Explorative Analysis of Single Cycle to Multi Cycle 32-bit Pipelined MIPS Design for Signal Processing Applications

[1]Jaffar Ali Shaik, [2]Dr. Neelima K, [1]Nagella Mani Teja, [3]Muneer Khan

[1]UG Scholar, Department of ECE, Sree Vidyanikethan Engineering College, Tirupati, India, jaffar.ali.shaik.s@gmail.com, maninagella4447@gmail.com

[2]Assistant Professor, Department of ECE, Mohan Babu University erstwhile Sree Vidyanikethan Engineering College, Tirupati, India, neelumtech17@gmail.com

[3]Graduate Research Scholar, Columbia University, New York, mk4523@columbia.edu

Abstract

Processors being the heavy computational units, demand careful design for digital signal processing applications, medical signal processing applications, etc. With the evolution of Artificial Intelligence, there is huge scarcity for high computing processors such as graphical processing units. Processors support peripheral interfaces with good performance, low power consumption, scalability, accuracy, efficient handling of extra tasks such as coefficient updating and handling for real time processing. For signals such as ECG, EEG, EMG, etc MIPS (Microcontroller without Interlocked Pipeline Stages) processor proves to be a better choice due to its load - store and Harvard architecture with low power, less cost and high performance. MIPS processor is also used for applications that include mobile, embedded systems, networking hardware, consumer electronics, high performance computing, Internet of things and artificial intelligence. This paper proposes aspects like single cycle with 3 input mux, with better branching operations, extra instruction, pipelined approaches for better resource utilization and improved speed, register file extra circuitry to avoid hazard during read and write in single clock cycle, etc. The designs are modeled in Verilog HDL for 28nm "xc7z020clg484-1" FPGA using Xilinx Vivado 2023.2. The performance is evaluated for various factors like LUTs, Flip-Flops, Power Dissipation, Path Delay, performance metrics like execution time, speed up, CPI, throughput, CPU time, etc affecting performance are analyzed. From results, it is clear that DIV instruction utilizes much resource. The area utilized in terms of LUTs by design 5 is less by 3.3% and 8.7% when compared with design 6 and design 3 respectively. The least delay is observed for design 5, however with a slight compromise of 0.25% increase in delay, design 6 proves to be a better design in terms of functionality. The power dissipation is observed to be less in design 5 as 29.092mW but with a slight increase in power dissipation of 2.9%, design 6 proves to be a better choice. The pipelined architecture reduces power dissipation by 40.2%, delay by 50.5% with a slight compromise in area for flip-flops by 17.08%. However to reduce the complexity in the design DIV instruction can be implemented by using shift operations.

Keywords –Microcontroller without interlocked pipeline stages, RISC, CISC, Pipelining, Hazard Detection Unit, Stalls, Parallel Processing, Super Scalar Processor.

Introduction:

The CPU (Central Processing Unit) being the basic component in any computing system and in almost all electronic devices. It is responsible for Instruction execution and running applications, processing data like audio, video and rendering graphics. In recent days, GPU's (Graphical Processing Unit) gained popularity due to their parallel computational capability and large-scale data processing capability which is suitable for deep learning-based applications. Processors with low latency prove to be a good choice for DSP (Digital Signal Processing) applications. The type of architecture decides the role of processor. RISC (Reduced Instruction Set Computer) Processors have minimal instructions and hardware to improve efficiency, cost and power. Hence, RISC processor is efficient for Dedicated DSP applications, low power DSP processors, along with other tasks can also be performed with lower cost, speed and accuracy. The DSP applications with a processor design approach utilize Cook Toom algorithm.

The Cook Toom algorithm, called as Toom-3 algorithm, is a fast convolution algorithm used to process large integers. It balances speed and efficiency in embedded systems and is used for DSP applications where large number of multiplications are involved. It offers advantage of substantial improvement in efficiency for moderate data applications. It also improves speed, efficiency and security in cryptography applications. It can be used in various applications like filtering, frequency-domain processing, number theory transforms, etc. Though, dedicated hardware offers better performance, processor-based implementation Toom-3 algorithm provides other functions like coefficient handling, updating and automating the process.

Among the available variants, as signals such as ECG, EEG, EMG, etc are considered, MIPS processor proves to be a better choice. MIPS (Microprocessor without Interlocked Pipeline Stages) is renowned for its simple load-store architecture, efficiency and low cost. It is a RISC processor with diverse applications from embedded systems to high-performance workstations. It fosters efficient execution of instructions and offers multiple ISAs (Instruction Set Architecture) with different applications. Its architecture is scalable up to 64,128 bits with a support for multimedia and digital signal processing tasks. Being good at efficiency and performance, it is used for wide range of applications such as embedded systems, networking hardware, consumer electronics, high performance computing, Internet of things and artificial intelligence.

This paper discusses with basic processor mechanism, preceded by explanation of the advantages of MIPS processor and its applications in DSP applications. The existing design as per the architecture [1] is discussed. The proposed single cycle design without and with pipelining technique is implemented. Register file simultaneous read and write hazard is discussed and a new custom instruction is implemented. Finally, the paper is concluded with quantitative analysis of single and multi-cycle processors with results and future scope.

Literature Review

From prose, the comparison of the performance definitions and metrics [1] are considered. The single cycle processor where the read and write operation at the same clock edge was not addressed [2] is attempted to resolve in our design. The branch circuitry is proposed in decode stage which can avoid wastage of clocks from three to one is developed to prevent

control hazards [3]. The implementation aspects of Single cycle and multi cycle MIPS processor using Verilog HDL was provided in [4].

CISC processor has complex instruction set which is versatile with adaptable microcodes for backward compatibility of instructions. It has variable instruction size and hence it is difficult to pipeline. Unlike RISC has fixed instructions, fix sized instructions in a fixed format. RISC has load-store architecture with less cost and fast design time [5]. Commercial companies like Apple uses simple RISC as simple and faster instructions are better than complex and slower instructions. It takes less transistors and compiler becomes simple [6].Simple ISAs (Instruction Set Architecture) gives the capability for designing pipelined superscalar RISC processors that can achieve two to four times performance than CISC. It can allow more FUs (Functional Unit) in the chip and lowering cost [7-13].

The authors in the paper implemented pipelining to a 32-bit non-pipelined MIPS Processor [14]. Some authors could analyse some papers of non-pipelined processor, pipelined processors and implemented a low power, less delay and high-performance processor by using 2 phased clocks. They used dummy instruction approach to prevent hazards. They proposed that power gating, clock gating and pipelining can decrease combinational path delay and decrease power by a factor of 3. They placed the branch prediction circuitry to predict at memory stage which wastes 3 clocks [14]. The authors implemented 16-bit RISC Non-pipelined Processor with Harvard architecture and 11 instructions are implemented using Finite state machine approach with 4 states- idle, fetch, decode, execute. They proposed this approach for single cycle [15].RISC processor uses Thread level parallelism and instruction level parallelism to enhance the register set and internal parallelism [16]. The implementation examples for signal processing that are specific for convolution applications like Winograd using processor [17].High-confidence computing depends on ISA, kernels and operating systems. Branch predictions and pipelining plays a vital role in security and in performance [18].

MIPS processor executes millions of instructions without interlocking qualitatively many advanced desirable functions. It is superior alternative to mask-programmed ASICs. Power gating can decrease much power by saving power in standby mode. Deeper pipelines can increase speed by 14 times [19]. MIPS Processor can be applied low power techniques like clock gating, power gating to lower dynamic power during the stored logic doesn't change using halt instruction. It can be applied for branch instruction to save power [20].

A RISC processor uses Load-Store architecture, in which only load store instructions interact with memory. MIPS support embedded systems, connectivity and mobile market. There are mainly two architectures: MIPS32 and MIPS64. The comparison of RISC processors with MIPS, ARM, SPARC is depicted [21]. MIPS proves to be cheaper and allows open-source implementations. So, it is most recommended in educational purposes. MIPS advantages are it is low overhead, power efficient and less complicated.

In general, programmers use ISA as an abstraction to understand type of instructions and their meaning at logic level which makes physical design easy to complete.

Execution time= (# instructions) (Cycles per instructions) (Critical path delay)

Critical path delay decides frequency of operation of processor and clock period should be chosen to fit the load word which is the slowest instruction [22].

The real time loading has been implemented where external input to the processor has been given to instruction memory using UART. This method can be used to interface mips processor for DSP applications [23]. Hazard Detection unit and operand forwarding has been implemented to resolve hazards. Low power modes clock gating, multi threshold techniques have been applied to decrease power by 90%, with compromise of 27% area increase [24].Data Dependency methods have been implemented in this paper. The power increase with frequency has been proved [25].The MIPS is implemented using floating point ALU and I/O port capability which can be used for filtering and DSP applications [26].The MISP can be verified using MARS (MIPS Assembler and Runtime Simulator) IDE to generate MIPS code from MIPS ALP language [27-28].

Further the MIPS Processor, can be devised for single cycle and multicycle approach. Single cycle mips processor is a simple approach of processor that executes an instruction in single clock cycle. But it's clock cycle delay is restricted by the slowest instruction. Though the instruction gets executed in one clock cycle. It is efficient for small instruction sets. It is best suited for simple dedicated applications as it can execute every instruction in one clock cycle and conditional and non-conditional branch executes in 2 clock cycles.

Existing Design

A single cycle processor is implemented by reference to the book computer architecture [1]. The single cycle MIPS processor is implemented with 3 input mux at the input of PC (Program Counter). It is done by adding extra hardware which leads to decrease of complexity, improves branch handling, helpful for subroutines and jump instructions. The single cycle processor is applied pipelining technique to further enhance its performance and efficiency of resource usage. A general hazard which occurs in register file when an instruction is getting read into register and the same register is read at same clock edge. It is eliminated by adding an extra logic. Their performances are analysed comparatively. The Single Cycle MIPS Processor uses low power and compact features that make it best suitable for microcontroller applications like wearable electronics, Internet of Things, in medical devices, etc.

