FPGA Implementation of RISC-based Memory-centric Processor Architecture

Danijela Efnusheva[1]
Computer Science and Engineering Department
Faculty of Electrical Engineering and Information Technologies
Skopje, North Macedonia

***Abstract*—The development of the microprocessor industry in terms of speed, area, and multi-processing has resulted with increased data traffic between the processor and the memory in a classical processor-centric Von Neumann computing system. In order to alleviate the processor-memory bottleneck, in this paper we are proposing a RISC-based memory-centric processor architecture that provides a stronger merge between the processor and the memory, by adjusting the standard memory hierarchy model. Indeed, we are developing a RISC-based processor that integrates the memory into the same chip die, and thus provides direct access to the on-chip memory, without the use of general-purpose registers (GPRs) and cache memory. The proposed RISC-based memory-centric processor is described in VHDL and then implemented in Virtex7 VC709 Field Programmable Gate Array (FPGA) board, by means of Xilinx VIVADO Design Suite. The simulation timing diagrams and FPGA synthesis (implementation) reports are discussed and analyzed in this paper.**

***Keywords*—FPGA; memory-centric computing; processor in memory; RISC architecture; VHDL**

I. INTRODUCTION

The growing technological progress over the last several decades has caused dramatic improvements in processor performances, providing speed-up of processor's working frequency, increased number of instructions that can be issued and processed in parallel, [1], [2], multithreading, pre-fetching, etc. According to Moore's law, [3], [4], the integrated circuits production technology has enabled doubling of the number of transistors on a chip every 18 months, which resulted with the creation of multi-core processors over the last decade. This trend of processor technology growth has brought performance improvements on the computer systems, but not for all the types of applications, [5]. The reason for such divergence is due to the bottleneck problem in the communication between the processor and the main memory (which is by default placed out of the processor), caused by the growing disparity of memory and processor speeds, [6]. Therefore, we can say that not long ago, off-chip memory was able to supply the processor with data at an adequate rate. Today, with processor performances increasing at a rate of about 70 percent per year and memory latency improving by just 7 percent per year, it takes a dozens of cycles for data to travel between the processor and the main memory, [7], [8], which is basically placed outside of the processor chip.

The computer systems that are used today are mainly based on the Von Neumann architecture, [9], which is characterized by the strict separation of the processing and memory resources in the computer system. In such processor-centric system the memory is used for storing data and programs, while the processor interprets and executes the program instructions in a sequential manner, repeatedly moving data from the main memory in the processor registers and vice versa, [1]. Assuming that there is no final solution for overcoming the processor-memory bottleneck, modern computer systems implement different types of techniques for "mitigating" the occurrence of this problem, [10], (ex. branch prediction algorithms, speculative and re-order instructions execution, data and instruction pre-fetching, and multithreading, etc.). In fact, the most applied method for approaching data closer to the processor is the use of multi-level cache memory, as faster, but smaller and more expensive data storage than the main memory. Regarding that, the research stated in [11] discusses that the capacity and the area of on-chip cache memory have shown steady growth, as a result of the increased number of on-chip processing cores, which have imposed even greater requirements to the memory system. For example, up to 40% of the chip area in Intel's 65nm processors is occupied by caches, [12], used solely for hiding the memory latency.

Despite the grand popularity of cache memory used in the modern computer systems, we should note that each cache level presents a redundant copy of the main memory data that would not be necessary if the main memory had kept up with the processor speed. According to [13], cache memory causes up to 40% increase of the system's energy consumption, because it adds extra hardware resources and requires the implementation of complex mechanisms, [14], for maintaining memory consistency. Besides that, the misses in cache memory bring unpredictability in the timing of the program, which is not very suitable for real-time systems.

On the other hand, the development of some powerful processor architectures, such as vector, [15], wide superscalar, [16], VLIW (very long instruction word), [17], and EPIC (explicitly parallel instruction computing), [18], did not achieve the expected success, because of their inability to provide fast and high throughput access to the memory system. Considering the difference between the processor and the memory speeds, we believe that the relatively small number of fast GPRs in the processor is the major obstacle for achieving high data throughput. This is mainly expected in the case of executing a program that works with larger data set that needs to be placed into the processor for a short time, but there are not enough free registers. Examples for such applications are:

processing of data flows, calculating vast logical-arithmetical expressions or traversing complex data structures, etc. In such cases, the high speed of access to GPRs doesn't bring many advantages, because the complete set of required data cannot be placed into the register set at the proper time. Therefore, the author of [19] purposes a register-less processor which uses only cache memory (inside and outside of the processor) to communicate with the main memory. Additionally, the authors of [20] and [21] suggest the use of Scratchpad memory as a small software-managed on-chip memory that is separated of the cache memory and can be accessed in a single proc. cycle.

A few decades ago, in the 1990ties, some researches predicted that the memory behavior would be preponderant over the global performances of the computer system. Their proposals suggested the design of "smart memories" that will include processing capabilities. Therefore, several memory-centric approaches of integrating or approaching the memory closer to the processing elements have been introduced, including: computational RAM, [22], Mitsubishi M32R/D, [23], DIVA, [24], Terasys, [25], intelligent RAM, [26] - [28], parallel processing RAM, [29], DataScalar, [30], and an intelligent memory system, known as active pages model, [31]. Within these memory-centric systems, the processor can be realized as some simple RISC or complex superscalar processor and may contain a vector unit, as is the case with the Intelligent RAM.

The aim of this paper is to develop a novel RISC-based memory-centric processor architecture, which suggests an integration of processor and memory on the same chip die and proposes removal of general-purpose registers and cache memory (inside and outside of the processor) from the standard memory hierarchy. Contrary to the other memory/logic merged chips, which mostly use the standard memory hierarchy model for data access, the proposed RISC-based memory-centric processor provides direct access to the data into its on-chip memory (without the use of explicit LOAD and STORE instructions) and includes specialized control unit that performs 4-stage pipelining of instructions, allowing every (arithmetical, logical, branch and control) instruction to be completed in a single tact cycle. If this logic is manufactured as an ASIC (application-specific integrated circuit) it cannot be reused for further extensions, so in this paper we are investigating the possibilities to utilize a reconfigurable hardware platform - Virtex7 VC709 FPGA board, [32]. In that process, we are developing a VHDL model of the proposed RISC-based memory-centric processor, and then we are simulating the functionalities of the proposed processor and analyzing the characteristics and the complexity of its FPGA implementation, by means of Xilinx VIVADO Design Suite. In fact, FPGA technology is very suitable for the purposes of this research since it represents a good compromise between performance, price, and re-programmability, [33].

The rest of this paper is organized as follows: Section II gives an overview of different techniques and methods used to alleviate the processor-memory bottleneck and also discusses several memory-centric approaches of computing. Section III presents the proposed RISC-based memory-centric processor, describing its basic architectural characteristics, including instruction set, addressing modes, pipelining support, data forwarding, access to on-chip memory, etc. Section IV presents simulations and synthesis results from the FPGA implementation of the proposed RISC-based memory-centric processor. The paper ends with a conclusion, stated in section V.

II. Current State

The extraordinary increase of microprocessor speed has caused significant demands to the memory system, requiring an immediate response to the CPU (central processing unit) requests. Considering that the memory price, capacity, and speed are in direct opposition, an ideal memory system cannot be implemented in practice, [2]. Therefore, today's modern computer systems are characterized with hierarchical memory, organized in several levels, each of them having smaller, faster and more expensive memory, compared to the previous level.

