DOI: https://doi.org/10.33103/uot.ijccce.21.2.2

FPGA-Based Multi-Core MIPS Processor Design

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

[1,2,3]*Computer Engineering Dept, University of Technology, Baghdad, Iraq*

[1]*Saramahdi2018@gmail.com*, [2]*60166@uotechnology.edu.iq*, [3]*bassamacm@gmail.com*

Abstract— *This research presents a study for multicore Reduced Instruction Set Computer (RISC) processor implemented on the Field Programmable Gate Array(FPGA).The Microprocessor without- Interlocked Pipeline Stages (MIPS) processor is designed for the implementation of educational purposes, as well as it is expected that this prototype of processor will be used for multimedia or big data applications. 32- bit MIPS processor was designed by using Very High speed Hardware Description Language (VHDL). Pipelined MIPS processor contains three parts that are : data path 32-bit MIPS pipeline, control unit, and hazard unit. The single cycle MIPS system was subdivided into five pipeline stages to achieve the pipeline MIPS processor. The five parts include: instruction fetch (IF), Instruction Decode (ID), execution (EXE), memory (MEM) and Write Back (WB). Three types of hazard: data hazard , control hazard and strctural hazard are resolved. Certain components in the pipelined stage for the design processor were iterated for four core SIMD pipelined processors. The MIPS is developed using Xilinx ISE 14.7 design suite. The designed processor was implemented successfully on Xilinx* Virtex-6 XC6VLX240T-1FFG1156 FPGA. *The total power analysis of multi-core MIPS processor is obtanined 3.422 watt and the clock period was 7.329 ns (frequency: 136.444MHz).*

Index Terms— *FPGA, MIPS, RISC, VHDL.*

I.INTRODUCTION

Modern parallel computers use commodity processors, often multi-core, which allow parallel systems to have immediate access to rapid improvements in processing speed and energy efficiency [1]. Reduced Instruction Set Computer (RISC) is a kind of architecture for microprocessors using a highly optimized set of instructions. RISC processors use huge numbers of registers. The RISC computer load storage method is based on the load and store instruction set, and these instructions only use memory access. Microprocessor without Interlocked Pipeline Stage (MIPS) is a RISC set of guidelines created by MIPS technologies. Many reviews from MIPS instruction set are present, such as MIPS-32, MIPS- 64 and others [2]. The current revision of MIPS-32 is implemented for 32 bit. Pipelining is one of the features of RISC, which reduces the cycles per instruction at the expense of the number of instructions per program [3].

Today, Field Programmable Gate Arrays (FPGAs) are commonly used to implement multi-purpose logic. FPGAs are built from a complex set of basic logical functions. There are a number of FPGAs available in the market from many vendors. Additional developments in packaging allow for high performance. A Hardware Description Language (HDL) is commonly used to configure FPGAs. A very High Integrated Circuit (IC) Hardware Description Language (VHDL) is one of the hardware description languages of very high speed integrated circuits [4].

The design MIPS processor specifications have five stages: Instruction Fetch (IF), Instruction Decode (ID), Execution (EXE), Memory (MEM) and Write Back (WB), 32 bit data bus, memory 32 bit X 1024 and 32 bit internal addressable register.

Received 8/12/2020; Accepted 29/4/2021

The nature of data in such applications, including (image, audio and video) provides the opportunity for high level of parallelism in processing.

The researchers in [5] focused on the microarchitecture of MIPS Instruction Set Architecture (ISA) implemented with FPGA, since it closely follows the theory of RISC. This is desirable since it simplifies implementation and is easy for students to understand, so upgraded with 6 MIPS required instruction. It is developed using the common Hardware Description Language (HDL), Verilog. [6] presented the design and simulation of a high-performance 32-bit (MIPS) 5-stage pipelined, processor-based on RISC architecture. MIPS processor aims to execute a minuscule set of instructions, to increase the processor speed. This RISC processor was designed in five stages of a pipeline. The designing of this RISC processor is developed using the (HDL) hardware description language Verilog code in the modelsim simulator and uses the Register Transfer Level (RTL) logic Xilinx tool. While in [7] they proposed the design and implementation of a RISC 16-bit Processor by using VHDL. This processor can execute 16 instructions from different classes like arithmetic, logical, conditional and unconditional jumps and for the memory interface instructions. The authors have compared between 8-bit and 16-bit in regards to power- efficiency and programmability requirements. The authors have investigation in their research the methodology of soft-core processor development. The package uses (Xilinx ISE 14.5). The target device chosen was Spartan 6 FPGA. The simulation was executed in ISE Simulator ISIM. In [8] described the design and implementation of a dual and the quad-core pipelined RISC processor using Verilog code and implemented their processor on vertex 6 FPGA. Dual-core and quad-core processor consumes less power with high performance. Harvard memory architecture have been deployed by the authors to get separate access logically and physically to the data and instruction memories. The pipelined approach adopted by the authors has increased the execution speed for the 23 instruction set of their RISC processor. Moreover, [9] describes the a soft core five stages pipeline processor with a basic instructions set that can be changed on-demand due to the configurable nature of FPGA. This processor improves the efficiency by using pipeline concept, because of the increase in development of the processor and System On Chips (SOCs). This system uses Xilinx ISE design 14.1 and Verilog language and was implemented on FPGA Xilinx Spartan 6 XC6SLX9-3CSG324. In [10] presented proposed 32-bit pipelined MIPS processor. The proposed MIPS pipelined processor has a five-stage pipeline, 32- bit register file, and 32-bit arithmetic logic unit. The results show that the MIPS pipelined processor works on three times less power than MIPS non-pipelined. It uses Xilinx ISE software, Verilog VHDL code and implementation on Spartan 3E device XC3S500E.

The motivations for this work are: minimizing execution time, reducing the area utilized by the processor, low power consumption, and increasing the size of the processed data; therefore, the compatible multi-core processor circuit is designed, tested and synthesized by using VHDL in this work.

This paper is organized as follows; section II gives an overview of pipeline multicore MIPS processor architecture. Section III introduces and defines instruction set architecture. In section IV, the pipelined MIPS processor design gets explained. Section V, describes the pipeline instruction set. Section VI explains SIMD instructions for multicore processor. The results are discussed in section VII; finally, conclusions are lighted in section VIII, followed by references.

Received 8/12/2020; Accepted 29/4/2021

DOI: https://doi.org/10.33103/uot.ijccce.21.2.2

II. OVERVIEW OF PIPLENE AND MULTICORE MIPS PROCESSOR

A Microprocessor without Interlocked Pipelined Stages (MIPS) is a type of RISC architecture developed by computer systems. RISC is a basic architecture that has become a commonplace in recent years [11]. MIPS has attained a big name in the history of computers for several reasons. First, it is a groundbreaking processor, one of the first of its kind. Secondly, it was developed in an educational institute rather than in a big industrial organization such as Intel or Motorola. Third, it has had a huge impact on teaching or understanding of computer architecture [12]. The principles of the MIPS system were used to explain the pipeline and avoid any interlocking conditions. In general, the pipeline deploys the task of operating instructions into several steps, and beginning work on the "first step" of instructions before previous instructions are completed. The architecture of the MIPS processor omitted a range of valuable instructions to complete certain moves [13]. The MIPS basic pipeline is shown in *Fig.1*. [14].

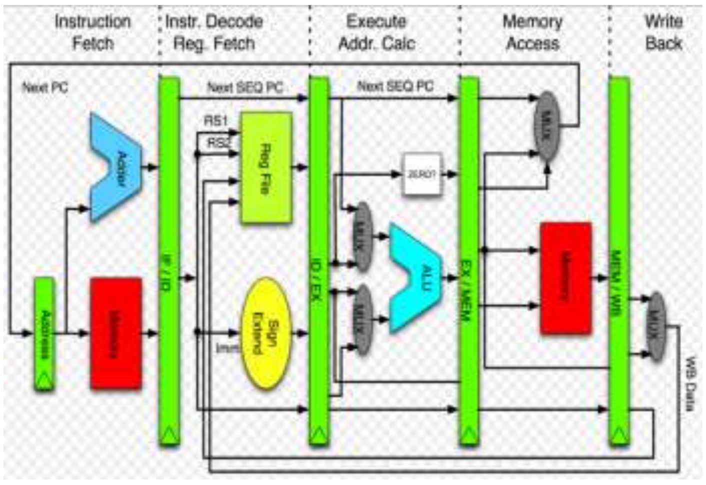


FIG.1.THE MIPS PROCESSOR BASIC PIPELINE [14].

