Design Tools for Reconfigurable Embedded Systems

Manuel Almeida, Valery Sklyarov, Iouliia Skliarova, Bruno Pimentel
Department of Electronics and Telecommunications/IEETA
University of Aveiro 3810-193 Aveiro Portugal
manuel.almeida@ieeta.pt, skl@det.ua.pt, iouliia@det.ua.pa, pimentel@ieeta.pt

Abstract

This paper describes the developed hardware/software tools, libraries and design methods for FPGA-based embedded systems which include: a kernel prototyping board with the Xilinx Spartan 3 FPGA; a set of projects for reusable FPGA-based circuits; utilities for FPGA programming; software/hardware tools that provide support for reconfiguration; and programs enabling the designers to partition the functionality of the developed system between software, running on a PC computer, and hardware, implemented in FPGA. A special attention has been paid to data exchange between a host computer and the kernel prototyping board based on compression/decompression techniques. Examples of practical applications are also presented.

1. Introduction

With the advent of Field Programmable Logic Devices (FPGA in particular) it became possible to design and implement digital systems without the need for technological steps dealing with silicon.

Developing digital devices on the basis of high capacity FPGAs requires using numerous tools, such as FPGA-based boards for rapid prototyping, computer-aided design (CAD) systems, libraries, IP (intellectual property) cores, etc. The paper suggests and describes such tools targeted to the design of heterogeneous embedded systems, whose specifications may change continuously [1]. The developed tools include:

- Core Xilinx Spartan 3 FPGA-based prototyping board with an external USB interface. The latter is a removable block, which might be replaced if necessary;
- Extension boards for connecting typical peripheral equipment and solving application-specific problems, such as implementation of interfaces, communication with standard devices (such as memories), etc.;
- Software tools for configuring the FPGA, debugging hardware projects, interactions with the board and providing the required experiments;
- Hardware/software tools for data compression and decompression that enable the volume of data transmitted from/to FPGA to be reduced significantly;
- Hardware/software support for reconfiguration of FPGA-based circuits through reloading reconfiguration bitstreams, which have to be preliminary stored in a flash memory available on the board;
- Hardware/software tools for executing operations over binary/ternary vectors and matrices;
- VHDL templates for interactions with typical peripheral devices, such as an USB controller, a VGA monitor, a static RAM, a keyboard, a mouse, an LCD panel, etc.;
- VHDL templates and IP cores for application-specific circuits;
- Useful VHDL and Handel-C projects providing essential support for developing embedded systems.

The considered above set of hardware/software tools can easily be retargeted to different engineering application areas. Thus, it is very well suited to the design of heterogeneous systems. Any particular problem can be solved using just a selected subset from the considered set, which includes only the needed hardware/software components and excludes all the other available components that are not required for the developed application. In case if the desired components are not available they can easily be constructed (i.e. a new extension board can be designed, validated and tested and all these steps are well supported by the developed tools).

The remainder of this paper is organized in nine sections. Section 2 describes the designed Spartan 3 FPGA-based prototyping board. Section 3 shows how

to configure onboard hardware and discusses the developed software tools. Section 4 describes methods for data compression and decompression used in the designed software and hardware. Section 5 shows how to construct systems based on the developed board and applying the technique of dynamic reconfiguration. Section 6 explains how functional capabilities of the board can be extended. Section 7 characterizes the developed libraries, IP cores, hardware templates and projects for the board. Section 8 discusses potential practical applications. The conclusion is given in section 9.

2. Extendable prototyping board

The basic architecture of the board (see fig. 1) has been selected in such a way that allows satisfying the objectives indicated in the previous section and enabling many design problems from the scope of embedded systems to be solved.

