The VLSI High-Level Synthesis for Building Onboard Spacecraft Control Systems

O.V. Nepomnyashchiy, I.V. Ryjenko, V.V. Shaydurov, N.Y. Sirotinina and A.I. Postnikov

Abstract Using small spacecrafts for a wide range of research and applied purposes is one of the major trends in the aerospace field. Modular-network architectures implemented on the "system-on-chip" hardware platform provide required characteristics of onboard control systems. Selecting this system architecture significantly increases demands on very large-scale integration (VLSI) design efficiency and project solution quality. In this paper, we propose a new approach to VLSI high-level synthesis based on a functional-flow parallel computing model. The modified VLSI design flow uses a functional-flow parallel programming language Pythagoras, which allows describing a VLSI operation algorithm with the maximal degree of parallelism. An offered intermediate representation of VLSI architecture in the form of a control-flow graph and a data-flow graph provides an opportunity for synthesizing circuits and verifying projects on the stage of a formal description, without returning to previous hierarchical levels of the project. A set of software tools supporting new design process is developed. The proposed technology is successfully tested on the example of a digital signal processing function. Further, this technology is suggested for use in the synthesis of onboard control system components for small spacecrafts.

O.V. Nepomnyashchiy (✉) · I.V. Ryjenko · V.V. Shaydurov · N.Y. Sirotinina · A.I. Postnikov
SibFU—Siberian Federal University, Krasnoyarsk, Russia
e-mail: 2955005@gmail.com

I.V. Ryjenko
e-mail: rodgi.krs@gmail.com

V.V. Shaydurov
e-mail: shidurov@icm.krasn.ru

N.Y. Sirotinina
e-mail: nsirotinina@sfu-kras.ru

A.I. Postnikov
e-mail: alpost@mail.ru

K.V. Anisimov et al. (eds.), *Proceedings of the Scientific-Practical Conference "Research and Development - 2016"*, https://doi.org/10.1007/978-3-319-62870-7_25

Keywords Small spacecraft · Control systems · Parallel computing
Data flow · Functional programming · Integrated circuit · Algorithm
Formal verification · High-level synthesis

Introduction

Using small spacecrafts for a wide range of scientific and applied problems is one of the major trends in the aerospace field. This is due to a number of reasons: development time reduction, relatively low cost, universal reconfigurable platform. An onboard control system is the main element in the spacecraft control. Currently accepted approaches to building onboard spacecraft control systems are based on a centralized hierarchical architecture. This approach does not meet modern requirements for onboard control systems. Modular-network onboard control system architectures have significant advantages over centralized. However, their implementation requires changing conceptual approaches to designing onboard control systems. Technological risks of introducing a qualitatively new architecture can be reduced using the latest achievements in the field of Russian microelectronics, as well as implementing new approaches to designing VLSI, which would reduce development time while increasing the reliability of onboard equipment.

Studying the subject area shows that the "system-on-chip" (SoC) technology allows implementing a functionally complete module set for the small spacecraft onboard control system on the base of a single VLSI chip. Benefits of using the SoCs as a hardware platform are the following:

- reducing the cost of VLSI production, especially in case of medium- and small-batch production,
- reducing the time-to-market cycle of new product,
- supporting flexible configurability of a system in accordance with requirements of the particular project,
- providing high product reliability through testing regular platform structure that performs by the manufacturer,
- debugging support,
- reducing power consumption.

The principle of extra-integration based on SoC technology provides an opportunity to optimize the internal structure and to reduce redundancy, which is typical of systems using universal components. This optimization determines the high economic efficiency of project decisions, both through direct savings (reducing the number of component on the printed board, board area and so forth), as well as by indirect savings (low energy consumption, high reliability, productivity, minimal amount of hardware debugging, etc.).

