DOI: 10.15514/ISPRAS-2022-34(1)-5

Parallelism reduction method in the high-level VLSI synthesis implementation

[1,2] D.S. Romanova, ORCID: 0000-0002-9020-4802<daryaooo@mail.ru>
[1] O.V. Nepomnyashchiy, ORCID: 0000-0002-2459-6414 <2955005@gmail.com>
[1] I.N. Ryzhenko, ORCID: 0000-0002-3069-9102 <rodgi.krs@gmail.com>
[3] A.I. Legalov, ORCID: 0000-0002-5487-0699 <legalov@mail.ru>
[1] N.Y. Sirotinina, ORCID: 0000-0001-8816-4226 <nsirotinina@sfu-kras.ru>

[1] Siberian Federal University,
79, Svobodny pr.79, 660041, Krasnoyarsk, Russia
[2] Krasnoyarsk State Agrarian University
Mira pr. 90, 660049, Krasnoyarsk, Russia
[3] National Research University - Higher School of Economics
Myasnitskaya str.20, 101000, Moscow, Russia

Abstract. In the article the problems and solutions in the field of ensuring architectural independence and implementation of digital integrated circuits end-to-end design processes are considered. The method and language of parallel programming for functional flow synthesis of design solutions is presented. During the method implementation, the tasks of reducing parallelism and estimating the occupied resources were highlighted. The main feature of the developed method is the introduction of the additional meta-layer into the synthesis process. Algorithms for the parallelism reduction have been developed. The results of software tools development for design support and practical VLSI projects are presented.

Keywords: integrated circuit; algorithm; program; parallel computing model; high-level synthesis; functional-stream language

For citation: Romanova D.S., Nepomnyashchiy O.V., Ryzhenko I.N., Legalov A.I., Sirotinina N.Y. Parallelism reduction method in the high-level VLSI synthesis implementation. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 1, 2022, pp. 59-72. DOI: 10.15514/ISPRAS–2022–34(1)–5

Acknowledgements. The article is based on the materials of the report at the Seventh International Conference «Actual Problems of System and Software Engineering» (APSSE 2021).

Метод редукции параллелизма в процессе высокоуровнего синтеза цифровых интегральных схем

[1,2] Д.С. Романова, ORCID: 0000-0002-9020-4802<daryaooo@mail.ru>
[1] О.В. Непомнящий, ORCID: 0000-0002-2459-6414 <2955005@gmail.com>
[1] И.Н. Рыженко, ORCID: 0000-0002-3069-9102 <rodgi.krs@gmail.com>
[3] А.И. Легалов, ORCID: 0000-0002-5487-0699 <legalov@mail.ru>
[1] Н.Ю. Сиротинина, ORCID: 0000-0001-8816-4226 <nsirotinina@sfu-kras.ru>

[1] Сибирский федеральный университет,
660041, Россия, г. Красноярск, пр. Свободный, 79
[2] Красноярский государственный аграрный университет
660049, Россия, г. Красноярск, пр. Мира, 90
[3] Национальный исследовательский университет «Высшая школа экономики»
101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. Рассмотрены проблемы и решения в области обеспечения архитектурной независимости и организации процесса сквозного проектирования цифровых интегральных схем. Представлен метод и язык параллельного программирования для функционально потокового синтеза проектных решений. При реализации метода выделены задачи редукции параллелизма и оценки занимаемых ресурсов. Предложен способ свертки, базирующийся на введении дополнительного, мета-слоя в процесс синтеза. Разработан принцип и алгоритмы редукции параллелизма. Представлены результаты разработки программного инструментария поддержки проектирования и реализованные на практике проекты СБИС.

Ключевые слова: интегральная схема; алгоритм; программа; модель параллельных вычислений; высокоуровневый синтез; функционально-потоковый язык

Для цитирования: Романова Д.С., Непомнящий О.В., Рыженко И.Н., Легалов А.И., Сиротинина Н.Ю. Метод редукции параллелизма в процессе высокоуровнего синтеза цифровых интегральных схем. Труды ИСП РАН, том 34, вып. 1, 2022 г., стр. 59-72. DOI: 10.15514/ISPRAS–2022–34(1)–5

Благодарности. Статья подготовлена по материалам доклада на Седьмой Международной конференции «Актуальные проблемы системной и программной инженерии» (АПСПИ 2021).

1. Introduction

The current level of digital integrated circuits (IC) development is characterized by constantly increasing requirements for the systemic organization of the entire design flow. One of the most important tasks is to reduce the time to get the final result. The main reasons for slowing down the design flow are iterative operations that lead to returning to previous stages. The elimination of iterative operations provides an “end-to-end design flow” resulting in lower financial costs and increased product competitiveness. Another urgent task is to ensure the architectural independence of the product being developed, in other words, design solutions portability between the target platforms of the IC. Portability allows the developer to optout of being tied to the target implementation platform that provides more efficient solutions for key product specifications such as speed, chip area, etc.