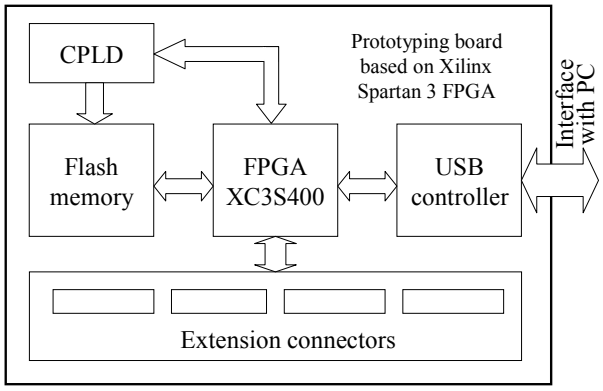


Figure 1. Basic architecture of the kernel prototyping board

The following features have been provided:

1. Powering and programming the board from a PC or any host microcomputer through an USB port. If necessary, an external power source can also be used. Since onboard USB controller is a removable block it can be replaced with wireless controller, such as Bluetooth. This permits to program the board remotely easily verifying alternative implementations.

2. Keeping the bitstream for the FPGA in a flash memory, which permits to use the board as an autonomous device without any connection to PC and only an external powering has to be provided.

3. Keeping more than one bitstream in the flash memory for reconfiguration of FPGA and testing alternative design projects. In fact, the capacity of the flash memory permits up to 6 bitstreams to be stored. This is very practical not only for run-time reconfiguration but also for verification of different types of alternative and competitive implementations, which is very important for the design of embedded systems.

4. User-friendly software interface for programming the board, data exchange with PC and reconfiguration.

5. Data exchange with any other device supporting standard USB port.

6. Extension connectors for the design of application-specific and education-targeted externally connected boards.

The flash memory can be divided in two or more sections. The first section contains a bitstream that has to be pre-loaded to the FPGA in order to be able to download an application-specific bitstream to the second section. This technique has already been used in the Trenz Electronic prototyping boards [2]. The other sections enable the designer to store more than one application-specific bitstreams for configuring the FPGA. This memory can also be used for keeping any arbitrary data such as bitmaps for a VGA monitor. The CPLD controls the flash memory and push buttons for pre-loading the initial bitstream and for running other applications. The USB controller provides data exchange with PC (or any other host microprocessor-based system) for downloading FPGA bitstreams and communications between FPGA circuits and general-purpose software running on a host computer. Extension connectors permit many application-specific external boards to be attached, which enables the designer to optimize resources, to improve performance and to extend the functionality. Fig. 2 presents a photograph of the board which has more than 80 available pins for extension connectors.

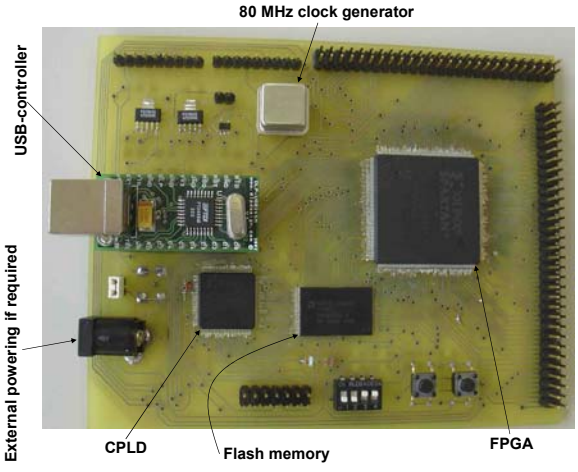


Figure 2. The designed prototyping board

3. Programming the board

Fig. 3 demonstrates basic components of software for configuring the board and interactions with a computer. Feasible design flows for the Xilinx ISE [3] and Celoxica DK design suite [4] are also shown.

Any user project can completely be developed in the Xilinx ISE or in any similar environment, which finally generates a bitstream that is ready for downloading to the FPGA. On the other hand system-level specification tools (such as Handel-C) can also be used. For example, in fig. 3 the DK of Celoxica translates a Handel-C project description to an electronic design interchange format (EDIF) file, which is further converted in the ISE to a bitstream for the FPGA.

Fig. 4 demonstrates the technique, which is used for configuring the FPGA and downloading bitstreams for user projects.

