FPGA based MIPS Pipeline Processor with SIMD Architecture

Sarah M. Al-Sudany[1], Ahmed S. Al-Araji[2], Bassam M. Saeed[3]

Computer Engineering Department, University of Technology, Baghdad, Iraq

Abstract: ***The aim of this study is to develop a MIPS pipeline processor based on FPGA with using VHDL. This architecture can be used for academic purposes and to set up a multifunctional system for the processing of digital signals or images. In order to do so, a subset of MIPS instructions is chosen to show functionality in the simulation and synthesis processor inside a five-stage pipeline (instruction fetch, instruction decode, execution, memory and writing back). The hazard control network has been set up to manage data transfer and stalling. A single cycle multiplication functionality and a single-cycle SIMD instruction have been added to the basic MIPS architecture. The SIMD instructions have been selected to execute binary operations for possible mathematical morphology. A Software (XUP) box containing Xilinx Virtex7 xc7vx330tFPGA was the board used to test the processor. This makes it possible for the processor to perform four 32-bit data sets per cycle when the SIMD pipeline is complete. The research explained the Field Programmable Gate Array (FPGA) technology with extensive explore of MIPS pipeline architecture. In addition to the research highlights many aspect of this process such as the role of The SIMD principle in the support of the hardware acceleration feature that required by multicore overall processors and instruction set and the multiple instructions executed in this processor from each register and multiply them as standard MIPS applications instead of the entire 32 bits.***

Keywords: MIPS Pipeline Processor, SIMD Architecture, FPGA, RTL, VHDL

1. Introduction

Processor is one of the basic component of any digital system. It is also called the brain of a computer or a system. Moreover, processor is the electronic circuitry within a computer that executes instructions that make up a computer program .Design computing processors based on FPGA and computational SIMD arrays that has already been proven to perform supercomputer class tasks for limited amounts of supercomputer costs. Such two architectures consist of a number of limited, but various processing components. Such similarity is the secret to understanding the remarkably high performance of silicon on the FPGA and SIMD chips. Huge data parallelism makes high use of SIMD computers[1].

Through specific task computer configuration and pipelining, FPGA machines achieve a higherutilization. In either case, numerous thousand bits per cycle transformed compared with a standard 64 bits microprocessor. Today, field-programmable gate arrays (FPGAs) commonly used for the implementation of multipurpose logic. FPGAs built from a sophisticated collection of basic logical functions [2]. There are a number of FPGAs available on the market from many vendors. Further developments in packaging allow high-performance [3]. Integrated SIMD arrays to be manufactured at reasonable cost. Such two array structures are identical. Both use a sophisticated collection of logical components. The logical unit performs a basic logical function in certain states and some array inputs, and either updates its own status to record the computation results or shares the results with other entities in the array [4].Given the similarities, there have been very different directions in the design of FPGAs and SIMD arrays.

In this paper, it implements MIPS processor model on the FPGAs and use SIMD architecture.The manner in which FPGA and SIMD architecture solve problems is contrasted in Section 2, including FPGA technology. In section 3, introduces and defines the specific terms such as MIPS and SIMD. In section 4, explains MIPS pipeline architecture and instruction set architecture. The results are discussed in section 5; finally, conclusions are cited in section 6.

2. Field Programmable Gate Array (FPGA) Technology

Advances continually recorded in Field Programmable Gate Array (FPGA) technology. FPGAs are a preferred integration platform in many industries by high speed and versatility, the ability to take advantage of the inherent fundamental parallelism of many frameworks and algorithms, a fast time to market, strong cost-effectiveness, vast quantities of embedded resources and the availability of IP-specific cores [3 and 5]. Limited knowledge of the design and technology techniques, lack of adequacy of these techniques, size, and the absence of advanced hardware functionality conditioned the industrial penetration of this second program [6]. The difficulty of the FPGA architecture and the need to develop some hardware design expertise is one of the key obstacles to further acceptance of programmable computation as a new paradigm [7].

With FPGAs evolving by scaling down production fabrication technology, vendors have started developing soft processor cores, whichcan be incorporated from standard FPGA resources, and have embedded (hard) processors incorporated in their products [8]. This pattern has seen an immense growth, to the point that there are countless current solutions.This has led to a paradigm change in the past dichotomy of design strategies that constitutes the key existing asset of FPGAs thatcan no longer be seen merely as hardware accelerators, but also as very efficient system-on-Chip (SoC) platforms [9]. Combining embedded (or soft) processors with high performance custom-optimized hardware peripherals in a single chip open the door to unrestricted use of FPGA in any digital design field for commercial applications [10]. Perhaps unexpectedly, in recent years the features, design techniques, methods, and implementation areas of these devices were studied extensively, to mention only the work focused primarily on industrial systems. Additional features are constantly emerging. The authors therefore believe that a study of the latest advances in FPGA technology would be useful for the research community in industry informatics[11].FPGAs have very different hardware capabilities between manufacturers and models. The most beneficial options for designers have been studied in: standardized functional frames, i/ O signal processing, partial modification, IP security and special [12]. New manufacturing technologies (14 nm, 3-D tri-gate transistors)permit the incorporation of even more useful features into contemporary FPGAs, with more than 50 millions equivalent logical gates in some cases and operating frequencies of over500MHz while maintaining a rational power consumption[13].Indeed, while one of the conventional FPGAs problems is its high power requirement compared to microcontrollers, it is possible that the energy usage of FPGAs could be equivalent to that of the microcontrollers required in complex applications where multiple machines need to be used. Within these features, those, which enable the implementation of modern, advanced digital systems for commercial applications (often based on complex algorithms) due to their high computer power and/or the low run-time they deliver [14].