The design of the microprocessor was typically driven by higher requirements. Several methods of the design were used to eliminate various types of parallel applications. In recent years, computer engineering has achieved technological advances that have had a great mutual effect, especially in the dissemination of Single Instruction Multi-Data (SIMD) for individual education [15]. To improve performance and decreased power consumption, they were attached in a single chip by duplicate four components of pipelined core processors that are iterated to create the SIMD architecture. It can increase the overall speed of the programs by processing multiple data at the same time [16].

Multicore is an Integrated Circuit (IC), typically a single processor that includes two or more independent processing units called cores on a chip. Multicore chips widely enter in the industrial and consumer markets, began in special-purpose, niche markets like high-performance graphics and the network devices. Multicore chips have also recently been launched into the general-purpose market[17]. A general multi-core block diagram for this work is shown in *Fig.2*.

Received 8/12/2020; Accepted 29/4/2021

DOI: https://doi.org/10.33103/uot.ijccce.21.2.2

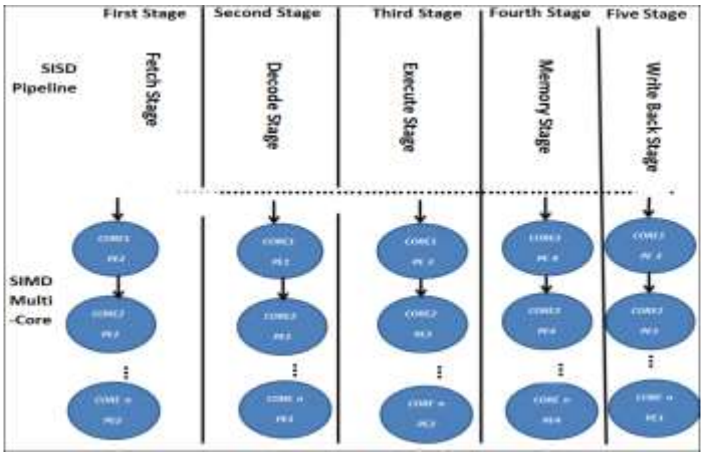


FIG.2. GENERAL BLOCK DIGRAM OF MULTICORE [18]

Fig.2 illustrates two phases; SISD and SIMD Instructions. SISD consists of five stages: instruction fetch (IF), Instruction Decode (ID), execution (EXE), memory (MEM) and write back (WB) for single core. The SIMD instructions consist of the pipeline stage of the proposed processor and iterate each processing element when it needs to be duplicated, this produces a core multi-core pipelined processor.

Advantages of multi-core systems include reduction of execution time, scalability of the data size and processor performance, cost saving, and concurrency. Nowadays, most desktop and high-performance computer processors are adopting the multicore architecture in their designs. Therefore, multi-core processors are getting an increased popularity in different fields [18].

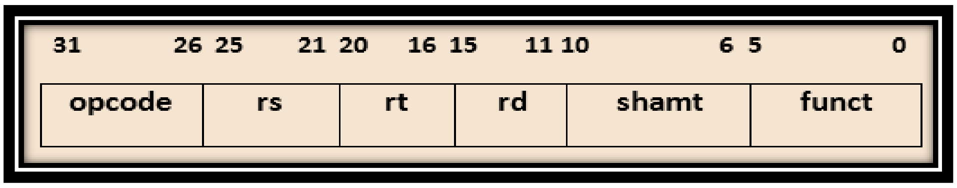
III. INSTRUCTION SET ARCHITECTURE (ISA)

A programming language must be spoken to control the computer. Control words in machine language are called instructions, and their vocabulary is called a set of instructions. In RISC processor, for design simplicity, all instructions should keep the same length and should have a single instruction format. MIPS uses 32-bit instructions and set three formats of instructions, which are [19]:

a. Register type instructions (R-type): the most popular instruction style is Register-type. As operands, they use three registers: two as sources registers, and one as destination register [20]. The format of R-type is shown in *Fig.3*.

b. Immediate type instruction (I-type): immediate-type uses two operand registers and one immediate 16-bit operand [20]. The format of I-type is shown in *Fig.4*.

c. Jump instructions type(J-type): jump-type is only used with the jump instructions and uses a 26-bit single address operand [20]. The format of J-type is shown in *Fig.5*.



31 26	25 21	20 16	15 11	10 6	5 0
opcode	rs	rt	rd	shamt	funct

FIG.3. REGISTER-TYPE INSTRUCTIONS FORMAT [20]

Received 8/12/2020; Accepted 29/4/2021

DOI: https://doi.org/10.33103/uot.ijccce.21.2.2

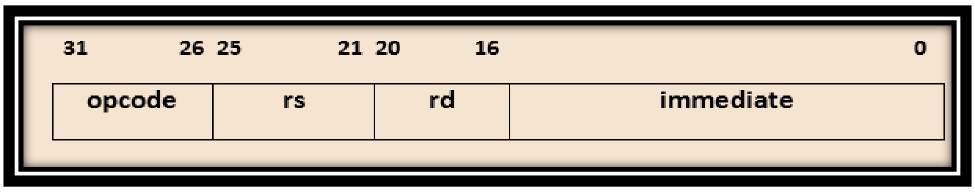


FIG.4. IMMEDIATE –TYPE INSTRUCTIONS FORMT [20]

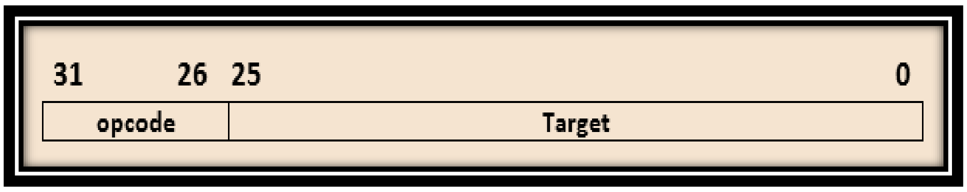


FIG.5. JUMP-TYPE INSTRUCTIONS FORMAT [20]

Where, op: basic instruction operation which is traditionally called an opcode.
rs: the first source register operand.
rt: the second source register operand.
rd: destination register operand, it gets operation result.
shamt: shift amount is used to hold a shift amount in shift instructions.
funct: function, it selects the part variant of the operation in the opcode field.
Immediate: 16-bit address is used in the instructions for the data transfer.
Target: 26-bit address is used in instructions for jumping.

IV. PIPELINE MIPS PROCESSOR DESIGN

In the MIPS processor with a pipeline architecture of five stages, the instructions are executed in five phases, where each stage takes a fixed time period. The specified period normally is a clock cycle of the processor [21]. All instructions follow up the same pipe stage sequence, even though the instruction does nothing in some stages. Stages in the developed MIPS processor are as follows: Instruction Fetch (IF), Instruction Decode (ID), Instruction Execution (IE), Memory Stage, Write Back (WB) Stage [22]. MIPS pipeline processor contains three parts: 32-bit data path on pipeline, control unit, and hazard detection unit.

A. Data Path 32-Bit MIPS Pipeline

The data path is the combination of different functional units such as arithmetic logic unit, multiplier, registers, and busses that perform processing operations including the control unit. The MIPS data path is divided into five stages for the ease of development and implementation [23]. *Fig. 6* displays the phases used in the proposed processor. Each stage in the proposed MIPS processor has a particular function, these functions will be illustrated in the upcoming subsections:

Received 8/12/2020; Accepted 29/4/2021

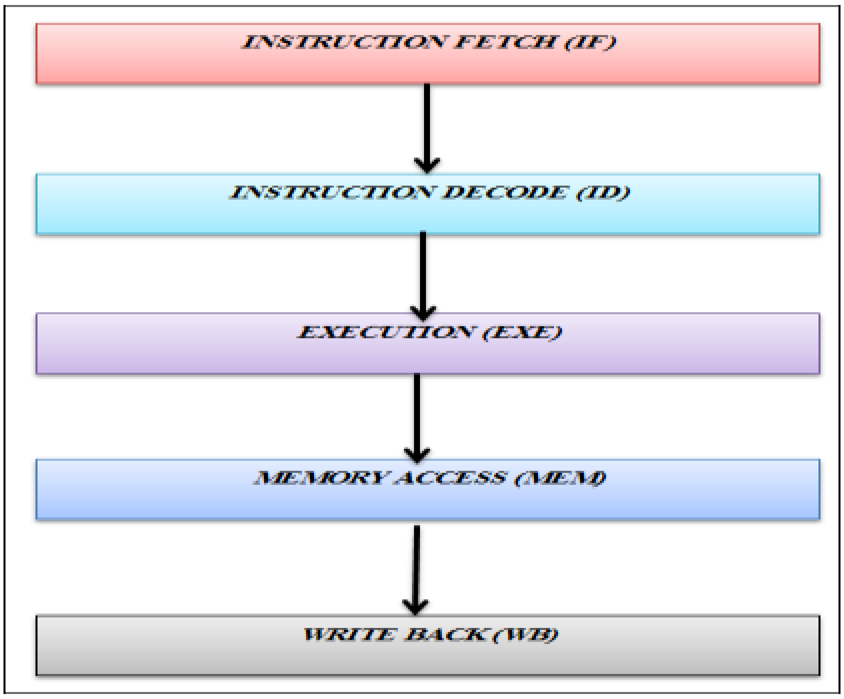


FIG.6. STAGES OF PROPOSED PROCESSOR [10]

• Instruction Fetch (IF):is the first stage in the pipeline MIPS processor. It consists of the following:

- Program counter register (pc),
- Multiplexers (Mux),
- Adder to increase PC register by 4,
- OR gate,
- 32-bit word-addressable instruction memory read only memory (ROM).

