DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

Development of Customized MIPS_32 Core Processor for Image Processing Applications

Omar Nowfal Mohammed Taher[1], Mohammed Najm Abdullah[2], Hassan Awheed Jeiad[3]
[1, 2, 3]Computer Engineering Department, University of Technology, Baghdad, Iraq
[1]omar.n.mohammedtaher@uotechnology.edu.iq, [2]mohammed.n.abdullah@uotechnology.edu.iq, [3]hassan.a.jeiad@uotechnology.edu.iq

Abstract— *Definitely, image processing operations without advanced and expensive microprocessors consume more time, power, and larger programs. So, improving the reasonable cost of microprocessors is crucial in this situation. This paper proposes an improvement for the MIPS_32 architecture that is called a Customized MIPS_32 (CMIPS_32) to enhance the capabilities of image processing (IP) operations. The proposal aims to increase throughput by minimizing the iterative fetching of instructions required by a certain IP operation into a single customized IP instruction. The architecture of MIPS_32 was developed in two phases. Firstly, the Register File, control unit, and ALU are modified to manipulate the information related to the IP operations. Secondly, two new units, the address calculation unit and the last pixel detection unit, were proposed to determine a certain image's starting and ending addresses. Furthermore, the MIPS_32 pipeline is customized to have five to six stages depending on the intensity of operation required by a certain IP instruction to decrease the number of machine clocks and the power consumed. The proposal was implemented using the Zed-Board XC7Z020CLG484-1 FPGA. The results showed that the computation speedup increased by a factor equal to the number of standard instructions required to execute the same operation performed by one of the proposed IP instructions. The CMIPS_32 consumed less power than other models that were implemented on Spartan3-XC3S1500L, Virtex5-XC5VFX30T, Virtex6-XC6VLX75T, and Virtex6-Low-Power-XC6VLX75T by 0.0138W, 0.6468W, 1.31W, and 0.7898W, respectively. Comparing the power consumed by the proposal with the GPU proved that the CMIPS_32 consumes less than the NVIDIA-GPU-GTX980 by 63.8698W.*

Index Terms— *FPGA, Image processing instructions, MIPS_32, Verilog.*

I. INTRODUCTION

Modern electronic smart devices have entered many areas of life. Some have become the backbone of many sensitive fields, including health care, science, and education. Most smart electronics devices depend on parallel processing, which supports multi-core system devices. Reduced Instruction Set Architecture (RISC) is an architectural type that uses a fixed size of instructions with many general-purpose registers [1]. The research community widely uses this type of architecture because of its simplicity and development capability [2]. A Microprocessor without Interlock Pipeline Stage (MIPS) is a kind of RISC architecture developed by MIPS technologies. However, the MIPS pipeline, including MIPS_32 and MIPS-64, has five stages: Instruction Fetch (IF), Instruction Decoding (ID), Execute (EX), Memory Access (MA), and Write back (WB) [3, 4]. Field

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq
ISSN (Print) 1811-9212 *ISSN (Online) 2617-3352*

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

Programming Gate Arrays (FPGAs) are a programmable package allowing the developer to implement and develop the processor design. The arrays contain many complex sets of essential logical functions using Hardware Description Language (HDL). Verilog is a kind of (HDL) that is standardized by IEEE 1364 and used to implement electronic systems [5, 14]. This work aims to exploit the energy conservation and reconfigurability introduced by the architecture of MIPS_32 to design a CPU capable of executing dedicated image processing instructions. This work is with improved computation speed and reasonable cost in contrast to GPU or extended vector processors, which are expensive and, in their nature, have a complexity from the side of manufacturing and manipulation. MIPS_32 core was implemented by the authors in [6] using Virtex 7 FPGA and MATLAB HDL coder to achieve optimal resources and high throughput. They optimized the target implementation for performance and resources using the HLS directive. S. H. S. Dizaji et al., in [7], implemented multi-methods of steganography using two approaches of parallelism, one by using a Graphical Processing Unit (GPU) and the other by using Networks On Chips (NOC). They verified that both approaches decreased the execution time compared with the serial approach.A RISC-V 32b processor was designed in [8] to support many additional commands, including hardware loop, save instructions, additional ALU instructions, and downloading transferring and transferring PS data instructions; the authors concentrated on speed and power consumption of the design by using the correct clock method and design low power RISC processor respectively. H. S. Mahmood and S. S. Omran in [9] implemented a dynamic branch predictor using VHDL-FPGA. They used the MIPS processor to implement their work because of its ability to increase prediction accuracy. They combined gshare and biomodel branch techniques by dividing the pattern history table into two branches corresponding to taken and not-taken. The result of using this predictor is to increase the MIPS performance.Ionel Zagan and Vasile Gheorghiţă Găitan in [10] implemented a soft-core processor using FPGA based on different instruction sets to enhance the performance of the real-time system through producing dedicated hardware threads contexed and a real-time operating system.Shobhit Shrivastav and colleagues [11] implemented a MIPS_32 processor with five pipeline stages. They compared their design against a simulation of a processor without pipeline stages regarding timing and power consumption. They found that the MIPS_32 processor operated thrice with less power than the processor without pipeline stages. The authors in [12] used an approximate adder to decrease the delay in the cost of the design area, and this design was implemented for the MIPS_32 processor. As a result, the timing performance was improved by 253.4% compared to the lookahead adder. The MIPS_32 processor was implemented using FPGA-Verilog. Al-Sudany Sarah and co-workers used MIPS_32 with multi-core architecture to implement SIMD multimedia extension architecture and speed up multimedia and DSP applications [13]. They introduced the advantages and disadvantages of using array and vector-processing architecture in SIMD architecture. The authors in [14,15] presented a design and implementation for a multi-core processor using MIPS_32 architecture. They introduced commands based on SIMD architecture, including addition, subtraction, multiplication, and memory read and write instructions. They suggest the proposed design be used in multimedia and big data processing.Daniel Castaino, et al. in [16] compare GPU with CPU performance by exploiting the GPU and CPU with simple and complex functions operation and measuring the performance of each case. The result presented in this work is that the GPU is better than the CPU, especially when working on extensive data and complex operations.P. Indira, M. Kamaraju, and Ved Vyas Dwivedi in [17] implemented a 32-bit MIPS processor based on FPGA fabric with a 6-stage pipeline. This design is compared with others regarding area, power, and frequency. It performs the power consumed equal to 0.129 W and LUT utilization of 421 using a gatting power technique that makes the circuits get turn-off which state of these circuits inactive.Fahad Siddiqui, et al. in [18] used k-mean clustering and traffic sign recognition algorithms

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq
ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

implemented on the 16 cores of a soft processor (IPPRO) to evaluate the processor performance regarding power consumption and area.

