ISSN No:-2456-2165

# Dataflow Computer Architecture Generator using Field Programmable Gate Array

Katarína Perželová, Branislav Madoš
Department of Computers and Informatics,
Faculty of Electrical Engineering and Informatics, Technical University of Košice,
Košice, Slovak Republic

**Abstract:- The paper deals with the effort to design dataflow computer architecture generator that is capable to generate dataflow computer architectures in the form of the hardware description in VHDL source code that can be implemented in Xilinx FPGA chip. In this work textual form of the dataflow graph is designed along with the generator which translates this dataflow graph into the description of the computer hardware in VHDL language.**

***Keywords***:- *Dataflow architecture, dataflow graph, VHDL, FPGA, Xilinx.*

## I. INTRODUCTION

The most common computing model (i.e., a description of how a program is to be evaluated) is the von Neumann control flow computing model. This model assumes that a program is a series of addressable instructions, each of which either specifies an operation along with memory locations of the operands or it specifies (un) conditional transfer of control to some other instruction.

A control flow computing model essentially specifies the next instruction to be executed depending on what happened during the execution of the current instruction. The next instruction to be executed is pointed to and triggered by the use of the program counter. This instruction is executed even if some of its operands are not available yet (e.g., uninitialized) [1][2][3].

The dataflow model represents a radical alternative to the von Neumann computing model since the execution is driven only by the availability of operands. It has no program counter and global updatable store, i.e., the two features of the von Neumann model that became bottlenecks in exploiting parallelism. The serialization of the von Neumann computing model has a serious limitation for exploiting more parallelism in nowadays microprocessors. In data flow computing parallelism is limited only by the actual data dependencies between instructions in the program.

Computer architectures with data flow computation control are very important alternative to the mainstream computer architectures in which control of program execution is represented by the flood of instructions. Although in recent years data flow principles of computing were in the background, today, when further increase of the performance of computers is achieved mainly by parallelization, dataflow architectures are again becoming important and it is leveraged again how to use them [4].

Computational models based on dataflow principles are providing possibility of using natural parallelism of program and it can significantly reduce the time that is necessary for execution of calculation. Advantage of this kind of computer architectures is in allowing detection of parallelism just on the machine instructions level [5]. A prototype of the dataflow computer was developed at the Department of Computers and Informatics. This prototype was verified and tested with the support of Xilinx WebPackISE software and Xilinx Spartan 3AN Development board [6][7][8][9][10].

The aim of this work is to design dataflow computer architecture generator which is capable to generate hardware description in VHDL language of the architecture of the dataflow computer. As the input of this generator the source code of the dataflow graph is used. It means that generator analyses program source code and translates it into computer hardware, described in VHDL language, which performs desired program.

A dataflow computer executes a program by receiving, processing and sending out data packages, called tokens. Each of them consists only of a value and a destination address. Processing starts when a set of matched tokens arrives and can be processed by the execution unit. In a dataflow computer, a program is not represented by a linear instructions sequence, but by a dataflow graph which is a directed graph consisting of named nodes that are representing instructions, and arcs that are representing data dependencies among instructions. During the execution of the program, data propagates along the arcs in tokens. This flow of tokens enables some of nodes to be fired and instructions that are in nodes to be executed [11]. The fig. 1 shows example of a dataflow graph.

## II. DATA FLOW COMPUTING PARADIGM

Dataflow paradigm of computation was popularized in 60’ and 70’ and describes non Von Neumann architectures with the ability of fine grain parallelism in computation process. In dataflow architecture the flow of computation is not instructions flood driven.

There is no concept of program counter implemented in this architecture. Control of computation is realized by data flood. Instruction is executed immediately in condition there are all operands of this instruction present. When executed, instruction produces output operands, which are input operands for other instructions.