The interior buses connect those different units. The program instructions are fetched and forwarded to the next stage for decoding. The instruction memory obtains the location address for each instruction from the program counter which always enables it to provide the proper address as long as the processor is operated. The multiplexers are used to select the address and be supplied and then transmitted to the instruction memory and PC register simultaneously. The instruction memory is a single port type v 7.3 Xilinx IP core. The instruction memory is set up as a ROM, which reads in a coefficient (.coe) file that holds the instructions, with a width of (32) and depth of (1024).

• Instruction Decode (ID): is the second stage in the proposed MIPS processor. This part is where the program instructions are analyzed to generate the control signals needed to execute each instruction. It consists of the following components:

- Register bank (register file),
- Next Program Counter (NPC) logic,
- Branch logic unit,
- Control unit,
- Hazard detection unit.

The register bank consists of 32-General Purpose Registers (GPR). The branch logic is the unit that determines the presence of any instruction that can cause a change in the sequential execution of the program instructions, such instruction could be anyone of the group branch on equal (beq), branch not equal (bne), branch greater or equal zero (bgez) and branch less than zero (bltz). The inputs to this unit are supplied from the forwarding multiplexers (MUX), The module of NPC uses the ISBJ (branch or jump) signal, which produced by the control unit and forwarded to the NPC module for

Received 8/12/2020; Accepted 29/4/2021

calculating the next address point at which program control must be transferred and the control unit is the module responsible for instructions decoding and setting the signals necessary to pave the way for appropriate execution.

• Instruction Execution (EXE): is the third stage MIPS pipeline. It executes the instruction. All the ALU operations are done in this stage, that are composed from the following units:

- Shifter,
- Comparator,
- Arithmetic Logic Unit (ALU)
- ALU controller,
- Multiplier.

The shifter module is responsible for executing the shift instructions considered in this work. These instructions include; Shift word Left Logical (SLL), Shift word Right Arithmetic (SRA) and Shift word Right Logical (SRL). The comparator module is responsible for executing the instructions: Set Less Than (SLT), Set Less Than Unsigned (SLTU), Set Less Than Immediate (SLTI), set less than immediate unsigned. The ALU controller module has two inputs; the arthmatic logic unit opration (ALUOP) which is a 5-bit width input is provided from the previous decode stage specifically from the controller module at that stage, and the input is function part from the instruction which a 6-bit width as previously known. A multiplier unit v11.2 is Xilinx IP core. The multiplier module is developed to execute the multiplication instruction. It multiplies the 16 least significant bits and produces a 32- bit result. The ALU module performs all the arithmetic and logic operations.

• The stage memory (MEM) of the fourth stage of the proposed MIPS pipelined processor is the data memory which is used as the name suggests to store the data needed in the performed processing. The data memory is brought from the Xilinx IP core which is a single port block v7.3. The memory is configured for reading and writing operations and can be set up with a coefficient (COE) file containing data required to perform the planned operations.

• Write Back Stage (WB): In the processor's final stage, the role of this stage is to forward either the result of the multiplier, a result value or a data memory to the register bank to write the correct register.

B. Control Unit (CU)

The data path of the pipelined sends the (opcode) and (func) fields from instruction to the control unit in the (ID) stage. The control unit receives opcode and function, it produces the control signals necessary for executing the instruction with the corresponding opcode. To stay synchronized with instruction, these control signals are pipelined along with the data across the inter stags register. The control unit consists of the following:

- **Main Control Unit:** uses opcode (instructions bit 31 down to 26) field and (funct) (instructions bit 5 down to 0) field as inputs in the decode stage and produces the control signals that are shown in Table 1.

Received 8/12/2020; Accepted 29/4/2021

DOI: https://doi.org/10.33103/uot.ijccce.21.2.2

TABLE 1. THE EFFECT OF CONTROL UNIT SIGNAL

Signal Name	**Description**
REGWRITEH	goes to the register bank and check when to allow register writes.
BREAKPOINT	goes to the IF stage and to the inter-stage registers in the processor to disable them from going to the next instruction.
MEMTOREG	Connect to the MUX write register to determine if memory data is stored. This should be written into the register bank or a calculation should be written and it function as the activation signal to the read memory enable.
MEMWRITEH	used as activation signal to the memory write input.
ISBJ (is branch or jump)	Goes to the NPC module to determine with other signal the next address in cases of branch and jump.
ISJAL	It used as an input to the hazard controller (detection) to handle the jal instruction.
ALUOP	is input to the ALU control unit.
ALUSRC	passes to MUX B, which calculate to pass value either from forward M MUX or the output of the ALU B source extender.
REGDST	Is used to identify register bank to write back to.
EXTCTRL	Connects to the extension logic to decide whether there is sign extend or zero.

- **ALU Control Unit**: receives ALU op from the main control unit in the (ID) stage and (funct) (instructions 5 down to 0) from instruction in order to produce the signals as shown in the Table 2.

TABLE 2. THE EFFECT OF ALU CONTROL SIGNALS

Signal Name	**Description**
SHEXTMODE	Goes to the shifter unit, and calculate a shift between zero and sign.
SHDIR	Calculates the shift direction and determines what direction to shift.
SIGNEDCOMP	Pass to the comparator unit and select whether to compare signed or unsigned.
FSEL	Calculate which operation will be performed inside the ALU.
MSEL	Select between the outputs of the shifter unit, comparator unit output, and ALU output using the MUX computation.

C. Hazard Detection Unit

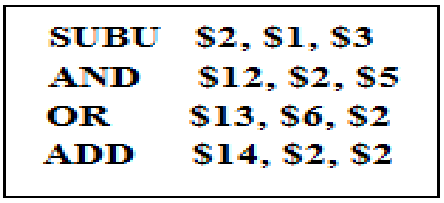
If the instructions prevented at any stage from being executed, it means that there is a problem with the pipeline, called hazard. It is identified by the hazard detection. Three types of hazards generally exist:

- **Structural Hazard:** this is caused when resource scarcity. If instructions and data require same resource at the same time, the structural hazard occurs. To resolve this problem the subsequent instruction must be stall to from entering the pipeline or another solution can be

Received 8/12/2020; Accepted 29/4/2021

made for the problem by adding one memory for instructions and another one for data. In this work it uses a second method by adding two memories to resolve this problem.

- **Data Hazard:** data hazard would occur if an instruction in specific stage e.g., EXE stage requires a data item supposed to be produced by a preceding instruction currently in further stage e.g., ID stage but this data is not available yet. There are two functions in the framework for hazard analysis the first approach is to forward data from the stage where the preceding instruction has produced the data required by the subsequent instruction for being used as one operand in the intended operation. The second is to stall the pipeline when the correct value for this clock cycle is not available. The hazard unit stops data transfer in cases Write After Write (WAW) and Write After Read (WAR) hazards. Write After Write (WAW) cannot happen in the proposed design in this work because, all instructions required five stages, and write operation always happens in the fifth stage 5. Also, Write After Read (WAR) hazards cannot happen in the proposed design since the operand read operation take place in the ID stage, and write operations performed in the WB stage. 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 *Fig.7* below.

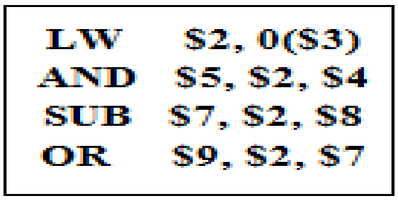


```
SUBU  $2, $1, $3
AND   $12, $2, $5
OR    $13, $6, $2
ADD   $14, $2, $2
```

FIG.7. FORWARDING HAZARD SEQUENCE

If the SUBU instruction enters the execution stage, the AND instruction enters the decode stage. The update value for register $2 is required for the AND instruction and the value must be transmitted from the execution stage. For the OR instruction, the same approach applies. When the OR instruction enters the decode stage, SUBU instruction enters the WB stage. Register $2 value is not updated yet, therefore, the correct value must be forwarded to the ID stage from the WB stage for this case. The result of SUBU is added to the $2 in the register bank when the ADD instruction enter the decode stage.

