FSM) with dynamically changeable functionality;
- They consume very little resources;
- They are universal in a sense that any imaginable operation over vectors can be executed;
- They can be considered as a basis for future improvements involving parallel computations.

Figure 3 shows how some sequential operations can be replaced with parallel operations and vice versa. Additional details about implementation of various operations over Boolean and ternary vectors can be found in [6].

Since all potentially required operations for the explored combinatorial search algorithms can be implemented sequentially (see examples in Figures 3a and 3c) we can conclude that any of such algorithms can be realized within the same relatively simple computational system. Since the majority of the operations can be accelerated in a way shown in Figures 3b and 3d, the efficiency of implementation and the relevant performance might be increased significantly. The sequential implementation is very helpful because it allows to estimate influence of any instruction on the performance and to conclude which instructions need to be accelerated.

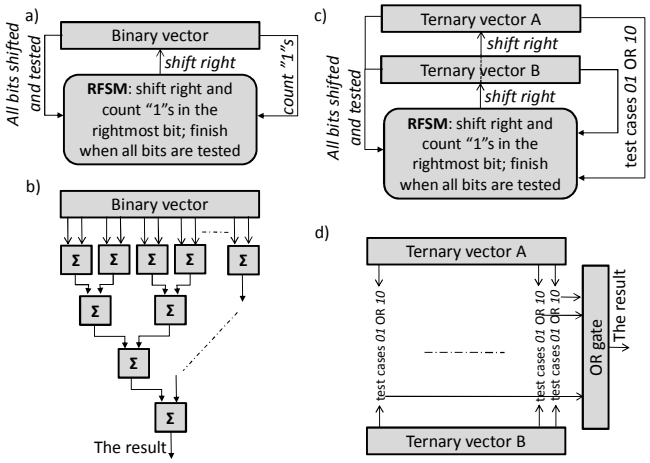

Figure 3. Sequential counting the number of "1"s in a binary vector (a); parallel (combinatorial) counting the number of "1"s in a binary vector (b); sequential verification of vectors' orthogonality (c); parallel (combinatorial) verification of vectors' orthogonality (d).

The control unit is implemented on the basis of FSM with dynamically changeable functionality. Specification of variable instructions is provided with the aid of hierarchical descriptions. All necessary details will be explained through an example. Suppose we would like to explore two different combinatorial search problems. The first problem requires instructions $z^1_1,\ldots,z^1_{q1}$ and the second problem requires instructions $z^2_1,\ldots,z^2_{q2}$; $\{z^1_1,\ldots,z^1_{q1}\} \neq \{z^2_1,\ldots,z^2_{q2}\}$; $\{z^1_1,\ldots,z^1_{q1}\} \cap \{z^2_1,\ldots,z^2_{q2}\} \neq \varnothing$. Let us describe execution steps for all necessary instructions $z_1,\ldots,z_p \subseteq \{z^1_1,\ldots,z^1_{q1}\} \cup \{z^2_1,\ldots,z^2_{q2}\}$ by micro-programs $\mu_1,\ldots, \mu_p$. Any micro-program is executed sequentially with minimal potential numbers of clock cycles satisfying the requirements "the fewer the better". Suppose that each micro-program $\mu_i \in \{\mu_1,\ldots, \mu_p\}$ is described by a hierarchical graph scheme (HGS) Γ_i. Thus, the set of HGS $\Gamma_1,\ldots, \Gamma_p$ (representing micro-programs) enables all the required operations for two considered problems to be described. It is known that a set of HGS is synthesizable and can be implemented in a hierarchical finite state machine (HFSM). Suppose, we would like to minimize hardware resources and to implement the instructions $z^1_1,\ldots,z^1_{q1}$ (or the relevant HGS $\Gamma^1_1,\ldots, \Gamma^1_{q1}$) for the first problem and the instructions $z^2_1,\ldots,z^2_{q2}$ (or the relevant HGS $\Gamma^2_1,\ldots, \Gamma^2_{q2}$) for the second problem. It has been done using the following proposed technique:

- The HFSM has been organized in such a way that

allows the set $\{\mu^i_1,\ldots, \mu^i_p\}$ for the problem i with the maximum required number of instructions to be implemented and an HGS with the maximum complexity (from all HGS $\Gamma^1_1,\ldots, \Gamma^1_{q1}, \Gamma^2_1,\ldots, \Gamma^2_{q2},\ldots$) to be synthesized. The paper [7] describes all necessary details and the methods that permit to satisfy such requirements;

- The HFSM is constructed on the basis of memory (RAM) blocks using the methods [7]. Applying the results [7] the HFSM functionality can be altered by reloading the RAM contents. Thus, we are able to change one set of instructions $z^i_1,\ldots,z^i_{q1}$, needed for the one problem i to another set of instructions $z^j_1,\ldots,z^j_{q2}$, needed for another problem j through reloading the RAMs.

