Journal of Siberian Federal University. Engineering & Technologies 6 (2014 7) 674-684
~ ~ ~

УДК 004.315

High-Level Design Flows for VLSI Circuit

Oleg V. Nepomnyashchy*,
Alexander I. Legalov and Natalia J. Sirotinina
Siberian Federal University
79 Svobodny, Krasnoyarsk, 660041, Russia

Received 08.04.2014, received in revised form 15.06.2014, accepted 21.07.2014

Design flows for very-large-scale integration circuit are considered. The problems arising from the realization of the project using top-down and system design methods are highlighted. The technology of an architecture-independent design for computer systems on a chip is suggested. The technology is based on a functional data-flow presentation of the initial algorithms and their subsequent step-by-step transformation at the register transfer level into description of the system being designed. Using methods of formal description and verification allows developers to transfer initial high-level parallel algorithms on topologies of the configurable integrated circuits being developed.

Keywords: integrated circuit, high-level design, verification, algorithm, parallel data flows, program transformation, functional data-flow programming.

Маршруты высокоуровневого синтеза сверхбольших интегральных схем

О.В. Непомнящий,
А.И. Легалов, Н.Ю. Сиротинина
Сибирский федеральный университет
Россия, 660041, Красноярск, пр. Свободный, 79

Рассмотрены маршруты синтеза однокристальных вычислительных систем. Выделены проблемы, возникающие при создании проекта методами нисходящего и системного проектирования. Предложена технология архитектурно независимого проектирования вычислительных систем на кристалле. Технология базируется на функционально-потоковом представлении исходных алгоритмов и их последующем поэтапном преобразовании в описание проектируемой системы на уровне регистровых передач. Использование при этом методов формального описания и верификации позволяет разработчикам осуществлять перенос параллельных высокоуровневых исходных алгоритмов на топологии разрабатываемых конфигурируемых интегральных схем.

* Corresponding author E-mail address: 2955005@gmail.com

Ключевые слова: интегральная схема, высокоуровневое проектирование, верификация, алгоритм, параллельные потоки данных, преобразование программ, функционально-потоковое параллельное программирование.

1. Introduction

A design flow for very-large-scale integration circuit (VLSI) describes all the stages of creating the project from the concept to production of final products.

There are the following basic design flows [1]: a traditional top-down design flow, design flow based on preliminary abstraction of the project at the system level (ESL – Entire System Level) and highly specialized technologies of bottom-up design.

In the latter case, a number of companies, for example, Synopsys, offer for large projects a bottom-up design technology called Automated Chip Synthesis or RTL Budgeting (RTL – Register Transfer Level). This technology involves preliminary determination of temporary and other limitations for each project component.

Then register and gate structure are synthesized for each part of the project with constructive design to follow. Due to decomposition the duration of synthesis is by 5-10 times shorter compared to the duration of conventional top-down design [1].

2. Conventional design flow

The traditional three-level presentation (Fig. 1) of the VLSI structure identifies functional, structural and chip topology (geometry) levels. Each level of presentation can be subdivided into sub-levels. For example, the following sub-levels can be identified in a VLSI structure:

- the level of structural modules, including CPU, memory, ready-made module, etc;
- the register level, which defines register communication and describes small functional units;
- the level of logic gates;
- circuit (transistor) level.

The traditional design flow is based on the representation of the project on one of the hardware description languages (HDL) and goes down through the above mentioned levels simultaneously moving along the axis of hierarchy of other areas.

Within this approach, the terms of reference for design are developed first, which includes development of project specification packages that serve as the basis for the behavioral model development. Program codes on universal high-level programming languages (C/C++) and descriptions on algorithmic problem-oriented languages (eg, SystemC) are used as the initial specifications. Models of individual system components presented at the functional level (eg, using Matlab-Simulink) or ready-made software (SW) and hardware (HW) blocks on hardware description languages (VHDL/Verilog) may also be considered.

The next stage involves elaboration and creation of the project model at the functional-algorithmic level. At this stage the description for the algorithms of interaction between the system being designed and external environment is formed, which is followed by layout the high-level representation of SW/HW modules into a single computing system and its verification to form the structure of the system being designed.