3. Literature Review

Market products such as handheld devices and home entertainment will deliver high-quality audio, video, picture and graphics output to connect with the new generation. The technology that provides higher data flow is especially essential for business applications of scientific / high-performance computing and data mining [15].SIMD (Single Instruction Multiple Data) is the primary technologies for connect with the new generation. Modern CPU principles that increase efficiency by enabling effective concurrent processing of vehicle activities. These advanced computing specifications streamlined and accelerated [16].

The MIPS SIMD Architecture (MSA) technology integrates a programmable approach into the CPU for managing evolving codecs or a limited number of features, which are not supported by specialised hardware in consumer electronics, whereas unusable hardware assists the Processor, and GPU with multimedia codecs.This programmable approach facilitates consistency of the system.

The MSA is also intended to speed up a variety of computing applications by providing standardised compiler assistance. MSA innovations is introduced in full accordance with the architecture standards of RISC (Reduced Instruction Set Computer) [17]. With simple directions, MIPS Architecture have developed the MSA to contribute to less complicated applications. The carefully chosen, basic SIMD instruction set is not only programmer friendly and easy to compile, but also quick, range and power sensitive. The requirements for MSA technologies are extendable and capable of meeting potential demands. In the following section we will clarify more detail about SIMD AND MIPS [18].

3.1 The Role of the SIMD Principle in New Technology

The SIMD principle is a method to boost efficiency in applications where highly repetitive operations are needed. SIMD is a method to execute the same procedure on multiple pieces of data concurrently, whether arithmetic or otherwise. When an algorithm is typically coded and a single process may be done over a wide dataset, a loop is used to iterate any entity in the dataset and execute the efficient approach. At each execution, a single piece of data is done for a single process [19].This is called Single Instruction Single Data processing (SISD). Loops may well iterate thousands of times and quite inefficient. Theoretically, the amount of iterations of a loop requires to be decreased to improve efficiency. One way to decrease loops is called loop unrolling. This takes the single procedure in the process and executes it numerous times in increasing iteration[20].

The SIMD principle goes a step forward by integrating and executing several acts in each loop repetition. Through SIMD, it is not only feasible to minimise the amount of loop iterations, but it can also minimise the necessary multiple operations to an efficient single action. This will be achieved by SIMD utilising vectors. The argument for a particular instruction may be used as a SIMD vectors that are then executed concurrently on all items in the vector [21].

The amount of values, whichcan be installed into the vector, thusaffects output directly; the more values are processed simultaneously, the more easily a full dataset can be processed. Two items are needed for this scale; the data type used and the SIMD specification. When values are processed and operated on in SIMD vectors through a SIMD method, they are simply transferred to a different collection of CPU registers whereby parallel processing happens. The SIMD design specifies the size and amount of such registers [22].

SIMD utilises several Kernel functional units; separate CPU functional units for continuously performing arithmetic and Boolean Procedures. The SIMD functionality can be enhanced by pipelining the guidance of the system.The instruction pipelines are to decompose instruction execution in a linear sequence of autonomous phases, which permit each step to concurrently execute an execution process component such as decoding, productive address measurement, fetch operand, execution and storage [23].The diagram of the SIMD architecturecan be shown in **Figure (1)**.

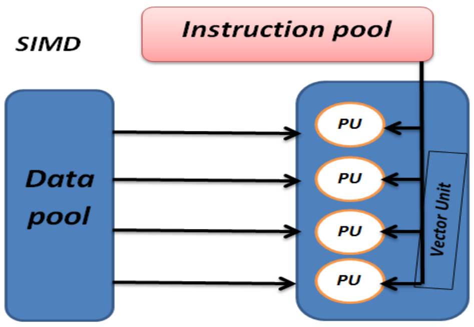


Figure 1: Big picture of SIMD architecture.

3.2 The Importance of MIPS in the Design of FPGA

John L. Hennessy first engineered the MIPS system in 1985, while work began at Stanford University in 1981. The MIPS processor architecture intended by using deep instruction pipelines to enhance processor efficiency. Instead of the full instruction cycle like the conventional designs, the clock rate of the processor centred on the crucial path in one of the stages before switching to the next. One significant feature of MIPS architecture was to demand that all instructions followed by just one cycle [24].