The second problem for hazard detection unit is the need to stall the pipeline with a latency that the forwarding multiplexer will handle take care of with the instruction load. However, if the instructions for the loaded register are in the decode stage. AND instruction is require $2 value but result is not visible until the WB stage like the load word instruction. The hazard unit sends a stall signal to stop the IF and ID stages as the other stages are continue. The next stage is the WB load instruction and the value is at the memory output and is returned to the EXE stage for the instructions needed. This stall process is shown in the *Fig.8*.



```
LW    $2, 0($3)
AND   $5, $2, $4
SUB   $7, $2, $8
OR    $9, $2, $7
```

FIG.8 : STALL HAZARD SEQUENCE

- **Control Hazard**: while performing the main program, the need for a branch arises and hence the execution path changes from the main to the sub-program. There is a need to stall

Received 8/12/2020; Accepted 29/4/2021

instructions that were already fetched. The design of this processor allows delay slot after the branch instruction; the branch or jump instructions will continue their execution. The branch logic unit to let the pipeline to keep the instruction directly follows the branch instruction and create no operation (NOP) in place of the killed instruction in the pipeline path. The branch logic module could be resolved in the execution stage but there are two instructions in the pipeline path to be killed. Therefore, the instructions in this work will be executed in the decoding stage, wherein the instructions should only be killed at the fetch stage.

The inter-stage links connect the data path for all the processor stages, control unit and hazard detection unit to composed the complete processor.

V. PIPELINE INSTRUCTION SETS

MIPS pipelined processor instruction sets have the following functions:

- All the MIPS instructions are of the same length, this makes it easier to fetching of instructions in the first stage and the decoding of instructions in the second stage.
- MIPS processor only has a few formats of instruction, in the register format, source fields are placed in the same location for each instruction. This symmetry means that the decode stage of the pipeline will start reading the register file in the same time as the hardware decides the type of instruction that was received.
- Memory operand only appears with load and store instructions. This restriction means that the execution phase will calculate the memory address and the data memory will be accessed in the next phase [26].

A group of twenty eight represents the most commonly used SISD instructions that were investigated, this instruction set appears in Table 3, whereas a group of eleven SIMD instructions appears in Table 4. Both groups have been considered for investigation and execution by the proposed MIPS pipelined processor.

TABEL 3. INSTRUCTION SET FOR SISD

Instruction Name	Description
ADDIU	Unsigned register addition and immediate value.
AND	Logical and, of two register.
OR	Logical or of two registers.
XOR	Logical xor of two registers.
SLL	The shift left logically.
SRA	The shift right arithmetic.
SLTI	Set on less than immediately.
LU	Instant upper load.
BLTZ	Branch on 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	Logical and the register and the immediate value.
ORI	Logical or between a register and immediate value.
XORI	The xor logical of register and immediate value.
SLTIU	Set less than the unsigned immediate.
BEQ	The Branch on equal.
BGEZ	Branch greater than or equal to zero.
SLTU	Set on less than unsigned.
BNE	Branch on inequality.
LW	The word load.
MUL	A signed multiplication of two registers.
SLT	Set on less than that.
BREAKPOINT	Breakpoint exception.

Received 8/12/2020; Accepted 29/4/2021

DOI: https://doi.org/10.33103/uot.ijccce.21.2.2

Instruction Name	Description
J	Jump.
JAL	Jump and link.
JR	Jump register.

TABEL 4. INSTRUCTION SET FOR SIMD

Instruction Name	Description
MADDIU	addition of two unsigned registers to n- paires.
MADDU	addition unsigned of register and immediate value to n-pairs.
MMUL	multiplication of two registers to n-pairs.
MSW	n store of words.
MAND	logical and of two registers to n-pairs.
MANDI	logical and of register and immediate value to n-pairs.
MOR	logical or of two registers to n-pairs.
MORI	logical or of register and immediate value to n-pairs.
MXOR	logical xor of two registers to n-pairs.
MXORI	logical xor pairs of register and immediate value to n-paries.
MLW	n load of word.

The addition, subtraction, and multiplication operations cover the arithmetic process; Whereas the logical operation are carried out through the following instruction set: or, ori or, ori, xor, xori, and, andi, sll, srl and sra cover changing needs; the configuration of less than and instructions requires comparisons; jump instructions**.** The load and store instructions are memory-related instructions, where their executions require data memory access.

VI. SIMD INSTRUCTIONS FOR MULTICORE PROCESSOR

To achieve a multi-core processor, it employs the SIMD architecture and SIMD instructions. It is needed to add and connect several basic units within the architecture proposed for the multi core processor. The block diagram of this architecture is shown in *Fig. 9*.

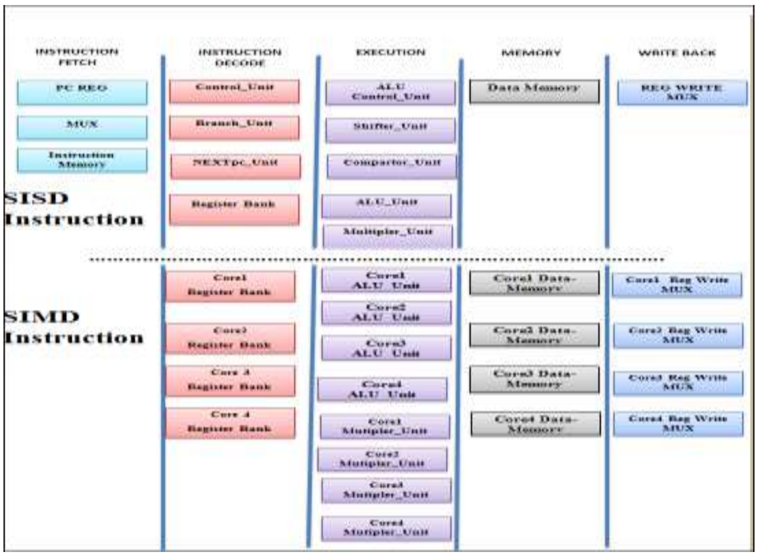


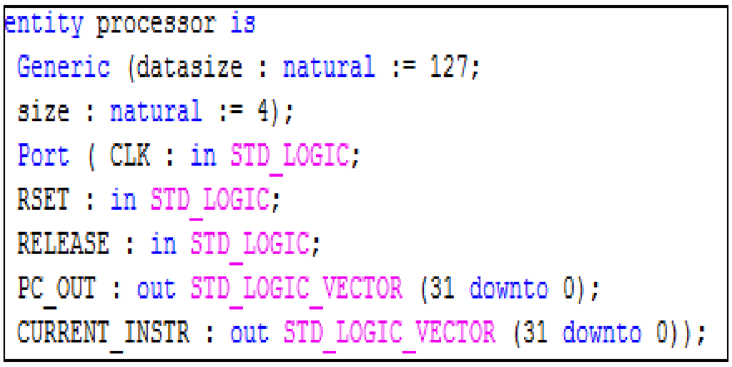
FIG.9.MULTI CORE FOR PROPOSED PROCESSOR

Received 8/12/2020; Accepted 29/4/2021

The modules in the processor stages include parameters SISD; the instructions fetch stage consisting of the PC reg, multiplexer, and instruction memory. The decode stage includes control unit, register file, nextpc module, and branch unit. Execute stage consists of ALU control, shifter unit, compartor unit, ALU unit and multiplyer unit. Memory stage contain data memory. Write back stage include register write back. The replication of units can be adjusted i.e. incremented or decremented according to the design targets and processor needs. Four units are instantiated according to the default value (N=4). Each unit includes a register bank, multiplier, ALU, data memory, inter-stage registers and forwarding multiplexers.

Adding SIMD instructions to the program does not change the program instructions flow, therefore all the modifications supposed to be made in the ID stage were limited to the controller module, three control signals were added to manipulate the vector register write (MREGWRITE), vector memory reads (MMEMTOREG) and vector writes (MMEMWRITE) for SIMD instructions such that to prevent any conflict between the datapath for each the SISD and SIMD. These signals will control the N-instantiated SIMD units, where the same function will be performed by each instantiated unit. The development of the SIMD processor has adopted distributed memory organization so that every memory unit will receive the same address, this approach can simplify the addressing mode utilized by the SIMD processor.