Different bitstreams can be stored in a host computer (PC, for example) memory. The required bitstream for downloading can be selected and transmitted to the second section of the flash memory (see fig. 4). From the side of hardware the downloading is managed by the circuit whose bitstream is permanently kept in the first section of flash memory. The bitstream from the first section is used for configuring FPGA as soon as the *Configuration* pushbutton is pressed. After that the board is ready for downloading any user bitstream to the second section of the flash memory with the aid of the developed software components (see fig. 3). As soon as the user bitstream is loaded pressing the *Project* pushbutton initiates configuration of the FPGA from the second section of the flash memory, i.e. for the user application. The CPLD is used for controlling the flash memory and the push buttons because during configuration the FPGA cannot execute these tasks. The CPLD generates also an initial *reset* signal for FPGA circuits as soon as a new configuration is completed.

Obviously, downloading the initial bitstream to the first section of the flash memory cannot be done in the mode considered above. For such purposes a JTAG connector [3] (see fig. 4) is used. Note that JTAG mode has to be provided just once during the board fabrication (initialization). There exists also an opportunity to reload the bitstream to the first section if required.

The developed software permits to send (to write) arbitrary data to the board and to receive (to read) the board data through a USB as well as to exchange data between the board and PC programs developed in any programming language (see fig. 3). Finally, convenient windows-based user-friendly interface is provided.

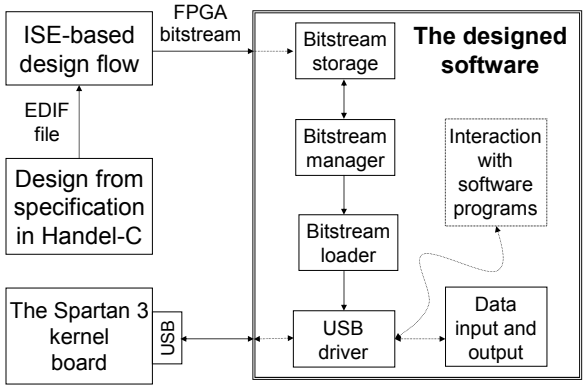


Figure 3. The developed software components and feasible design flows

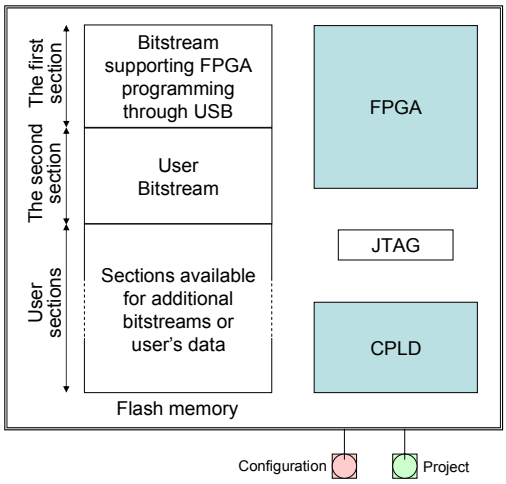


Figure 4. The technique for configuring the FPGA and downloading user bitstreams

More than one bitstream can be copied from PC to the onboard flash memory. In this case the second and other bitstreams will be stored in the user sections. The developed software permits also to read data from the flash memory to the host PC. Additional bitstreams can be used in the two following modes:

1. Autonomous test of different projects without connections to PC. In particular this mode can be used for comparing and validating alternative/competitive implementations. Selection of the proper flash memory source section, keeping the desired bitstream, is provided with the aid of a simple additional switcher attached through an extension connector and supplying

information about the number of the section to the CPLD.

2. Programming FPGAs installed on extension boards including capabilities for dynamic reconfiguration. In the last case the kernel board FPGA is considered to be a controller (manager) for a run-time reconfigurable system, which permits to implement circuits that require more resources than resources available on the kernel and extension boards. These capabilities will be considered in section 5 with more details.

