ICROS-SICE International Joint Conference 2009
August 18-21, 2009, Fukuoka International Congress Center, Japan

Virtual Environment for Prototyping and Experiments

Valery Sklyarov and Iouliia Skliarova

Department of Electronics, Telecommunications and Informatics,
University of Aveiro/IEETA, Portugal (Tel : +351-234-401-539;
E-mail: skl@ua.pt, iouliia@ua.pt)

Abstract: The paper suggests the methodology for validation of designed systems and experiments through the use of the developed simulation tools and interactive models. The primary idea is prototyping on the basis of virtual visual samples imitating interacting physical objects. The desired electronic functionality is implemented in reconfigurable hardware and, therefore, can easily be changed, which significantly simplifies such problems as verification of alternative implementations and eliminating potential errors. The desired physical functionality is modeled through observation of and experiences with interacting images which look like physical objects. The proposed methodology is very helpful and effective for many practical applications, such as mechanical systems control, robotics, process automation, transportation systems, computational devices, etc.

Keywords: visual simulation, prototyping, FPGA

.

1. INTRODUCTION

Modeling and simulation are useful techniques that permit the designed systems to be verified and evaluated. Dependently on target requirements these techniques can be applied differently. For example, when we consider electronic devices, they can be checked with the aid of a test-bench model that provides input values (test cases) to the device and tests if output values from the device are correct. Very often a test-bench model is composed of entities, coded in a hardware description language (such as VHDL [1]) and executed in software simulator in such a way that one entity provides inputs, another one imitates the functionality of the device and the results are presented as waveforms of output signals displayed on a monitor screen. Different varieties of test-bench models and the relevant techniques are described in [1].

For many practical applications it is easier to verify the functionality of developed devices using visual modes. Such modes are especially beneficial for devices with slowly changing states, which is quite common for mechanical systems, robotics, transportation systems, etc. For example, verification of a traffic light control considered in [1] as an example, can be much easier done on a picture of a traffic light displayed on a monitor screen. Such technique has already been used in numerous debugging tools, such as [2]. A more advanced technique is described in [3] where a combined software/hardware model of a system that consists of a control part (mainly implemented in hardware) and a datapath (modeled partially in hardware and partially in software) was proposed. Hardware (implemented on the basis of Field Programmable Gate Arrays - FPGA) and software (running in general-purpose computer) components communicate through a pre-established interface. Note that the capabilities of modern FPGA make it possible to implement the complete system in hardware and this paper reports the relevant results. The primary idea is prototyping on the basis of virtual visual objects imitating interacting physical entities in such a way that:

- The desired electronic functionality is implemented in a reconfigurable hardware and, therefore, can easily be changed, which significantly simplifies such problems as verification of alternative implementations and eliminating potential errors;
- The desired physical functionality is modeled through observation of and experiences with interacting images whose shapes look like physical objects.

One example is shown in Fig. 1 [4] that depicts two interacting objects of a self-controlled transport section: a robot on the left-hand side; and a container on the right-hand side that can move from left to right and vice versa. Functionalities of the robot and the container have to be provided in parallel (the relevant algorithms Z_1 for the robot and Z_2 for the container are given in [4]). In a simplified mode the robot is controlled by two actuators *move left* and *move right*, which force the respective motions. The track, where the robot is moving, is bounded by the sensors *left sensor* and *right sensor* in such a way that if *left sensor=1*/*right sensor =1*, the robot is at the left/right edge. The container has exactly the same behavior, except capital letters are used instead of lower-case letters in the description. The transfer is also controlled by algorithms Z_3- Z_6 in such a way that Z_3 forces the robot to take something at the left-hand side of the transfer line; Z_4 delivers the object carried by the robot to the container; Z_5 unloads the container at the right-hand side, and Z_6 tests the entire system before initiating the working stage. The algorithm in Fig. 1 assumes that several iterations are needed for the robot to load the container.

