Hardware and Software Modelling and Testing of Non-Conventional Data-Flow Architecture

Yuri Shikunov[1], Dmitry Khilko[2], Yuri Stepchenkov[3]
Advanced computer systems architectures dept.
Institute of Informatics Problems, Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences,
(IPI FRS CSC RAS), IPI RAS
Moscow, Russian Federation
[1]YIShikunov@gmail.com, [2]DHilko@yandex.ru, [3]YStepchenkov@ipiran.ru

***Abstract*— This paper covers new recurrent data-flow computational model, as well as architecture that implements principles and ideas of this model. Basic differences of this model from the existing ones and examine key aspects of this new computational model including its implementation in the form of Hybrid Recurrent Architecture of Digital Signal Processor are described. The approach and methodology of hardware and software modelling and testing based on new architecture are being proposed. We introduce the model of implementation of the proposed architecture as well as imitation modelling tools of recurrent data-flow architecture, implementing said model. Functionality of imitation model and its role in software development suite for new architecture software development is being described. We introduce the notion of the target modelling platform called GAROS IDE. The results of platform testing on several subtasks of isolated words recognition problem are presented.**

Keywords—data-flow computing; multicore processing; software prototyping

I. Introduction

Since the creation of the first computer, information technology market was completely dominated by von Neumann (VN) Architecture. But, according to experts, the era of that architecture is coming its end. International Technology Roadmap for Semiconductors (ITRS) notes that the evolutionary development of the von Neumann model of the processor which dominated the last fifty years in the market, comes to naught [1].

The most famous attempts to get away from the von Neumann model of computation and provide "natural" parallelism are data-flow machines and reduction machines. Both types of machines are providing order of execution – "by preparedness" or "on execution" A departure from the traditional principles of VN architecture and the transition to the organization of the computational process (CP) by the data availability (data flow) in principle, allows the maximum time parallelization of the CP. However, effective implementation of data-flow architectures encounters a number of significant problems such as the problem of implementation of recursion, cycles, iterations, constants and others.

Data-flow architecture is most suitable for non-recursive algorithms. Typical tasks of this nature are transforms and FIR filters, which are the main tasks for parallel limited dimension computing in the field of signal processing, such as the problem of recognition of isolated words [2].

While researching the ways to improve the data-flow model, an idea for the new data-flow model has been proposed [3], [4] which subsequently was developed and refined by us. Testing of the proposed architecture and performance evaluation of the implementation of parallel limited dimension computing is carried out in the field of digital signal processing (DSP) in the first phase. The resulting computational model have been called "recurrent data-flow model" [5].

On this basis we developed the multicore recurrent data-flow architecture (MRDA) for implementing the parallel limited dimension computing in the field of signal processing. Currently, a prototype called recurrent signal processor is implemented. It consists of classic VN processor as a controller unit and several data-flow processors – recurrent operational units (ROU) [6]. That architecture implementation is called "Hybrid Architecture for Recurrent Signal Processing" (HARPS), which allowed testing and debugging of developing system using existing tools created for VN architecture.

The uniqueness of the recurrent computation process of the MRDA means that existing programming methodologies (structural, functional, object-oriented, and others) does not fully meet the needs of developers. Due to those aspects, a need to develop a new methodology called recurrent data-flow programming methodology has emerged [7].

One of the reasons hindering the implementation of the potential benefits of data-flow architectures are the use of methods and means of the design, established and used as part of the synchronous style of equipment design. To remove this contradiction and support data-flow architectures special synchronous flow graphs have been developed. They are the main tool for data-flow architecture modelling. The contradiction has been removed at the cost of underutilization of asynchronous data-flow concept potential.