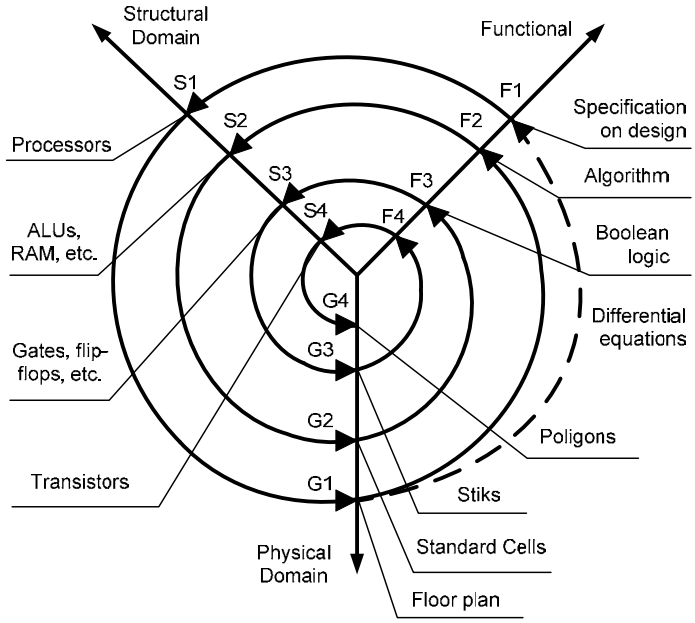

Fig. 1. Domains and levels of VLSI design models. Gajski-Kuhn chart (Y-chart) [1]

The systems used for assembling and verification allow carrying out simultaneous high-level debugging on different languages and joint logical modeling, for example, Active-HDL, ModelSim and Matlab/Simulink.

As a result the developers get a debugged structural description of the system at the processor-memory-switch (PMS) level.

The main advantages of the tools used at this stage are absolute abstraction from the final realization and possibility of prompt modification and modeling of the system being designed.

The main disadvantages include:

- lack of tools for describing parallelism;
- a semantic gap between the control of computations in the source program and its realization at the microcircuit level;
- impossibility to convert the initial presentation of the project into the presentation on register transfer level in the form of hardware description language program without full or partial, manual or semiautomatic redesign;
- need to purchase or develop your own libraries for your design environment for a specific target platform or chip.

The next stage is the development of a description of a functionally complete system in the form of a chip layout plan. In the past, the methods for description of the system on the functional logic level were highly specialized. The tools were either nonintegrated software products supplied to the market by third companies or highly specialized systems designed for internal use of specific developer or manufacturer. The automation systems, for example, Quartus by Altera or Xilinx ISE by Xilinx, currently used at this stage of design are integrated and include all the tools required for the effective project management starting from the design of the system structure.