For certain applications (such as [5]) the developed software enables the designer to partition the functionality of the designed system between software, running on a host computer, and hardware, implemented in FPGA. This opportunity has been provided for solving search problems that can be formulated over Boolean and ternary matrices. It is known that the most common approach used for such purposes is based on construction of a search tree [6]. The root of the tree corresponds to the matrix **M**, which contains initial data for the given problem. All other nodes of the tree represent minors (sub-matrices) of the initial matrix which have been constructed during the search process. Any minor is built up by removing some columns/rows from the matrix corresponding to the parent node. For example a hardware/software SAT solver discussed in [6] executes the following sequence of steps beginning from the root of the tree:

- If FPGA has sufficient resources to handle the initial matrix **M** the SAT problem is completely solved in hardware;
- If FPGA does not have sufficient resources then software applies methods [6] allowing the initial matrix to be reduced. These methods are repeated while a minor (sub-matrix) obtained at a current step does not satisfy constraints settled by the available FPGA resources. As soon as these constraints have been satisfied the FPGA hardware will be responsible for the subsequent steps, i.e. the minor will be transmitted to FPGA for further processing;
- If hardware solves the problem the result is dispatched to the host computer;
- If the considered branch of the search tree does not allow a solution to be found the software is informed through a special signal and a new branch of the tree is selected. If a new branch does not exist the problem is unsatisfiable.

The considered technique involves operations over discrete matrices in software, which make possible to activate some applications-specific procedures, and to initiate data exchange with hardware as soon as this is required. A software library, which provides support for such operations, has been designed.

4. Data exchange

We have already mentioned (see the previous section) that for a number of practical applications the complexity of FPGAs is not sufficient and the resources have to be partitioned between software running on a host computer (such as PC) and hardware (such as an FPGA). Some examples of applying this technique can be found in [6]. This involves multiple data exchange between hardware and software, which is time consuming. The same problem might appear when several boards interact through extension connectors. To reduce the volume of data (and consequently the number of pins for data exchange) a compression/decompression technique has been applied.

The method of compression/decompression, used for data exchange with the board, is based on Huffman algorithm with some modifications making possible to take into account the number of repetitions of consecutive segments that have to be coded. Implementation of the method in FPGA has been provided with the aid of recursion using the model of a hierarchical finite state machine [7]. Any segment for compression is considered to be a binary vector. It is allowed the size of segments to be changed, which enables us to verify the influence of the size on the compression ratio. Fig. 5 demonstrates the basic steps for data compression.

In our example (see fig. 5, a) each segment consists of 7 bits and totally there are 6 segments, some of which are repeated either consecutively (such as C) or non-consecutively (such as B). Characteristics of the segments are saved in the table of repetitions (see fig. 5, b). Binary tree for sorting (see fig. 5, c) takes into account the number of non-consecutive repetitions. The successive steps (see fig. 5 d,e,f) for constructing the Huffman tree have been provided in accordance with the method [8] and the final tree is depicted in fig. 5,f. The codes for the compressed segments, taken from the Huffman tree in fig. 5,f, are shown in the coding table (see fig. 5, g). As a result the compression ratio (see fig. 5, a, h) is equal to 5.25.

Decompression is executed over resulting codes (see fig. 5,h) with the aid of the coding table (see fig. 5, g). Segments that can be repeated are indicated by a special flag. Thus, it is known that after the segment code the indicated number of repetitions with predefined size has to be taken into account.

Software/hardware implementation details for the considered methods are given in [9]. The implemented algorithms make possible to change the size of segments for compression and decompression.

The relevant software/hardware tools have also been included in the respective library namely C++ functions and classes for PC computers and VHDL/Handel-C projects for FPGAs.