The suggested methodology utilizes that potential and supports architecture implementation with the self-timed circuitry. Achieving that conceptual unity of the asynchronous

* The reported study was funded by RFBR according to the research of time costs project № 13-07-12068 ofi_m).

nature of the recurrent paradigm at the architectural level and self-timed hardware implementation allows us to exploit algorithmic parallelism in the most natural and effective way. In the process of implementation of the new architecture a method of hardware and software simulation was developed. A set of models and a prototype have been developed under the procedure [8]. In order to achieve the highest possible performance of proposed architecture, the development of appropriate software optimized to perform in said environment is necessary. The specific features of the architecture, not inherent in other classes, made it impossible to apply the known methods and programming techniques in full. Therefore there is a need to develop new methodological framework for efficient programming in MRDA environment [7]. It was also necessary to develop software to support software development.

By integrating a set of models, software development tools, and methodological framework GAROS IDE was developed [5], [7]. By testing this software and hardware at the problem of recognition of isolated words we obtained results, which allow us to speak about the potentially high efficiency of this architecture for a given domain.

II. MRDA Features and Distinctions

Paper [6] provides an analysis of different architectures of computing systems that identifies several fundamental classes of them.

First fundamental class consists of architectures based on von Neumann principles. In order to execute the program with a classical VN architecture a certain amount of storage that is functionally (and for Harvard architecture physically) divided into instructions and data areas is required. The instruction flow being extracted from program memory of CPU (instruction flow has a primary role) initiates computing. The initiating program is fully stored at instruction memory in static form. Thus there is a problem to identify the moment of data preparedness for computing. Such computational model was classified as Control Flow/Static (CF/S).

Second fundamental class consists of data-flow (DF) architectures. 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). Furthermore data memory stores data (operands) with additional functional fields (tagged fields) in its cells. Thus total amount of memory used is approximately the same. Generally functional fields are used to store information about instruction address to be extracted from program memory. The instruction set is fully stored in static form as well. Therefore such model was classified as Data-Flow/Static (DF/S).

The new computational model being proposed belongs to the class of data-flow models, thus it inherits the majority properties of them. However there are several essential properties of recurrent data-flow computational model it stands out drastically:

- Proposed model combines a traditional for existing computational models pair of data and instruction flows into single ″self-extracting″ flow of self-sustained data. Due to this particular feature and other architectural solutions of MRDA described below the execution time of single instruction can be halved in comparison with other architectures.
- One of the most important aspects of data-flow architecture is an implementation of ″firing rule″ and corresponding memory organization, which would provide comparison process for tagged data. Within the MRDA ″match memory″ has been implemented. This component is a specific variation of memory organization and makes use of most advantages of ″associative memory″ that is a common memory type for data-flow architectures. Simultaneously ″match memory″ is very energy-efficient in comparison with ″associative memory″. However ″match memory″ provides lesser flexibility.
- The main property of new computational model has been called ″recurrency″, which is the dynamic evolution of computational process. In paper [5] is said: ″In terms of self-sustained data each following step is calculated as a function of previous 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 recurrency principle. Within the MRDA ″Tag Transformer″ (TT) component is included as a part of CPUs. It provides recurrent self-extraction functionality of computational process. Tag Transformer operates in parallel with CPU and determines next computational step action (transforming functional fields).
- Match memory doesn't 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/Dynamic (DF/D).

The comparison by structure and instruction execution principle of fundamental classes described is illustrated at Fig. 1 and Fig. 2 respectively.

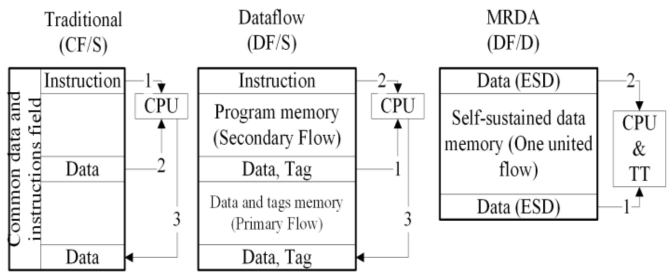
Traditional (CF/S)
Dataflow (DF/S)
MRDA (DF/D)
Common data and instructions field
Instruction
Data
Data
CPU
1
2
3
Instruction
Program memory (Secondary Flow)
Data, Tag
Data and tags memory (Primary Flow)
Data, Tag
CPU
2
1
3
Data (ESD)
Self-sustained data memory (One united flow)
Data (ESD)
CPU & TT
2
1