II. STANDARD MIPS_32 OVERVIEW

The architecture of MIPS_32 is shown in *Fig. 1*, while the main components of MIPS_32 are as follows [19, 20]:

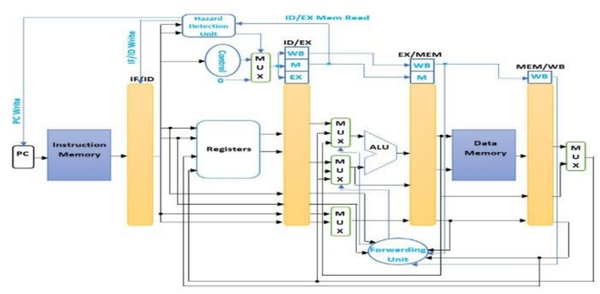


FIG. 1. MIPS_32 ARCHITECTURE [20].

(1) Program Counter (PC): is responsible for giving the instruction memory the address location of the specific instruction. **(2) Instruction Memory and Data Memory**: MIPS_32 design splits the instruction and data memory into two memories, one for instructions and the other for data. **(3) Registers file (RF)**: contains 32 general-purpose registers; each register has 32 bits. **(4) Arithmetic Logic Unit (ALU)**: is used to perform the arithmetic and logical operation. As well as this unit is responsible for calculating the physical address of the data memory. **(5) Pipeline registers**: these registers forward information from one stage to another. **(6) Hazard Detection Unit**: this unit checks any hazards and decides whether to forward or stall the pipeline. MIPS_32 was designed to exploit the pipeline technique, one of the parallel processing techniques in which multiple instructions are executed in sequential and overlapped order [20], [21]. MIPS_32 has three instruction formats shown in *Fig. 2*: **(1) R-Type (Register type instructions)**: This type of instructions format includes all arithmetic and logical instructions. **(2) I-Type (Immediate type instructions** include store, load, and conditional jump instructions. This type of instruction uses two operand registers (*rs* and *rt*) and 16 bits of immediate address. The *rs*, and immediate value is used to calculate the physical address of the memory (base register) in load and store instructions. In contrast, the *rt* is used as a source that contains the register number in RF, which holds the data required to store in memory, or the number in RF that stores the value loaded from memory. On the other side, I-Type format, when used in jump instruction, the fields *rs* and *rt* are the source registers that are compared for equality. **(3) J-Type (Unconditional jump instruction)**: This instruction only needs an opcode value and a 26-bit address [22].

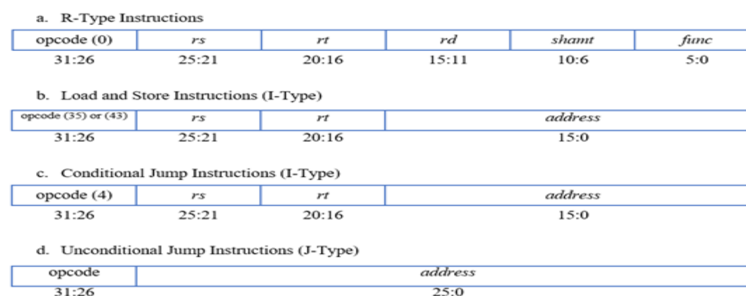


FIG. 2. MIPS_32 INSTRUCTION FORMATS [20].

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq
ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

III. PROPOSED MIPS_32 FOR IMAGE PROCESSING

This section presents the design of the Customized MIPS_32 (CMIPS_32) that highly supports image processing and standard instructions (IPI and SI). Generally, in its standard version, CMIPS_32 was introduced to execute instructions similar to any other 32-bit processor, using a basic Datapath, control unit, and ALU, considering the RISC architecture properties. Image data consist of many identical data units called pixels, and most image processing instructions affect these pixels. Therefore, any image processing instructions need to read all image pixels in sequential order, which means a group of certain instructions must be executed with iterations equal to the number of pixels. In this work, particular instructions are built-up to operate with image pixels brilliantly. Basically, the architecture of the Datapath, control unit, and ALU of the standard MIPS_32 needs to be reformulated to provide the capability for implementing an extra set of instructions related to image processing. However, the modified Datapath, which includes modified RF and the modified CU, will be presented in the next sections.

A. Pipeline Stages of CMIPS_32

CMIPS_32, which enhances the image processing instructions, also uses a pipeline structure with five-to-six stages depending on the operation required by the instruction. Essentially, the significant stages are reformulated in their function to have the ability to execute the IPIs. The CMIPS_32 processor may configure a single execution stage for particular image instructions; hence, the pipeline will have five pipeline stages. In contrast, the same processor may configure two sequential execution units to accomplish other instructions so that it will have six pipeline stages. The motivation for making the number of pipeline stages configurable and extendable according to the complexity of IPI is to keep the design of ALU for any of the execution stages of the pipeline as simple as possible and less the power consumption and to keep the number of clock cycles as less as possible. This method ensures that each of these extended execution units consumes no more than one machine clock. *Fig. 3* shows the conceptual block diagram of the configurable pipeline of the CMIPS_32. The figure illustrates that the basic construction of the pipeline comprises six stages which are Instruction Fetch (IF), Instruction Decode and Register Read (ID), Execution 1 (EX1), Execution 2 (EX2), Memory Access (MA), and Write Back (WR). The jumper line between EX1 and EX2 stages indicates that EX2 may be involved within the other five stages or just jumped without any effect on the other stages, depending on the intensity of the operation of the IPI.

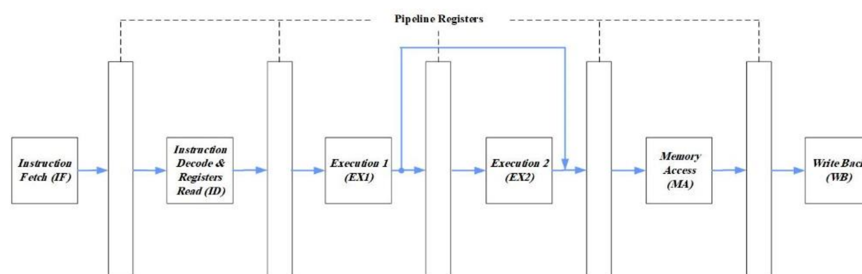


FIG. 3. CONCEPTUAL DIAGRAM FOR PIPELINE STAGES OF THE CMIPS_32.

The detailed architecture for the pipeline stages of the CMIPS_32 is illustrated in *Fig. 4.* This pipeline starts at the IF stage. In IF, the instruction is fetched from the instruction memory because it is well known that the instructions are stored in a separate memory

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

called instruction memory, compared with the data memory that stores the operands and other related data that the instructions may need.The next stage of the pipeline is the ID stage which decodes the instruction type if it is SI or IPI, through the instruction type field and accesses the register file to assign the registers determined in the instruction format. In its core operation, instruction decoding involves setting up the control signals for the rest of the pipeline components. The third stage of the CMIPS_32 pipeline is EX1. This stage has been redesigned to treat both SI and IPI. Addresses Calculation Unit (ACU) is proposed to be added to the EX1 stage of the MIPS_32 pipeline to deal with the IPIs. ACU will be disabled when the instruction is of SI type, and EX1 will return to work as a standard execution unit within the MIPS_32 pipeline. The redesigned EX1 stage has three significant calculation phases when the instruction is decoded as an IPI; the first phase performs the arithmetic operation determined by the IPI using the available ALU-1. The following two phases exploit the proposed ACU to calculate the memory address required to store the output of the first phase and calculate the memory address to read the next pixel of the assigned image stored in block and kernel memories. Stage EX2, the fourth pipeline stage, is activated only when the image processing instruction needs more than one operation, such as gamma correction and edge detection instructions, to obtain the final result for the calculated image pixel. Otherwise, if the instruction can be accomplished by EX1 only, this stage will be passed without any activity and without consuming any machine clock, and the result of the EX1 stage will be forwarded to the MA stage directly. The decision to involve EX2 within the proposed pipeline or not is determined by the CU, which is illustrated in *Fig. 6*.