Thus avoiding any interlocking specifications. The architecture of the MIPS processor omitted a range of valuable instructions to complete certain moves [25]. The MIPS basic pipeline is shown in the **Figure (2).**

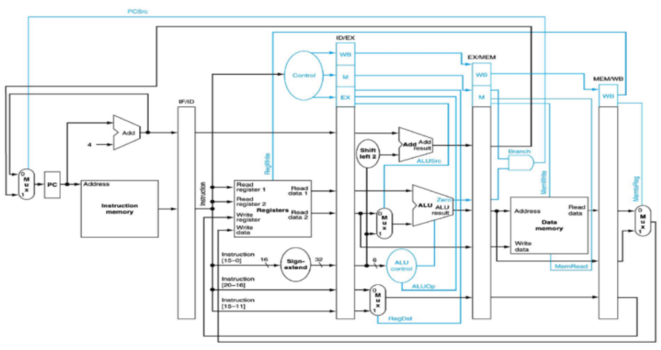


Figure 2: MIPS basic pipeline [25]

The lengthy instructions were taken out, as a consequence of the processor operating at much higher clock speeds, the machine output was considered to be significantly enhanced. It was difficult to increase the pace with interlocking, because the locks took up additional chip areas, which decreased the speed [26].

Several analysts believed that the MIPS concept should not be used with the absence of guidance in commercial products. The point was that of CISC (complex instructions computer set) vs RISC (reduced instructions set): that a complicated instruction would reduce speed by substituting several simpler instructions [27]. The claim overlooked the reality that the construction speed arises from the pipelines and not from the instructions themselves. Across several academic curricula, the principle of the MIPS system used to explain the pipeline [28].

The MIPS-based DLX system uses VHDL code to simulation the processor, but without significant alteration, it can not be synthesised [29]. The fundamental MIPS architecture has built a five-stage pipeline that separates the data path combination into stages. By splitting the processor into smaller parts, the overall frequency increased and the vital path from the single cycle route to a smaller portion of this route decreases [30].

4. MIPS Pipeline Architecture and Instruction Set Architecture

One of the most important functions of subset while the MIPS processor uses instructions set. Every instruction has a length of 32 bits and is characterised by six least significant or most significant bits. Three types of instructions are used with the MIPS: access to register (R-type), instant registration, immediate type (I-type) and jumping (J-type). The instructions R-type are mainly used for directions of method of function, i.e. addition and subtraction. The (I-type) instructions like the R type instructions were used, except in their instructions they use an immediate 16-bit meaning. Instructions for jump are used for the (J-type). In the following **Table1**displays the 28 instructions used:

Table 1. The MIPS Instruction Set

Instruction Name	Description
Addiu	unsigned registry addition and immediate value
And	AND logical, of two register
Or	OR logical of two registers
Xor	the xor logical of two registers
Sll	the shift left logically
Sra	the shift right arithmetic
Slti	set on less than immediately
Lu	Instant upper load
Break	exception to breakpoint
Jal	jump and link
Bltz	branch is less than zero
Sw	store the word
Srl	shift the right logic
Subu	unsigned subtraction of two registers
Addu	the unsigned addition of two registers
Andi	the logical and the register and the immediate value
Ori	the OR logical the register and the immediate value
Xori	the XOR logical of the register and the immediate value
Sltiu	set less than the unsigned immediate
Beq	the Branch equal
Jr	jump the registry
Bgez	branch greater than or equal to zero
Sltu	set on less than unsigned
Bne	branch not the equal
Lw	the word load
Mul	a signed multiplication of two registers
Slt	set on less than that
J	Jump

The multiple instructions executed in this processor would take 16 least bits from each register and multiply them as standard MIPS applications instead of the entire 32 bits. The multiplier instruction requires just 16 bit instead of 32 bit as the processor uses a Xilinx IP core for the multiplier, not allowing 32 bit processes. The understanding and analysis purposes the most important 28 instructions are mentioned here.

The addition, subtracting, and multiplying operations cover the arithmetic process; the logical procedure, sll, srl and sra cover changing needs; the configuration of less than and division instructions requires comparisons; lw and sw are memory-capable; Division and jump instructions contain circuits and division jumps; and the software debugging and checking split orders are feasible.

4.1 Elements of Pipeline Data Analysis

A pipeline is a collection of data processing devices linked in sequence, such that the outputs of one component is the input of the next component. These components or phases of a pipeline implemented in parallel time-sliced format with a certain volume of buffer capacity added between phases. Steps in this pipeline are instruction fetch, decode, memory, and write back. Such pipeline phases have also introduced in the basic MIPS processor.

4.1.1 Instruction Fetch (IF)

The Instruction fetch is first stage in the MIPS pipeline. The instruction fetch stage controls the program's flow and recovers the memory instructions. IF stage consists of the register program Counter (PC), synchronous instruction memory and passed signals from the ID stage with the logic through branches, jumps also PC-stalled signals. The PC is still enabled; the combined logic is used to decide the address, which should be registered and then sent to the instruction memory. When the pipeline is stall or a break instruction released, the CPU keeps its current value such that instructions are not lost. When a branch or jump is active, the PC takes the value NEXTPC with the target address of the branch or jump. The default is to go to the sequential instruction with the [PC + 4] as shown in **Figure (3).**

