International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

FPGA Implementation Of 32-bit RISC-V ISA Extension with DSP Application

Jocy Merin John[1], Dhivya Raj[2], Deepa Susan Jacob [3], Laghima P M[4]

[1]PG Scholar, Electronics Engineering Govt. Model Engineering College, Kerala,
[2,3,4]Assistant Professor, Electronics Engineering Govt. Model Engineering College, Kerala,

Abstract
A RISC-V core capable of DSP operations is increasingly in demand for signal processing in applications such as audio and video processing, wireless communication, and image processing. Adding DSP extensions to the RISC-V architecture enables efficient signal processing and reduces the need for separate DSP chips. In this paper, a 32-bit RISC-V implemented using Verilog HDL on Basys3 Artix-7 FPGA as a processor capable of performing 14 instructions where an instruction named MAC-MULC16-32 is developed which is used instead of MULC16-32 instruction for saving power by incorporating a MAC unit and along with that, an FIR filter implemented with the help of the MAC unit demonstrating DSP application, are being discussed.
Index Terms—single cycle RISC-V processor, MAC unit, ISA Extension, DSP Application, FIR filter

1.INTRODUCTION

Articulated as " risk-five," RISC-V (Reduced Instruction Set Computer - V generation) is an ISA (Instruction Set Architecture) based on RISC standards. It is provided under an open-source license. RISC-V does not incorporate any novel or innovative features on a technical level. It employs a load-store architecture and adheres to established RISC principles. The specification defines different levels of instruction sets, just like many other ISAs. This includes versions with 32 and 64 bits as well as extensions that can support floating point instructions. Versions that are suitable for a variety of applications, including desktop personal computers and supercomputers with vector processors to small embedded microcontrollers, can now be developed.

DSP(Digital Signal Processing) operations require specialized instructions that can perform complex mathematical computations, such as multiplication, addition, and filtering. This integration of DSP capabilities within the processor can lead to cost savings, power efficiency, and increased performance for DSP-intensive applications. Furthermore, the open-source nature of RISC-V enables customization and optimization for specific use cases, allowing for the development of specialized processors tailored to specific applications, such as DSP-intensive ones. This flexibility and modularity of RISC-V make it an attractive option for the development of DSP-capable processors.

The document [2] describes the RISC-V unprivileged architecture and discusses the RISC-V instruction sets for 8-bit, 16-bit, and 32-bit. This paper develops code for the implementation of an 8-Bit microprocessor that implements instruction pipelining. The design and realization of an 8-bit microprocessor are discussed in [3]. After integration, the proposed processor design can be implemented on an FPGA platform. Functional verification and performance evaluation using Xilinx tools and

ModelSim demonstrate promising results. The results highlight the reduction in program execution time achieved by the pipelined architecture compared with its non-pipelined counterpart.

In [4] using register files, and the ALU(Arithmetic Logic Unit) forwarding unit is inserted. It also supports hazard detection and data forwarding to prevent data hazards.The article [5] focuses on the design of fixed-width multipliers with the Baugh-Wooley algorithm and discusses the implementation of the Multiply and Accumulate (MAC) unit using a fixed-width signed multiplier and a fixed-width Baugh-Wooley multiplier. With an n-bit multiplier and n-bit multiplicand, the Baugh-Wooley algorithm-derived fixed-width multipliers produce an n-bit output product. Both full-adders and half-adders have been utilized in the design of the fixed-width signed multipliers in previous studies. They have created the fixed-width Booth-Wooley multiplier in this paper by employing only the final Ripple Carry Adder (RCA) and an array of full adders.