MIPS Processor mainly comprises of Datapath and controller. Datapath contains digital circuit elements like adders, registers and alu etc., controller is a combinational component which generates control signals required for the MUXs, ALU and memories in Datapath as shown in figure 1. Datapath contains two memories as the design considered is a Harvard architecture. Instruction and data memory both are implemented separately to avoid memory conflicts and allow streamlined instruction fetch mechanism.

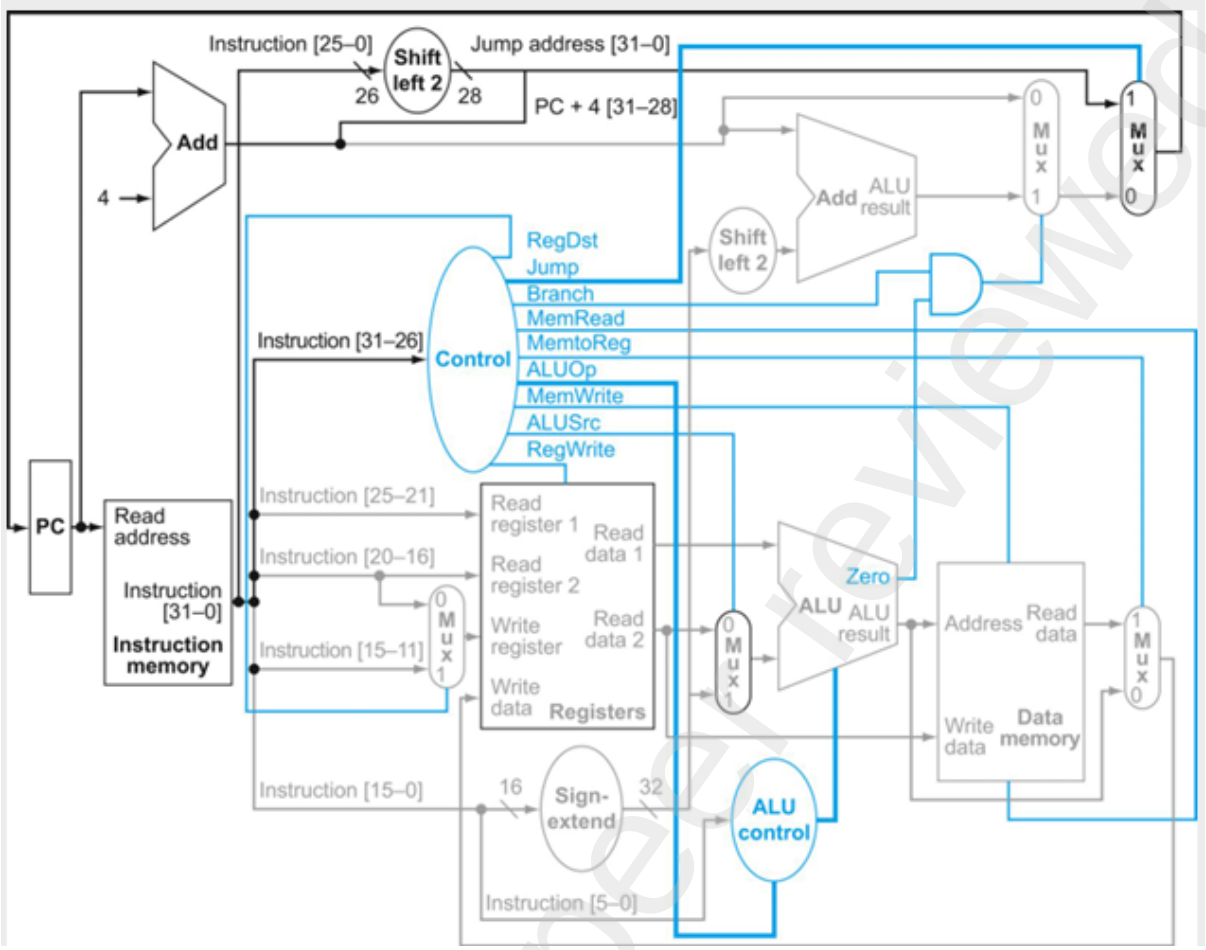


Figure 1. Single Cycle MIPS Architecture

The working mechanism includes program counter, register file and data memory as the only components with enabled clock. PC (Program Counter) is the driving element for instruction execution which proceeds streamlined execution. The data path is divided into 5 blocks each with designated functions to perform. The design is mentioned in each stage with character at the end of signal. The respective stages are

1. **Fetch Stage(F):** PC initiates the fetch process by providing address to instruction memory. Instruction memory fetches the instruction and send to next stage. Simultaneously, the next instruction address is calculates using the adder.
2. **Decode Stage(D):** Register file decodes the instruction as per the instruction format in the figure 2 and gives the operands. Controller generates the control signals based on the decoding of instruction. The hardware sign extend, left shift are used to find effective address of the next instruction if branch or jump are decoded. The destination register value is also selected in this stage using a MUX.
3. **Execute Stage(E):** The operand is operated in ALU to do an operation. During I-type instructions, the effective address and operand value is selected in this stage as input to second operand to ALU. It also bypasses the data to be written into memory during SW instruction.
4. **Memory Access Stage(M):** If the instruction is SW, then it's stored into data memory. If it's LW then the data is read from memory. Else the ALU data from previous stage is bypassed to next stage.

5. **Write Back Stage(W):** The result data after an instruction is write into the register file.

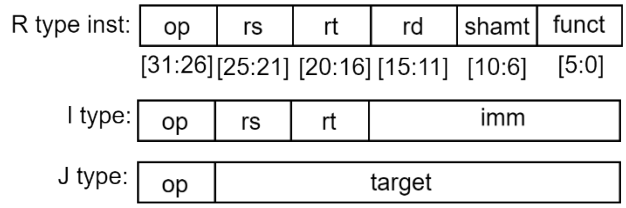


Figure 2. Instruction Format

For Opcodes and Instruction set, almost 122 Instructions are mentioned in the book computer organization and design [1]. But, only nine instructions are implemented in the processor to perform basic general purpose tasks as shown in table 1.

Table 1. Instruction Set with Opcode

Type	Instruction	Instruction	Opcode	funct	Operation
R	**Arithmetic**	ADD	000000	100000	$d=$s+$t
		SUB	000000	100010	$d=$s-$t
	Logic	AND	000000	100100	$d=$s&$t
		OR	000000	100101	$d=$s|$t
		NOR	000000	100111	$d=~($s|$t);
	Compare	SLT	000000	101010	$d=($s<$t)
I	**Load-Store**	LW	100011	-	$t=Dmem[$s+Imm]
		SW	101011	-	Dmem[$s+Imm]=$t
	Branch	BEQ	000100	-	If($s==$t) PC=PC+[Imm<<2]

For a 32-bit processor, the ALU control lines utilize 4-bits to implement various ALU operations as shown in table 2.

Table 2. **ALU Control Lines:**

aluop	Alu Operation
0000	Aluout=A&B;
0001	Aluout=A|B;
0010	Aluout=A+B;
0110	Aluout=A-B;
0111	Aluout=(A<B);
1100	Aluout=~(A|B);

The existing MIPS processor has few addressing modes to assign data into registers and operate in ALU.They specify how operands are retrieved during instruction execution. It plays crucial role in determining memory access patterns, instruction performance.

The addressing modes in the design are as follows:

1. Register addressing mode

2. Base or Displacement addressing mode.

3. PC Relative Addressing mode.

The examples for the above-mentioned addressing modes are as shown in table 3.

Table 3. Addressing Modes and examples

Addressing mode	Instruction	function
Register addressing mode	ADD R1, R2, R3	R1=R2+R3
Base addressing mode	LW R1, 20(R2)	R1=Mem[R2+20H]
PC Relative addressing	BEQ R3, R5, <label>	Address=PC+label

The Register addressing is simplest and execution speed is more in comparison with other addressing modes because it doesn't involve any memory access. In memory accesses effective address computation is required and it takes more time. The base addressing is also called indirect addressing. It points the memory location in base register and immediate offset is added to the value in the base register to get effective address. The offset size is limited to 16 bits and it's a signed number and 2's compliment can be used to give input a negative value. The PC relative addressing is used to compare the two operands and branches to effective address if the operand is equal. The effective address is obtained by extending bits of immediate 16 bits data to 32 bits and adding PC value to it. The negative value is assigned in 2's compliment and for one jump immediate value has to be one.

The decoder logic of control unit used to generate the respective signals is as shown in table 4. In overall view, there are four distinct types of instructions with distinct signals. Opcode is the operational code that represents the operation in an instruction. Op is the intermediate signal between main decoder and ALU decoder in controller. RegDst is used to decide the destination of the result to be write in R-type instruction. ALUSrc manages ALU input B. Jump and Branch indicates change in direction of program. MemWrite is used to write into data memory. RegWrite is to write into register file, and MemtoReg is for selecting data to write into register in write back stage as shown in figure 1.

Table 4. Controller Logic for generation of signals

Instruction	Opcode	op	RegDst	ALUSrc	Jump	Branch	MemRead	MemWrite	RegWrite	MemtoReg
LW	100011	00	0	1	0	0	1	0	1	1
SW	101011	00	x	1	0	0	0	1	0	x
R-Type	000000	10	1	0	0	0	0	0	1	0

BEQ	000100	01	x	0	0	1	0	0	0	x

For example, if instruction is ADD R1, R2, R3 then the R-Type is chosen with opcode=000000, op = 00, RegDst=1, ALUSrc=0, Jump=0, Branch=0, MemWrite = 0, MemRead=0, RegWrite = 1 and MemtoReg = 0.

The existing design uses 2 MUXsat PC, creating complexity for branch handling. It has very small instruction set to be used for wide range of applications. It doesn't support efficient usage of hardware. Hence, we moved onto proposed design as described in next section.

Proposed Design

In MIPS Single Cycle Processor, we are enlarging the instruction set of the processor and adding new instructions to support DSP applications. Instructions like SRLV, SLLV, J, I-type etc., are added. It helps in applying algorithms and processing filter coefficients. We have added a three-input mux as input to the PC. It increases efficiency in handling the branches. Conditional and non-conditional branches are handled with better efficiency. A comparator is used instead of ALU to generate equal signal. The comparator is as considered from [2]. The proposed single cycle MIPS diagram is as shown in figure 3. The improvements over the existing design include efficient handling of branch instructions, hardware reduction by using single multiplexer in the place of two multiplexers, dedicated comparator in ALU, data memory is set as asynchronous read, etc.

