ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 1, No. 1, 2016

SELF-CONFIGURABLE FPGA-BASED COMPUTER SYSTEMS: BASICS AND PROOF OF CONCEPT

Viktor Melnyk

Lviv Polytechnic National University, 12, Bandera Str., Lviv, 79013, Ukraine
Author e-mails: *viktor.a.melnyk@gmail.com*

Submitted on 30.11.2015

Abstract: **Computer systems performance is today improved with two major approaches: general-purpose computer computing power increase (creation of multicore processors, multiprocessor computer systems, super-computers), and adaptation of the computer hardware to the executed algorithm (class of algorithms). The last approach often provides application of hardware accelerators – ASIC-based and FPGA-based, also named reconfigurable, and is characterized by better performance / power consumption ratio and lower cost as compared to the general-purpose computers of equivalent performance. However, such systems have typical problems. The ASIC-based accelerators: 1) they are effective only for certain classes of algorithms; 2) for effective application there is a need to adapt algorithms and software. The FPGA-based accelerators and reconfigurable computer systems (that use FPGAs as a processing units): 1) the need in the process of writing a program to perform computing tasks balancing among the general-purpose computer and FPGA; 2) the need of designing application-specific processors soft-cores; and 3) they are effective only for certain classes of problems, for which application-specific processors soft-cores were previously developed.**

This paper covers the scope of questions regarding concept of design, architecture, and proof of concept of the Self-Configurable FPGA-Based Computer Systems – an emerging type of high-performance computer systems, which are deprived of specified challenges. The method of information processing in reconfigurable computer systems and its improvements that allow an information processing efficiency to increase are shown. These improvements are used as a base for creating a new type of high-performance computer systems with reconfigurable logic, which are named self-configurable ones, and a new method of information processing in these systems. The structure of self-configurable FPGA-based computer system, the rules of application of computer software and hardware means necessary for these systems implementation are described. Major processes on the stages of program loading and execution in the self-configurable computer system are studied, and their durational characteristics are determined. On the basis of these characteristics, the expressions for evaluating the program execution duration in the self-configurable computer system are obtained. The directions for further works are discussed.

Key words: **field programmable gate arrays, high performance computing, reconfigurable architectures, reconfigurable logic, self-configurable computer system.**

I. INTRODUCTION

Today one of the most promising areas of activity in the field of high performance computing is creation of the reconfigurable computer systems (RCCS). RCCSs compete with other types of high-performance computer systems due to high speed characteristics of modern field-programmable gate arrays (FPGAs) – hardware base of reconfigurable computing environment (RCE) of RCCS, and due to advances in design technology of application-specific processors to be synthesized in RCE of RCCS.

Co-functioning of computer system based on general-purpose processors with application-specific processors synthesized in RCE, the structure of which considers executed algorithms features, allows one to increase its overall performance by 2-3 orders of magnitude. Reconfigurability and ability to synthesize an application-specific processor (ASP) with a new structure and functionality in RCE allows one to change the functional commitment of created thereby RCCS with preserving its high performance at the new class of problems.

Along with high performance, ensured by the RCCS, there are also problems associated with their application. These particularly are significant timing expenses for task distribution between the processing units, often lack of IP cores required for implementation in the RCE, what forces to develop them from scratch, and high additional requirements to RCCS users' qualification, as they, beside modeling and programming, must perform system analysis, design ASP's architecture, perform their synthesis and implementation in RCE.

Deprived of the above mentioned problems are self-configurable computer systems (SCCS), where these labor-intensive and time-consuming tasks are fully automated and replaced from the operator to the

computer system. Taking into account the necessity of further improvement of computer means performance and extension of reconfigurable devices application in computer systems, further development in the field of self-configurable computer systems is a topical task of scientific research and engineering.

II. RELATED WORK

The concept of the self-configurable FPGA-based computer systems design, the method of information processing in them, and its structure are proposed in the paper [1]. The problem of labor intensive and time-consuming tasks, which is characteristic of RCCS, is removed in SCCS owing to the change of the information processing method in it.

The software means that should be used in SCCS as its components are available today. In this regard we may consider an approach to the development of computational load balancing systems between general-purpose computer and hardware accelerator proposed in [2]. The IP cores' generators [3], [4], IP cores' libraries [5], and system level design tools and solutions are available on the market that could be used for ASP design on the basis of its algorithm description in the high-level programming language [6–8].

III. PROBLEM STATEMENT

For the purpose of verification of the SCCS design concept main principles and in order to prove its fundamental advantages over the RCCS, the task arises to investigate organization and architecture of SCCS as well as compositional and durational characteristics of main processes during information processing in SCCS, and that is a goal of the study carried out in this paper.

IV. RECONFIGURABLE COMPUTER SYSTEMS: PROBLEMS AND SOLUTIONS

A. Method of Information Processing in Reconfigurable Computer System

As a starting point of our research, we shall consider a method of information processing in the reconfigurable computer system.

Information processing in RCCS can be represented as sequential execution of the four stages. At the first stage the user creates the program P_{in} written in the high-level programming language, divides this program into the subprogram P_{GPC} of computer and the subprogram P_{RCE} of RCE, performs compilation of subprogram P_{GPC}, generates its executable file *obj* and stores it in the computer memory. In order to perform these actions, the following tools should be used:

1. In order to develop and debug the program written in the high-level programming language – integrated programming environments, such as Visual C + + [9] from Microsoft.

2. In order to balance the computational load between computer and RCE – the modeling tools or profilers, which provide the statistics of the program, including time and frequency of execution for its separate fragments. This enables detection in the program of the fragments that require the largest amount of computations.

3. In order to compile the subprogram of computer and create its executable file – an arbitrary compiler from the language that subprogram is represented in into the object code that can be directly executed by the computer.

At the second stage the user develops (or uses a ready-made solution) an HDL-model *ASPM* of ASP, which is intended to perform the subprogram P_{RCE} of RCE, performs logical synthesis of the ASP and loads configuration files $\mathbf{conf} = \left\{conf_q,\ q = \overline{1...K_{FPGA}}\right\}$ to RCE, where K_{FPGA} is the number of FPGAs, which form RCE, and, thus, creates an ASP in RCE. In order to perform these actions, the following tools should be used:

1. In order to design and debug HDL-models of ASPs – integrated environments for design, modeling and verification of the projects described in the hardware description languages, for example, ModelSIM from Mentor Graphics, Active-HDL from Aldec.

2. In order to perform logical synthesis of ASPs and RCE configuring – tools for hardware design in FPGA, for example, ISE, Alliance, Foundation from Xilinx, Quartus II, Max + II from Altera.