Problem Statement

Selecting the SoC as a hardware platform for small spacecraft onboard control systems significantly increases demands on VLSI development process efficiency and project solution quality. Modern computer-aided design tools and environments, despite the impressive achievements, have a number of significant problems. First, we should mention the global problem of a gap between a number of gates that could be implemented on a single chip and a number of items that can feasibly be designed and verified in a reasonable amount of time. For complex projects, verification and testing come to the fore. The end-to-end verification is important because late error detecting always turns into significant additional time and financial costs. The share of time spent on project verification comes up to 60–80% of the total development time [1, p. 51].

The analysis of the current problem state shows that designing complex single-chip systems requires a new, architecture-independent approach. On the one hand, it should provide separating an initial algorithm representation from a target hardware architecture; on the other hand, it should make possible translating the initial algorithm representation into the RTL-level with simultaneous verification. In this connection, the task of finding such a method for describing parallelism of initial algorithms that further could be "compressed" in accordance with resource-specified constraints of a target hardware platform is still relevant.

Related Works

Attempts to solve these problems are mainly aimed at eliminating the semantic gap between high-level- and low-level descriptions of the system being developed. Analyzing the current state of affairs in integrated circuit design flows, we outline the following ways of improving the initial representation of a VLSI operating algorithm:

1. Expanding Hardware Description Languages (HDL) with special constructions for high-level behavioral description.
2. Creating VLSI synthesis tools on the base of an existing high-level programming language, which is used in software development.
3. Creating a new high-level hardware description languages by way of synthesizing of HDL and high-level programming languages.

An example of the first approach is SystemVerilog [2, p. 7]. This hybrid language is formed from HDL Verilog by introducing high-level constructions for behavioral VLSI description.

The second approach is called High-Level Synthesis (HLS) [3, p. 72] supported by Xilinx company. This method is based on introducing a number of constructions into the basic high-level programming language for describing integrated circuit topologies.

The last option is implemented in languages such as HandelC [4, p. 18]. Also are known a number of projects on the use of functional programming languages for the

VLSI synthesis (Lava, Hume, Erlang) [5, p. 244; 6, p. 93; 7, p. 192]. The use of a functional language for this purpose is the most promising, since the algorithm description in a functional language is most suitable for a parallelization.

The algorithm representation in the form of a data-flow graph provides the most complete description of natural parallelism of an algorithm. The works of Dongarra [8, p. 36; 4, p. 3] convincingly demonstrate that solution of portability problem for parallel programming lies in changing programming paradigm. This paradigm [9, p. 6] must meet the following requirements:

- the absence of an imperative control on calculation process (control on the base of data readiness),
- the data-flow model,
- massive parallelism.

An effective implementation of the data-flow graph for VLSI, in particular for the FPGA platform, is discussed in [10, p. 6; 11, p. 12]. However, described models use explicit control of computation, are oriented on the limited application area (digital signal processing), do not support the portability, and are not used as a high-level languages for VLSI synthesis [10, p. 67].

Proposed Approach

The review of the subject area shows that existing methods do not meet modern demands. VLSI circuit design needs a new architecture-independent approach, which should provide an opportunity for automatic translating the initial algorithm representation from top to bottom with simultaneous verification and independence of the initial algorithm representation from the target hardware architecture. Another important requirement is the opportunity for initially describing the VLSI operation algorithm with maximum parallelism, which would be "reduced" during design in view of resource constraints of a target hardware platform.

The generalized scheme of data processing in a VLSI circuit contains a data signal set and a control signal set. The control signal set includes data ready signals. Each block of the circuit can process data independently and in parallel with other under condition of availability of data confirmed by the data ready signal. If an analogy with parallel programming languages, such a model corresponds to the data-flow model. Thus, effective solutions in this direction can be found using a functional parallel programming paradigm.

One of such models is the functional-flow model of parallel computing. Functional-flow parallel (FFP) programming language Pythagoras [12, p. 71] is designed on the basis of this paradigm. The language provides the initial description of a task with the maximum parallelism.

Portability of a parallel algorithm between different architectures in this case is achieved by a parallelism convolution.

The proposed technology of VLSI architecture-independent design is based on the FFP paradigm [12, p. 71; 13, p. 35]. It complements ESL VLSI design methodology.

