DOUBLE INPUT OPERATORS OF THE DF KPI SYSTEM

Liberios VOKOROKOS, Norbert ÁDÁM
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic, tel +421 55 602 2839,
e-mail: liberios.vokorokos@tuke.sk, norbert.adam@tuke.sk

ABSTRACT

Dataflow architectures can be used advantageously for computation-oriented applications that exhibit a fine grain parallelism. The implementation of the dataflow computer architecture depends on the form of execution of the dataflow program instructions, which is implemented as a process of receiving, processing and transmission of data tokens. The architecture described in this paper belongs to a class of dynamic dataflow architectures with direct operand matching. The concept of direct operand matching represents the elimination of the costly process (in terms of computing time) related to associative searching of the operands. This process is associated with the processing units of the proposed system. The processing units are designed as a dynamic multi-function pipelined unit of five segments, Load-Fetch-Operate-Matching-Copy. This pipeline stages handle processing of operand matching of dataflow operators. From the many types of operators, this paper describes microprogram managing for double input operators.

Keywords: *dataflow, operand matching, coordinating processor, double input dataflow operators*

1. INTRODUCTION

With the requirements on high performance, a great focus was given to a specific class of parallel computers in the 60's, denoted as dataflow architectures. In dataflow architectures the computing process is managed by the operands flow accessed on different levels for executing instructions of dataflow program. The dataflow computational model uses a dataflow graph, to describe a computation. This graph consists of nodes (vertices), which indicate operations, and arcs (edges) from one node to another node, which indicate the flow of data between them. Nodal operations are executed when all required information has been received from the arcs into the node. Typically, a nodal operation requires one, two or N ($N \geq 3$) operands (for conditional operations a Boolean input value) and produces one result. Hence one, two or N arcs enter a node and one arcs leave it. Once a node has been activated and the nodal operation performed (i.e. the node has fired) result is passed along output arc to waiting node or nodes, if the result is copied. This process is repeated until all of the nodes have fired and the final result has been created. Executing the program instructions can be done sequentially, in a flow (pipelining), parallel, or in different hybrid modes, depending on the used dataflow computational model. The fundamental idea behind the data flow computational model is the mapping of tasks to the computing elements, which can increase the rate of parallelism. Dataflow architectures can be used advantageously for computation-oriented applications that exhibit a fine grain parallelism. Examples of such applications are image processing, scene analysis, aerodynamic simulation, weather prediction etc. The dataflow concept has been utilized in the design of various processors [1], [2], [3], [4], [5], [6], [7], [8], [9]. The renewed interest in dataflow architectures is in part sparked by the underlying elegance of the model, but also motivated by the changes wrought by continued technology scaling [10]. The WaveScalar architecture [8] is an example.

2. ARCHITECTURE OF THE DF-KPI SYSTEM

The task the computer designer faces is a complex one: Determine what attributes are important for a new computer, then design a computer to maximize performance while staying within cost, power, and availability constraints. This task has many aspects, including instruction set design, functional organization, logic design, and implementation. The implementation may encompass integrated circuit design, packaging, power, and cooling. Optimizing the design requires familiarity with a very wide range of technologies, from compilers and operating systems to logic design and packaging [11]. The DF-KPI system [12], being developed at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice, has been designed as a dynamic system with direct operand matching. The architecture model of the DF-KPI computer is a part of a dataflow complex system, which includes support components for dataflow computing environment for the implementation of the defined application targets.

The structural organization (Fig. 1) of the DF-KPI computer architecture model consists of the following components:

Coordinating Processors (CP) are intended to manage, coordinate and process instructions of the dataflow program, based on the presence of their operands, which are enabled at the CP.DI input port of the coordinating processor - either from its CP.DO output port or from the CP.DO output ports of other CPs through an interconnection network, or from a Data Queue Unit and from the Frame Store. The structure of the CP is a dynamic pipelined multiple-function system.

The *Data Queue Unit* (DQU) is a unit designed to store the activation symbols (data tokens), which represent operands waiting for matching during program execution.