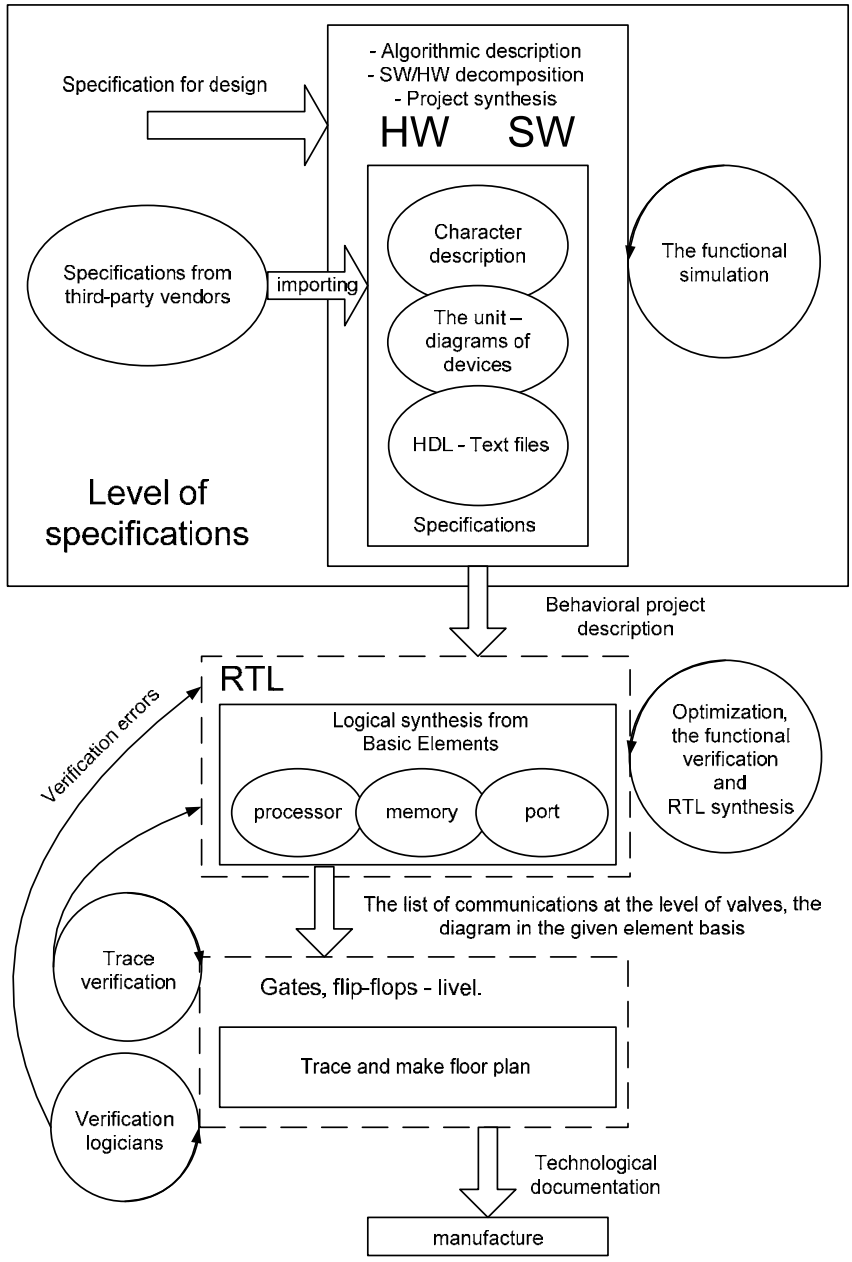

Fig. 2. Traditional single-chip computer system design flow

The key stage of the traditional VLSI design flow is the block synthesis stage. The distribution of algorithm operations into time slots and functional HW units and selection of the memory type are performed at this stage. Fig. 2 shows the key stages of single-chip computer system development in accordance with the traditional VLSI design flow.

The result of the block synthesis in traditional VLSI design flow is a chip layout plan that corresponds to the functional logic of the system being designed. This is the first step to the project presentation on RTL.

At the second stage the circuit structure on RTL is determined including selection of block types (combinational or sequential) and parallelization and pipelining of computations are performed. Then the developers do the final modeling of the system at the level of transfer of information between registers.

At the final stage the RTL presentation of the project on a hardware description language is transformed into gate structure. The result of this stage is a model of the system on the gate level. Then the developers test the model and check the timing of signals. After that the final wiring is routed out on the physical level [2].

The main problems of traditional VLSI design flow are as follows :

- The presence of a semantic gap between representations of the project on the system and RTL levels.
- Initial separation of the hardware and software components, which includes consistent iterative designing and modeling of each part separately and independently.
- Partial or full manual conversion of the project at the transitions from level to level with tight binding of the libraries used to the selected target realization.
- Manual development of software for the hardware designed.
- Limited possibilities for analysis of alternative variants of realization for the system being developed.

These problems are caused by the significant difference between abstraction degree of functional and RTL levels of the project description. A number of operating parameters of the system (eg, timing) are not considered when designing the system on the top (abstract) levels while when describing the system on the HDL-language they are often of critical importance. This brings up the problem of transition from the description on an algorithmic language to HDL.

Within this approach, the system is initially developed for a specific architecture and functional composition of the embedded VLSI elements based on standard or recommended by the manufacturer libraries that are used for a particular implementation. If the libraries are developed by the designer, there is no guarantee that the manufacturer will be able to implement them in the VLSI production process. And even for designing the chip based systems of the same manufacturer it may require to use different design systems.

For example, QUARTUS design system by Altera doesn't support the first generations of the family of MAX and Flex simple programmable logic integrated circuits (FPGA). MAX II Plus design system by the same company is not designed to work with older FPGA families: ACEX, Mercury, APEX, Excalibur and Cyclone. ISE WebPack free package is used for most systems based on Xilinx FPGAs but for older FPGA families by this company developers will have to buy the full-fledged version of Xilinx ISE design system.