The paper [6] proposes a Software Defined Radio (SDR)-based architecture for Distributed Satellite Systems (DSS) to support multi-satellite communication using reconfigurable ground stations. It addresses the limitations of conventional single-satellite communication systems by enabling multiple satellite signals to be processed simultaneously. The study evaluates software transceiver performance, decoding success rates, and potential FPGA acceleration bottlenecks at different symbol rates. An embedded system architecture is also proposed to improve communication efficiency and scalability. A fully synthesizable, 64-bit, single-core RISC-V (RV64I) processor with an in-order architecture and low hardware complexity is presented in [7], targeting resource-constrained embedded applications while [8] describes the design and verification of a 32-bit ARM-compatible RISC processor featuring a three-stage pipeline, 32-bit ALU, six register banks, and a four-cycle multiplier. A latch-based design approach is adopted here to achieve low power consumption and reduced hardware complexity. The proposed approach in [9] optimizes SM4 encryption across AVR, RISC-V, and ARM processors using efficient register allocation, instruction-level optimization, and SIMD parallel processing. It achieves up to 12.93× performance improvement over reference implementations while providing flexibility in speed, memory, and code-size optimization. Low-power IoT nodes struggle to balance performance within milliwatt limits. [10] discusses the challenges of achieving high performance within the limited power budget of battery-operated IoT devices. It proposes architectural diversity through two RISC-V cores, Zero-riscy and Micro-riscy, designed for different workloads: control-oriented tasks and mixed arithmetic/control applications. This approach aims to improve energy efficiency and performance adaptability while supporting a common software environment. The open-source instruction set architecture known as RISC-V was developed with the intention of accommodating custom extensions and architectures. Software-defined radio applications are the focus of the RISC-V ISA instruction set extension described in [11]. Not only do endpoint devices for the Internet of Things need to function within a very limited power envelope of a few milliwatts, but they also need to be adaptable in terms of their computing capabilities, which can range from a few KOPS to GOPS. Near Threshold (NT) activity can accomplish higher energy proficiency, and the exhibition adaptability can be acquired through parallelism. In [12], portray the plan of an opensource RISC-V processor center explicitly intended for NT activity in firmly coupled multi-center groups. To reduce pressure on the shared memory hierarchy and increase computational density, we employ instruction extensions and microarchitectural optimizations.

This paper presents the design and implementation of a single-cycle, 32-bit RISC-V processor with a 32-bit data width and an instruction set comprising 14 instructions. These instructions are classified into three categories: Complex Arithmetic Instructions (ADDC16, SUBC16, CRASC16, CRSAC16, SRAC16, SRAIC16, and SLLIC16), Reconfigurable Multiplication Instructions (MUL2ADD16-32, MULC8-16,

MULC16, and MULC16-32), and Automatic Gain Control (AGC) Instructions (CLRSB, PACK-INIT, and PACK). Among these, the MULC16-32 instruction exhibits relatively high power consumption. To address this limitation, the instruction is modified by incorporating a dedicated Multiply-Accumulate (MAC) unit, resulting in a modified instruction designated as MAC-MULC16-32. Furthermore, to demonstrate the processor's Digital Signal Processing (DSP) capabilities and evaluate the functionality of the integrated MAC unit, a Finite Impulse Response (FIR) filter is implemented.

The rest of this paper is structured as follows: Section 2 describes a brief overview of the architecture of RISC-V selected for the proposed processor, MAC unit and FIR filter. In Section 3, the Instruction Set of the proposed processor is discussed. The results obtained are discussed in Section 4. Finally, Section 5 concludes the paper.

2.THE PROPOSED WORK

A. Architecture Of RISC-V Selected For The Proposed Processor

The architecture of RISC-V selected for the proposed processor is shown in figure 1. It is a single cycle RISC-V processor, in which the data width is equal to 32 bits.

1. Architecture

The Reduced Instruction Set Computing (RISC) principles serve as the foundation for the RISC-V processor. It follows a load/store architecture, meaning that arithmetic and logical operations can only be performed on data in registers, and memory accesses are done explicitly using load and store instructions.

2. 32-bit Instruction Format

The RISC-V instruction set has a fixed 32-bit instruction format. Instructions are aligned to word boundaries in memory.

3. Single-Cycle Execution

In a single-cycle processor, each instruction takes exactly one clock cycle to execute. This means that the processor completes an instruction fetch, decode, execute, and memory access (if required) all within a single clock cycle.