Dataflow paradigm of computing is using directed graph

$$G = (V, E)$$

ISSN No:-2456-2165

called DataFlow Graph (DFG). DFG is used for the description of behavior of data driven computer. Vertex v form the set V is an actor, a directed edge e from the set E describes precedence relationships of source actor to target actor and is guarantee of proper execution of the dataflow program. This assures proper order of instructions execution with contemporaneous parallel execution of instructions without the need of program counter.

Tokens are used to indicate presence of data in DFG. Actor in dataflow program can be executed only in case there is a presence of a requisite number of data values (tokens) on input edges of an actor. When firing an actor execution, the defined number of tokens from input edges is consumed and defined number of tokens is produced to the output edges. An important characteristic of dataflow program is its ability to detect parallelism of computation. This detection is allowed on the lowermost basis – on the machine instructions level.

There are static, dynamic and also hybrid dataflow computing models. In static model, there is possibility to place only one token on the edge at the same time. When firing an actor, no token is allowed on the output edge of an actor. Disadvantage of the static model is impossibility to use dynamic forms of parallelism, such a loops and recursive parallelism. Computer with static dataflow architecture was designed by Denis and Misunas.

Dynamic model of dataflow computer architecture allows placing of more than one token on the edge at the same time. To allow implementation of this feature of the architecture, the tagging of tokens was established. Each token is tagged and the tag identifies conceptual position of token in the token flood. For firing an actor execution, a condition must be fulfilled that on each input edge of an actor the token with the same tag must be identified. After firing of an actor, those tokens are consumed and predefined amount of tokens is produced to the output edges of an actor. There is no condition for firing an actor that no tokens must be placed on output edge of an actor. The architecture of dynamic dataflow computer was first introduced at Massachusetts Institute of technology (MIT) as a Tagged Token Dataflow Architecture.

Hybrid dataflow architectures are the combination of control flow and data flow computation control mechanisms.

Dataflow computing is predominantly domain of the research laboratories and scientific institutions, and has limited impact on commercial computing nowadays because of difficulties which are connected to the cost of communication, organization of computation, manipulation with structured data and the cost of matching [12] [13]. Paradigm of tile computing in combination with the dataflow computing brings new possibilities to overcome some of deficiencies of dataflow architectures.

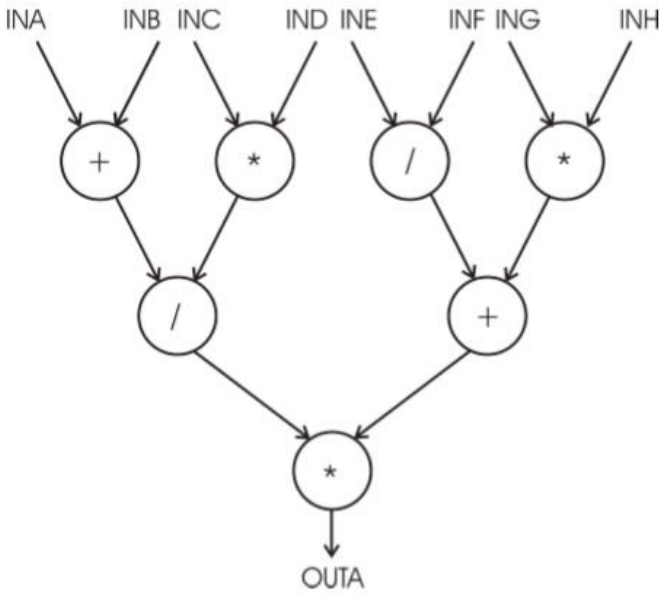


Fig 1:- Example of the dataflow graph which uses six inputs and one output and represents program for the computer with data driven computation model.

➢ *Design of Computer Architecture Generator*

The design of the computer architecture generator, which has been designed as the part of this research, consists of the generator of ad-hoc architecture that is implementing architecture on the basis of program source code. Structure of the source code is also defined in this work. Architecture designed in the work is unique by the reason of not using concept of operating memory. It is writing program directly to the computers structure and it is representing hardware realization of the particular dataflow program.