An integrated circuit is, in fact, a system for parallel processing of information flows. At the final stages of the synthesis, the architecture and operation algorithm of the IC are presented in hardware description languages. Therefore, efficient solutions for ensuring architectural independence can be found in the area of portable parallel programs. In turn, end-to-end design can be ensured through the use of a parallel computing model of the functional flow and the presentation of the original algorithms in the form of acyclic structures [5, 10, 12, 13].

2. Related work

A key feature of well-known research in this area is the use of a functional-flow parallel computing model and language for the initial description of algorithms for the operation of IC.

Currently, there is a steady tendency to increase the abstraction of the initial algorithms from the final implementation of the project. At the same time, methods for describing the IC architecture develop in several ways. The most common is the introduction of constructs for high-level description into existing hardware description languages (HDL) [1]. Such solutions led to the emergence of the SystemVerilog language based on the classic Verilog [1]. But even in this case, the level of abstraction from the specific architecture of the target chip is not fully provided.

There are a number of solutions based on the use of adapted imperative high-level programming languages (primarily C and C ++) as hardware description languages, for instance, SystemC [2], Handel-C [3], and Impulse-C [4]. However, these languages were created to solve narrow problems, for example, to implement streaming applications or to support alternative programming models, so they do not provide architectural independence of the designed solutions. They are predominantly sequential languages; therefore, they do not support parallelism, which is necessary for describing parallel processes occurring in the IC.

Of particular interest is the COLAMO programming language [5], which is a high-level language with an implicit description of parallelism. Parallelization is achieved by declaring variable access types and indexing array elements, which is typical for data flow languages. Currently, this language is used for programming reconfigurable computing systems. It allows developing parallel application programs with high specific performance [5]. However, this language is focused on solving applied problems in the field of high performance computing for multichip systems.

The most effective solutions are obtained using functional languages that have a more powerful abstraction mechanism and a developed type system. The initial IC operating algorithms described in similar languages are easier to transform and verify [6]. For example, the languages Hydra [7] and uFP [8] use flows to describe signals and recursive expressions to provide schema transformations. Lava, like Hydra, is a built-in subset of Haskell. It possesses powerful circuit design tools derived from its predecessor. At the same time, Lava is simpler and more convenient to use due to the extended type system for describing hardware. However, the mechanism of "lazy" calculating inherent in Haskell and the sequential structure of lists impose restrictions on parallelism transformations and do not allow efficient automatic parallelization of programs [12].

The works of Donnagara [10, 11] show that the effective portability of the IC initial description can be implemented by using functional programming languages and presenting algorithms as data flow graphs. The high efficiency of this approach is proved in practice using the example of parallel software (PaRSEC) for high performance computing [11].

In this case, the solution lies in the application of a programming paradigm, which must meet the following conditions [10]:

- Lack of explicit control over computations (control by data availability / readiness);
- Data flow model;
- Parallelism at the level of operations.

The model that meets the listed requirements is the basis of the functional-flow parallel (FFP) programming languages [12]. This gives grounds to consider the FFP programming methods and the corresponding computation model as the most suitable for solving the problem of high-level architecture-independent IC synthesis.

Among the programming languages that support the FFP model, the Pifagor language is chosen [12]. This language allows a developer to describe an initial algorithm without resource constrains, supports a data flow model and parallelism at the operation level, which is the most important for the architecture-independent end-to-end design of IC. In Pifagor language, architectural independence is achieved by describing only informational connections existed in the program.

Unlike Haskell, Pifagor only supports explicitly describing delayed computations. This allows executing alternative program fragments only when they are needed. Also, there are no loop operators in Pifagor. This prevents conflicts when using different data with the same fragments of a parallel program. In theory, this allows a program to start executing any function as soon as its input is ready. In practice, only resource limits are imposed on the maximum concurrency specified in the language.

The authors have proposed a method for an architecture-independent high-level VLSI synthesis on the basis of the FFP computation model and the Pifagor language that supports it [13]. When implementing the method, the dynamic type system was replaced by a static one, which is supported in hardware description languages, delayed computations were excluded, and it was proposed a method to transform them in compliance with the target IC platform. Also, in the modified Pifagor language, a mechanism for converting recursion into an iterative scheme with a subsequent transition to a pipeline scheme was introduced [13].

The proposed method for ensuring architectural independence in the course of high-level synthesis assumes that a program in Pifagor is transformed into an intermediate representation in the form of a pair of graphs: a data–flow graph (DFG) and a control-flow graph (CFG). DFG specifies information connections, and presenting CFG in an explicit form allows a more detailed description of the computational control process.

The DFG is shaped during the program translation, and the CFG can be elaborated both dynamically and statically. The last approach is used to switch from a program in Pifagor to a program for describing IC in an HDL language.

The intermediate presentation of the FFP of the program during translation is developed in two stages. At the first stage, the initial code is translated into DFG. Then, a control-flow graph is formed from the obtained data-flow graph. CFG can be formed from DFG in various ways. For example, in accordance with the model of computation control by data flow, either it can be reduced to a sequential traversal of the DFG, or provide another strategy for managing computations.

To switch to the target IC platform, the DFG is converted into a pipeline scheme. The pipeline computing scheme is a tiered-parallel form of DFG.

