Innovative Operating Memory Architecture for Computers using the Data Driven Computation Model

Liberios Vokorokos, Branislav Madoš, Norbert Ádám, Anton Baláž

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia
{liberios.vokorokos, branislav.mados, norbert.adam, anton.balaz}@tuke.sk

Abstract: This paper deals with the data flow computing paradigm, with the characteristics of program execution control using a flow of data instead of a flow of instructions. Data flow computing represents an important alternative to the control flow computing paradigm, which is currently the mainstream computing paradigm represented by architectures mostly based on the Von Neumann principles, in which the flow of program execution is controlled by a flow of instructions. This paper also deals with the tile computing paradigm – a modern approach to designing multi-core microprocessors with components laid out in two-dimensional grids with various geometries of cores, memory elements and interconnection networks with architectures using both data flow and control flow program execution control. In this paper we introduce a data flow computer architecture designed at the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia, employing the principles of tile computing. Efficient data flow architectures applying a different approach to the control of program execution flow also demand a different operating memory design. As the part of our effort to build data flow computer architectures, we have designed an innovative memory architecture that fulfils the specific needs of data flow computer architectures. In the final section of the paper we describe the functions and operations possible with this kind of memory.

Keywords: tile computing; data flow computing; data driven architecture; data flow graph; operating memory

1 Introduction

In the previous decades – starting from the middle of the last century – we have witnessed a tremendous expansion of the computer and electronics industry, connected to a long-term, unprecedented increase in the communication and computing performance of integrated circuits, accompanied with the increase of memory chip capacities. This trend was driven by the increase of the number of basic building blocks [1] integrated onto the silicon chips and also by the increase of the density of their integration. This situation has brought the industry to the opportunity to gradually increase the communication and computing performance of superscalar monolithic uniprocessors. Unfortunately, it is not easy to use this opportunity to appropriately increase the computing performance of monolithic uniprocessors parallel to the increase of the number of transistors integrated on chips.

Microprocessor designs reflect the problem outlined above using the multi-core approach or single-chip multiprocessors [2]. Commercially available microprocessors are integrating more than one complex superscalar core on a single chip.

Other strategies are promoting downsizing of the superscalar core reducing its complexity and footprint in favour of the possibility to integrate even more cores on the chip. The core complexity ratio and the number of cores integrated onto the chip can therefore become an important parameter of multi-core microprocessor architectures. The future will bring the integration of not only tens, but hundreds and even thousands of cores. Integrating memory onto a single chip with the processing elements may become a further step in avoiding bottleneck problems related to accessing operating memory.

Researchers evaluate various layouts of cores, memory elements and especially interconnection networks, along with various program execution organization and control approaches [3]. Not only control flow driven architectures, along with Very Long Instruction Word (VLIW) approaches are being used, but the possibilities of architectures using the data driven computation model, called data flow architectures, are being continually investigated, too [4] [5] [6] [7] [8] [9] [10] [11].

A very important approach to microprocessor architecture design is tile computing, introduced in [12]. One can describe tile computing as the use of standardized components – including processing elements (small and simple processing elements, arithmetic and logic units or more complex cores), memory elements and modules, as well as various types of communication networks – in easily and highly scalable microprocessor and computer-on-a-chip architectures, where scaling the architecture up requires only minimal changes to the components in use and the architecture as such.

Tens of cores are being integrated not only in experimental architectures, but also in commercially available microprocessors – with more than hundred cores in some cases.

There are commercially available, as well as research and experimental processors and computer-on-the-chip architectures, in which the control of program execution is done by means of a control flow and a data flow; besides general purpose architectures there are also specialized designs, such as digital signal processors.

Architectures range from 16 to 121 cores, such as TilePro64 (Figure 1), TeraScale, VGI, TRIPS, Monarch and others [13] [14] [15] [16] [17] [18].