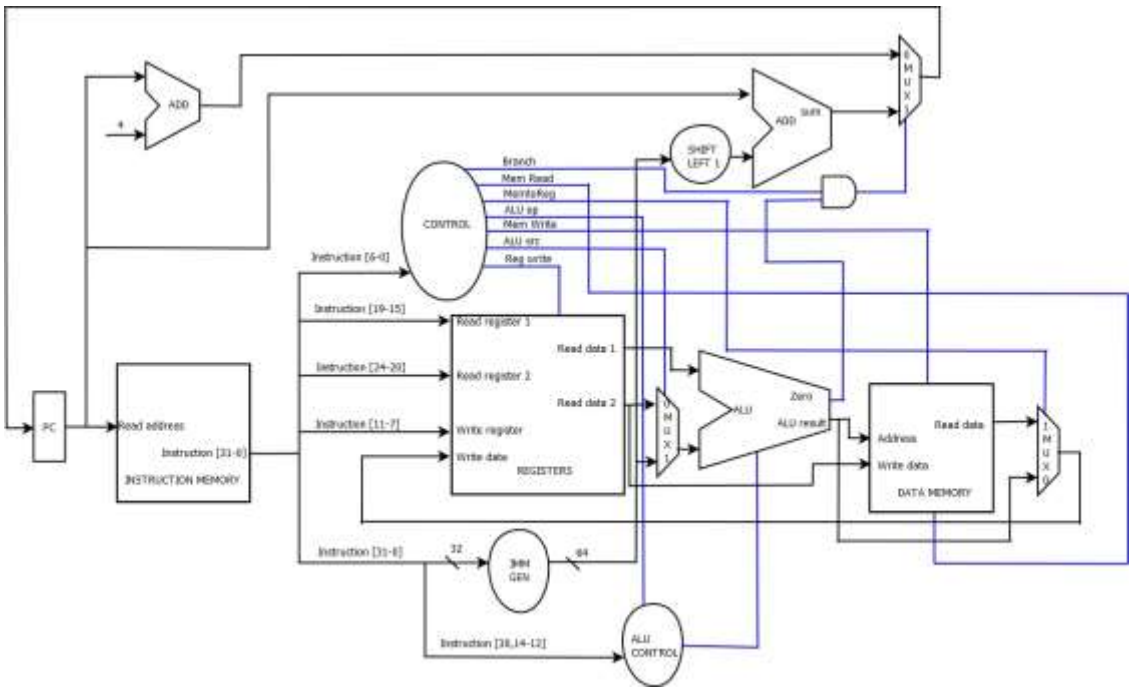


Fig. 1. 32-BIT SINGLE CYCLE RISC-V ARCHITECTURE

4. Instruction Fetch

The processor fetches instructions from memory using the program counter (PC) as the address. The PC is incremented after each fetch to point to the next instruction.

5. Instruction Decode

The fetched instruction is then decoded to determine the operation to be performed and the operands involved.

6. Execution

The instruction is executed in the execution stage. Arithmetic, logical, and control operations are performed based on the instruction type. Data is read from registers and operated upon.

7. Memory Access

If the instruction requires a memory access, such as a load or store instruction, the memory stage is performed. The effective memory address is computed, and the data is either read from or written to memory.

8. Write-Back

The result of the execution stage is written back to a register if the instruction updates the register.

9. Control Unit

The control unit generates control signals that coordinate the execution of various stages and control the data flow within the processor.

10. Register File

The processor includes a register file that holds a fixed number of general-purpose registers (typically 32 registers in RISC-V). These registers are used for storing data during execution.

11. ALU (Arithmetic Logic Unit)

The ALU performs all arithmetic and logical operations on data. It supports basic operations like addition, subtraction, logical AND/OR, shift operations, and comparison.

12. Branching and Jumping

RISC-V supports branching and jumping instructions that alter the program flow based on certain conditions or unconditional jumps.

13. Exceptions and Interrupts

The processor should handle exceptions (e.g., divide-by-zero) and interrupts (e.g., timer interrupt) by saving the current state, transferring control to an exception/interrupt handler, and restoring the state after handling.