The hierarchical approach of memory organization is based on the principle of temporal and spatial locality, [1], [2], and the rule "smaller is faster" which states that smaller pieces of memory are usually faster and hence more expensive than the larger ones. According to that, cache memories have lower access time, but on the other hand they bring indeterminism in the timing of the program, as a result of the misses that can occur during the memory accesses (read or write). This is also confirmed with equations 1 and 2, which give the expressions for computing average memory access time and program execution time, accordingly. The relation between these equations is expressed with the CPI (cycles per instruction) parameter, which value depends on the average memory access time. Therefore, if many misses to intermediate memory levels occur, the program's execution time will increase, resulting in many wasted processor cycles.

Average memory access time =

$$= \text{Hit time} + \text{Miss rate} * \text{Miss penalty} \quad (1)$$

$$\text{Execution time} = \text{Instructions number} * \text{CPI} * \text{Clock period} \quad (2)$$

According to the previous assumptions, we can say that multi-level cache memories can cause reduction of the memory access time, but at the cost of additional hardware complexity, increased power consumption, unpredictable program's timing and extra redundancy in the system. Other techniques for memory latency reduction include a combination of large cache memories with some form of branch predictive speculation, or out-of-order execution, [14]. These methods also increase the chip area and cause extra complexity on both the hardware and software level. Even other more complex approaches of computing like vector, wide superscalar, VLIW and EPIC suffer from low utilization of resources, implementation complexity, and immature compiler technology, [15] - [18]. When it comes to processor architectures, we can say that the integration of multiple cores or processors on a single chip die brings even greater demands to the memory system, increasing the number of slow off-chip memory accesses, [8].

In order to tolerate the memory latency and allow the processor to execute other tasks while a memory request is being served, a separate group of memory latency tolerance techniques was introduced. Some of the most popular methods in this group are multithreading, [2], instruction and data pre-fetching, [1] and non-blocking caches, [34]. In general, the

usage of these methods contributes to the "reduction" of the memory latency, but on the other hand it increases the memory traffic, leading to a higher instruction and data rate. As a result of the limited bandwidth on the memory interface, additional latency can be generated.

Besides the previously discussed memory latency reduction and tolerance methods, there are several proposals, which present some modifications into the classic multi-level memory hierarchy and provide nonstandard faster access to the main memory. For example, the author of [19] proposes a register-less processor that performs all the operations directly with the cache memory, organized in several layers (on-chip and off-chip), excluding the explicit use of GPRs. Additionally, the authors of [21] suggest the use of Scratchpad memory as a small high-speed on-chip memory that maps into the processors address space at a predefined memory address range. Opposite to the cache memory, the Scratchpad memory is allocated under software control and is characterized with deterministic behavior, allowing single-cycle access time. This small on-chip memory is mostly used for storing in-between results and frequently accessed data, so it requires developing of complex compiler methods for effective data allocation.

Contrary to the standard model of processor-centric computing (Von Neumann model), [9], some researchers have proposed alternative approaches of memory-centric computing, which suggests integrating or placing the memory near to the processor. These proposals are known as computational RAM, intelligent RAM, processing in memory chips, intelligent memory systems, [22] - [31], etc. These merged memory/logic chips implement on-chip memory which allows high internal bandwidth, low latency, and high power efficiency, eliminating the need for expensive, high-speed inter-chip interconnects, [35]. This makes them suitable to perform computations which require high data throughput and stride memory accesses, such as FFT, multimedia processing, network processing, etc., [28].

The integrated on-chip memory in the merged memory/logic chips is usually implemented as SRAM or embedded DRAM, which is mostly accessed through the processor's cache memory. Although the processing in/near memory brings latency and bandwidth improvement, still the system has to perform unnecessary copying and movement of data between the on-chip memory, caches, and GPRs. Besides that, the processing speed, the on-chip memory size, and the chip cost are limited due to the used implementation technology and the production process. Moreover, it is even a greater challenge to develop suitable compiler support for the system, which will recognize the program parallelism and will enable effective utilization of the internal memory bandwidth.

Having in mind that modern processors are lately dealing with both technical and physical limitations, while the memory capacity is constantly increasing, it seems that now is the right time to reinvestigate the idea of placing the processor in or near to the memory in order to overcome their speed difference, [36] - [38]. A promising approach that targets this problem is presented by the Hewlett Packard international information technology company that suggests novel computer architecture, called the Machine, [39], which utilizes non-volatile memory as a true DRAM replacement. A more detailed study about other proposals for overcoming the processor-memory bottleneck is presented in our previous research, given in [40].

Considering the adjustments of the standard memory hierarchy model, presented in some of the previously discussed approaches (ex. PERL, Scratchpad, Machine), we can say that the extension or revision of their work can be a good starting point for further research. In that process, we can first perceive that the relatively small number of fast GPRs in the highest level of the memory hierarchy is the major obstacle for achieving high data throughput. After that, we can consider that the cache memory is a limiting factor in real-time computing, and is also a redundant memory resource, which adds extra hardware complexity and power consumption into the system. Therefore, our research will continue into the direction of developing a novel RISC-based memory-centric processor similar to PERL, which will provide direct access to the memory that is integrated into the processor chip, without the use of GPRs and cache memory. The proposed replacement of the two highest memory hierarchy levels with an on-chip memory is intended to provide: exclusion of unnecessary data copying and individual or block data transfer into the GPRs and cache memory, a decrease of the capacity of redundant memory resources, simplification of the accesses to memory and removal of complex memory management mechanisms.

III. Design of RISC-based Memory-centric Processor Architecture

As a referencing point for designing the proposed RISC-based memory-centric processor, we make use of a RISC architecture implementation (MIPS), which is widely applied in the embedded industry and additionally is well documented and presented in the leading world's literature in the field of processor architectures. The selected MIPS implementation of a single-cycle pipelined RISC architecture, presented by D. A. Patterson and J. L. Hennessy in [1], is also used as a basis in the PERL processor architecture design. In general, MIPS processor is characterized with: fix-length instructions, simple addressing modes, memory accesses with explicit load and store instructions, hardwired control unit, large GPR set and pipeline operation in five stages (fetch, decode, execute, memory access and write back), as shown in Fig. 1.

According to Fig. 1, a MIPS processor includes: Program counter - PC, Instruction Register - IR, pipeline registers, 32 general-purpose registers, separated instruction and data cache memory, 32-bit arithmetical and logical unit, control unit (marked with blue), and other selecting and control logic (multiplexers, decoders, adders, extenders etc.). Therefore, MIPS operates only on operands found in its local GPRs, requiring frequent data transfer between the memory and the processor's registers, via load and store instructions. In order to provide easier access and manipulation of memory data, this paper proposes a modification and extension of the original pipelined RISC architecture and creation of a novel MEMRISC (Memory Access Reduced Instruction Set Computing) pipelined processor architecture, shown in Fig. 2.