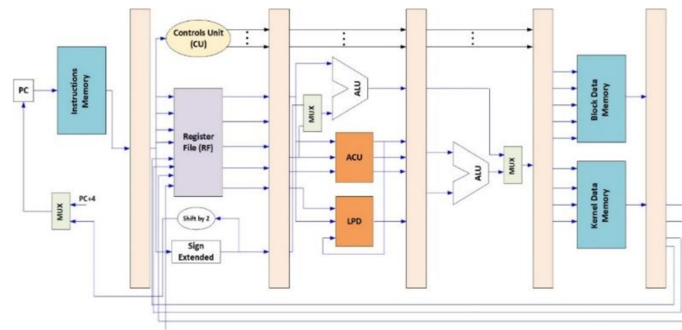


FIG. 4. CMIPS_32 BLOCK DIAGRAM.

The next stage of the proposed CMIPS_32 belongs to the memory access, MA. The function of this stage differs from the standard MIPS_32. For the CMIPS_32, the reading from and writing to the memory is done during the positive and negative clock edges, respectively. The result of ALU-1 or ALU-2 is written to the memory, and the subsequent image pixels are read from the block and kernel memories. However, the memory function of SI is to read new data from memory or write data values to the memory, but not both. Finally, in the last stage of the pipeline of the proposed CMIPS_32 is the write-back, WB. The values that were read from block and kernel memories are returned to the RF to prepare for use in the next iteration of the current IPI. On the other side, either the reading value of the block memory or the resulting value of the ALU-1 is returned to the RF when SI is executed.

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq
ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

B. CMIPS_32 Instruction Formats

The CMIPS_32 is presented with assembly instructions that deal with image processing applications. To realize what is required by such applications, two instruction formats are suggested that have the necessary fields to store the instruction type, operation code, source and destination registers, and image size. The proposed instruction formats for CMIPS_32 consist of 32-bit word lengths and are compatible with instruction word lengths for the standard MIPS_32. *Fig. 5* shows the format of the two instruction words for image processing to execute ten assembly instructions. These instructions represent common image processing operations such as adjusting the brightness of a certain image, ANDing two images, inverting an image, and gamma correction. The details of the ten instructions are explained in subsection E. From the left side, the two instruction formats shown in *Fig. 5* share the same field called instruction type with a size of 6 bits. This field is assigned to decode the type of instruction format to distinguish between IPI and SI formats. If the six bits of this field are all one, the control unit will decode and assign this instruction as IPI; otherwise, it is SI. The second field is the opcode of a 6-bit size that stores the opcode of a specific IPI. The third field, named *rs*, stores the register number that holds the starting address of the source image; however, if the instruction uses two images to be processed or needs an immediate value to accomplish the required operation, the following field, which is named *rt* (*Fig. 5, type a*) can be exploited for that purpose. The rt field is unset or unused when the IPI operates with a single image. The *rd* field (*Fig. 5, type a and b*) stores the register number that holds the starting address of the memory where the resulting image will be stored. The last field (the first one on the right side), named *rz*, is of 5 bits and specifies the register number that holds the number of pixels of the specified image.

FIG. 5. IPI INSTRUCTIONS FORMATS FOR CMIPS_32.

C. Instruction Flow of CMIPS_32

In general, for SI, the standard MIPS_32 executes the instructions in sequential and overlapping order by passing them via the whole five stages of the pipeline. Thus, to perform simple image processing operations such as ANDing or ADDing, the standard MIPS_32 needs to execute a program with a set of SIs that will be iterated for times equal to the number of pixels the image consists of. On the other hand, for the CMIPS_32 structure, a single IPI will be executed, and that IPI will process the image pixels in sequential and overlapped order. Furthermore, the CMIPS_32 will overlap some pipeline stages for the IPI type to process the whole image pixels and leave others. As a result, the number of machine clocks needed to execute a single IPI for a certain image size will be lower than that needed to process the same image by a program containing a set of SIs. *Fig. 6* shows the instruction flow for the CMIPS_32 pipeline executing both SI and IPI types.

Fig. 6 illustrates that the IF stage of the pipeline will be involved only once at the beginning of executing the IPI for fetching instructions. In contrast, ID, EX1, EX2, MA, and WB stages will be involved repeatedly for n times, where n is the number of image pixels required to be processed. Section IV presents an evaluation to calculate the clocks required to execute SIs and IPIs types. To ensure all the image data will be processed and to

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq
ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

Iraqi Journal of Computers, Communications, Control & Systems Engineering (IJCCCE), Vol. 24, No. 1, March 2024

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

avoid any structured hazards, the program counter (PC) remains constant until the last image pixel is processed, then, the PC will be increased by four to fetch the next instruction.

D. CMIPS_32 Datapath

The CMIPS_32 is designed to operate with SI and IPI types. This subsection illustrates the modification of datapath units required to make the standard MIPS_32 work with the proposed image processing instructions with the highest degree of pipeline exploitation. The modified units are RF and CU, and new basic units were inserted into the datapath: the Addresses Calculation Unit (ACU) and the Last Image Pixel Detection Unit (LPDU).

i. Registers File (RF) and Control Unit (CU)

The proposed CMIPS_32 is suggested to contain 32 general-purpose registers, each 32-bit size, and one special register called SR. The data in the SR register is used mainly for IPI, which requires the pipeline to have six stages to accomplish its task. The RF of CMIPS_32 is modified to meet the requirements of both SI and IPI, which need different information to accomplish their tasks. RF outputs supply the ALU_1 operands in SI and IPI by the value of the fields *rs* and *rt* and the starting addresses for ACU and LPDU (fields *rd* and *rz*) in IPI. *Fig. 7* shows the block diagram of RF. The block diagram shows that RF has two 32-bit ports (*Write_data_1* and *Write_data_2*) for data retrieved from the WB stage. That is because the data returned from the WB stage is either read from the block memory or both block and kernel memories in the case of IPI. While for SI, so single port is used for data returned from the block or ALU_1. Table I shows the RF inputs and outputs and their function.