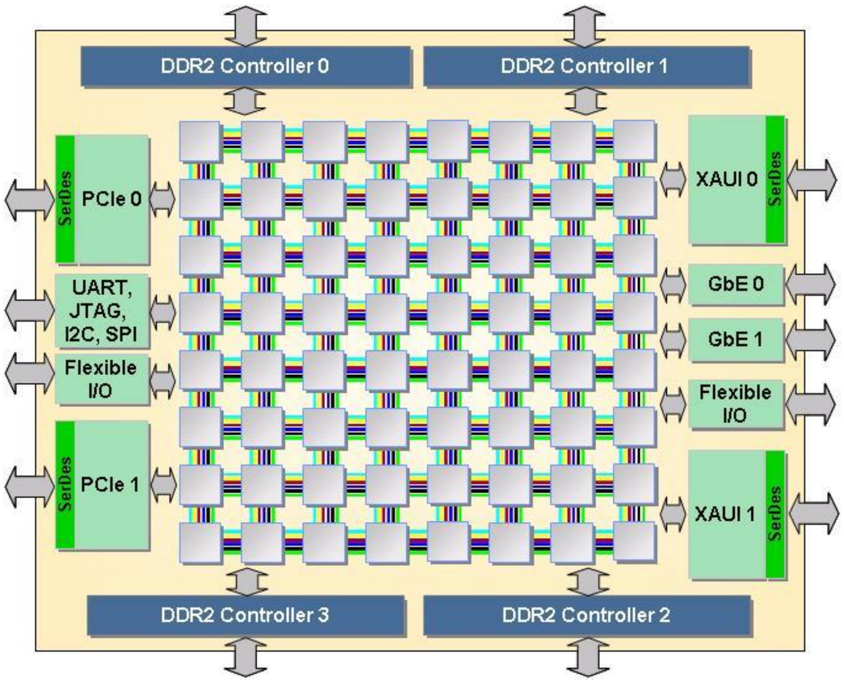

Figure 1
Tilera TilePro64 Block Diagram. Source: Tilera Corporation.

2 The Proposed Architecture

The architecture of the dataflow computer, proposed as a part of our research is a 32-bit computer-on-the chip with program execution driven by the flow of operands, according to the dataflow computing paradigm. The architecture was designed with the principles of tile computing in mind, with the aim to prepare a highly scalable design that could be scaled up in terms of integration of further processing elements and other components of the architecture on the chip with minimal changes to the design. One may scale this architecture up not only by redesigning the chip, but also by using the proposed computer-on-the-chip in a multichip configuration – in which chips are placed in a bi-dimensional grid layout, forming a consistent array of processing elements.

2.1 Elements of the Chip

The proposed computer architecture utilizes small elements with a simple design to allow the integration of as many elements as possible, in accordance with the number of transistors available on the chip. The basic element of the architecture is the Processing Element (PE), executing the dataflow program. Each processing element contains an integrated operating memory, used as the Activation Frames Store (AFS). All PEs are in a bi-dimensional grid, forming a Processing Array (PA). Other components integrated on the chip are Input/Output Elements (I/O) interconnecting PEs with the neighbouring components of the computer (Fig. 2).

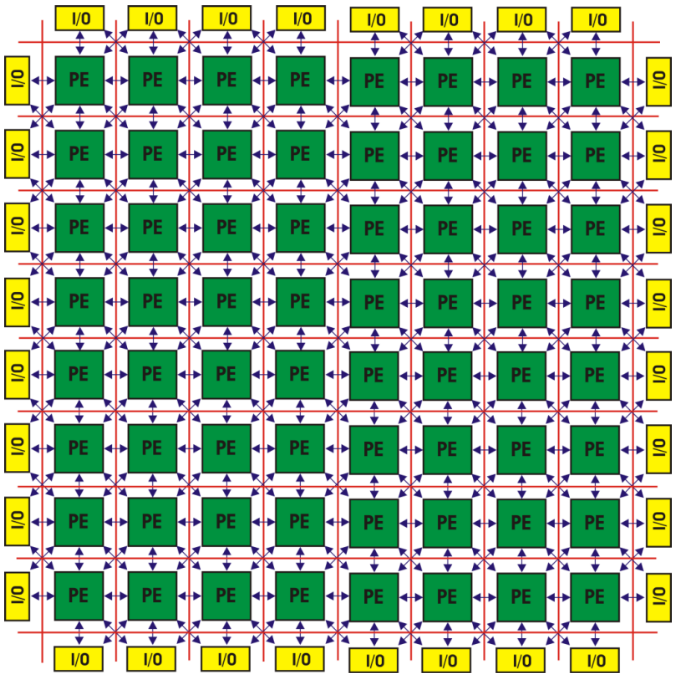

Figure 2
The processing array containing 64 processing elements, located in a two-dimensional, 8 × 8 layout, 32 input/output elements, communication channels intended for short-range communication (SCN) and the bus intended for global communication (GCN).

In the prototype built using Xilinx Field Programmable Gate Array (FPGA) technology, 64 processing elements (PE) are accompanied with 32 input/output elements (IO). The ratio (R) of the number of the integrated processing elements and the input/output elements may be calculated using formula (1). The ratio increases with the value of *n*.

$$R = \frac{n^2}{4n} \tag{1}$$

where

n is the number of PEs forming the edge of PA

2.1.1 Processing Elements

Each processing element (Figure 3) has a simple design and it consists of the following:

- an *Activation Frames Store (AFS):* a unit consisting of the operating memory developed as the part of the present research. The AFS has a capacity of 1024 × 144 bits. It is not only used as the instruction store but it also performs the Fetch phase of instruction cycle and it is capable of other operations, as described in section 3 of this paper.
- an *Arithmetic and Logic Unit (ALU):* the unit performing the Execute phase of the instruction cycle of integer arithmetic and logic operations.
- a *Control Unit (CU):* the unit controlling the execution of instructions and the communication with the neighbouring elements.