The number of SIMD units can be increased or even decreased by adjusting the parameter used for this purpose in the VHDL program section related to the description of main inputs and outputs i.e. the entity part of the processor in the most top-level VHDL description as shown in *Fig.10.*



```
entity processor is
 Generic (datasize : natural := 127;
 size : natural := 4);
 Port ( CLK : in STD_LOGIC;
 RSET : in STD_LOGIC;
 RELEASE : in STD_LOGIC;
 PC_OUT : out STD_LOGIC_VECTOR (31 downto 0);
 CURRENT_INSTR : out STD_LOGIC_VECTOR (31 downto 0));
```

FIG.10 SIMD PROCESSOR ENTITY

The size parameter in the figure represents the number of SIMD units that are going to be instantiated whereas the datasize parameter stands for (the number of SIMD units x 32 -1). The data signals in the design of the SIMD processor use the form of array which is a user-defined type in VHDL, this enumerated data type is used to connect SIMD ALU modules, the multiplexers and registers. The whole length of the data-array type must conform with the datasize parameter.

VII. RESULTS ISE OF SIMULATE

The processor was implemented with 2 phases, at first the SISD processor was developed as a five stages pipeline processor, thereafter, the original design was upgraded to yield four cores pipelined SIMD processor through adding all the architectural requirements at the different stages form the pipeline. Each stage has been tested using the tests for program instruction that are shown in the next section. The programs were verified by simulating all the processor stages using Xilinx ISE 14.7 Project navigator.

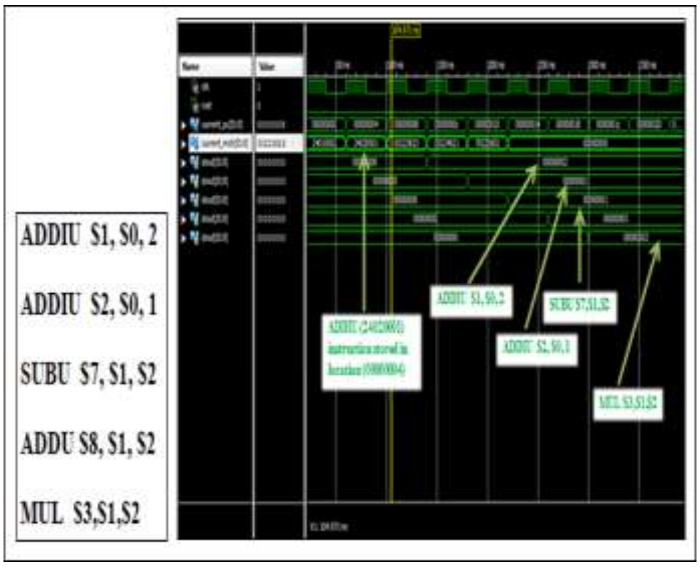
Received 8/12/2020; Accepted 29/4/2021

DOI: https://doi.org/10.33103/uot.ijccce.21.2.2

A. Simulation Results

This section will present the simulation results for arithmetic / logical instructions, memory instructions and branches instructions for SISD pipeline processor and all SIMD instructions for multi-core processor.

Fig.11 shows a code snippet and waveform resulted from testing the arithmetic SISD instructions, which include the instructions ADDIU, ADDU, SUBU and MUL. The processor goes into reset state when the rset signal becomes active (active-low), then the first instruction can be seen in the waveform, the result is written to register \$1, also other ADDIU result is written in the \$2 register. Instruction SUBU \$7,\$1,\$2 is performed, first register \$1 value and register \$2 value is read then the subtraction result is stored into the destination register \$7, when instruction ADDU is performed \$8, \$1,\$2, first register \$1 value and register \$2 value are read and then the addition of these two registers is stored into the destination register \$8.



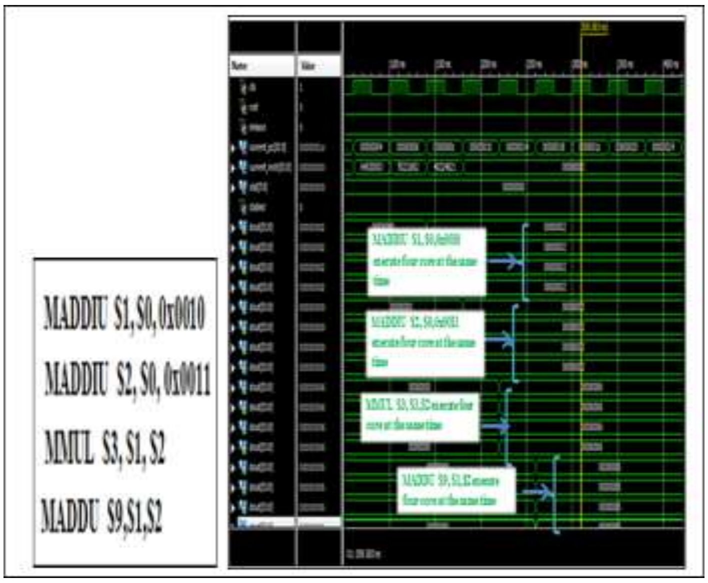
```
ADDIU $1, $0, 2
ADDIU $2, $0, 1
SUBU $7, $1, $2
ADDU $8, $1, $2
MUL $3,$1,$2
```

FIG.11. TEST CODE AND WAVEFORM FOR ARITHMETIC SISD INSTRUCTIONS

Fig.12 lists the arithmetic SIMD instructions and the waveform for the complete execution of the instructions, MADDIU, MMUL and MADDU. The processor will reset when rset signal goes low then the first instruction MADDIU goes to fetch stage. The result is written in the \$1 register, also the other MADDIU result is written in the \$2 register, when instruction MMUL \$3, \$1,\$2 is performed, first register \$1 value, and register \$2 value are read and then the multiplication of these two registers is stored into the destination register \$3. Moreover, instruction MADDU \$9, \$1,\$2 is performed, first register \$1 value, and register \$2 value are read and then the addition of these two registers is stored into the destination register \$9.

Received 8/12/2020; Accepted 29/4/2021

DOI: https://doi.org/10.33103/uot.ijccce.21.2.2

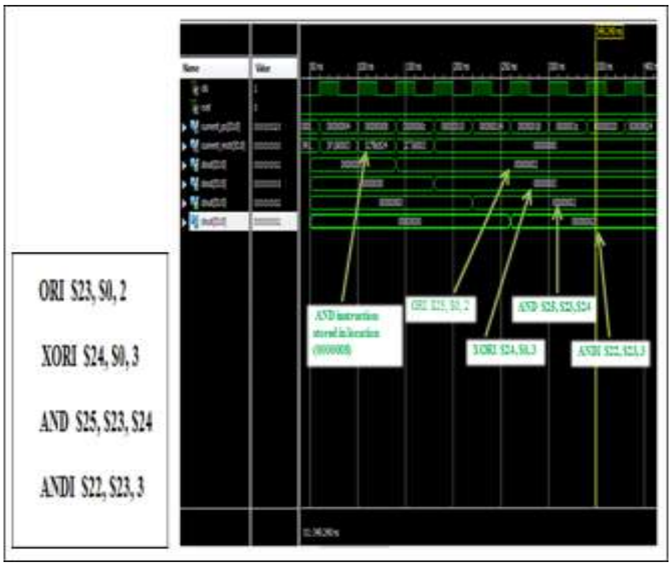


```
MADDIU $1,$0,0x0010
MADDIU $2,$0,0x0011
MMUL $3,$1,$2
MADDU $9,$1,$2
```

MADDIU $1,$0,0x0010 execute four core at the same time
MADDIU $2,$0,0x0011 execute four core at the same time
MMUL $3,$1,$2 execute four core at the same time
MADDU $9,$1,$2 execute four core at the same time

FIG.12. TEST CODE AND SIMULATION RESULT FOR ARITHMETIC SIMD INSTRUCTIONS

Fig.13 lists the test code and the waveform for the complete execution for the instructions, ORI, AND, XORI and ANDI. In instruction ORI $23, $0,2 is performed. Immediate data is read and stored into the destination register $23.