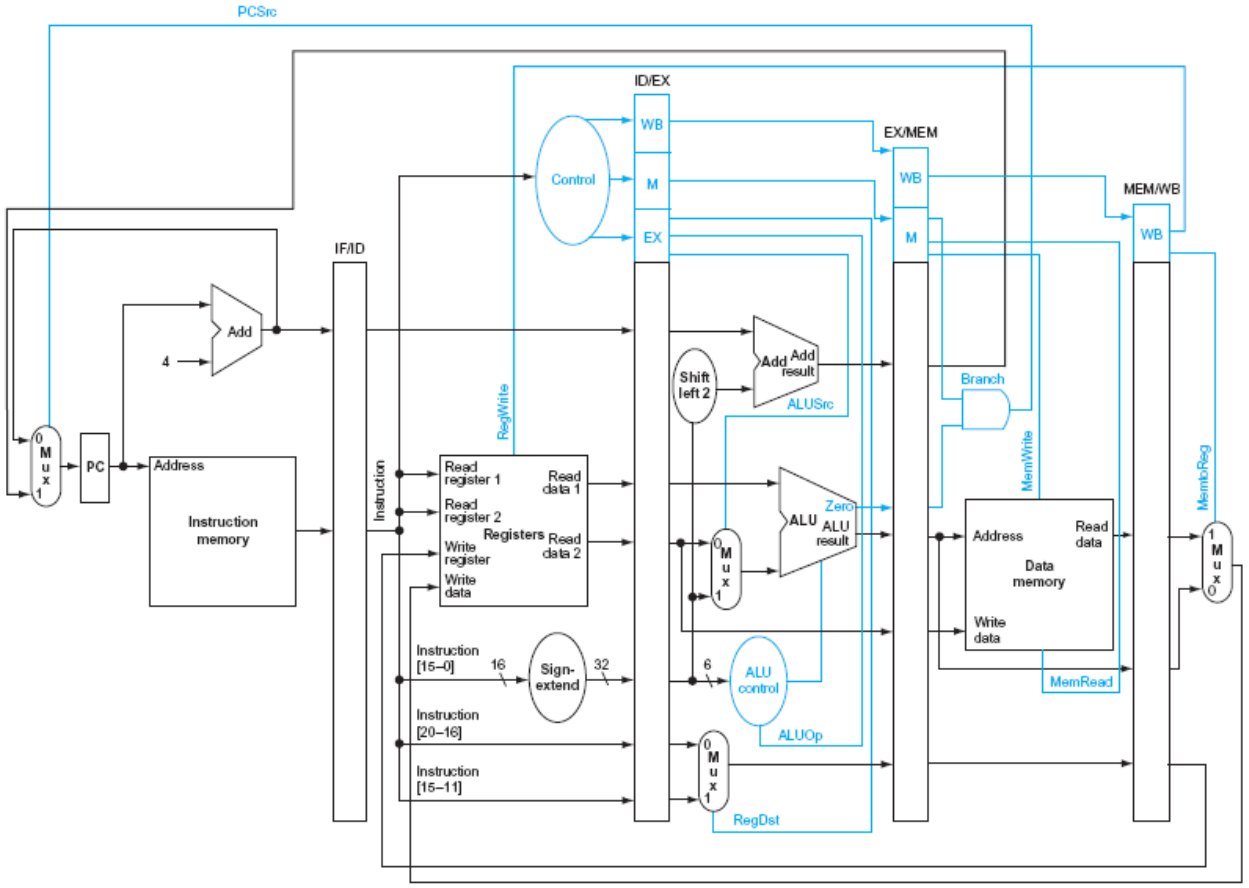


Fig. 1. Pipelined RISC-based MIPS Processor, [1].

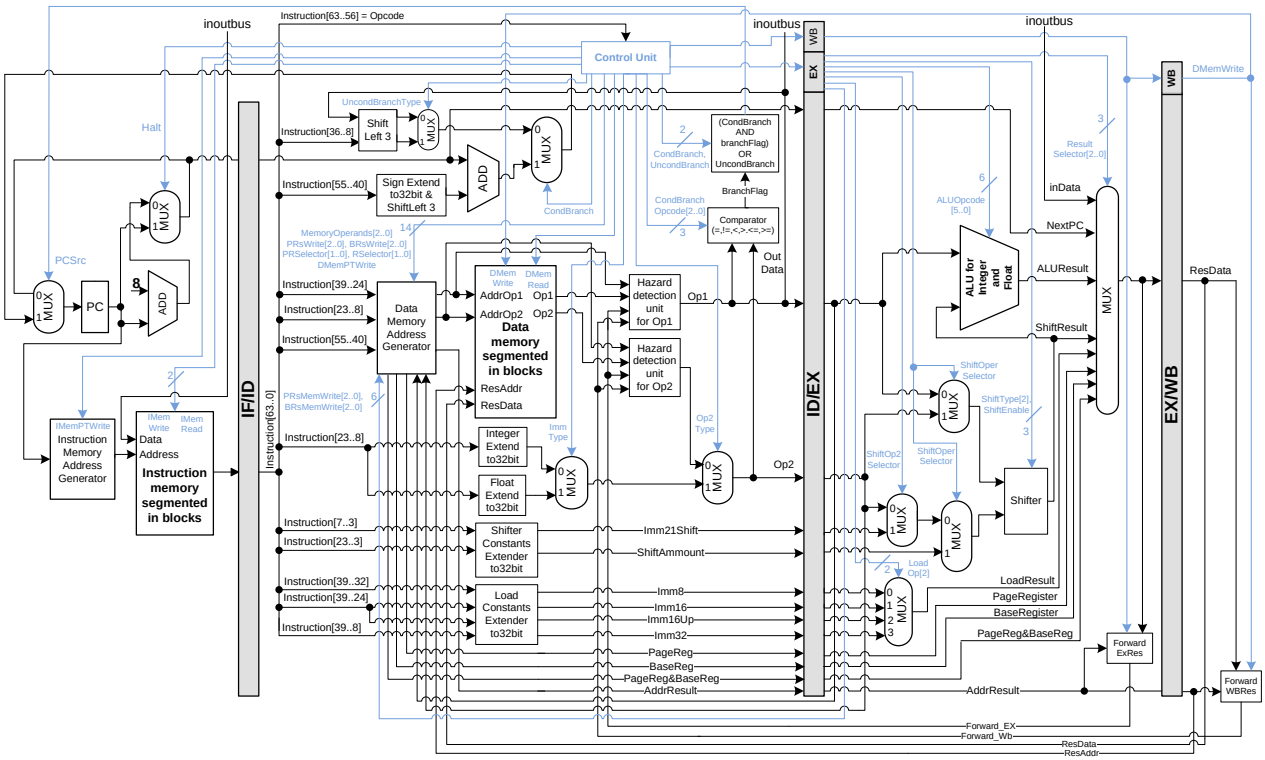


Fig. 2. Pipelined MIMOPS Processor with MEMRISC Architecture.

As shown in Fig. 2, the proposed processor with MEMRISC architecture uses separated on-chip data and program memory, instead of GPRs and on-chip cache memory. This means that the given processor executes all the operations on values found in its on-chip memory, avoiding the unnecessary and redundant data copying and movements, which are performed in the MIPS processor, during (load/store) data transfers. Therefore if the RISC-based MIPS processor is able to execute a million instructions per second, then the proposed processor with MEMRISC architecture would be able to execute a million instructions on memory operands per second, which is the reason why it is called MIMOPS processor in continuation.

The proposed MIMOPS processor excludes the GPRs and the cache memory from the memory hierarchy and thus allows direct and simultaneous access to two sources and one result operand, specified in the instruction. These operands are selected by a specialized memory address generator unit that is used to perform the translation of the input virtual addresses into physical addresses of the paged on-chip memory. Once the operands are read from the on-chip data memory, the operation is executed and then the result is written back to the on-chip data memory. In fact, the MIMOPS processor operates in a 4-stage pipeline (instruction fetch, instruction decode, execute and write back), excluding the MEM phase, and allowing every (arithmetical, logical, branch or control) MIMOPS instruction to be completed in a single tact cycle. The instructions that are supported by the proposed MIMOPS processor are MIPS alike, but the way of their interpretation and execution is slightly different.