Figure 4 presents additional explanations. Suppose that all application-specific instructions from the set $z^1_1,\ldots,z^1_{q1}$ needed for solving the Boolean satisfiability problem have been implemented. The decoder of the instructions tests the instruction code and activates the appropriate micro-program from the set $\mu^1_1,\ldots, \mu^1_{q1}$, executing the relevant micro-instructions. Assuming that the Boolean satisfiability does not take the maximum number of instructions, some decoder outputs (such as $z^i_{q1+1},\ldots$) do not invoke any micro-program (see Figure 4). A virtual table, shown in Figure 4, establishes a correspondence between the names of instructions (these names are used in a program of the fixed instruction set processor) and micro-programs that have to be invoked for execution of the instructions. Any association might be established and, thus, any element from the set of instructions can be associated with any micro-program.

Programs of the fixed instruction set processors have to be developed taking into account the established association with micro-programs and, thus, with the aid of the virtual table.

Let us assume that we would like to solve another combinatorial problem, such as matrix covering. This would require executing the following steps:

- The set $z^i_1,\ldots,z^i_{q1}$ of SAT-targeted instructions is changed to the set $z^j_1,\ldots,z^j_{q2}$ of covering-targeted instructions. Consequently the number of instructions could also be altered i.e. it becomes either larger or smaller. In the first case some of empty micro-programs will be replaced with real micro-programs. In the second case some of real micro-program will be replaced with empty micro-programs;
- Each unique for the SAT problem micro-program will be either completely removed or replaced with a unique for the covering problem micro-program. The latter is done with the aid of the reconfiguration controller through the reprogramming of HFSM RAM blocks;
- The virtual table is properly updated and, thus, the program of the fixed instruction sets processor has to be rewritten in terms of new operations indicated in the virtual table.

Note that any instruction might be implemented differently, for example, either sequentially (in such a way that is shown in Figures 3a and Fig. 3c) or in parallel (in such a way that is shown in Figures 3b and Fig. 3d) and, thus, influence the performance and the needed hardware resources. It permits to conclude that the proposed technique is very flexible for solving combinatorial search problems with different optimization criteria and undoubtedly helpful for experiments and evaluations of implementation platforms (*e.g.* sequential vs. combinatorial implementation of problem-specific operations) for different algorithms).

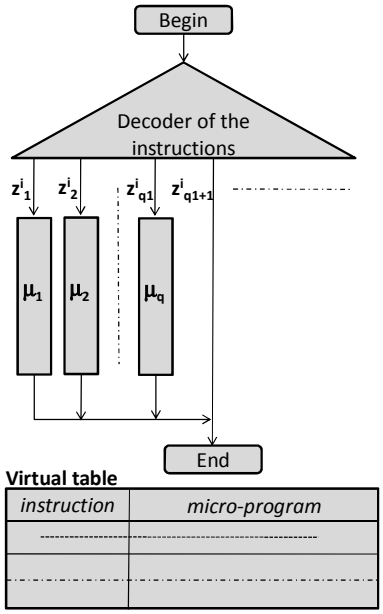

Figure 4. Implementation details for variable instructions.

V. PROGRAM EXECUTION

All available instructions are divided in fixed and variable groups. Fixed instructions are implemented much like [4,5]. Variable instructions are handled by a virtual table (see Figure 4) with lines that look something like the follows:

Name -Micro-program,

where *Name* is an alterable sequence of characters that indicate the relevant instruction (*e.g. ORT* for the orthogonality), *Micro-program* is the code of the module that has to be called applying hierarchical specification of micro-programs proposed in [8]. Since both the *Name* and the code of *Micro-program* can be altered in the virtual table, we are able to associate arbitrary names with different instructions and the respective micro-programs. Indeed, suppose *ORT* is needed for the SAT problem [5] and it is associated in the virtual table with a micro-program for verifying orthogonality between two ternary vectors. The same line of the virtual table might be reprogrammed in a way indicating that, for example, *AL0* (verifying if the vector contains all zeros that is needed for covering problem [5]) calls a micro-program for executing *AL0*. The change of functionality of RFSM (that enables the implemented micro-programs to be modified) is provided with the aid of methods [7].

Execution of any program is very similar to [4] with the only difference that the program instructions are distributed

dependently on their names and executed either in the processor or in the coprocessor.

