ISSN 2078-7707

МЭС (MES)

ПРОБЛЕМЫ РАЗРАБОТКИ ПЕРСПЕКТИВНЫХ МИКРО- и НАНОЭЛЕКТРОННЫХ СИСТЕМ (МЭС)

PROBLEMS OF ADVANCED MICRO- AND NANOELECTRONIC SYSTEMS DEVELOPMENT (MES)

СБОРНИК ИЗБРАННЫХ ТРУДОВ НА АНГЛИЙСКОМ ЯЗЫКЕ
VII Всероссийской научно-технической конференции МЭС-2016
Часть II

SELECTED ARTICLES
of the VII All-Russia Science&Technology Conference MES-2016
Part II

Федеральное государственное бюджетное учреждение науки
Институт проблем проектирования в микроэлектронике
Российской академии наук
Institute for Design Problems in Microelectronics
of the Russian Academy of Sciences

Москва - 2017

CONTENTS

Verification and Testing

High-performance Computing Microelectronic Systems

Recurrent Data-flow Architecture: Features and Realization Problems

Yu.A. Stepchenkov, Yu.G. Diachenko, D.V. Khilko, V.S. Petrukhin

The Institute of Informatics Problems, Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences,

ystepchenkov@ipiran.ru, diaura@mail.ru, dhilko@yandex.ru, cokrat2@rambler.ru

***Abstract* — Results of development of the multi-core recurrent data-flow architecture (MRDA) focused on effective implementation of digital signal processing (DSP) algorithms are presented. Principal differences between MRDA and existing computer architectures are shown. Such differences make it possible to process the instructions in almost half the normal time using singular self-sufficient recurrently represented data-flow. Additional mechanisms that enhance the performance of computations for a number of DSP algorithms have been listed. Some of the proposed mechanisms can also be used in DSP systems of traditional architecture.**

***Keywords* — data-flow architecture, recurrence, digital signal processing, match memory, superscalarity.**

I. INTRODUCTION

The relevance of computing architectures based on data-flow paradigm development is undoubted, since it can potentially provide much higher performance comparing to conventional von Neumann architecture. However, effective implementation of data-flow architectures (DFA) encounters a number of significant problems, such as the implementation of recursion, cycles, iterations as well as working with constants and others.

In a number of foreign experimental projects conducted from the early 1980s to the mid 2000s, where dynamic data-flow (DF-D) architectures have being developed, effective ways to solve these problems had not been found [1-3]. Disadvantages inherent in DFA:

- Hardware complexity and comparison time of tagged markers in the Match Memory (MM) are quite high. Therefore maximum possible architecture performance could be achieved by implementing Match Memory in the form of Associative memory. However, the large amount of memory required to store pending tokens makes this approach problematic.
- The degree of parallelism in programs being executed is out of control. A number of applications can generate more of concurrently executable fragments than the capabilities that hardware provides. This results in decrease of real performance.
- Sequential sections of code are executed inefficiently. The loss of time in these areas due to the comparison of tagged markers, their exchange, formation and access to various structures of the memory device cannot always be compensated by simultaneous processing of data streams in parallel sections of the code.
- Theoretically an amount of concurrently running loop iterations and procedures has no limitation. This leads to the growth of hardware complexity due to an increase in the size of the tags and the time-loss for their communication over the network.
- A wide variety of memory types, such as: Match Memory, Overflow memory (block of deferred markers), Program Memory and Constant Memory, Memory of tagged markers and others, - makes memory management very complicated.
- It is impossible to use register structures with the same efficiency as in a conventional multi-processor system.

An effective solution of these and other problems not mentioned is possible only on the basis of fast and capacious Associative memory (AM), which will be the most hardware and energy consuming resource. Therefore AM is only useful for systems of mass parallelism.

In Russia, serious results in the development of universal high-performance systems based on data-flow principles were achieved by a team led by Academician Burtsev V.S. (see, for example, [4]). After the death of Academician Burtsev V.S. and the transition of most of his team from the IPI RAS to IPPM RAS, work on improving this data-flow architecture has been continued there.

Attempts to use the data-flow paradigm in the field of digital signal processing also have a long history [5-8]. The authors of these studies note that the principles of the DFA and the requirements of the DSP algorithms are well combined with each other in applications that are characterized by a high degree of internal parallelism. The main constraining factors for the wide practical use of such integration are the high price of multi-processor implementations as well as the performance losses in the off-chip implementation of the communication interprocessor network. For most DSP applications, the dynamic allocation of data is unnecessary, since the predictability of program execution time ensures the viability of static methods of their distribution.

At the Institute of Informatics Problems of the Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences the concept of a fundamentally new Multicore Recurrent Dataflow Architecture (MRDA) has been developed. The architecture is initially

developed as a specialized one, intended for realization of parallel computing processes of signal processing in real time. It is based on a recurrent-dynamic computational paradigm, which can be considered as the improvement of a data-flow paradigm, but built on another principle, namely, on the principle of self-contained, recurrently compressed data (RD).

II. FEATURES OF COMPUTATIONAL PROCESS ORGANIZATION IN MRDA

The proposed architecture is unconventional and radically different in its main points not only from the classical von Neumann architecture, but also from other non-traditional parallel architectures, in particular, from data-flow-based architectures.

Existing traditional and non-traditional computer architectures have two basic flows: flow of instructions and flow of data. In MRDA both flows are combined into one single ″self-extracting″ flow of self-sustained data that encapsulates data itself as well as control and service information needed for processing them. A clear graphical representation of the comparative qualities of the architectures in question is given by their comparison, shown at Fig. 1 and Fig. 2, by following criteria:

- memory organization and computational process initiation (Fig. 1);

- number of steps required to execute the instruction (Fig. 2).

same. Generally functional fields are used to store information about instruction address to be extracted from program memory. This instruction defines actions that have to be performed on the incoming components of the pair (arrow "2" at Fig. 2,b). The instruction set is fully stored in static form as well. Therefore such model was classified as Data-Flow/Static (DF/S).

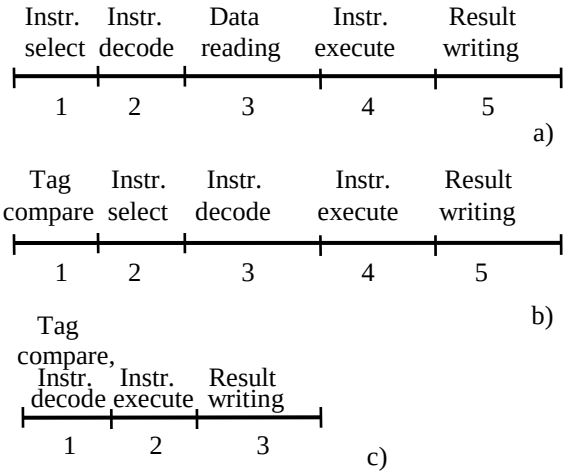

Fig. 2. Instruction execution in compared architectures a) – c) – identically to Fig. 1

MRDA belongs to the class of DFA but unlike them it

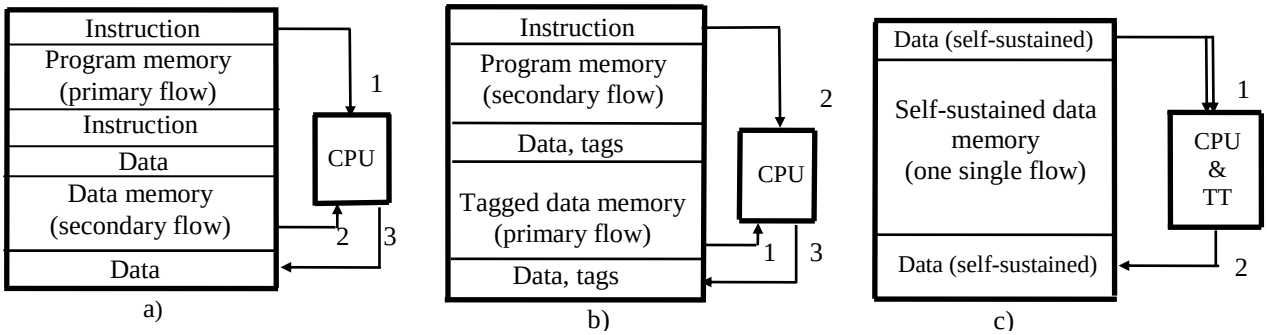

Fig. 1. The principal differences of compared architectures a) traditional (CF/S), b) data-flow (DF/S), c) MRDA (DF/SD)

The instruction flow being extracted from program memory of CPU (instruction flow has a primary role) initiates computing (arrow "1" in Fig. 1, a). The program-initiator of the process involves data for processing: a secondary flow (arrow "2" in Fig. 1, a). The initiating program is fully stored at instruction memory in static. Thus there is a problem to identify the moment of data preparedness for computing. We have classified such computational model as Control Flow/Static (CF/S).