Taking into account all the considered above details we can conclude that Fig. 1 describes the functionality of a simplified real-world system. Functionality of this system can be verified using different techniques. For example, a test-bench model can be created for testing electronic components implementing the algorithms

$Z_1,\ldots,Z_6$. This paper suggests another way illustrated in Fig. 1, a. The system (Fig. 1, b) is displayed on a monitor screen (Fig. 1, a) and all motions are visualized in real time. Electronic components are implemented in an FPGA interacting with the monitor (and possibly with some other peripheral devices) and they control mechanical components in a similar way that is used in a physically realized system. The paper is dedicated to such type of environment that is very helpful for prototyping and experiments.

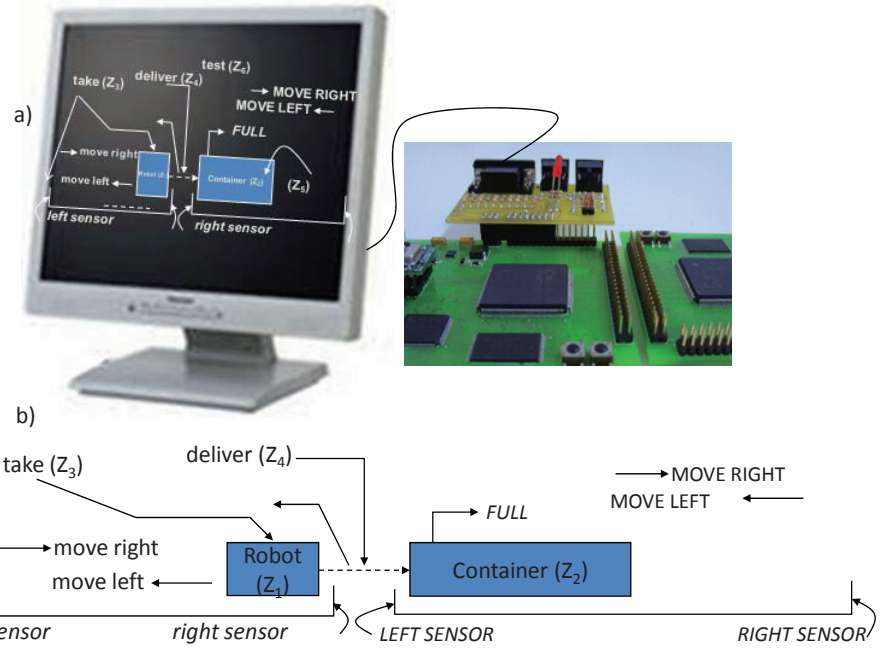


Fig. 1 Visual simulation (a) of a self-controlled transport section (b).

The remainder of the paper is organized in four sections. Section 2 suggests architecture of a visual environment and its basic components. Section 3 presents different examples of prototyping. Section 4 discusses implementation details. It also compares hardware with software/hardware techniques and points to their application areas with the relevant advantages and disadvantages. The conclusion is given in Section 5.

2. SYSTEM ARCHITECTURE

Fig. 2 depicts the proposed architecture of the simulation environment. Let us assume that we would like to verify the functionality of an electronic system that controls a transfer line [3]. Dynamic behavior of the line is displayed on a monitor screen and we can see a set of steps that are needed in order to complete some desired treatment of metallic components. Each step is controlled by a customizable sub-system implemented in hardware and communicating with a visual sub-system through a pre-established interface. Two sub-systems, as well as the interface, are implemented within the same FPGA.

There are three functional components in Fig. 2:

A. FPGA circuits for interaction with virtual controlled objects (such as a self-controlled transport section shown in Fig. 1);

B. An interface, transmitting signals to actuators and from sensors of virtual objects (block A) from/to the electronic controller (block C);

C. Physically implemented circuits realizing control algorithms that have to be verified and validated.

There are also some auxiliary components (such as a keyboard D, shown in Fig. 2) for human intervention.