Unlike existing HLS methods, the proposed methodology is based on the FFP description of VLSI functioning algorithm with the use of the Pythagoras language. The language allows a designer to create the architectural-independent single-chip system description in the form of parallel program with the maximal parallelism [12, p. 71]. Then, the initial algorithm is translated into data-flow and control-flow graphs. At the next step, HDL-graph is formed based on the mentioned graphs. As a result, the designer obtains a set of architectural solutions in terms of hardware description language with a given degree of parallelism [13, p. 35].

It is expected that the proposed approach provides an opportunity for effective conversions of a high-level representation, formal verification, and functional testing with maximum coverage of VLSI architecture on initial design stages. In addition, it simplifies Design Space Exploration (DSE) process. The main advantages of the methodology are the effective VLSI representation at higher levels of abstraction in ESL design, and as a consequence, the optimal architecture solutions at the gate level.

The authors propose HLS design flow for the VLSI synthesis based on FFP approach. The flow implies the following additional design stages:

The development of the VLSI operation algorithm with the use of FFP programming language. At this stage, the initial algorithm represents in architecture-independent high-level form with maximum degree of parallelism.

The formal verification, optimization and debugging of the FFP code. This stage includes the formal debugging of the VLSI operation algorithm according to the design task without binding to a target platform.

The synthesis of intermediate VLSI representation the in form of data-flow and control-flow graphs. In addition to the graph synthesis, this step provides the intermediate representation of argument types and constants in accordance with the initial program code and constrains.

Data typing and graph optimization. Data typing is performed according to the data type specifications obtained on the previous step.

The intermediate synthesis of data-flow and control-flow graphs. This stage is implemented in cases of transition to the corresponding phase of the conventional HLS flow or combination cycles of conventional and FFP synthesis.

The design space exploration. This stage aims at searching the optimal implementation of the VLSI operating algorithm taking into account the given hardware platform constraints. The problem is solved by reducing and transforming the algorithm parallelism in accordance with the resource constraints. The initial maximal-parallel representation of the algorithm provides the maximal coverage of the solution space and automated search of the optimal one.

The start and final stages: the problem statement, the formation of technical task and specification package, the synthesis of circuit and control signals, the VLSI synthesis on the register-gate level are fully comply with the conventional HLS flow [13, p. 35; 14, p. 239].

Tool Support for the Proposed High-Level Synthesis Flow

The initial description of the algorithm implemented on a VLSI is performed using an integrated development environment of FFP programming language Pythagoras. The Pythagoras language compiler provides the conversion of FFP functions into a set of reversive data-flow graphs (RDFG), describing the information dependences between operations. An additional utility generates a control-flow graph (CFG) using RDFG as input. CFG defines the order of the RDFG vertices execution. This approach provides a wide variety of computing control strategies, from serial to data flow. Selection of concrete computing control strategy defines by the developer. RDFG and CFG together form a program, which can be performed by Pythagoras interpreter for checking its correctness prior to RTL synthesis.

The developer can use additional tools to perform the following functions:

- RDFG and CFG optimization for improving FFP program effectiveness,
- debugging and code analysis at run time, thus providing error searching and tracing,
- formal verification.

These tools allow the developer to debug and test the initial algorithm prior to synthesis VLSI architecture.

A synthesizer transforms the intermediate FFP algorithm presented in the form of RDFG and CFG in accordance with the proposed HDL synthesis algorithm. At the first stage, it synthesizes the graph representation including type declaration and calculation, constant calculation. At the next stage, the synthesizer deletes the calculated graph vertices. The result of this stage is the optimized (minimized) graph representation. The final step includes generating input and output ports and a name (register) table, connecting input ports with registers and generating HDL description for the developed module in interpretation mode.

Results

Consider implementation of a typical digital signal filtering function—multiply accumulate. The function was created and debugged using the developed toolkit. The function implements multiplication of two argument pairs and summation of the multiplication results. The initial function description in FFP language is given below (Fig. 1).

The initial function translates into data-flow graph (Fig. 2).