At the third stage, after program initialization, operating system loads the executable file *obj* of the computer subprogram to its main memory using the standard loader.

At the fourth stage RCCS executes program P_{in}. The computer executes its own subprogram P_{GPC}, RCE executes its own subprogram P_{REC}. The computer interacts with ASP synthesized in RCE under the control of the operating system. In order to organize this interaction, the driver of RCE and hardware parts of the controllers of interfaces, which this interaction is carried out through, should be used. These tools are provided by manufacturers together with RCE, for example, by DRC for their reconfigurable processor units RPU [10], by Nallatech for their reconfigurable accelerators of H100 series [11], by Celoxica for their reconfigurable accelerators RCHTX [12].

If the same program must be re-executed, the user sequentially performs the third and the fourth stages.

If the user has an executable file and configuration files, then he/she loads configuration files to RCE and further sequentially performs the third and the fourth stages.

An execution time for information processing in RCCS according to a given program can be represented by the expression:

$$T_{RCCS} = {}_U t^P_{distr} + t^{GPC}_{compile} + t^{obj}_{store} + {}_U t^{ASPM}_{develop} + \\ + {}_U t^{ASPM}_{synth} + {}_U t^{conf}_{load} + t^{obj}_{load} + t^P_{exe} \quad , \tag{1}$$

where: ${}_U t^P_{distr}$ is duration of program distribution by the user on the computer subprogram P_{GPC} and RCE subprogram P_{RCE}; $t^{GPC}_{compile}$ is duration of the computer subprogram compiling; t^{obj}_{store} is duration of computer subprogram executable file storing into its memory; ${}_U t^{ASPM}_{develop}$ is duration of ASP HDL-model designing; ${}_U t^{ASPM}_{synth}$ is duration of ASP logical synthesis; ${}_U t^{conf}_{load}$ is duration of configuration files loading into RCE; t^{obj}_{load} is duration of the computer subprogram executable files loading into its main memory; t^P_{exe} is duration of program P_{in} executing in RCCS. Index ${}_U*$ marks actions performed directly by the user.

The duration of information processing in the case of re-execution of the program can be represented by the expression:

$$T'_{RCCS} = t^{obj}_{load} + t^P_{exe}. \tag{2}$$

If the computer subprogram executable file and RCE configuration files are available, the duration of information processing can be represented by the expression:

$$T''_{RCCS} = {}_U t^{conf}_{load} + t^{obj}_{load} + t^P_{exe}. \tag{3}$$

B. Problems Hindering Efficiency of the Reconfigurable Computer Systems

By analyzing the above-described method of information processing in RCCS, one can outline problems that significantly impede the improvement of its efficiency, namely:

– in order to execute each new program in RCCS, all four stages, in the first two of which all actions are performed by the user, have to be sequentially performed, which requires significant amount of time;

– in order to execute the program when executable file and configuration files are available, the user has to load configuration files into RCE before the third and the fourth stages, which also requires significant amount of time;

– the list of algorithm, that RCCS is effective on, is narrow, and depends on the functional characteristics of implemented in RCE ASPs, and provided the problems to be solved must be changed, at least, the steps of the second stage of the information processing method described above have to be performed;

– the complexity of information processing is high, because the user, beside modeling and programming, should also perform system analysis, design the ASPs architecture, perform their logical synthesis and implementation in FPGAs.

C. Ways to Improve Reconfigurable Computer Systems Efficiency

A key approach to solve the above problems is automation of all stages of information processing in RCCS. Let us analyze information processing in RCCS.

The development of the ASP's HDL-model at the second stage requires from the user a significant amount of time and knowledge of the system-level design technology. However, as mentioned above, today the software tools are available allowing automatic creation of the ASP's HDL-models from high-level description of the algorithm to be implemented in. These software tools transform the algorithm described in the high-level programming language into the HDL-model of ASP. By linking the operations of ASP's HDL-model generation, ASP's logical synthesis and configuring of RCE in automatically executable sequence, the computer system can load the configuration codes into RCE automatically without the user's intrusion.

It should be noted that the use of generation tools imposes condition of availability of high-level algorithm description to be implemented in ASP. The user creates this description at the first stage when dividing the input program into two subprograms, and this also requires from the user a significant amount of time. Automation of load balancing, beside reduction of time, will allow one:

1. to link operations of load balancing and compilation of computer subprogram in one startup sequence, and as a result, to obtain subprogram executable file without the user's intrusion;
2. to link operations of load balancing, generation of ASP HDL-model, ASP's logical synthesis and RCE configuration in one startup sequence, and as result, to load configuration codes into RCE automatically without the user's intrusion.

Consequently, automation of steps that are performed on the first two stages of information processing method in RCCS, i.e. automatic obtaining of computer subprogram executable file and automatic creation and loading of ASP configuration files into RCE for RCE subprogram execution, will allow one:

1. To reduce the execution time for information processing;
2. To reduce the complexity of information processing since the user no longer has to perform the systems analysis, design ASPs architecture and ASPs logical synthesis.

However, automation of the two first stages execution does not solve another problem mentioned above – namely, the list of algorithms that RCCS is effective on remains narrow and depends on the functional characteristics of ASPs implemented in RCE. Their change requires, at least, repeating the second stage's steps of the above-mentioned method of information processing, which should be done by the user.

This problem can be solved by improving the method of information processing in RCCS in the way that loading into RCE of obtained after logical synthesis configuration files is carried out not by the user but by the operating system, and not at the second stage but at the third one, in parallel with loading the computer subprogram executable file into its main memory after program initialization. This implies that configuration files should be stored in the computer memory after logical synthesis. Thus, because the configuration files are formed automatically in parallel with computer subprogram executable file and stored in its memory, the entire sequence of actions from the beginning of the load balancing up to obtaining the executable file and the configuration files should be treated as a single stage of program compiling.

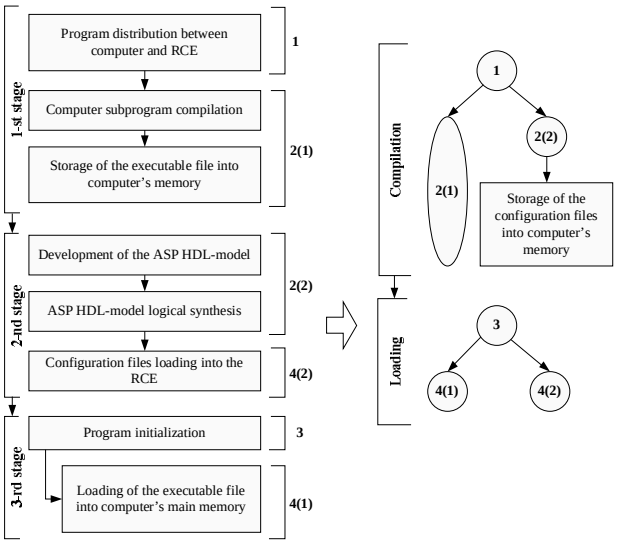

Fig. 1. The diagram showing how to improve information processing in the RCCS

The proposed improvements of the method of information processing in RCCS are shown in the diagram presented in Fig. 1. As a result of those improvements of the information processing method in RCCS, and automation of its first two stages, all the problems pointed above will be solved, because:

1. All actions, starting from the load balancing and till obtaining the executable file and the configuration files, are executed automatically at the stage of program compilation without the user's intrusion.