The proposed architecture can be used in several ways:

- For verifying and validating functionality of software and hardware, controlling such objects for which correctness of behavior can be evaluated visually (e.g. mechanical systems control, robotics, process automation, transportation systems). Hardware is implemented in FPGA. Software is running on soft/hard processing elements often available as an IP-core or embedded in FPGA;
- For attaching virtual peripheral devices (*e.g.* liquid crystal displays, joysticks, dipswitches, pushbuttons) displayed on a monitor screen, which permits to verify prospective ideas for data input and output;
- For verifying design ideas. For example, the self-controlled transport section in Fig. 1 can be controlled by software running on built-in FPGA processing elements or, alternatively, by communicating finite state machines (such as that are studied in [4]) mapped to hardware circuits. In this case for the same A and B components of Fig. 2 we can compare and evaluate different circuits for the block C.

Fig. 3 shows more detailed architecture of the developed system. In particular, the main blocks of component A (see Fig. 2) are shown in Fig. 3, which

include a VGA controller, a VGA memory, and a customizable symbol memory. The latter can be programmed to keep and supply different images (see examples in Fig. 3) that compose the picture on a monitor screen.

Let us assume that we would like to model a system described in [5]. In this case the memory symbols (primitives) are cars, traffic lights, etc. Each particular primitive has an associated binary code. In order to display the primitive on a monitor screen, its code has to be written to a proper position of VGA memory. The latter can be organized differently depending on the size of matrix for the primitives in video pixels. Memory organization (as well as other similar parameters) is customizable through the relevant hardware description language (HDL) generic statements.

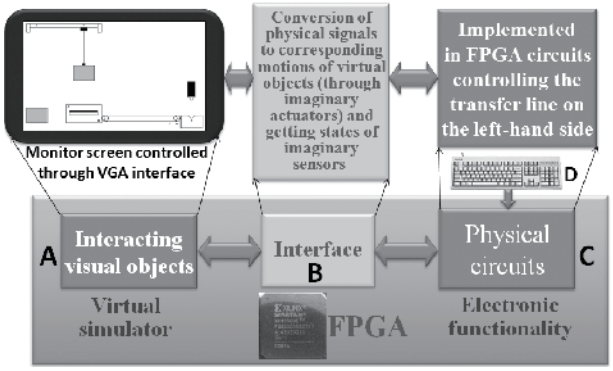


Fig. 2 Architecture of the simulation environment.

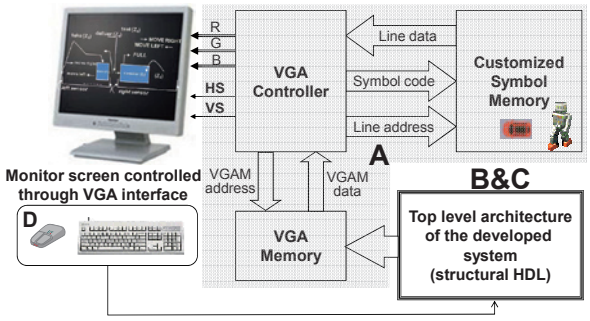


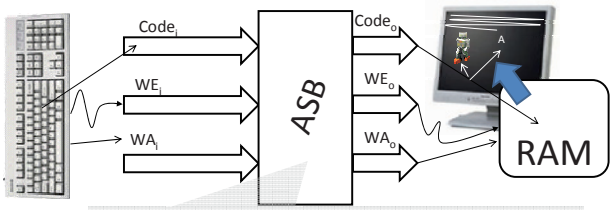
Fig. 3 More detailed architecture of the simulation environment.

Motions of visual objects are provided through periodic update of the VGA memory addresses for the selected from the symbol memory visual objects. This process is controlled by the interface circuits (B), which convert signals to actuators to the proper sequences of memory addresses with parameterizable rates of change. The components D enable the user to change parameters, such as speed of moving objects. The VGA controller [6] can be customized for the majority of commercial VGA monitors and video modes.