Second fundamental class consists of DFA. These architectures use same memory partitioning as CF/S models, but in contradistinction to CF/S the data flow initiates computing (data flow has a primary role – arrow "1" in Fig. 1,b). Furthermore data memory stores data (operands) with additional functional fields (tagged fields) in its cells. Thus total amount of memory used is approximately the

operates with self-sustained data. This type of data contains information about: the component of the pair to be matched; instruction to be performed on the pair; address where the operation result has to be stored. There are several possible destinations for the operation result that can be: stored at inner registers of current CPU for later use; transferred to other CPUs through the communication network; stored into Match Memory for later use; transferred to the external environment as an output result.

The main property of MRDA has been called ″recurrence″, which is the dynamic evolution of computational process. We define dynamic evolution as a functional transform of originally compressed tagged fields taking into account possible variety of transforming conditions. Following this definition in terms of self-sustained data each following step is calculated as a function of pre-

vious step during evolution of computational process. Thus, the original instruction flow is folding recurrently, thereby allowing to drastically reduce the overhead associated with storing of computational process planned trace.

In contradistinction to DF/S tagged data are also self-sustained. Such tagged data are called recurrently unfolding and contain compressed information to keep computing based on recurrence principle. Within the MRDA ″Tag Transformer″ (TT) component is included as a part of CPUs. It provides recurrent self-extraction functionality of computational process. TT is initialized by the operands that have to be processed at CPU, it operates in parallel with CPU and determines next computational step action (transforming functional fields). This device is a relatively simple (in terms of hardware costs) combination circuit that contains the means for configuration (in necessary cases) on the subject area.

Match memory of MRDA does not contain program being executed in traditional meaning. There are only original values of functional fields that are recurrently unfolded by TT dynamically. Thus, MRDA was classified as Data-Flow/Static Dynamic (DF/SD). In order to execute the algorithm, you must specify the initial values of the functional tags. Such representation of program in the MRDA was called a capsule.

In order to obtain the instruction result in CF/S and DS/S following sequence of steps 1-5 is performed (see Fig. 2a and 2b respectively). As for DS/SD architecture, the amount of steps is reduced to its logical minimum of 3 as shown at Fig. 2c. Moreover, a total volume of memory resources required and an amount of data transfers between functional blocks of a computational system are also reduced. Thus if the reduction in the number of steps in MRDA to 4 is obvious (there is no need to select the instruction), then the possibility of combining the tags comparison and the instruction deciphering procedures at one step will be shown below when considering the structure of a recurrent operational unit (ROU).

III. HYBRID TWO-LEVEL OPTION OF MRDA IMPLEMENTATION

The historical experience of the computer architectures development testifies [9] that the new architecture development may fail for the following reasons:

• if it is based on extremely complex hardware mechanisms to support its intended computational model;

• if it is incompatible with existing computing environments;

• if it ignores the programmability problems.

The results obtained on various implementations of ROU convincingly testify that it does not require complicated hardware mechanisms to support the inherent Capsule programming model. As for the second statement, the compatibility with existing computing environments is not implied within the development of ROU. On the contrary, it involves the development of a unique computing environment, that takes into account the specifics of the MRDA at the maximum extent.

The search for a compromise solution, that includes compatibility with existing computing and hardware environments, is possible when certain requirements are met. Primarily it is required to support the data-flow nature of the computational process implemented in the ROU

Implementation of this approach is possible on the basis of a hybrid architecture variant of a recurrent signal processor (HARSP): a processor with a reduced NIOS2 instruction set and FPGA organized as Programmable Logic Device (PLD) within the Altera family of chips, code-named Stratix IV [10] (see Fig. 3).

The structure of HARSP includes a control level with control processor (CP), based on von Neumann architecture, and an operational level (ROU) that consists of: a distributor (controls data distribution), four single-type recurrent computing blocks (CB) and an interface for interprocessor exchange. In turn, each CB consists of: match memory, computing device (based on 16-bit ALU and multiplier with accumulation MAC) and a tag transformer.

As an interface device between CP and ROU (the data stream between them is of an intensive nature), it is proposed to use a special type of dual-port buffer memory (BM) implemented in the FPGA. Simultaneous reading of the capsule from one port and recording the computational results at another port will surpassingly allow balancing the data flow between the control processor and four computing devices.

The control level of HARSP is entrusted with following functions: preliminary preparation of capsules, implementation of consecutive parts of the executable program and recording of the results at one BM port. The operational level of the RSP provides: reading of capsules that are ready for execution, parallel calculations in the CB and recording the results of calculations at another BM port. Also the introduction of a dual-port BM into the RSP structure will make it possible to exclude the time-loss for moving data between two levels of the HARSP architecture.

Buffer memory is designed to store capsule patterns. In data exchange mode between BM and ROU the dual-port implementation of memory provides simultaneous reading and writing from the ROU side.

Attribute memory (data ready bit, DR-bit) controls the availability of data in the capsule patterns. If the data at the specified address in the BM is ready for reading, then the DR-bit is set to '1', otherwise to '0'. In data exchanging process between BM and ROU, it becomes necessary to simultaneously change the attributes at two addresses (dropping DR-bit for the read data, and setting up DR-bit for the received data). The dual-port organization of the BM provides simultaneous recording at port 1 and reading from port 2 or one-time recording at port 1 and port 2.

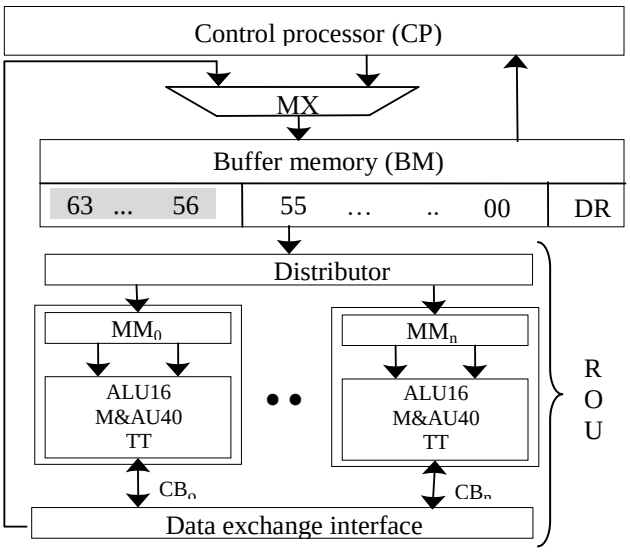

Fig. 3. HARSP structural scheme
MM – match memory; DR – data readiness;
ROU – recurrent operational unit; TT – tag transformer;
M – multiplier; AU – arithmetical unit

There is a restriction of maximum acceptable length for BM activity cycle that equals to four and it is defined by the operating time of the slowest stage of ROU pipeline. Apart from this restriction, there are several requirements that have to be met by BM:

• it should store and operate with operands that have 56-bits (extended to 64-bits) size;

• it should be able to send into ROU up to 4 operands for one cycle;

• it should be able to write up to 4 operands sent from ROU in a single cycle.

In order to operate with required performance BM has been separated into 8 banks (8-bits each) that are working in parallel (B0 – B7).

The capsule can contain, and ROU can accept, four types of operands (Table 1). These operand types are indicated in the mandatory field for all operands – [t].