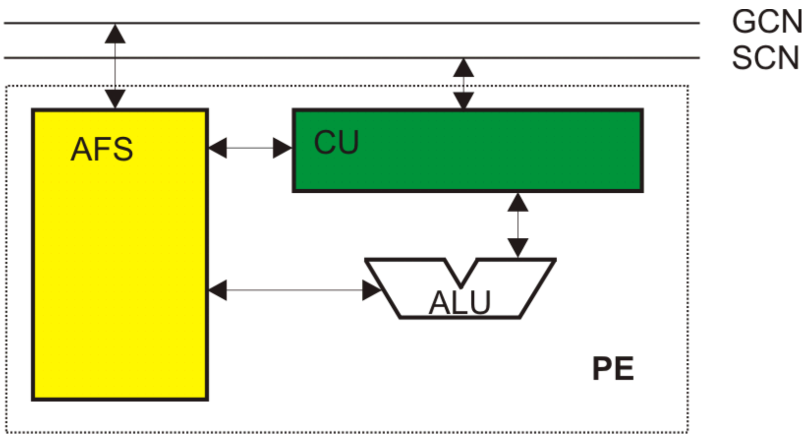

Figure 3
The structure of the processing element (PE), consisting of the Activation Frames Store (AFS), the Arithmetic and Logic Unit (ALU) and the Control Unit (CU), connected to the Global Communication Network (GCN) and a set of Short-range Communication Network (SCN) channels

Each activation frame stored in the AFS is addressable and all activation frames of all PEs integrated on the chip form a virtual cube of addressable space of the computer with addresses consisting of three components: X(2:0), Y(2:0) and Z(9:0). The X and Y components of the memory address, integrated in the processing element of the processing array and the Z component represents the address of the activation frame in the AFS of the particular PE. Each PE is allowed to address only activation frames stored in operating memory integrated in the particular PE; it is not allowed to address activation frames of other PEs.

2.1.2 Input/Output Elements

Processing elements situated on the edges of the processing element array are connected to the input/output elements surrounding this processing array. Each input/output element allows communication with other components of the computer system, when serving as the entry point of operands produced beyond the processing array, or as the exit point, when operands are leaving processing array and are used as inputs of other components of the computer system.

In a multichip configuration, two input/output elements, each on different chip form an I/O bridge that interconnects two processing elements of different chips and allows the transmission of operands between the chips. An I/O bridge forms an interconnection with the same function as a local network communication channel (Figure 4).

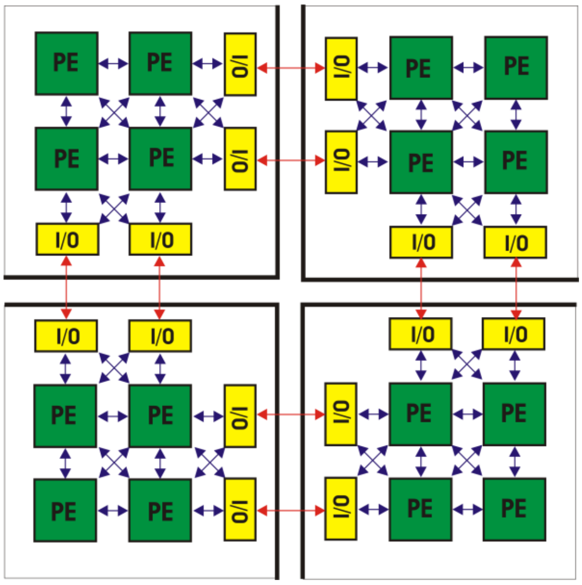

Figure 4

The multichip computer configuration allows scaling the computer beyond the borders of the chip. This figure presents four chips in a bi-dimensional layout, interconnected by I/O elements.

2.2 Communication Networks

To produce an efficient computer architecture design with the tile computing principles in mind, one also has to focus on the efficient communication of computer components. In the proposed architecture we have created two types of communication networks: for short-range and global-range communication.