2. In the case of executable file and configuration files availability, loading these files into the computer main memory and into RCE is respectively performed by the operating system after program initialization. Thereby the task of expanding the list of algorithms that RCCS are effective on is solved.

3. The requirements to the system user experience are simplified up to knowing the high-level programming language.

V. SELF-CONFIGURABLE COMPUTER SYSTEM AND METHOD OF INFORMATION PROCESSING IN IT

We suggest naming the self-configurable the computer system with reconfigurable logic, where program compilation includes automatically performed actions of creating configuration, and which acquires that configuration automatically in the time of program loading for execution.

In the Self-Configurable Computer System (SCCS), automated is execution of: 1) computational load balancing between the general-purpose computer and RCE, and 2) creation of ASP's programming model, and the method of information processing is improved in the way that loading into RCE of configuration files obtained after logical synthesis is carried out not by the user but by the operating system in parallel with loading the computer subprogram executable file into its main memory after the program initialization [1].

From a user's perspective, SCCS operates similarly to the traditional general-purpose computer, as the user, in accordance with the proposed method of information processing, a) develops a program written in the high-level language and submits it into SCCS for compilation; b) initiates a program after compilation; c) loads the data to be processed and receives the results. Thus, SCCS reconfigures itself according to the features of the computational algorithm described by the computer program, unlike it is done in RCCS, where these actions are performed by the user.

The diagram of the method of information processing in SCCS is shown in Fig. 2. This method can be represented as a sequential execution of three stages: program compilation, its loading and execution.

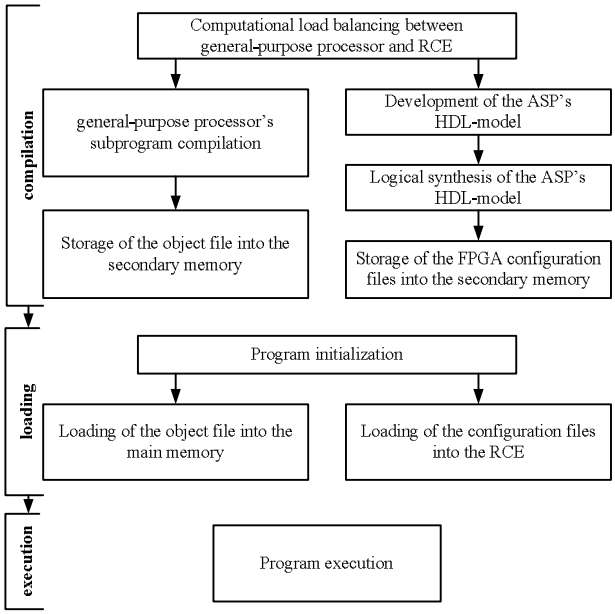

Fig. 2. The diagram of the method of information processing in the SCCS

According to the mentioned method, the user creates a program P_{in} written in the high-level programming language and submits it into SCCS. During compilation, SCCS automatically performs the following actions: divides this program into the computer subprogram P_{GPC} and the RCE subprogram P_{RCE}, performs computer subprogram P_{GPC} compilation, generates its executable file *obj*, creates ASP's HDL-model *ASPM* to perform RCE subprogram P_{RCE}, performs ASP's logical synthesis, and stores in the computer memory the obtained executable file *obj* and the configuration files of RCE $\mathbf{conf} = \{conf_q, q = \overline{1...K_{FPGA}}\}$, where K_{FPGA} is the number of FPGAs that form RCE.

In order to perform these actions, the SCCS has to contain the following means:

1. Computational load balancing system for load balancing between computer and RCE. This system should automatically select from the program P_{in} fragments, execution of which in RCE reduces its execution time, and divide the program P_{in} into the computer subprogram P_{GPC}, replacing selected fragments in it by instructions for interaction with RCE, and on RCE subprogram P_{RCE}, formed from the selected fragments. One implementation of such system is described in [2]. This system creates the RCE subprogram in the x86 assembly language, thus it must be supported by the means for the assembly language code translation into the high-level language to be used in SCCS. The tool of this type is available on the market, for example Relogix Assembler-to-C Translator [13] from MicroAPL.

2. Generating system for the ASP HDL-model creation, which should automatically generate a model *ASPM* from the RCE subprogram P_{RCE}, like Chameleon system from Intron [7], [14], Agility Compiler [15] and DK4 Design Suite [16] from Celoxica, CoDeveloper from Impulse [17].

3. The tools that are used in RCCS for performing computer subprogram compilation and creation of its executable file, and for logical synthesis of the ASPs.

At the stage of program loading after its initialization SCCS loads the executable file *obj* of the computer subprogram into its main memory using standard loader and, at the same time, loads the configuration files $\mathbf{conf} = \{conf_q, q = \overline{1...K_{FPGA}}\}$ into RCE and thus creates an ASP in there. Then, the stage of the program execution is performed in the same way as in the RCCS. In order to perform these actions, the same tools can be used in SCCS as in RCCS.

The structure of the self-configurable computer system that implements the proposed method of information processing is shown in Fig. 3.