Unlike the MIPS processor that is given in Fig. 1, the MIMOPS processor operates directly with the on-chip memory and thus simplifies the access to the operands, the execution of the instructions (pipelining without MEM phase) and the instruction set (removes explicit LOAD/STORE instructions). This way of operation of the MIMOPS processor is managed by a specialized control unit (marked with blue on Fig. 2), which provides support for several addressing modes (ex. direct, immediate, base, PC-direct, and PC-relative addressing). Generally, the memory operands are addressed directly, while the translation of the virtual addresses to physical addresses is performed via specialized hardware support for virtual memory that is implemented inside the MIMOPS processor. This refers to segmentation of the on-chip memory and filling it with virtual pages, and implementation of page translation tables and page replacement mechanisms (ex. FIFO).

The proposed MIMOPS processor implements separated on-chip instruction and data memories that are segmented into M equal-sized physical blocks (for virtual pages). Each of these local memories is organized as an array of N contiguous byte-sized elements, whereas each element has a unique physical address. To provide support for address translation and simultaneous access to the separated instruction and data on-chip memories, the proposed MIMOPS processor implements two dedicated hardware units, called instruction and data memory address generators. These units translate virtual addresses on the fly, performing a look-up in inverted page tables, [14], stored inside the processor's fetch and decode hardware logic, whose contents are managed by the operating system. According to the implemented approach of pipelining, MIMOPS can simultaneously access to a single instruction of an on-chip instruction memory block, and to three operands of up to three on-chip data memory blocks (some operands might be in the same block), as shown in Fig. 3.

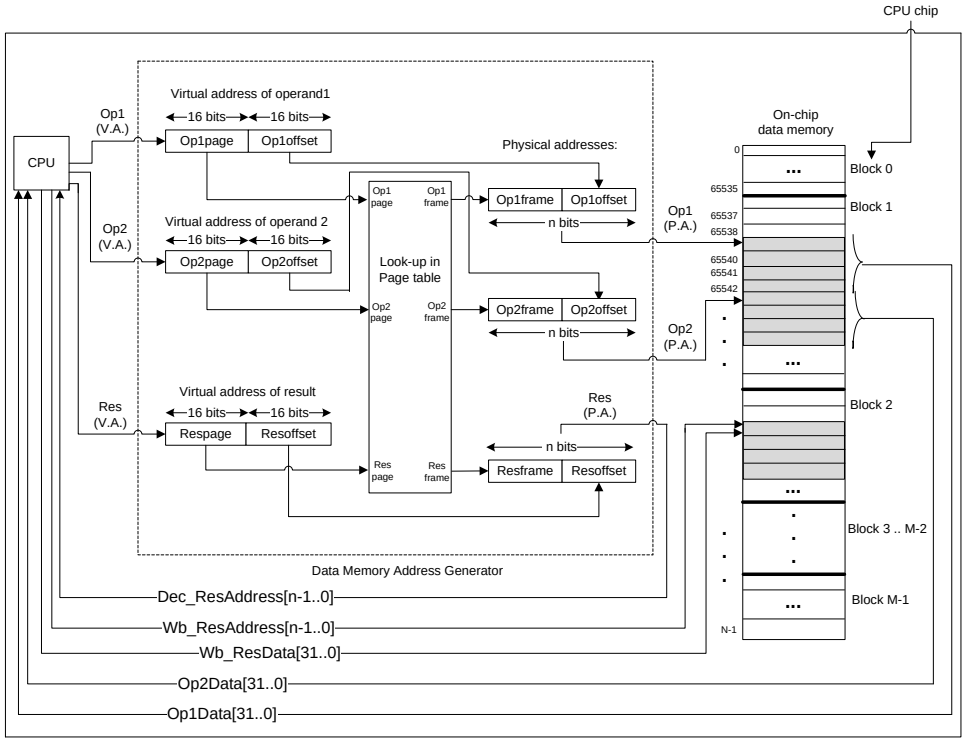


Fig. 3. Virtual Addressing of on-Chip Data Memory.

Fig. 3 shows how the CPU accesses to the on-chip data memory, during the instruction decode pipeline stage. Once the CPU decodes the instruction, it passes three virtual memory addresses (for operand1, operand2, and result) to the data memory address generator unit. This unit performs a look-up in a page table in order to find the appropriate frame numbers for the input page numbers, and thus to generate the effective physical addresses of the two input operands and the result operand. After that, the physical address of operand1 and operand2 are passed to the data memory over the memory bus, while the physical address of the result operand, (Dec_ResAddress), is sent to the next CPU pipeline stage, to be further used during the write-back stage.

According to Fig. 3, the CPU can simultaneously perform two reads and a single write to the on-chip data memory. This is achieved in such a way that the processor fetches two 4-byte (32-bit) data operands, starting at the generated physical addresses of operand1 and operand2, and in parallel stores the received 32-bit result data (Wb_ResData), starting at the physical address (Wb_ResAddress) of the result operand, which is actually passed from the write-back pipeline stage. Similarly to the result data and address forwarding (Wb_ResData, Wb_ResAddress), the fetched operands (operand1 and operand2) are sent to the next CPU pipeline stage, to be further used as input operands for computing some ALU operation in the execute stage.

When it comes to pipelines, it can be noticed that both MIPS and MIMOPS processors provide overlapping of the execution of the instructions, by implementing pipeline registers for every inter-phase (ex. instruction fetch/instruction decode). Besides these similarities, the MIMOPS processor differs from the MIPS processor in many ways, since it allows: reducing of the pipeline stages number by one, finishing the execution of conditional and unconditional branches in the decode pipeline stage and support of data forwarding for overcoming data hazards during parallel instructions execution. Additionally, the MIMOPS processor implements a separate shifter logic that is purposed to generate a second flexible operand for the arithmetical-logical unit (ALU). This is achieved by shifting the second operand by a specific constant value before it is being used by the ALU (this is similar to the ARM - Advanced RISC Machine architecture, [39]). Therefore, the ALU of the MIMOPS processor is able to perform operations over two integer or floating-point input numbers, where the second operand might be previously shifted.

Basically, the instruction set architecture of the proposed MIMOPS processor is RISC-like and includes three types of instructions (M-type, I-type and J-type), organized in four different groups. M-type instructions operate with memory operands placed in the on-chip data memory (similar to registers in R-type MIPS instructions), while I-type and J-type instructions operate with immediate values, whereas J-type instructions are used for unconditional branching. Depending on the function of the instructions, they can belong to arithmetical-logical, shifting, branching or control group. The arithmetical-logical group of instructions includes addition with overflow detection, subtraction, multiplication, integer division (div), modulo division (mod) and AND, OR, XOR and NOT logical bit-wise operations. The shifting group of instructions consists of left and right logical and arithmetical shifts and rotations. The branching group includes instructions for conditional and unconditional change of the program flow. The last group is the auxiliary group, consisting of instructions for program termination and system halt, SET instructions that update the base address units, load instructions for storing 8-, 16- or 32-bit immediate values in the on-chip data memory and IN/OUT instructions for communication with external devices.

The execution of MIMOPS instructions is managed by the control signals generated by the control unit that is specifically defined for the MEMRISC architecture. This unit provides support for several addressing modes, including base, direct, immediate, PC-direct and PC-relative. In addition to the control unit, the MIMOPS processor also includes: arithmetical - logical unit that can operate with integers and floating-point numbers, units for pipelining support, hazard detection unit for overcoming data hazards during pipeline execution of instructions, units that provide hardware support for virtual memory (memory segmentation in blocks, page tables etc), mechanisms for exception handling (ex. incorrect result), I/O (in-/output) control, and additional control and selection logic.