At present algorithmic languages, for example C/C++, or MatLab platform are used for the system presentation in the traditional design flow. They can be used at the top level of abstraction. At this level the general algorithmization of the system is done but development of machine algorithms is not yet possible. HDL is applied at a lower level where the system is tied to a particular architecture. This is the cause of the semantic gap between the presentation of VLSI at the upper levels of abstraction and its embodiment in the chip.

The system-level algorithmic languages based on the representation of C language and having built-in HDL data types – SystemC and HandelC – are used. These languages are created specifically

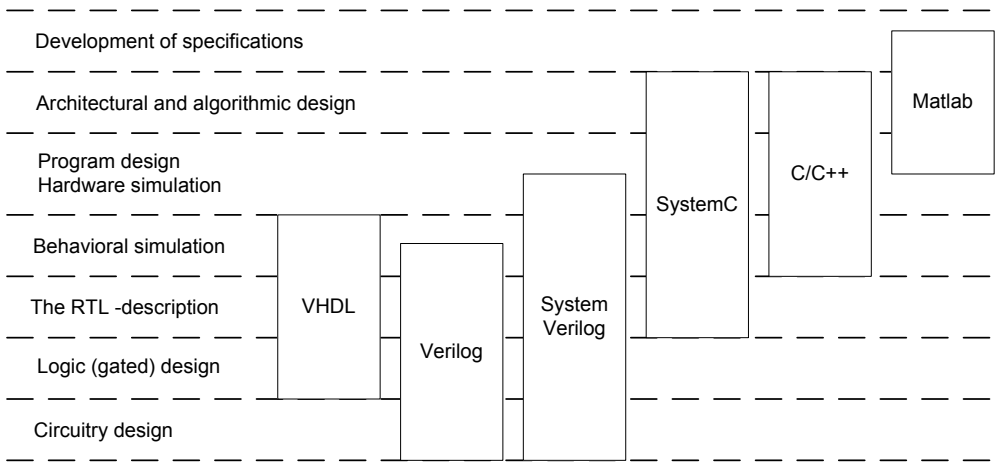

Fig. 3 The place of language tools [3]

as a C-like hardware description languages, so their level of abstraction from RTL is not high enough. On the other hand, a code based on these languages is modeled faster than, for example on such languages as VHDL or Verilog. Nevertheless, system simulation and debugging are rigidly tied to a particular architectural solution.

Tools for synthesis of these languages are rather cumbersome, and efficient translation in RTL-code, which is done by hand, requires high qualification of the engineers and mastering of translation and verification tools. Fig. 3 shows the place of language tools used at various stages of the project development under the traditional design flow [3].

3. Methods of high-level synthesis

Further development of the traditional method of the VLSI design by adding functional models at the top level of abstraction or its analysis on RTL-level led to the concept of designing on an abstract system level (Electronic System Level, ESL), which has been actively developed since 2004 and is supported by all major FPGA manufacturers and in the first placeby Xilinx.

ESL means any level of abstraction above the register transfer level. Design languages used at this level are closer to the syntax and semantics of ANSI C language than HDL. Within this approach, the design is carried out “top-down” and is based on simulation modeling. For this purpose the structure, links between the components and methods of communication are clearly described in the model being developed. In addition, a bank of graphic images is available for each model.

Within this approach two additional levels [2], the Message Level and the Transaction Level, are added at the top level of abstraction to the traditional stage of system description and specification package formation. In this case, the system specification is described in form of object diagrams (Fig. 4).