Short-range communication (SCN) is performed by means of communication channels interconnecting each PE with each of its eight neighbouring elements. This allows pairs of chip elements to communicate in parallel; the high bandwidth short-range parallel communication of chip elements is implemented as follows: each PE is allowed to communicate with one of its neighbouring elements by passing tokens. A token consists of a 32-bit operand accompanied by a 10-bit target activation frame address and a 1-bit target operand flag (Figure 5). Short-range communication is used in *Compute Mode*, when the data flow program is executed.

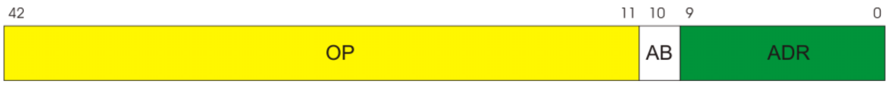

Figure 5
Each token consists of a 32-bit operand (OP), a 1-bit target operand flag (AB) and a 10-bit activation frame address in the AFS (ADR)

Global-range communication (GCN) is implemented as a bus interconnecting all PEs, allowing data flow graph mapping onto the operating memory of each PE in three different manners developed as the part of the present research, as described below. This communication network is used in *Load Mode*, when the data flow graph is mapped onto the operating memory of each of the processing elements.

2.3 Data Flow Graph Mapping

A unique characteristic of the proposed architecture is that in *Load Mode* it is possible to map the data flow program as a data flow graph into the activation frames stores of the processing elements; this is done by means of the global range communication network, utilizing three different approaches. One can switch among these dynamically, immediately after each activation frame is mapped.

The first mode is *sequential mapping;* this allows mapping the instructions onto activation frames in the traditional way: one activation frame at once. Concurrent instruction mapping to the different AFS's of different PEs is not possible. All three components (X, Y and Z) of the activation frame are used.

The second mode, *global multi-mapping* allows mapping the same instruction onto all AFS's of all PEs in the PA concurrently in one machine cycle; the activation frame is addressed using the Z component of the address only.

The third mode, *mask multi-mapping mode* allows concurrent mapping of the same instruction into the AFS's of selected PEs of the PA. The PEs may be selected using a mask. The Z-component of the address, along with two 8–bit vectors representing the mask are used to specify which activation frames are targeted.

One may switch between the data flow graph mapping modes during the process to optimize the execution time of data flow graph mapping process. The mapping techniques are described in detail in [19].

2.4 Activation Frame

Instructions are represented as 144-bit vectors, mapped as activation frames onto the activation frame stores of the particular processing elements at time of data flow graph mapping, when the computer and the memory are in *Load mode*. Each instruction vector consists of components specified in Figure 6.

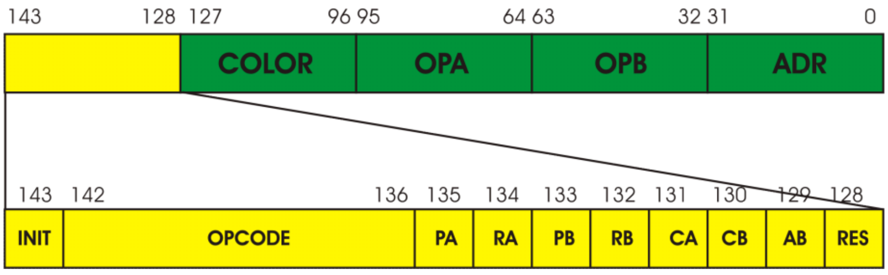

Figure 6
Instructions are represented by 144-bit vectors stored in activation frames

where:

INIT is a 1-bit instruction execution flag;

OPCODE is the 7-bit operation code of the instruction;

PA is a 1-bit flag indicating that operand A is required for instruction execution;

RA is a 1-bit flag indicating the availability of operand A;

PB is a 1-bit flag indicating that operand B is required for instruction execution;

RB is a 1-bit flag indicating the availability of operand B;

CA is a 1-bit flag indicating immediate operand A;

CB is a 1-bit flag indicating immediate operand B;

AB is a 1-bit flag indicating the target operand instructions;

RES is a 1-bit reserved flag;

COLOR is a 32-bit subgraph tag;

OPA is the 32-bit operand A;

OPB is the 32-bit operand B;

ADR is 32-bit target address of the instruction.

3 The Memory Subsystem of Processing Elements

The conventional concept of using operating memory in the control flow model – utilizing Random Access Memory (RAM) – has the advantage of simple design, high integration density of memory elements and low power consumption; however, it appears to be less useful for use in data flow computing. One of the basic operations of a data flow computer is to search for the next executable instruction stored in memory; therefore – in the worst case – all memory must be searched sequentially. This kind of operation requires logic beyond the memory and can be extremely time-consuming.