The proposed MIMOPS processor with MEMRISC architecture is expected to save many timing and hardware resources since it removes the complex cache memory management mechanisms and eliminates the use of explicit load and store instructions. Indeed, the MIMOPS processor excludes the many redundant copies of data that occur in GPRs and caches of processors which operate with standard memory hierarchy. This way of operation is very suitable for applications that perform many arithmetical-logical operations over some data set that is accessed with a high degree of locality. Examples of such type of applications are those that perform computations with matrices, such as matrix multiplication programs.

In order to present the performance gains (in terms of speed) of the novel MEMOPS processor with MEMRISC architecture, a comparative analysis between three similar processors is made. This refers to a MIMOPS processor, a register-less PERL processor, and a RISC-based MIPS processor. It is considered that the proposed MIMOPS processor includes on-chip memory with a capacity equal to the amount of cache memory into the MIPS and PERL processors (128KB L1 and 2M L2 cache). The actual analysis measures the execution time of a 32x32 matrix-multiplication program for each of the given processors. The program simulation is done with a MIMOPS instruction-set simulator, explained in [41], a MARS simulator for MIPS, [42] and a special instruction-set simulator for PERL, given in [19].

The results of the analysis are shown in Fig. 4 and Fig. 5, where Fig. 4 shows the execution time of the test program run on each of the three processors (PERL, MIPS, MIMOPS), while Fig. 5 illustrates the improvement that is achieved by MIMOPS. Referring to these results, it can be noticed that PERL provides an improvement of 8.82% in comparison to MIPS, but on the other hand the MIMOPS processor outperforms both of them, achieving 1.33 times (25%) better results than MIPS and 1.21 times (17.7%) better results than PERL. This analysis is made just to show and emphasize the performance potential of the proposed MIMOPS processor.

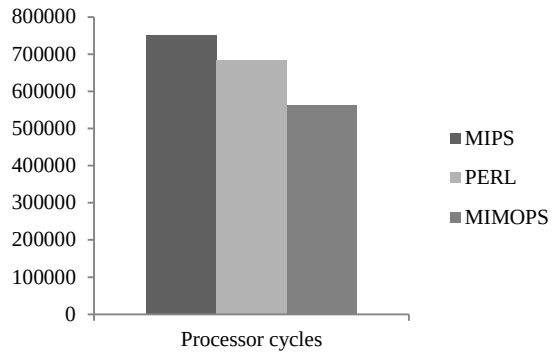


Fig. 4. Execution Time of 32x32 Matrices Multiplication on Three different Processors: MIPS, PERL and MIMOPS.

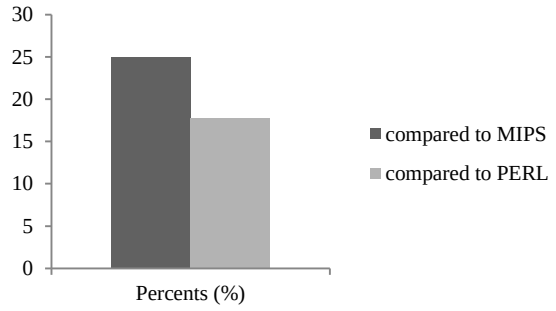


Fig. 5. Percentage Speedup of Execution Time of 32x32 Matrices Multiplication on MIMOPS Processor.

IV. FPGA Implementation of the Proposed RISC-based Memory-centric Processor Architecture

The proposed MIMOPS processor is described in VHDL, by means of Xilinx VIVADO Design Suite. This software environment enables hardware designers to synthesize (compile) their HDL codes, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure (program) a target FPGA device. In fact, all these functionalities are achieved by several different tools, including: Vivado regular synthesis and XST (High-Level Synthesis) compiler, Vivado implementation tool (translate, map, place, and route), Vivado Intellectual Property integrator, Vivado Simulator, Vivado serial I/O and logic analyzer for debugging, XDC (Xilinx Design Constraints) tool for timing constraints and entry, Vivado programming (Xilinx impact) tool etc. In general, Vivado is a design environment for FPGA products from Xilinx and is tightly-coupled to the architecture of such chips. Therefore, we use the Vivado tools suite in order to perform FPGA implementation of the proposed MIMOPS processor on Virtex7 VC709 Xilinx evaluation platform, [32].

The VHDL model of the proposed MIMOPS processor is organized in four modules (fetch, decode, execute, write-back) that form the processor's pipelined data-path and an additional module that provides communication with I/O devices. This is also presented in Fig. 6, where a block diagram (schematic) of the VHDL model of MIMOPS processor, generated in Vivado Design Suite, is given.

The fetch module is purposed to read an instruction from the on-chip instruction memory and to generate the next PC value that will be used for instruction fetch in the next tact cycle. This module includes three separate components: instruction memory, instruction page table and IF/ID pipeline register; that are accessed during the instruction fetching phase.

The decode module is purposed to decode the instruction that is sent from the fetch module and to read the instruction operands that are placed inside the on-chip data memory. Besides that, this module also executes shift and sign-extension operations for immediately-addressed operands, comparisons for conditional branching, data hazards detection, and produces control signals with the control unit. This module includes several separate components: data memory, data page table, ID/EX pipeline register, comparator, control unit, and a few multiplexers and extenders; that are accessed during the instruction decoding phase.

The execute module is purposed to execute shifting and arithmetical-logical operations and to select the result value (result from ALU, result from the shifter, etc.) that should be written back to the on-chip data memory. In addition to that, this module also performs forwarding of the result value and address to the decode module in order to prevent the occurrence of data hazards. This module includes several separate components: ALU for integer and real numbers, shifter, EX/WB pipeline register, result selector multiplexer, and several other multiplexers; that are accessed during the instruction executing phase.

The write-back module is purposed to write the result value to the on-chip data memory and to provide forwarding of the result to the decode module in order to prevent the occurrence of data hazards. This module acts as an interface to the decode module, which actually executes the operations of writing and resolving data conflicts.

The I/O communication module is purposed to transfer data between an I/O device and the MIMOPS processor (instruction or data on-chip memory) with IN or OUT instructions. Accordingly, this module uses an in/out data bus to receive data from an I/O device to its on-chip memory (when IN instruction is executed) or to send data to an I/O device from its on-chip memory (when OUT instruction is executed).

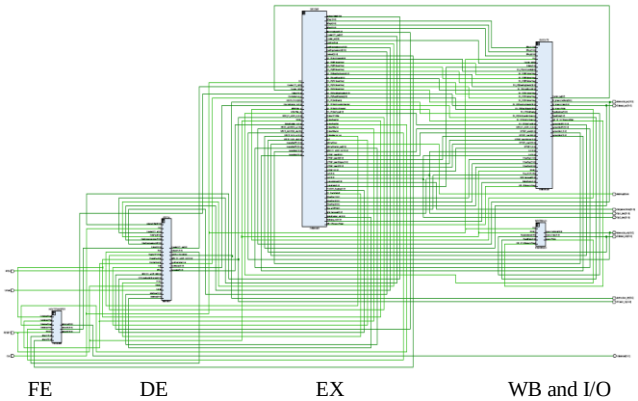


Fig. 6. Block diagram of the VHDL model of MIMOPS.