Communication between the processor and the co-processor is established as follows:

- When the program of the fixed instruction-set processor requires just application-independent or shared (common) application-specific instructions, it is executed entirely in the processor;
- As soon as an application-specific (not common) instruction is required, the processor sends the instruction code to the co-processor. The latter activates a micro-program based on given association in the virtual table. As soon as the results are ready they are sent back to the processor.

VI. Wireless Interactions

Wireless interactions provide support for remote change of the processor's program and the co-processor's instructions aimed at optimization of the designed system for a particular problem (such as the considered above SAT and covering). Data needed for reconfiguration have to be prepared before execution and stored in a memory of a device responsible for reconfiguration. During execution the data have to be read and transferred to the dedicated device.

To change remotely the processor program and the co-processor instruction set an additional (auxiliary) FPGA was used, that communicates with the dedicated (processor + co-processor) FPGA through a wireless interface and includes memories containing all necessary configuration data. For the considered above example these data enable the processor and the co-processor to be customized for solving either the first (SAT) or the second (covering) combinatorial problem. Wireless communications is established with the aid of CC1101 module [9] and the developed protocol stack. The module CC1101 is connected with the FPGA through an interface [9].

VII. Implementation and Experiments

The considered computational system has been implemented in two prototyping boards with allocation of the dedicated FPGA (Spartan-3E family) in Digilent Nexys-2 [10] and the auxiliary FPGA in Celoxica RC10 [11]. Two different modes (wired and wireless) have been verified. The first mode was used for debugging purposes (see the left-hand part of Figure 5). The second mode (see Figure 5) enables the functionality of the entire system described in the paper to be verified.

The results of experiments confirm the correctness of the intended functionality. The reprogramming time allowing to optimize the system for solving either the SAT or the covering problem is about 18 seconds with a 5 Kbits/s wireless connection speed. This time can be reduced significantly in case of wired interactions between the dedicated and auxiliary circuits. The maximum distance between two FPGAs interacting through wireless interface is about 240 meters. Additional details are given in [12].

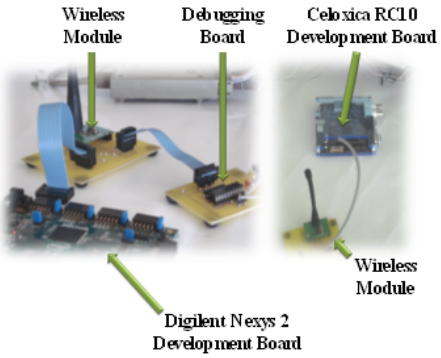

Figure 5. Organization of experiments.

VIII. Conclusion

The paper suggests architecture of communicating fixed and variable instruction set processors and presents a case study illustrating applicability of the proposed system for selected problems from the scope of combinatorial search algorithms. The experiments confirm the correctness of the intended functionality and show advantages of the technique for real-world problems.

References

[1] D. Atienza, F. Angiolini, S. Murali, A. Pullini, L. Benini, G. De Micheli, "Network-on-Chip design and synthesis outlook", *Integration, the VLSI Journal* 41, 2008, pp. 340-359.

[2] A. Sangiovanni Vincentelli, D. Densmore, J. Cong, R. Marculescu, "System-Level Synthesis - Functions, Architectures, and Communications", ASP DAC 2008 *Tutorial*.

[3] J. Liu, F. Chow, T. Kong, R. Roy, "Variable Instruction Set Architecture and Its Compiler Support", IEEE Trans. on Computers, vol. 52, no. 7, July 2003, pp. 881-895.

[4] Available at: http://www-md.e-technik.uni-rostock.de/lehre/vlsi_i/proc8/index.html.

[5] I. Skliarova, V. Sklyarov, "Design Methods for FPGA-based implementation of combinatorial Search Algorithms", *Proc. Int. Workshop on SoC Design - IWSOC'2006*, Yogyakarta, Indonesia, Dec. 2006, pp. 359-368.

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

[7] V. Sklyarov, "Reconfigurable models of finite state machines and their implementation in FPGAs", *Journal of Sys. Arch.*, 47, 2002, pp. 1043-1064.

[8] V. Sklyarov, "Hierarchical Finite-State Machines and Their Use for Digital Control", *IEEE Trans. on VLSI Systems*, vol. 7, no. 2, pp. 222-228, 1999.

[9] CC1101 Low-Cost Low-Power Sub-1GHz RF Transceiver: www.ti.com.

[10] Available at: http://www.digilentinc.com/.

[11] Available at: http://www.embeddedstar.com/press/content/2005/5/embedded18313.html.

[12] J. Lima, "Remotely Reprogrammed Variable Instruction Set Processor", *M.Sc. Thesis*, University of Aveiro, Portugal, 2009. Available at: http://radiosrd.com.sapo.pt/thesis.rar.