Table 1

Operand types in ROU

t~field		**Operand type description**
Symbol	**Code**	
E:	0	Empty
A:	1	Assistant
C:	2	Control
D:	3	Data-containing

Only *data-containing* operands can carry input data (so this type is called D-type) to be processed during the computation. This type of operands can be processed and received by all ROU modules.

The *empty* operand (E-type) has no structure, doesn't carry information and is not processed in ROU (it has a debug purpose).

Assistant operands (A-type) contain service information that does not affect the organization and the course of computational process. Minimal set of these operands includes: *capsule terminator*, *data terminator*, *global configurator* (configures ROU as a whole), *initializer* and *template*. Each of them has its own specific format and is intended for setting up one of the ROU units.

Control operands (C-type) control the progress of the computational process, coordinate the work and configure some modules of the structure. Examples of C-type operands are: *the configurator of a separate sectio*n, it also performs the tuning of TT, if necessary; *the brake*, it serves as a software stop function; *the driver*, it reinitializes the computational process; there are also operands of *unconditional* and *conditional* transitions.

Capsular programming style is implemented in HARSP, where for each algorithm a capsule template is developed, the general form of which is presented in Table 2.

Table 2

Capsule structure

№	Operand	Structure	Description
1	***Ani***	101%Ncpx@Ntvad	Input capsule identifier
2	***Adi***	*131%Nc1*	Data set identifier
3	***Acg***	*151%Cxpcres*	Global configurator
4	***Ai***	*160%I0nis%I1nis*	Data set initializer
5	***Aso***	*110%Stn[OcuSmDrse]*	Out data starter
6	***Atm (0-3)***	*161%Tcuhxmrse[OuShmD_]*	Template (for sections 0-3)
7	***Ccs***	*200@Cjlder[0cutShmD0001se]*	Configurator of a separate section (for sections 0-3)
8	***Di***	*32@sV0_[OcutShmDrse]*	Data-containing operand V0
9	***Di***	*32@sV1_[OcutShmDrse]*	Data-containing operand V1
10	***Asi***	*111%St[OcutSmDrse]*	Input data starter
11	***Di2 + Di4***	*32V2_Sh_V3_Sh_V4_Sh*	Data-containing operandsV2,V3 and V4
12	***Di(n-1) + Di(n-4)***	*32Vn-1_Vn-2_Vn-3_Vn-4*	Data-containing operands Vn-1 and Vn
13	***Az***	*141%Ncp@Ntvad*	Capsule terminator

Any capsule template contains a set of assistant and control operands that have their DR-bits set up in the BM, as well as a series of constants whose values are known in advance. Therefore, as soon as the CP initializes the capsule execution, the operands with DR-bits being set up begin to be processed by ROU.

Data-containing operands have following notation and subfields t@sV0_[OcutShmDrse], where V is the data value field (see Table 3):

- subfield of the data dimensionality @s (16/38 bits);
- subfield of the operation of the computing device [Oc];
- subfield of the mode of using the operand [Ou] in the MM;
- subfield of operation type [Ot]: normal, cyclic, transition initiation, constants reading, etc .;
- subfields of match codes [Shm] in the MM;
- replication mask subfield [Dr];
- subfield of transfer of the operand between the sections [Ds];
- subfield of transfer of the operand to the Em-bus [De].

Table 3

Data-containing operand structure

	Control part			**Functional part**								**Recurrent functional part**					**Data part**
00000000	T	~~~	@s	Dr	Sh	Sm	Ou	Oc	Ot	Ds	De	Sm	Ou	Oc	Ds	@s	V
00000000	11	000	0														
8	2	3	1	4	1	3	2	5	3	2	1	3	2	5	2	1	16
63...56	55...54	53...51	50	49...46	45	44...42	41...40	39...35	34...32	31...30	29	28...26	25...24	23...19	18...17	16	15...0

Figure 4 shows the detailed structure of the basic version of ROU. In addition to general-purpose devices (Distributor, Collector, Implicator and others), Fig. 4 shows the structure of one of its four identical sections.

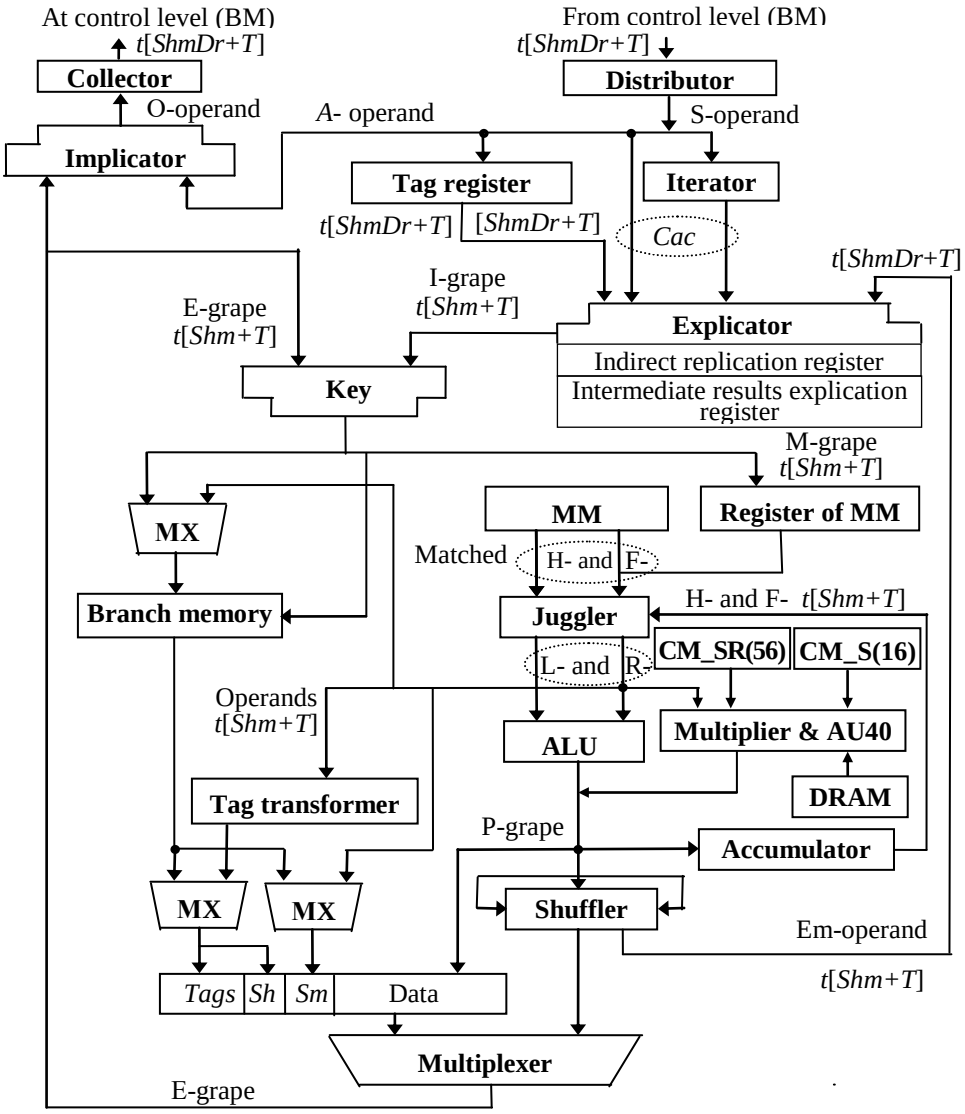

Fig. 4. Basic version of ROU structure

Distributor - is a FIFO-buffer that receives sequences of operands with DR-bits being set up, which are contained in capsules (BM). Distributor is the source of the original S-operands that appear at its output in the order in which they are located in the capsule. The task of the Distributor is not only to supply operands to the input of the "Explicator" module, but also to analyze the type of the operand at its output (in the top cell - the FIFO vertex) and to distribute some types of the assistant and control operands to their destination. FIFO-buffer depth of the Distributor may vary.