A solution to this problem is Content Addressable Memory (CAM), typically used in active network devices – for searching in routing tables [20] or in cache memory [21], for example. CAM has been designed with the aim to allow search of the entire memory in a single operation and it allows much faster search operations for immediately executable instructions, in comparison to RAM. However, Unlike RAM (having simple storage cells), each CAM bit is accompanied by a comparator to detect a match between the stored bit and searched bit and an additional circuit to evaluate if all bits of the data word match. This additional circuitry increases the physical size of the CAM chip, or – in other words – decreases the density of the memory elements, increases manufacturing costs and power consumption of the memory, as it is discussed in [22] [23]. With Ternary Content Addressable Memory (TCAM) it is possible to use not only 1 and 0 values, but also X as the “Don’t care” value for one or more bits in the data word to make searches more flexible.

In our research, we are proposing a memory (Figure 7) philosophically based on TCAM. This memory incorporates not only memory cells, but also combinational logic to perform search operations, with which the immediately executable operation can be found, making use of parallel searches of the whole memory. We have altered the CAM concept and incorporated a priority generator into the memory to select the matching data word with the lowermost memory address. At any given time, there is only a single data word – representing the activation frame containing the immediately executable instruction – in the output of the memory.

One can write to the memory in an addressed fashion: a specific part of the data word will be stored at the specified address. This is used to store operands during program execution.

Another important feature of the proposed computer architecture is the ability to initialize or de-initialize subgraphs of the data flow graph with/without the preservation of operands. This is required to evaluate if the activation frame is part of the subgraph – that is why this operation must be performed on each activation frame stored in memory. After the evaluation, the instructions, which are part of the target subgraph, are initialized or de-initialized, according to the instruction type. All of those operations are performed on each activation frame stored in the activation frame store in parallel in a single machine cycle, by means of the built-in combinational logic circuit.

The memory operations of the proposed memory can be divided into two main groups, according to the operating mode of the memory and the computer: *Load mode (L)* and *Compute mode (C)*.

Operations performed in the traditional manner – including reading and writing of a specific activation frame using an address – are the first group of operations (I) performed in *Load mode*, when memory behaves as conventional RAM.

Another group of operations are data-flow-specific operations (II): these are performed on particular activation frames stored in memory (IIa) using an address or a specific attribute of the activation frame; or they are performed on all activation frames stored in memory in parallel (IIIa). The two aforementioned operation subgroups are available only in *Compute mode* and they are performed in connection with particular instructions of the data flow graph during data flow graph execution.

I. Load mode

1. Read;
2. Write.

II. Compute mode (data flow specific operations)

a. Operations on a single activation frame:

1. Writing operand A;
2. Writing operand B;
3. Reading the activation frame.

b. Operations on all activation frames:

1. Data flow graph initialization;
2. Data flow subgraph initialization with operands;
3. Data flow subgraph initialization without operands;
4. Data flow graph de-initialization;
5. Data flow subgraph de-initialization.

Instead of having separate Load and Compute inputs to control the two groups of operations, we have defined the $L/\overline{C}$ control signal. If the $L/\overline{C}$ input is set, the memory is in *Load mode*, otherwise it is in *Compute mode*. In *Load mode*, the $R/\overline{W}$ signal determines the particular operation of reading and writing, respectively (Figure 7).

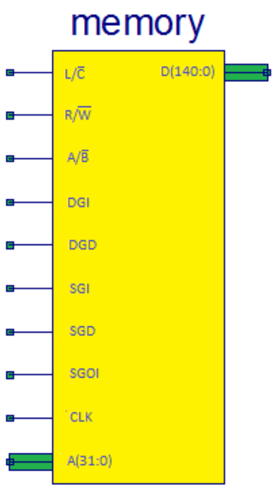

Figure 7
The schematic symbol of the data flow computer memory

3.1 Read Operation

The cycle of reading the activation frame memory is performed by setting the $L/\overline{C}$ and $R/\overline{W}$ signals to 1. The activation frame is addressed in the traditional manner, as used in RWM memory, with the A_α address present at the address input (A), while the activation frame can be read at the memory data output (DO).

$$L/\overline{C}\,.\,R/\overline{W}\,)\ DO := M[A_\alpha] \tag{2}$$

3.2 Write Operation

The cycle of writing operates on a memory address provided by the PE. The PE applies the address to the address input (A), the activation frame vector to the data input (DI) and sets the $L/\overline{C}$ signal to 1 and the $R/\overline{W}$ signal to 0. Then, the PE transfers the data to the activation frame during the positive transition of the next Ti pulse, which can also change the address and checks if there is another memory request. In this mode, the data flow graph is mapped onto the operating memory of the processing element sequentially for subsequent execution. No data flow program activation frame is executed until $L/\overline{C}$ is set, even if there are executable instructions in the memory.