14. Performance Limitations

Single-cycle processors have a fixed instruction execution time, which can result in lower performance compared to pipelined processors. Complex instructions take the same amount of time as simple ones.

15. Control Hazards

In single-cycle processors, control hazards occur when a branch or jump instruction changes the flow of the program. Resolving control hazards can be challenging in a single-cycle design and often requires flushing the pipeline.

B. MAC Unit

A MAC unit is a specialized hardware component used for performing multiply-accumulate operations. It combines the operations of multiplication and addition in a single unit. The MAC operation involves multiplying two numbers and adding the product to an accumulator. It is commonly used in various computational tasks, such as digital signal processing (DSP), linear algebra operations, and filtering algorithms. The MAC unit typically consists of three main components: a multiplier, an adder, and an accumulator.

1. Multiplier

The multiplier is responsible for multiplying two input operands. It takes the two inputs and produces their

product as an output. The multiplier can be implemented using various techniques, such as shift-and-add, array multiplication, or Booth's algorithm.

2. Adder

The adder takes the output of the multiplier and adds it to the value stored in the accumulator. It performs the accumulation step of the MAC operation. The adder can be a simple arithmetic adder or a more complex adder with additional features like carry-save adders or pipelined adders for high-performance applications.

3. Accumulator

The accumulator is a storage element that holds the accumulated value. It stores the intermediate result of the MAC operation and provides it as an input to the adder in the subsequent cycles. The accumulator retains its value across multiple MAC operations, allowing for iterative accumulation.

4. Parallelism and Pipelining

MAC units can be designed to exploit parallelism and pipelining techniques to improve performance. Parallel MAC units can perform multiple MAC operations simultaneously by duplicating the multiplier, adder, and accumulator components. Pipelined MAC units divide the MAC operation into multiple stages and process multiple MAC operations in a pipeline fashion, enabling higher throughput.

5. Data Width

MAC units have a specific data width, which determines the size of the operands and the resulting product. The data width may vary based on the application requirements and the targeted system architecture.

MAC units find extensive use in digital signal processing applications, such as audio and video processing, image processing, and telecommunications. They are also employed in neural networks and machine learning algorithms for performing matrix multiplications and convolution operations. MAC units are critical for achieving high-performance com-puting in applications that heavily rely on multiply-accumulate operations. Optimizations can be applied to the MAC unit design, such as using dedicated hardware accelerators, em-ploying SIMD (Single Instruction, Multiple Data) instructions, or utilizing parallelism and pipelining techniques, to enhance performance and efficiency.

C. FIR Filter As DSP Application

A Finite Impulse Response (FIR) filter is a digital filter used to process discrete-time signals by performing a convolution operation between the input signal and a finite set of filter coefficients. It is termed finite because its impulse response consists of a limited number of samples and becomes zero beyond a specified duration. FIR filters are widely employed in Digital Signal Processing (DSP) applications, particularly for signal conditioning, noise reduction, frequency selection, equalization, and system identification.

The characteristics of an FIR filter are primarily determined by its impulse response, which represents the filter coefficients or taps. The output is computed as a weighted sum of the present and previous input samples. Mathematically, the output of an \(N\)-th order FIR filter is expressed as:

$$y[n] = \sum_{k=0}^{N} (h[k]x[n-k])$$

where x[n] represents the input signal, y[n] denotes the output signal, h[k] represents the filter coefficients, and N is the filter order. The filtering operation involves a sequence of multiplication and accumulation (MAC) operations, making the MAC unit a fundamental computational element in FIR filter implementation.

One of the major advantages of FIR filters is their ability to provide an exact linear phase response when their coefficients exhibit appropriate symmetry or anti symmetry. This ensures a constant group delay

across the frequency spectrum, thereby preserving the relative phase relationships among the frequency components of the input signal. Consequently, FIR filters are particularly suitable for applications such as audio processing, communication systems, and biomedical signal processing, where phase distortion must be minimized. Furthermore, FIR filters are inherently stable due to their finite impulse response and can be designed to achieve desired frequency characteristics using techniques such as windowing, frequency sampling, and least-squares optimization.