To convert parallelism of the initial algorithm to the target IC platform taking in account the specific resource constraints, the following stages are performed: determining the boundaries of changing parallelism, an algorithm for changing parallelism, and evaluating the result based on resource constraints.

The main task of the transition from the unrestricted parallelism of the FFP model to the target platform is reduction of parallelism. This is the key moment of all ongoing transformations in the architecture-independent method of VLSI synthesis. To implement the reduction mechanism, a new meta-layer called "HDL graph" was introduced. It is an intermediate layer for making changes to the FFP model. According to the authors, the term "HDL graph" most fully corresponds to the method of VLSI synthesis. HDL graph allows you to specify connections between elements of lists of connected vertices, which in turn allow performing optimization transformations by calculating the operation of selecting data from the list. With the help of the introduced meta-layer, when processing types in a statically typed model, a restriction is introduced on the dynamic resizing of lists during computations.

Recursive computations often become an obstacle when porting such programs to some real computing platforms, since at a significant depth of recursion memory overflow can occur. The introduction of the HDL graph made it possible to solve this problem by transforming such computations into iterative ones using tail recursion and specifying the recursion depth at the translation stage.

*3. Parallel conversion*3.1 Parallelism reduction algorithm

The main feature of the introduced synthesis method is the transition from parallelism induction into the algorithm description to reducing the initial maximum-parallel algorithm description according to specific resource constraints of the target platform. As shown in [10, 11], this ensures the portability of parallel algorithms to various platforms. The main advantage of parallelism reduction compared to induction is significant decreasing in the number of steps required to obtain the final result. When reducing a maximally parallel data flow graph, the number of maximally admissible transformations is set at the synthesis stage.

In the process of synthesis, the following tasks are solved:

- Assessment of the resources of the resulting architectural solution;
- Evaluation of the performance of the resulting solution;
- Calculation of the reduction factor for each class of resources;
- Parallelism reduction of the circuit to achieve the required coefficients.

To estimate resources, an intermediate representation of the program HDL graph is used, in which architecture-dependent data are already specified. An HDL graph is an acyclic graph in a tiered-parallel form, at each node of which types and widths of data are specified [14]. The N_k resource classes whereby the scheme should be evaluated are determined depending on the target platform.

For example, the main resource classes of the FPGA platform include:

1) Number of registers N_r;
2) Number of logical cells N_{lc};
3) The amount of block memory N_m Nm;
4) Number of arithmetic and other specialized computing units N_{dsp}.

Resources can be divided into two subsets:

- Memory resources $N_{mem} = \{N_r, N_m\}$;
- Computing resources $N_{comp} = \{N_{lc}, N_{dsp}\}$;

Within the subset, there are restrictions for fungible resources. For memory resources, any data storage can be implemented on block memory, while its implementation on triggers has volume restrictions. For computational resources, it is possible to implement any computation on logical cells, while the type and set of operations for specialized blocks is limited. Resource estimation results are used to further parallelism reduction.

3.2 Memory resources estimation

To estimate the required memory amount, the total amount of resources can be referred to the total amount measured in bits (kbit).

As an example for estimating the circuit resource, consider its HDL graph. Each k-th layer of the tiered-parallel form consists of a set of information inputs B_k and a set of operations O_k. After data typing, each information input of HDL graph vertices has a width of W_k. Based on the input number and the bit width of each input of a specific graph layer, it is possible to determine the amount of memory required to store the result in the corresponding graph state:

To calculate the resource, it is necessary to traverse the graph and sum up the bit widths of the inputs and outputs of all vertices:

$$NR = \sum NR_k.$$

After an initial evaluation of the required memory resource for the initial maximum parallel implementation of the circuit, two options are possible:

The required resource is less than the available one $NR < N_{m/lc}$, where $N_{m/lc}$ is an available resource;

The required resource is greater than the available one $NR \geq N_{m/lc}$

The first option does not require the calculation of the reduction factor for memory resourcesG_m. But when the scheme is changed during its reduction, for other resources, the memory resource must be evaluated and checked again.

In the second case, the reduction factor G_m is calculated using the following formula:

$$G_m = \frac{NR}{N_{m/lc}}$$

This factor is used in the parallelism reduction algorithm of the scheme.

In addition to the memory limitation, the memory performance limitation must also be considered. If the memory is built on registers/ flip-flops, his limitation does not exist, since the width of the data bus is equal to the amount of data. For block memory, the amount of data read per clock cycle is less than the amount of stored data. If resource NR exceeds the available volume of N_r registers, it is necessary to calculate the total data interface to the memory and the reduction factor by the memory interface G_{md}.

3.3 Algorithm for determining the reduction coefficient

To determine the minimum required reduction factor over the memory interface, a set of stages with the maximum total interface implemented in registers is selected from all stages of the pipeline. To do this, a subset of the stages is selected from the set of pipeline stages, such that:

$$(N_{m/lc} - NR_r) > max \sum NR \;||\; \sum NR_k.$$

The || sign denotes logical addition.

The G_{md} factor is defined as the ratio of the total memory interface of the remaining stages to the total block memory interface IM.