Fig. 1. Fundamental classes comparison by structure

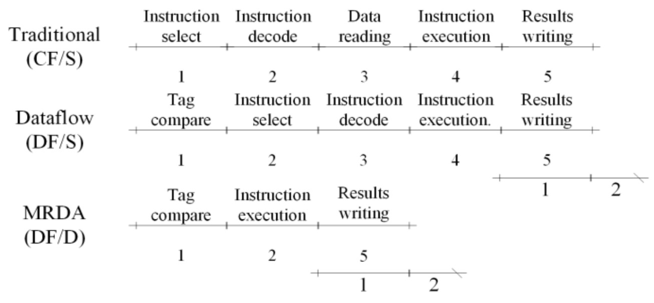

Fig. 2. Instruction execution is compared classes

III. Methods and Means of Architecture Modelling

A. Methods of Modelling and Debugging

By analogy with the HIL methodology [9] hardware and software simulation and debugging methodology for HARSP was developed:

1) Developing of the mathematical (imitational) model, using the C# programming language as a platform;

2) Developing of VHDL-model using Altera software products;

3) Development of test that verify the correspondence of imitation and VHDL models and their subsequent modelling;

4) Project implementation on the Altera FPGA;

5) System debugging;

6) Integration of steps 1) – 5) into a single iterative design cycle;

7) Establishment of software development tools and unified modelling environment that integrates the above models and tests;

8) Iterative development and debugging of the HARPS by means of a unified modelling environment.

With the use of this technique RPS imitation model has been created.

B. Imitational Model Description

There are several basic approaches for imitation model (IM) development. To implement our model discrete event modelling paradigm has been chosen. The essence of this approach is to present the functioning of the system as a chronological sequence of events.

.NET Framework platform and C# programming language software environment has been chosen as a mean for imitation model implementation. A set of classes implementing the basic elements of discrete-event approach (clocks, lists of actions and events, a set of states, etc.) have been created for each model component. We have created a cross component interaction mechanism as well. RSP computational process was split into 3 conveyor stages. On Fig. 3 we can distinguish 3 main groups of components that form the stages of conveyor: Buffering Memory + Distributor + Secondary Control Unit (Stage 1), Pair Memory + Branch Memory + Constants Memory (Stage 2) and Computer + Shuffler + Implicator (Stage 3). Combined state of all the components in one stage determines the state of that stage.

Conveyor implementation of RSP required the stage synchronization on each computational step. To solve this task locking mechanism has been applied. For that we implemented 3 functional primitives: DoReadBusData, DoStep, DoFinalizeStep and several states to work with them. These primitives implement basic synchronization points, allowing management of computing threads and their synchronization at every step. Fig. 4 illustrates that interaction.

Paper [10] covers said model implementation in depth.