The component A (see Fig. 3) was described in VHDL [6] and the interaction of a tested application-specific block (ASB) with this component is established through the following interface:

```
ASB: entity work.name_of_ASB
  generic map (
     -- generic values
                )
  port map (
          ASCII_in => ASCII,
          ASCII_out => RAMData,
          Address_in => WriteAddress,
          Address_out=> RAMWriteAddress,
          clk          => Clock,
          rst          => Reset,
          WE_in        => WriteEnable,
          WE_out       => RAMWriteEnable,
-- other peripheral components, for example
          Button       => Button,
          Led          => Led,
          Switch       => Switch,
          segments     => segments,
          displays     => displays     -- etc.
               );
```

Fig. 4 demonstrates a possible use of the considered above interface. Let us assume that ASB receives data from a keyboard. In a trivial case when the keyboard supplies inputs (ASCII codes) and the symbol memory (see Fig. 3) converts ASCII codes to the relevant symbols on the screen, we can built a simple text editor through the connections of the keyboard outputs with the VGA memory (*e.g.* ASCII_out <= ASCII_in). In a more complex case the symbol memory converts input codes to special images, such as a robot shown in Fig. 4. Finally, the ASB supplies all necessary functionality of a simulated system, such as that is demonstrated in Fig. 1. In the last case the top part of Fig. 4 becomes more complicated (see Fig. 5).




```
entity name_of_ASB is
generic (    AddressBits          : natural;
             NumberOfColumns      : natural; -- ..............
          );
Port ( ASCII_in      : in  STD_LOGIC_VECTOR (7 downto 0);
       ASCII_out     : out STD_LOGIC_VECTOR (7 downto 0);
       Address_in    : in  STD_LOGIC_VECTOR (AddressBits - 1 downto 0);
       Address_out   : out STD_LOGIC_VECTOR (AddressBits - 1 downto 0);
       clk           : in  STD_LOGIC;
       rst           : in  STD_LOGIC;
       WE_in         : in  STD_LOGIC;
       WE_out        : out STD_LOGIC;
       -- ..............................
     );
end name_of_ASB;
```

Fig. 4 An example of interaction between the blocks A and C (see Fig. 2 and 3).

Simulation-targeted input/output blocks enable a user to supply input actions (if required) and to provide functionality in a visual mode based on interaction of visual objects on a monitor screen. There is also an

opportunity to change working clock frequency, which allows to increase or decrease speed of simulation. The next section presents some examples.

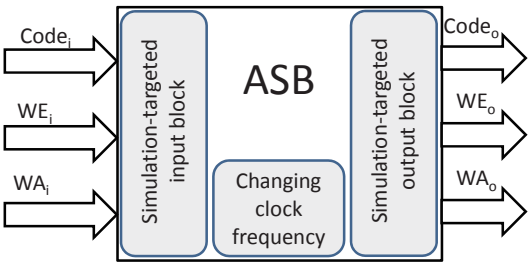


Fig. 5 ASB with attached input/output blocks supporting interaction with a user and visual simulation

3. EXAMPLES OF PROTOTYPING

Let us consider two potential modes of simulation that are message based and graphics based. The first mode is significantly easier to implement because it just allows displaying states of sensors (inputs) and actuators (outputs) in form of text messages. Suppose the functionality of a robot and a container in Fig. 1 is described by parallel hierarchical algorithms Z_1 and Z_2 [4] shown in Fig. 6.

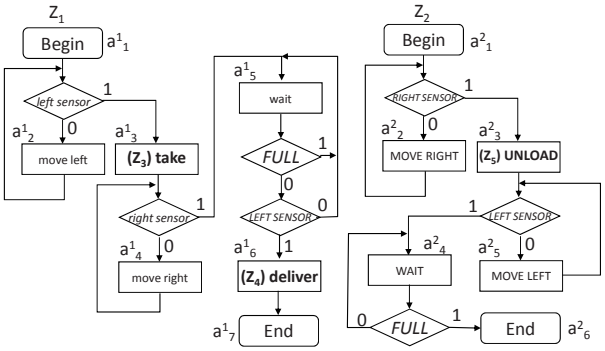


Fig. 6 Modules Z_1 and Z_2 for robot and container in Fig. 1.