The following algorithm is used to determine the reduction factor over the memory interface:

To select a subset of the pipeline stages such that the sum of resources NR_k of the selected stages will be less than NR_r and selected values sum NR_k will be maximum;

To calculate the memory resource implemented on block/sequential memory:

$$NR_m = NR - \sum NR_k,$$

where k belongs to a subset of the pipeline stages selected at step 1.

Calculate the coefficient G_{md}:

$$G_{md} = 1 + Int(NR_m/IM),$$

where Int is rounding to an integer value.

As an example, the calculation of a 4-point FFT (Fast Fourier Transform) will be considered, where the discrete Fourier transform is calculated by the formula:

$$c_n = \frac{1}{n}\sum_j^{n-1} S_j * W_n^{-kj} + \frac{1}{n}\sum_j^{n-1} S_j * W_n^{kj}.$$

The input data type is signed 16-bit.

The Data-Flow graph after transformation into HDL graph and reduction to a tier-parallel form (TPF) is shown in fig. 1.

The value of recourses NR_k for each stage of the pipeline is calculated as follows:

$$NR_1 = 10 * 2 * 16 = 320;$$
$$NR_2 = 16 * 4 + 8 * 32 = 320;$$
$$NR_3 = 16 * 4 + 4 * 33 = 196;$$
$$NR_4 = 33 * 4 + 4 * 34 = 268.$$

Total resource value is:

$$NR = 1104\ bits.$$

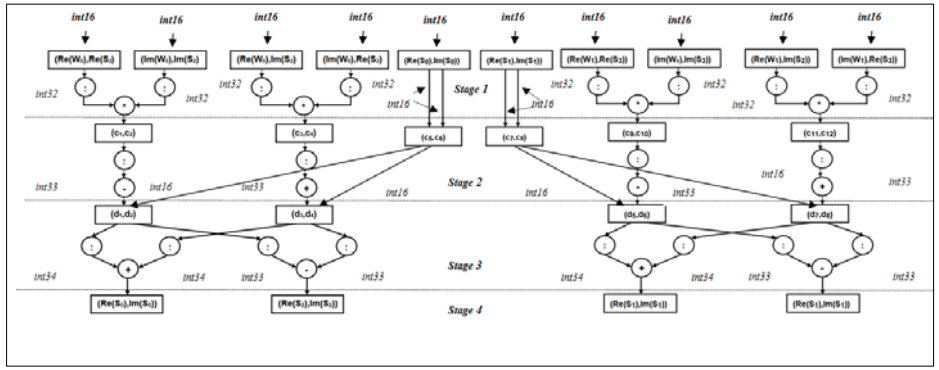

Fig. 1. HDL graph of the maximum parallel form of 4-point FFT

Let's consider two architectures, A1 and A2. Value $N_{m/lc}$ for both architectures is 1536 bits. In the A1 architecture, the entire memory resource is in registers. For such architecture, the 4-point FFT scheme in maximum parallel form is implemented unchanged, as in fig. 2.

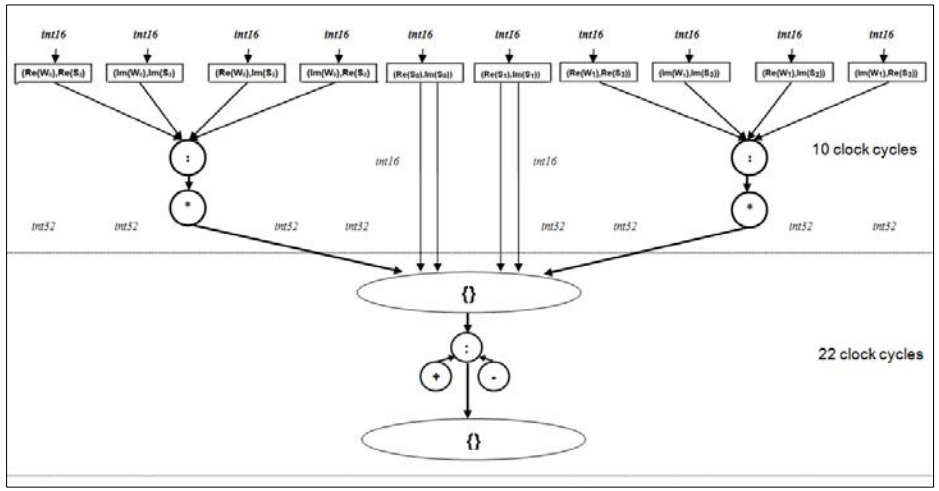

Fig. 2. HDL graph after reduction

In the A2 architecture, the register resource is 512 bits, and 1024 bits are presented in the form of block memory with a data interface of 36 bits. The total block memory interface value IM is 36 bits.

In accordance with the algorithm for calculating G_{md}, a subset of stages that are implemented in registers and have a maximum interface is selected. In this example, it can be either stage 1 or stage 2.

In this case, value NR_m will be calculated as follows:

$$NR_m = NR - NR_1 = 1104 - 320 = 784 \text{ bits.}$$

The value of the reduction factor by the memory interface will be:

$$G_{md} = \frac{784}{36} = 22.$$

In this case, the data feed period becomes equal to G_{md} and pipeline stages 2, 3, 4 or 1,3,4 are implemented sequentially, since the result is written to one memory block. The scheme obtained as a result of the reduction with the corresponding ratio G_{md} is shown in fig. 2.