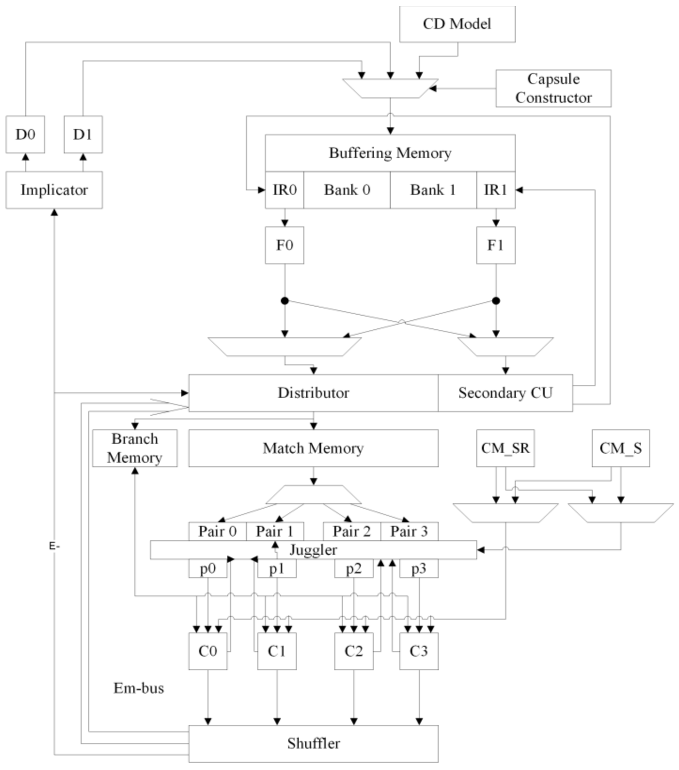

Fig. 3. RSP Imitation Model Architecture

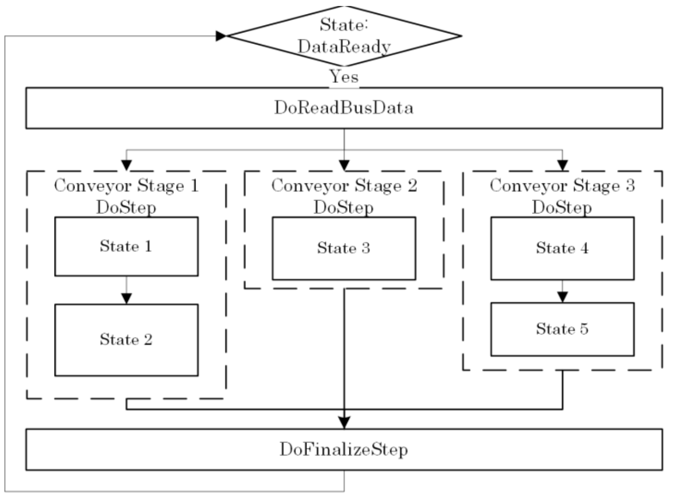

Fig. 4. Cross-component interaction flowchart

IV. INTEGRATED DEVELOPMENT ENVIRONMENT GAROS

During the research of HARSP programmability recurrent data-flow programming methodology have been developed [7]. In that methodology the software design process, its key stages as well as debugging methods are successfully formalized.

Formal representation of HARSP programming workflow became the foundation for prototyping of integrated development environment called GAROS IDE which capabilities cover all stages introduced in said methodology.

We are currently working on improving GAROS IDE, which purpose evolved from simple programming environment to the full-scale complex of programming, modelling and debugging. Paper [5] presents overall structure of that environment prototype, its architecture and component description. Fig. 5 shows GAROS IDE structure.

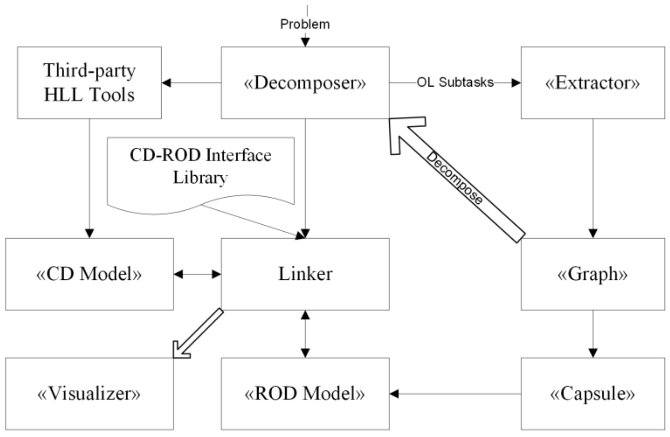

Fig. 5. GAROS IDE components architecture

V. CONCLUSIONS

This paper aggregates main results obtained during development and debugging process of new computational model and architecture based on it. Methods, hardware and software have been developed to be used for MRDA demonstration with word recognition problem. This problem has been chosen due to its earlier realization within Microchip collaboration on dsPIC30F microcontroller. The results obtained for dsPIC30F have been used as an algorithmic baseline for MRDA and its prototype HARPS potential performance estimation.