Some ROU components process not only single operands, but packages of operands that have been called "grapes". Therefore, before processing, the operands from the capsule must be transformed into "grape" form. This transformation is called *the explication of "grapes"* and is performed by the *Explicator* module. In fact, Explicator is the Distributor's sub component that is preparing "grapes" for transferring at corresponding data routes of sections.

Explication itself is a complex operation, because it involves *replication* (multiple reproduction or "vectorization") of the *operand* and strictly speaking *explication* (forming a "grape"). For replication in the functional subfield [*Dr*] of all source operands, an explicit *replication mask* is provided. Each bit of the mask corresponds to a specific section; therefore being set up as "1" it directly specifies the section into which the operand is to be sent. This type of mask has been called *direct replication mask* accordingly. Operands with the same value of direct replication mask are always sent to the same sections. In case when [Dr] subfield has zero value within ROU a set of special mechanisms of indirect replication has been defined. Thus, this type of mask has been called *indirect replication mask*.

A prepared "grape" (called I-grape) is sent to the MM modules. The MM module is available in each section of ROU and is capable of performing the following functions:

- selection and commutation of "grapes" that are being memorized;
- operands storing;
- selection of operands whose subfields [Shm] have coincided. While the pair component is being read from the MM, the operation code subfield [Oc] of the operand that has been sent into register of MM, can be decrypted. This ensures the simultaneity of the actions at step 1 for the MRDA (see Fig. 2).

Within ROU the computing block has been called "Computer" and it is able to perform following operations:

- arithmetic-logical functions over the operands data, using the information coming from the L and R buses: L – tV [] and R – tV [Oc];
- recurrent unfolding of functional fields, using information supplied to the R-bus: [Ocut_ShmD_] at Tag Transformer.

The resultant E-operands are either stored in one of "Computer's" registers (A, B or C) or returned to the MM via E- or Em-buses for further participation in the calculations (intermediate results) or for output to the control level using Implicator and Collector components. These results may be classified as final output or as temporary data to be reused at ROU.

As in standard digital signal processors, "Computer" contains a MAC hardware block that allows to multiply two 16-bit operands and to accumulate the multiplication result with value containing in one of the two internal 40-bit registers in one clock cycle. The dimensionality of 16 bit for input data is enough for a number of applications in the field of speech processing. For applications where 16 bits are not sufficient, the ROS architecture provides 38 bit input and output operands.

In addition to multiplication with accumulation, MAC unit is able to execute the commands of the arithmetical, logical shifting and rounding results. CB also contains a 16-bit arithmetic logic unit (ALU) that performs basic arithmetical and all logical commands.

Each of the listed nodes and registers of CB represents an independent hardware resource, idle time price of which is quite high. Therefore, the CB within the ROU is able to execute in parallel up to two instructions per clock cycle, i.e. it has superscalar architecture.

A computing device is called superscalar (the term was first used in 1987), if it simultaneously executes more than one scalar command. Thus, the implementation of CB with superscalar architecture made it possible to achieve parallelism at the instruction level.

The instruction set of ROU has been expanded with the *special multi-cycle operation "FFT"* (basic operation for Radix-2 FFT-algorithm), that is simultaneously using the maximum number of functional nodes of "Computer" module. This approach made possible to reduce to a minimum the number of logical steps required for implementation of "butterfly" operation due to the optimal allocation of resources of the CB (see [12]). When implementing such a command, all existing hardware nodes of CB are used in parallel, namely a 16-bit ALU, a 16-bit multiplier and a 40-bit adder.

In order to minimize overhead in the implementation of cyclic procedures, the following tools are introduced into the RSP architecture for each section:

1) one 8-bit cycle counter in the structure of the CB;

2) one functional block – Branch memory;

3) four formats of operands: the tag loader, the loop counter loader, the internal loop controller and the external loop controller;

4) one 2-bit field of operation type.

The above list of mechanisms implemented in the RSP allows increasing the performance of computations for a number of DSP algorithms.

IV. CONCLUSION

The analysis of the correspondence degree of the data-flow computational paradigm principles to the requirements of the DSP algorithms has shown the expediency of developing RSP as a new generation of signal processor that is based on a new recurrent-dynamic computational paradigm.

One of the most promising approaches to the developing

of RSP is its implementation in the form of a two-layer architecture, composed of leading Von Neumann processor at the control level and the number of recurrent processors on the operational level (ROU) directly related to each other. The control processor is assigned with the following functions:

- connection of multi-core data-flow system with the environment;
- interface between standard software and capsules that are stored in BM;
- computing device for successive parts of the algorithm;
- control device for exception handling in a multiprocessor network;
- device-linker of self-sustained capsules for their execution at the operational level of the architecture.

Such solution made it possible to confirm the potential application efficiency of the data-flow paradigm in the DSP area [11, 12] and to ensure compatibility with existing computing environments.

The implementation of mechanisms described along this paper in ROU will allow increasing the performance of computations for a number of DSP algorithms. Some of these innovations concerning the instruction set and the operating modes organization of the "Computer" modules are not directly related to the features of MRDA and can also be implemented in other DSPs by maximizing the use of existing hardware.

Another feature of the MRDA being developed is the orientation to self-timed circuitry: self-timing at the logical level (by the readiness of the source data) is well combined with self-timing at the hardware level (by the readiness of the results). Therefore, despite the fact that synchronous circuitry is currently used, the interaction between the functional blocks and the stages of the computational pipeline is performed asynchronously.

In the future, it is expected to move from FPGA-basis with the control processor NIOSII and synchronous implementation of the ROU to a custom CMOS-basis with KOMDIV as a control processor and self-timed implementation of ROU as a coprocessor. Moreover there are self-timed coprocessors for KOMDIV that have already been developed: 64-bit device of division and of square-root extraction [13] and 64/32-bit device of multiplication-addition with a single entry rounding [14].

ACKNOWLEDGEMENTS

In conclusion, we want to express our gratitude to N. V. Morozov, D. Yu. Stepchenkov and Yu. I. Shikunov for an invaluable contribution to the development of the software and hardware model of MRDA.

SUPPORT

The study was partially supported by the 2016's subprogram no. 4 of ONIT RAS department. (Project 0063-2015-0016 III.3).

REFERENCES

[1] J. Gurd, C. Kirkham, and I. Watson. The Manchester prototype dataflow computer. Commun. ACM, 28(1), Jan. 1985. P. 34-52.

[2] Arvind, and R.S. Nikhil. Executing a program on the MIT tagged token dataflow architecture. IEEE Trans. Comput., 39(3), Mar. 1990. P. 300-318.

[3] K. Hiraki, S. Sekiguchi, and T. Shimada. Status report of SIGMA-1: A data-flow supercomputer. In J.-L.Gaudiot and L.Bic, editors. Advanced Topics in Data-Flow Computing, chapter 7, Prentice Hall, Englewood Cliffs, New Jersey, 1991. P. 207-224.

[4] Burcev V.S. Overlapping of computing processes and development of architecture of the supercomputer. Moscow, Torus Press, 2006. 203 p. (in Russian).

[5] Sundararajan Sriram. Minimizing Communication and Synchronization Overhead in Multiprocessors for Digital Signal Processing. Ph.D. Dissertation, Dept. of EECS, Technical Report UCB/RL 95/90, University of California, Berkeley, CA 94720, October, 1995.

[6] M. Chase. A Pipelined Data Flow Architecture for Digital Signal Processing: The NEC μPD7281. IEEE Workshop on Signal Processing, November 1984.

[7] Iiro Hartimo. DFSP: A Data Flow Signal Processor. IEEE Transactions on Computers. January 1986, V. C-35, N 1, P. 23-33.

[8] K. Kronlof, J. Skytta, 0. Simula, I. Hartimo. Simulation of a digital signal processing architecture based on the data flow principle. Proc. ISCAS'82, Rome, Italy, May 10-12, 1982. P. 1053-1056.

[9] Arvind. The Evolution of Dataflow Architecture from Static Dataflow to P-RISC. Proc. of Workshop on Massive Parallelism: Hardware, Programming and Application, Amalfi, Italy, October 1989. Academic Press, 1990.