Individual instructions of program are realized by implementation of appropriate modules represented in VHDL code and those modules are used in the architecture as many times as the instructions in the program. For the proof of the concept the size of operands is limited to 8 bits and number of data types, on which individual operations can be executed, is limited to signed and unsigned integer types. This choice does not bound possibility of testing designed concept and in next phase of the development it is possible to increase the size of operands to 16, 32 or 64 bits and also to increase the number of operand types.

Design of architecture allows to define 8 input and 8 output ports, which are capable to make input or output of operands to resp. from the computer. Incoming and outgoing operands are in the size of 8 bits and each port has one other bit used as the indicator of validity of operand. This bit, along with the value of operand, represents the token.

*A. Syntax of the source code*

The source code contains dataflow graph written in textual form, which represents program executed by designed architecture. Its form is also designed as the part of this work and example of the dataflow program source code is shown on the Fig. 2.

ISSN No:-2456-2165

```
1 add A:5 B:INA D:A6
3 mul A:INB B:INC D:B6
6 SUB D:A2
2 SQR D:OUTA
```

Fig 2:- Example of the dataflow program source code

Content of the source code is not case sensitive. It consists of lower or upper case letters and lines, where each line represents one instruction and contains one of the following parameters, divided with blank space:

1. *Number of line* - each line has to be numbered and there is not allowed to use twice the same number, although numbering of lines is not necessarily ordered.
2. *Mnemonic shortcut of instruction* - the shortcut is a string and belongs to defined list of instructions.
3. *Operand A* - left input operand can have one of the following forms:

- Can be filled by generator while source code is analyzed and it is created as the result of some instruction execution. That is why it can stay blank in time of preparing the source code of the program.
- Defined by string 'A:' after which directly follows decimal number, which represents immediate operand used for execution of instruction.
- Defined by string 'A:' after which directly follows one of strings INA, INB, ..., INH that is representing one of the input port.

4. *Operand B* - right input operand can have one of the following forms:

- Can be filled by generator while source code is analyzed and it is created as the result of some instruction execution. That is why it can stay blank in time of preparing the source code of the program.
- Defined by string 'B:' after which directly follows decimal number, which represents immediate operand used for execution of instruction.
- Defined by string 'B:' after which directly follows one of strings INA, INB, INH that is representing one of the input port.

5. *Destination address* - address where the result of instruction is sent. It can have one of the following forms:

- Defined by string 'D:' after which directly follows one of the string A, B that represents address of the operand and it is followed by decimal number representing the number of instruction line.
- Defined by string 'D:' after which directly follows one of the string OUTA, OUTB, ..., OUTH that is representing one of the output ports.

*B. Generator of the hardware descriprion in VHDL*

The generator of the hardware description in VHDL language is designed as a console application which starting parameter is the name of the dataflow program source code file. This source code file is used by the generator as the input that contains dataflow graph in the necessary form. Generator determines which instructions are necessary to include into instructions package of generated ad-hoc architecture of dataflow computer.

The source codes representing modules that realize those instructions, are consequently connected to main module, which is representing source code of a computer's top module. Top module is in VHDL language and it is possible to use it as the source code for Xilinx software and to generate designed architecture that is implementable into the Xilinx FPGA chip.

The designed generator executes syntax and semantic analysis on the dataflow graph for the error detection. Also it is performing simple optimization of the source code with the aim to lower hardware demands in the final realization of computers hardware. The following figure 3 represents designed architecture of dataflow computer that is performing computation

OUTA = (INA + INB) * (INC - IND).