$$L/\overline{C}\,.\,\overline{R/\overline{W}}\,)\,M[A_{\alpha}] := DI \tag{3}$$

3.3 Writing Operand A

The process of writing operand A to the memory can be performed in *Compute Mode*, which is indicated by setting both the $L/\overline{C}$ and the $R/\overline{W}$ signals to 0. The activation frame is addressed at the addressing input of the memory (A) and the set $A/\overline{B}$ signal indicates that operand A will be stored in the activation frame. Operand A must be present in the appropriate part (bits 95:64) of the memory data input (DI) during writing.

$$\overline{L/\overline{C}}\,.\,\overline{R/\overline{W}}.A/\overline{B}\,)\,M[A_{\alpha}] := DI(95:64) \tag{4}$$

3.4 Writing Operand B

A change of the $A/\overline{B}$ signal to 0 indicates the process of writing operand B to the memory, when the $L/\overline{C}$ and $R/\overline{W}$ signals are set to 0. Operand B must be present in the appropriate part (bits 63:32) of the memory data input (DI) during writing.

$$\overline{L/\overline{C}}\,.\,\overline{R/\overline{W}}.\overline{A/\overline{B}}\,)\,M[A_{\alpha}] := DI(63:32) \tag{5}$$

3.5 Evaluation of Instruction Executability

An important operation that the memory performs on each stored activation frame is the evaluation of instruction executability. Each activation frame contains information as to whether the instruction is initialized or not (the INIT bit of the activation frame vector is set when the instruction is initialized). After mapping the data flow program into the operating memory and during program execution, instructions can be in an initialized state (i.e. they are ready for being executed, assuming that their operands are present), or the instructions are de-initialized (i.e. the instructions were executed or they have been prepared for subsequent initialization). Another condition of instruction executability is that the instruction operands must be present. Unary instructions require only a single operand – operand A – while binary instructions need operands A and B. The operand presence requirement is indicated by the PA and BB bits of the activation frame vector. If the operand is present, the RA and RB bits are set, respectively.

An instruction is executable, if the following conditions are fulfilled: INIT is set, RA = PA and RB = PB. The evaluation can be performed by a small logic circuit, as stated in Figure 8.

$$Y = \mathrm{INIT} \wedge ((\mathrm{RA} \oplus \mathrm{PA}) \wedge (\mathrm{RB} \oplus \mathrm{PB})) \qquad (6)$$

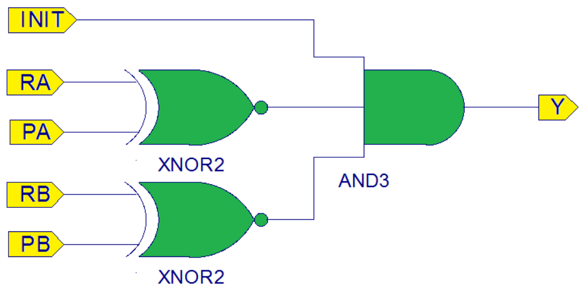

Figure 8
A logic circuit for evaluating instruction executability

The Y signal is set if the instruction is executable. If more than one instruction is executable at the same time, the memory evaluates the situation and selects (utilizing the priority generator) a single executable activation frame with the lowest memory address.

3.6 Reading the Activation Frame

Reading the activation frame in *Compute mode* is controlled by the operating memory itself and there is no need to activate the address input of the memory (A). The $\mathrm{L}/\overline{\mathrm{C}}$ control signal is set to 0 and the $\mathrm{R}/\overline{\mathrm{W}}$ signal is set to 1 during activation frame reading. At any time during program execution, the data output of the memory activation frame contains the activation frame vector of an instruction that is executable immediately. After the execution of the instruction, this frame is de-initialized and another frame is selected for execution. Executability evaluation is performed on each activation frame in parallel via the logic circuits described above. If more than one activation frame is executable concurrently, the priority generator is used select the appropriate one.