The *Instruction Store* (IS) is a memory of instructions of the dataflow program, in the form of a proper data flow graph.

The *Frame Store* (FS) is a memory of matching (pairing) vectors, by means of which the CP detects the presence of operands to perform the operation defined by the operator (node) in the data flow graph. The short description of the item format of MV matching vector in the FS is <FS>::= <AF><V>, where AF is a flag of the operand's presence (Affiliation Flag) and V is the value of the given operand.

Supporting components of the dataflow system are needed to create a realistic computing environment. In the given architecture they are formed by the following:

The *Main computer* (HOST) provides standard functions of the computer system during dataflow computing process.

The *Information Technology* unit is a unit used to create dedicated application environments (virtual reality, diagnostics, e-learning).

The *I / O processors* for fast direct inputs/outputs into the dataflow module (standard I/Os are implemented by the main computer).

The structure of the CP is a dynamic system with pipeline processing, composed of Load, Fetch, Operate, Matching and Copy segments also indicates the states of system.

The *Load* segment is used for loading the data token and its preparation for further processing in the Fetch segment. This segment is the first segment of the processor.

The *Fetch* segment reads the word DFI from the instruction memory IS. The word DFI defines the format of the dataflow instruction. This segment is reachable by the processor from the segment Load and Operate. In case that the segment Operate and segment Load requires the access to the Fetch segment a conflict occurs. The priority system decides which segment will be preferred.

The *Operate* segment handling the data token processing based on the operation code OC stored in word DFI. The execution units of the coordinating processor are accessed from this segment. The result of the operation is sent to the Load segment. In case that the result of the operation is intended to matching in the FS memory, then it is sent to the Matching segment. If the result is intended to Load and this segment is occupied, the token will be sent to another processor by the interconnection network, or will be saved into the DQU.

The *Matching* segment ensures matching of operands on the basis of flag in DFI.

Instructions needed for copying the operation results are processed in the *Copy* segment.

The format of the dataflow instructions is as follows:

$$\langle DFI \rangle ::= \langle OC \rangle \langle LI \rangle \langle \{DST, [IX]\}^{n} \rangle$$

Where OC is the operation code; LI is a literal (e.g. number of copies of the result); DST represents the target address for operation result; IX is a matching index for the operations.

The dataflow program instruction represented by a data token is stored in the Instruction Store at the address defined by DST field. The data token has the following format

$$\langle DT \rangle ::= \langle P \rangle \langle T, V \rangle \langle MVB \rangle \langle \{DST, [IX]\} \rangle$$

Where P is the priority of the data token; T represents the data type of operand with a value V; MVB defines a base address of matching vector in the Frame Store and

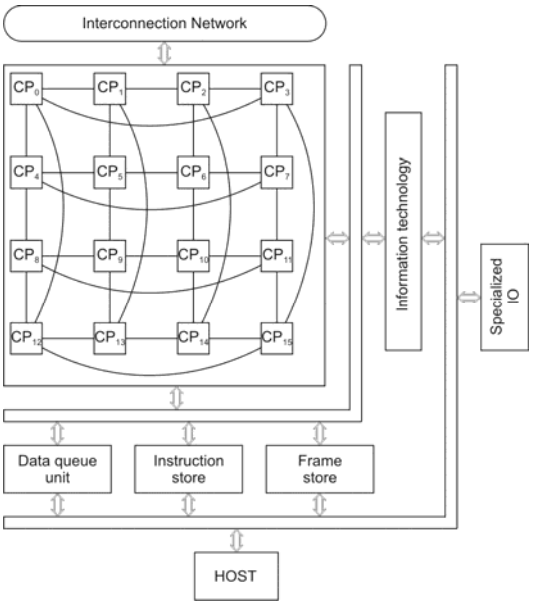

Fig. 1 The DF-KPI System

DST specifies a destination address of the resulting DT data token.