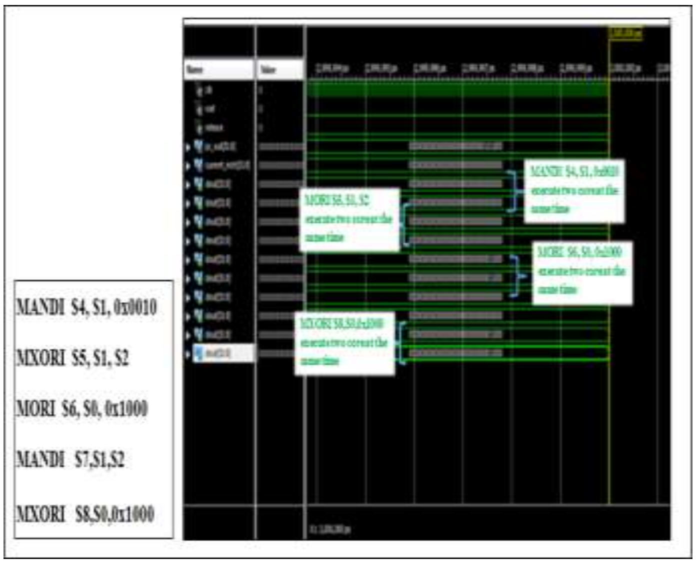
```
ORI $23,$0,2
XORI $24,$0,3
AND $25,$23,$24
ANDI $22,$23,3
```

FIG.13. TEST CODE AND SIMULATION RESULTS FOR LOGIC SISD INSTRUCTIONS

Fig.14 lists the code logic SIMD instructions and waveform for the complete execution of the instructions, MANDI, MORI and MXORI, when the MORI instruction is performed, the immediate data undergoes to MORI operation with content of register $0, whereas the result is stored in register $6.

Received 8/12/2020; Accepted 29/4/2021

DOI: https://doi.org/10.33103/uot.ijccce.21.2.2

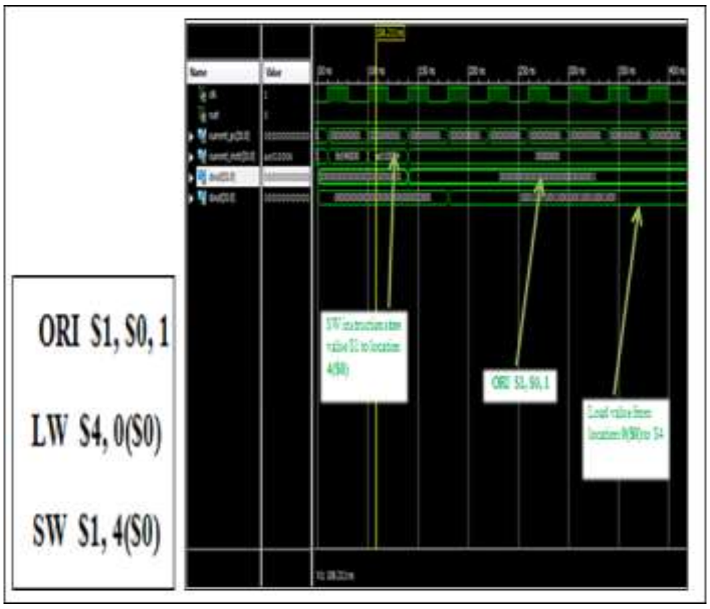


```
MANDI $4, $1, 0x0010
MXORI $5, $1, $2
MORI $6, $0, 0x1000
MANDI $7,$1,$2
MXORI $8,$0,0x1000
```

MANDI $4, $1, 0x0010 execute two core at the same time
MORI $5, $1, $2 execute two core at the same time
MORI $6, $0, 0x1000 execute two core at the same time
MXORI $8,$0,0x1000 execute two core at the same time

FIG.14. TEST CODE AND WAVEFORM RESULTS FOR LOGIC SIMD INSTRUCTIONS

Fig.15 shows a list of the test code for memory instructions and waveform for test code to complete execution of the LW and SW instructions. The instruction LW $4, 0($0) will read value of location zero and store in register $4 while instruction SW $1, 4($0) will write the value of register $1 in location 4 after adding with contents of register $0.



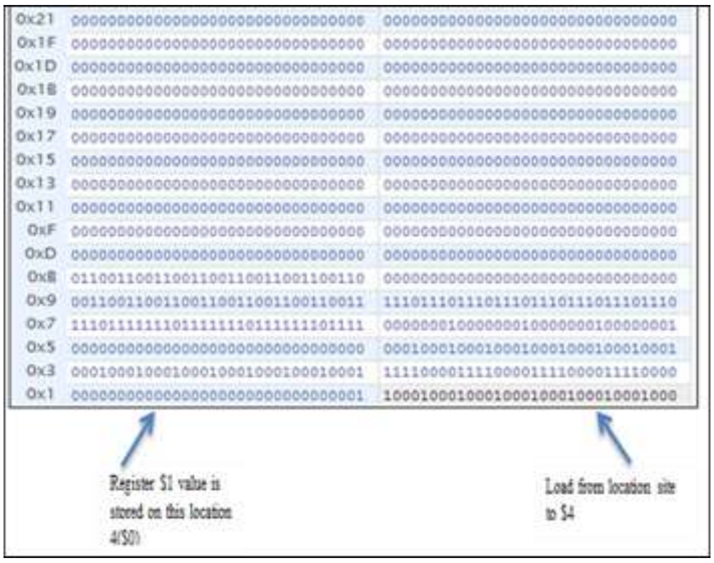
```
ORI $1, $0, 1
LW $4, 0($0)
SW $1, 4($0)
```

SW instruction store value $1 to location 4($0)
ORI $1, $0, 1
Load value from location 0($0) to $4

FIG.15. LIST CODE AND WAVEFORM FOR MEMORY SISD INSTRUCTIONS

Fig.16 shows the contents of data memory after executing test code for memory instruction. The first location in data memory is used to store the results in this test code.

Received 8/12/2020; Accepted 29/4/2021

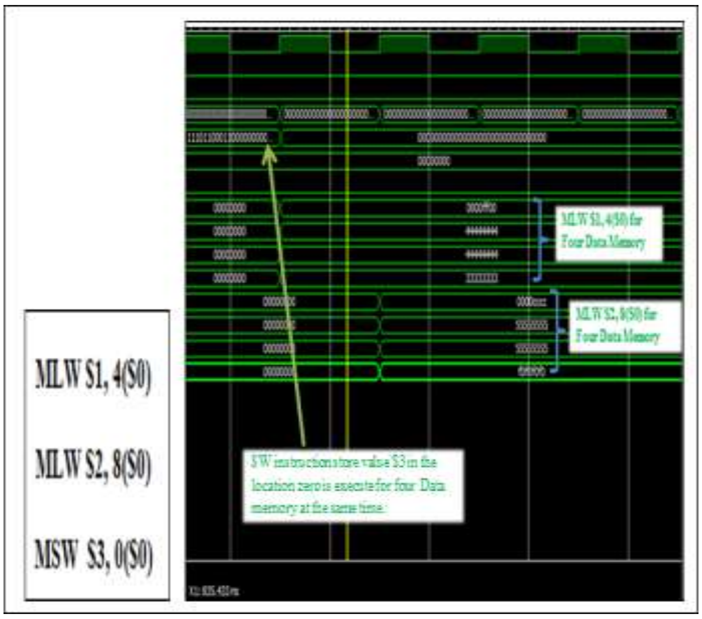


0x21	00000000000000000000000000000000	00000000000000000000000000000000
0x1F	00000000000000000000000000000000	00000000000000000000000000000000
0x1D	00000000000000000000000000000000	00000000000000000000000000000000
0x1B	00000000000000000000000000000000	00000000000000000000000000000000
0x19	00000000000000000000000000000000	00000000000000000000000000000000
0x17	00000000000000000000000000000000	00000000000000000000000000000000
0x15	00000000000000000000000000000000	00000000000000000000000000000000
0x13	00000000000000000000000000000000	00000000000000000000000000000000
0x11	00000000000000000000000000000000	00000000000000000000000000000000
0xF	00000000000000000000000000000000	00000000000000000000000000000000
0xD	00000000000000000000000000000000	00000000000000000000000000000000
0xB	01100110011001100110011001100110	00000000000000000000000000000000
0x9	00110011001100110011001100110011	11101110111011101110111011101110
0x7	11101111111011111110111111101111	00000001000000010000000100000001
0x5	00000000000000000000000000000000	00010001000100010001000100010001
0x3	00010001000100010001000100010001	11110000111100001111000011110000
0x1	00000000000000000000000000000001	10001000100010001000100010001000

Register $1 value is stored on this location 4($0)
Load from location site to $4

FIG.16. DATA MEMORY CONTENTS AFTER EXECUTING LW AND SW INSTRUCTIONS

Fig.17 lists the code for memory SIMD instructions and shows the simulation for the complete execution of the MLW and MSW instructions. In instruction MLW $1, 4($0) is a load value for location four to the register $1 and MSW $3, 0($0) is a store value register$3 in the location zero.



```
MLW $1, 4($0)
MLW $2, 8($0)
MSW $3, 0($0)
```

MLW $1, 4($0) for Four Data Memory
MLW $2, 8($0) for Four Data Memory
SW instructions store value $3 in the location zero is execute for four Data memory at the same time

FIG.17. LIST CODE AND WAVEFORM FOR MEMORY SIMD INSTRUCTIONS

Fig.18 shows the contents of data memory after executing test code for memory instruction. The first location in data memory is used to store the results in this test code.