The filter order determines the number of coefficients required for implementation and directly influences its frequency-selective performance. Increasing the filter order generally improves the ability to achieve sharper transition bands and better frequency-response characteristics. However, this improvement comes at the expense of increased hardware complexity, computational requirements, power consumption, and potentially greater processing latency.

FIR filters find extensive applications in audio and speech processing, image processing, biomedical instrumentation, radar systems, digital communication, and equalization systems. They can be implemented in both real-time and offline processing environments. In real-time applications, the implementation must satisfy the required throughput and latency constraints while maintaining computational and power efficiency. Therefore, efficient hardware architectures, particularly those employing optimized MAC units and multiplier-accumulator structures, are important for achieving high-performance FIR filter implementations.

3.THE INSTRUCTION SET

The instruction set (taken from [1]) of the proposed processor without modification using a MAC unit shown in Figure 2 is designed as per the specifications in [1].

Mnemonic	Instruction	Operation
ADDC16 rd, rs1, rs2, imm	16-bit Addition & Shift Right Arithmetic Immediate	rd.L = (rs1.L + rs2.L)>>imm rd.H = (rs1.H + rs2.H)>>imm
SUBC16 rd, rs1, rs2, imm	16-bit Subtraction	rd.L = (rs1.L - rs2.L)>>imm rd.H =(rs1.H - rs2.H)>>imm
CRASC16 rd, rs1, rs2, imm	16-bit Cross Add & Sub	rd.L = (rs1.L + rs2.H)>>imm rd.H = (rs1.H - rs2.L)>>imm
CRSAC16 rd, rs1, rs2, imm	16-bit Cross Sub & Add	rd.L = (rs1.L - rs2.H)>>imm rd.H = (rs1.H + rs2.L)>>imm
SRAC16 rd, rs1, rs2	16-bit Shift Right Arithmetic	rd.L = rs1.L>>rs2 rd.H = rs1.H>>rs2
SRAIC16 rd, rs1, imm	16-bit Shift Right Arithmetic Immediate	rd.L = rs1.L>>imm rd.H = rs1.H>>imm
SLLIC16 rd, rs1, imm	16-bit Shift Left Logical Immediate	rd.L = rs1.L<<imm rd.H = rs1.H<<imm
MUL2ADD16-32 rd, rs1, rs2, imm	Two "16x16" and Signed Addition	rd = [(rs1.L * rs2.L) + (rs1.H * rs2.H)]>>imm
MULC8-16 rd, rs1, rs2, H1, H2, C, imm	if C=0: 8-bit complex multiplication, *conj*=1 if C=1: 8-bit complex conjugate multiplication, *conj*=-1	if Hx = 1, {ix,qx} = {rsx.B2,rsx.B3} if Hx = 0, {ix,qx} = {rsx.B0,rsx.B1} rd.L = (i1 * i2 - *conj* * q1 * q2)>>imm rd.H = (i1 * q2 + *conj* * i2 * q1)>>imm
MULC16 rd, rs1, rs2, C	if C=0: 16-bit complex multiplication, *conj*=1 if C=1: 16-bit complex conjugate multiplication, *conj*=-1	rd.L = (rs1.L*rs2.L)>>16 - *conj**(rs1.H*rs2.H)>>16 rd.H = (rs1.H*rs2.L)>>16 + *conj**(rs1.L*rs2.H)>>16
MULC16-32 rd1, rd2, rs1, rs2, C	if C=0: 16-bit complex multiplication, *conj*=1 if C=1: 16-bit complex multiplication, *conj*=-1	rd1 = (rs1.L * rs2.L - *conj* * rs1.H * rs2.H) rd2 = (rs1.H * rs2.L + *conj* * rs1.L * rs2.H)
CLRSB rd, rs	Count leading redundant sign bits	if rs = 0, rd = 0 if rs > 0, rd = clz (rs) - 1 if rs < 0, rd = clz (~ rs) - 1
PACK-INIT rs	Initialize packing barrel-shifter	*r_sh_pack* = rs
PACK rd, rs1, rs2	Pack a 16 bit complex value on 32 bits after removing *r_sh_pack* redundant sign bits	Ival = rs1 << *r_sh_pack* Qval = rs2 << *r_sh_pack* rd = Qval[31:16] :: Ival[31:16]