The structure of the DST field is the following

$$\langle DST \rangle ::= \langle MF \rangle \langle IP \rangle \langle ADR \rangle$$

Where MF is a matching function, with a defined set of labels {M, B}, M stands for matching (of two DTs), B stands for bypass (without DT matching); IP defines an input port {L(eft), R(ight)}; ADR is the address of the operator or function.

If the operands enter the two-input or multi-input operators, operand matching occurs. The DF-KPI architecture uses the direct operand matching control mechanism. It is based on the allocation of a Matching Vector in the Frame Store according to the activation code (procedure, call). Allocated Matching Vectors are represented as a matching record in the Frame Store.

The format of the Matching Vector in the Frame Store is as follows:

$$\langle FS[B_{ACT} + H + IX + 1] \rangle ::= \langle RC, MVS \rangle \langle B_{OLD} \rangle \langle DST_{RET} \rangle \langle D\{[B_{NEW}]\{D\}\} \rangle$$

Where B_{ACT} is a pointer to the current top record; H is the size of a header of record; MVS defines the size of a matching vector; RC is the reference counter; B_{OLD} is a pointer to the previous token; DST_{RET} specifies the return address; B_{NEW} defines the base address for new matching record and D represents an operand value.

The RC field is set according to the size of the matching vector at compile-time. After the function associated with the operator has fired, the value of RC is decremented. If RC = 0, the Matching Vector in the frame store is released.

3. OPERAND MATCHING

One of the most important steps based on the dynamic dataflow model is direct operand matching [12], [13]. The concept of direct operand matching represents the elimination of the costly process (in terms of computing time) related to associative searching of the operands. In this scheme, a matching vector is dynamically allocated in the Frame Store memory for each token generated during

the execution of the data flow graph. The current location of a matching vector in the Frame Store is determined at compile time, while the Frame Store location is determined after the program starts.

Each calculation can be described using an instruction address (ADR) and the pointer to the matching vector MVB in the Frame Store. The <MVB, ADR> value pair is part of the token. A typical action is the searching for the operands pair in the Frame Store. The matching function provides searching for the tokens marked identically. After the operand has arrived to the Coordinating Processor, the matching function detects if a commonly entered operand is present in the Frame Store. Detection is performed according to matching IX index. If the operand is not there yet, it is stored in the Frame Store, in the Matching Vector specified by base address of the MVB operand, into the item specified by index IX.

The operand matching process control at the operator input is influenced by the process of matching, instruction execution and generation of a result at its output. Using a compiler producing DFG output with forward searching that allows for the detecting and eliminating of redundant computations and change order of token processing, process control can be defined as the transition of activation signs along the edges of the data flow graph (Fig. 2), between the "producer" (P) operator and the "consumer" (C) operator.

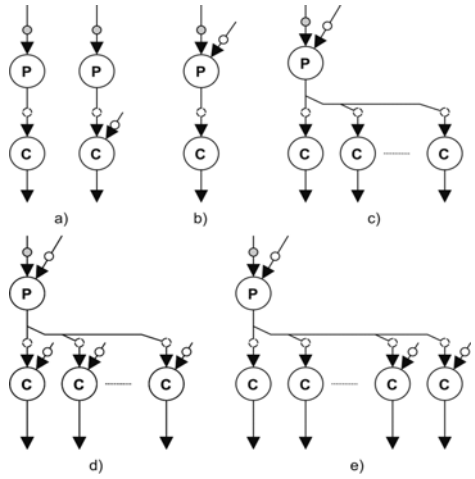

Fig. 2 The operand matching
(a – P-single input, b – P-double input, c – P-double input/C-single input, d – P-double input/C-double input, e – P-double input/C-u-single, v-double input)

In this article the proposed operand matching control for configuration shown in Fig. 2b is described.

3.1. Process Control for P-double Operators

The binary information stored in DF KPI system can be classified as either data or control information. The main components of this system are the CPs, wherein the data is manipulated in a datapath by using microoperations (microop), implemented with register transfers. These operations are implemented with adder/subtractors, shifters, registers, multiplexers and buses. The control unit of the CP provides signals that activate the various microop within the datapath to perform the specified processing tasks. The control unit of the CP also determines the sequence in which the various actions are performed.