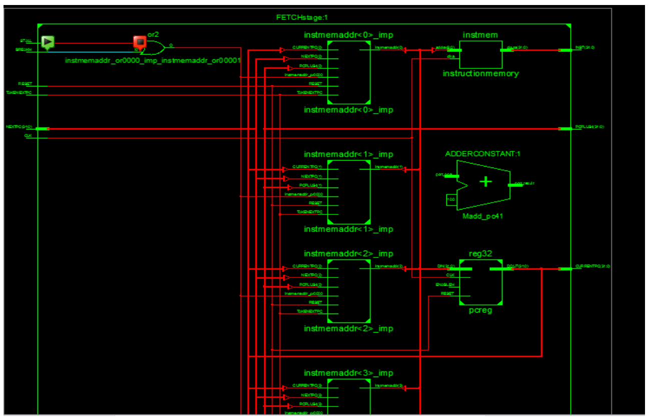


Figure 3: RTL Schematic of the Instruction Fetch Stage

The instruction memory is an IP core of Xilinx: single port block memory v7.3 is used as instruction memory. The memory is configured as a ROM. The memory works as read-only memory and can be read in a .coe file, which holds the instructions. The size of memory is 32: 1024, 32 is the width and 1024 is the depth of the memory.

4.1.2 Instruction Decode (ID)

Instruction decode is the second stage of MIPS pipeline. It decodes the instruction in instruction register (IR), computes the next PC, and reads any operands, which is required from the register file. The ID stage includes the register file, the next PC logic module, a branch logic module and controller. The diagram of the internal structure of instruction decoding stage shown in **Figure (4).**

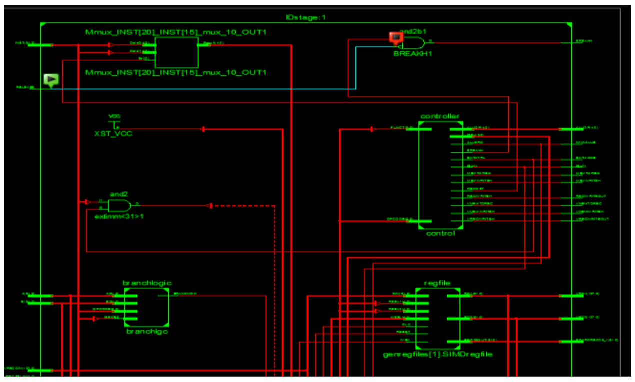
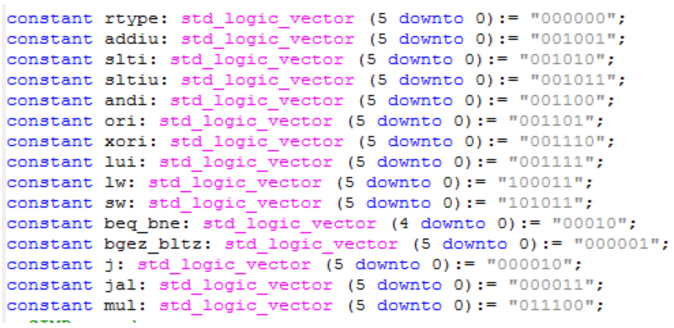


Figure 4: RTL Schematic of the Instruction Decode Stage

The work of branch module is to determine the if BEQ, BNE, BLTZ will take the branch. The register file consists of 31 general-purposeregisters and one zero register, which cannot be modified.

The control unit controls the whole unit. Control unit is describedin **Figure (5)**. The most significant six bit is used as an opcode by the controller to sets the signal for proper execution. The least significant six bits tells about the function. All types of instructions have different opcodes. R-type instructions have the same opcode. All the parameter for opcode are defined in **Figure(5)**.



```
constant rtype: std_logic_vector (5 downto 0):= "000000";
constant addiu: std_logic_vector (5 downto 0):= "001001";
constant slti: std_logic_vector (5 downto 0):= "001010";
constant sltiu: std_logic_vector (5 downto 0):= "001011";
constant andi: std_logic_vector (5 downto 0):= "001100";
constant ori: std_logic_vector (5 downto 0):= "001101";
constant xori: std_logic_vector (5 downto 0):= "001110";
constant lui: std_logic_vector (5 downto 0):= "001111";
constant lw: std_logic_vector (5 downto 0):= "100011";
constant sw: std_logic_vector (5 downto 0):= "101011";
constant beq_bne: std_logic_vector (4 downto 0):= "00010";
constant bgez_bltz: std_logic_vector (5 downto 0):= "000001";
constant j: std_logic_vector (5 downto 0):= "000010";
constant jal: std_logic_vector (5 downto 0):= "000011";
constant mul: std_logic_vector (5 downto 0):= "011100";
```

Figure 5: Instructions opcodes for different types of instructions.