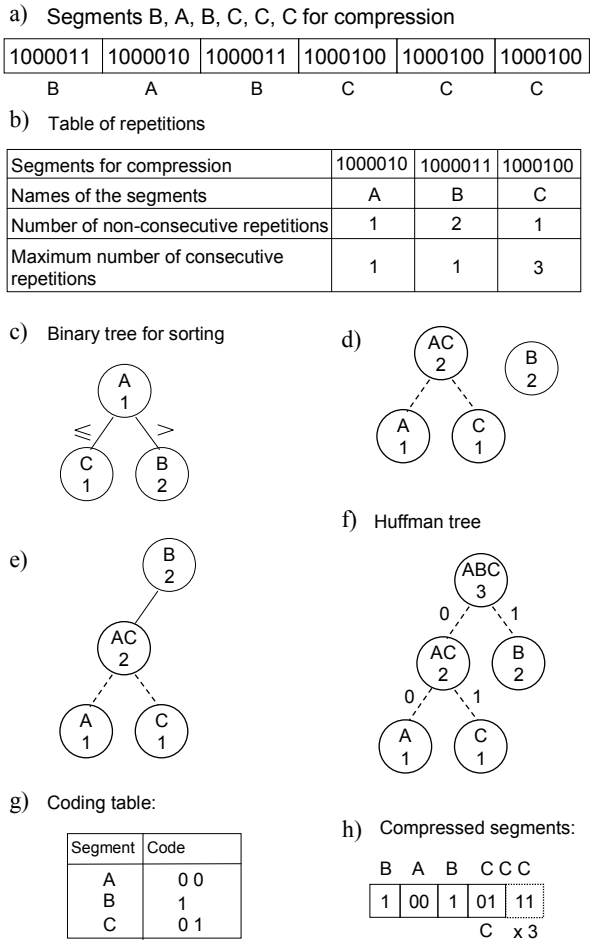


Figure 5. Basic steps of the compression method

5. Dynamic reconfiguration

Fig. 6 demonstrates basic steps of the method that can be used for dynamic reconfiguration of FPGA. There exists also an opportunity to construct multi-board dynamically reconfigurable systems in such a way that each board can be autonomously reconfigured. This opportunity enables the designer to built very complicated embedded systems composed of circuits that can be flexibly modified and updated during run-time. Using this technique we can implement a system that requires some hardware resources R_c, on available hardware that has resources R_h, where $R_c > R_h$. Applying the method for downloading of bitstreams to the flash memory allows to construct adaptive embedded systems, which might change their functionality dependently on external events that can be analyzed by the host computer The latter permits to form and to download the proper configuration in response to external events.

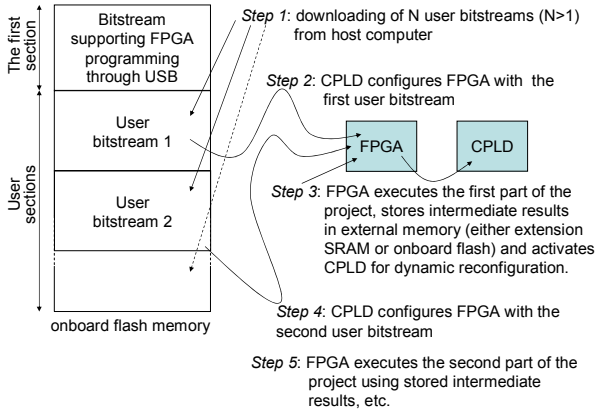


Figure 6. Reconfiguration of FPGA

6. Extension boards

Extension boards enable the designers to assemble the required embedded system from relatively complicated sub-systems that are supported by hardware (libraries, projects and IP cores) and software (application-specific programs running on a host computer) tools (see the next section for details). Fig. 7 demonstrates possible use of extension boards. Each extension board contains either interface circuits (and/or peripheral-specific connectors) or application-specific circuits that will be considered below in more detail.

Fig. 8 illustrates an application-specific extension board, which has been designed for problems of combinatorial optimization formulated over Boolean and ternary matrices. The relevant tasks have to be solved for some embedded systems (see, for example, [5,10]).

Ternary matrices are saved using the method [6]. Two FPGAs, shown in fig. 8, enable different operations over rows and columns of the matrix to be executed in parallel. The flash memories can be loaded from the kernel board and this permits different

problems over discrete matrices to be solved using the same extension board.