The values c_1-c_{12}, d_1-d_8 and S_0-S_3 are calculated in this scheme sequentially. Since stages 2, 3 of the original scheme after reduction require 22 cycles to execute, stage 1 can also be increased to 22 cycles without affecting the overall performance of the system, which will lead to a proportional decrease in the computational resource of stage 1.

3.4 Estimation of the required computing resources

The total number of layers (stages of the pipeline) is M. At each j-th layer of the graph, a certain set of operations is implemented: O_j , where $j = 1, \dots, k$.

For the entire HDL graph, the number of each operation:

$$F_k = \sum_{j=0}^{M} O_k^j.$$

Let L_k be the total number of different types of operations, where $k = 0, \dots, L$. The type of operation here means the type of arithmetic / logical, etc. operations along with the indication of the data width. For example, adding 16-bit data, comparing 20-bit data, etc.

After calculating the total number of operations of each type based on the available a specific architecture resource, it is necessary to assess the degree of parallelism with which it is possible to implement the scheme.

It is supposed that the amount of resource for each specific type of operation is known. Let Y be the type of resource (logical cells, specialized arithmetic blocks DSP, etc.), $V(Y)_k$ will be the amount of resource of type Y required to implement an operation of type k. Then the total resource of type Y for all operations in the HDL graph will be:

$$V(Y) = \sum_{k=0}^{L} VY_k * F_k.$$

For each class of computing resources, the reduction factor G_y can be calculated as the ratio of the total required resource to the resource of the target architecture, rounded to the nearest larger integer:

$$G_y = Int\big(V(Y)/N_y\big).$$

So, the final reduction factor for computing resources is determined as the maximum among all reduction factors:

$$G_{calc} = max\{G_y\}$$

This algorithm is considered using the example of the graph diagram shown in fig. 2. The total number of pipeline stages for a given graph is 4. The number of operation types L will be 5: 16-bit multiplication ($k = 0$), addition and subtraction of 33 and 16 bits ($k = 1.2$), addition of 34-bit data, and subtraction of 33-bit data ($k = 3.4$). The number of operations on the layers of thegraph will be:

$$O_1 = \{8, 0, 0, 0, 0\};$$
$$O_2 = \{0, 2, 2, 0, 0\};$$
$$O_3 = \{0, 0, 0, 4, 4\};$$
$$O_4 = \{0, 0, 0, 0, 0\};$$

The number of each operation of the k-th type:

$F_0 = 8,\ F_1 = 2,\ F_2 = 2,\ F_3 = 4,\ F_4 = 4.$

Let there be architecture with two types of resources for implementation of computations: logical cells ($Y = 0$) and DSP sections ($Y = 1$). The following resource values for each type of operation are taken:

For logic cells:

$$V(0)_0 = 0,\ V(0)_1 = 5,$$
$$V(0)_2 = 5,\ V(0)_3 = 10,\ V(0)_4 = 10.$$

For DSP sections:

$$V(1)_0 = 1, V(1)_1 = 0,$$
$$V(1)_2 = 0, V(1)_3 = 0,\ V(1)_4 = 0.$$

The total resource for each type for all operations in the graph will be:

$V(0) = \sum_{k=0}^{L} VY_k * F_k = 0*8 + 5*2 + 5*2 + 4*10 + 4*10 = 100,$

$V(I) = \sum_{k=0}^{L} VY_k * F_k = 1*8 + 0*2 + 0*2 + 0*10 + 0*10 = 8$

The architecture where the number of DSPs is $N_1 = 2$, the number of logical cells is $N_0 = 200$ is considered. According to it, the reduction factor for each type of resource is:

$$G_0 = Int\left(\frac{V(0)}{G_0}\right) = \frac{100}{200} = = 0.5;$$

$$G_1 = Int\left(\frac{V(1)}{G_1}\right) = \frac{8}{2} = 4.$$

The value of the reduction factor for computing resources G_{calc} is:

$$G_{calc} = \max\{0.5, 4\} = 4.$$

After reducing each stage with a factor of 4, the delay of each stage will increase to 4 clock cycles, while the resource will also decrease by 4 times. The resulting circuit is shown in fig. 3.

Note that changing the computational resource may change the memory resource.

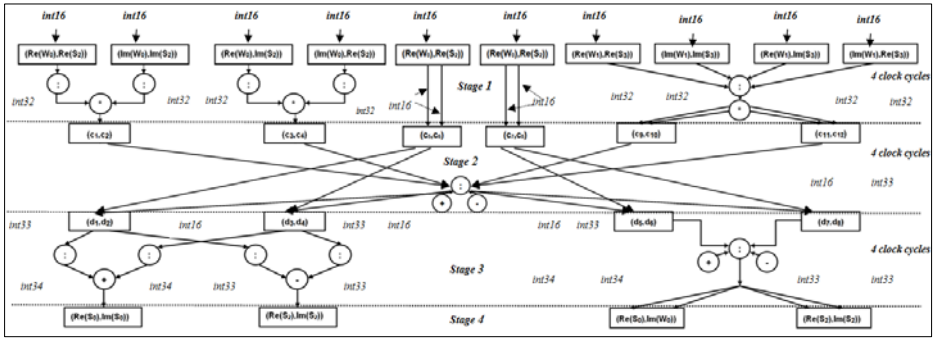

Fig. 3. Scheme of HDL graph after reduction by Gcalc

3.5 Generalized Parallelism Conversion Algorithm

Let the ratio of the resource required to implement the circuit in the original maximally parallel form to the available resource be denoted as R. It is assumed that the available resource is the smallest of the resources. There are three options for transforming the original maximum parallel scheme can be distinguished depending on the available resource of the target platform.