An execution time for information processing according to a given program P_{in} in SCCS can be represented as a sum of durations of three stages: compilation $T_{SCCS}^{COMPILE}$, loading T_{SCCS}^{LOAD} and execution T_{SCCS}^{EXE}, by the following expression:

$$T_{SCCS}(P_{in}) = T_{SCCS}^{COMPILE}(P_{in}) + T_{SCCS}^{LOAD}(P_{in}) + T_{SCCS}^{EXE}(P_{in}), \quad (4)$$

where:

$$T_{SCCS}^{COMPILE}(P_{in}) = t_{distr}^{P} + \max\begin{pmatrix} \left(t_{compile}^{GPC} + t_{store}^{obj}\right), \\ \left(t_{generate}^{ASPM} + t_{synth}^{ASPM} + t_{store}^{conf}\right) \end{pmatrix}, \quad (5)$$

$$T_{SCCS}^{LOAD}(P_{in}) = \max\left(t_{load}^{conf}, t_{load}^{obj}\right), \quad (6)$$

and $T_{SCCS}^{EXE}(P_{in})$ is the same as in RCCS and is equal to t_{exe}^{P}. Therefore:

$$\begin{aligned} T_{SCCS} = {} & t_{distr}^{P} + \\ & + \max\left(t_{compile}^{GPC} + t_{store}^{obj}, t_{generate}^{ASPM} + t_{synth}^{ASPM} + t_{store}^{conf}\right) + \\ & + \max\left(t_{load}^{conf}, t_{load}^{obj}\right) + t_{exe}^{P} \end{aligned} \quad (7)$$

where, in relation to the expression (1), the value t_{store}^{conf} is the duration of the configuration files storing in the computer memory is added.

As can be seen from the expression (4), significant reduction of the execution time for information processing in SCCS versus RCCS is ensured by:

– much less duration of automatic execution of four actions presented in expressions (1) and (2) as compared to that of their execution by the user, i.e. $t_{distr}^{P} <<_{U} t_{distr}^{P}$, $t_{generate}^{ASPM} <<_{U} t_{develop}^{ASPM}$, $t_{synth}^{ASPM} <<_{U} t_{synth}^{ASPM}$, i $t_{load}^{conf} <<_{U} t_{load}^{conf}$;

– parallel execution of several actions, namely, 1) sequence of computer subprogram compilation and storage of executable file into secondary memory with sequence on the ASP model generation, its synthesis and storage of binary configuration file into secondary memory, and 2) loading of executable file into the main memory and loading of the configuration file into RCE.

In two cases, namely, at the program re-execution, and in the presence of pre-formed executable file and configuration files, the duration of information processing in SCCS will be the same and can be represented by the expression:

$$T'_{SCCS} = \max\left(t_{load}^{conf}, t_{load}^{obj}\right) + t_{exe}^{P}. \quad (8)$$

In comparison with the expression (2) one may note that in case of repeated execution of the same program, if $t_{load}^{conf} > t_{load}^{obj}$, the duration of information processing in SCCS versus RCCS may increase. Therefore, it is worth exploring in more details these two processes and, if necessary, develop the solution that will ensure the least possible value of t_{load}^{conf}.

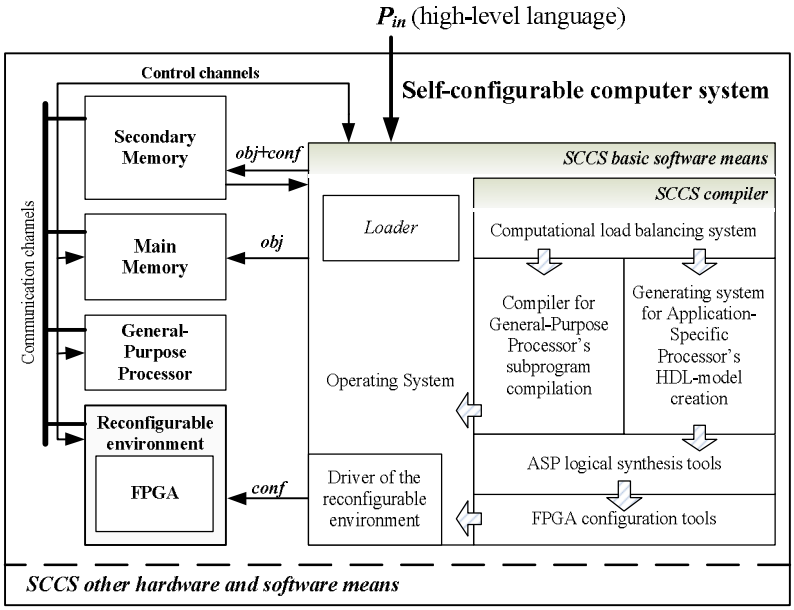

Fig. 3. The structure of the self-configurable computer system

VI. INVESTIGATION OF DURATION OF COMPUTER SUBPROGRAM EXECUTABLE FILE LOADING INTO MAIN MEMORY

When the computer subprogram executable file is loading the system tasks are being executed by the operating system as well as the file transfer from the computer external to main memory. The list of actions of that process depends on the type and features of a particular operating system, but, in general, is as follows:

determining the location of the program executable file in the secondary memory;

1) reading the header of the program executable file out to the buffer of the operating system loader;

2) determining required memory (RAM, stack, dynamic) resources for program execution and creation of the main process and environment variables for the application;

3) code segments of executable file loading into appropriate addresses in the main memory;

4) data segments of executable file loading and deployment into appropriate addresses in the main memory;

5) determining the location of required libraries, their loading or their physical addresses determination, and their placement in the main memory (only for programs with DLL);

6) the processor registers initialization and the start of process of execution.

So, duration of the computer subprogram executable file loading depends on a type of operating system, type of executable file and its size Q_{obj}, and on the computer technical characteristics, namely: the rate of data reading from the secondary memory V_{read}^{EM}, the rate of data writing V_{write}^{MM} into the main memory V_{write}^{MM}, throughput V^{int} of the interface between the secondary and main memories. If we denote duration of the system tasks execution associated with organization of the executable file loading as $tSYS_{load}^{obj}$, then the duration of the computer subprogram executable file loading can be expressed as:

$$t_{load}^{obj} = tSYS_{load}^{obj} + \frac{Q_{obj}}{\min\left(V_{read}^{EM}, V^{\text{int}}, V_{write}^{MM}\right)}. \qquad (9)$$

In the most modern computers, the secondary memory is implemented as a hard drive on the magnetic disks. The rate of data reading from the secondary memory V_{read}^{EM} today achieves 210 MB/s. For instance, hard drive WD1000DHTZ with capacity 1 TB from Western Digital got this index equal to 209.14 MB/s, Barracuda 7200.14 with capacities 1 TB and 2 TB from Seagate – more then 205 MB/s [18], [19].

Hard drives are attached to computer with standard peripheral interfaces, the most common among which today is 3-rd generation SATA. The throughput V^{int} of this interface is up to 6 GB/s [20].

However, the performance of the RAM chips main memory is built on is much higher. One of leading memory chips manufacturers – Micron Technology – today offers dynamic memory DDR4 with following characteristics [21]: capacity 4 GB, frequency 933/1067 MHz (clock cycle duration is 1.07/0.938 ns), throughput – 14928/17064 MB/s.