[10] Volchek V.N., Stepchenkov Yu.A., Petrukhin V.S., Prokofyev A.A., Zelenov R.A. Digital Signal Processor With Non-Conventional Recurrent Data-Flow Architecture. Problems of Perspective Micro- and Nanoelectronic Systems Development - 2010. Proceedings, edited by A. Stempkovsky, Moscow, IPPM RAS, 2010. P. 412-417 (in Russian).

[11] Yu. Shikunov, D. Khilko, Yu. Stepchenkov. Hardware and Software Modelling and Testing of Non-Conventional Data-Flow Architecture. Proceedings of the 2016 IEEE North West Russia Section Young Researchers in Electrical and Electronic Engineering Conference (ElConRusNW). 2016. P. 360-364.

[12] Yu. Stepchenkov, V. Volchek, V. Petrukhin, A. Prokofyev. Hardwere maintenance for digital processing of speech signals in the recurrent dataflow processor. Systems and means of informatics – TORUS PRESS, Moscow, 2010. Vol. 20. No. 1. P. 31-47 (in Russian).

[13] Stepchenkov Y., Diachenko Y., Zakharov V., Rogdestvenski Y., Morozov N., Stepchenkov D. Quasi-Delay-Insensitive Computing Device: Methodological Aspects and Practical Implementation. PATMOS'2009: Proceedings of the International Workshop on power and timing modeling, optimization and simulation. – Delft, The Netherlands, Springer 2010. P. 276–285.

[14] Stepchenkov Yu., Zakharov V., Rogdestvenski Yu., Diachenko Yu., Morozov N., Stepchenkov D. Speed-Independent Floating Point Coprocessor. Proceedings of IEEE East-West Design & Test Symposium (EWDTS'2015), Batumi, Georgia. 2015. P. 111-114.

Recurrent Data-flow Architecture: Technical Aspects of Implementation and Modeling Results

D.V. Khilko, Yu. A. Stepchenkov, D. I. Shikunov, Yu. I. Shikunov

The Institute of Informatics Problems, Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, IPI FRS CSC RAS,

dhilko@yandex.ru, ystepchenkov@ipiran.ru, shikunovdima@gmail.com, yishikunov@yandex.ru

***Abstract* — The paper covers methods and features of implementing a prototype architecture based on a new recurrent data-flow paradigm of computing designed to solve problems of digital signal processing. Demonstration of key principles and technical solutions implemented in the new architecture is presented, with the example of the Fast Fourier Transform task, as well as estimation of the speed of this task with respect to its solutions on processors of traditional single-core and specialized data-flow multi-core architectures. Comparative estimates of the effectiveness of the implementation of algorithms for isolated words recognition in the environment of the recurrent architecture with respect to von Neumann single-core one are shown.**

***Keywords* — data-flow architecture, recurrence, digital signal processing, fast Fourier transform.**

I. INTRODUCTION

Nowadays, the tasks of digital signal processing (DSPs) are becoming more and more relevant with specialized digital signal processors (DSP) being used to solve them. One of the possible options for implementing DSP is the use of the data-flow architecture. Due to the structure of DSPs algorithms and the data-flow paradigm having high synergy they are "well suited" to each other. Besides the obvious advantages of using data-flow architectures for DSPs tasks, there are a number of issues that prevent their successful commercial implementation [1]–[3]. The main problems are the complexity of the implementation of pipelined data processing, recursive calculations, cyclic procedures, interaction with constants and repeated use of program code.

During the search for a solution for the said issues, the advanced computer systems architectures department of IPI FRS CSC RAS have developed a concept of the new multicore recurrent data-flow architecture (MRDA). Main aspects of MRDA architecture are described in [4-5]. The most important features of MRDA are *self-sustained data* and *recurrence*.

Self-sustained are data that incorporate both data and instructions (Tags), required for their processing. In other words, two traditional flows – data flow and instruction flow are combined into a single self-sustained flow. *Recurrence* is the property of dynamic development of the trajectory of the computational process by calculating new values of tag fields of self-sustained data by means of tag transformation.

This paper discusses the technical aspects of the quad-core MRDA prototype implementation and the results of its experimental approbation for the DSPs tasks. The prototype includes tools to support concurrency at different levels (threads, commands and operations of individual instructions), as well as hardware supported DSPs (including specialized instructions and various types of memory constants). The problem of isolated words recognition (IWR) and a reference DSPs algorithm – fast Fourier transform were selected as a demonstration tasks.

The FFT and the majority of IWR algorithms have been implemented and modeled at software and hardware models. Comparative results of simulation of these algorithms on the MRDA architecture by the calculation time parameter are shown (in the number of logical steps) with similar results for a single-core DSP of a traditional architecture. The obtained values of the acceleration coefficients ***range from 2 to 17***, which allows estimating the potential speed of the DSP designed on the basis of the MRDA.

II. MULTICORE RECURRENT DATA-FLOW ARCHITECTURE PROTOTYPE OVERVIEW

During the search for an acceptable implementation of the ideas laid down in the MRDA, it was established that the most appropriate option for implementing a recursive signal processor (RSP) based on FPGAs is a hybrid two-layer RSP architecture (HARSP) with a leading von Neumann processor on the control (upper) level (CU) and a number of data-flow processors at the lower level – the recurrent operating units (ROU) [6]. The interaction between the CU and the ROU is carried out by means of a special dual-port buffer memory (BM) that provides simultaneous access to read and write for both CU and ROU. Single-port access conflicts are resolved in favor of the ROU. Total BM capacity is 8196 64-bit operands and additional cell vacancy bits.

The key HARSP component is the ROU, for which both imitational software as well as hardware VHDL models have been created [7]–[8]. Fig. 1 shows ROU architecture. CM_S –constants memory sectional; CM_SR – constants memory sectional register; CM_SL – constants memory sectional loadable; C – the computer (in the current implementation amounts to 4 units), which includes the component TT – tag transformer.

For most data-flow architectures, it is typical to use a resource-intensive component that produces the matching

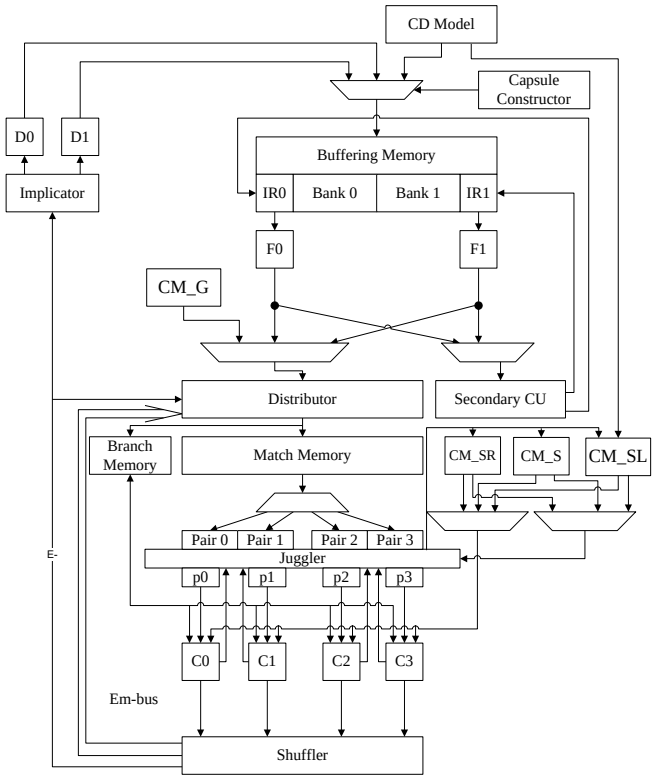

Fig. 1. ROU Architecture

pairs of tagged data – associative memory. In the current implementation of the ROU, the universal associative memory has been replaced with a directly addressable small-volume Match memory (MM - 16 cells in this implementation) with an additional data ready bit (DR-bit) in the addressable cell. This decision imposes strict restrictions on the sequence of input data for the ROU, because it removes the computational step context support. At the same time, this data structure requires significantly less component overhead, has higher energy efficiency and faster performance than associative memory.