The first option: if $R > 1$, a reduction in parallelism is required.

In this case it is possible to increase performance by placing several circuits in parallel:

$$S = Int(1/R).$$

Consider a reduction algorithm using the reduction coefficients described in subsections 3.2-3.4.

1) Calculating the reduction factors G_m and G_{calc};
2) Choosing the maximum coefficient $G_{max} = \max(G_m, G_{calc})$;
3) Reducing the parallelism of the circuit to G_{max};
4) Recalculating the coefficients G_m and G_{calc} for the modified circuit;
5) If they are less than 1, the algorithm is finalized;

6) If some operations can be implemented using a different type of resources, change these operations to another type of resources without changing the R coefficient and recalculate the G_m and G_{calc} coefficients;
7) If any of the coefficients are greater than 1: $G_{max} = 1 + G_{max}$ and return to step 3.

A sequential increase in the reduction factor allows selecting the minimum possible ratio to meet the resource requirements and at the same time achieve maximum performance (the minimum possible reduction in performance relative to the initial maximum parallel version).

Second option: when $R < 1/2$, an increase in the number of circuits is possible.

The third option: when $1/2 < R < 1$ the resource is enough to accommodate 1 maximum parallel version of the circuit.

In the second and third options, no conversions of the maximally parallel circuit are required.

4. Results

In the framework of the research, the authors have implemented a set of software tools that perform the following functions:

- Transformation of the source code in the FFP language into an intermediate representation in the form of an DFG and CFG;
- Optimization transformations that increase the efficiency of FFP programs;
- Debugging and analysis of FFP code at runtime, including finding errors and tracing;
- Compilation of the intermediate representation of FFP programs into the description of VLSI in HDL languages [15].

Fig. 4 shows the architecture of the developed design support tools based on the proposed high-level synthesis method

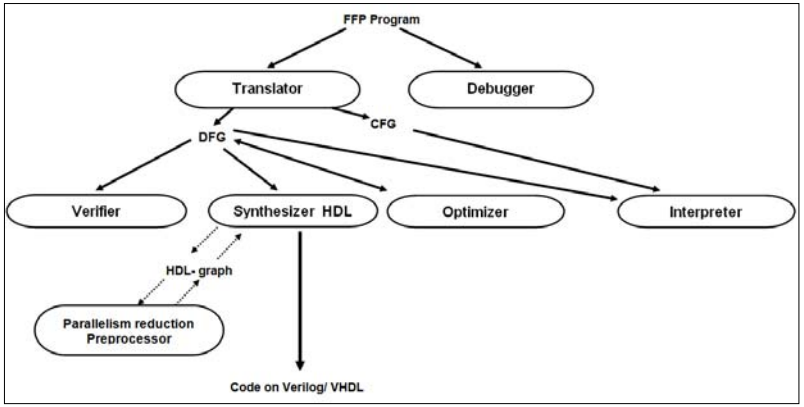

Fig. 4. Architecture of design support tools

The interpreter provides executing the program developed in the FFP language. The input data for the interpreter is the data flow and control graph, as well as the argument of the top-level function. The argument is presented in the format of the DFG description, for this it is processed by the translator.

The optimizer also uses the input intermediate representation and performs optimization of the DFG, the result is saved as an intermediate representation of the DFG.

Optimizing transformations carried out at this stage include:

A. Removing unused code;
B. Optimizing repetitive calculations;
C. Direct function substitution;
D. Removing duplicate code;
E. Optimizing based on equivalent transformations of the FFP algebra of the model.

The translator checks the syntax of programs developed in the Pifagor functional data flow parallel programming language and converts the program into its intermediate representation in the form of DFG and CFG [16]. It transforms a functional-flow description into a description at the level of combinational logical circuits. The translator includes a debugger, a DFG generator, and a CFG generator. The result of the translator's functioning is a set of debugged functions implemented in the Verilog / VHDL languages.

The developed software package also includes a parallelism reduction preprocessor and resource estimation preprocessor. The parallelism reduction preprocessor automatically converts the parallelism of programs intended for translation into an HDL language, taking into account resource restrictions. The preprocessor gets an intermediate representation of the IC operating algorithm in FFP language in the form of a typed data flow graph (HDL graph) and resource constraints of the target platform obtained using the resource constraints preprocessor. The result of the operation of the parallelism reduction preprocessor is the data flow graph, transformed taking into account resource constraints. The resulting representation is used by the circuit synthesizer to obtain a description of the IC in HDL languages.