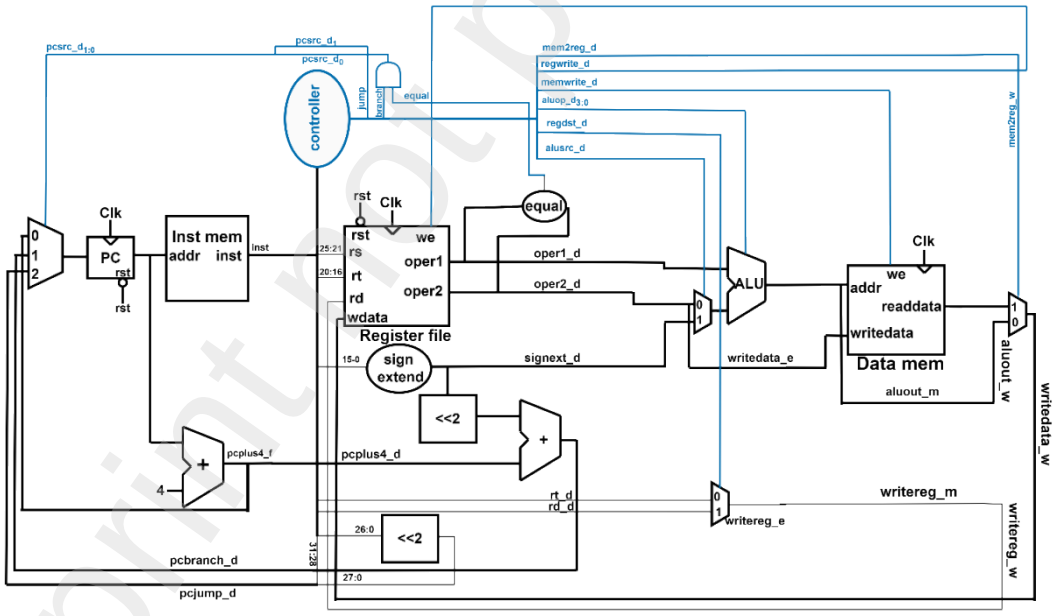


Figure 3. Proposed Single Cycle Diagram

Table 5. Opcode and Instruction Set:

Type	Instruction	Instruction	Opcode	funct	Operation
R	**Arithmetic**	ADD	000000	100000	$d=$s+$t
		SUB	000000	100010	$d=$s-$t
		MUL	000000	011000	$d=$s*$t

		DIV	000000	011010	$d=$s/$t
	Logic	AND	000000	100100	$d=$s&$t
		OR	000000	100101	$d=$s|$t
	Compare	SLT	000000	101010	$d=($s<$t)
	Rotate	SLLV	000000	000100	$d=$t<<$s
		SRLV	000000	000110	$d=$t>>$s
I	**Load-Store**	LW	100011	-	$t=Dmem[$s+Imm]
		SW	101011	-	Dmem[$s+Imm] =$t
	Branch	BEQ	000100	-	If($s==$t) PC=PC+[Imm<<2]
	Arithmetic	ADDI	001000	-	$t=$s+Sign_ext {Imm}
		MULI	001111	-	$t=$s*Sign_ext {Imm}
	Logic	ANDI	001100	-	$t=$s&Sign_ext {Imm}
		ORI	001101	-	$t=$s|Sign_ext {Imm}
J	**Unconditional jump**	J	000010	-	PC=[Imm<<2]

A custom instruction MULI has been implemented to make it convenient for DSP applications in applying algorithms and filtering. It comes handy during immediate coefficient multiplications. The table 5 contains all the instruction set and opcodesthat are included in the design. The operation represents the mechanism of instructions.

Table 6. Logic for controller to generate control signals

Instruction	**Opcode**	**op**	**regdst**	**alusrc**	**jump**	**branch**	**memwrite**	**regwrite**	**mem2reg**
LW	100011	00	0	1	0	0	0	1	1
SW	101011	00	0	1	0	0	1	0	1
R-Type	000000	10	1	0	0	0	0	1	0
ADDI	001000	00	0	1	0	0	0	1	0
MULI	001111	11	0	1	0	0	0	1	0
ANDI	001100	11	0	1	0	0	0	1	0
ORI	001101	11	0	1	0	0	0	1	0
BEQ	000100	01	0	0	0	1	0	0	0
J	000010	00	0	0	1	0	0	0	0

From Table 6, it is obvious that the single signal can be used for memory write and asynchronous read is introduced.

The Table 7 represents the ALU operations and logic for decoder to generate ALU control signal. The decoder contains one decoder to generate control signals and another to generate ALU control signals as shown in Tables 6 and 7.

Table 7. Controller Logic for generation ALU signals

Op and funct fields	**aluop**	**Alu Operation**
(Op==2'b10)&&(funct==6'b100100)||(Op==2'b11)&&(opcode==6'b001100)	0000	Aluout=A&B;
(Op==2'b11)&&(opcode==6'b001101||(Op==2'b10)&&(funct==6'b100101)	0001	Aluout=A|B;

(Op==2'b10)&&(funct==6'b100000)||(Op==2'b00)	0010	Aluout=A+B;
(Op==2'b10)&&(funct==6'b011010)	0011	Aluout=A/B;
(Op==2'b10)&&(funct==6'b100010)||(Op==2'b01)	0100	Aluout=A-B;
(Op==2'b11)&&(opcode==6'b001111)||(Op==2'b10)&&(funct==6'b011000)	0101	Aluout=A*B;
(Op==2'b10)&&(funct==6'b101010)	0110	Aluout=(A<B);
(Op==2'b10)&&(funct==6'b000100)	0111	Aluout=B<<A;
(Op==2'b10)&&(funct==6'b000110)	1000	Aluout=B>>A;

The performance of the Processor is same as the existing in terms of functionality. But while handling the branches, the proposed design is seen to perform better with single clock cycle execution capability as shown in Figure 4.

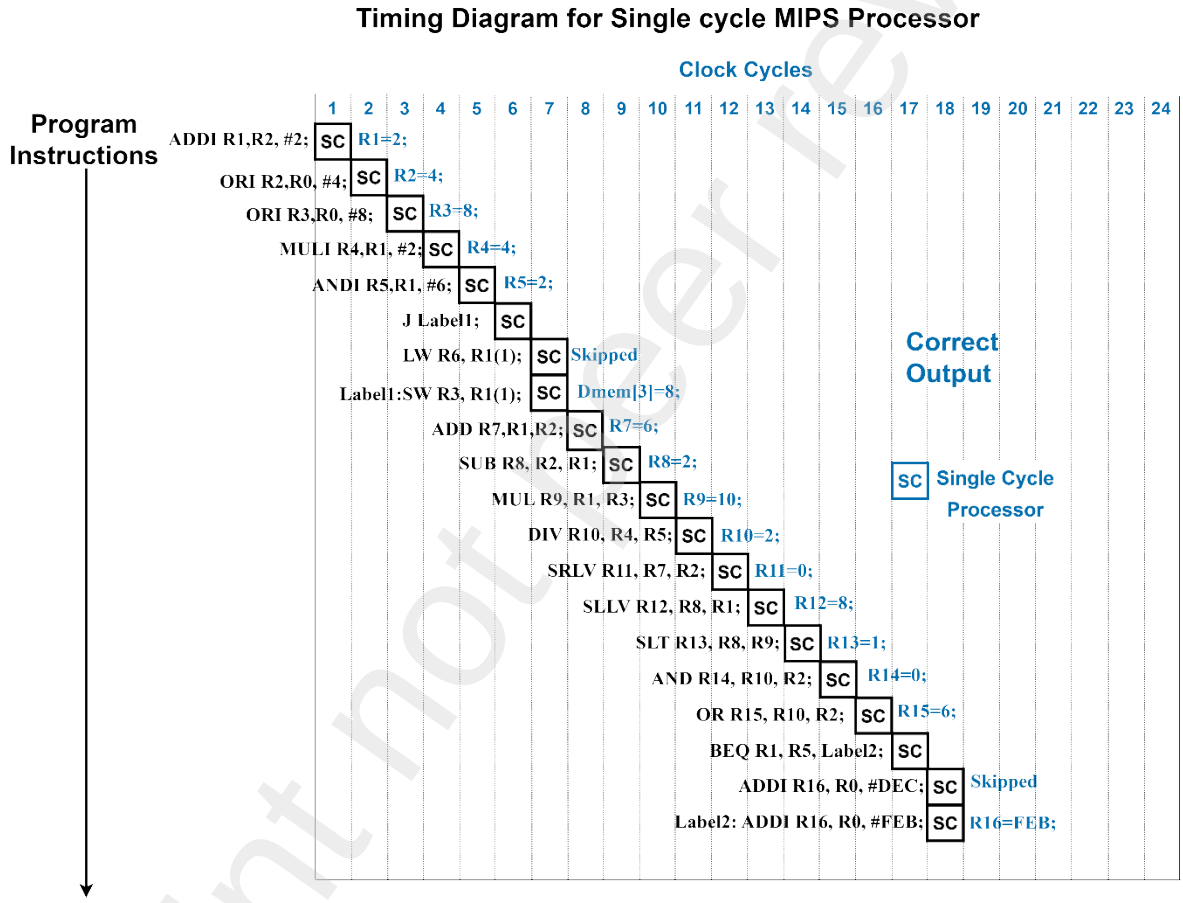


Figure 4. Timing Diagram for MIPS Single Cycle Proposed Design

The timing diagram in the figure 4 depicts the instruction execution with clock count graphically. It shows that all instructions are executed in one clock and branch except that jump instructions take one clock to get branched. However, in this proposed design, the resource utilization is not the fullest. Hence, to improve the efficiency and to make the processor work faster, we considered multi cycle MIPS processor.