4.1.3 Instruction Execution (IE)

Instruction execution is the third stage MIPS pipeline. It actually executes the instruction. In fact all the ALU operations are done in this stage. ALU stand for arithmetic logic unit which perform all the arithmetic operations including addition, subtraction, shifting, and rotating. The scheme for the EXE stage is shown in the **Figure (6).**

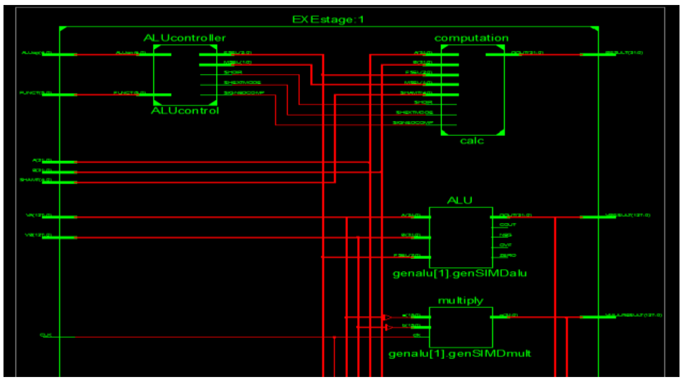
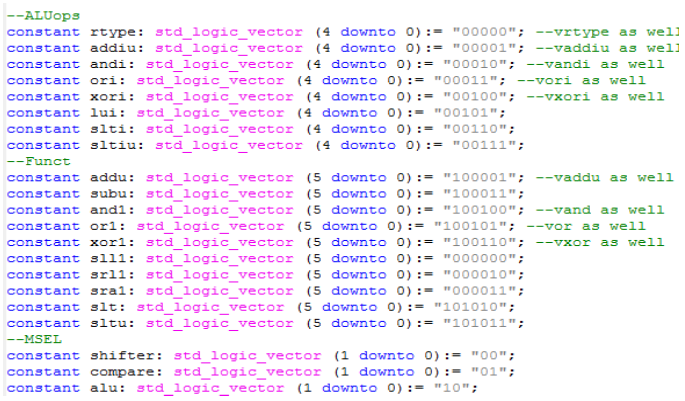


Figure 6: RTL schematic of the instruction execution stagemux and shifting modules

All the modules are controlled by the controller. Comparator is used as shifter for all shifting instructions including SLT, SLTU, SLTI, and other related instructions. The shifting unit actually compare the status flag from ALU for the correct result.For the multiply command, the multiply module is included. This multiplies the 16 small bits and recovers the output by 32 bits. The multiplier is an IP component for Xilinx: multiplier v11.2.

The functional of ALU controller is to control the ALU. The ALU controller is triggered on the FUNCT or the ALUOP and uses case statements to set the ALU controls FSEL for the ALU and MSEL for the 32x3 MUX output. The parameters for the ALUOP, FUNCT, and MSEL are given in **Figure (7)**.



```
--ALUops
constant rtype: std_logic_vector (4 downto 0):= "00000"; --vrtype as well
constant addiu: std_logic_vector (4 downto 0):= "00001"; --vaddiu as well
constant andi: std_logic_vector (4 downto 0):= "00010"; --vandi as well
constant ori: std_logic_vector (4 downto 0):= "00011"; --vori as well
constant xori: std_logic_vector (4 downto 0):= "00100"; --vxori as well
constant lui: std_logic_vector (4 downto 0):= "00101";
constant slti: std_logic_vector (4 downto 0):= "00110";
constant sltiu: std_logic_vector (4 downto 0):= "00111";
--Funct
constant addu: std_logic_vector (5 downto 0):= "100001"; --vaddu as well
constant subu: std_logic_vector (5 downto 0):= "100011";
constant and1: std_logic_vector (5 downto 0):= "100100"; --vand as well
constant or1: std_logic_vector (5 downto 0):= "100101"; --vor as well
constant xor1: std_logic_vector (5 downto 0):= "100110"; --vxor as well
constant sll1: std_logic_vector (5 downto 0):= "000000";
constant srl1: std_logic_vector (5 downto 0):= "000010";
constant sra1: std_logic_vector (5 downto 0):= "000011";
constant slt: std_logic_vector (5 downto 0):= "101010";
constant sltu: std_logic_vector (5 downto 0):= "101011";
--MSEL
constant shifter: std_logic_vector (1 downto 0):= "00";
constant compare: std_logic_vector (1 downto 0):= "01";
constant alu: std_logic_vector (1 downto 0):= "10";
```

Figure 7: ALU Power Parameters from the.vhd ALU controller

4.1.4 Memory Stage

The Memory Stage is the fourth stage in the MIPS pipeline.The Memory stage (MEM) accesses the data memory when either a SW or LW is used is given instruction. The data storage also constitutes the Xilinx IP core single port frame Memory v7.3.The memory is configured as a RAM. The memory designed to read and write and can be configured with a data carrying .coefile. The memory stage operates in the system at the same time as the EXE multiplier. It is necessary since the data file address is determined in the ALU and the value passed to the file prior to registration in the inter stage registrar. The load word data is ready for the next clock period as the memory is synchronised.

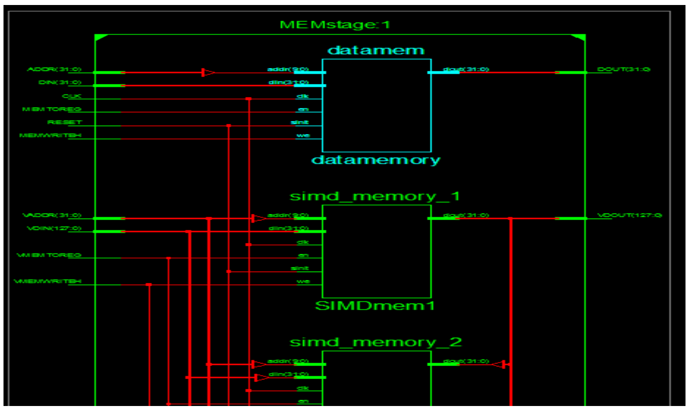


Figure 8: RTL Schematic of the Memory Stage.