Each of the given VHDL modules is represented with a block diagram (schematic) that is generated by the Vivado Design Suite. In addition to that, the Vivado Simulator is used to verify the operation of these VHDL modules with separate test-bench programs, written for that purpose. Finally, the complete MIMOPS processor is simulated, and its overall functionality is verified. Therefore, a test-bench is written to analyze the processor's behavior during the execution of a test program that is placed in the processor's instruction memory, (given in Fig. 7(a)). Additionally, it is considered that the processor's data memory is already filled with data, as shown in Fig. 7(b). The results of the test-bench simulation are presented in Fig. 7(c).

```
signal InstructionMemory : rom_type := (
0 =>X"0000100000000100",--mem(16)=mem(0)==mem(1)      comparision
1 => X"0400110000000100", --mem(17)=mem(0)!=mem(1)    comparision
2 => X"0800120000000100", --mem(18)=mem(0)> mem(1)    comparision
3 => X"0C00130000000100", --mem(19)=mem(0)>=mem(1)    comparision
4 => X"1000140000000100", --mem(20)=mem(0)<mem(1)     comparision
5 =>X"1400150000000100", --mem(21)=mem(0)<=mem(1)     comparision
6 => X"1800160000000100", --mem(22)=mem(0)+ mem(1)    addition
7 => X"1C00170000000100", --mem(23)=mem(0)- mem(1)    substruction
8 => X"2000180000000100", --mem(24)=mem(0)/mem(1)     division
9 => X"2400190000000100", --mem(25)=mem(0) mod mem(1) mod div
10 => X"28001A0000000100", --mem(26)=mem(0)* mem(1)   multiply
11 => X"2C001B0000000100", --mem(27)=mem(0)and mem(1) AND
12 => X"30001C0000000100", --mem(28)=mem(0) or mem(1) OR
13 => X"34001D0000000100", --mem(29)=mem(0) xor mem(1) XOR
14 => X"38001E0000000100", --mem(30)=mem(0) xnor mem(1) XNOR
15 => X"3C001F0000000100", --mem(31)=not mem(1)       NOT
others => X"D000000000000000" --no operation          NOP );
```

a) State of MIMOPS instruction memory.

```
signal DataMemory : ram_type:= (
0 => X"00000001",
1 => X"00000002",
2 => X"00000000",
3 => X"00000010",
4 => X"00000003",
5 => X"00000004",
6 => X"3fc00000", --1.5 when used as float number
7 => X"3fc00000", --1.5 when used as float number
8 => X"00010000",
9 => X"00000014",
others => X"00000000");
```

b) State of MIMOPS data memory.

c) Results of MIMOPS VHDL Model Simulation.

Fig. 7. Simulation of VHDL model of MIMOPS Processor.

Once the VHDL model of the MIMOPS processor is simulated and verified, the next step is to perform synthesis and implementation of the particular processor in Vivado Design Suite. These activities are performed automatically with the synthesis and implementation tools, which are previously set to target the processor's FPGA realization on Virtex7 VC709 evaluation board, shown in Fig. 8. In general, the VC709 evaluation board provides a hardware environment for developing and evaluating designs targeting Virtex7 XC7VX690T-2FFG1761C FPGA, [32]. This board allows features common to many embedded processing systems, such as DDR3 memories, an 8-lane PCI Express interface, general-purpose I/O, and a UART interface. Other features can be added by using mezzanine cards attached to the VITA-57 FPGA mezzanine connector (FMC) provided on the board.

In the synthesis stage, the VHDL model of the MIMOPS processor is converted to a "netlist", which is composed of generic circuit components interconnected with connections. After the synthesis, the Vivado implementation tool is used to perform: translate, map, place, and route sub-steps. This way, the MIMOPS processor is translated and mapped to Xilinx Virtex7 XC7VX690T FPGA components and after that these components are physically placed and connected together (routed) on the appropriate FPGA board. Fig. 9 presents the state of the Virtex7 VC709 FPGA device after the synthesis and implementation of the MIMOPS processor.

Once the processor's implementation is finished, more detailed reports about the hardware characteristics of the designed MIMOPS processor are generated. According to the resource utilization report, shown in Fig. 10 it can be noticed that the proposed MIMOPS processor can be implemented in Virtex7 VC709 evaluation platform, by utilizing less than 1% of the slice registers and 36% of the slice LUT resources. This result is expected since the MIMOPS processor integrates the memory inside the chip and it implements complex mechanisms that provide hardware support for virtual memory (includes memory address generators with on-chip page tables and performs management of memory blocks etc). In addition to that, the MIMOPS processor includes a more complex control unit that provides support for direct access to memory operands. Besides the control unit, additional complexity is introduced with the implementation of data hazard detection unit, comparison logic purposed for conditional branching in the decode phase, ALU unit that is extended to operate with floating-point numbers and shifter unit that provides support for second source flexible operand. All these hardware units are implemented with the aim to improve the computing performances of the MIMOPS processor, which is actually achieved, but the chip complexity is increased.

In order to program the Virtex7 VC709 FPGA, a constraint file has to be prepared. This file is used to assign the VHDL code signals of the MIMOPS processor to the device pins found on the Virtex7 VC709 evaluation board. For example, the reset signal is assigned to the on-board CPU reset push button switch, which allows the user to manually reset the processor. Similarly, the CLK signal is assigned to the 200 MHz system clock of the FPGA board that is active on a positive edge. In addition to that, the last 8 bits of the ResultData signal that is forwarded from executing to the write-back stage are assigned to the 8 user LEDs of the FPGA board. More details about the Virtex7 VC709 board I/O pin assignments are given in Table 1.

After the FPGA programming, the user can analyze the execution of some program that is already loaded inside the processor's on-chip instruction memory, just by observing the changes of the LEDs state. It is considered that the given program operates with numbers that are in the range of [0-255]. Considering that a MIMOPS processor that works with 200 MHz system clock executes the program very fast, an additional component is defined in order to scale the input 200 MHz clock signal to 1 Hz clock signal (with a period of 1 s). This way, the state of the LEDs changes slowly, so the user can easily monitor the test program's execution.

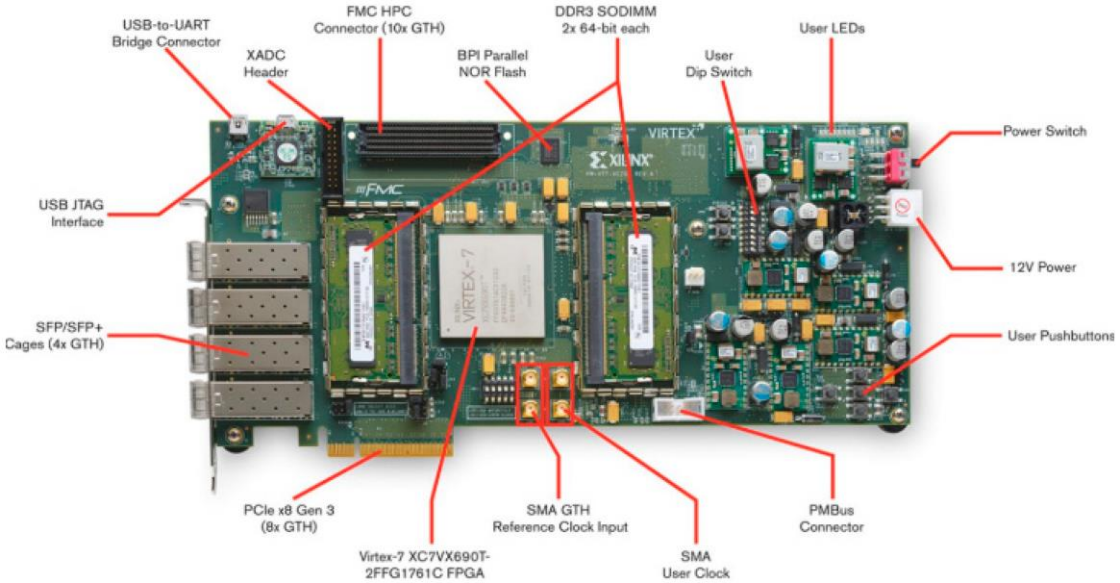
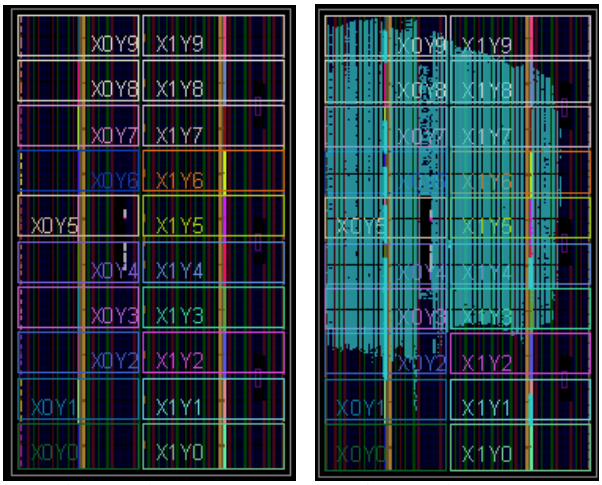


Fig. 8. Virtex7 VC709 evaluation board, [32].



a) State of FPGA after synthsis. b) State of FPGA after implementation.