Unlike the traditional VLSI design flow where hardware and software are initially separated, there is no such separation at ESL design early stages of creating the system. At this stage, the developers set the design task, describe the composition of the developed system as a set of functional blocks and links between them. Then the developers debug the blocks interaction in order to check

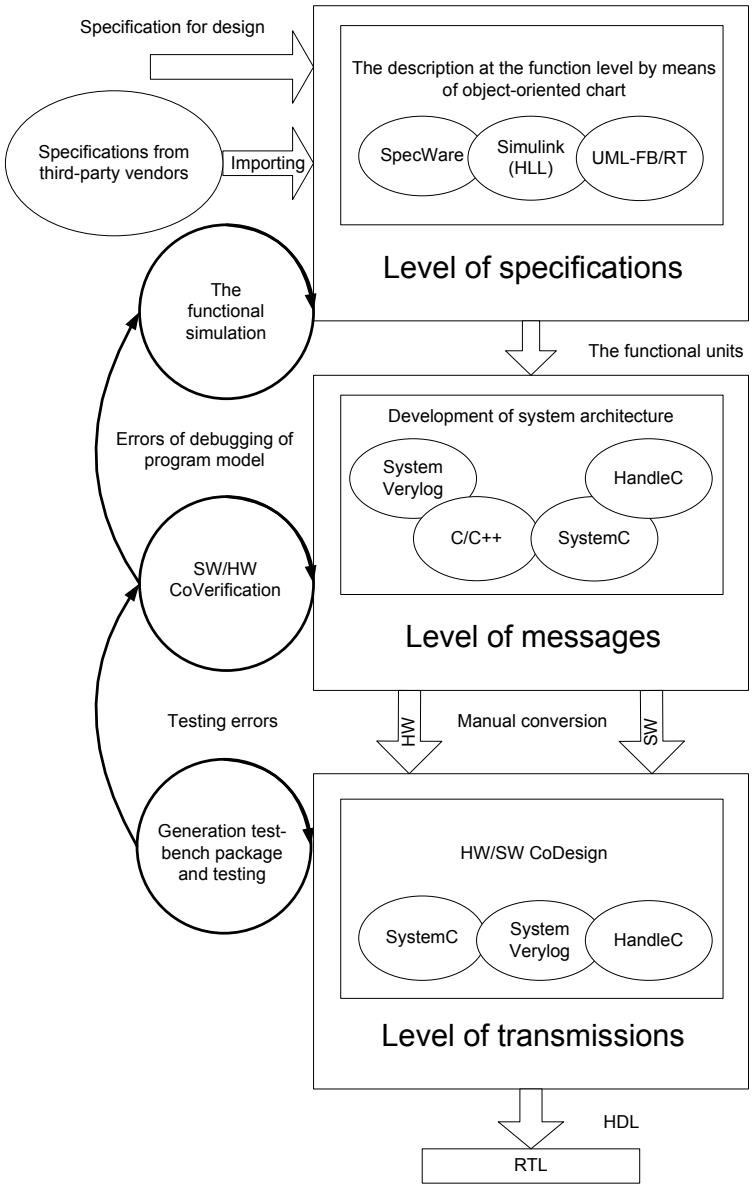

Fig. 4. System-wide level of VLSI design

the compliance of the designed functional composition with the assigned task. In this case, the development, debugging and verification of algorithms that solve the given task are completely separated from the subsequent process of transposition of these algorithms on the architecture of the radiation-resistant FPGA.

After completing the model the developer go to developing the system architecture. During this process, the blocks formed at the previous stage must be described on standard programming languages also without division into hardware and software parts. Test inputs, fragments of debugging software and tests for future hardware modules are generated. Detailed verification of the program model is based on tests and developed software.

If errors or discrepancies to initial task are detected during the verification, the design process returns back to the initial design level, or decomposition, or new block specifications are developed. This process is repeated until the complete elimination of errors and discrepancies.

Unfortunately, there is no guarantee that all the possible options of the project realization will be considered. However, debugging at the top level allows the developers to reduce the total design time significantly although it requires a high qualification of problem-oriented developers.

After achieving an acceptable result the manual decomposition of the developed system on hardware and software parts is done for the purpose of binding to the target standard hardware (microprocessor) and software platform. Within this approach, the resultant typical structures do not provide support for parallelism of the problem under solution though the local tasks for a particular structure can be solved inside it in parallel.

During the final design stage, hardware and software co-verification and debugging is performed at ESL level in order to verify system compatibility. For this purpose each unit is presented on SystemC/ SystemVerylog again and a test suite is generated to check the hardware of the developed system. At this stage of development the simulation of the system units is used with the possibility to disclose each of the system units along the hierarchy tree to the presentation at the gate level in order to identify the discrepancies between the required specifications and the design task. This process can be repeated several times to achieve the desired parameters.