The control unit that generates the signals for sequencing the microop is a sequential circuit with states that dictate the control signals for the system (Fig. 3).

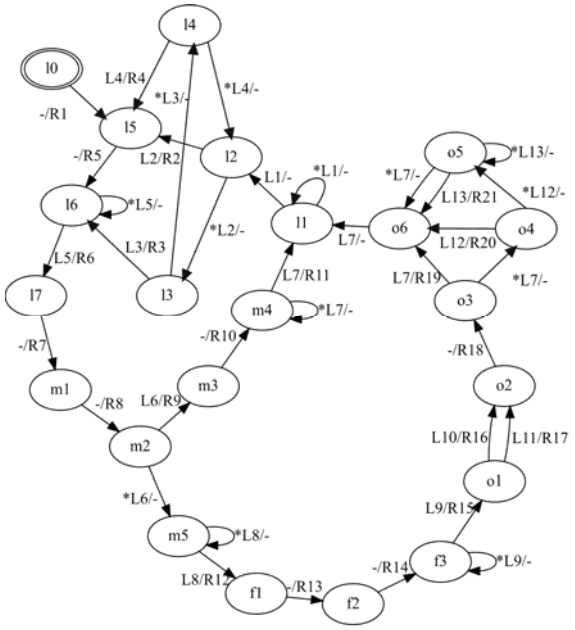

Fig. 3 State (Mealy) Machine

At any given time, the state of the sequential circuit activates a prescribed set of microop. Using status conditions and control inputs, the sequential control unit determines the next state. The digital circuit that acts as the control unit provides a sequence of signals for activating the microop and also determines its own next state.

The control unit of the CP allows transition between different states denoted as Load, Fetch, Matching, Copy, and Operate. These states (Fig. 3) are represented by different segments (l – Load, f – Fetch, m – Matching, c – Copy, o – Operate) of the control unit in different order. The segments work in overlapped manner. Transitions between segments are controlled by the microprogram.

Formal notation of the microprogram (which results from the functional system specification and its decomposition to operational/datapath and control unit) is represented by the program scheme. Program model scheme of the operational part is expressed as a sequence of marked pairs

$$n_i : p_i \tag{1}$$

Where n_i are labels, and p_i - are instruction(s).

Instructions p_i in respect to (1) display different elementary instructions – microinstructions, which initiate execution of different elementary operations – microoperations. Further following basic command types are defined

$$X / n \tag{2}$$

$$\alpha_1 : n_1, \alpha_2 : n_2, \ldots, \alpha_k : n_k \tag{3}$$

$$\alpha_1 : X_1 / n_1, \alpha_2 : X_2 / n_2, \ldots, \alpha_k : X_k / n_k \quad (4)$$

STOP (5)

where

$X, X_1, X_2, \ldots, X_k$ – operations or microoperations;
STOP – special operation;
$n, n_1, n_2, \ldots, n_k$ – labels;
$\alpha_1, \alpha_2, \ldots, \alpha_k$ – predicates (conditions), with the characteristics

$$\alpha_i \wedge \alpha_j = 0 \quad \text{pri} \quad i \neq j, \quad \overset{k}{\underset{i=1}{\forall}} \alpha_i = 1$$

Command (2) shows the microinstruction intended for the execution of microop, after which a transition is made to an instruction marked with the label *n*. Commands (3 – 4) represent control instructions (microinstructions) intended for execution of branch and conditional jumps in the program (microprogram). These commands test the condition defined by the predicate. If the condition α_i is valid a transition is made to an instruction marked with the label n_i (3) or an operation X is simultaneously executed (4).