Multicycle represents the execution of an instruction in multiple clock cycles. Multicycle approach has the ability to make the hardware components work simultaneously like for example, it can make read into data memory and write into register file simultaneously in a single clock. It uses the hardware components efficiently by using them in parallel. It is

implemented by adding nonarchitecture registers to hold immediate results from different components in data path. This technique is known as pipelining.

Pipelining is a technique used for improving instruction execution throughput by overlapping the execution of different stages of multiple instructions. For faster execution, it allows streamlined continuous execution and paves the way for more advanced performance techniques like Instruction Level Parallelism (ILP) and Data Level Parallelism (DLP).The figure5 depicts the pipeline registers in blue colour in diagram.The total single cycle processor is divided into 5 stages to make a five-stage pipelined processor with pipelined registers in between them which works on positive edge of clock.

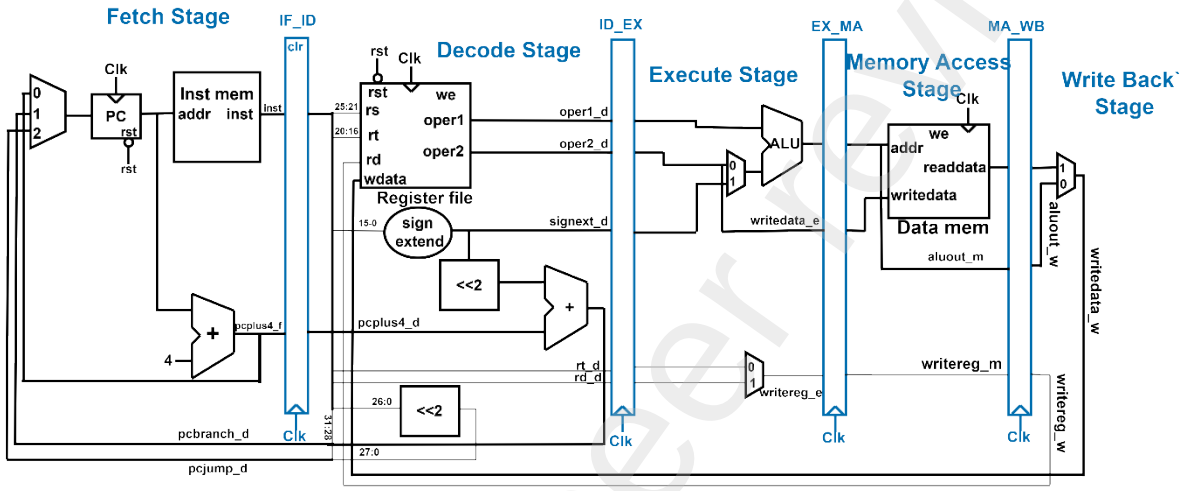


Figure5. Pipelining of MIPS Processor for data path

This design works similar to single cycle MIPS processor except that the controller sends signals in pipelined fashion to make it easier for the instruction to flow through pipeline.This multicycle MIPS processor uses five clocks to execute one instruction. So, CPI (Clocks per Instruction) ideally becomes one. It decreases hardware cost, improves efficiency, throughput, scalability to multiple cores. However, CPI increases due to branching and other instructions. Hazards are the uncertainty in the output that can be caused when an instruction requires data from preceding instruction. The stages are divided based on combinational blocks capable of working independently. The Registers are placed in between the stages of processor which are sequential making the processor combination of combinational and sequential. It passes the intermediate results on positive edge of clock.

This processor supports various instruction types and implements instructions such as branch, load, store, R - type, I – type, etc that are combined to frame the instruction set architecture (ISA). The branch instructions change the order of execution for the program which can be of two types, i.e., conditional and un-conditional branch. Both are implemented using immediate addressing modes. Conditional branch instructions include ‘J (jump <target>)’. Un-conditional branch includes ‘BEQ (branch when equal)’. Load instruction is used to load the data from memory to registers. It is implemented using displacement addressing mode. Store instruction also implemented using the same addressing mode and it moves data from register to memory. ALU instructions are implemented using direct and immediate addressing modes, which does arithmetic and logical operations in the ALU.

In this design, two new addressing modes have been added to support jump and I-type instructions. The addressing modes are as follows:

1. Register addressing mode

2. Immediate addressing mode.

3. Base or Displacement addressing mode.

4. Pseudo-Direct Addressing mode.

5. PC Relative Addressing mode.

Table 8. Addressing Modes

Addressing mode	**Instruction**	**function**
Register addressing mode	ADD R1, R2, R3	R1=R2+R3
Immediate addressing mode	ADDI R1, R2, #20H	R1=R2+20H
Base addressing mode	LW R1, 20(R2)	R1=Mem[R2+20H]
Pseudo-direct addressing mode	J <target>	PC=[PC+target<<2]
PC Relative addressing	BEQ R3, R5, <label>	Address=PC+[label<<2]

The immediate addressing mode which doesn't require extra memory to fetch operand is implemented to execute faster. The immediate operand size is limited to 16 bits. Even in jump conditionally, this mode is used. Pseudo-direct addressing has 26-bit target address. The effective address is calculated by concatenating most significant 4 bits of program counter (PC) to 26 bits target. And lower 2 bits are left shifted zeros are added. It allows 64 MB jump from total 4GB memory.

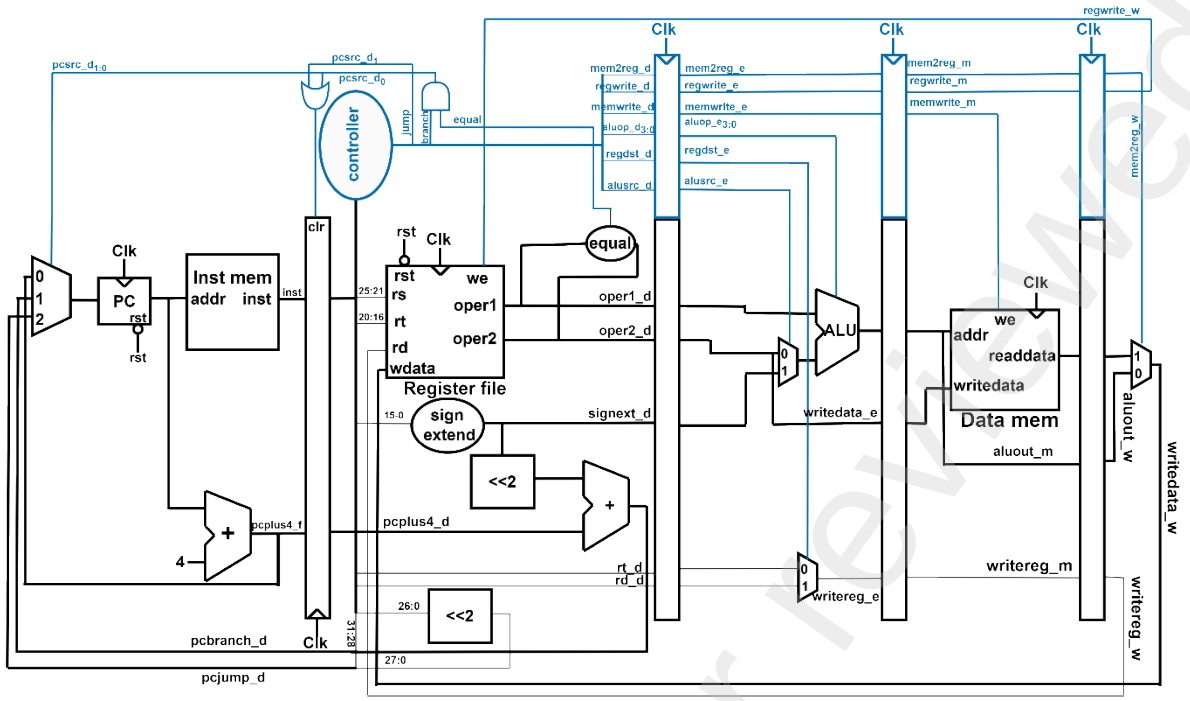


Figure 6. Pipelined MIPS Processor

The controller is placed in decode stage which takes 2 clocks to branch and get to branched instruction as shown in figure 6, that is same with jump instruction. The ALU R-type and I-type instruction takes different paths to the ALU for input B. It is depending on the controller logic and it is implemented by mux at the input of ALU selected by control signal alusrc. The process of execution is almost same as single cycle.

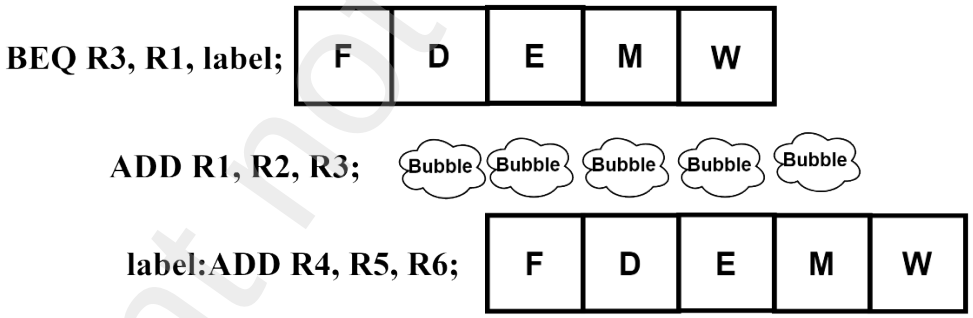


Figure 7. Branch Instruction Pipeline Diagram

The figure 7 shows the branch instruction pipeline where bubble represents the processor hardware becomes idle without executing an instruction by clearing the pipelined register between fetch and decode stage. The wastage of a clock can be minimised if we follow some advanced techniques like dynamic branch prediction being complex and quiet expensive. Here, we implemented static branch prediction by assuming that fetched instruction is not a branch instruction. When its branch the instruction in pipeline is removed by clearing IF_ID register and the MUX at the input of PC is made to select precomputed target instruction address. The working of Jump instruction is same as this branch instruction pipeline diagram.