Changing clock frequency enables us to execute Z_1 and Z_2 slower or faster and the frequency can be altered during execution time. There are 3 modules (Z_3, Z_4 and Z_5) in Fig. 6 that have to be activated hierarchically. Suppose Z_1 and Z_2 are activated simultaneously and the following sequence of messages is displayed:

robot left sensor	Off
robot right sensor	Off
robot move left	No
robot move right	No
container FULL	No
container LEFT SENSOR	Off
container RIGHT SENSOR	Off
container MOVE LEFT	No
container MOVE RIGHT	No
(Z_1) robot	Active
(Z_2) container	Active
(Z_3) take	Passive
(Z_4) deliver	Passive
(Z_5) unload	Passive

The values of inputs can be changed manually and these permit to verify correctness of control circuits that execute given algorithms (such as Z_1 and Z_2).

Graphical mode permits manipulations with images that are used for such objects as a robot and a container. We can distinguish simple objects associated with one symbol (with one unique code) and composed objects associated with more than one symbol (with a group of unique codes). The same symbol (graphical primitive) can be included in different objects. Operations (like "*robot move left*") define potential manipulations with the relevant objects. Conditions (like "*robot left sensor*") are formed through checking coordinates of the object image in the VGA memory. Thus, customizable symbol memory (see Fig. 3) makes it possible to create images for objects and operations with the VGA memory (see Fig. 3) enable logic conditions (representing sensors) to be generated. A complete example of this type is considered in [5].

At the moment, just images for relatively simple objects have been created and tested. They can be manipulated in two dimensions. This is because the primary objective is to verify FPGA-based circuits that control such equipment whose functionality can be verified visually. Note that the environment does not provide support for complex 3D graphical images such as that might be needed for simulation of six degree of freedom articulated robot arm. This is because the primary objective was to verify circuits inside FPGA but not to create sophisticated support for graphics that would undoubtedly consume considerable FPGA resources. If essential support for sophisticated graphics is indeed needed we would recommend either to use a dedicated to graphics separate FPGA or to apply software-hardware co-simulation described in [9].

There are numerous potential applications for the presented technique and we will describe just three of them from different application areas.

In [5] an automatic system for garage control is considered. The objective is to provide for automatic car parking in a garage. Fig. 7 demonstrates the basic ideas of visual simulation. In this case a car is a symbol. Motion of a car is controlled by an electronic circuit. Inputs of the circuit are supplied by sensors (such as that used for intelligent cruise control). Outputs of the circuit change locations (addresses) of the symbol (the car) in the VGA memory (see Fig. 3). An image of a car might also be composed of more than one symbol, which allows visualizing smoother motions. So, object primitives are symbols (matrices of pixels) and they are used for simulation instead of manipulating individual graphical pixels, which allows hardware resources needed for simulation in graphical mode to be reduced significantly. A number of parameters can be controlled during simulation, for example, the number of arriving cars per time slot, speeds of cars inside the garage, etc. Results of all necessary experiments (testing alternative

circuits, examining deadlocks, etc.) are represented visually in form of moving images of cars and this is very close to physical (real-time) functionality.

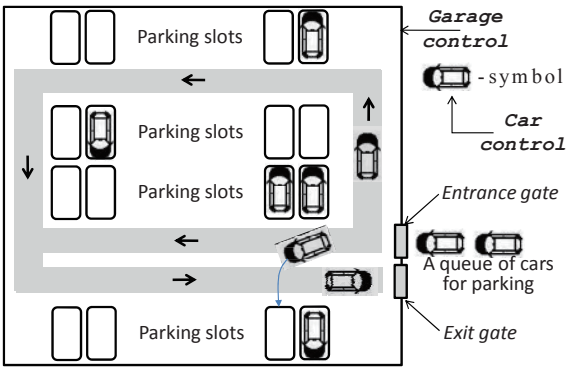


Fig. 7 Simulation of functionality of an automatic system for garage control

Similar simulation can also be implemented with autonomous electronic circuits and this is especially interesting for education. For example, an extended architecture of a customizable 8-bit FPGA-based processor [10] has been used within different disciplines for students of Aveiro University (Portugal). The results of simulation are presented in a general form shown in Fig. 8.