Selecting the executable activation frame is very important for proper and effective program execution in data flow computers. It requires a lot of time when using traditional operating memories.

$$\overline{\mathrm{L}/\overline{\mathrm{C}}}\,.\,\mathrm{R}/\overline{\mathrm{W}}\,)\ \mathrm{DO} := \mathrm{M}[\mathrm{A}_{\alpha}] \qquad (7)$$

3.7 Data Flow Graph Initialization

Every time the data flow graph is executed, the data flow graph may be initialized. The DGI input of the memory is set and the initialization operation is performed in parallel on each activation frame vector stored in memory, by setting its INIT bit. All operands stored in memory are removed. This operation may be used to activate another iteration of the data flow graph execution, without the need for data flow graph mapping.

$$\overline{L/\overline{C}}\,.\,\mathrm{DGI}\,)\ M[A_\alpha](0) := 1 \tag{8}$$

3.8 Data Flow Subgraph Initialization with Operands

Every time the data flow graph is executed, the data flow subgraph may be initialized. The SGOI input of the memory is set and the subgraph colour (DFG_{color}) is indicated at the appropriate memory data input.

$$\overline{L/\overline{C}}\,.\,\mathrm{SGOI}\,)\ \mathrm{DI} := \mathrm{DFG}_{\mathrm{color}} \tag{9}$$

A logic circuit acting as a comparator connected to each activation frame evaluates if the memory operation is valid for the given activation frame. If this condition is fulfilled (Y_{SGOI} = 1), the activation frame is initialized and the operands present are preserved. By this initialization, the subgraph is prepared for another execution iteration.

$$L_{\mathrm{SGOI}} \mid Y_{\mathrm{SGOI}} = 1 \tag{10}$$

3.9 Data Flow Subgraph Initialization without Operands

The initialization of the data flow subgraph can be performed by setting the SGI memory input to 1. The subgraph colour is indicated at the appropriate data input of the memory.

$$\overline{L/\overline{C}}\,.\,\mathrm{SGI}\,)\ \mathrm{DI} := \mathrm{DFG}_{\mathrm{color}} \tag{11}$$

The comparator, connected to each activation frame, checks if the memory operation is valid for the given activation frame. If this condition is fulfilled (Y_{SGI} = 1), the activation frame is initialised. Operands present are removed. Now, the data flow subgraph is ready for another iteration of execution.

$$L_{\mathrm{SGI}} \mid Y_{\mathrm{SGI}} = 1 \tag{12}$$

3.10 Data Flow Graph De-Initialization

If the DGD memory input is set to 1, de-initialization is performed on each activation frame vector stored in memory in parallel, by resetting their INIT bits. All operands stored in memory are preserved. This operation may be used to deactivate the data flow graph execution immediately, using the KG (Kill Graph) instruction that is part of instruction set of the architecture.

$$\overline{\mathrm{L}/\overline{\mathrm{C}}}\,.\,\mathrm{DGD})\ \mathrm{M}[\mathrm{A}_{\alpha}](143) := 0 \tag{13}$$

3.11 Data Flow Subgraph De-Initialization

Setting the SGD memory input to 1 indicates the de-initialization process of the data flow subgraphs having the specific colour presented at the appropriate data input of the memory. This operation may be used to deactivate the data flow subgraph execution immediately, using the KSG (Kill SubGraph) instruction that is part of instruction set of the architecture.

$$\overline{\mathrm{L}/\overline{\mathrm{C}}}\,.\,\mathrm{SGD})\ \mathrm{M}[\mathrm{A}_{\beta}](143) := 0 \tag{14}$$

The comparator connected to the activation frame checks if the operation is valid for the given activation frame. If this condition is fulfilled, the activation frame is de-initialized by resetting the INIT bit, while the operands present are preserved. The data flow subgraph is now ready for another initialization.

$$\mathrm{L}_{\mathrm{SGD}} \mid \mathrm{Y}_{\mathrm{SGD}} = 1 \tag{15}$$

Conclusions

The presented data flow computer architecture with tiled organization of processing elements is the result of the research performed at the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia. Within the present research, we propose a memory with activation frame store functionality based on Ternary Content Addressable Memory (TCAM), which behaves as conventional RAM in *Load Mode* and CAM in *Compute Mode* and is capable of executing operations specific for the proposed data flow computer architecture. This memory represents an innovative approach to the construction of data flow computers; in future research it may serve as a basis to solve problems of data flow computers that limit their field use.

Acknowledgements

This work was supported by the Slovak Research and Development Agency under contract No. APVV-0008-10. The project is being implemented at the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia.