Thus, because $V_{read}^{EM} < V^{\text{int}} < V_{write}^{MM}$, the expression (9) can be transformed as follows:

$$t_{load}^{obj} = tSYS_{load}^{obj} + \frac{Q_{obj}}{V_{read}^{EM}}. \qquad (10)$$

VII. INVESTIGATION OF DURATION OF CONFIGURATION FILE LOADING INTO RCE

D. Specific Features of Computer Interaction With Reconfigurable Environment in the SCCS

When investigating duration t_{load}^{conf} of the configuration file loading into the reconfigurable environment it is necessary to take into account the following specific features:

according to the method of information processing in SCCS, the compiled program (executable files and configuration files) is stored in the computer secondary memory;

FPGAs are the volatile devices, therefore configuration codes should be loaded in there each time they are powered on. In order to automate this task and, thus, shorten FPGA configuration time, the Flash memory chip (configuration memory) is attached to FPGA [22], [23], where the configuration codes are retained and automatically loaded into FPGA after turning power on;

since operations of data reading and writing in the configuration memory cannot coincide in time, prior to configuration loading into FPGA it should be loaded from the computer secondary memory to the configuration memory.

Therefore the configuration files loading into RCE can be expressed as sequentially executed actions of configuration loading into the configuration memory and then into FPGA, as its duration t_{load}^{conf} can be represented by the following expression:

$$t_{load}^{conf} = t_{load}^{conf}(ConfMem) + t_{load}^{conf}(FPGA), \quad (11)$$

where $t_{load}^{conf}(ConfMem)$ is duration of configuration codes loading into the configuration memory; $t_{load}^{conf}(FPGA)$ is duration of configuration codes loading into the FPGA.

Now let us estimate the terms of the expression (11).

E. Duration of Configuration Codes Loading Into the FPGA

According to the analysis performed, duration $t_{load}^{conf}(FPGA)$ of configuration codes loading into FPGA depends on the FPGA type, amount of its resources, its synchronization frequency in configuring mode, and the width of interface that configuration codes are loaded through. It can be determined from following expression [24]:

$$t_{load}^{conf}(FPGA) = \frac{Q_{conf}}{F_{conf}^{FPGA} \cdot W_{conf}^{FPGA}}, \quad (12)$$

where Q_{conf} is configuration size (in bits); F_{conf}^{FPGA} is the FPGA synchronization frequency in the configuring mode; W_{conf}^{FPGA} is the width of the FPGA interface, that configuration codes are loaded through.

The values of parameters F_{conf}^{FPGA} and W_{conf}^{FPGA} depend on the configuration loading mode. The list and characteristics of the modes for different types of FPGAs may vary. For instance, for the Xilinx Spartan-3 devices configuring two basic modes are used – a bit-wide one via the Serial Peripheral Interface (SPI) and a byte-wide one via the Byte-wide Peripheral Interface (BPI), each of them having several options for FPGA connection to the configuration memory [25]. For Xilinx Virtex-6 FPGAs configuring eight basic modes and five different interfaces are used. In addition to the bit-wide SPI and Serial interfaces, 8– and 16-bit BPI and 8-, 16-, and 32-bit SelectMAP interfaces are supported [26].

If RCE contains more then one FPGA chip, then configuration codes may be loaded sequentially (***daisy-chain mode***) from one configuration memory, or in parallel, what requires separate Flash memory for each FPGA chip.

Synchronization signal used to synchronize configuration loading, is generated by the FPGA internal frequency generator. The frequency of this signal during configuration loading is specified in the configuration file with the relevant directives. For instance, for Xilinx Spartan-3 devices the frequency is specified with the *ConfigRate* directive, and it is equal to 1 (by default), 3, 6, 12, 25, and 50, for some devices even 80 MHz [27]. The allowed frequency is determined by the frequency of data writing / reading from the configurations memory.

Configuration codes represent the binary “image” of FPGA that determine the state of all its configurable components and are usually stored in one file. The volume Q_{conf} of the configuration codes depends on the FPGA resources and does not depend on the resources that synthesized device occupies in it, like, for instance, the volume of configuration codes for the synthesis of inverter and MPEG processor in the same FPGA will be the same. For the Spartan-3 FPGA family from the Xilinx volume of configuration ranges from 437, 312 bits for XC3S50A/AN up to 13, 271, 936 bits for XC3S5000 [25], for Virtex-6 family – from 26, 239, 328 bits for XC6VLX75T up to 184, 823, 072 bits for XC6VLX760 [26].

F. Duration of Configuration Codes Loading into the Configuration Memory

The process of configuration files loading into the configuration memory from the secondary memory might be represented as a sequence of the following tasks: 1) determining location of the configuration file in the secondary memory and its reading out; 2) configuration file transfer from the secondary memory into the RCE; 3) configuration file loading into the configuration memory. These actions might combine in time, thus, duration $t_{load}^{conf}(ConfMem)$ of the configuration file loading into the configuration memory can be determined from the expression:

$$t_{load}^{conf}(ConfMem) = \frac{Q_{conf}}{\min\left(V_{read}^{EM}, V^{\text{int}}, F_{conf}^{CM} \cdot W_{conf}^{CM}\right)}, \quad (13)$$

where F_{conf}^{CM} is the frequency of the configuration codes loading into the configuration memory; W_{conf}^{CM} is the width of the configuration memory interface used for configuration loading.

As the analysis of the works [22], [23] has shown, configuration memory is directly attached to FPGA, while the permissible rate of configuration codes loading into FPGA is determined according to the frequency of data writing/reading from the configuration memory. Thus, we can assume that $F_{conf}^{CM} = F_{conf}^{FPGA}$, $W_{conf}^{CM} = W_{conf}^{FPGA}$.

Analyzing the above characteristics, we may conclude that $V_{read}^{EM} \approx F_{conf}^{CM} \cdot W_{conf}^{CM} < V^{\text{int}}$ and, thus, $t_{load}^{conf}(ConfMem) \quad t_{load}^{conf}(FPGA)$.

G. Ways to Reduce Duration of Configuration Loading into RCE in SCCS

When analyzing the method of information processing in RCCS [28], we may note that the use of attached to FPGA configuration memory there for configuration codes retaining allows one to shorten and simplify information processing if computer subprogram executable file and RCE configuration file are available. This is achieved because on a second stage of the method the user should not load configuration files into RCE as they are loaded automatically just after the power switch-on in RCCS.