Let the execution of the operations X_i be launched by the control word R_i consisting of the control signal sequence

$$R^i = \tilde{R}_{m,1} \tilde{R}_{m,2} \ldots \tilde{R}_{m,k}, \ \forall \tilde{R}_{m,j} \in \{0,1\}, j = 1, \ldots, k \quad (6)$$

Let the predicates α_p be represented by the status information word

$$L^p = \tilde{L}_{n,1} \tilde{L}_{n,2} \ldots \tilde{L}_{n,q}, \ \forall \tilde{L}_{n,r} \in \{0,1\}, r = 1, \ldots, q \, . \quad (7)$$

Then the micro program of executed P-double input operators DFG (Fig. 2b), through microop for the various stages of multifunctional pipeline unit has a form

$l_0 : R^1 / l_5$
$l_1 : L_1 / l_2, \overline{L_1} / l_1$
$l_2 : L_2 : R^2 / l_5, \overline{L_2} / l_3$
$l_3 : L_3 : R^3 / l_6, \overline{L_3} / l_4$
$l_4 : L_4 : R^4 / l_5, \overline{L_4} / l_2$
$l_5 : R^5 / l_6$
$l_6 : L_5 : R^6 / l_7, \overline{L_5} / l_6$
$l_7 : R^7 / m_1$
$m_1 :: R^8 / m_2$
$m_2 : L_6 : R^9 / m_3, \overline{L_6} / m_5$
$m_3 : R^{10} / m_4$
$m_4 : L_7 : R^{11} / l_1, \overline{L_7} / m_4$
$m_5 : L_8 : R^{12} / f_1, \overline{L_8} / m_5$
$f_1 : R^{13} / f_2$
$f_2 : R^{14} / f_3$
$f_3 : L_9 : R^{15} / o_1, \overline{L_9} / f_3$
$o_1 : L_{10} : R^{16} / o_2, L_{11} : R^{17} / o_2$
$o_2 : R^{18} / o_3$
$o_3 : L_7 : R^{19} / o_6, \overline{L_7} / o_4$
$o_4 : L_{12} : R^{20} / o_6, \overline{L_{12}} / o_5$
$o_5 : L_{13} : R^{21} / o_6, \overline{L_{13}} / o_5$
$o_6 : L_7 / l_1, \overline{L_7} / o_6$

The labels *l*, *f*, m, *o* represent the segments Load, Fetch, Matching and Operate. The microoperations and predicates of individual control words and status words are listed in Tab. 2 and Tab. 3.

The function isFree (X) tests the busy state of segment X. Micro-operations, which can be executed in parallel, are placed in a single command block (Tab. 1). Initialization of the coordinating processor is done by sign Init = 1. The boot command of the data flow program loads the data token from the DQU to the LOAD segment, sets the busy flag for the LOAD segment to 1 (i.e. the LOAD segment is occupied) and in next step blocks the processing of the following tokens (GetDT = 0). If the next segment, the Matching segment, is free, the token is loaded into the Load/Matching register. After that, the microprogram releases the LOAD segment, activates the loading of other tokens into the coordinating processor, and performs operand matching for defined double input operator.