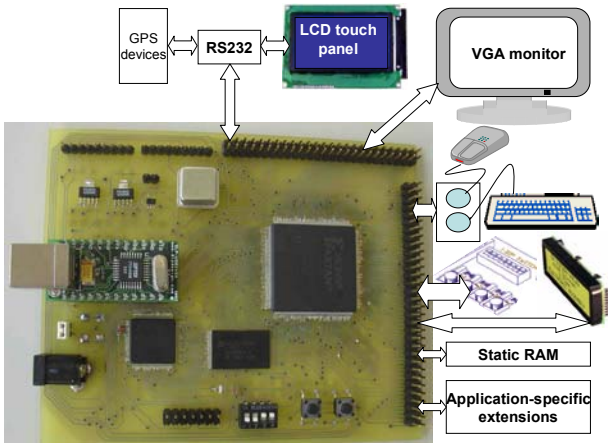


Figure 7. Feasible extensions

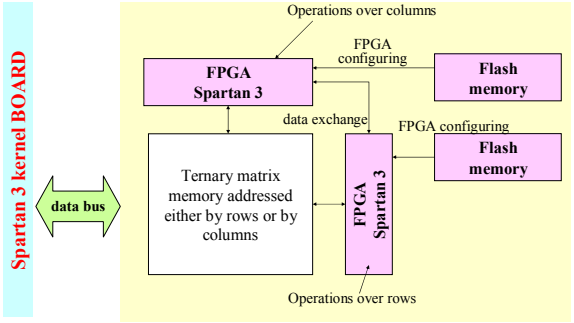
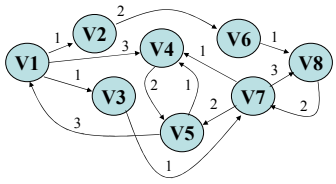


Figure 8. Extension board for solving problems of combinatorial optimization

Note that the same extension board can also be used for solving problems over graphs. Fig. 9 shows an example demonstrating how to code the given directed weighted graph with weights 1,2,3.

Using greater values for weights might easily be provided through onboard connectors for additional memory blocks (see static RAM block in fig. 7) allowing to store N-ary discrete matrices (N>3). Thus, many known problems over graphs can also be solved and the desired customization can be provided through reloading available flash memories (see fig. 8), which enables different algorithms to be implemented. It is important that the table representing graph in memory (see fig. 9) can be addressed either by rows or by columns, for example, we can immediately find all edges with any destination vertex and at the same time (i.e. in parallel) all edges with any source vertex.



	V1	V2	V3	V4	V5	V6	V7	V8
V1		1	1	3				
V2						2		
V3							1	
V4					2			
V5	3			1				
V6								1
V7				1	2			3
V8							2	

0 has to be written in all empty boxes

Figure 9. Coding graphs in memory shown in fig. 8

7. Design libraries and IP cores

The core (kernel) board has been designed for the primary use in the following three areas:

1. Research activity that requires: validating novel algorithms; comparing different methods for hardware design; providing for experiments with the designed circuits.
2. Engineering design, where the considered methodology enables the designer to assemble the developed embedded system from the existing general-purpose and application-specific components, to implement just a new functionality and to provide for rapid prototyping.
3. Education, where there exists an opportunity to study many typical and new devices, to simplify the design of quite complicated systems, to concentrate the students' efforts just on original components that have to be developed within the relevant disciplines and to test the design in a physical environment using recent reconfigurable devices and CAD systems.

In general, the library components and IP cores suggested can be divided into two groups [11]. The first group provides for interaction with various external devices that support data input and output. The second group is composed of application-specific circuits.

The circuits can be employed as either components of more complicated devices implemented in an FPGA or as auxiliary blocks that simplify debugging, testing, and experiments with FPGA-based embedded systems [11]. In the second case they can be connected to another device in order to provide for interfaces with a variety of peripheral equipment for supplying input data to the tested circuit and displaying output data generated by the circuit. Input data can be packaged in order to build a control sequence, which represents a test bench. Outputs from the circuit, which are generated in response to the control sequence, can be saved in storage for future examination and analysis (see details in [11]).

Fig. 10 demonstrates two application-specific library components. Many embedded systems require the execution of various operations over Boolean and ternary matrices. Such systems deal with two-dimensional arrays with elements having either two (0 and 1) or three (0, 1, and don't care) values. As a rule, an access has to be provided to individual row/columns of the matrices and this requires dual-address storage.