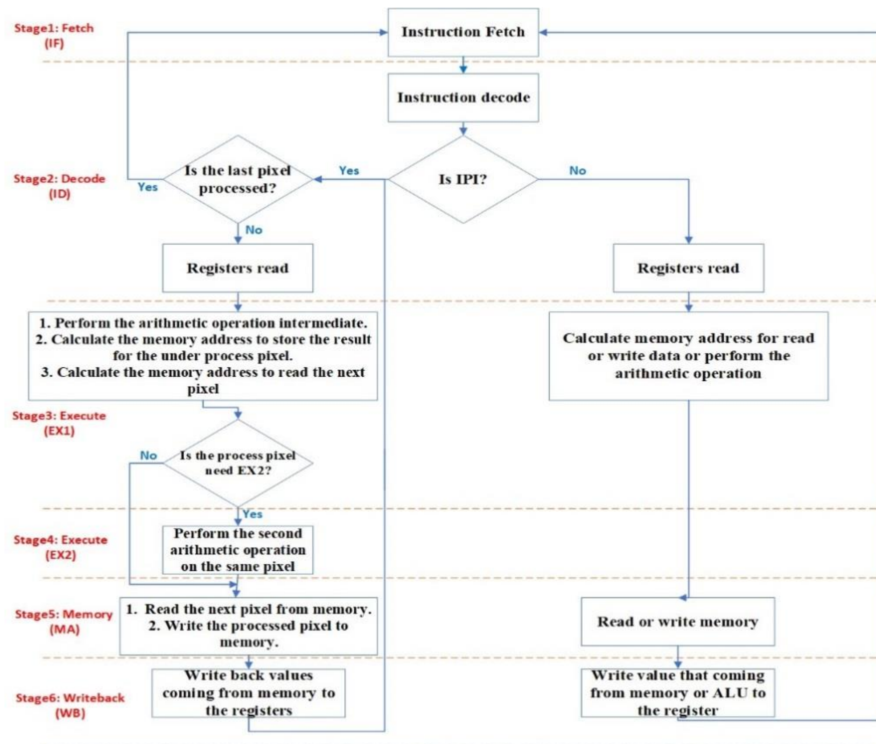


FIG. 6. INSTRUCTION FLOWS FOR THE CMIPS_32.

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq *ISSN (Print) 1811-9212* *ISSN (Online) 2617-3352*

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

The CU generates the multiple lines of signal that control most parts of the datapath of CMIPS_32. The CU unit generates these control signals according to instruction type. The inputs of this unit are the opcodes of SI or IPI and a 1-bit signal named *Reset&update*. The latter acts as a reset signal that re-initializes the whole CU output to decode the next instruction. The output signals of the CU are illustrated in *Fig. 8.* It is worth noting that signals *Lo* and *St* regulate the work of the ACU, which is responsible for calculating the addresses of reading and writing of the memories in IPI. The functions of the rest of the control unit output signals and their function are illustrated in Table II.

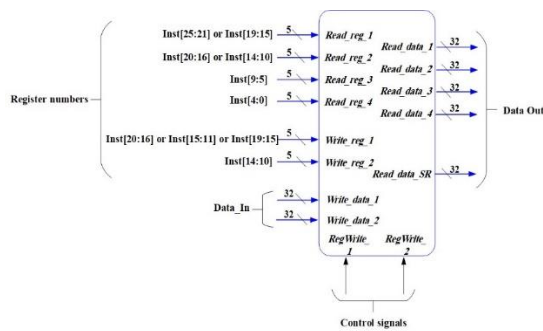


FIG. 7. BLOCK DIAGRAM OF THE RF.

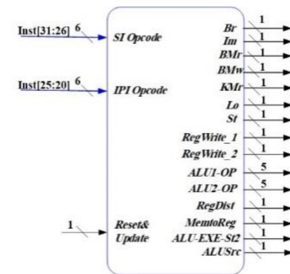


FIG. 8. BLOCK DIAGRAM OF THE CU.

TABLE I. RF INPUTS & OUTPUTS AND THEIR FUNCTION

Control or Data	State	#Bits	Function
Read_Reg_1, Read_Reg_2	input	5	Specify the registers containing the value of *rs* and *rt* in SI and IPI.
Read_Reg_3, Read_Reg_4	input	5	Specify the registers containing the information of *rd* and *rz* in IPI.
Write_Data_1, Write_Data_2	input	32	Data retrieved by the WB stage.
Writr_Reg_1, Write_Reg_2	input	5	Specify the register number that stores the data retrieved.
Reg_Write_1, Reg_Write_2	input	1	Enabling the data written to the specified registers.
Read_Data_1, Read_Data_2	output	32	Supply *rs* and *rt* values to the two inputs of ALU_1 in SI and IPI. Also, supply the starting address of the image read from block and kernel memories to the ACU in IPI.
Read_Data_3	output	32	Supply the starting address (*rd*) to store the operation result of the IPI to ACU.
Read_Data_4	output	32	Supply the size of the image under the process to the LPDU.
Read_Data_SR	output	32	Supply the value of the second operand of ALU_2 in the EX2 stage.

TABLE II. CU OUTPUT SIGNALS AND THEIR FUNCTION

CU Output Signals	#Bits	Function
Br	1	Branch indicator.
Im	1	IPI indicator.
Mr, Mw	1	Enabling read-from and write to block memory.
Kr	1	Enabling read from kernel memory.
Lo, St	1	Signals operate with ACU and LPDU for regular operations.
ALU1-OP, ALU2-OP	5	ALU-1 and ALU-2 operation codes.
RegWrite_1, RegWrite_2	1	Enabling data to be written to the RF.
MemtoReg	1	Activated at load instruction to write the memory output to the RF.
ALU_EXE_St2	1	Enabling the *EX2* pipeline stage.
ALUscr	1	Specify the second ALU-1 input from the RF register or the Sign-Extended unit.
RegDist	1	Specify the *rd* of SI type either from inst[20:16] or inst[15:11]

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq
ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

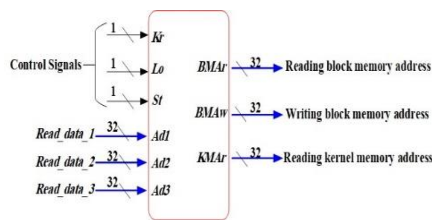
ii. Addresses Calculation Unit (ACU)

This unit calculates the addresses to write to and read from block and kernel memories instead of ALU-1 as in a standard MIPS_32. The RF unit supplies three addresses: the two beginning addresses to read from block and kernel memories (*Ad1* and *Ad2*) and the beginning address to store the data in the block memory (*Ad3*). The ACU contains three temporary registers, A, B, and C, which store the last three values of the calculated memory addresses. These values are later used to determine the addresses of the next iteration to process new pixel(s). The block and circuit diagram of ACU is shown in *Fig. 9*. The control signals *Lo*, *St*, and *Kr* organize the three 32-bit outputs, *BMAr*, *BMAw*, and *KMAr*, of this unit according to the following expressions:

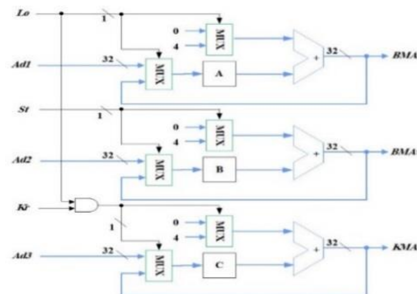
$$BMAr = \begin{cases} Ad1 = A & When\ Lo = 0 \\ A + 4 & When\ Lo = 1 \end{cases} \quad (1)$$

$$BMAw = \begin{cases} Ad2 = B & When\ St = 0 \\ B + 4 & When\ St = 1 \end{cases} \quad (2)$$

$$KMAr = \begin{cases} Ad3 = C & When\ Lo\ or\ Kr = 0 \\ C + 4 & When\ Lo\ and\ Kr = 1 \end{cases} \quad (3)$$