In the current version of MRDA, self-sustained data are represented as operands of specialized types: data-containing (carry the data), assistant (global configuration options) and control (precise configuration or special usage). Within the ROU, the executable program is a sequence of operands called the capsule. Buffer memory stores a capsule template that is filled with data by CU. Fig. 2. shows a fragment of capsule listing for FFT computation.

The recurrence property is represented in the current version of MRDA via the universal tag transformer (TT), which executes logical right shift of each tag set on every computational step. TT reconfiguration for the specific transform function is a heavily time-consuming process (as shown in [3] for dynamically reconfigurable FPGA). According to the developers, the use of a universal TT is sufficient for problems from the selected DSPs area. In paper [9] the convergence of the recursive computational process was proven (universal TT is a special case). The technical

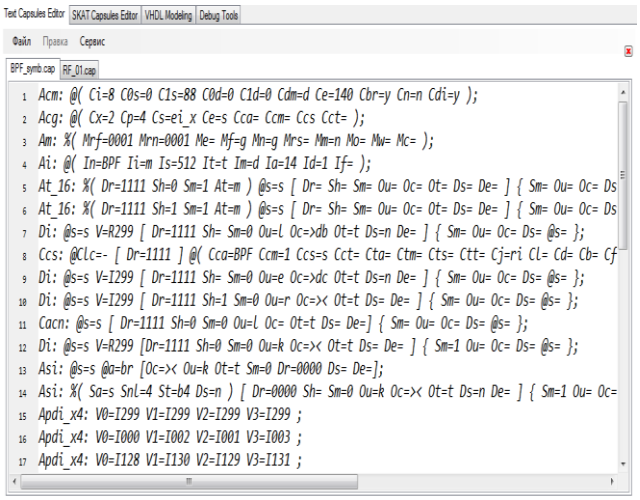

Fig. 2. FFT capsule listing fragment

implementation of such a mechanism for the functioning of the TT has required to introduce redundant tag fields into the operands allocated by delimiters ″ {″ and ″} ″ (Fig. 2).

ROU computational process is organized as follows:

1) Distributor component selects operands from BM and sends them to the appropriate streams (sections), specified by tag fields;

2) Match memory compares tag fields and forms operand pairs;

3) Juggler separates the self-sustained data flow into data and instruction flows, as well as ordering them to the appropriate inputs of the Computer;

4) Computer performs computation of the results and tag transformation;

5) Shuffler transfers the obtained results between parallel flows, as well as into the Implicator;

6) Implicator writes output data into the BM.

Such organization of instructions processing in the established computing process will be carried out in 2.5 steps. Thus, it can be concluded that a two-stage pipeline should be sufficient for the most effective implementation of the mechanism incorporated in the MRDA. However, real tests showed that in the case of combining the Distributor, MM and Juggler components into one stage of the pipeline, its performance is much lower than the performance of the stage at which the Computer and Implicator are located. Because of that, the Distributor was separated to the third pipeline stage.

This solution significantly expanded the functionality of the Juggler component and reduced the limitations imposed by the rejection of associative memory. This is achieved due to the added functionality of resolving contradictions between the fields of matched operands, providing interaction with Branch Memory and various subsets of the Constants Memory, and also by separating and preparing data and instructions for superscalar processing on the Computer.

III. HARSP ARCHITECTURAL FEATURES FOR DATA-FLOW DSP ARCHITECTURE ISSUES SOLUTIONS

A. Superscalar Computer organization. Recursive algorithm support

Recursive algorithms implementation is a common problem of parallel and data-flow systems. Such algorithms are characterized by high data dependency. At the same time, most DSPs algorithms use recursion in one form or another. Therefore, one of the main tasks that have to be solved when developing a DSP based on data-flow architecture is to support recursive computations.

The main mechanism for supporting recursive computations in the ROU is recursive convolution performed at the programming stage and recurrent involution (transformation of tags) carried out by TT. The state of the recursive computational process is saved in the functional fields that are stored in the MM as well as in the intermediate operands. This approach allows the implementation of any necessary depth of recursion. In addition, the Computer also contains a number of additional mechanisms for recursive computations support.

Computer contains: set of buffer registers L, R, B, C and accumulator A; hardware multiplication block (M); ALU16 (in this implementation, 16-bit) and 40-bit arithmetic unit (AU40); shift and round off logic (BS & R); conditional branch processing support blocks. In addition to the input buses L- and R-, each of the buffer registers and the accumulator, as well as the Constant Memory bus, can be data sources for binary operations. In addition, each of the blocks - M, ALU16, AU40, BS & R - can function simultaneously in case the necessary data is provided. Fig. 3 shows the detailed structure of the Computer component.

Thus, the Computer has a super-scalar architecture. In this implementation of the ROU, multiply-accumulate operations are supported, as well as two superscalar modes covering a large variety of possible combinations of pairs (in rare cases triples) of operations that can be performed within the context of one computational step. This, in turn, provides efficient transition between the next steps of the recursive algorithm without any loss in accuracy or additional logical steps that are necessary to ensure the support of strong dependence on data.

B. Constant processing mechanisms

Most DSPs algorithms use constants in one way or another. In this regard, all modern DSP processors use specialized memory types called constants memory. The proposed architecture is no exception. Fig. 1 shows that HARSP has 4 different types of constants memory, each capable of storing sequences of constants and tag fields in a form of self-sustained data.

Each of the constants memory types has its own areas of accessibility and mechanism of functioning. Purpose and specific mechanics of each of the memory types are described below.

1) CM_G - global constants memory, available at the Distributor level. As well as Distributor, CM_G is a central

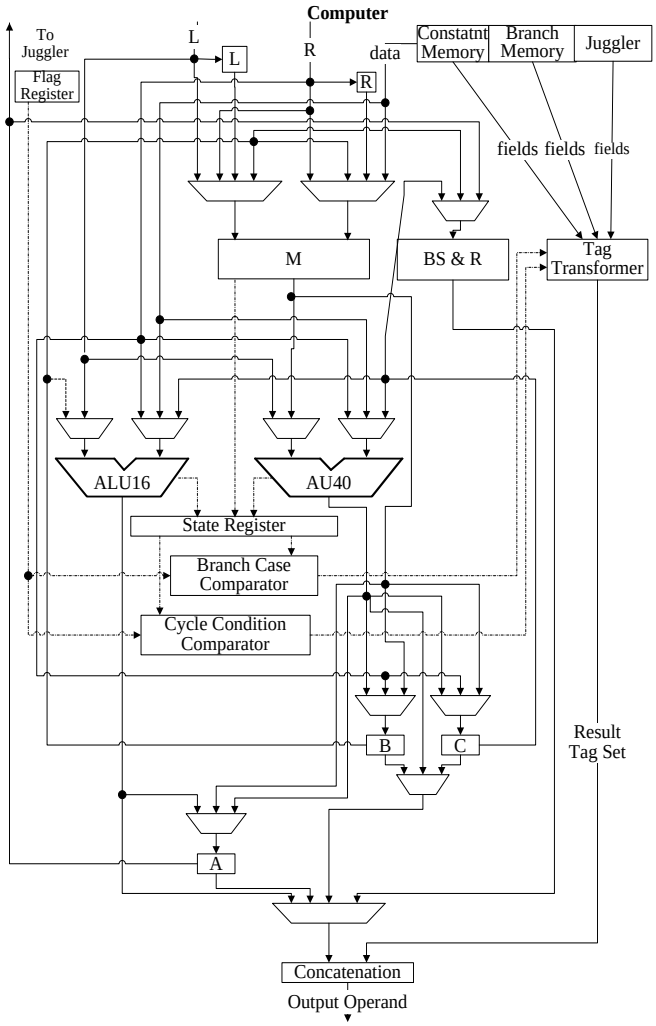

Fig. 3. Computer structural scheme

resource for all sections. Serves for loading constants that are transferred using the same rules as input date coming from BM. This approach allows for loading of full operands into Distributor for further use by Juggler in resolving contradictions in tag fields of pair coming from MM.