Hazard during Read and Write in one clock edge:

Here a general hazard or situation that is common in any pipelined processor arises when an instruction is decoding a register in decode stage whose value is simultaneously writing into register file in write back stage. This doesn't allow a register or any flip-flop to read and write the input value at same clock edge. Though, the read operation is insensitive to clock and the write is clock sensitive. To overcome this, we have proposed an extra logic as shown in figure 8 by using two comparators are used to comparers, rt field values with rd. The result is ANDed with write enable and the operand is selected using a MUX based on the logic.

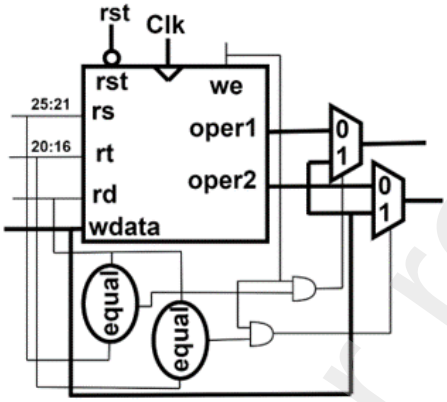


Figure 8. Updated Register file circuitry

The Logic for Register file read write hazard in same clock edge is modelled as

if((rs==rd) && we==1'b1) op1= data

elseop1=Regfile[rs]

if((rt==rd) && we==1'b1) op2= data

elseop2=Regfile[rt]

The timing diagram that describes this situation is as shown in figure 9. The overlap of instruction is observed and the output is generated per clock cycle. Initially there will not be any output for four clock cycles.

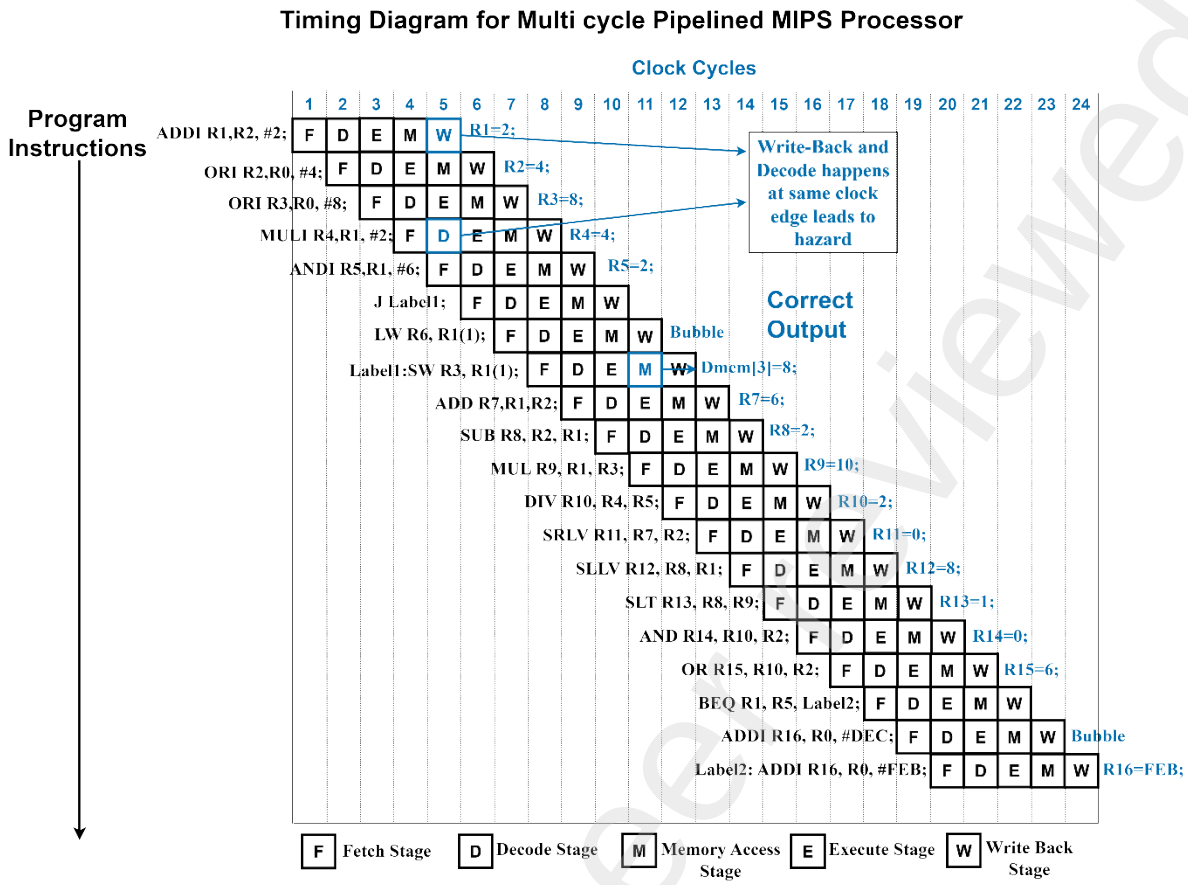


Figure 9. Timing Diagram for Pipelined MIPS Processor

Quantitative Analysis

The performance metrics to analyse the performance of MIPS processors include execution time, speedup, efficiency, CPI, throughput, CPU time, etc.

Execution time is the time to execute all the instructions in a program. It is ideal time to execute the instructions in pipelined processor. But some instructions like branch take more time to execute. It is formulated as

$$Texe = N * CPI * Tclk$$

For pipelined processor,

$$T_{exe}=[N+(n-1)] *T_{clk}$$

Where

T_{exe} - Execution Time

N-number of instructions in program

n-number of pipelined stages

T_{clk} -time period of clock

Speedup is the performance enhancement due to pipelining from a non-pipelined processor. It indicates the execution speed enhancement. It depends on number of pipelined stages (n). if

there is no waste of clocks or bubble or stalls, the speedup will be n. these stalls occurs when an instruction requires data form previous instruction which hasn't write into register file yet.

$$\text{Speedup (S)} = \left(\frac{\text{PerformanceofPipelined Processor}}{\text{Performance ofnon} - \text{pipelinedProcessor}} \right)$$

The Performance is inversely proportional to execution time.

$$S = \frac{N * n * Tclk}{(N + (n - 1)) * Tclk} = N * \frac{n}{N + (n - 1)}$$

when instructions are many, N>>n, S=n.

Where

T_{exe} - Execution Time

N-number of instructions in program

n-number of pipelined stages

T_{clk} -time period of clock

Efficiency of the processor can be measured in different ways. Pipelined processors have much potential for high efficiently execution by overlapping instruction execution. But it has to minimize stalls. The single cycle processor doesn't utilize resources efficiently.

$$\text{Efficiency} = \frac{\text{Sgiven}}{\text{Smax}}$$

$S_{max}=n$

Where n -number of pipelined stages

The main factors to measure efficiency are:

Instruction per Cycle (IPC) is the average number of instructions can complete in one clock cycle. Higher the IPC, the better the efficiency.

Cycles per instruction (CPI) is the inverse of IPC. It indicates the average number of clocks required per instruction. The lower it is, the better the efficiency.

CPI= (number of CPU clock cycles)/ (total number of instruction)

$CPI=(\sum^{k}_{i=1} CPI_i * I_i)/N$

For pipelined processors when stalls are present,

CPI= 1(Base CPI) + (number of stalls* fraction of instructions * fraction of their occurrence)

Where k=number of categories

Throughput represents the amount of work required to complete an instruction per unit time. It can also refer to number of instructions executed per unit time, measured as instructions per second.

Throughput=$N/(T_{exe}*1000000)$ Million Instructions per second.

Where T_{exe} - Execution Time

N-number of instructions in program

CPU Time is the number of clock cycles that CPU needs to complete an instruction.

$$\text{CPU time} = (\text{Number of clocks per program} * T_{clk}) = N*CPI*T_{clk.}$$

Where

T_{exe} - Execution Time

N-number of instructions in program

n-number of pipelined stages

T_{clk} -time period of clock

For the designs mentioned here, based on assembly code we consider clock period used as 10ns, and then the above metrics are evaluated as the details mentioned in tables 9 and 10.

Table 9. CPI of Assembly Code for Pipelined processor

Instruction Type	CPI	Percentage of Instructions
Load	5	1(5%)
Store	4	1(15%)
Branch	2	2(10%)
ALU	5	16(80%)

Table 10. Performance Metrics

Parameters	Single Cycle (ideal)	Single Cycle (practical)	Multi cycle (ideal)	Multi cycle (practical)
Execution Time	200ns	180ns	240ns	240ns
Speedup	-	-	5	4.16
CPI	1	0.9	1	1.2
Throughput	100 MIPS*	100 MIPS*	83 MIPS*	83 MIPS*
Efficiency	-	-	1	0.832
CPU Time	200ns	180ns	200ns	240ns

*MIPS- Million Instruction Per Second.

The assembly code shown in timing diagrams is taken and the CPI details are mentioned in Table 9. The variation in ideal and practical values in single cycle processor is due to branch instructions can skip the instruction in the delay slot within one clock. The Throughput and execution time may not seem to improve a lot. But the performance will increase significant as the instruction count increases as pipelined processor speed increases i.e., for single cycle only one clock is considered then execution time is 200ns but for multicycle five clock cycles are considered then the actual execution time is 240ns/5 = 48ns. The delay gets decreases due to pipeline registers decrease the critical delay by a factor of number of pipeline stages. The factors affecting the performance of the processor are represented in table 11.

Table 11. Factors affecting Performance

Factors	CPI	Clock Cycle Time	Instruction Count
Instruction Set Architecture	✓	-	✓
Compiler Optimization	✓	-	✓
Program	✓	-	✓
Hardware Technology	-	✓	-
Hardware Organization	✓	✓	-

In any processor, CPI depends on ISA, Compiler, Program and Organization. Clock Cycle time is mostly technology dependent. Instruction count is decided by ISA, Compiler optimization and the way of program written.As the instructions count increase, CPI decreases and provides improve in performance.The CPI depends on various factors such as instruction count, compiler effects on instruction count, ISA effects on instruction count, algorithm implemented, clock rate. All these calculations lead to CPI calculation. So, bench marks are used to compare different architectures. Benchmarks in a sense they are programs with different type of instructions called work load characterisation to compare the performance parameters to compare the architectures. Standard benchmarks, synthetic benchmarks and benchmarks suites (SPEC06, PARSEC, SPLASH) are used.