(a) ACU BLOCK DIAGRAM

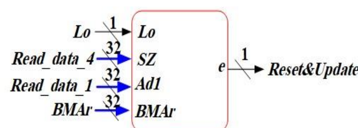


(b) ACU CIRCUIT DIAGRAM

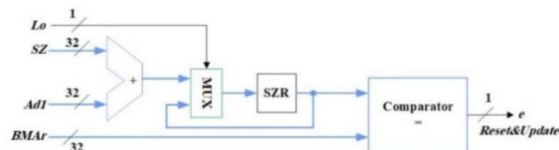
FIG. 9. ACU BLOCK AND CIRCUIT DIAGRAMS FOR CMIPS_32.

iii. Last Pixel Detection Unit (LPDU)

Image size is defined as the number of pixels in the image. The register number containing the size of the is stored in the 5-bit *rz* field of the proposed format instructions for image processing, shown in *Fig. 5*. The LPDU uses *rz* and *BMAr*, which is the current address of the read operation of block memory supplied by ACU, to detect the last pixel of the image under process. Afterward, this unit sends logic 1 to the CU through *Reset&Update* to finish executing the current IPI and fetching the next instruction. The circuit diagram of the LPDU is illustrated in *Fig. 10*.



(a) LPDU BLOCK DIAGRAM



(b) LPDU CIRCUIT DIAGRAM

FIG. 10. LPDU BLOCK AND CIRCUIT DIAGRAMS FOR CMIPS_32.

At the first time and when the Lo signal is equal to zero, the LPDU accomplishes operations. Firstly, adds the address of the first pixel stored in block memory, *Ad1*, with the value image size, *SZ*, to determine the last address of the image. Secondly, compares the 32-bit current address, *BMAr*, with

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq
ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

the result of the addition process done by the first operation stored in the SZR register. Conversely, for the *Lo* signal being 1, the LPDU compares the contents of the SZR register with *BMAr* only. After that, the output of LPDU will be set to 1 when the *BMAr* address equals the SZR register value that indicates the end of the currently processed image.

E. Proposed Instruction Set for Image Processing

This work presents ten instructions for image processing that deal with image enhancement and edge detection. Table III shows these instructions with a description for each.

TABLE III. INSTRUCTION SET FOR IMAGE PROCESSING

No	Instruction	Description	Operation
1	IncBri *rs*, *rt*, *rd*, *rz*	[R(*rd*)] =[R(*rs*)]+R(*rt*)	Increase the pixel's brightness addressed by *rs* with *rt* value and store the result in the *rd* address.
2	DecBri *rs*, *rt*, *rd*, *rz*	[R(*rd*)] =[R(*rs*)]-R(*rt*)	Decrease the pixel's brightness addressed by *rs* with *rt* value and store the result in the *rd* address.
3	AndIm *rs*, *rt*, *rd*, *rz*	[R(*rd*)] =[R(*rs*)]&[R(*rt*)]	And two pixels addressed by *rs* and *rt* and store the result in the *rd* address.
4	AddIm *rs*, *rt*, *rd*, *rz*	[R(*rd*)] =[R(*rs*)]+[R(*rt*)]	Add two pixels addressed by *rs* and *rt* and store the result in the *rd* address.
5	SubIm *rs*, *rt*, *rd*, *rz*	[R(*rd*)] =[R(*rs*)]-[R(*rt*)]	Subtract two pixels addressed by *rs* and *rt* and store the result in the *rd* address.
6	ThIm1 *rs*, *rt*, *rd*, *rz*	[R(*rd*)] =255 … if [R(*rs*)] ≥ R(*rt*). Else [R(*rd*)] =0	Convert grayscale to binary for pixel addressed by *rs*, and store the result in the *rd* address.
7	NagIm *rs*, *rd*, *rz*	[R(*rd*)] =255-[R(*rs*)]	Convert grayscale to negative for pixel addressed by *rs*, and store the result in the *rd* address.
8	ThIm2 *rs*, *rt*, *rd*, *rz*	[R(*rd*)] =[R(*rs*)] … if [R(*rs*)] ≥ R(*rt*) Else [R(*rd*)] =0	Threshold function by value *rt* to pixel addressed by *rs*, and store the result in the *rd* address.
9	GamIm *rs*, *rt*, *rd*, *rz*	[R(*rd*)] =[R(*rt*)]*[R(*rs*)]^R(*SR*)	Gamma Correction of pixel addressed by *rs* for Gamma factor *SR*, and store the result in the *rd* address.
10	EdgIm *rs*, *rt*, *rd*, *rz*	[R(*rd*)] =[R(*rs*)] … if |[R(*rs*)]-[R(*rt*)]| ≥ R(*SR*) Else [R(*rd*)] =0	Edge detection of pixel addressed by *rs* with threshold *rt*, and store the result in the *rd* address.

IV. IMPLEMENTATION, SIMULATION RESULTS, AND COMPARATIVE ANALYSIS

A. Implementation and Simulation Results

This section presents the implementation and simulation results for the proposed CMIPS_32. CMIPS_32 and the standard MIPS_32 from the sides of resource utilization and power consumption have also been compared. CMIPS_32 is implemented using Verilog language in Xilinx ZedBoard XC7Z020CLG484-1 FPGA and MATLAB HDL. The implemented model of CMIPS_32 is shown in *Fig. 11*. The CMIPS_32 model comprises six stages: IF, ID, EX1, EX2, MA, and WB. The modified RF and CU, as well as the proposed ACU and LPDU, are shown in *Fig. 11*.

Fig. 12 shows the simulation results in the waveform for executing SI and IPI-type. In *Fig. 12-a*, the following SI of R-type instruction is executed: ***Add $R1, $R2, $R3***, with values of *R1*=70, *R2*=20.

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

The register number of *rs*, *rt*, and *rd* are (00000b), (00001b), and (00010b), respectively, So *rs = R1*, *rt = R2*, and *rd = R3*=90. The image instruction indicator (*Im*) and control signals (*Lo* and *St*) are zero.

Fig. 12-b illustrates the simulation waveform of the following IPI: ***AndIm $R1, $R2, $R3, $R4***, which requires five pipeline stages. First, registers *R1*, *R2*, *R3*, and *R4* values are set to 10, 10, 3500, and 3025, respectively. This instruction reads two sets of pixels from the block and kernel memories which leads to setting the memories read signal (*Mr*, *Kr*) to 1, and the ANDing operator is performed between them. The result of this operator is stored in the block memory with the starting address *rd*, which means the (*Mw*) signal is set to 1. However, the registers *rs*=(00000b), *rt*=(00001b), *rd*=(00010b), and *rz*=(00100b), which means *rs=R1*, *rt=R2*, *rd=R3*, and *rz=R4*.