4.1.5 Write Back Stage

The write back Stage is the five stage in the MIPS pipeline.The Write Back (WB) stage takes the multiplying results, result value and or memory info, and return it to the register file to compose a suitable register.

4.1.6Hazard Detection

The Detection of Hazards risk analysis has two functions. The first is to forwarding data from the currently in the pipeline instructions that have not been written back to the instruction register file in ID stage instruction that requires value. The second is to stall the pipeline when the correct value for this clock cycle is not available.

The hazard controller stops data transfer from reading after writing hazards. Writing after reading and writing after writing hazards can not happen in this execution as both orders are performed to be run according to the programme. The question of data forwarding arises when an instruction writes to a register and then the next instruction uses the register as one of its operands. Taking the directions explained in the following display:

Addu r2, r1, r3 ------ r2 = r1 + r3
xor r4, r2, r3 ------ r4 = r2 | r3
subu r5, r2, r1 ------- r5 = r2 – r1
andr6, r2, r2 ------ r6 = r2 & r2

5. Simulation Results

The schematics Register Transfer Level (RTL) of the processor can be shown after the synthesis is shown in the **Figure (9)**.

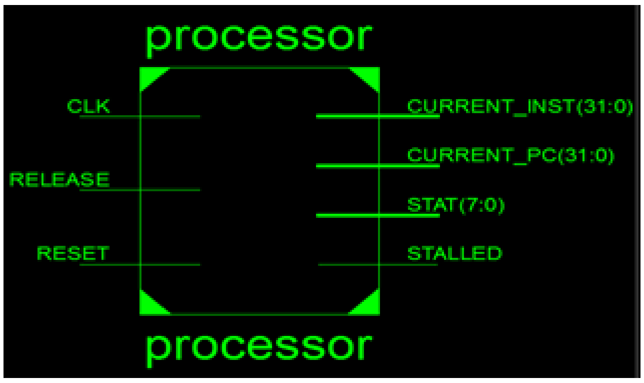
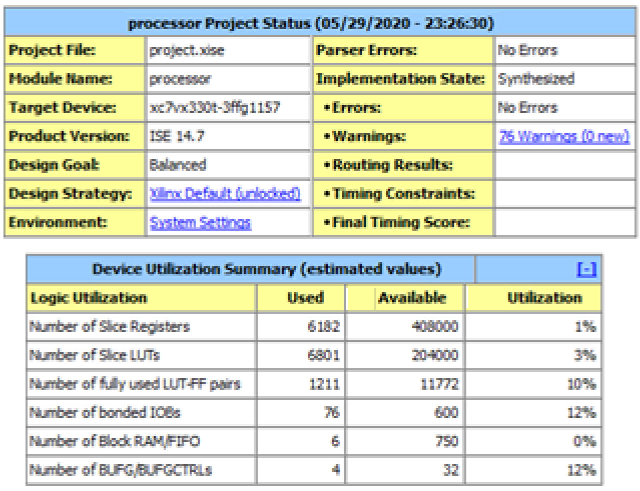


Figure 9: RTL schematic for the processor.

Device utilization for the MIPS processor 32-bit is executed by utilizing Virtex 7 on FPGA board. The design process is simplified using VHDL to design MIPS processor. The results got are appeared in the **Figure (10)**.



processor Project Status (05/29/2020 - 23:26:30)			
Project File:	project.xise	**Parser Errors:**	No Errors
Module Name:	processor	**Implementation State:**	Synthesized
Target Device:	xc7vx330t-3ffg1157	**• Errors:**	No Errors
Product Version:	ISE 14.7	**• Warnings:**	76 Warnings (0 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	6182	408000	1%
Number of Slice LUTs	6801	204000	3%
Number of fully used LUT-FF pairs	1211	11772	10%
Number of bonded IOBs	76	600	12%
Number of Block RAM/FIFO	6	750	0%
Number of BUFG/BUFGCTRLs	4	32	12%

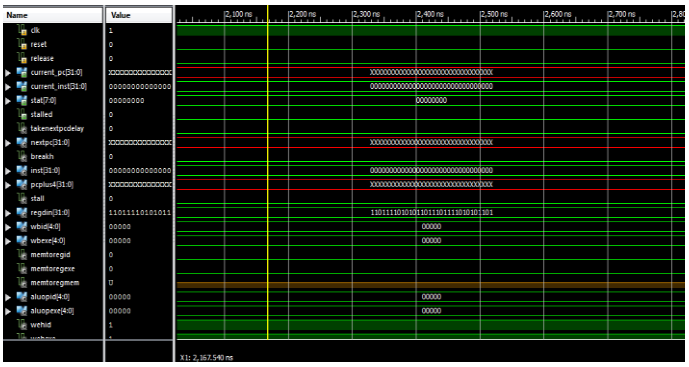
Figure 10: Summary of the Device utilization for MIPS processor

The design process is simplified using VHDL to design MIPS processor as only 1% slices register, 10% flip flops and 3% LUTs are utilized. By comparison with the results of previous studies[31]-[34], we development and get the best utilization rates for the device space. As mentioned in**Table. 2**.