SPEC Ratio=Execution time of reference processor/ Execution time of given machine.

Results and Discussion:

The designs are modeled in Verilog HDL and implemented for Zynq 7000 series FPGA with part number in xc7z020clg484-1 using Xilinx vivado 2023.2.

RTL Design for Single Cycle MIPS Processor:

The RTL design is as shown in figure 10. Here as the fetch components contain mux, PC, instruction memory and adder are placed in one module, the output is made as a wire to 16 bits of data memory at index zero (Dmem[0]) to convert it into synthesizable design.

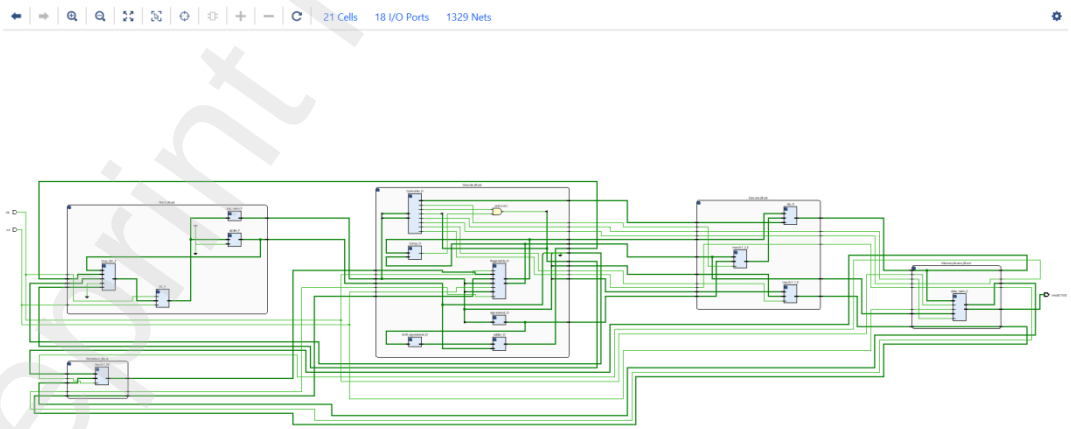


Figure 10. RTL Schematic of Single Cycle MIPS Processor

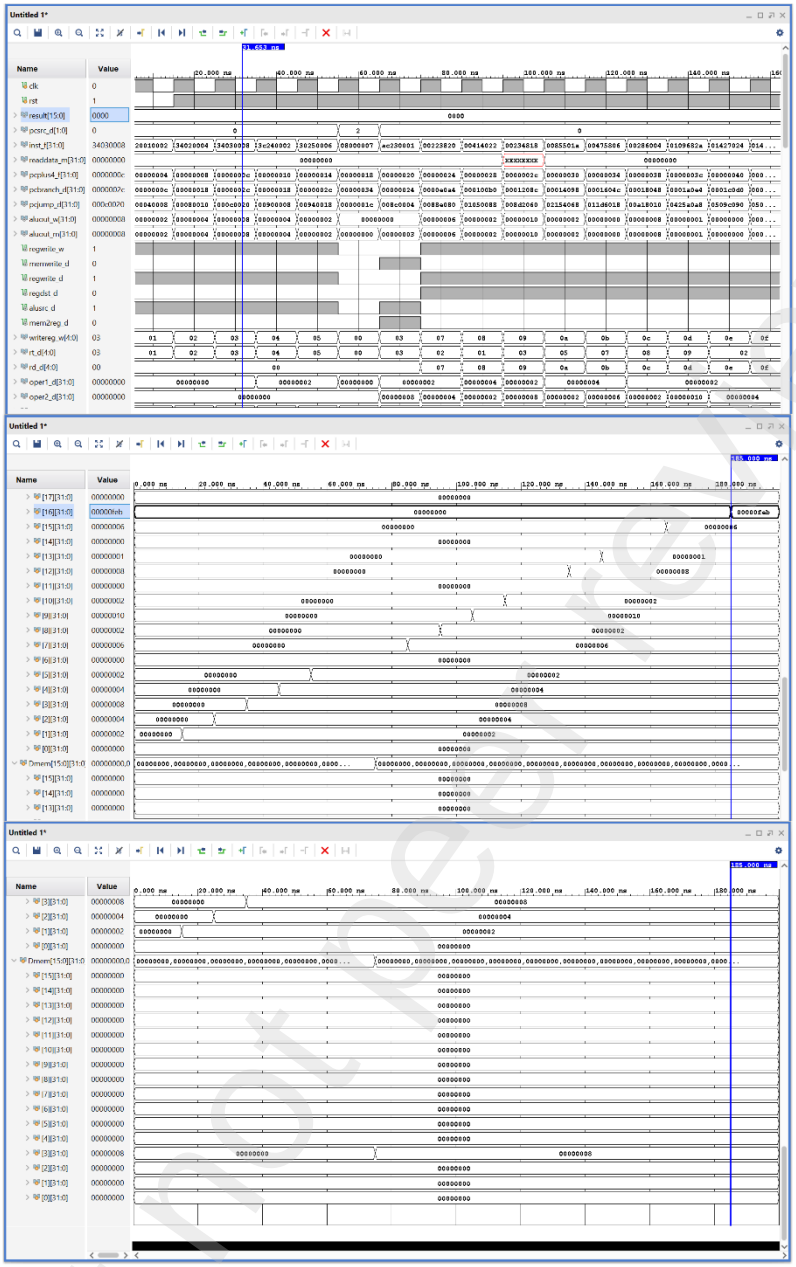


Figure 11. Simulation Waveform of Single Cycle MIPS Processor

The Figure 11 depicts the working of single cycle MIPS processor shown in simulation waveform same as that of figure 4. When reset pin is held low, the register file resets and once made high, the register file and data memory start to update.

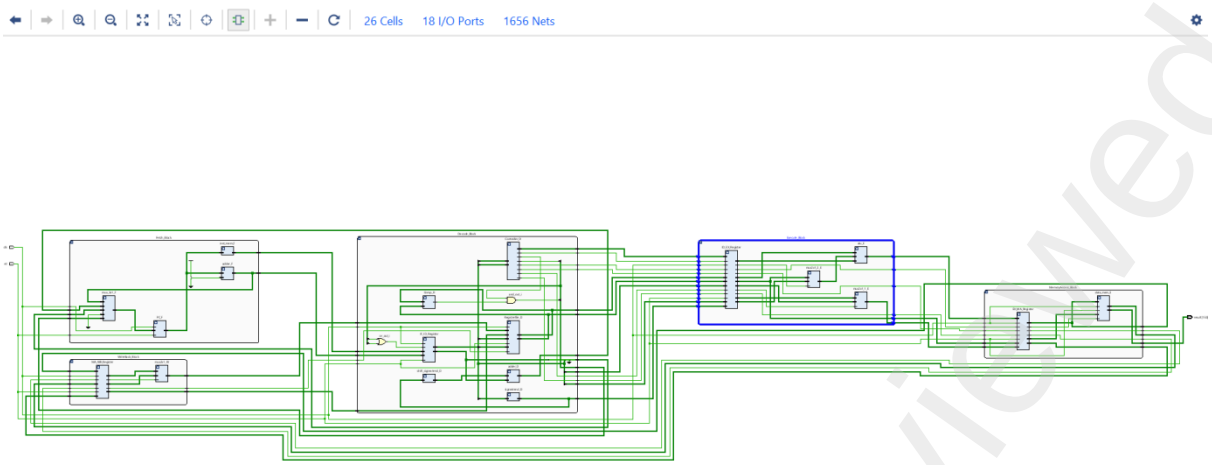


Figure 12. RTL Schematic for Multi Cycle without register file hazard circuitry

The figure 12 shows the RTL Schematic for Multi Cycle without register file hazard circuitry. The result is the lower 16 - bits least significant bits of data memory at index zero are made as output for synthesizable design.