Fig. 12-c shows the IPI-type simulation waveform result, which calculates the edge detection between two images. This IPI is complete with two execution stages of the pipeline (EX1 and EX2), meaning the *ALU-EXE-2* signal is set to 1, and the ALU-2 operator does not equal zero. That means two different operators will be applied to complete the execution of this instruction on the same image pixels.

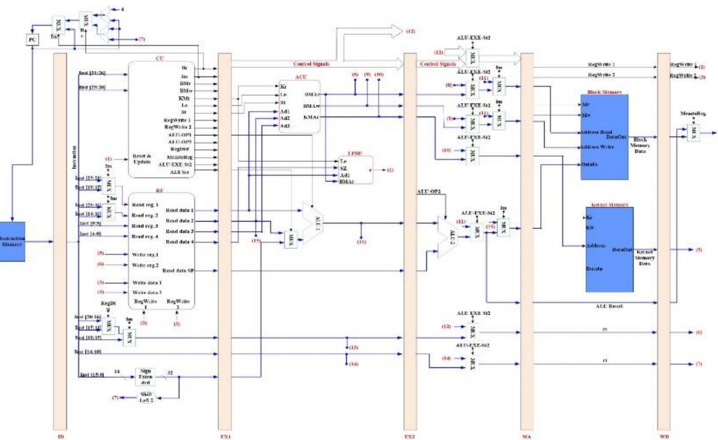
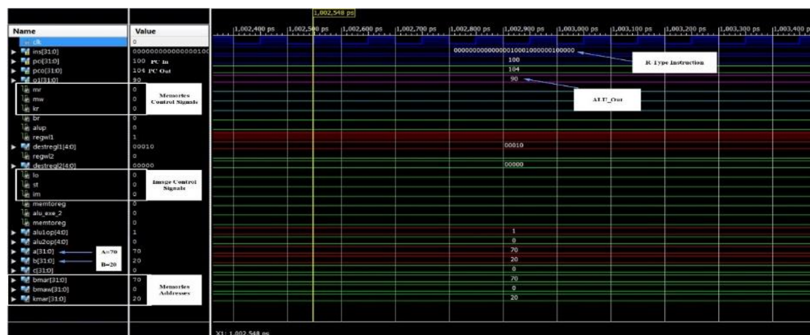


FIG. 11. CMIPS_32 IMPLEMENTATION DIAGRAM.

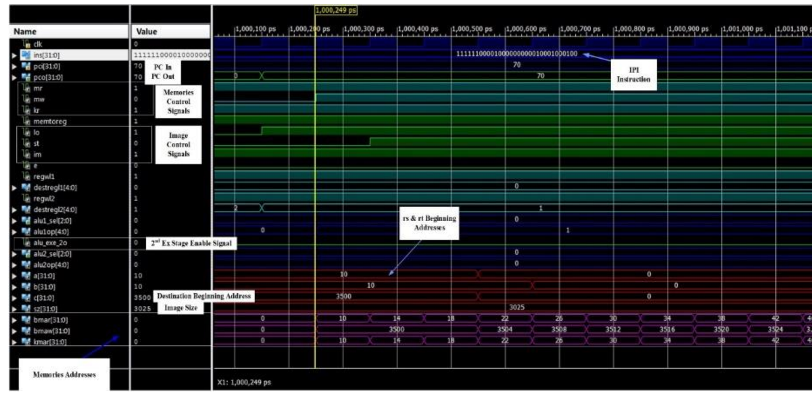
This work uses Matlab tools to read and resize the image (Lena.jpeg) to 55x55 pixels and convert it to a hexadecimal file. The hexadecimal file is loaded to the memory of the proposal as initial data. *Fig. 13-a* shows the original image, while *Fig. 13-b* and *Fig. 13-c* displays the binary and negative versions of the image, respectively, created using instructions 6 and 7 from Table III.



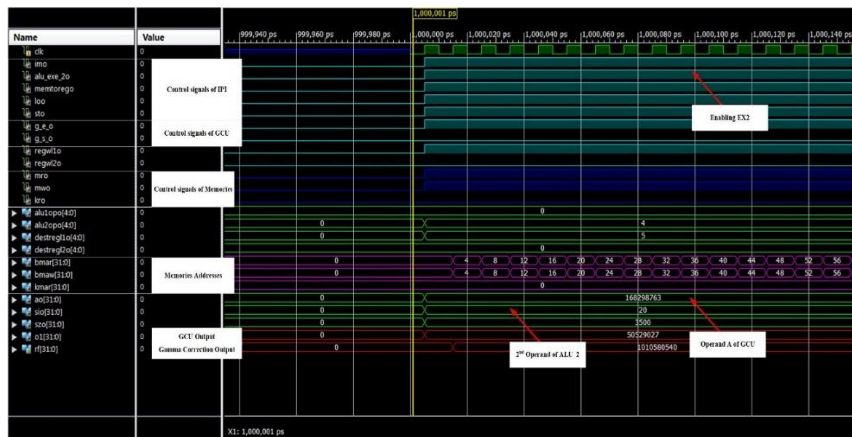
(A) SI OF R-TYPE INSTRUCTION WAVEFORM.

Received 11/July/2023; Accepted 15/August/2023

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3



(B) IPI-TYPE INSTRUCTION WITH FIVE STAGES PIPELINE WAVEFORM.



(C) IPI-TYPE INSTRUCTION WITH SIX STAGES PIPELINE.

FIG. 12. WAVEFORM FOR THE SIMULATION OF CMIPS_32.

(A) ORIGINAL IMAGE

(B) BINARY IMAGE

(C) NEGATIVE IMAGE

FIG. 13. TEST IMAGE AFTER EXECUTING BINARY AND NEGATIVE INSTRUCTIONS.

B. Comparative Analysis and Evaluation

Table IV show the utilization summary and power consumed by the CMIPS_32. Table V. compares CMIPS_32 with standard MIPS_32 implemented with different FPGA platforms regarding resource utilization and power consumption. Table V shows that the power consumption of the proposal is less than the major of other types of the standard MIPS_32 implemented with different FPGA platforms. The power improvement obtained was 9.58% compared with the nearest model one, which consumes 0.144 W even though one model consumes less power than the proposed one. The significant reasons behind the other work consuming high power compared with CMIPS_32 are: firstly, the LUTs and FFs used in other works are greater than those used in CMIPS_32. Secondly, The static power of Xilinx 7, 6, and 5 series is greater than that of ZedBoard XC7Z020CLG484.