Fig. 10,a depicts the respective reusable block, which includes RAM, mask registers that permit some rows and columns to be excluded from the matrix to select a minor, and columns/rows registers that provide dual-address access. Fig. 10,b demonstrates the functionality of a reusable circuit that selectively generates the necessary RAM addresses. The latter allows the rows/columns, which are not required for the minor that is being extracted during the current step, to be skipped. Let us assume that a matrix is composed of 8 rows *1,2,…,8* and 7 columns *A,B,…,G* (see fig. 10,a). We want to remove all the rows that have "0" in column *A* and all the columns that have just don't care values ("-"). Fig. 10,a shows the contents of the mask registers for this case. Suppose that all the rows that are indicated by ones in the row mask register have to be skipped. Fig. 10,b demonstrates how the relevant reusable block implements this requirement.

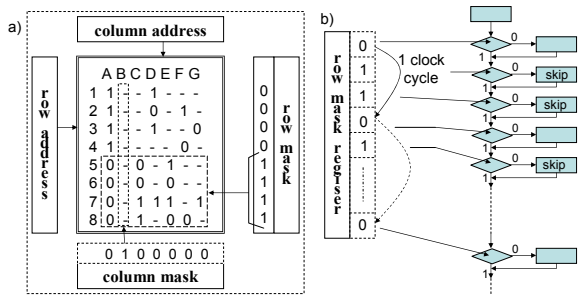


Figure 10. Application-specific library components

8. Potential applications

The mentioned above and the other designed library/hardware/software components can be employed for both practical design and educational purposes. For example, a number of tutorials that explain how to work with various tools targeted to FPGAs and how to use these tools to the design of different types of FPGA-based circuits are available at [12]. They are organized in such a way that any circuit being tested is considered to be the core of some working environment and the remainder of the working environment provides for interaction of the circuit with the external world. Some examples of the developed components, partially available at [12], are listed below:

- VHDL code for interface with a VGA monitor;
- VHDL template for hierarchical finite state machines;
- VHDL code for interaction with static RAM;
- VHDL code for interaction with flash RAM;
- Xilinx ISE projects for interactions with push buttons, DIP switchers and LEDs;
- Xilinx ISE project for interaction with an LCD panel with touch capabilities through RS232 interface;
- Handel-C projects for data compression and decompression;
- VHDL projects for recursive and iterative data sorting;
- Handel-C projects for solving combinatorial problems formulated over Boolean and ternary matrices;
- Handel-C/VHDL projects for application-specific computations, etc.

Fig. 11 depicts one potential example that is very helpful for education purposes. It demonstrates how to implement a simple microcomputer on the basis of proposed methods and tools. The microcomputer can be assembled from the kernel Spartan 3 prototyping board, the extension connectors and some peripheral devices, which can be attached, namely a VGA monitor, a keyboard and a mouse. The designed libraries make possible to include blocks for implementing all the required interfaces such as that are needed for communication with the peripheral devices and only the circuits for the microcomputer itself have to be designed.

A number of other helpful examples demonstrating the use of the developed tools in educational process can be found in [13].

The other important scope for the considered methods and tools is the design space exploration and experiments. For example, two alternative strategies, which can be applied to the design of many computational devices, are iteration and recursion [14]. They require non-equivalent resources for implementation in software and in hardware. For example, in most high-level programming languages, a function call incurs a bookkeeping overhead. Recursive functions magnify this overhead because a single initial call to the function might generate a large number of recursive invocations of the function. On the other hand in [8] it is shown that recursion can be

implemented in hardware much more efficiently. This is because any activation of a recursive subsequence of operations has been combined with the execution of the operations (micro operations) that are required by the respective algorithm. The same event takes place when any recursive sub-sequence is being terminated, i.e. when control has to be returned to the point after the last recursive call and an operation of the algorithm that follows the last recursive call has to be activated. The number of states required for the execution of recursion in hardware can be reduced compared to software. Besides, such states are accommodated on stacks typically implemented on FPGA built-in memory blocks, which are relatively cheap. The considered board makes possible to compare recursive and iterative solutions for complex algorithms. This approach was used in [14].