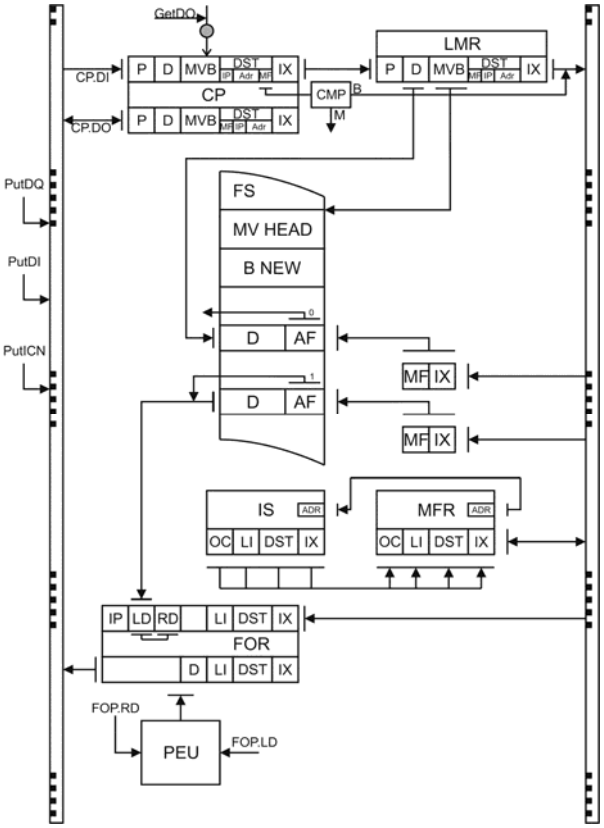

Fig. 4 Pipeline system for processing of double input operators

The control mechanism copies the content of the Load/Matching register to the Matching/Fetch register and determines the DF address operator based on the MFR.DST.ADR address. In the next step the operator will be loaded from the instruction store (IS) into the FOR.

If the Operate segment is not busy (isFree (Operate) = true), the operator is fetched from the Fetch/Operate register and processed. In the next step, if the CP is not busy, the result of the operator processing is available for processing in the same CP. Otherwise; the result is propagated to another CP through the interconnection network. If all CPs are busy, the token is stored in the DQU.

The proposed architecture at a logical level of operand matching control is show in Fig. 4. FIFO registers with the following specifications have been inserted to increase the throughput coefficient between the various stages of coordinating processor:

Between the stages L and M → register LMR
Between the stages M and F → register MFR
Between the stages F and O → register FOR

Table 1 Control words

Control words	Microoperations
$R^1 = R_{1,1}R_{1,2}R_{1,3}$	GetDT:=1 || Load_free:=0 || CP.DI:=DQU.OUT
$R^2 = R_{2,1}R_{2,2}$	PutDT:=0 || CPi.DI:=CPj.DO
$R^3 = R_{2,1}R_{3,1}$	PutDT:=0 || CP.DI:=ICN.OUT
$R^4 = R_{2,1}R_{4,1}$	PutDT:=0 || CP.DI:=DQU.OUT
$R^5 = \overline{R_{1,1}}$	GetDT:=0
$R^6 = R_{1,1}\overline{R_{1,2}}R_{6,1}R_{6,2}$	GetDT:=1 || Load_free:=1 || Matching_free:=0 || LMR:=CP.DI
$R^7 = R_{7,1}$	IS.ADR:=LMR.DST.ADR
$R^8 = R_{8,1}R_{8,2}$	FS.MVB:=LMR.MVB || FS.IX:=LMR.IX
$R^9 = R_{9,1}$	FS[MVB+H+IX+1].V:=LMR.D
$R^{10} = R_{10,1}$	FS[MVB+H+IX+1].AF:=1
$R^{11} = \overline{R_{1,2}}\,\overline{R_{6,1}}$	Load_free:=1 || Matching_free:=1
$R^{12} = \overline{R_{6,1}}R_{12,1}R_{12,2}$	Matching_free:=1 || Fetch_free:=0 || MFR:=LMR
$R^{13} = R_{13,1}$	IS.ADR:=MFR.DST.ADR
$R^{14} = R_{14,1}R_{14,2}R_{14,3}$	FOR.OC:=IS[ADR].OC || FOR.DST:= IS[ADR].DST || FOR.LI:= IS[ADR].LI ||
$R^{15} = \overline{R_{12,1}}R_{15,1}R_{15,2}$	Fetch_free:=1 || Opearte_free:=0 || FS.MVB:=L.DI
$R^{16} = R_{16,1}R_{16,2}$	FOR.LD:= FOR.D || FOR.RD:= FS[MVB+H+IX+1].V
$R^{17} = R_{17,1}R_{17,2}$	FOR.LD:=FS[MVB+H+IX+1].V || FOR.RD:=FOR.D
$R^{18} = R_{18,1}R_{18,2}R_{18,3}$	CP.DO.D:= PEn(FOR.LD,FOR.RD) || CP.DO.DST:=FOR.DST || CP.DO.IX:=FOR.IX
$R^{19} = R_{1,1}\overline{R_{2,1}}\,\overline{R_{15,1}}$	GetDT:=1 || PutDT:=1 || Operate_free:=1
$R^{20} = R_{1,1}R_{20,1}R_{20,2}$	GetDT:=1 || PutICN:=1 || ICN.IN:=CP.DO
$R^{21} = R_{1,1}\overline{R_{2,1}}\,\overline{R_{15,1}}R_{21,1}$	GetDT:=1 || PutDT:=1 || Operate_free:=1 DQU.IN:=CP.DO