2) CM_S - constants memory sectional, available directly to Computer. There are as many modules of this type of memory as there are sections. This is the most commonly used type of constants memory. It is necessary for superscalar operations involving three or more input data sources. If required, it can provide data for TT.

3) CM_SR - constants memory sectional register, available to Juggler for writing and Computer for reading. It has small fixed size and can be used if the amount of constant data is very limited. It is loaded by special control operands and used similarly to CM_S.

4) CM_SL - constants memory sectional loadable, available to control level for writing and Computer for reading. It is used in case if the volume of constant pool is too big for CM_S. For example, Viterbi algorithm imple-

menting recognition by finding probability extremum by comparing it with parameters of models of all the recognizable words in the database. It is clear that each such parameter is a constant but thousands of them have to be processed. CM_SL has small size and is constantly updated within computing process.

C. Cycles support in ROU

Many DSPs algorithms include accumulation of data or repeating the same part of program over defined number of iterations, and then change the behavior. Therefore, special tools for cycle operations support have been added to HARSP.

Iterator component has been added to the Distributor. Iterator creates special control operands with pre-set delay. These operands can use value in accumulator A as a second part of the data pair and have full set of tag fields. Tag fields can be used by Juggler to resolve contradictions as well as to overcome the limitations of universal tag transformer. In other words, using such operands we can organize cycle procedures with set number of iterations able to work without loading data from capsule.

Another resource supporting cycle procedures in ROU is the block shown on Fig. 3 as "cycle condition comparator". This block includes customizable counter and transition mechanism that activates when counter reaches zero value. Within the ROU transition is defined as TT choosing tags from Branch Memory instead of Juggler. End of cycle condition in ROU can be tested in "short" and "long" modes.

"Short" mode means that transition can occur on the same step as counter was modified. This effect is achieved by using the special regime of the Computer. The information responsible for activation of this regime is held in *Ot* tag field (for example operand 7 in Fig. 2). Organizing counter checking and transitions in such a way allows us to perform the cycle procedures with the body of small size (5-10 computational steps) with maximum efficiency. "Short" mode has a disadvantage in a fact that *Ot* field is a constant and cannot be transformed by TT which can lead to wrong tag fields on further operands and has to be tracked carefully.

"Long" mode is when counter check-up for zero value is initiated by special operand – cycle controller. This type of check organization is called "long" because it uses the whole computational step. This approach has some advantages. In case when cycle body is relatively big (includes large amount of computational steps) "long" mode can prove itself useful. Firstly, we don't need to modify *Ot* field and secondly – cycle exit procedure can be formed for multiple sections simultaneously (for example when cycle bodies are the same for all the sections). It is worth noting that conditional branching in ROU follows the exact same pattern.

D. Program repetition and reusing output data

Some DSP algorithms imply computing a parameter to then refine it as a result of multiple iterations. For such algorithms, ability to repeat program code (in ROU case – repeated use of capsule) as well as rewriting output data for repeated use on future iterations can be useful.

Within the ROU output data can be processed in two regimes. In first regime output data are simply accumulated and sent to control level for processing. Such data are placed in output section of capsule. In second regime output data are rewritten into capsule template into set addresses. Thus, capsule is getting prepared for the next iteration.

Program repetition is implemented in ROU using Support control unit (SCU) that is responsible for setting the boundaries of capsule's fragment to be repeated. SCU also controls the addressing in BM, thus controlling the flow of data into Distributor. Such regime was used most fully in our implementation of FFT. In particular, hardware support for FFT was implemented in ROU, including bit-reverse addressing and "butterfly" operation primitive.

IV. HARSP FEATURE DEMONSTRATION ON FAST FOURIER TRANSFORM IMPLEMENTATION

In order to demonstrate architecture potential of HARSP FFT Radix 2 (256 points) algorithm was chosen as it is a standard used in most DSPs applications. The heart of the problem is to compute 1024 ″butterfly″ operations (128 operations in 8 steps). Implementation efficiency has been assessed based on computational time (in logical steps). Computational speed was compared between HARSP implementation and single-core microcontroller dsPIC30F [10].

According to [10] library FFT implementation on dsPIC30F is computed in 476 μs on 40 MIPS, which means it takes roughly 19000 instructions including all data rewrite operations.

For HARSP implementation, the time cost of data rewrite can be neglected because operations on intermediate data are taken care of by hardware FFT support and are parallel to the computational process. Fig. 2 shows a fragment of FFT capsule consisting mostly of packaged operands, which helps to reduce capsule size upwards of 4 times. Fig. 4 shows fragment of FFT HARSP graph-capsule. It is shown for 2 sections because computational process in parallel sections is completely identical.

Each section computes 4-cycle hardware supported "butterfly" operation. This way 4 "butterflies" are calculated in 4 computational steps. Therefore, including reconfiguration costs it takes 130-135 steps to compute 128 "butterfly" operations, which adds up across 8 steps of algorithm to 1024 "butterflies" in 1100 computational steps. Such an impressive result is achieved thanks to superscalar computational blocks of ROU, constants memory, capsule repetition (8 steps) and rewriting data into capsule. As shown in Fig. 4, at each computational step multiple actions take place in superscalar mode, such as: summation, multiplication with inputs from constants memory, rounding, writing data into internal registers. This process makes good use of all the technical solutions described earlier.

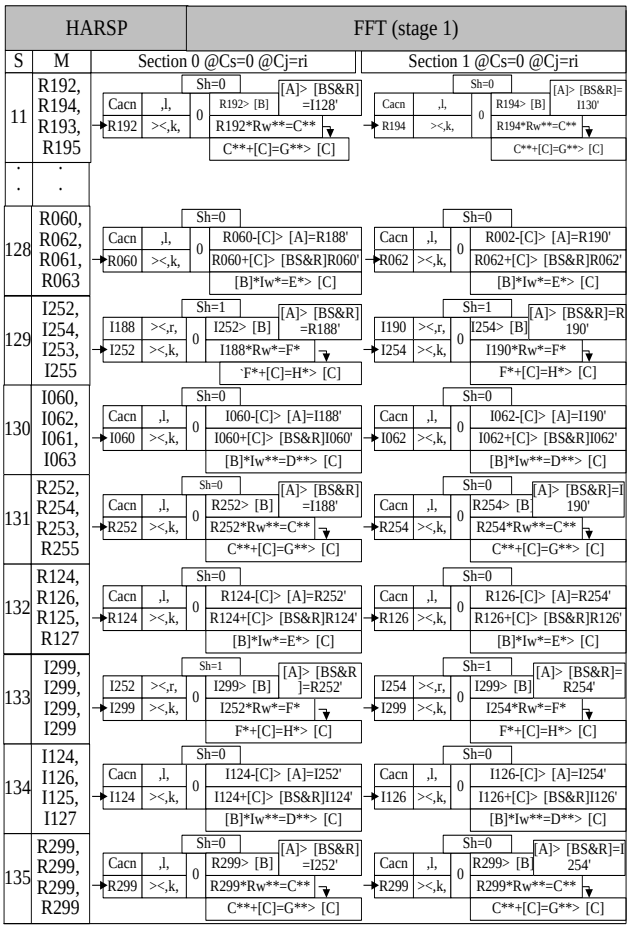

Fig. 4. FFT graph-capsule fragment

Microcontroller dsPIC30F has been chosen as a reference point because IPI RAS team created the commercial implementation of word recognition system for that processor (recognition library in [10]). We consider the given estimations objective because they are based on real results in IWR field.

V. HARSP HARDWARE AND SOFTWARE MODELLING RESULTS

For experimental approbation of HARSP, the IWR problem implemented earlier for dsPIC30F microcontroller for Microchip company, as well as standard Radix 2 256 point FFT algorithm, were chosen. Specialized methodology for developing capsules within HARSP is presented in [9]. Almost all of the IWR algorithms were implemented for ROU by applying aforementioned methodology. Software and hardware platform GAROS IDE [11] helped simulate each of the implemented algorithms and estimate its computational speed. It is worth mentioning that acceleration factor estimates presented below are subject to the following limitations:

1) dsPIC30F microcontroller is based on von Neuman architecture;

2) HARSP has 4 sections and 4 cores accordingly, while dsPIC30F is single-core;

3) HARSP computing blocks have a much wider superscalar regime support, compared to dsPIC30F, that has very limited command pool defined for such semantics;

4) It is inferred that all the input data is either ready or is loaded by control level in time during computational process. Thus, time it takes to prepare the data was not taken into account for our estimations for both dsPIC30F and HARSP.