The results of simulations for several realization variants are compared in order to determine the optimal separation of the hardware and software parts.

The high-level model that is debugged at a functional and algorithmic level with distinctly defined hardware and software modules presented on algorithmic languages is the result of designing the system on ESL.

Further transition into RTL level is done either manually, by rewriting the initial code into HDL, or semi-automatically, for example by generating Verilog/VHDL code from SystemC presentation.

Full automation of the transition to RTL is practically impossible, except for the SystemC synthesizing code. The result of this synthesis is an internal VLSI microarchitecture in the form of a “black box.” Note that gain in the size of the firmware file for the chip achieved through manual convention in comparison with the size of the file produced using logic synthesis based on behavioral RTL-model is less than 10% [4]. The rest of the design process is the same as the traditional design flow.

Note that ESL tools allow the developers to:

1. To do an “intelligent” FPGA segmentation and automatic conversion of software functions into equivalent functions of hardware.

2. To try much more embodiments of the system than using the traditional design, verify different strategies for segmentation of the developed system into software and hardware components and perform a quick comparative analysis of variants to achieve the optimum compromise between performance and size of the FPGA. As a result, the developers are often able to achieve higher system performance with shorter time put in design than when using traditional methods of conversion into RTL-code.

Work at a higher level of abstraction allows the developers that have experience in programming on universal problem-oriented programming languages to implement their ideas in the equipment in

a shorter time and without the involvement of a highly-experience hardware designer. ESL is most efficient for applications with a large amount of computation and number of cycles.

4. The main design issues

Despite ESL being more advantageous compared to conventional approaches, a number of issues specific to both methods remain unresolved. In particular, the developers have to analyze multiple variants of the project realization to obtain the optimum performance/ area occupied on the chip ratio. In this case it is difficult to predict the characteristics of the system such as area occupied on the chip, energy consumption, etc. Each design stage requires a mandatory system verification, which, in its turn, requires development and generation of a test package for each stage. However, the designers cannot guarantee that all possible alternatives are considered and the selected variant is the best of them.

In the course of the project realization it is difficult to make changes. At the algorithmic level modifications are simple but when passing to RTL description the entire project may again require reconfiguring and reverifying.

When designing at the levels below RTL the entire range of traditional design issues associated with the tight binding to the target platform of realization remains unresolved.

Improvement of the design process efficiency for complex SoC requires a new architecture-independent approach which, on the one hand, provides maximal abstraction of initial algorithms from the target chip architecture and, on the other hand, allows realizing a mechanism for transition into RTL level with parallel verification in the design without returning to the previous levels of the project creation.

Therefore, VLSI developers need a way of representing algorithms at the top level of the hierarchy that, during the subsequent top-down design flow, provides preservation of parallelism set at the top-level and allows performing a continuous verification in the process of FPGA topology formation without returning to previous levels of the hierarchy.

In this regard, the problem of finding relevant methods to describe parallelism of the source algorithms, which could further only is “compress” within the resource-specific limits.

5. Functional-dataflow model of parallel computations

Hardware description languages at the lower level are intended for graph description consisting of nodes (system elements) exchanging data and connections that ensure the process of exchange. Therefore, it is reasonable to form a system description starting from the upper level, using data flows rather than an imperative style. When implementing this approach it is necessary to rely on the use of functional data-flow presentation of algorithms rather than the programming on imperative languages.

In addition, the developers should verify system algorithms at the top formal level of abstraction, while the project in whole must be verified at the functional data-flow level, then generate RTL-code automatically and form the final gate description using RTL-synthesis tools.

Complete exclusion of the RTL-stage from the design flow is not possible due to the following reasons. First, plugged Intellectual Property cores (IP-cores) are available in the form of RTL-descriptions. Second, additional project verification and its debugging using the existing tools at RTL level is much faster than at the gate level. Third, the project correction at the gate-level is much more

complex than at RTL level. Finally, the analyses of power consumption, preliminary analysis of the wire routine and built-in test logic modules are now available only at the RTL level.

Given these reasons, substitution of the existing methods of high-level (applied) description of the problem being solved for the language and tools that support an architecture-independent parallel programming is of interest.