In order to demonstrate potential performance of HARPS word recognizer has been decomposed into set of algorithms. Each of those has been evaluated individually against dsPIC30F implementation by the amount of computational steps. The obtained results are presented in Table 1.

Following test conditions have been used to compare HARPS and dsPIC30F performance:

- dsPIC30F is based on VN architecture,
- HARPS has four CPUs, while dsPIC30F has one,
- both devices are capable of functioning in superscalar mode
- data preparation time has not been taken into consideration
- both devices employ special instruction sets optimized for digital signal processing tasks.

The development and approbation results of new computational model and architecture based on it presented in this paper are found to be a successful attempt to create non-traditional data-flow architecture in DSP domain. However a number of problems have not been solved yet. Thus further research and development in this area has to be done.

TABLE I. WORD RECOGNITION ALGORITHMS IMPLEMENTATION RESULTS

Algorithm Name	**dsPIC30F logic steps**	**ROU logic steps**	**Amplification coefficient**
Butterworth filter (one section)	679	269	~2,52
Butterworth filter (two sections and two filters)	2760	682*)	~4,05*)
Band filter (single band)	1428	442	3,23
Natural logarithm, Exponent	36	19	1,89
Natural logarithm, Exponent x4	36*4	26	~1,38*4
RASTA filter	153	45	~3,4
Cosine inversed DFT	36	17	2,12
Durbin recursion	~440	~72	~6,1
PLP parameters	144	37	~3,9
PLP parameters*)	144	22*)	6,55*)
Viterbi algorithm	91*N-143	99*N	$\frac{(1-(8*N+143)}{(99*N)}*4$
*) modified variation of architecture specification N – is an amount of observations in observation vector (N>5)			

References

[1] ITRS System Drivers report (2005) [Online] Available: http://www.itrs.net/ITRS%201999-2014%20Mtgs,%20Presentations%20&%20Links/2005ITRS/SysDrivers2005.pdf

[2] I. Hartimo, K. Kronlof, O. Simula, and J. Skytta, “DFSP: A Data Flow Signal Processor”, IEEE Transactions on Computers, VOL. C-35, No.1, pp 23-33, Jan. 1986.

[3] T. Agerwala and Arvind, Data flow systems, IEEE Computer, 15 (Feb. 1982): 10-13.

[4] Arvind and R.A. Iannucci, A critique of multiprocessing von Neumann style, in Proc. 10th ISCA, June 1983: 426-436.

[5] Khilko D. V., Shikunov Yu. I. “Development of the Instrumental Environment for the Recurrent Data-Flow Computational Model”, Fourth “School of Young IPI RAS Scientists” conference, Proceedings of conference – M.: IPI RAS, 2013. – pp. 65-78 (In Russian).

[6] Stepchenkov Ju. A., Petruhin V.S. 2008. “The hybrid variant FPGA realization features of recurrent signal processor”, The Systems and Means of Informatics, Add.: pp. 118-129 (In Russian).

[7] Khilko D.V., Stepchenkov Yu.A. Theoretical Aspects of Recurrent Architecture Programming Methodology Development. // The Systems and Means of Informatics, 2013, Vol 23 №2, pp. 133-156 (In Russian).

[8] Khilko D. V., Stepchenkov Yu. A, Diachenko Yu. G., Shikunov Yu. I., Morozov N. V. “Hardware and Software Modelling and Testing of Recurrent Operational Unit” // The Systems and Means of Informatics, 2015 №4. pp. 78-90 (In Russian).

[9] Halvorsen Hans-Petter. Introduction to Hardware-in-Loop Simulation. Telemark University College. 2012.

[10] Khilko D. V., Shikunov Yu. I., Stepchenkov Yu. A. “Multicore Recurrent Data-flow Architecture Imitational Model Implementation Features” // Second youth scientific conference “Modern Problems in Informatics”, 2015. Proceedings of conference. pp. 220-227 (In Russian).