Reuse Technique in Hardware Design

Valery Sklyarov, Iouliia Skliarova
University of Aveiro, Department of Electronics, Telecommunications and Informatics, IEETA
3810-193 Aveiro, Portugal
skl@det.ua.pt, iouliia@det.ua.pt

Abstract

The paper presents a technique for the design of digital systems on the basis of reusable hardware templates, which are circuits with modifiable functionality that might be customized to satisfy requirements of target applications, such as a highly optimized implementation of the selected problem-specific operations. Reusability is provided at two levels that are the level of hardware circuits and the level of specifications. The paper demonstrates how hardware templates can be modeled in software and implemented in hardware.

1. Introduction

Tremendous progress in the scope of field-programmable gate array (FPGA) technology has made it possible to advance configurable microchips from simple gate arrays that appeared on the market in the mid-1980s to multi-platform FPGAs containing more than 10 million system gates and targeted to the design of very complicated engineering systems. Today, the way to evolve higher performance computing from a general-purpose computer, proposed more than 45 years ago [1], has been finally implemented in reality. The market for FPGAs and other programmable logic devices is expected to grow from $3.2 billion in 2005 to $6.7 billion in 2010 [2]. An analysis presented in [3] clearly demonstrates that the largest FPGA consumers will be in engineering with numerous applications in the scope of electronic system design, from glue logic to high-complexity application specific (ASIC-type) devices. Pioneering products such as Xilinx's Virtex [4] or Altera's Stratix [5] FPGA families will find their main applications in the development of high-volume products.

Developing engineering systems on the basis of high capacity FPGAs puts forward a fundamental question – how to cope with rapidly growing complexity, or, in other words, how to use efficiently enormous and continuously rising hardware resources in the design process? This is actually very important because according to the Moore's law [6] every two years the density of microelectronic devices is generally doubled permitting at the same time better performance, and improved features. Fig. 1 demonstrates the relevancy of this law to static memory, Xilinx FPGAs [7] and Intel processors [6]. As we can see from Fig. 1 chip complexity grows exponentially with time. But what is important is that the number of available transistors grows faster than the ability to meaningfully design with them. This situation is a well known design productivity gap, which was inherited by FPGA from ASIC and which is increasing continuously. Therefore the design productivity will be the real challenge for future systems.

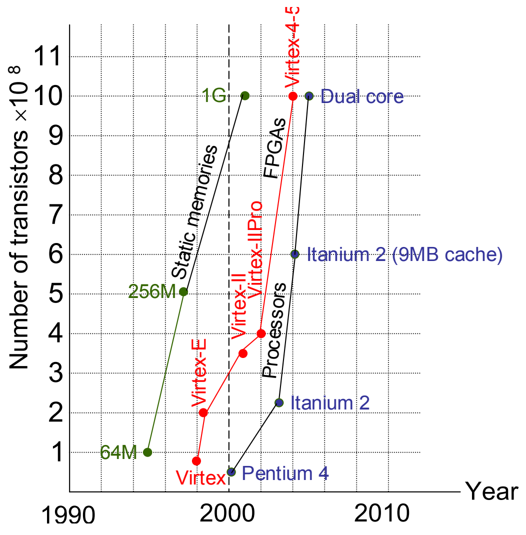

Fig. 1. Relevancy of the Moore's law to different microelectronics products

One possible answer to the above question is to apply a reuse technique and an evolutionary strategy in such a way that permits parameterizable and highly optimized project components to be involved once again in future projects. Thus, the existing experience might be powerfully integrated in the future products.

The majority of available synthesis tools permit to construct circuits, which, in general, cannot be reused. Any desired modification to the circuit functionality requires repeating the synthesis from the beginning, including debugging, testing and other steps that are considered as a part of the design process. This procedure can be significantly simplified with the aid of field-programmable technology (such as that is implemented in FPGAs) but in any case it requires additional time and high-experienced human resources. The paper shows that for some practical applications the considered problem can be alleviated. It might be achieved for digital systems that can be decomposed in a reusable core (which is extendable in general case) and a reprogrammable control unit(s). The latter can be modeled by a reprogrammable finite state machine (RFSM). The proposed design method is based on reusable hardware templates (HT). The HT is a circuit with predefined structure that has already been implemented in hardware (for example, in FPGA). The basic components of the circuit are RAM-blocks, and by reprogramming them we can implement different functionality. This might be done with the aid of an additional removable component of the designed system called reconfiguration controller, which can be used, in particular, just for debugging purposes. It makes possible to evaluate various alternatives to the functionality of the developed system avoiding many traditional steps of digital design that are time and resource consuming.