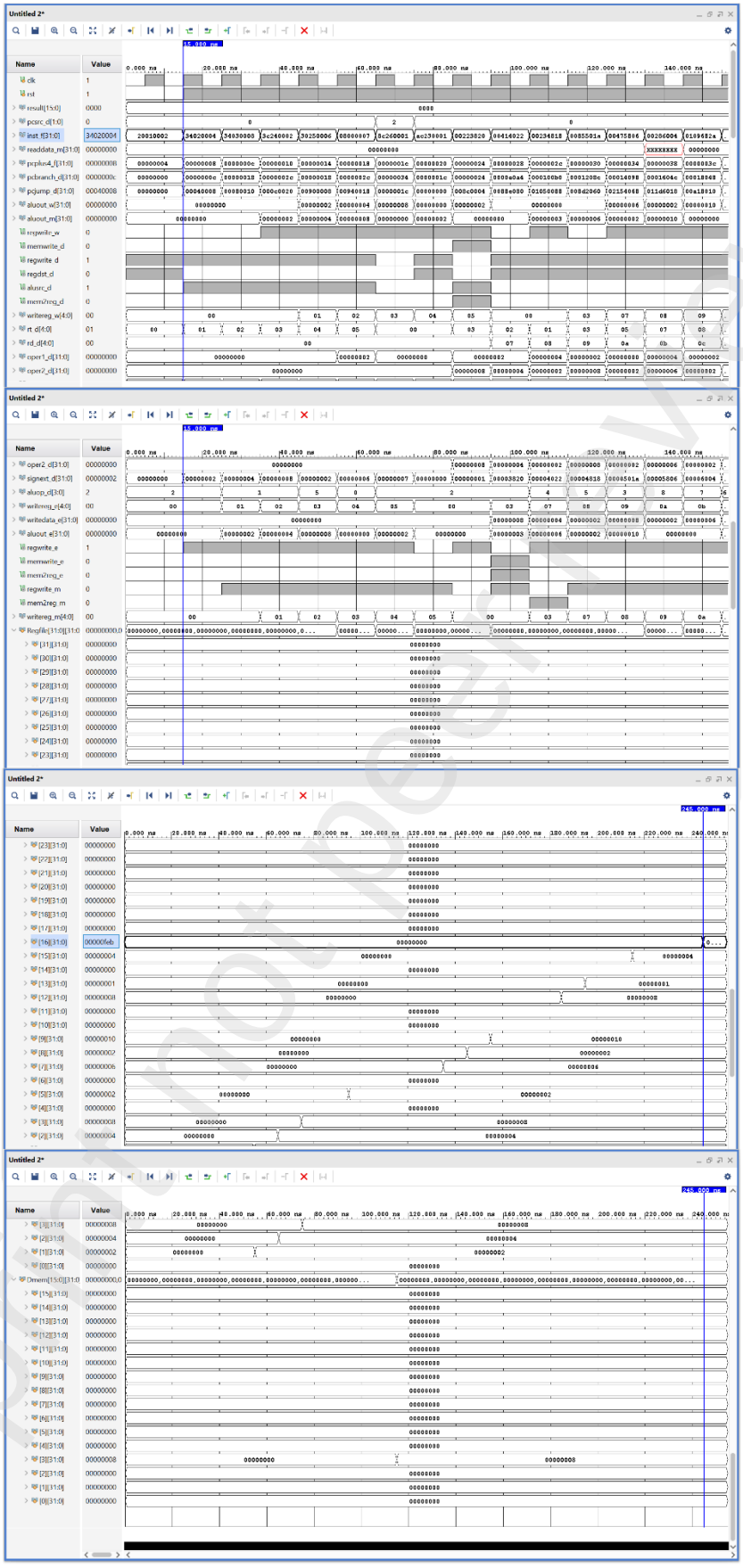


Figure 13. Simulation Waveform of Multi Cycle without register file hazard circuitry

The figure 13 depicts the working of pipeline and the updating of register file registers and data memory same as that of the timing diagram as shown in figure 9. It is shown that there is an error in R4 register due to register file hazard.

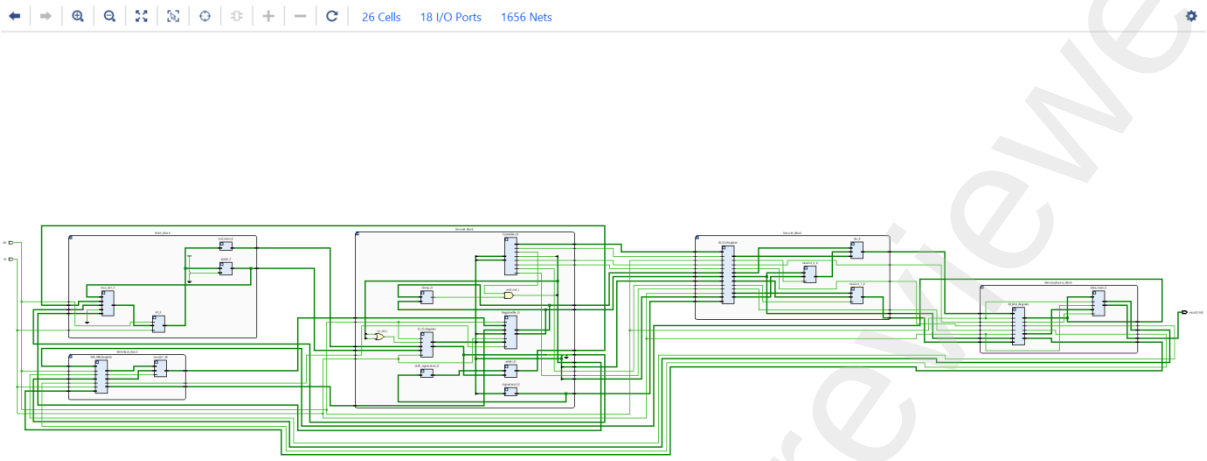
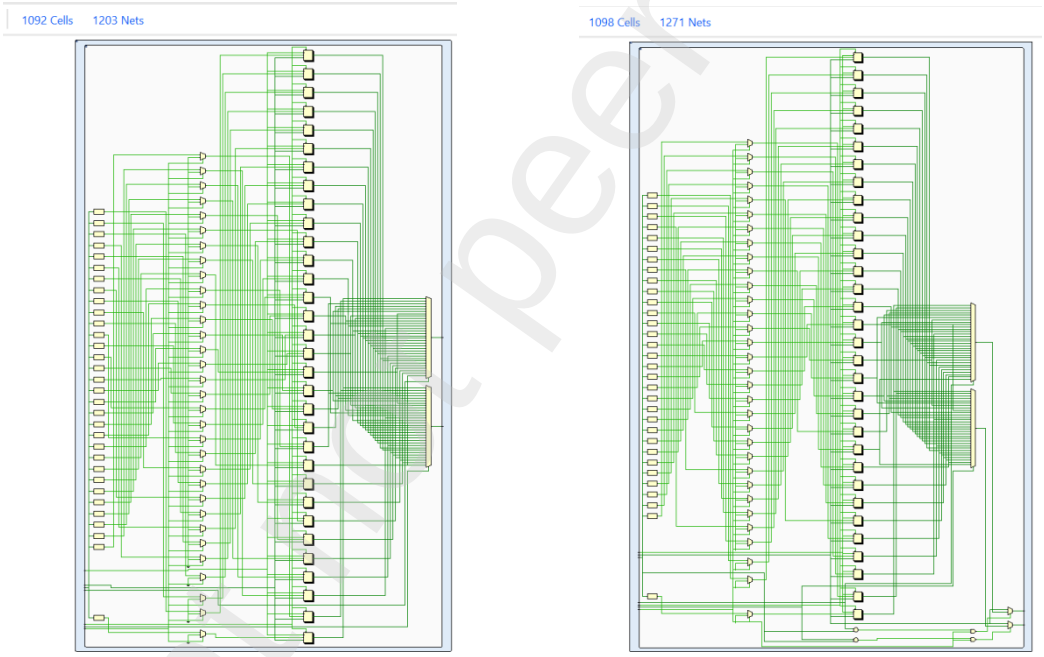


Figure 14. RTL Schematic for Pipelined MIPS Processor



(a) without Extra Circuitry (b) with Extra Circuitry

Figure 15. Register File without Extra Circuitry

The RTL schematic of Pipelined MIPS processor is shown in figure 14. Its exclusive register file without and with extra circuitry is shown in figure 15.

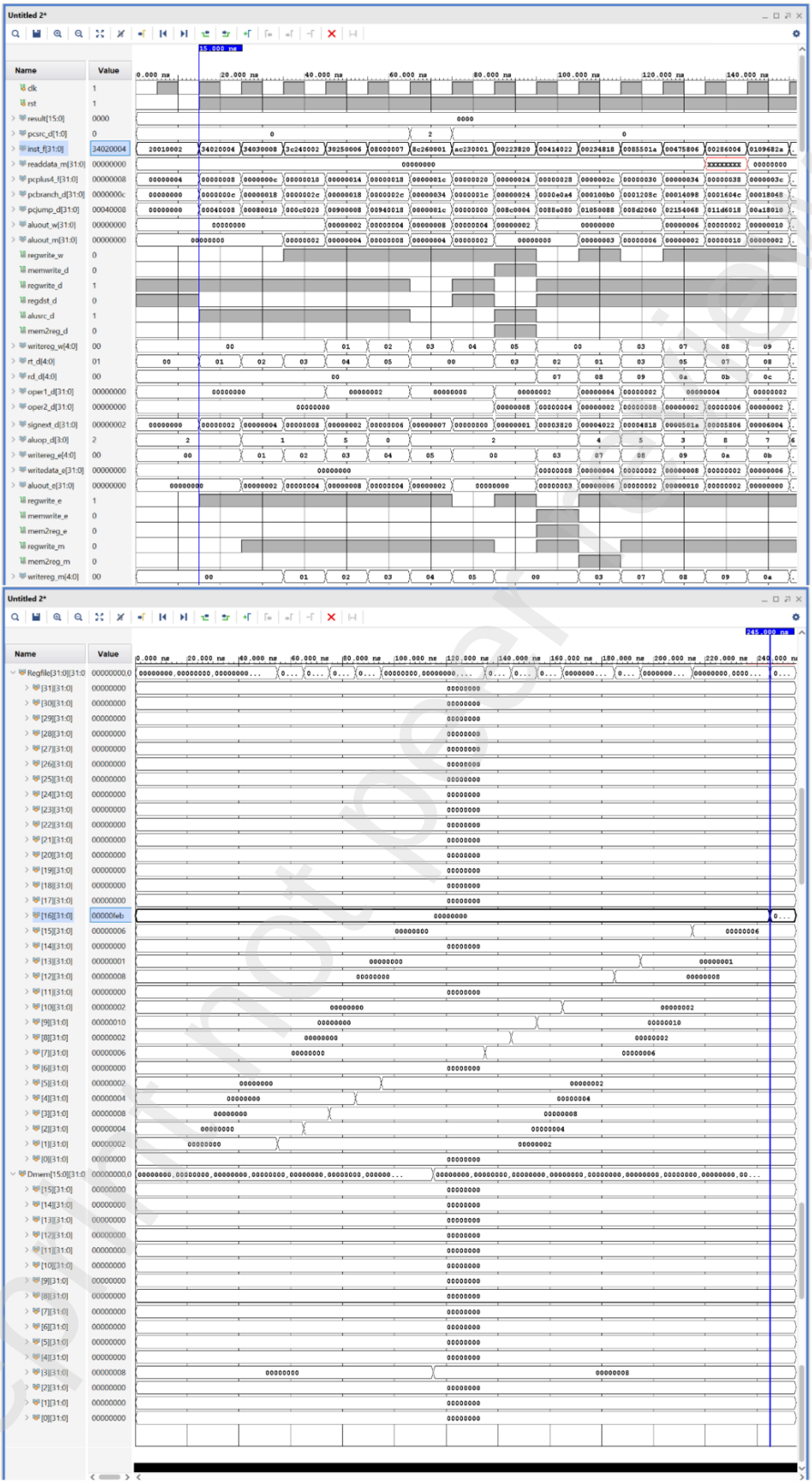


Figure 16. Simulation Waveform for Pipelined MIPS Processor with Extra Circuitry

The Simulation Waveform for the Pipelined MIPS Processor with extra circuitry in register file is shown in figure 15 where the read and write in same clock hazard is eliminated.