References

[1] Gy. Györök, M. Makó, J. Lakner: „Combinatorics at Electronic Circuit Realization in FPAA“, In: Acta Polytechnica Hungarica. Vol. 6, No. 1 (2009), pp. 151-160, ISSN 1785-8860

[2] B. Grot, J. Hestness, S. W. Keckler, O. Mutlu: „Express Cube Technologies for On-Chip Interconnects“, Proceedings of the 15th International Symposium on High-Performance Computer Architecture, February 14-18, 2009, Raleigh, North Carolina

[3] J. Kopják, J. Kovács: „Timed Cooperative Multitask for Tiny Real-Timeembedded Systems“, IEEE 10th Jubilee International Symposium on Applied Machine Intelligence and Informatics (SAMI 2012) Herl'any, Slovakia, 2012, pp. 377-382

[4] J., B. Dennis, R. P. Misunas: „A Preliminary Architecture for a Basic Data Flow Processor“, Proceedings of the 2nd Annual Symposium on Computer architectures., 1974

[5] A. Veen: “Dataflow Machine Architecture”, ACM Computing Surveys, December 1986, pp. 365-396

[6] P. Fanfara, E. Danková, M. Dufala: “Usage of Asymmetric Encryption Algorithms to Enhance the Security of Sensitive Data in Secure Communication”, 2012. - 1 CD-ROM. In: SAMI 2012: 10th IEEE Jubilee International Symposium on Applied Machine Intelligence and Informatics : proceedings : Herľany, Slovakia, January 26-28, 2012. - Budapest : IEEE, 2012 S. 213-217. - ISBN 978-1-4577-0195-5

[7] V. V. Vlassov, A. V. Kraynikov, B. A. Kurdikov: "A Data Flow Computer System". In: Izvestiya LETI (Proceedings of Leningrad Electrotechn.Inst.), St. Petersburg, Vol. 436, 1991, pp. 3-7

[8] L. Vokorokos: “Data Flow Computer Principles” (in Slovak), Copycenter, spol. s.r.o., Košice, Slovakia, 2002. ISBN 80-7099-824-5

[9] A. Arvind, D. E. Culler: "The Tagged Token Dataflow Architecture (preliminary version)".Tech. Rep. Laboratory for Computer Science, MIT, Cambridge, MA, 1983

[10] T. Shimada, K. Hiraki, K. Nishida, S. Sekigughi: "Evaluation of a Prototype Dataflow Processor of the SIGMA-1 for Scientific Computations", In: Proc. 13th Annual Symposium On Computer Architecture, 1986, pp. 226-234

[11] N. Ádám: „Single Input Operators of the DF KPI System“, In: Acta Polytechnica Hungarica. Vol. 7, No. 1 (2010), pp. 73-86, ISSN 1785-8860

[12] E. Waingold et al.: “Baring it All to Software: RAW Machines”, IEEE Computer, Sep. 1997, Technical Report, MIT Cambridge, Massachusetts, USA

[13] Tilera Corporation: TILEPro64 Processor Product Brief, 2011

[14] Sriram Vangal, et al. "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS," l. IEEE Journal of Solid-State Circuits, Vol. 43, No. 1, Jan 2008

[15] V. P.Sirini, J. Thendean, S. Z.Ueng, J. M. Rabaey: “A Parallel DSP with Memory and I/O Processors. In Proceedings SPIE Conference 3452, pp. 2-13, 1998

[16] S. Swanson, K. Michelson, A. Schwerin, M. Oskin, “WaveScalar”, Proc. of the 36th International Symposium on Microarchitecture (MICRO-36 2003) 2003, pp. 291-302, ISBN 0-7695-2043-X

[17] Garnacki, J., J. a M. D. V,: MONARCH a High Performance Embedded Processor Architecture with Two Native Computing Modes, in High Performance Embedded Computing, 2002

[18] K. Mai, T. Paaske, J. Nuwan, R. Ho, W. Dally, M. Horowitz: “Smart Memories: A Modular Reconfigurable Architecture”, ISCA 00, Vancouver, British Columbia, Canada, ACM 1-58113-287-5/00/06-161

[19] L. Vokorkos, B. Madoš, A. Baláž, N. Ádám: „Architecture of Multi-Core Computer with Data Driven Computation Model“, In: Acta Electrotechnica et Informatica. Roč. 2010, č. 4 (2010), s. 20-23. - ISSN 1335-8243

[20] A. J. McAuley and P. Francis: "Fast Routing Table Lookup Using CAMs", IEEE INFOCOM 1993, pp. 1382-1391, March 1993

[21] A. Efthymiou and J. D. Garside: "An Adaptive Serial-Parallel CAM Architecture for Low-Power Cache Blocks", In Proc. of the ISLPED, pp. 136-141, 2002

[22] H. Miyatake, M. Tanaka, and Y. Mori: “A Design for High-Speed Low-Power Cmos Fully Parallel Content-Addressable Memory Macros,” IEEE J. Solid-State Circuits, Vol. 36, pp. 956-968, Jun. 2001

[23] I. Y.-L. Hsiao, D.-H. Wang, and C.-W. Je:, “Power Modelling and low-Power Design of Content Addressable Memories,” in Proc. IEEE Int. Symp. Circuits and Systems (IS- CAS), Vol. 4, May 2001, pp. 926-929