The next stage is argument typing. The function receives as arguments two input lists containing two values each: ((int.16, int.16), (uint.15, uint.15)). This example uses two 16-bit and two 15-bit unsigned numbers.

The following phase is optimization. Calculating types, calculating, and removing optimizable vertices are executed in course of traversing graph in a

```
MultSum << funcdef Param {
 A <<Param:1;
 B <<Param:2;
 ((A:1,B:1):*,(A:2,B:2):*):+>>return;
}
```

Fig. 1 The initial function description in FFP language Pythagoras

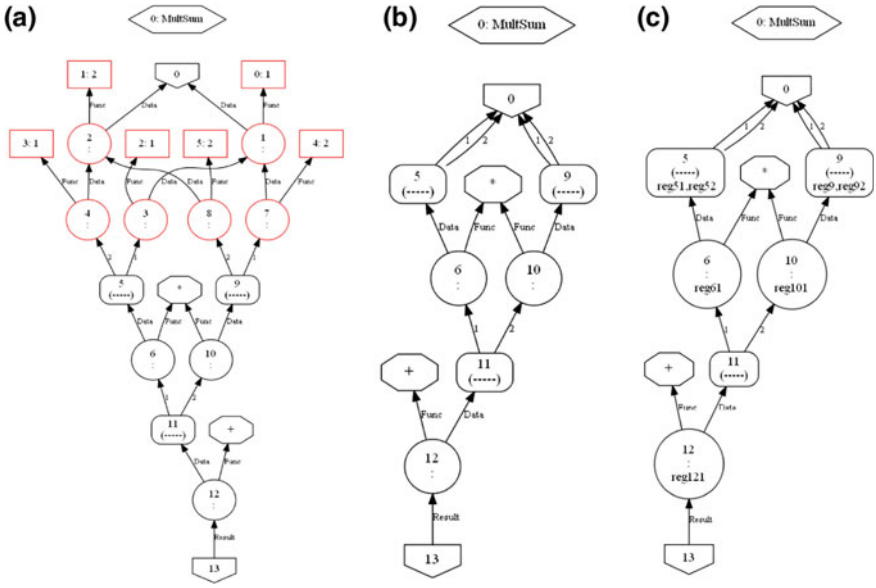

Fig. 2 **a** Initial reversive data-flow graph, **b** optimized data-flow graph, **c** typed HDL-graph

"symbolic interpretation" mode. Figure 2b shows the result of calculating and removing graph vertices.

At the next stage, a name table is generated in the following order. First, registers for all vertices related to the vertex of argument are generated and connected with input ports. Registers for vertices related to interpretation operation are generated at the next step. Then registers for vertices related to results are generated and connected with output ports. At last the name table is generated for the entire module as a whole (Fig. 3c)

The vertices of the graph (Fig. 3c) contain generated registers (table function names).

HDL description is synthesized on the basis of the optimized data-flow graph. Automatically synthesized HDL description in the Verilog language for this example is shown in Fig. 3.

Fig. 3 The function description in Verilog language

```
module MultSum
(
 input clk,
 input rst,
 input [15:0] in1,
 input [15:0] in2,
input signed [14:0] in4,
input signed [14:0] in5,
output reg signed [31:0] out1
);
reg signed [14:0] reg92;
reg signed [30:0] reg101;
reg signed [31:0] reg121;
reg signed [14:0] reg52;
reg signed [30:0] reg61;
reg [15:0] reg91;
reg [15:0] reg51;
begin
always @(posedgeclk)
begin : input_process
 reg51 <= in1;
 reg91 <= in2;
 reg52 <= in4;
 reg92 <= in5;
end
always @(posedgeclk) begin
 reg101 <= reg91 * reg92;
end
always @(posedgeclk) begin
 reg61 <= reg51 * reg52;
end
always @(posedgeclk) begin
 reg121 <= reg61 + reg101;
end
always @(posedgeclk) begin
out1 <= reg121;
end
endmodule
```

Note that the result of the synthesis presents the so-called “pure HDL”, i.e., the code without binding to a target platform. This code can be implemented based on any VLSI of any manufacturer, which meets project resource constraints.