The remainder of the paper is organized in five sections. Section 2 makes an overview of available design specification methods and discusses the future of reconfigurable systems. Section 3 is devoted to hardware templates and their potentialities. Section 4 describes the proposed reuse technique for hardware design. The conclusion is given in Section 5.

2. The present and the future of reconfigurable systems

Designers of FPGA-based systems must wade through several layers of design before programming the actual device. The typical FPGA flow includes the following major phases, which are design entry, synthesis, mapping, placement and routing, FPGA programming, and verification. The latter can occur at different levels such as behavioral simulation, functional simulation, static timing analysis, post-layout timing simulation and, finally, in-circuit verification. If we focus our attention on the design entry, four different specification methods can be envisioned, which are schematic entry, hardware description languages, system-level specification languages and, finally, general-purpose programming languages.

The schematic-based approach is nowadays not very appropriate for specifying the functionality of modern systems because instead of thinking in terms of algorithms and data structures it forces the designer to deal directly with the hardware components and their interconnections. Contrariwise, the hardware description languages - HDLs (such as VHDL and Verilog) are widely used for design specification since they typically include facilities for describing structure and functionality at a number of levels, from the more abstract algorithmic level down to the gate level.

Recently, commercial tools allowing digital circuits to be synthesized from system-level specification languages (SLSL), such as Handel-C and SystemC, have appeared on the market. This fact allows the designer to work at a very high level of abstraction, virtually without worrying about how the underlying computations are executed. Consequently, even computer engineers with a limited knowledge of the targeted FPGA architecture are capable of producing rapidly functional, algorithmically optimized designs.

Even higher level of abstraction is achieved with general-purpose programming languages (GPPL), such as C++ or Java. During the last year commercial tools (for instance, Catapult Synthesis from Mentor Graphics and CoDeveloper from Impulse) started appearing on the market allowing the respective high-level descriptions to be transformed automatically to an HDL, which is further used for synthesis. In this case the code portions that can be executed in parallel are identified automatically by the design tools.

Besides of the mentioned design specification methods there exist other opportunities such as vendor libraries, graphical finite state machine editors, parameterizable IP (intellectual property) cores and so on.

In Fig. 2 different design specification methods are assessed according to such criteria as performance, FPGA resource usage, portability, ease to learn, ease to change and maintenance and, finally, the development time (in the first five groups of vertical bars the higher is the better, and for the last group the lower is the bar the better is the respective method).

From the graph we can see that schematic-based approach allows circuits with very good performance and efficient resource usage to be created. However, when we speak about portability and ease to learn, change and maintenance and the associated development time, schematic entry is an obvious outsider. As it was noticed in a recent EE Times survey “The days of designing FPGA with schematics are gone.” [8].

Hardware description languages are currently the golden mean of the design entry methods. They allow creating high-performance circuits, optimized from the resource usage point of view, the development time is not so long and providing changes to the design is not very difficult. The only weak point is that HDLs are not so easy to learn.

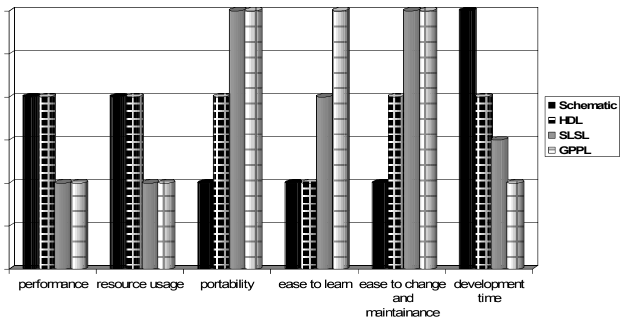

Fig. 2. Comparison of specification methods

Obviously, system-level and high-level languages possess the best portability and the highest level of abstraction. Of course, the higher level of abstraction leads to some performance degradation and not very efficient resource usage. On the other hand SLSLs and GPPLs have many advantages such as ease to learn, ease to change and maintenance, and a very short development time. Therefore, we can expect that as the tools responsible for generating hardware (more specifically, either an EDIF – electronic design interchange format file or an HDL file) from high-level source code advance, the SLSLs and GPPLs may become the predominant hardware description methodology, in the same way as general-purpose high-level programming languages have already supplanted microprocessor assembly languages.