The comparison results in terms of area, delay and power dissipation are shown in table 12.

Table 12. Comparison of processor designs in terms of area, delay and power dissipation

Components	Design-1: Single Cycle MIPS Processor	Design-2: Multi Cycle MIPS Processor Without Register file Hazard Circuitry	Design-3: Multi Cycle MIPS Processor with Register file Hazard Circuitry	Design-4: Single Cycle without DIV instruction	Design-5: Multi Cycle without Register file circuitry and DIV instruction	Design-6: Multi Cycle with Register file circuitry and without DIV instruction
LUTs (Out of 53200)	2261	2182	2217	1210	1104	1142
FF(Out of 106400)	1557	1833	1833	1557	1823	1823
Total path delay	156.092ns	131.100ns	133.794ns	33.155ns	16.346ns	16.388ns
Dynamic Power	86.479mW	61.492mW	63.732mW	49.101mW	28.052mW	28.942mW
Total Power	87.518mW	62.532mW	64.771mW	50.14mW	29.092mW	29.981mW

The DIV instruction utilizes much resource, so if DIV instruction is removed, then a drastic change in implemented results is observed. The internal digital signal processors used are held same as 3 out of 220 for all designs. The I/Os utilized are 18 out of 200, only one BUFG is utilized among 32 and the static power is observed to be 1.039mW. The area utilized in terms of LUTs by design 5 is less by 3.3% and 8.7% when compared with design 6 and design 3 respectively. The least delay is observed for design 5, however with a slight compromise of 0.25% increase in delay, design 6 proves to be a better design in terms of functionality. The power dissipation is observed to be less in design 5 as 29.092mW but with a slight increase in power dissipation of 2.9%, design 6 proves to be a better choice. However even though single cycle design is easy, multi cycle design without DIV instruction by using shift operations with register file circuitry proves to be the best among the mentioned. Hence, the pipelined architecture reduces power dissipation by 40.2%, delay by 50.5% with a slight compromise in area for flip-flops by 17.08%.

Conclusion

Processors being the crux of systems, demand careful design for critical digital signal processing applications, medical signal processing applications, etc. With evolving AI based designs, processors need to support peripheral interfaces with good performance, low power consumption, scalability, accuracy, efficient handling of extra tasks such as coefficient updating and handling of real time signal processing. MIPS processor has load - store and Harvard architecture is the design of choice. This paper proposes aspects like single cycle with 3 input mux, with better branching operations, extra instruction, pipelined approaches for better resource utilization and improved speed, register file extra circuitry to avoid hazard during read and write in single clock cycle, etc. The designs are modeled in Verilog HDL for 28nm “xc7z020clg484-1” FPGA using Xilinx Vivado 2023.2. The performance is evaluated for various parameters. From results, it is clear that DIV instruction utilizes much resources

which can be compensated by using shift operations. Also the pipelined architecture reduces power dissipation by 40.2%, delay by 50.5% with a slight compromise in area for flip-flops by 17.08%. In future, the designs are optimized to overcome read after write hazard, load use-load store hazard, control hazard, etc.

References

[1] John L. Hennessy, David A. Patterson, "Computer Architecture: A Quantitative Approach Sixth Edition", Morgan Kaufmann Publishers.

[2] David A. Patterson, John L. Hennessy, "Computer Organization and Design: The Hardware/Software Interface, Sixth Edition", Morgan Kaufmann Publishers, 2021.

[3] David Money Harris, Sarah L. Harris, "Digital Design and Computer Architecture", Morgan Kaufmann Publishers.

[4] Yamin Li, "Computer Principles and Design in Verilog HDL", Published by John Wiley & Sons Singapore Pte Ltd, 2015.

[5] Kirti Tokas, Dhruv Sharma, Lokesh Yadav," RISC AND CISC ARCHITECTURE", International Journal of Innovation Research in Technology (IJIRT), ISSN: 2349-6002, Vol. 1 Issue 06, 2014, pp. 2095-2097.

[6] hahzeb, Naveed Hussain, Amanulllah, Furqan Ahmad and Salman Khan, "Comparative Study of RISC AND CISC Architectures", International Journal of Computer Applications Technology and Research (IJCAT), ISSN: 2319-8656, Vol. 5 Issue 07, 2016, pp. 500–503,

[7] Syed Roohullah Jan, Khan, F., Muhammad Tahir, Shahzad Khan, (2016) "Survey: Dealing Non Functional Requirements At Architecture Level", VFAST Transactions on Software Engineering, (Accepted 2016)

[8] M. A. Jan, "Energy-efficient routing and secure communication in wireless sensor networks," Ph.D. dissertation, 2016.

[9] M. A. Jan, P. Nanda, X. He, and R. P. Liu, "A Lightweight Mutual Authentication Scheme for IoT Objects," IEEE Transactions on Dependable and Secure Computing (TDSC), "Submitted", 2016.

[10] M. A. Jan, P. Nanda, X. He, and R. P. Liu, "A Sybil Attack Detection Scheme for a Forest Wildfire Monitoring Application," Elsevier Future Generation Computer Systems (FGCS), "Accepted", 2016.

[11] Puthal, D., Nepal, S., Ranjan, R., & Chen, J. (2015, August). DPBSV--An Efficient and Secure Scheme for Big Sensing Data Stream. InTrustcom/BigDataSE/ISPA, 2015 IEEE (Vol. 1, pp. 246-253). IEEE.

[12] Puthal, D., Nepal, S., Ranjan, R., & Chen, J. (2015). A Dynamic Key Length Based Approach for Real-Time Security Verification of Big Sensing Data Stream. In Web Information Systems Engineering–WISE 2015 (pp. 93-108). Springer International Publishing.

[13] Puthal, D., Nepal, S., Ranjan, R., & Chen, J. (2016). A dynamic prime number based efficient security mechanism for big sensing data streams.Journal of Computer and System Sciences.

[14] Shobhit Shrivastav, Shubham Kumar, Sarthak Gupta and Bharat Bhushan, "Qualitative Analysis of 32 Bit MIPS Pipelined Processor", International Journal of Engineering

Research & Technology (IJERT), ISSN: 2278-0181, Vol. 9 Issue 05, May-2020, pp. 558-561.

[15] M Design of 16-bit RISC Procesor International Journal of Scientific Research in Physics and Applied Sciences, 1(1), Feb 2017, pp 25-30.

[16] Ankitha C.S and Dr. Kiran V," Design and development of a 5-stage Pipelined RISC processor based on MIPS", International Research Journal of Engineering and Technology (IRJET), e-ISSN: 2395-0056, p-ISSN: 2395-0072, Vol. 09, Issue 10, 2022.

[17] Ramana K, Mrs. G. Sree Lakshmi and Dr. M. Ch. P. Jagdissh, "16-Bit RISC Processor Design for Convolution Applications", International Journal of Engineering Research Applications (IJERA), ISSN: 2248-9622, Vol. 3, Issue 6, Nov-Dec 2013, pp.1291-1294

[18] Shuangbao (Paul) Wang, "Design high-confidence computers using trusted instructional set architecture and emulators", High-Confidence Computing 1, Published by Elsevier B. V. on behalf of Shandong University, 2021.

[19] P. Indira, M. Kamaraju and Ved Vyas Dwivedi, "Design and Analysis of a 32-Bit Pipelined MIPS RISC Processor", International Journal of VLSI deisgn& Communication Systems (VLSICS), Vol. 10, No 5, October 2019.

[20] Vijay Kumar Jinde, Nagaraju Boya, Swapna ChinthakuntaandRamanjappaThogata, "Design and Implementation of Low Power Pipelined 64-Bit RISC Processor using FPGA", International Journal of Advanced Research in Engineering and Technology (IJARET), ISSN 0976 - 6480 (Print) ISSN 0976 - 6499 (Online) Volume 5, Issue 2, February (2014), pp. 61-69.

[21] Sarah El Kady, Mai Khater, and MerihanAlhafnawi, "MIPS, ARM and SPARC- an Architecture Comparison", Proceedings of the World Congress on Engineering 2014 Vol I, WCE 2014, July 2 - 4, 2014, London, U.K. ISBN: 978-988-19252-7-5.

[22] Mohammad Zaid and Prof. Pervez Mustajab, "Design and Application of RISC Processor", IMPACT 2017, IEEE, pp 242-246.

[23] Mazen Bahaidarah, Hesham Al-Obaisi, Tariq Al-Sharif, Mosab Al-Zahrani, Mohammad Awedh and Yasser Seddiq, "A Novel Technique for Run-Time Loading for MIPS Soft-Core Processor", IEEE, 2013.

[24] P.V.S.R.Bharadwaja ,K. Ravi Teja, M.Naresh babu and K.Neelima, "Advanced Low Power RISC Processor Design using MIPS Instruction Set", IEEE Sponsored International Conference on Electronics and Communication Systens (ICECS), 2015. pp. 1252-1257.

[25] Mohit N. Topiwala and N. Saraswathi, "Implementation of a 32-bit MIPS Based RISC Processor using Cadence", IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), 2014, ISBN No. 978-1-4799-3914-5, pp 979-983.

[26] Mr. S. P. Ritpurkar, Prof. M. N. Thakare and Prof. G. D. Korde, "Synthesis and Simulation of a 32Bit MIPS RISC Processor using VHDL", IEEE International Conference on Advances in Engineering & Technology Research (ICAETR), August 01-02, 2014, Dr. Virendra Swarup Group of Institutions, Unnao, India

[27] Rohit J and Raghavendra M, "Implementation of 32-bit RISC processors without interlocked Pipelining on Artix-7 FPGA Board", Proceeding of Second International conference on Circuits, Controls and Communications, IEEE, 2017, pp. 105-108.

[28] MARS
compilertutorial:https://www.courses.missouristate.edu/KenVollmar/mars/tutorial.html