However, since in the SCCS configuration codes loading is performed after program initialization, and the compiled program is already stored in computer's memory, the need in configuration memory no longer exists. Moreover, as the analysis carried out above shows, the speed of data reading from the external memory reached in recent years is almost equal to that from the configuration memory. Regarding the above mentioned and taking into account the fact that $F_{conf}^{CM} = F_{conf}^{FPGA}$, $W_{conf}^{CM} = W_{conf}^{FPGA}$, and, thus, $V_{read}^{EM} \approx F_{conf}^{FPGA} \cdot W_{conf}^{FPGA} < V^{\text{int}}$, equation (11) for determining the duration t_{load}^{conf} of the configuration codes loading into the RCE can be transformed into the following:

$$t_{load}^{conf} = \frac{Q_{conf}}{\min\left(V_{read}^{EM}, F_{conf}^{FPGA} \cdot W_{conf}^{FPGA}\right)}. \quad (14)$$

Further duration reduction of configuration loading into FPGA is possible through architectural improvements of SCCS, firstly, by introducing into its structure the special memory for configuration files storing for all compiled programs, that is optimized for fast interaction with RCE. In order to reduce the configuration codes volume, the compression is used e.g. in RLE, LZSS, Huffman and other methods, which in some cases provides reduction of duration of configuration loading by 50-60 % [29], however for some FPGAs it requires lowering of frequency of the configuration codes loading, particularly for Spartan-3 and Spartan-3E FPGAs – down to 20 MHz [25].

VIII. EVALUATION OF PROGRAM LOADING STAGE DURATION IN SCCS

For quantitative evaluation of duration of the computer subprogram executable file loading the experimental studies on three computing platforms have been performed:

1) **PLATFORM 1**:

processor: AMD Athlon II X4 645, MMX, 3DNow (4 CPUs), 3.1 GHz; main memory: 3326 MB; data read speed from the hard drive: 100 MB/s; operating system: Windows XP Professional (5.1, Build 2600) Service Pack 3.

2) **PLATFORM 2**:

processor: Mobile AMD Sempron 3400+, MMX, 3DNow, 1.8 GHz; main memory: 2032 MB; data read speed from the hard drive: 50 MB/s; operating system: Windows XP Professional (5.1, Build 2600) Service Pack 3.

3) **PLATFORM 3**:

processor: Intel Celeron 2.53 GHz; main memory: 760 MB; data read speed from the hard drive: 80 MB/s; operating system: Windows XP Professional (5.1, Build 2600) Service Pack 3.

For each of these platforms the duration of data reading from the external memory was calculated. The results of experimental studies and calculations are shown in charts (Fig. 4), where the solid line labels the plots of executable files loading duration, and dotted line – those of reading duration from the external memory.

The above estimations have been performed with the use of AppTimer (free software tool) from PassMark Software [30]. As seen from the plots, the duration of loading of the executable files that have size from hundreds of KB up to tens of MB (the vast majority of software executable files have that size) ranges from a few to several hundred milliseconds, and a part of this time is taken by the system tasks associated with their loading.

On the basis of the results of the above analysis the calculations of duration of the configuration codes loading to the FPGA have been performed. The results of these calculations are shown in the plots (Fig. 5). As seen from the plots, for modern FPGAs from leading manufacturer the value of duration of the configuration code loading lies in the range from a few to several hundred milliseconds.

As a result of our studies we may conclude that the values of duration of computer subprogram executable files loading and those of configuration files loading are

of the same order, i.e. $t_{load}^{conf} \approx t_{load}^{obj}$, and the difference between these values does not fundamentally affect the SCCS performance. This confirms the effectiveness of the principles underlying the method of information processing in SCCS, since the FPGA configuring in SCCS does not slow down program loading stage as compared to the duration of this stage in RCCS.

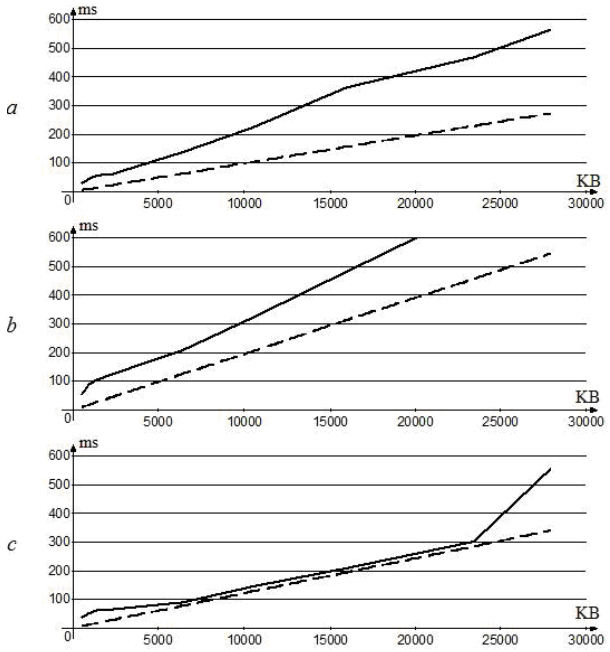

Fig. 4. Dependences of the computer subprogram executable files loading duration on their size for platforms 1 (a), 2 (b) and 3 (c)

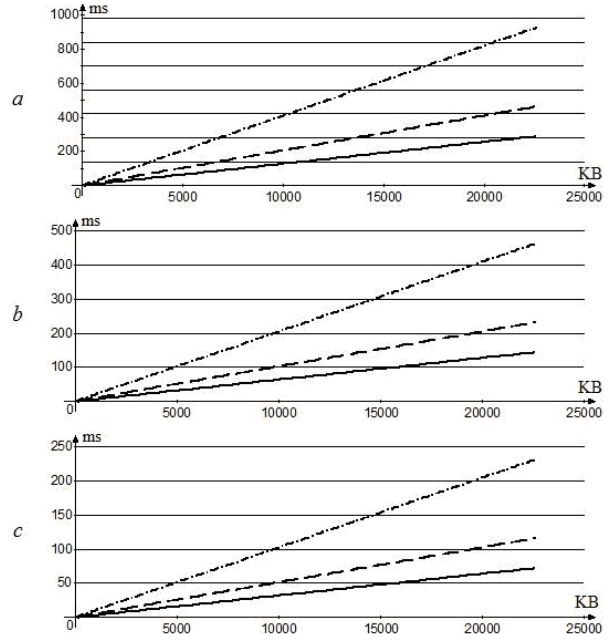

Fig. 5. Dependences of the configuration files loading duration on their size for interfaces with the widths of 8 (a), 16 (b) and 32 (c) bits

IX. INVESTIGATION OF MAIN PROCESSES DURATION ON THE PROGRAM EXECUTION STAGE

According to the method of information processing in SCCS [1], the stage of program execution in SCCS is the same as in RCCS. Denoting the duration of the computer subprogram P_{GPC} execution by the general-purpose processor GPP as $T_{GPP}(P_{GPC})$, and duration of given by subprogram P_{RCE} algorithm execution by the application-specific processor ASP as $T_{ASP}(P_{RCE})$, we can determine the duration $T_{SCCS}^{EXE}(P_{in})$ of the program P_{in} execution stage from the following expression:

$$T_{SCCS}^{EXE}(P_{in}) = T_{GPP}(P_{GPC}) + T_{ASP}(P_{RCE}) - \\ - T_{GPP,\,ASP}^{par} + ttr_{data}, \quad (15)$$

where ttr_{data} is the duration of the data transfer between the GPP and ASP processors during program P_{in} execution; $T_{GPP,\,ASP}^{par}$ is the time during which the GPP and ASP processors perform calculations in parallel.