In order to increase the design productivity the following challenges must be met. First of all, design reuse must be encouraged. Reusable high-level functional blocks (such as IP blocks) offer great potential for productivity gains because design effort for the reused logic is only a portion of the effort needed for newly designed logic. According to ITRS, it is expected that reuse rate for system-level design will increase from 32% in 2005 to 55% in 2020 [9].

Second, reuse techniques have to be incorporated in the synthesis process. In other words, as soon as we want to apply synthesis tools to construct a new circuit, these tools have to predict how different components of the circuit can be reused in the future products. Note, that if reuse techniques applied at the level of entire blocks (such as IP cores and design libraries) are widely employed, the same techniques at the level of synthesis process are not developed at all. The paper is intended to present some results in the scope of synthesis reuse and to demonstrate some benefits that can be obtained.

The third point is that design abstraction levels must be raised. Higher levels of abstraction allow many forms of verification to be performed much earlier in the design process, reducing lead-time to market and lowering cost by discovering problems earlier [10]. We have already mentioned that tools are currently emerging allowing for hardware design at a very high level of abstraction. Thus synthesis reuse techniques have to be adapted to different levels.

And finally, the fourth point is to increase the level of automation which inevitably will allow the number of design iterations to be reduced. In case of platform-based design, further improvements in automated software/hardware partitioning tools are strongly required.

Now it is clear, that reconfigurability will certainly be the key aspect of future systems since it will be needed for fault tolerance, for example, in molecular-scale systems, and for development of adaptive and self-correcting or self-repairing circuits. In addition, reconfigurability itself increases reuse, since existing devices can be reprogrammed to fulfill new tasks. According to ITRS forecast more and more SOC functionality will become reconfigurable [10].

3. Hardware templates

In general, hardware templates can be seen as reusable blocks at the circuit level. FPGAs are microelectronic devices that can be reused in such a way that the same microchip can implement different types of functionality. This is achieved through uploading a reconfiguration bitstream to FPGA's static memory, which customizes the FPGA hardware for the required project. On the contrary, the HT is a customizable circuit that has already been implemented in an FPGA. In other words the FPGA has been already configured but it still can be reconfigured through reprogramming some blocks of the HT. This permits, in particular, to reduce reconfiguration time significantly and to keep all predefined timing constraints in hardware circuits.

Thus, on the one hand any HT is an application-specific circuit but on the other hand it can be customized enabling us to implement different functionality, which is provided in such a way that basic characteristics of the circuit (such as the longest path delay) are optimized.

A general architecture of HT depends on the selected application-specific area. For example, templates might be constructed for different operations over binary and ternary vectors, for reprogrammable interfaces between digital circuits, for specific computational algorithms, etc.

Suppose we have to solve some tasks from a given area. For example, we want to construct a template for circuits that implement different types of interfaces. Note that the basic operations for such type of applications are very similar but on the other hand each particular interface might be unique. In order to model a template and to implement it in hardware the following technique has been employed. Initially the considered problem is studied in detail and the basic HT architecture is proposed. This architecture has to be able to implement all the desired functionalities (all the required interfaces for our example). On the other hand it has to be built in such a way that it can be optimized for any particular functionality that is needed for the considered device. In other words, we would like to be able to personalize and to optimize this architecture for any customized circuit that belongs to the selected application-specific area. At the second step, the HT has to be implemented and tested in software. This allows to validate its correctness and to analyze its effectiveness for solving various particular tasks. At the next step, the template has to be implemented and verified in hardware. Finally, we can use the HT for solving problems from the selected area.

It is very important that there exists a personalization of HT implementations from “more targeted to software specifications” to “more targeted to hardware specifications”. Thus, relationships between different levels of hardware specifications considered in the previous sections can easily be established. Indeed, mutual interdependence of the levels enables the designers to achieve a number of advantages, such as the following:

- Specification of the HT can be done at any desired level, i.e. general-purpose language, system-level specification language and hardware description language.
- Since there exists software implementation of HT, all the required configurations can be generated, debugged and tested in general-purpose computer and prepared for future uploading to the relevant hardware.
- There exists an opportunity of run-time reconfiguration (reprogramming HT blocks), which is a very challenging feature for virtual adaptive systems.