Conclusion

A new methodology of VLSI synthesis on the base of FFP paradigm is proposed. This methodology extends the conventional VLSI HLS flow. Unlike existing HLS methods, the proposed methodology allows us to develop an initial description of the VLSI algorithm with the maximal parallelism. In addition, a high-level project description is translated into RTL automatically. These particular specificities coupled with architectural independence should enable a developer to perform formal verification, effective testing, debugging, and optimization. We assume that all mentioned advantages should improve effectiveness, reliability, and quality of developed projects and decrease time expenditures.

For implementing the proposed approach, the modified HLS design flow and appropriate tool support is developed. A series of test projects on creating graph

models, functional verification, and debugging, as well as generating HDL description with the following synthesizing RTL representation has been executed. Testing confirms the suitability of the proposed technology for SoC design. Further, this technology is suggested for use in the synthesis of onboard control system components.

Acknowledgements Researches are carried out with the financial support of the state represented by the Ministry of Education and Science of the Russian Federation. Agreement (contract) no. 14.578.21.0021 05.Jun 2014. Unique project Identifier: RFMEFI57814X0021.

References

1. Losev, V., Lumps And., Putr M. Institut of design of devices and systems (Cadence university): Quality, stability, future. Electronic components. **4**, S.49–54 (2008)
2. IEEE Std 1800-2012: IEEE standard for systemverilog-unified hardware design, specification, and verification language, 1275 p (2013)
3. Vivado design suite user guide. High-level synthesis. URL: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_4/ug902-vivado-high-level-synthesis.pdf
4. Handel-C Language Reference Manual, 348 p. Celoxica Limited (2005)
5. Alves, C.: Erlang inspired Hardware. In: International Conference on Field Programmable Logic and Applications, pp. 244–246. Milano (2010)
6. James-Roxby, P.: Lava and J Bits. From HDL to bit stream in seconds, pp. 91–100. In: 9th IEEE Symposium Field-Programmable Custom Computing Machines (2001)
7. Sérot J., Michaelson, G.: Compiling hume down to gates. In: Draft Proceedings of 11th International Symposium on Trends in Functional Programming, pp. 191–226. Madrid (2011)
8. Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Herault, T., Dongarra, J.: PaRSEC: Exploiting heterogeneity to enhance scalability. IEEE Comput. Sci. Eng. **15**(6), 36–45 (2013)
9. Bosilca, G., Bouteiller, A., Danalis, A., Dongarra, J.: Dataflow-based task execution through PaRSEC for HPC. In: IEEE 8th International Symposium on Embedded Multicore/ManycoreSoCs, pp. 1–52. Aizu-Wakamatsu, Japan (2014)
10. Ab Rahman, AAHB.: Optimizing dataflow programs for hardware synthesis. Ph.D. thesis, p. 170. EPFL, Lausanne (2014)
11. Johansen, J.: Design and Implementation of a Novel Dataflow Model and an Intermediate Representation Language for High Level Synthesis on Field Programmable Gate Arrays [Электронный ресурс]. URL http://projekter.aau.dk/projekter/files/71270246/main.pdf (датаобращения: 20.11.2016)
12. Legalov, A.I.: The functional language for creation of architecture-independent parallel programs. Comput. Technol **1**(10), 71–89 (2005)
13. Nepomnyashchy, O.V., Shaydurov, V.V., Legalov, A.I., Ryzhenko, I.N.: The technology of architecture-independent, high-level synthesis for VLSI, vol. 57, No. 3, pp. 35–39. Reports of Academy of Science of Russian Federation. Novosibirsk, NGTU (2014)
14. Nepomnyashchy, O., Legalov, A., Tyapkin, V., Ryzhenko, I., Shaydurov, V.: Methods and algorithms for a high-level synthesis of the very-large-scale integration. WSEAS Trans Comput **15**(Art. #22), 239–247 (2016)