Fig. 9. FPGA Implementation of MIMOPS Processor on Virtex7 VC709 Evaluation Board.

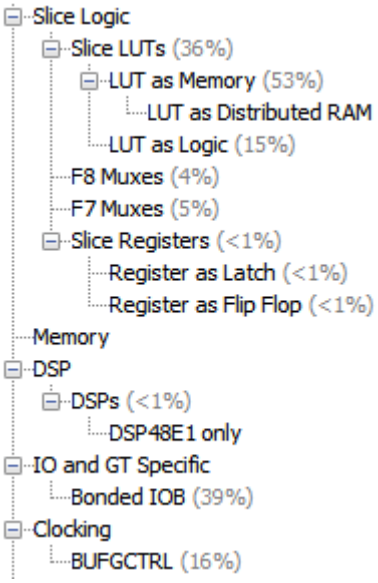
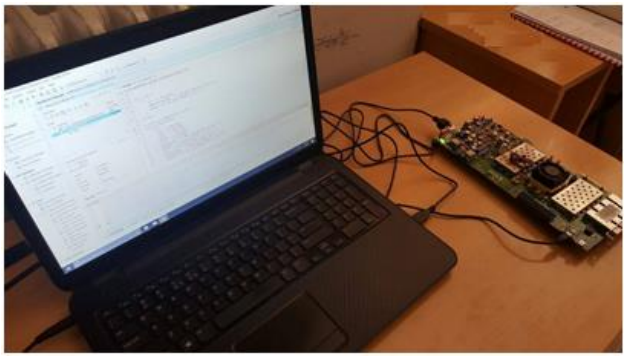


Fig. 10. FPGA utilization(%) of MIMOPS Processor.

TABLE. I. VIRTEX7 VC709 BOARD I/O PINS ASSIGNMENT TO MIMOPS PROCESSOR'S SIGNALS

Signal	*Pin*	*Pin Type*	*Pin Function*
CLK	H19	IN	200 MHz system clock active on positive edge
Reset	AV40	IN	Reset push button (PB) switch
ResultData[7]	AU39	OUT	User LED 7
ResultData[6]	AP42	OUT	User LED 6
ResultData[5]	AP41	OUT	User LED 5
ResultData[4]	AR35	OUT	User LED 4
ResultData[3]	AT37	OUT	User LED 3
ResultData[2]	AR37	OUT	User LED 2
ResultData[1]	AN39	OUT	User LED 1
ResultData[0]	AM39	OUT	User LED 0



a) Programming of Virtex7 VC709 FPGA with Xilinx Impact Tool.

b) Simulation of MIMOPS Processor in Real Hardware.

Fig. 11. Hardware Prototype of MIMOPS Processor in Virtex7 VC709 FPGA Board.

Finally, the processor is completely ready to program onto the FPGA device, by means of the Xilinx impact tool. In that process, a bit stream file is generated and used to program the target FPGA device by JTAG cable. The created prototype of the proposed MIMOPS processor in real hardware i.e. Virtex7 VC709 XC7VX690T FPGA board is shown in Fig. 11. After that the test program that is shown in Fig. 7.a is simulated and executed in real FPGA, and the results are verified, according to the FPGA LEDs state and Fig. 7.c simulation diagrams.

V. CONCLUSION

This paper proposes a memory-centric processor core that is based on a standard MIPS implementation of RISC architecture, which is further improved to operate with separated on-chip data and program memory, by excluding the use of GPRs and cache (in and out of the processor chip). The memory-centric approach of processing provides 4-stage pipelining with direct access to the on-chip memory (without MEM phase), fast and simple access to the on-chip memory (without explicit load/store instructions), avoidance of copy operations and transfers of redundant data and blocks into GPRs and cache memory, decrease of capacity of redundant on-chip memory resources, high internal memory bandwidth, and removal of complex cache memory management mechanisms. Actually, it is shown that a MIMOPS processor achieves 25/17.7% better results than a MIPS/PERL processor when executing a 32x32 matrix-multiplication program.

The main focus of this paper is the FPGA implementation of the proposed MIMOPS processor. This includes designing of VHDL hardware model of the proposed processor and experimenting with Xilinx VIVADO Design Suite software environment, which provides support for Virtex7 VC709 FPGA evaluation board. In that process, the hardware model of

the proposed RISC-based memory-centric processor is first simulated, by means of Xilinx VIVADO Design Suite Simulator tool. The simulation is performed with test bench programs that generate timing diagrams, which are further used for analyzing the behavior of the hardware model of the proposed processor and its components. The VIVADO synthesis and implementation tools are next employed in creating an RTL model of the proposed processor and implementing the synthesized processor in Virtex7 VC709 FPGA board. The reports that are generated from these tools present that the MIMOPS processor utilizes less than 1% of the slice registers and 36% of the slice LUT resources. The I/O mapping of the MIMOPS processor interfaces with the Virtex7 VC709 FPGA board pins and the programming of the given FPGA Virtex7 VC709 FPGA board are performed at the final stage. The created hardware prototype is used for simulating and analyzing of the proposed MIMOPS processor in real hardware, by means of Virtex7 VC709 FPGA component. This approach makes use of FPGA re-programmability, which has proven to be an ideal solution for achieving reasonable speed at a low price.

The proposed MIMOPS processor provides many advantages, especially in terms of processing speed, but on the other hand it imposes additional requirements to the system's hardware and software, which cause limitations in its application area. Accordingly, the proposed MIMOPS processor implements specific ISA and several special-purpose hardware components that provide direct operation with the on-chip memory. Therefore, it is obvious that a specific software support for the proposed MIMOPS processor should be developed in the future. The primer requirement would be designing of a dedicated compiler that would be able to translate high-level language programs to MIMOPS assembler, (that significantly differs from the assembler of other RISC-based processors) while keeping the standard programming model. Afterward, the next research activities would include developing of a dedicated operating system with process scheduler, which would be able to manage the MIMOPS on-chip memory and to coordinate the complete virtual address space, while multiple processes are being executed. Furthermore, assuming the recent innovation in processing in memory architecture and technology it may become desirable to build a scalable multi-processor MIMOPS-based system in very near future.

REFERENCES

[1] D. A. Patterson, J. L. Hennessy, Computer Organization and Design: The hardware/software Interface, 5th ed., Elsevier, 2014.

[2] J. L. Hennessy, D. A. Patterson, Computer Architecture: A Quantitative Approach, 5th ed., Morgan Kaufmann Publishers, 2011.

[3] "Moore's law is dead - long live Moore's law," in IEEE Spectrum Magazine, April 2015.