Let us consider an example of simulation. Suppose the following sequence of two instructions has to be executed:

SUBi, 64, 1,
STAi, 128, ADDR_LED,

The first instruction (SUBi, 64, 1,) reads data from RAM (64_{10} is the base address of RAM) using offset 1; subtracts the data from a processor accumulator; and stores the result in the accumulator. The second instruction (STAi, 128, ADDR_LED,) displays the value of the accumulator on LEDs (128_{10} is the base address of input/output devices and ADDR_LED is an offset address for the LEDs). During simulation each instruction will be sequentially displayed in a window dedicated to "*Running Assembly Language Instruction*". The selected RAM address for the first instruction will be shown as 1. We can see the value of RAM at the address 1 and how this value is copied to the accumulator. Finally, we can see which LEDs are *on* and *off*. The working processor clock frequency can be changed during run time and this gives very convenient opportunities for visual debugging.

The last example is dedicated to verification of advanced finite state machines (FSM), such as hierarchical FSM, recursive hierarchical FSM, parallel FSM and FSM with dynamically alterable functionality (reprogrammable FSM). Fig. 9 gives an example. A recursive algorithm (Fig. 9, a) for data sorting in a binary tree (Fig. 9, b) was taken from [11], where all necessary details can be found. The tree is coded in RAM as it is shown in Fig. 9, c, and explained in detail in [11].

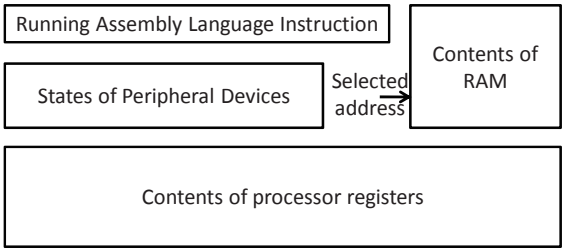


Fig. 8 Simulation of functionality of an 8-bit processor

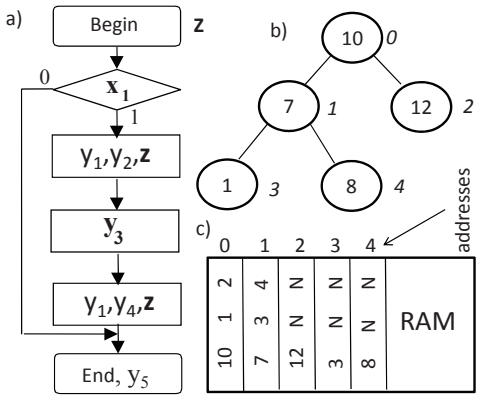


Fig. 9 Recursive sorting algorithm (a); a binary tree for sorting (b); coding of the binary tree in memory (c)

As we can see in Fig. 9, a, the algorithm (module Z) calls itself, i.e. it is recursive. The algorithm is implemented in a VHDL template for a recursive hierarchical FSM [11], which uses stacks instead of a register memory. The results of simulation are presented much like it is shown in Fig. 8 (for example, the selected memory addresses and the currently active FSM states/modules are indicated). Clock frequency for FSM can be changed during execution time. For the example in Fig. 9, b, the following sequence can be examined: begin from RAM address 0; jump to RAM address 1; jump to RAM address 3; display the value 3 from RAM at the address 3; return back to RAM address 1; display the value 7 from RAM at the address 1; jump to RAM address 4; display the value 8 from RAM at the address 4; return back to RAM address 1; return back to RAM address 0; display the value 10 from RAM at the address 0; jump to RAM address 2; display the value 12 from RAM at the address 2; return to RAM address 0; end. The sequence 1, 7, 8, 10, 12 represents the sorted values. In case of any error it can easily be detected.

4. IMPLEMENTATION DETAILS

All customizable functional blocks of Fig. 3 have been implemented [6] using Xilinx ISE and tested in three prototyping boards: DETIUA-S3 [7], NEXYS2 [8] and Celoxica RC10 (information about some of former Celoxica products is available through [2]).

The customizable symbol memory in Fig. 3 permits to store symbol images that are used as moving objects during visual simulation. Each image is coded by a matrix composed of VGA pixels. The size of matrices might be adjusted in accordance with the necessary requirements (complexity of images, etc.).

The developed virtual environment was tested for the system shown in Fig. 1, for traffic control, micro-robot competition, remotely controlled processor with variable instruction set, and an automatic system [5] (these systems were examined within different projects developed and implemented by students).