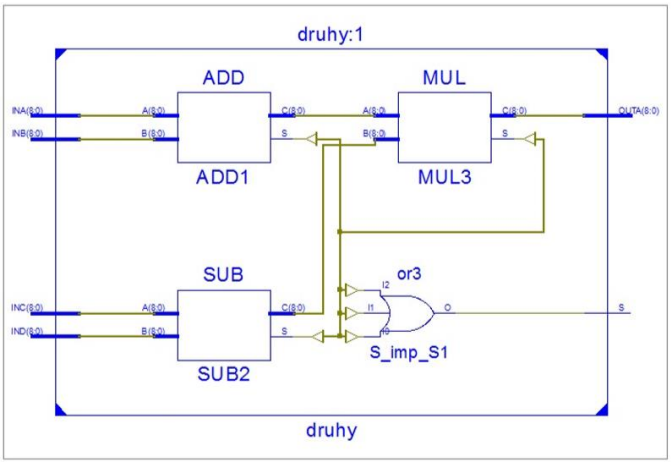


Fig 3:- RTL schema of the designed architecture representing particular program source code.

*C. Main principles of the design of ad-hoc architecture*

Among the main characteristics of designed computer architectures belongs that a computer memory is not used for saving instructions code of dataflow program and also there are no registers used for temporary storage of results of executed operations when they are transmitting data from the source to the destination instruction.

Designed architecture represents network, through which data flows via signal routes between the areas, where the dataflow program instructions are implemented. In order to monitor data flow over the network except the values of data, there are created signs (tokens) that indicate presence of operands in individual parts of network, which represents the dataflow graph.

Next important feature of the design is that only source code of those instructions, which are really used within

particular dataflow graph are included into description of the hardware. Because of this, instructions architecture of ad-hoc dataflow computer is always the subset of the overall instructions architecture designed in the work. This is very important because it can save hardware resources, when implementing designed computer hardware into FPGA.

For testing purpose, when hardware description in VHDL language was used and implemented into FPGA chip, the Xilinx Spartan 3AN Development Board was used along with the WebPackISE software.

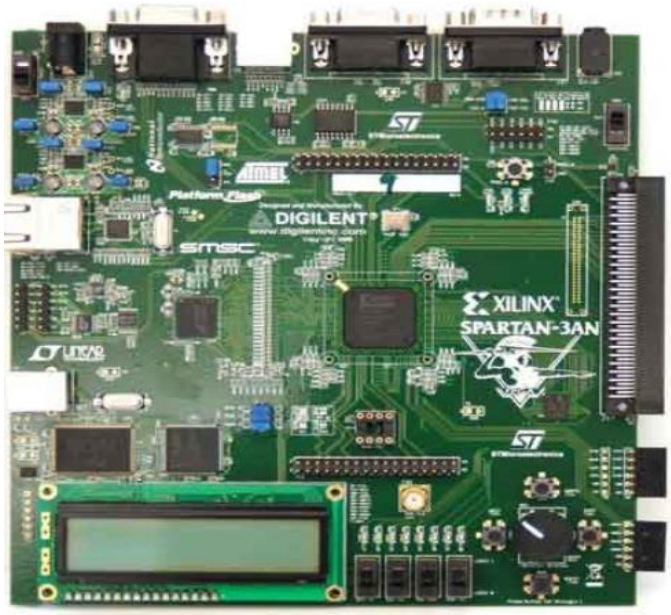


Fig 4:- Xilinx Spartan 3-AN Development Board using FPGA technology

## III. CONCLUSION

The paper introduces dataflow computing paradigm, which is driving the computation process with the flow of data, not with the flow of instructions, as control flow computers are doing. Static, dynamic and hybrid data flow computer architectures were introduced. In the next part of the paper syntax of the textual form of data flow graph, designed as the part of this work was briefly described and then the generator which function is to translate data flow graph (the source program code for data flow computer) into description of hardware in VHDL language. Generator literally translates program source code into equivalent hardware representation, which can be then implemented into Field Programmable Gate Array (FPGA) chip.