Received 8/12/2020; Accepted 29/4/2021

DOI: https://doi.org/10.33103/uot.ijccce.21.2.2

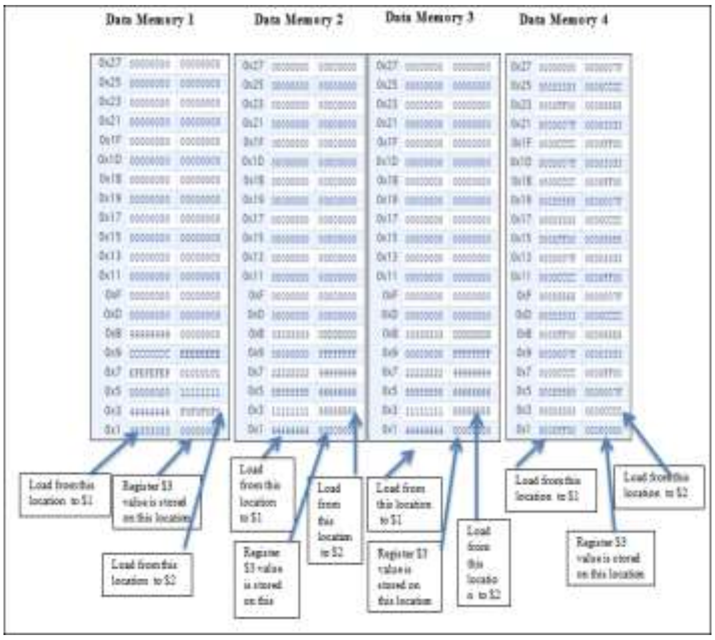

Data Memory 1
Data Memory 2
Data Memory 3
Data Memory 4
Load from this location to $1
Register $3 value is stored on this location
Load from this location to $2
Load from this location to $1
Register $3 value is stored on this
Load from this location to $2
Load from this location to $1
Register $3 value is stored on this location
Load from this location to $2
Load from this location to $1
Load from this location to $2
Register $3 value is stored on this location

FIG.18. DATA MEMORY CONTENTES AFTER EXECUTING MEMORY SIMD INSTRUCTIONS

The total time to execute the SISD instructions is the same as the total time to execute the SIMD instructions, but the difference is the SIMD instructions which four core execute with one clock, while the SISD instructions are executed one core execute with one clock, thus increasing the size of data processed by SIMD instructions128 bit data bus while 32 bit data bus in SISD instructions.

B. Synthesis Result

B.1 RTL Schematic

The schematics Register Transfer Level (RTL) of the processor can be shown after the synthesis is shown in the *Fig. 19.*

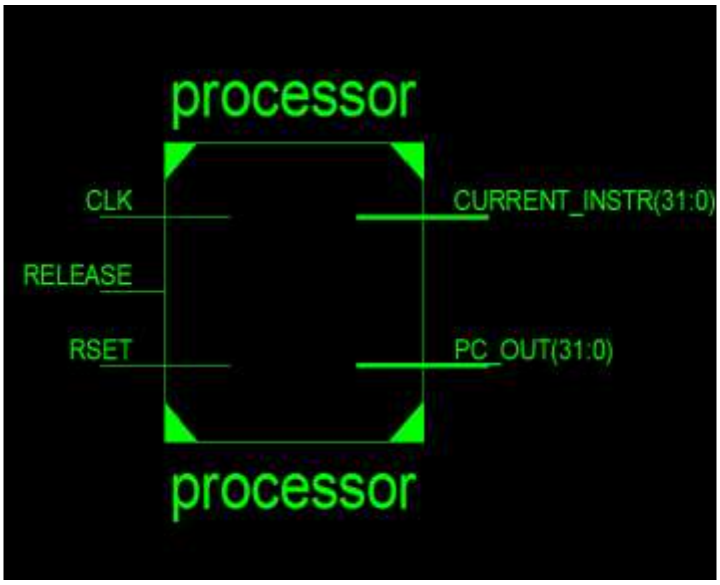


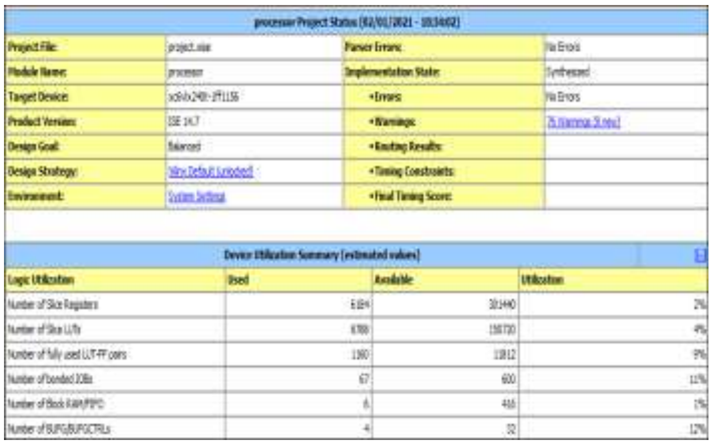
FIG.19. RTL SCHEMATIC FOR PROPOSED MIPS PROCESSOR

Received 8/12/2020; Accepted 29/4/2021

DOI: https://doi.org/10.33103/uot.ijccce.21.2.2

B.2 Device Utilization Summary

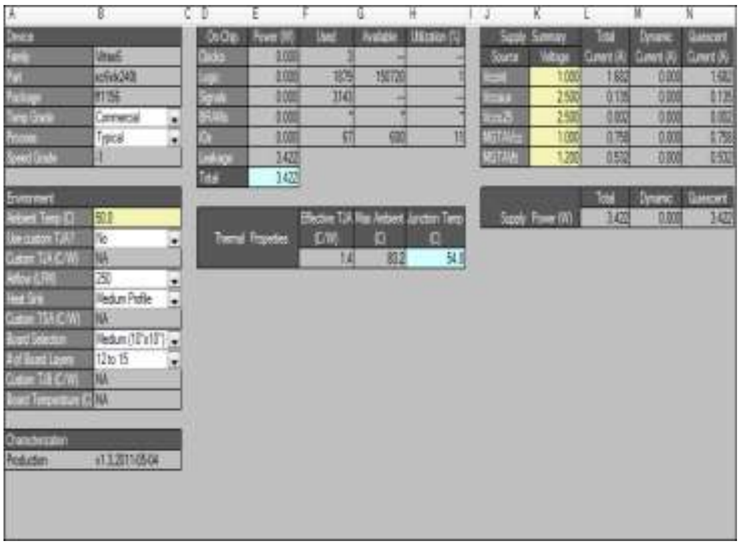
In this research, the proposed MIPS processor is developed by using VHDL language and implemented on Virtex-6 XC6VLX240T-1FFG1156 FPGA. From the proposed model that has pipelined and multi core MIPS processor, the utilization summary as shown in *Fig. 20* for the proposed processor has (6183 number of slice register), (6790 number of slice LUTs), (1162 number of fully used LUT-FF pairs), (67 number of bounded IOBs), (6 number of block RAM/FIFO). The total power analysis of multicore MIPS processor is equal to 3.422 watt and the clock period: 7.329 ns (frequency: 136.444MHz) is obtainned. The summary of power analysis of proposed processor is shown in *Fig. 21*.



processor Project Status (02/01/2021 - 10:34:02)			
Project File:	project.xise	Parser Errors:	No Errors
Module Name:	processor	Implementation State:	Synthesized
Target Device:	xc6vlx240t-1ff1156	•Errors:	No Errors
Product Version:	ISE 14.7	•Warnings:	75 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	6184	301440	2%
Number of Slice LUTs	6788	150720	4%
Number of fully used LUT-FF pairs	1160	11812	9%
Number of bonded IOBs	67	600	11%
Number of Block RAM/FIFO	6	416	1%
Number of BUFG/BUFGCTRLs	4	32	12%

FIG.20. DEVICE UTILIZATION SUMMARY FOR THE PROPOSED PROCESSOR



Device	
Family	Virtex6
Part	xc6vlx240t
Package	ff1156
Temp Grade	Commercial
Process	Typical
Speed Grade	-1