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

TABLE IV. RESOURCE UTILIZATION AND POWER CONSUMPTION OF THE CMIPS_32

LUTs	FFs	BRAMs	I/Os	Dynamic Power (W)	Static Power (W)	Total Power (W)
75	414	9	33	0.0242	0.106	0.1302

TABLE V. COMPARE THE CMIPS_32 WITH THE STANDARD MIPS_32 IMPLEMENTED WITH DIFFERENT FPGA PLATFORMS

FPGA Platforms	LUT	Frequency (MHz)	Power (W)	Power enhancement %
MIPS_32 (Spartan3: XC3S1500L) [17]	417	98.090	0.144	9.58
MIPS_32 (Virtex5: XC5VFX30T) [17]	300	321.048	0.777	83.24
MIPS_32 (Virtex6: XC6VLX75T) [17]	307	401.881	1.440	90.96
MIPS_32 (Virtex6Low Power: XC6VLX75TL) [17]	307	335.233	0.920	85.85
MIPS_32 (Spartan3: XC3 1600e-FG484) [17]	421	285.583	0.129	-0.9
Proposed CMIPS_32 (ZedBoard XC7Z020CLG484)	75	100	0.1302	--------

Furthermore, CMIPS_32 is compared with other models, IPPRO, GPU, and eGPU, from the side of the power consumption. Table VI lists the result of that comparison. The results showed that the power consumption by the proposal improved by 58% compared with IPPRO and by 91.65% compared with eGPU.

TABLE VI. COMPARE THE CMIPS_32 WITH THE MODIFIED MIPS_32 AND GPU

Processor	Frequency (MHz)	Static Power (W)	Dynamic Power (W)	Total Power (W)	Power enhancement %
IPPRO [18]	337	0.15	0.03	0.31	58
GPU [18]	1127	37	27	64	99.80
eGPU [18]	600	0.12	----	1.56	91.65
CMIPS_32	100	0.106	0.0242	0.1302	--------

Speedup is an important factor in measuring computer performance. In general, the archived speedup of computer (A) that executes (P) instruction in contrast to computer (B), which executes the same (P) instructions, can be calculated as the following equation (4).

$$Achieved\ Speedup = \frac{time\ required\ to\ execute\ P\ instructions\ using\ computer\ A}{time\ required\ to\ execute\ P\ instructions\ using\ computer\ B} \quad (4)[22]$$

CMIPS_32 performance is evaluated in contrast to the standard MIPS_32 by picking up one of the proposed IPIs listed in Table III to be executed on a particular image(s) with a determined number of pixels, then suggesting a sequence of instructions of SI type to perform the same operation on the same image(s). First, the number of machine clocks required by both cases is calculated, and then, using equation (5), the obtained speedup ratio between CMIPS_32 and MIPS_32 is determined.

$$Achieved\ Speedup = \frac{No.of\ clocks\ required\ by\ program\ of\ SI\ type}{No.of\ clocks\ required\ by\ the\ IPI} \quad (5)$$

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq
ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

For instance, the instruction ***AddIm $R1,$R2,$R3,$R4*** of type IPI is selected to calculate the achieved speedup when it is used. This instruction performs an ADDing operation for two images with the beginning address stored in *R1* and *R2*, respectively. The code in *U* is a sequence of instructions of SI type, which perform the same operation that AddIm performs. This work assumes that completing each ALU operation and pipeline stage consumes only one clock and assumes an image size of 3028 pixels (55*55), where each pixel equals one byte. Then, using the standard MIPS_32 architecture with a pipeline of five stages, the number of clocks required to execute the first instruction shown in *Fig. 14* is five. The first instruction is executed only once. After that, one instruction per clock is executed for the rest of the instructions (six SI-type instructions). It is important to notice that these six instructions are looped for times equal to the number of pixels divided by four since four pixels will be read from the memories at every clock. So, the number of clocks required to execute the six instructions is 6*757 + 5 = 4547. The number of clocks needed to execute the instruction ***AddIm*** is equal to the number of pixels of the image divided by four, which equals 757. So, the achieved speedup through using AddIm instruction (IPI type) instead of the sequence of SI instructions shown in *Fig. 14* via equation (5) is:

```
1:  MOV $R6,00          // Set the counter (R6) to zero
2:  LW $R1,00($R6)      // Load R1by the first 4 pixels from the first address of the base image.
3:  LW $R2,00($R6)      // Load R2 by the first 4 pixels from the first address of the mask image.
4:  ADD $R3,$R1,$R2     // Performs the adding operation between R1 and R2, then stores the result in R3
5:  SW $R3,7000($R6)    // Store the results in memory with starting address 7000.
6:  ADD $R6,04          // Set the counter to process the next 4 pixels.
7:  BNE $R6,$R5,2       // Process all pixels for the two images.
```

FIG. 14. SI INSTRUCTION SET TO PERFORM ADDING BETWEEN TWO IMAGES.

Achieved Speedup using CMIPS_32 = 4546 / 762 = 6 which is the same number of instructions executed during each code loop shown in *Fig. 14*. Essentially, the main reason behind improving the speedup for CMIP_32 in contrast to the standard MIPS_32 is the large number of SIs that must be executed to perform a certain image processing operation compared with a single instruction of IPI type for the same operation. As a general formula, For code consists of j of SI instructions within a loop size of i, where i represents the number of image pixels divided by 4, then, the number of clocks required to execute j * i instructions of SI type will be n (n = j*i) + k representing the number of pipeline stages for CMIPS_32. Conversely, the number of clocks to execute a single IPI on an image consists of i pixels using CMIPS_32 comprising a pipeline of k stages is i/4 + k. By assuming that k might be neglected since j and i >> k as the number of SIs increases when performing more complicated image processing operations or the image size becomes large, then the maximum speedup achieved by CMIPS-32 is: Maximum speedup for CMIPS_32 = (j*i/4)/(i/4) <= j. That means that as the number of instructions of SI type within the main loop becomes high to represent certain image processing operations, the achieved speedup also becomes high. Moreover, the value of k, the number of stages of the CMIPS_32 pipeline, does not significantly affect the achieved speedup. Still, any growth of k will be reflected in resource utilization.

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

C. Discussion Result

Central processing units (CPUs) and Graphics processing units (GPUs) are the processors often most used for image processing. GPUs are characterized by high speed, high power consumption, and high cost compared to CPUs. So, it is very attractive to improve the performance of the available processors characterized by low power consumption and reasonable cost, such as MIPS_32, to operate in specific situations requiring low power consumption and reasonable computational power. CMIPS 32, which improves image processing instructions, employs a pipeline structure with five to six stages, depending on the operation required by the instruction. The CMIPS 32 processor can configure a single execution stage for some image instructions; thus, the pipeline will have five stages. In contrast, the same processor can configure two sequential execution units to execute additional operations, resulting in six pipeline stages. The obtained results showed the performance of the proposed CMIPS_32 model is better than the other MIPS_32-based models that were suggested for the same purposes regarding power consumption and resource utilization. Improving the computation capabilities for the two execution stages of the pipeline, EX1 and EX2, to perform more complicated IPIs can be recommended as future work.