Analyzing expression (15), one may conclude that in order to get as least as possible value of duration $T_{SCCS}^{EXE}(P_{in})$ of the program P_{in} execution in SCCS the following conditions must be fulfilled:

$$\begin{cases} \dfrac{T_{GPP}(P_{GPC}) + T_{ASP}(P_{RCE})}{T_{GPP}(P_{in})} = j\,T \to \min; \\ \min\left(T_{GPP}(P_{GPC}), T_{ASP}(P_{RCE})\right) - T_{GPP,\,ASP}^{par} = dT \to \min; \\ ttr_{data} \to \min. \end{cases} \quad (16)$$

where $T_{GPP}(P_{in})$ is the duration of the program P_{in} execution by a general-purpose processor.

Let us analyze these conditions.

In order to ensure the least possible value of the relation $j\,T$ the subprogram P_{RCE} should contain such parts of the program P_{in} that accommodate most computational load (are characterized by the highest computational complexity) and execution of which requires the greatest amount of GPP resources.

The value of difference dT depends primarily on the spatial characteristics of the algorithm described by the program P_{in} (the presence of the parts in it that can be executed in parallel). In order to ensure as least as possible value of the difference dT these characteristics must be taken into account during the program P_{in} distribution to the subprograms P_{GPC} and P_{RCE}.

The duration of the data transfer between the GPP and ASP processors depends on the amount N_{data} of the data to be transferred and on the interface characteristics: its throughput V^{int}, delay I_d, and reaction time I_{rt}:

$$ttr_{data} = f\left(N_{data}, V^{\text{int}}, I_d, I_{rt}\right), \quad (17)$$

moreover, time delays I_d and I_{rt} occur each time of transfer initialization. Therefore, the data exchange between the subprograms P_{GPC} and P_{RCE} should be organized in the way that ensures the least possible amounts of the interaction sessions and the data to be transferred.

The fulfillment of the above conditions is realized in SCCS during the distribution of the computational load between the computer and the reconfigurable environment, and should be taken into account when developing the conceptual foundations of the computational load balancing system.

As can also be noted form the expression (15), the duration $T_{SCCS}^{EXE}\left(P_{in}\right)$ of the program P_{in} execution stage in SCCS depends on the value $T_{ASP}\left(P_{RCE}\right)$, and, hence, on the performance of the application-specific processor ASP. Therefore, to implement this processor, 1) appropriate architectural approaches should be taken – hardware orientation and parallel data processing, and 2) maximum efficiency of the FPGA resources usage should be provided. These tasks are solved in SCCS during generation of the ASPs models, and should be taken into account when developing the conceptual foundations of generating system for ASP HDL-model creation.

X. FURTHER WORK DIRECTIONS ON SCCS CREATION

The organization, architecture and characteristics of SCCS were shown above in this paper. Therefore, we can highlight a list of further works on the SCCS creation. The following tasks should be included primarily into this list:

– selection of the generation system for creating the ASP's HDL-model and adapting it for use in SCCS;

– selection of the computational load balancing system and its adaptation for use in SCCS;

– development of the interaction models of SCCS software means according to the proposed method of information processing;

– trial operation of SCCS and its testing.

Implementation of these tasks will allow one to use all the potential provided by the self-configuration property and will ensure for SCCS one of the leading places among the most promising means of high-performance computing.

XI. CONCLUSIONS

In this paper, organization, architecture, and characteristics of the self-configurable FPGA-based computer systems are shown. As the starting point of our research we took the method of information processing in RCCS and the rules of application of the computer software and hardware means necessary for its implementation.

It has been identified that the main problems hindering the efficiency of RCCS are related to the fact that in order to implement each new program the four stages of information processing in it must be sequentially executed, two first of which being performed by the user, first of all, the computational load balancing and the design of ASPs that require significant amount of time.

The analysis performed has shown that the identified problems can be solved by automatic load balancing between the computer and RCE, by automatic synthesis of ASPs HDL-models, and by improving the method of information processing in such a way that loading the configuration files into RCE is carried out by the operating system instead of the user, and this is done simultaneously with the computer executable file loading into its main memory.

The above improvements have been taken as a basis of a new type of high-performance computer systems with reconfigurable logic named self-configurable ones and a new method of information processing in these systems. The structure of SCCS and the expressions for estimating the duration of information processing in it are shown. It has also been shown that durations of two concurrent processes, namely, the configuration file loading and the computer subprogram executable file loading, are critical in SCCS. Therefore, a detailed exploration of these two processes has been carried out in order to ensure SCCS effectiveness.

The main processes on the stage of program loading have been identified and their durational characteristics have been investigated. The characteristics of modern computer means have been analyzed, and calculations of duration of the computer subprogram executable file loading to its main memory, and configuration codes loading to the reconfigurable environment, have been performed. On the basis of the above calculations, as well as according to the results of experimental studies on a number of computing platforms, it has also been determined that durations of these processes are of the same order and the difference of these values does not fundamentally affect the performance of SCCS. This, hence, confirms the effectiveness of the principles underlying the method of information processing in SCCS, since the FPGA configuring in SCCS does not slow down the program loading stage as compared to the duration of this stage in RCCS. It has also been determined that in SCCS, unlike RCCS, there is no need for the configuration memory, and the removal of this memory makes it possible to reduce the time of configuration loading to the FPGA.

The main processes on the stage of program execution have been identified and their durational characteristics have been investigated. The conditions of shortening the program execution stage in SCCS have been determined. These conditions can be fulfilled in SCCS at the time of distribution of computational load between the computer and the reconfigurable environment, and at the time of generation of the ASP's HDL-model, and should be taken into account when developing the conceptual foundations of the SCCS basic components – the computational load balancing system and generation system for creation of the ASP's HDL-models.

The directions of further works on the SCCS creation, implementation of which will allow one to use all the potential provided by the property of self-configuration and ensure for SCCS one of the leading places among the most promising means of high-performance computing, have been discussed.