Table 2 Status words

Status words	Microoperations
L_1	GetDT = 1
L_2	PutDT = 1
L_3	isEmpty(ICN) = 0
L_4	isEmpty(DQU) = 0
L_5	isFree(matching) = 1
L_6	FS[MVB+H+IX+1].AF= 0
L_7	isFree(Load) = 1
L_8	isFree(Fetch) = 1
L_9	isEmpty(Operate) = 1
L_{10}	FOR.IP = L
L_{11}	FOR.IP = R
L_{12}	isEmpty(ICN) = 1
L_{13}	isFree(DQU) = 1

4. CONCLUSIONS

The efforts to construct high-performance systems leads to constant research and using the benefits of chosen architecture or even to a development of a new, in which the increase of performance is expected, to suit the current demands for power. One of many scopes of computer architecture and principles of its work is the way of organizing and managing the computing process – information processing.

Computation process is represented by a sequence of states, in which changes are due to execution of program instructions - by this fact the computational process of the computer is organized.

The organization of computational process introduces the description of these changes by defining the conditions for executing instructions and their consequences. From the organization of computational process a managing process results. The proposed architecture described in this article is based on dataflow computational model, in which the computation is controlled on the basis of dataflow processing. Essential element of DF architecture is coordinating processor (CP), which is responsible for managing and organizing the execution of instructions. Structural organization of CP is designed as a dynamic multifunctional system.

By executing operations CP can pass through different states, which results that this unit is a dynamic flow system. Transitions and the order of transitions between different states are defined by the type of executed operator by interpreting flow of operands.

From the many types of operators, this article provides microprogram managing for double input operators. The currently developed DF-KPI system with its principle of data processing and its parameters is intended for solving tasks requiring brute force. The development of the DF-KPI system is focused on the fields of virtual reality [14] and computer security [15], [16], [17].

ACKNOWLEDGMENTS

This work is the result of the project implementation: Development of the Center of Information and Communication Technologies for Knowledge Systems (ITMS project code: 26220120030) supported by the Research & Development Operational Program funded by the ERDF.

REFERENCES

[1] GYÖRÖK, GY. – MAKÓ, M. – LAKNER, J.: Combinatorics at Electronic Circuit Realization in FPAA, *Acta Polytechnica Hungarica*, Vol. 6, No. 1, pp. 151–160, 2009.

[2] ARVIND–CULLER, D. E.: Dataflow Architectures, *Annual Review in Computer Science*, Vol. 1, pp. 225–253, 1986.

[3] CARLSTRÖM, J. – BODÉN, T.: Synchronous Dataflow Architecture for Network Processors, *Micro IEEE*, Vol. 24, Issue 5, pp. 10–18, Sept. – Oct. 2004.

[4] DENNIS, J. B.: Data-Flow Supercomputers, *Computer*, pp. 48–56, Nov. 1980.

[5] GURD, J. R. – KIRKHAM, C. C. – WATSON, I.: The Manchester Prototype Data-Flow Computer, *Commun. ACM*, Vol. 28, pp. 34–52, Jan. 1985.

[6] JAMIL, T. – DESHMUKH, R. G.: Design of a Tokenless Architecture for Parallel Computations Using Associative Dataflow Processor, *Proc. of Conf. on IEEE SOUTHEASTCON ´96, Briging Together Education, Science and Technology (Cat. No. 96CH35880)*, Tampa, FL, USA 1996, pp. 649 – 656.