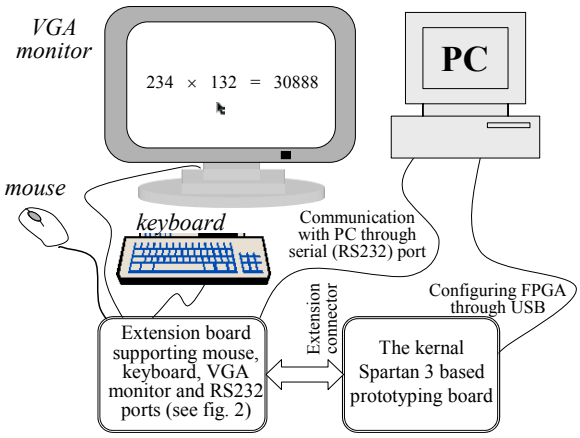


Figure 11. Using the designed tools in education

9. Conclusion

FPGA-based circuits have proven their effectiveness for many practical applications. The paper suggests one potential technique and particular tools for rapid prototyping of embedded systems and design space exploration. The tools include the developed core (kernel) prototyping board and two types of extensions for standard peripheral devices and special-purpose systems. This enables the users to assemble complex systems from the supplied blocks supported by the proposed libraries and IP cores, which significantly simplifies the design process and shortens the design lead time. The technique gives an easy opportunity to extend and to customize the functionality of available hardware allowing constructing circuits that are optimized for particular applications. This enables the users to combine advantages of block-based and application-specific design. The developed libraries, projects and examples can be seen as additional resources for successful use of the proposed tools in research activity, in engineering practice, and in education.

10. References

[1] S. Edwards, L. Lavagno, E.A. Lee, A. Sangiovanny-Vincentelli, "Design of Embedded Systems: Formal Models, Validation, and Synthesis", *Proc. of the IEEE*, vol. 85, no. 3, March, 1997, pp. 366-390.

[2] Available at: www.trenz-electronic.de.

[3] Available at: http://www.xilinx.com/.

[4] Available at: http://www.celoxica.com/

[5] R. Feldman, C. Haubelt, B. Monien, J. Teich. "Fault Tolerance Analysis of Distributed Reconfigurable Systems Using SAT-Based Techniques". *Proc. of FPL'2003*, Lisbon, Portugal, 2003, pp. 478-487.

[6] I. Skliarova, A.B. Ferrari, "A Software/Reconfigurable Hardware SAT Solver", *IEEE Trans. on VLSI Systems*, vol. 12, n. 4, April, 2004, pp. 408-419.

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

[8] V. Sklyarov. "FPGA-based implementation of recursive algorithms". *Microprocessors and Microsystems*. Special Issue on FPGAs, 2004, Vol. 28/5-6 pp 197-211.

[9] B. Pimentel, J. Arrais. "Implementação de Algoritmo de Compressão e Descompressão de Dados para Modelo de Co-processamento baseado em FPGA's". *Electrónica e Telecomunicações*, Vol. 4, no. 10, Jan. 2004, pp. 215-220.

[10] J. Gu, P. W. Purdom, J. Franco, B. W. Wah, "Algorithms for the Satisfiability (SAT) Problem: A Survey", DIMACS *Series in Discrete Mathematics and Theoretical Computer Science*, vol. 35, 1997, pp. 19-151.

[11] V. Sklyarov, I. Skliarova, P. Almeida, and M. Almeida, "Design Tools and Reusable Libraries for FPGA-Based Digital Circuits", *Euromicro Symposium on Digital System Design*, Belek, Turkey, Sep. 2003, pp. 255-263.

[12] Available at: http://elearning.ua.pt.

[13] V. Sklyarov, I. Skliarova, "Teaching Reconfigurable Systems: Methods, Tools, Tutorials and Projects", *IEEE Trans. on Education*, Vol. 48, No 2, 2005, pp. 290-300.

[14] V. Sklyarov, I. Skliarova, B. Pimentel, "FPGA-based implementation and comparison of recursive and iterative algorithms". *Proc. of FPL'2005*, Tampere, Finland, 2005, pp. 235-240.