Besides, some capsules were implemented in two ways to find the one that shows higher performance. Variant 1 means that algorithm was implemented to use all 4 computational cores and variant 2 means it was implemented for four sets of input data computing each on its own core. Comparison results for chosen algorithms are presented in Table 1. Some of the results are preliminary (marked with "~") due to the fact that software and hardware modeling tools need to be revised and such revision can lead to reiterating on capsules.

Table 1

Word Recognition Algorithms Implementation Results

Algorithm Name	**dsPIC30F logic steps**	**ROU logic steps**	**Amplification coefficient**
FFT2_256	~19000	~1100	~17,2
Butterworth filter	1360	437	3,11
Band filter (single band)	1428	442	3,23
Natural logarithm x4	36*4	26	1,38*4
RASTA filter	153	45	3,4
Exponent x4	32*4	20	1,6*4
Cosine inversed DFT	36	17	2,12
Durbin recursion	~800	~124	~6,5
PLP parameters	144	32	4,5
PLP parameters *)	144	30	4,8
Viterbi algorithm	91*N-143	99*N	$\left(1-\frac{8*N+143}{99*N}\right)*4$
N – is an amount of observations in observation vector (N>5)			

VI. CONCLUSION

Technical decisions taken while implementing MRDA prototype allowed us on one hand to overcome some problems typical for data flow architectures, on the other hand to get a sizable performance increase compared to traditional DSP processors. The most important result is estimated efficiency of hardware support for FFT algorithm. Combining different ROU mechanisms, we reached acceleration coefficient upwards of ***17*** compared to traditional single-core DSP. Furthermore, acceleration coefficient estimations for most IWR algorithms are in ***3 – 6*** range which is a good result for quad core implementation.

Studying capsules that are implementing various DSPs algorithms we have estimated average computational resources use rate to be around 60-70%. Yet the computational resources use rate for FFT, cosine inverse DFT, Viterbi, Natural logarithm and exponent (variant 2) algorithms approaches 100%. Obtained result proves high efficiency of technical decisions taken in MRDA prototype.

It is planned to implement MRDA prototype for 32-bit floating point data based on self-timed hardware, which will allow for higher energy efficiency and fault tolerance. Our team has extensive experience in developing self-timed 32/64-bit co-processor hardware: 64-bit device of division and of square-root extraction [12]; device of multiplying with accumulation and 64/32-bit device of multiplication-addition with a single entry rounding [13].

Nevertheless, obtained results show the need of further development and improvement of the new architecture to increase the average resource use coverage and MRDA end performance.

ACKNOWLEDGEMENTS

In conclusion, we want to express our gratitude to N. V. Morozov, Yu. G. Diachenko and Yu. V. Rogdestvenskii for an invaluable contribution to the development of hardware model of MRDA.

SUPPORT

The study was partially supported by the 2016's subprogram no. 4 of ONIT RAS department (Project 0063-2015-0016 III.3).

REFERENCES

[1] E. A. Lee and J. C. Bier. Architectures for Statically Scheduled Dataflow. Parallel Algorithms and Architectures for DSP Applications, edited by Magdy A. Bayoumi. Dordrecht. Kluwer Academic Publishers, 1991. pp. 159-190.

[2] V.P. Srini. DFS-SuperMPx: Low-cost Parallel Processing System for Machine Vision and Image Processing. Proc. Third International Conference "Parallel Computing Technologies", PaCT-95. St. Petersburg, 1995, pp. 356-369.

[3] S. Voigt, M. Baesler, T. Teufel. Dynamically reconfigurable dataflow architecture for high-performance digital signal processing. Journal of Systems Architecture. 2010, no. 56, pp. 561-576.

[4] Yu. Shikunov, D. Khilko, Yu. Stepchenkov. Hardware and Software Modelling and Testing of Non-Conventional Data-Flow Architecture. Proceedings of the 2016 IEEE North West Russia Section Young Researchers in Electrical and Electronic Engineering Conference (ElConRusNW), 2016. pp. 360-364.

[5] Yu. Stepchenkov, V. Volchek, V. Petrukhin, A. Prokofyev. Mehanizmy obespechenija podderzhki algoritmov cifrovoj obrabotki rechevyh signalov v rekurrentnom obrabotchike signalov [Hardwere maintenance for digital processing of speech signals in the recurrent dataflow processor]. Sistemy i Sredstva Informatiki [Systems and means of informatics]. 2010, 20(1), pp. 31-47 (in Russian).

[6] Volchek V.N., Stepchenkov Yu.A., Petrukhin V.S., Prokofyev A.A., Zelenov R.A. Cifrovoj signal'nyj processor s netradicionnoj rekurrentnoj potokovoj arhitekturoj [Digital Signal Processor With Non-Conventional Recurrent Data-Flow Architecture]. Problems of Perspective Micro- and Nanoelectronic Systems Development - 2010. Proceedings, edited by A. Stempkovsky, Moscow, IPPM RAS, 2010. P. 412-417 (in Russian).

[7] Khilko D. V., Stepchenkov Yu. A, Diachenko Yu. G., Shikunov Yu. I., Morozov N. V. Apparatno-programmnoe modelirovanie i testirovanie rekurrentnogo operacionnogo ustrojstva [Hardware and Software Modelling and Testing of Recurrent Operational Unit]. Sistemy i sredstva informatiki [Systems and means of informatics]. 2015. Vol. 25. No. 4, pp. 78-90 (in Russian).

[8] Khilko D. V., Shikunov Yu. I., Stepchenkov Yu. A. Osobennosti programmnoj realizacii imitacionnoj modeli potokovoj rekurrentnoj arhitektury [Multicore Recurrent Data-flow Architecture Imitational Model Implementation Features]. Trudy Vtoroj molodezhnoj nauchnoj konferencii «Zadachi sovremennoj informatiki [Proc. of the Second youth scientific conference] Modern Problems in Informatics], Moscow, FRC ITCS RAS, 2015. pp. 220-227 (in Russian).

[9] Khilko D.V., Stepchenkov Yu.A. Teoreticheskie aspekty razrabotki moetodologii programmirovanija rekurrentnoj arhitektury [Theoretical Aspects of Recurrent Architecture Programming Methodology Development]. Sistemy i sredstva informatiki – [Systems and means of informatics]. 2013. 23 (2), pp, 133-156 (in Russian).

[10] URL: http://microchip.com.ru/Support/Download/13_64.pdf (accessed 04.04.2016).

[11] Khilko D.V., Stepchenkov Yu.A., Shikunov Yu.I. The instrumental software development environment for hybrid architecture of recurrent signal processor (GAROS IDE). Certificate of state registration of computer program No. 2015614004 from 01.04.15.

[12] Y. Stepchenkov, Y. Diachenko, V. Zakharov, Y. Rogdestvenski, N. Morozov, and D. Stepchenkov. Self-Timed Computing Device for High-Reliable Applications // Proc. International Workshop on power and timing modeling, optimization and simulation (PATMOS'2009), Delft, Netherlands, 2009. pp. 276-285.

[13] Sokolov I.A., Stepchenkov Y.A., Rozhdestvenskii Y.V., Diachenko Y.G. Samosinhronnoe ustrojstvo umnozhenija-slozhenija gigaflopsnogo klassa: metodologicheskie aspekty [Self-timed multiply-add unit of gigaflops range: methodological aspects]. Problems of Perspective Micro- and Nanoelectronic Systems Development - 2014. Proceedings, edited by A. Stempkovsky, Moscow, IPPM RAS, 2014. Part IV. pp. 51-56 (in Russian).