Table 2: Results my work compare with other studies

Name of the authors	Number of Slice Registers	Number of Slice LUTs	Number of fully used LUT-FF pairs	Number of bonded IOBs	Number of BUFG/ BUFGCTRLs
N.Alekya , P.Ganesh Kumar, [31]	22%	122%	0%	107%	56%
Anu Mariam John ShilpiVarshne, [32]	8%	5%	5%	42%	-
Tyamanavar, Vishala A Nidagundi, Jayashree C, [33]	4%	2%	-	1%	-
Raj, Vishal Patil, Rutuja Patil, Alpesh Vishwakarma, Vikas, [34]	1%	6%	33%	33%	6%
My work	1%	3%	10%	12%	12%

The simulation result of MIPS processor 32-bit single cycle designis shown in**Figure (11).**



Name	Value
clk	1
reset	0
release	0
current_pc[31:0]	XXXXXXXXXXXXXX
current_inst[31:0]	00000000000000
stat[7:0]	00000000
stalled	0
takenextpcdelay	0
nextpc[31:0]	XXXXXXXXXXXXXX
breakh	0
inst[31:0]	00000000000000
pcplus4[31:0]	XXXXXXXXXXXXXX
stall	0
regdin[31:0]	11011110101011
wbid[4:0]	00000
wbexe[4:0]	00000
memtoregid	0
memtoregexe	0
memtoregmem	U
aluopid[4:0]	00000
aluopexe[4:0]	00000
wehid	1

Figure 11: Simulation result of a 32-bit pipeline MIPS processor design

6. Conclusion

Pipelined MIPS processor 32-bit was designed and implemented on Xilinx Virtex7 xc7vx330t FPGA using VHDL. Xilinx ISE Design Suite 14.7 platform were used for simulation and testingthe SIMD processor introduced 39 instructions, utilising the Harvard memory strategy that usedfive stages pipeline to improved performance the speed of processor 191.150MHz, it is observed that the development in the speed of processor (191.150MHz) instead of (177MHz)compared to [32]. Results were obtained the total device utilization of as only 1% slices register, 10% flip flops and 3% LUTs are utilized. The instruction set is small but strong and can be easily reconfigured to fit future needs.

References

[1] Omran, S., &Jumma, L.,"Implementation of 4-way Superscalar Hash MIPS Processor Using FPGA". Journal of Physics: Conference Series,Vol. 1003, No. 1,pp.1-8, 2018.

[2] Husainali, S., Hitesh, N. &Abhishek, A.,"Design of 32-bit 3-Stage Pipelined Processor based on MIPS in Verilog HDL and Implementation on FPGA Virtex7". International Journal of Applied Information Systems, Vol.10,No. 9, pp.26-37, 2016.

[3] Al-Araji. A., "Development of an on-line self-tuning FPGA-PID-PWM control algorithm design for dc-dc buck converter in mobile applications". Journal of Engineering, Vol. 23, No. 8, pp. 84-106, 2017.

[4] Tanabe, S., Nagashima, T., & Yamaguchi, Y.," A study of an FPGA based flexible SIMD processor". Journal ACM SIGARCH Computer Architecture News, Vol. 39,No. 4, pp.86-89, 2011.

[5] Najjar, W., &Ienne, P.,"Reconfigurable Computing". IEEE Micro,Vol.34,No.1,pp. 4-6, 2014.

[6] Xiao, C., Huang, Z., & Li, D.," An Embedded Multicore Platform Exploration in Video Application Utilizing FPGA". Advanced Materials Research, Vol. 505,pp. 329-337, 2012.

[7] Bobrek ,M.,Bouldin, D., Holcomb, D., Killough, S., Smith, S., and Ward, C.,"Survey of Field Programmable Gate Array Design Guides and Experience Relevant to Nuclear Power Plant Applications". Engineering Science and Technology Division, 2007.

[8] Domínguez, C., Hassan, H., Crespo, A., &Albaladejo, J. "Multicore and FPGA implementations of emotional-