[7] ŠILC, J. – ROBIČ, B. – UNGERER, T.: Asynchrony in Parallel Computing: From Dataflow to Multithreading, *Parallel and Distributed Computing Practices*, Vol. 1, pp. 57–83, 1998.

[8] SWANSON, S. – MICHELSON, K. – SCHWERIN, A. – OSKIN, M.: WaveScalar, *Proc. of the 36th International Symposium on Microarchitecture (MICRO-36 2003),* 2003, pp. 291–302.

[9] VERDOSCIA, B. – VACARRO, R.: ALFA: A Static Data Flow Architecture, *Proceedings of Fourth Symposium on the Frontiers of Massively Parallel Computation*, McLean, VA, USA, 1992, pp. 318–325.

[10] MADOŠ, B.: Design of dataflow computer architecture with tile organization, *SCYR 2009: 9th Scientific Conference of Young Researchers*, Košice: FEI TUKE, pp. 207–209, 2009.

[11] HENNESSY, J. L. – PATTERSON, D. A.: *Computer Architecture – A Quantitative Approach*. San Francisco, CA: Morgan Kaufmann, 4th edition, p. 704, 2006.

[12] JELŠINA, M.: *Design of Data Flow KPI Computer System* (in Slovak). Košice, SR: elfa s.r.o., p. 214, 2004.

[13] VOKOROKOS, L.: *Data Flow Computer Principles* (in Slovak). Košice, SR: Copycenter, spol. s.r.o., p. 147, 2002.

[14] SOBOTA, B. – PERHÁČ, J. – STRAKA, M. – SZABÓ, CS.: *The Applications of Parallel, Distributed and Network Computer Systems to Solve Computational Processes in an Area of Large Graphical Data Volumes Processing* (in Slovak). Košice, SR: elfa s.r.o., p. 178, 2009.

[15] VOKOROKOS, L. – ÁDÁM, N. – BALÁŽ, A. – PERHÁČ, J.: High-Performance Intrusion Detection System for Security Threats Identification in Computer Networks, *Computer Science and Technology Research Survey*, Vol. 4, pp. 54–61, 2009.

[16] BALÁŽ, A. – TRELOVÁ, J. – KOSTRÁB, M.: Architecture of distributed intrusion detection system based on anomalies, *INES 2010: 14th international conference on Intelligent Engineering Systems: proceedings,* Las Palmas of Grand Canaria, Spain., pp. 79–83, 2010.

[17] AUGUSTÍN, M. – BALÁŽ, A.: Intrusion detection with early recognition of encrypted application, *INES 2011 : 15th IEEE International Conference on Intelligent Engineering Systems,* Poprad, High Tatras, Slovakia, pp. 245–247, 2011.

Received October 3, 2011, accepted December 15, 2011

BIOGRAPHIES

Liberios Vokorokos (prof., Ing., PhD.) was born on 17.11.1966 in Greece. In 1991 he graduated (MSc.) with honours at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. He defended his PhD. in the field of programming device and systems in 2000; his thesis title was "Diagnosis of compound systems using the Data Flow applications". He was appointed professor for Computers Science and Informatics in 2005. Since 1995 he is working as an educationist at the Department of Computers and Informatics. His scientific research is focusing on parallel computers of the Data Flow type. In addition to this, he also investigates the questions related to the diagnostics of complex systems. Currently he is dean of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice. His other professional interests include the membership on the Advisory Committee for Informatization at the faculty and Advisory Board for the Development and Informatization at Technical University of Košice.

Norbert Ádám (Ing., PhD.) was born on 30.8.1980. In 1998 he graduated (MSc.) with distinction at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice. He defended his PhD. in the field of Computers and computer systems in 2007; his thesis title was "Contribution to simulation of feed-forward neural networks on parallel computer architectures". Since 2006 he is working as a professor assistant at the Department of Computers and Informatics. Since 2008 he is the head of the Computer Architectures and Security Lab. at the Department of Computers and Informatics. His scientific research is focusing on the parallel computers architectures.