V. CONCLUSIONS

This paper presents an enhancement to the MIPS 32 architecture to allow more effective image processing (IP) tasks. This work proposes a design for a Customized MIPS_32 (CMIPS_32) capable of executing image processing and standard assembly instructions to increase throughput by compressing the number of iteratively fetched instructions needed for a specific IP operation into a single specialized IP instruction. The development of MIPS_32 architecture is built in two phases. The IP operations data is first manipulated by adjusting the Register File, control unit, and ALU. Second, a couple of new pieces of hardware, an address calculation unit and a last pixel detection unit, were proposed for finding the beginning and ending addresses of the image under process, respectively. Also, the pipeline of MIPS_32 was reconfigured to have five to six stages depending on the complexity of the arithmetic operation required by specific image processing instructions to reduce the number of clocks and power consumption. CMIPS_32 was implemented using Zed-Board XC7Z020CLG484-1 FPGA. An evaluation is done by measuring the number of machine clocks consumed by executing a specific image processing operation using CMIPS_32 compared to that consumed by the standard MIPS_32. The results show that the computation speedup for CMIPS_32 is increased by a factor equal to the number of standard instructions of MIPS_32 required to perform the same operation. Furthermore, CMIPS_32 consumed less power than the standard MIPS_32 implemented on Spartan3-XC3S1500L, Virtex5-XC5VFX30T, Virtex6-XC6VLX75T, Virtex6-Low-Power-XC6VLX75TL by 9.58%, 83.24%, 90.96%, and 85.85%, respectively. In addition, CMIPS_32 is better than the NVIDIA-GPU-GTX980 in terms of power consumption by 99.80%.

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq

DOI: https://doi.org/10.33103/uot.ijccce.24.1.3

REFERENCES

[1] Yu L., Kejiang Y., and Cheng-Zhong X, “Performance Evaluation of Various RISC Processor Systems: A Case Study on ARM, MIPS and RISC-V,” in *IEEE International Conference on Cloud Computing*, Volume 12989, 2021 pp. 61-74.

[2] C. Salazar and M. Bobby Birrer, "Instrumentation and Extension of reduced, simulated Single Cycle MIPS architecture to improve Student Comprehension," in *IEEE Frontiers in Education Conference (FIE)*, Uppsala, Sweden, 2020.

[3] A. Ashok and V. Ravi, "ASIC design of MIPS based RISC processor for high performance," in *International Conference on Nextgen Electronic Technologies: Silicon to Software (ICNETS2)*, 2017, pp. 263-269.

[4] P. V. Bharath and B. SanthiBhushan, "MIPS Based 32-Bit RISC Processor with Thermal Management Unit and Flexible Pipelining Structure," in *IEEE 9th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, 2022, pp. 1-6.

[5] Swaminathan T, Vaishnav Rengan V, and Ramesh SR, “Calculator Interface Design in Verilog HDL Using MIPS32 Microprocessor,” in *Conference: 2022 International Conference on Wireless Communications Signal Processing and Networking (WiSPNET)*, 2022.

[6] Sikka, P., Asati, A.R., and Shekhar, C, “Low-Area, High-Throughput Field-Programmable Gate Array Implementation of Microprocessor Without Interlocked Pipeline Stages,” *Lecture Notes in Electrical Engineering*, vol 777. Springer, pp. 647-656, 2022.

[7] Seyed Dizaji, S.H., Zolfy Lighvan, M., and Sadeghi, A., “Hardware-Based Parallelism Scheme for Image Steganography Speed up,” in *International Conference on Innovative Computing and Communications. Advances in Intelligent Systems and Computing*, Volume 1166, 2021, pp. 225-236.

[8] D.V. Soundari, M.K. Shanker Ganesh, Indira Raman, and R. Karthick, “Enhancing network-on-chip performance by 32-bit RISC processor based on power and area efficiency,” *Materials Today: Proceedings*, Volume 45, Part 2, pp. 2713-2720, 2021.

[9] Hadeel Sh. Mahmood and Safaa S. Omran, “Selective branch prediction schemes based on FPGA MIPS processor for educational purposes,” *IOP Conference Series: Materials Science and Engineering*, Volume 518, 2019.

[10] Zagan, Ionel & Găitan, and Vasile, “Soft-core processor integration based on different instruction set architectures and field programmable gate array custom datapath implementation,” *PeerJ Computer Science*, 2023.

[11] Shobhit Shrivastav, Shubham Kumar, Sarthak Gupta, and Bharat Bhushan, “Qualitative Analysis of 32 Bit MIPS Pipelined Processor,” *International Journal Of Engineering Research & Technology (IJERT)*, Volume 9, 2020.

[12] Amir E. Oghostinos, Kareem Moussa, Amr Elnaggar, Alaa AbdAlrhman, and Ahmed Soltan, “Single-Cycle MIPS Processor based on Configurable Approximate Adder,” *International Conference on Modern Circuits and Systems Technologies (MOCAST)*, 2022, pp. 1-4.

[13] Al-sudany Sarah, Al-Araji Ahmed, and Mohammed Saeed Bassam, “Architecture and Advantages of SIMD in Multimedia Applications,” *Xi'an Jianzhu Keji Daxue Xuebao/Journal of Xi'an University of Architecture & Technology*, 2020

[14] Sirhan N.N., “Multi-Core Processors: Concepts and Implementations,” *IJCSIT*, Volume 10, pp.1, 2020.

[15] Al-sudany Sarah, Al-Araji Ahmed, and Mohammed Saeed Bassam, “FPGA based MIPS Pipeline Processor with SIMD Architecture,” *International Journal of Science and Research (IJSR)*, Volume 9, pp. 444-450, 2020.

[16] Daniel Castaño-Díez, Dominik Moser, Andreas Schoenegger, Sabine Pruggnaller, and Achilleas S. Frangakis, “Performance evaluation of image processing algorithms on the GPU,” *Journal of Structural Biology*, Volume 164, pp. 153-160, 2008.

[17] P. Indira, M. Kamaraju, and Ved Vyas Dwivedi, “Design And Analysis Of A 32-Bit Pipelined Mips Risc Processor,” *International Journal of VLSI design & Communication Systems (VLSICS)*, Volume 10, 2019.

[18] Fahad Siddiqui, Sam Amiri, Umar Ibrahim Minhas, Tiantai Deng, Roger Woods, Karen Rafferty, and Daniel Crookes, “FPGA-Based Processor Acceleration for Image Processing Applications,” *Journal of Imaging*, Volume 5, Issue 1, https://www.mdpi.com/2313-433X/5/1/16#, 2019.

[19] Al-sudany Sarah, Al-Araji Ahmed, and Mohammed Saeed Bassam, “FPGA-Based Multi-Core MIPS Processor Design,’ *Iraqi Journal of Computer, Communication, Control and System Engineering*, Volume 21, pp 16-35, 2021.

[20] G. K. Dewangan, G. Prasad and B. C. Mandi, "Design and Implementation of 32 bit MIPS based RISC Processor,", *International Conference on Signal Processing and Integrated Networks (SPIN)*, 2021, pp. 998-1002.

[21] Lakshmi S.S and Chandrasekhar N.S, “FPGA Implementation of A Pipelined MIPS Soft Core Processor,” *International Journal of Innovative Research in Science, Engineering and Technology*, Volume 3, 2014.

[22] David A. Patterson, John L. Hennessy, "Computer Organization and Design MIPS Edition The Hardware/Software Interface" *Elsevier Inc*, 2021.

Received 11/July/2023; Accepted 15/August/2023
© 2017 University of Technology, Iraq
ISSN (Print) 1811-9212 ISSN (Online) 2617-3352