In order to provide a more compact software HT model we will use an object-oriented description proposed in [11] with the aid of C++ classes (see Fig. 3). The top-level specification includes just two classes that are an abstract class *datapath_template* and a concrete class *FSM_template*. Note that the datapath in Fig. 3 can also include a similar HT of a lower level.

It is assumed that the problem can be solved with the aid of the function *Solve*, which executes an algorithm described by the argument *FSM_template&*. An interaction between the classes *datapath_template* and *FSM_template* has been organized through the functions *Run* in such a way that the function *FSM_template::Run(X)* takes arguments presented in the input vector X and returns outputs in form of vector Y; the function *datapath_template::Run(Y)* takes arguments presented in the vector Y and returns outputs in form of vector X.

The class *datapath_template* describes a general structure of hardware that can be used for solving an application-specific subset of tasks. The functions *Solve* and *Run* are pure virtual functions because we intend to model just a general structure for a subset of similar tasks and we do not have yet any particular implementation. The latter can be described by a concrete class derived from the class *datapath_template* in such a way that the considered above general structure is either reused without any change or extended. In order to solve different problems with the aid of the same datapath (execution unit) we have to be able to change the sequence of operations that is generated by control circuits, i.e. we have to alter the behavior of the respective RFSM(s). It is important to provide such modifications not only for software model (which is flexible by definition), but also for the relevant hardware model. The latter is proposed in [11] and synthesis of RFSM can be done with the aid of methods [12]. In general, an RFSM is composed of two blocks (see fig. 3): a RAM-based core with customizable behavior, and a controller, which permits to configure the core in such a way that enables us to implement the desired particular behavior.

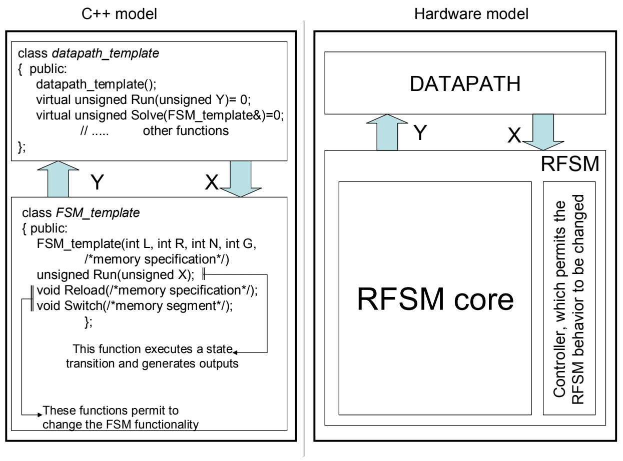
```
class datapath_template
{  public:
     datapath_template();
     virtual unsigned Run(unsigned Y)= 0;
     virtual unsigned Solve(FSM_template&)=0;
               // .....        other functions
};
```
```
class FSM_template
{ public:
    FSM_template(int L, int R, int N, int G,
               /*memory specification*/)
    unsigned Run(unsigned X);
    void Reload(/*memory specification*/);
    void Switch(/*memory segment*/);
               };
```
C++ model
Y
X
This function executes a state transition and generates outputs
These functions permit to change the FSM functionality
Hardware model
DATAPATH
Y
X
RFSM
RFSM core
Controller, which permits the RFSM behavior to be changed

Fig. 3. Relationship between software and hardware models

The functions *Run* and *Solve* can properly be described after we study a specific class of applications. For example, let us examine combinatorial search algorithms formulated over binary and ternary matrices, such as matrix covering, the Boolean satisfiability, graph coloring, etc. It is known that the execution unit (datapath) for such algorithms can be constructed in such a way that it will be common for different problems [13]. Customization of operations for the concrete problem can be done through modification of control algorithms implemented in the respective control unit (which is the RFSM). All the required details and examples of such customization are presented in [13]. The same technique can be applied to other problems, such as sorting and ordering algorithms [14], data compression/ decompression algorithms [15], etc. Note that for the design of reprogrammable control units the existing methods [12] can directly be used. The specification of control algorithms can be provided with the aid of a graphical language called hierarchical graph schemes (HGS) [16], which supports modular and hierarchical mechanisms implemented in the respective model of finite state machine (FSM), called hierarchical FSM (HFSM). For a number of practical applications HGS can easily be constructed from specifications in general-purpose languages, such as C/C++ [12]. Any HGS provides support for: modularity (each individual HGS is considered to be a module that, in general, can be reused); hierarchy and parallelism. It is known that HGS can be formally translated to the respective control circuits modeled by HFSM [16] and described by a hardware template [14]. The latter is composed of two concurrently executing VHDL processes. The first process describes two stacks that are needed to keep codes of modules (*M_stack*) and states (*FSM_stack*) in each module. The second one specifies functionality of combinational circuit which defines transitions between the modules and the states. The distinctive feature of HFSMs (comparing with traditional FSMs) is the ability to provide state transitions at the following two levels:

- At the first level the code of the top register of *M_stack* enables the process to recognize the module.
- At the second level the code of the top register of *FSM_stack* and values of external input variables (from the set $X = \{x_1,\ldots,x_L\}$) enable the process to recognize the proper state transition, i.e. to form the code of the next state (and/or) the code of the next module. Obviously, the states within different modules might have the same codes and all the states within the same module must have different codes.

The relevant synthesis and implementation details are described in [14] with examples of VHDL specifications for practical problems.

It is important to note that all VHDL constructions [14] can easily be replaced with equivalent constructions for system-level specification languages, such as Handel-C, and they are reusable and synthesizable, i.e. they can be automatically converted to the respective hardware circuits. Thus, the same functionality can easily be modeled in any language, i.e. general-purpose languages, system-level specification languages and hardware description languages. This is important because different specifications can be used within the same project (see section 2). As distinct from HTs for execution units, the considered HTs for control units can be seen as blocks allowing reusability at the specification level. Combining capabilities of reconfigurable and hierarchical FSMs we can construct such a control unit which integrates reusability of hardware circuits with reusability of algorithms. Indeed, the combinational circuit of an HFSM can be built from RAM blocks in such a way that the method [12] can be applied without any change. Finally, the same circuit is able to implement different functionalities. Using the template [14] we can construct a reconfigurable hierarchical FSM (RHFSM), which supports reusability at the level of algorithms (sub-algorithms) much like as for software development (i.e., for example, much like reusability of functions in C/C++ language). Thus, we can construct control circuits allowing to change their behavior based on RFSM model and to assemble algorithms from previously designed and tested algorithmic blocks with the aid of HFSM model.

4. Reusability in hardware design

Fig. 4 summarizes the proposed reuse technique in hardware design. At the topmost level, the circuit is considered to be a decomposition of an execution unit (datapath) and a control unit communicating through pre-defined reusable interface.

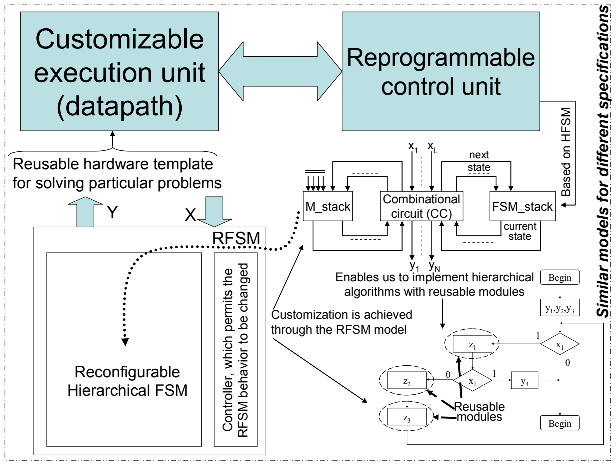
Customizable execution unit (datapath)
Reprogrammable control unit
Based on HFSM
Similar models for different specifications
Reusable hardware template for solving particular problems
Y
X
RFSM
Reconfigurable Hierarchical FSM
Controller, which permits the RFSM behavior to be changed
M_stack
Combinational circuit (CC)
FSM_stack
x_1
x_L
next state
current state
y_1
y_N
Enables us to implement hierarchical algorithms with reusable modules
Customization is achieved through the RFSM model
Begin
y_1,y_2,y_3
x_1
1
0
z_1
z_2
z_3
y_4
Reusable modules
Begin

Fig. 4. Summary of the proposed reuse technique for hardware design

The following methods have to be applied at the level of synthesis:

1. The execution unit is made customizable for a selected group of closely related problems. The respective customization methods are known and they have already been considered for different hardware circuits, such as combinatorial accelerators [13], etc. (many examples can be found in [13-15]).
2. The reprogrammable control circuit is considered to be an RHFSM with customizable behavior, which:
    a) Permits to use the same hardware for different algorithms;
    b) Can invoke reusable modules.
3. The RHFSM is constructed on the basis of the hardware template for HFSM [12,14].