based agent architectures". The Journal Of Supercomputing,Vol.71,No.2,pp.479-507, 2014.
[9] Poovendran, R., &Sumathi, S." An Area-Efficient FPGA Implementation of Network-on-Chip (NoC) Router Architecture for Optimized Multicore-SoC Communication". Sensor Letters, Vol.16, No.7, pp. 552-560,2018.
[10] Nouri, S., Rossi, D., &Nurmi, J.," Power mitigation of a heterogeneous multicore architecture on FPGA/ASIC by DFS/DVFS techniques". Microprocessors and Microsystems,Vol. 63, pp. 259-268, 2018.
[11] Sinha, S., Jarvinen, K., Vliegen, J., Vercauteren, F., &Verbauwhede, I.,"HEPCloud: An FPGA-based Multicore Processor for FV Somewhat Homomorphic Function Evaluation". IEEE Transactions on Computers,Vol.67,No.11, pp. 1637 – 1650, 2018.
[12] Vipin,K.,&Suhaib A.,"FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications". ACM Computing Surveys,Vol. 51,No.4, pp.1-39,2018.
[13] Kenny, R., Watt, J.,"The Breakthrough Advantage for FPGAs with Tri-Gate Technology Transistor".2010.
[14] Verma, G.,Kumar, M.,Khare, V.,& Pandey, B.," Analysis of Low Power Consumption Techniques on FPGA for Wireless Devices".Vol. 95,No. 2,pp.353-364.2017.
[15] Wang, G., &Gao, Y.,"An Implementation of Configurable SIMD Core on FPGA". Journal Applied Mechanics and Materials, Vol. 336-338, pp. 1925-1929, 2013.
[16] Rhu, M., &Erez, M.,"Maximizing SIMD resource utilization in GPGPUs with SIMD lane permutation". Journal ACM SIGARCH Computer Architecture News, Vol.41, No.3, pp.356-367, 2013.
[17] Anu, M., &Shilpi, V.," FPGA Implementation of 32-bit MIPS Processor with CISC Multiplication Operation". International Journal of Engineering Research and Technology, Vol.4, No.11, pp.675-678,2015.
[18] Hadizadeh, A., &Tanghatari, E.,"Parallel Processor Architecture with a New Algorithm for Simultaneous Processing of MIPS-Based Series Instructions". Emerging Science Journal, Vol.1, No.4,pp.226-232, 2018.
[19] Maheswari, R., Pattabiraman, V., &Sharmila, P.,"Reconfigurable FPGA based soft-core processor for SIMD applications" . Asian Journal of Pharmaceutical and Clinical Research, Vol.10,No.13, pp.180-186,2017.
[20] Yangzhao,Y , Naijie, G., Kaixin, R. ,& Bingqing, H. ,"An Approach to Enhance Loop Performance for Multicluster VLIW DSP Processor". Published in: ARCS 2014; 2014 Workshop Proceedings on Architecture of Computing Systems. pp. 1-8, 2014.
[21] Mahmood, B.,&Jbaar, M., "Design and implementation of SIMD Vector Processor on FPGA". International Symposium on Innovations in Information and Communications Technology, Amman, pp. 124-130, 2011.
[22] Evgueny,k. ,&Jamie, E. ,"Vectorization:A Key Tool To Improve Performance On Modern CPUs". Published on January 25, 2018.
[23] Indira ,P.,&Kamaraju, M.," Design and Implementation of 6-Stage 64-bit MIPS Pipelined Architecture". International Journal of Engineering and Advanced Technology, Vol.8, pp.790-796, 2019.
[24] Omran, S., &Jumma, L.," Design SHA-2 MIPS Processor Using FPGA". Cihan University-Erbil Scientific Journal, Vol.2017(Special-1),pp.1-12, 2017.
[25] Prasanth, V., Sailaja, V., Sunitha, P., &Vasantha, B.,"Design and implementation of low power 5 stage pipelined 32 bits MIPS processor using 28nm technology". International Journal of Innovative Technology and Exploring Engineering, Vol. 8,No.(4S2),pp.503-507, 2019.
[26] Singh, K.," Performance Improvement in MIPS Pipeline Processor based on FPGA". Conference: 3rd International Conference on Emerging Trends of Engineering Science Management and its ApplicationsAt: IIC, New Delhi, India,Vol. 4, No. 1,pp.57-64, 2016.
[27] Ruckmani,D., Srinivas,N., Shashi,S. Ruckmani,D., &Byrareddy,H., "Implementation and verification of RISC processor on FPGA using chipscope pro tool". International Journal of Current Engineering and Scientific Research, Vol.6, No.6, pp. 59-65,2019.
[28] Mahmood, H., &Omran, S.," Selective branch prediction schemes based on FPGA MIPS processor for educational purposes". IOP Conference Series: Materials Science and Engineering, Vol.518,No. 4,2019.
[29] Elkateeb, A.,"A Processor Design Course Project: Creating Soft-Core MIPS Processor Using Step-by-Step Components' Integration Approach". International Journal of Information and Education Technology, Vol.1, No.5,pp. 432-440, 2011.
[30] Indira, P .Kamaraju, M &Vyas , V.," Design and Analysis of A 32-bit Pipelined MIPS Risc Processor". International Journal of VLSI Design & Communication Systems, Vol.10, No.5, pp.1-18,2019.
[31] Alekya, N, Kumar, P.,"Design of 32-bit RISC CPU based on MIPS". Journal of Global Research in Computer Science .Vol.2, No.9,pp.2-6, 2011.
[32] John, A. &Varshney, S.,"FPGA Implementation of 32-bit MIPS Processor with CISC Multiplication Operation". International Journal of Engineering Research, Vol.4, No.11, pp. 675-678, 2015.
[33] Tyamanavar, V., Nidagundi, J.," FPGA Implementation of a 32-Bit MIPS Processor". International Journal of Engineering Research & Technology (IJERT), Vol.7,No.10,pp.1-5,2019.
[34] Raj, V., Patil, R.,Pati, A.,Vishwakarma, V.&Preeti-Hemnani,"32-bit Processor Design on FPGA".Journal of Applied Science and Computations.Vol. 6, No. 4, pp.3484-3490, 2019.