The results of evaluation of the proposed technique and experiments with the environment can be summarized as follows:

- The technique is applicable to a number of real-world problems when a visual mode of simulation is more appropriate;
- The technique is especially useful in pedagogical practice. The considered visual simulation was successfully used by students within different disciplines (see [12] for details).
- Additional resources required for the considered technique depend on the complexity of the problem. The circuit shown in Fig. 3 without an attached application-specific block consumes about 12% of xc3s500e FPGA available on the prototyping board [8], which is quite appropriate for simulation purposes.

Comparing the proposed technique with software/hardware co-simulation (see examples in [3,9]) permits to list the following advantages and disadvantages:

- Simulation just in hardware requires quite a long time for synthesis and implementation of FPGA-based circuits with built-in simulation tools. This disadvantage is not completely eliminated in case of software/hardware co-simulation;
- Supplementary software tools needed for simulation are easier to implement and test;
- Specific interfacing circuits are involved for software/hardware co-simulation and the relevant interaction requires longer time;
- The main advantage of simulation just in hardware is the simplicity to satisfy numerous real-time constraints;
- Software/hardware co-simulation makes sense when the modeled application-specific block is partially implemented in hardware and partially in software;
- Hardware simulation is preferable when we would like to verify an application-specific block completely implemented in hardware and when there exist hard real-time constraints.

5. CONCLUSION

The paper describes a virtual environment for prototyping and experiments with FPGA-based circuits. The basic architectural components of the environment establish communication with peripheral devices (such as a VGA monitor and a keyboard) that are used for visualizing the results and human interaction. There are two available modes for simulations that permit to display the results either in form of input/output messages or through moving graphical images imitating physical objects. Customization of object images is achieved through reloading of a symbol memory. Interactions with inputs (*e.g.* sensors) and outputs (*e.g.* actuators) are provided through periodic update of a monitor memory and verification of location addresses in the memory for object images. Examples from different application areas are presented and examined. The results demonstrate practicability and usefulness of the proposed technique, which is especially recommended for educational and experimental purposes.

REFERENCES

[1] P.J. Ashenden, *Digital Design. An Embedded System Approach Using VHDL*, Morgan Kaufmann, 2008, 573 pp.

[2] Available at: "http://agilityds.com/support/product-descriptions/DK5-SPD.pdf".

[3] V. Sklyarov, "Hardware/Software Modeling of FPGA-based Systems", *Parallel Algorithms and Applications*, vol. 17, No. 1, 2002, pp. 19-39.

[4] V. Sklyarov, I. Skliarova, "Design and Implementation of Parallel Hierarchical Finite State Machines", Proc. of the 2nd Int. Conf. on Communications and Electronics – HUT-ICCE 2008, Hoi An, Vietnam, June 2008, pp. 33-38.

[5] V. Sklyarov, I. Skliarova, A. Neves, "Modeling and Implementation of Automatic System for Garage Control", Proc. of ICCAS-SICE, Fukuoka, Japan, August 2009.

[6] Available at: "http://sweet.ua.pt/~a16360".

[7] M. Almeida, B. Pimentel, V. Sklyarov, I. Skliarova, "Design Tools for Rapid Prototyping of Embedded Controllers", Proc. of the 3rd Int. Conf. on Autonomous Robots and Agents - ICARA'2006, Palmerston North, New Zealand, Dec. 2006, pp. 683-688.

[8] Available at: "http://www.digilentinc.com/Products/Detail.cfm?NavTop=2&NavSub=451&Prod=NEXYS2".

[9] V. Sklyarov, I. Skliarova, B. Pimentel, M. Almeida, "Multimedia Tools and Architectures for Hardware/Software Co-Simulation of Reconfigurable Systems", Proc. of the 21st Int. Conf. on VLSI Design, Hyderabad, India, January 2008, pp. 85-90.

[10] Customizable 8-Bit FPGA Processor. Available at: "http://www-md.e-technik.uni-rostock.de/lehre/vlsi_i/proc8/index.html".

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

[12] Available at: "http://www.ieeta.pt/~skl".