Environment	
Ambient Temp (C)	50.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	250
Heat Sink	Medium Profile
Custom TSA (C/W)	NA
Board Selection	Medium (10"x10")
# of Board Layers	12 to 15
Custom TJB (C/W)	NA
Board Temperature (C)	NA

Characterization	
Production	v1.3,2011-05-04

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.000	3	---	---
Logic	0.000	1875	150720	1
Signals	0.000	3140	---	---
BRAMs	0.000	[illegible]	[illegible]	[illegible]
IOs	0.000	67	600	11
Leakage	3.422			
Total	3.422			

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	1.4	80.2	54.8

Supply Summary Source	Voltage	Total Current (A)	Dynamic Current (A)	Quiescent Current (A)
Vccint	1.000	1.682	0.000	1.682
Vccaux	2.500	0.135	0.000	0.135
Vcco25	2.500	0.002	0.000	0.002
MGTAVcc	1.000	0.758	0.000	0.758
MGTAVtt	1.200	0.502	0.000	0.502

	Total	Dynamic	Quiescent
Supply Power (W)	3.422	0.000	3.422

FIG.21. POWER ANALYSIS TABLE FOR PROPOSED PROCESSOR

VIII. CONCLUTION

A multi core MIPS processor, A 32-bit Multi core MIPS processor, was designed and implemented on Xilinx Virtex-6 FPGA using VHDL. The Xilinx ISE Design Suite 14.7 platform was used to simulate and test a proposed processor with 39 instructions. The multi-core architecture was designed with distributed memory to give the same address to every memory unit and used Harvard memory approach with two separate memories which were designed and implemented (instruction memory and data memory) to improve the performance processor by decreasing the area utilization of processor. The utilization summary of the proposed processor has (6183 number of slice register),

Received 8/12/2020; Accepted 29/4/2021

(6790 number of slice LUTs), (1162 number of fully used LUT-FF pairs), (67 number of bounded IOBs), (6 number of block RAM/FIFO). The total power analysis 3.422 watt of multi-core MIPS processor is obtainned and the clock period is equal to 7.329 ns (frequency: 136.444MHz) for the proposed processor. The big constraint of RISC processor design is the limitation of device utilization summary (number of slice register, number of slice LUTs, number of fully used LUT-FF pairs, number of bounded IOBs, and number of block RAM/FIFO...), when using Spartan-3E.

REFRENCES

[1] L. Wang, J. Tao, G. V. Laszewski, and H. Marten, "Multicores in Cloud Computing: Research Challenges for Applications," Journal of computers, Vol. 5, No. 6, pp. 958-964, 2010.

[2] A. Ashok, and V. Ravi, "ASIC Design of MIPS Based RISC Processor for High Performance", Conference Paper, pp.263-269, 2017.

[3] S. B. Ritpurkar, M. N. Thakare, and G. D. Korde, "Synthesis and Simulation of a 32Bit MIPS RISC Processor using VHDL", IEEE International Conference on Advances in Engineering & Technology Research, 2014.

[4] Al-sudany S. M, Al-Araji A.S., and Saeed B.M.," FPGA based MIPS Pipeline Processor with SIMD Architecture", International Journal of Science and Research (IJSR), Vol.9, No.6.pp 44-450, 2020.

[5] E. Jonathan, "Design and implementation of a Multimedia Extension for a RISC Processor Eduardo", Master Thesis Facultat d'Informàtica de Barcelona (FIB) Universitat Politècnica de Catalunya - BarcelonaTech , 2 July 2015.

[6] P. Bhardwaj, and S. Murugesan," DESIGN & SIMULATION OF A 32-BIT RISC BASED MIPS PROCESSOR USING VERILOG" IJRET: International Journal of Research in Engineering and Technology, Vol. 05, No. 11, pp. 166- 172, 2016.

[7] K. B. Thakor, and M. Shirodkar, "Design of a 16-bit RISC Processor Using VHDL". International Journal of Engineering and Technical Research, Vol.6, No.4, pp.238-244, 2017.

[8] M. R. Rakesh, "Design and Implementation of Dual Core and Quad Core Processor in Vertex 6 FPGA Using Pipelined RISC Architecture". International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering, Vol. 6, No.11, pp.22-28, 2018.

[9] V. Raj, R. Patil, V. Vishwakarma, and PreetiHemnani, "32-BIT PROCESSOR DESIGN on FPGA". JASC: Journal of Applied Science and Computations, Vol. VI, No. IV, pp. 3485- 3490, 2019.

[10] S. Kumar, and B. Bhushan, "Qualitative Analysis of 32-Bit MIPS Pipelined Processor", International Journal of Engineering Research and, Vol. 9, No. 05, PP. 558- 561, 2020.

[11] V. Prasanth, V. Sailaja, P. Sunitha, & B. Vasantha, "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.

[12] D. Ruckmani, N. Srinivas, S. Shashi, D. Ruckmani & H. Byrareddy, "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.

[13] P.S. Kelgaonkar and S. Kodgire, "Design of 32 Bit MIPS RISC Processor Based on Soc", International Journal of Latest Trends in Engineering and Technology, Vol.6, No. 3, pp.446-450, 2016.

[14] A.S. Radhamani, and E. Baburaj," Network Traffic Monitoring and Control for Multi core processors in cloud computing applications", International Journal of Computer Information Systems and Industrial Management Applications, Vol., 5 pp. 557-563, 2013.

[15] Al-sudany S. M, Al-Araji A.S., and Saeed B.M., "Architecture and Advantages of SIMD in Multimedia Applications", Journal of Xi'an University of Architecture & Technology, Vol. XII, No.VI, PP. 1452- 1459, 2020.

[16] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards , C. Ramey , M. Mattina , C. Miao, J.F Brown. and A. Agarwal, "On-Chip Interconnection Architecture of the Tile Processor", Micro, IEEE, Vol. 27, pp.15-31, 2007.

[17] B. Valli, A. U. Kumar , and B. V. Bhaskar , "FPGA Implementation and Functional Verification of a Pipelined MIPS Processor," International Journal Of Computational Engineering Research, vol. 2, no. 5, pp. 1159-1161, 2012.

[18] N. N. Sirhan1 and S. I. Serhan2"MULTI-CORE PROCESSORS: CONCEPTS AND IMPLEMENTATIONS", International Journal of Computer Science & Information Technology (IJCSIT) Vol 10, No 1, pp.1-10, February 2018.

Received 8/12/2020; Accepted 29/4/2021

DOI: https://doi.org/10.33103/uot.ijccce.21.2.2

[19] A. M. John, and S. Varshney, "FPGA Implementation of 32-bit MIPS Processor with CISC Multiplication Operation", International Journal of Engineering Research and Technology (IJERT), Vol.4, No.11, pp. 675-678, 2015.

[20] S. S. Omran, and A. J. Ibada, "FPGA Implementation of MIPS RISC Processor for Educational Purposes". Journal of Babylon University/Pure and Applied Sciences, Vol.24, No.7, PP. 5471 - 5415 2016.

[21] M. T. Kabir, M. T. Bari, and A. L. Haque, "ViSiMIPS: Visual Simulator of MIPS32 Pipelined Processor," in IEEE 6th International Conference on Computer Science & Education (ICCSE), pp. 788-793, 2011.

[22] M. R. Rakesh," Design and Simulation of Four Stage Pipelining Architecture Using the Verilog", International Journal of Science and Research (IJSR), Vol.3, No.3, pp 108-112, 2014.

[23] R. K Akshatha, and H. J. Basavaraj, " NOVEL DESIGN OF DUAL CORE RISC ARCHITECTURE IMPLEMENTATION", International Journal of Advances in Electronics and Computer Science, Vol.2, No.5, pp.31-34, 2015.

[24] K. P. Singh and D. Kumar, "Design of High Performance MIPS Cryptography Processor", Conference: 9th International Conference Heterogeneous Networking for Quality, Reliability, Security and Robustness (Springer LNICST), Vol.115, 2013.

[25] K. Singh," Performance Improvement in MIPS Pipeline Processor based on FPGA". Conference: 3rd International Conference on Emerging Trends of Engineering Science Management and its Applications At: IIC, New Delhi, India,Vol. 4, No. 1, pp. 57-64, 2016.

[26] S. P. Ritpurkar , M. N. Thakare, G. D. Korde, "Review on 32-bit MIPS RISC Processor using VHDL", International Conference on Advances in Engineering & Technology, pp.46-50, 2014.

Received 8/12/2020; Accepted 29/4/2021