REFERENCES

[1] Melnyk, A., Melnyk, V., "Self-Configurable FPGA-Based Computer Systems" Advances in Electrical and Computer Engineering, vol. 13, no. 2, pp. 33–38, 2013, doi:10.4316/AECE.2013.02005. [Online]. Available: http://www.aece.ro/abstractplus.php?year=2013&number=2&article=5

[2] Melnyk, V., Stepanov, V., Sarajrech, Z., "System of load balancing between host computer and reconfigurable accelerator", Proceedings "Computer systems and components" of Tchernivtsi National University. – Tchernivtsi, 2012. – T. 3. Ed. 1. pp. 6–16.

[3] A Proven EDA Solutions Provider makes all the difference. [Online]. Available: http://www.aldec.com/en.

[4] Xilinx Core Generator. Xilinx Inc. [Online]. Available: http://www.xilinx.com/ise/products/coregen_overview.pdf – 2005.

[5] Melnyk, A, Melnyk, V. "Organization of libraries of standardized and custom IP Cores for high-performance hardware accelerators", Proceedings of IV-th all-Ukrainian conference "Computer Technologies: Science and Education", Ukraine, Lutsk, 9–11 October 2009. – P. 113–117.

[6] Genest, G. "Programming an FPGA-based Super Computer Using a C-to-VHDL Compiler: DIME-C", Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference, 5–8 Aug. 2007. – P. 280–286.

[7] Chameleon – the System-Level Design Solution. [Online]. Available: http://intron-innovations.com/?p=sld_chame.

[8] ANSI-C to VHDL Compiler. [Online]. Available: http://www.nallatech.com/FPGA-Development-Tools/dimetalk.html.

[9] Ivor Horton. Beginning Visual C++ 2005. – John Wiley & Sons, 2005. – 1224 p.

[10] DRC Computer Corporation. RPU100-L60 DRC Reconfigurable Processor Unit. A breakthrough in coprocessor technology. [Online]. Available: http://www.drccomputer.com/pdfs/DRC_RPU100_datasheet.pdf.

[11] H100 Series FPGA Application Accelerators. Version 1.9. September 2008. [Online]. Available: http://www.skyblue.de/nallatech/5595.pdf.

[12] Celoxica Ltd. RCHTX-XV4 High Performance Computing (HPC) Application Acceleration Board Datasheet. Version 1.0. 2006. [Online]. Available: http://www.hypertransport.org/docs/tech/rchtx_datasheet_screen.pdf.

[13] Relogix Assembler-to-C translator. [Online]. Available: http://www.microapl.co.uk/asm2c/.

[14] Melnyk, A., Salo, A., Klymenko, V., Tsyhylyk, L. "Chameleon – system for specialized processors high-level synthesis", Scientific-technical magazine of National Aerospace University "KhAI", Kharkiv, 2009. No. 5, pp. 189–195.

[15] Handel-C Language Reference Manual For DK Version 4. Celoxica Limited, 2005. – 348p.

[16] Agility Compiler for SystemC. Electronic System Level Behavioral Design & Synthesis Datasheet. 2005. [Online]. Available: http://www.europractice.rl.ac.uk/vendors/agility_compiler.pdf

[17] C-to-FPGA Tools form Impulse Accelerated Technologies. Impulse CoDeveloper C-to-FPGA Tools. [Online]. Available: http://www.impulseaccelerated.com/products_universal.htm

[18] StorageReview.com – Storage Reviews. [Online]. Available: http://www.storagereview.com.

[19] Read Throughput Maximum: h2benchw 3.16. [Online]. Available: http://www.tomshardware.com/charts/hdd-charts-2012/-02-Read-Throughput-Maximum-h2benchw-3.16, 2900.html.

[20] Whitepaper. New SATA Spec Will Double Data Transfer Rates to 6 Gbit/s. SATA-IO. May 27, 2009. [Online]. Available: http://www.sata-io.org/documents/SATA-6Gbs-Fast-Just-Got-Faster.pdf.

[21] DDR4 SDRAM – Micron Technology, Inc. DDR4–Packing Power and Performance into a New Generation. [Online]. Available: http://www.micron.com/products/dram/ddr4-sdram#fullPart&236=0.

[22] Platform Flash XL High-Density Configuration and Storage Device. Product Specification. DS617 (v3.0.1) January 07, 2010, – 88p. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds617.pdf.

[23] Platform Flash XL Configuration and Storage Device User Guide. UG438 (v2.0) December 14, 2009, – 74 p. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug438.pdf.

[24] Eric Crabill. Powering and Configuring Spartan-3 Generation FPGAs in Compliant PCI Applications. Application Note: Spartan-3 Generation Family. XAPP457 (v1.0) June 8, 2007, Xilinx, Inc. – 9 p. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp457.pdf.

[25] Spartan-3 Generation Configuration User Guide. Extended Spartan-3A, Spartan-3E, and Spartan-3 FPGA Families. UG332 (v1.6), October 26, 2009, Xilinx, Inc. – 352 p. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug332.pdf.

[26] Virtex-6 FPGA Configuration User Guide // UG360 (v3.5) September 11, 2012, Xilinx, Inc. – 182 p. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug360.pdf.

[27] Spartan-3E FPGA Family Data Sheet. Product Specification. DS312 October 29, 2012, Xilinx, Inc. – 182 p. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf.

[28] Melnyk, V. "Principles of construction and operation of reconfigurable computer systems", Scientific journal "Technical Sciences" of Khmelnytskyj National University, Ukraine. – No. 6. – 2012. – P. 212–217.

[29] Koch, D., Beckhoff, C., and Teich, J. "Bitstream decompression for high speed FPGA configuration from slow memories", Proceedings of International Conference on Field-Programmable Technology (ICFPT'07). IEEE, 2007. – P. 161–168.

[30] PassMark AppTimer – Measure application startup time. [Online]. Available: http://www.passmark.com/products/apptimer.htm.

Viktor Melnyk is a professor of the Department of Information Technologies Security in the Lviv Polytechnic National University, Ukraine. He was awarded the academic degrees of philosophy doctor in 2004, and doctor of technical sciences in 2013 in Lviv Polytechnic National University. He has scientific, academic and hands-on experience in the field of computers and computer systems research and design, proven contribution into IP Cores design methodology and high-performance reconfigurable computer systems design methodology. He is experienced in computer data protection, including cryptographic algorithms, cryptographic processors design and implementation, wireless sensor network security. Mr. Melnyk is an author of more than 70 scientific papers, patents and monographs.