Translators have been developed for the Verilog and VHDL languages [17]. The program implements checking the initial description for suitability for synthesis, assembling the initial description from a set of functions, assigning data types in the original description and synthesizing the output circuit description in Verilog / VHDL languages.

A set of software tools functions as part of an integrated development environment and allows a developer to form a set of debugged functions for their implementation in the form of a IC. The shell provides information resources for organizing the entire process of high-level VLSI synthesis based on the FFP approach.

By means of the developed tools, a number of scientific and technical solutions have been obtained: a set of complex functional blocks of a single-chip driver of the on-board network of a spacecraft [18], VLSI of the DSP unit based on the BMK K5540TN014A from the MRK06 navigation device [19] and others.

5. Conclusion

The review of recent languages and methods for designing logical circuits made it possible to substantiate the choice of the functional-flow parallel computing model and the Pifagor parallel programming language for the development of an architecture-independent method for synthesizing integrated circuits.

In the process of developing the proposed method for synthesizing IC based on a modified FFP model, a parallelism transformation method was proposed, which consists in reducing the maximum parallelism of the IC operating algorithm when switching to specific target architecture. This approach provides portability of parallel architectures to different platforms.

The parallelism transformation includes resource estimation, calculating the parallelism reduction factor for each resource class, and reducing circuit parallelism to achieve the required characteristics of a specific target platform.

The presented reduction methods make it possible to change the parallelism of the initial algorithm description and provide the implementation of the transfer mechanism to different architectures with different resource constraints.

The proposed method, in contrast to the parallelism induction methods, reduces the complexity of the transfer process by reducing the enumeration of the number of implementation options obtained in the synthesis process, taking into account resource constraints. At the same time, resource estimation methods at the high-level stage require accounting for overheads when changing parallelism to more sequential schemes. For this case, it is necessary to increase the accuracy of the estimate. For this, neural networks and machine learning methods can be used, which, based on the parameters of the circuit evaluation at a high-level stage, can predict the values of the circuit parameters with the required accuracy after implementation at a low level.

Direct calculation of resource required to implement the developed circuit is complicated due to the many transformations that occur during the synthesis and implementation at the low-level stage. The implementation of accurate methods for assessing the resource will further reduce the number of circuit options considered in the synthesis process under resource restrictions.

On the basis of the proposed methods of parallelism reduction and resource estimation, a number of tools have been implemented, such as a translator of an architecture-independent description of automaton and combinational schemes, resource estimation and parallelism reduction preprocessors. Preprocessors calculate and quantify the required circuit resource for selected types of unit blocks (cells, memory, triggers, LUTs, etc.) on supported target platforms.

References / Список литературы

[1] IEEE Std 1800-2012: IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language, 2013.
[2] V.A. Alekhin. SystemC. Simulation of electronic systems. Goryachaya liniya – Telecom, 2018, 320 p. (in Russian) / Алехин В.А. SystemC. Моделирование электронных систем. Горячая линия – Телеком, 2018, 320 стр.
[3] Vivado Design Suite User Guide. High-Level Synthesis. UG902–Xilinx–2015. URL: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_4/ug902-vivado-high-level-synthesis.pdf
[4] Handel-C Language Reference Manual. Celoxica Limited, 2005, 26 p.
[5] I.I. Levin, V.A. Gudkov. Extension of high level language COLAMO for reconfigurable computer systems programming on the level of FPGA logic cells. Herald of computer and information technologies, no. 12, 2010, pp.10-17 (in Russian) / И.И. Левин, В.А. Гудков. Расширение языка высокого уровня COLAMO для программирования реконфигурируемых вычислительных систем на уровне логических ячеек ПЛИС. Вестник компьютерных и информационных технологий. no. 12, 2010 г., стр. 10-17.
[6] J. Sérot, G. Michaelson. Compiling Hume down to gates. In Draft Proc. of the 11th International Symposium on Trends in Functional Programming, 2011, pp, 191-226.
[7] J. O'Donnell, M.R. Barbacci, and C.J. Koomen. Hardware description with recursion equations. In Proc. of the 8th International Symposium on Computer Hardware Description Languages and Their Applications (CHDL '87), 1987, pp. 363-382.
[8] M. Sheeran. μFP, a language for VLSI design. In Proc. of the Conference Record of the ACM Symposium on LISP and Functional Programming (LISP '84), 1984, pp. 104-112.
[9] P. James-Roxby, S. Singh. Lava and JBits: From HDL to bitstream in seconds. In Proc. of the 9th IEEE Symposium on Field-Programmable Custom Computing Machines, 2001, pp. 91-100.
[10] J. Dongarra, G. Bosilca et al. PaRSEC: A programming paradigm exploiting heterogeneity for enhancing scalability. IEEE Computing in Science and Engineering, vol. 6, no. 15, 2013, pp. 36-45.
[11] J. Dongarra, A. Danalis et al. PTG: An Abstraction for Unhindered Parallelism. In Proc. of the Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing, 2014, pp. 21-30.
[12] A.I. Legalov. A functional language for creating architecture-independent parallel programs. Computational technologies, vol. 10, no. 1, 2005, pp. 71-89 (in Russian) / А.И. Легалов. Функциональный язык для создания архитектурно-независимых параллельных программ. Вычислительные технологии, том 10, no. 1, 2005 г., стр. 71-89.
[13] O.V. Nepomnyashchy, A.I. Legalov et al. Methods and algorithms for a high-level synthesis of the very-large-scale integration. WSEAS Transactions on Computers, vol. 15, 2016, pp. 239-247.