4. Customization of RHFSM functionality is achieved through reloading its RAM blocks with the aid of methods [12].
5. Reusability of modules can be provided applying the same technique which is widely employed in software engineering.
6. The considered models can be retargeted to any desired level of specification, such as general-purpose languages, system-level specification languages and hardware description languages. Thus, any design flow discussed in section 2 can be supported.
7. Software models enable the designers to use general-purpose computers to generate customization bitstreams and to verify the projects.

The following list summarizes advantages of the proposed reuse technique for hardware design:

- It permits to realize highly optimized and carefully tested design templates for vast varieties of similar problems.
- The designed circuits can be made adaptable to different external conditions. Indeed, dependently on external events the circuit can be properly modified, selecting an adequate module.
- The designed circuits can be made virtual in such a way that we can implement functionality in FPGA that does not possess sufficient hardware resources. This can be achieved through proper sequential reconfiguration, which is allowed for the considered models.

5. Conclusion

The paper describes a reuse technique that can be applied to the design of FPGA-based, application-specific digital systems. Reusability is achieved at the following two levels. First, the considered circuits (hardware templates) can be constructed in such a way that allows the same hardware to be employed for different types of closely related functionalities. Thus, reusability at the level of hardware circuits is provided. Second, the proposed specification and implementation methods (namely HGSs and HFSMs) enable the description fragments (or modules) to be employed in such a way that the developed algorithm is composed of novel and previously designed modules. Thus, reusability at the level of specifications is provided. Finally, the integrated reuse technique is proposed.

6. References

[1] G. Estrin, “Organization of Computer Systems – The Fixed Plus Variable Structure Computer”, in Proc. Western Joint Computer Conf., New York, 1960, pp. 33-40.

[2] EE Times (2006, May 30), “Semiconductor FPGA/PLD market to grow 14% in '06” [Online]. Available: http://www.eetasia.com/ART_8800419580_499485_0e42dd84200605.HTM.

[3] EE Times (2006, May 24), “FPGA Market Will Reach $2.75 Billion by Decade's End” [Online]. Available: http://www.us.design-reuse.com/news/news13441.html.

[4] Virtex-5 LX Platform Overview (May, 2006) [Online]. Available: http://direct.xilinx.com/bvdocs/publications/ds100.pdf.

[5] Altera Product Catalog. October 2006. Available at: http://www.altera.co.jp/literature/sg/product-catalog.pdf.

[6] Available at: http://www.intel.com/technology/mooreslaw/index.htm.

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

[8] EE Times (2006, July 31), R. Goering, “FPGA users rank challenges, tasks”.

[9] International Technology Roadmap for Semiconductors, System Drivers, 2005.

[10] International Technology Roadmap for Semiconductors, Design, 2005.

[11] V.Sklyarov, I.Skliarova. “Design of Digital Circuits on the Basis of Hardware Templates”. Proceedings of International Conference on Embedded Systems and Applications – ESA’03, Las Vegas, USA, CSREA Press, June, 2003, pp. 56-62.

[12] V.Sklyarov, “Reconfigurable models of finite state machines and their implementation in FPGAs”, Journal of Systems Architecture, 2002, 47, pp. 1043-1064.

[13] I. Skliarova, V. Sklyarov, "Design Methods for FPGA-based Implementation of Combinatorial Search Algorithms", Proc. Int. Workshop on SoC and MCSoC Design - IWSOC'2006, 4th Int. Conference on Advances in Mobile Computing and Multimedia - MoMM'2006, Yogyakarta, Indonesia, December 2006, pp. 359-368.

[14] V.Sklyarov, “FPGA-based implementation of recursive algorithms”, Microprocessors and Microsystems, Special Issue on FPGAs: Applications and Designs, 2004, 28/5-6, pp. 197-211.

[15] V. Sklyarov, I. Skliarova, B. Pimentel, "Using Compression/Decompression Technique for FPGA Targeted Matrix-Oriented SAT Solvers", Proc. XX Conference on Design of Circuits and Integrated Systems - DCIS2005, Lisbon, Portugal, November 2005.

[16] V.Sklyarov, “Hierarchical Finite-State Machines and their Use for Digital Control”, IEEE Trans. on VLSI Systems, 1999, vol. 7, no 2, pp. 222-228.