In order to support architecture-independent parallel program development in [5] there is proposed a functional data-flow paradigm underlying PIfagor language. The proposed language describes various algorithms and has advanced tools for presenting program and data parallelism. The use of implicit control based on data availability allows us to describe the maximal parallelism of the problem being solved. Architectural independence of the computation model used in the language is based on the following principles.

It is supposed that the virtual machine designed to perform functional data-flow parallel programs has unlimited computational resources. This allows allocating a new computing resource for each operation.

Looping is ensured by the use of recursion. This allows avoiding the resource limitations that are inherent for cyclic fragments. Asynchronous lists in conjunction with recursion allows creating algorithms with dynamically changeable parallelism [6].

PIfagor language constructions should eventually be transformed into a real parallel computing system architecture. This transformation is a serious challenge. The similar problems arise in case of a functional data-flow parallel program transformation into the FPGA topology. Solution of this problem contributes to improving the integrated circuits development process efficiency.

This approach can elevate debugging and verification efficiency due to the resource-independent description of parallelism. In this case the developers can use tools that are developed both for usual computers and for parallel computing systems with different architectures.

The intermediate data structures, which further on can be associated with different blocks implemented on FPGAs, are built during the program translation.

Among these intermediate structures there should be highlighted the following:

- reversive information graph describing the program operation dependences on the generated arguments;
- control flow graph defining control transfer from one performed operation to the other (depending on the characteristics of the program being developed, the control signal transfer may occur in parallel);
- data layer that ensures data storage;
- automata layer that responds to incoming values and change the status of the executed operations.

The intermediate structures generated during translation may be used for forming integrated circuit topology. They can be created using different variants of design flow and is consistent with existing flows. In particular, at the early stages it is possible to transform a functional data-flow parallel program into a program written in a language that supports ESL. This allows implementing the proposed technology promptly through the use of existing tools, libraries and VLSI design flows. The direct transition from the functional data-flow parallel program into architectural-dependent representation at a relevant language is also possible.

In the future, it is possible to move in two directions. The developers can focus on the implementation of the final topology using previously developed processor modules. The other way is to develop computing modules for the different intermediate structures of functional data-flow computation model, which will shape the VLSI topology.

Functional data-flow model of parallel computation can be supported by the following components:

- computing modules, that provide execution of operation in the nodes of the information graph (by means of these elements contribute to execution of arithmetic and logic operations and manipulation of data structures);
- modules intended for generating and transmitting control signals between the nodes of the program information graph;
- modules that implement automaton for status control of the information graph nodes.

Analysis of the data arrival rate allows determining the number of computing nodes (adders, multipliers, etc.) that ensure effective calculations in real time on the level of functional streaming parallel program.

6. Summary and Conclusions

This paper presents a new approach to forming the topology of FPGAs based on the use of functional data-flow paradigm that provides architectural independent description of implemented parallel algorithms. This approach allows describing the developed system on the algorithmic level without being tied to a specific realization and separation into hardware and software components but taking into account the initial parallelism of the problem being solved. Due to the lack of an explicit description of parallelism, debugging and verification of parallel programs become easier. Development of tools that provide a direct translation from the considered language into the language tools used in FPGA design improves the efficiency of the design process.

This article is prepared with support a federal target program, a project code 14.578.21.0021.

References

[1] *Rabaey J.M., Chandrakasan A., and Nikolic B.* Digital integrated circuits. A design perspective. M.: Viliams, 2007. 912 p.

[2] *Kolesnikov E.I.* // Collected reports of the VI-th Russian interuniversity conference of young scientists. SPb: ITMO, 2009. P. 175.

[3] *Shagurin I.A., Kanyshev V.I.* // Chip-News. Engineering microelectronics. 2006. № 9(112). P. 51.

[4] *Nepomnyaschy O.V., Alekminsky S.U.* // The nano and microsystem technique. M.: New technologies. 2010. №9(122). P. 4.

[5] *Legalov A. II.* // ICM RAS, Computing technologies. 2005. № 1 (10). P. 71.

[6] *A. I. Legalov, O. V. Nepomnyaschy, I. V. Matkovsky, M. S. Kropacheva* // Tail Recursion Transformation in Functional Dataflow Parallel Programs. Automatic Control and Computer Sciences. 2013. Vol. 47. No. 7. P. 366–372.