In the future research we will leverage the instructions set architecture from which instructions for ad-hoc hardware architectures are chosen. The architecture will be parametrized to allow from 8-bit to 64-bit operands and also the number of different operand types will be extended. Another important direction of the research will be focused on the optimization of generated architectures to allow translation of more complex source codes into their hardware counterparts with the aim to preserve architecture which is not using operating memory, registers and program counter concept and can optimally use the possibility of parallelization with accurate use of the hardware resources of particular FPGA chip.

## IV. ACKNOWLEDGMENT

This work was supported by KEGA Agency of the Ministry of Education, Science, Research and Sport of the Slovak Republic under Grant No. 003TUKE-4/2017 ”Implementation of Modern Methods and Education Forms in the Area of Security of Information and Communication Technologies towards Requirements of Labour Market” and under Grant No. 077TUKE-4/2015 „Promoting the interconnection of Computer and Software Engineering using the KPIkit“ and by the Slovak Research. This support is very gratefully acknowledged.

## REFERENCES

[1] M. Flynn, "Some computer organizations and their effectiveness", IEEE Trans. Computers, C-21 (1972), pp. 948 - 960.

[2] T. Agerwala and Arvind, "Dataflow systems", IEEE Computer, 15 (Feb. 1982), pp. 10 - 13.

[3] Arvind and D.E. Culler, "Dataflow architectures", Ann. Review in Comput. Sci., 1 (1986), pp. 225 - 253.

[4] L. Vokorokos, "Princípy architektúr počítačoov riadených tokom udajov", Copycenter Košice, 2002, 147s., ISBN 80-7099-824-5.

[5] N. Ádám, "Single Input Operators of the DF KPI System. Acta Polytechnica Hungarica" [online]. 2010, vol. 7, no. 1.

[6] L. Vokorokos, B. Madoš, A. Baláž and N. Ádám, "Architecture of multi-core computer with data driven computation model", Acta Electrotechnica et Informatica, december 2010, Košice, Slovakia, pp. 20-23, ISSN 1335-8243.

[7] B. Madoš and A. Baláž, "Data FLow Graph Mapping Techniques of Computer Architecture with Data Driven Computation Model", SAMI 2011 Proceedings of 9th IEEE International Symposium on Applied Machine Intelligence and Informatics, Smolenice, Slovakia, 27. - 29. january 2011, pp. 355 - 359, IEEE Catalog Number: CFP1108E-CDR, ISBN 978-1-4244-7428-8.

[8] L. Vokrokos, B. Madoš, N. Ádám and A. Baláž, "Priority of Instructions Execution and DFG Mapping Techniques of Computer Architecture with Data Driven Computation Model", SISY 2011: 9th IEEE International Symposium on Inteligent Systems and Informatics: 8. - 10.9.2011: Subotica, Serbia P. 483-488 Budapest : Obuda University, 2011.

[9] L. Vokorokos, B. Madoš, N. Ádám and A. Baláž, "Innovative Operating Memory Architecture for Computers using the Data Driven Computation Model", In: Acta Polytechnica Hungarica: Special Issue on Celebration of 60th Anniversary of the Foundation of Technical University of Koˇsice. Vol. 10, no. 5 (2013), p. 63-79. - ISSN 1785-8860.

[10] B. Madoš, "Architecture of Multi-Core System-on-the-Chip with Data Flow Computation Control", In:

International Journal of Computer and Information Technology (IJCIT). Vol. 3, no. 5 (2014), p. 958-965. - ISSN 2279-0764.

[11] J. Silc, B. Robic, T. Ungerer, "Processor architecture: from dataflow to seperscalar and beyond", Springer-Verlag, Berlin, New York, 1999.

[12] L. Vokorokos, N. Ádám, "Operators Matching in Dynamic Data Flow Architectures", Conference on Computation Cybernetics, Vienna, Austria, August 30 – September 1, Vienna, 2004, pp. 77–81, ISBN 3-902463-02-3.

[13] L. Vokorokos, N. Ádám, A. Baláž, "Flexible Platform for Neural Network Based on Data Flow Principles", HUCI, Budapest, November 18–19, Budapest Tech, 2005.