[14] H.J. Nussbaumer. Fast Fourier Transform and Convolution Algorithms. Springer, 1982, 288 p. / Г. Нуссбаумер. Быстрое преобразование Фурье и алгоритмы вычисления сверток. Радио и связь, 1985 г., 248 стр.

[15] O. V. Nepomnyashchiy, I. N. Ryzhenko et al. The VLSI High-Level Synthesis for Building Onboard Spacecraft Control Systems. In Proc. of the Scientific-Practical Conference "Research and Development – 2016", 2017, pp. 229–238.

[16] O.V. Nepomnyashchy, I.N. Ryzhenko et al. Translator of architecture-independent description of automata and combinational circuits. Certificate of state registration of software for computers No. 2021610682, 02/01/2021 (in Russian) / О.В. Непомнящий, И.Н. Рыженко и др. Транслятор архитектурно-независимого описания автоматных и комбинационных схем. Свидетельство о государственной регистрации ПО для ЭВМ № 2021610682, 01.02.2021.

[17] A.A. Komarov, I.N. Ryzhenko, O.V. Nepomnyashchy Program for synthesizing circuit descriptions in HDL hardware description languages from the Pifagor functional-parallel programming language. Certificate of state registration of software for computers No. 2015619175, 08/26/2015 (in Russian) / А.А. Комаров, И.Н. Рыженко, О.В. Непомнящий. Программа синтеза описания схем на языках описания аппаратуры HDL с языка функционально-параллельного программирования «Пифагор». Свидетельство о государственной регистрации ПО для ЭВМ № 2015619175, 26.08.2015.

[18] O.V. Nepomnyashchy, A.A. Komarov et al. Program for the driver of the onboard network of the spacecraft. Certificate of state registration of software for computers No. 2015616896, 06/26/2015 (in Russian) / О.В. Непомнящий, А.А. Комаров и др. Программа драйвера бортовой сети космического аппарата. Свидетельство о государственной регистрации ПО для ЭВМ №2015616896, 25.06.2015.

[19] O.V. Nepomnyashchy, A.A. Komarov, I.N. Ryzhenko Complex-functional block of the lowering adder-limiter. Certificate of state registration of software for computers No. 2016619714, 08/26/2016 (in Russian) / О.В. Непомнящий, А.А. Комаров, И.Н. Рыженко. Сложно-функциональный блок понижающего сумматора-ограничителя. Свидетельство о государственной регистрации ПО для ЭВМ № 2016619714, 26.08.2016.

Information about authors / Информация об авторах

Darya Sergeevna ROMANOVA – post-graduate student of the Department of Computer Engineering of the Siberian Federal University, assistant of the Department of IT&MOIS of the Krasnoyarsk State Agrarian University. Research interests: parallel algorithms, high-performance systems.

Дарья Сергеевна РОМАНОВА – аспирант кафедры вычислительной техники Сибирского федерального университета, ассистент кафедры ИТиМОИС Красноярского государственного аграрного университета. Сферы научных интересов: параллельные алгоритмы, высокопроизводительные системы.

Oleg Vladimirovich NEPOMNYASHCHIY – Professor, Ph.D. in technical sciences, Head of the Computer Science Department at the Siberian Federal University. Research interests: microprocessor systems, high-level analysis and synthesis of complex single-chip systems.

Олег Владимирович НЕПОМНЯЩИЙ – профессор, кандидат технических наук, заведующий кафедрой вычислительной техники. Область научных интересов: микропроцессорные системы, высокоуровневый анализ и синтез сложных однокристальных систем.

Igor Nikolayevich RYZHENKO is an Assistant Professor at the Computer Science Department. Research interests: VLSI high-level synthesis technologies, digital signal processing.

Игорь Николаевич РЫЖЕНКО является ассистентом кафедры вычислительной техники. Сферы научных интересов: технологии высокоуровневого синтеза СБИС, цифровая обработка сигналов.

Alexander Ivanovich LEGALOV – Doctor of Technical Sciences, Professor of the Faculty of Computer Science. His research interests include programming technologies, evolutionary software development, architecture-independent parallel programming.

Александр Иванович ЛЕГАЛОВ – доктор технических наук, профессор факультета компьютерных наук. Его научные интересы включают технологии программирования,

эволюционная разработка программного обеспечения, архитектурно-независимое параллельное программирование.

Natalya Yurievna SIROTININA – Ph.D. in technical sciences, Associate Professor, Department of Computer Science. Her research interests include neural networks and parallel programming.

Наталья Юрьевна СИРОТИНИНА – кандидат технических наук, доцент кафедры «Вычислительная техника». Ее научные интересы включают нейронные сети и параллельное программирование.