[4] J. Hruska, "Forget Moore's law: hot and slow DRAM is a major roadblock to exascale and beyond," in Extreme Tech Magazine, 2014.

[5] W. A. Wulf, S. A. McKee, "Hitting the memory wall: implications of the obvious," in ACM SIGARCH Computer Architecture News, Vol. 23, Issue 1, March 1995.

[6] Y. Yan, R. Brightwell, X. Sun, "Principles of memory-centric programming for high performance computing," in Proc. of Workshop on Memory Centric Programming for HPC, USA, 2017.

[7] D. Patterson, "Latency lags bandwidth," in Communications of the ACM, Vol. 47, No. 10, 2004, pp 71-75.

[8] D. Jakimovska, A. Tentov, G. Jakimovski, S. Gjorgjievska, M.Malenko, "Modern processor architectures overview," in Proc. of XVIII International Scientific Conference on Information, Communication and Energy Systems and Technologies, Bulgaria, 2012, pp. 239-242.

[9] R. Eigenmann, D. J. Lilja, "Von Neumann computers," in Wiley Encyclopedia of Electrical and Electronics Engineering, Volume 23, 1998, pp. 387-400.

[10] A. Bakshi, J. Gaudiot, W. Lin, M. Makhija, V. K. Prasanna, W. Ro, C. Shin, "Memory latency: to tolerate or to reduce?," in Proc. of 12th Symposium on Computer Architecture and High Performance Computing, 2000.

[11] S. Borkar, A. A. Chien, "The future of microprocessors," in Communications of the ACM, Vol. 54 No. 5, May 2011, pp 67-77.

[12] Intel Corporation, "New microarchitecture for 4th gen. Intel core processor platforms," Product Brief, 2013.

[13] W. Bao, S. Tavarageri, F. Ozguner, P. Sadayappan, "PWCET: power-aware worst case execution time analysis," in Proc. of 43rd International Conference on Parallel Processing Workshops, 2014.

[14] P. Machanick, "Approaches to addressing the memory wall," Technical Report, School of IT and Electrical Engineering, University of Queensland Brisbane, Australia, 2002.

[15] C. Kozyrakis, D. Patterson, "Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks," in Proc. of the 35th International Symposium on Microarchitecture, Instabul, Turkey, November 2002.

[16] J. Silc, B. Robic, T. Ungerer, Processor architecture: From Dataflow to Superscalar and Beyond, Springer, 1999.

[17] N. FitzRoy-Dale, "The VLIW and EPIC processor architectures," Master Thesis, New South Wales University, July 2005.

[18] M. Smotherman, "Understanding EPIC architectures and implementations," in Proc. of ACM Southeast Conference, 2002.

[19] P. Suresh, "PERL - a register-less processor," PhD Thesis, Department of Computer Science & Engineering, Indian Institute of Technology, Kanpur, 2004.

[20] P. R. Panda, N. D. Dutt, A. Nicolu, "On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems," ACM Transactions on Design Automation of Electronic Systems, 2000.

[21] V. Venkataramani, M. Choon Chan, T. Mitra, "Scratchpad-memory management for multi-threaded applications on many-core architectures," ACM Transactions on Embedded Computing Systems, Vol. 18, Issue 1, 2019.

[22] C. Cojocaru, "Computational RAM: implementation and bit-parallel architecture," Master Thesis, Carletorn University, Ottawa, 1995.

[23] H. Tsubota, T. Kobayashi, "The M32R/D, a 32b RISC microprocessor with 16Mb embedded DRAM," Technical Report, 1996.

[24] J. Draper, J. T. Barrett, J. Sondeen, S. Mediratta, C. W. Kang, I. Kim, G. Daglikoca, "A prototype processing-in-memory (PIM) chip for the data-intensive architecture (DIVA) system," Journal of VLSI Signal Processing Systems, Vol. 40, Issue 1, 2005, pp. 73-84.

[25] M. Gokhale, B. Holmes, K. Jobst, "Processing in memory: the Terasys massively parallel PIM array," IEEE Computer Journal, 1995.

[26] K. Keeton, R. Arpaci-Dusseau, and D.A. Patterson, "IRAM and SmartSIMM: overcoming the I/O bus bottleneck", in Proc. of the 24th Annual International Symposium on Computer Architecture, June 1997.

[27] C. E. Kozyrakis, S. Perissakis, D. Patterson, T. Andreson, K. Asanovic, N. Cardwell, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, R. Thomas, N. Treuhaft, K. Yelick, "Scalable processors in the billion-transistor era: IRAM," IEEE Computer Journal, Vol. 30, Issue 9, pp 75-78, 1997.

[28] J. Gebis, S. Williams, D. Patterson, C. Kozyrakis, "VIRAM1: a media-oriented vector processor with embedded DRAM," 41st Design Automation Student Design Contest, San Diego, CA, 2004.

[29] K. Murakami, S. Shirakawa, H. Miyajima, "Parallel processing RAM chip with 256 Mb DRAM and quad processors," in Proc. of Solid-State Circuits Conference, 1997.

[30] S. Kaxiras, D. Burger, J. R. Goodman, "DataScalar: a memory-centric approach to computing," Journal of Systems Architecture, 1999.

[31] M. Oskin, F. T Chong, T. Sherwood, "Active pages a computation model for intelligent memory," in Proc. of the 25th Annual International Symposium on Computer architecture, 1998, pp. 192-203.

[32] Xilinx, "VC709 evaluation board for the Virtex-7 FPGA," User Guide, 2019.

[33] J. M. P. Cardoso, M. Hubner, Reconfigurable Computing: From FPGAs to Hardware/Software Codesign, Springer-Verlag New York, 2011.

[34] S. Li, K. Chen, J. B. Brockman, N. P. Joupp, "Performance impacts of non-blocking caches in out-of-order processors," Technical Paper, 2011.

[35] S. Ghose, K. Hsieh, A. Boroumand, R. Ausavarungnirun, O. Mutlu, "The processing-in-memory paradigm: mechanisms to enable adoption," in book: Beyond-CMOS Technologies for Next Generation Computer Design, 2019.

[36] G. Singh, L. Chelini, S. Corda, A. Javed Awan, S. Stuijk, R. Jordans, H. Corporaal, A. Boonstra, "A review of near-memory computing architectures," in Proc. of the 21st Euromicro Conference on Digital System Design, 2018.

[37] E. Azarkhish, D. Rossi, I. Loi, L. Benini, "Design and evaluation of a processing-in-memory architecture for the smart memory cube," in Proc. of the 29th International Conference Architecture of Computing Systems, Germany, 2016.

[38] E. Vermij, L. Fiorin, R. Jongerius, C. Hagleitner, J. Van Lunteren, K. Bertels, "An architecture for integrated near-data processors," ACM Transactions on Architecture and Code Optimization, Vol. 14, Issue 3, 2017.

[39] Hewlett Packard Labs, "The machine: the future of technology," Technical Paper, 2016.

[40] D. Efnusheva, A. Cholakoska, A. Tentov, "A survey of different approaches for overcoming the processor-memory bottleneck," International Journal of Computer Science & Information Technology, Vol. 9, No. 2, April 2017.

[41] G. Dokoski, D. Efnusheva, A. Tentov, M. Kalendar, "Software for explicitly parallel memory-centric processor architecture," in Proc. of Third International Conference on Applied Innovations in IT, 2015.

[42] K. Vollmar, P. Sanderson, "MARS: an education-oriented MIPS assembly language simulator," in Proc. of the 37th SIGCSE Tech. Symposium on Computer Science Education, 2007.