Shorthand definitions:
r.B3→r[31:24], r.B2→r[23:16], r.B1→r[15:8], r.B0→r[7:0], r.H → r[31:16], r.L→ r[15:0].
"~" is bitwise not, "::" is concatenation, "clz" returns the number of leading zeros of an unsigned 32-bit value.

Figure. 2. INSTRUCTION SET

4.Results And Discussion

A. Simulation Result

The behavioral simulation result of the proposed processor's ALU showing the execution of all the 14 instructions is shown in Figure 3 and Figure 4.

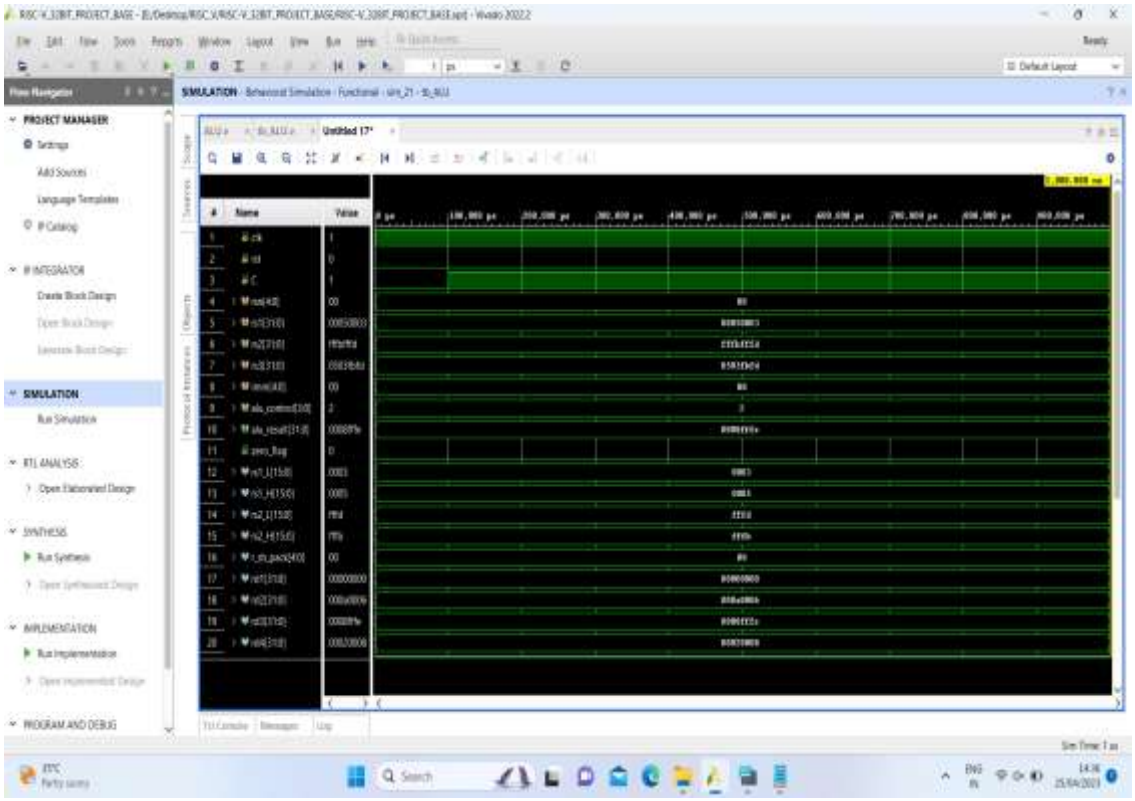


Figure. 3. 32-BIT RISC-V ALU-2

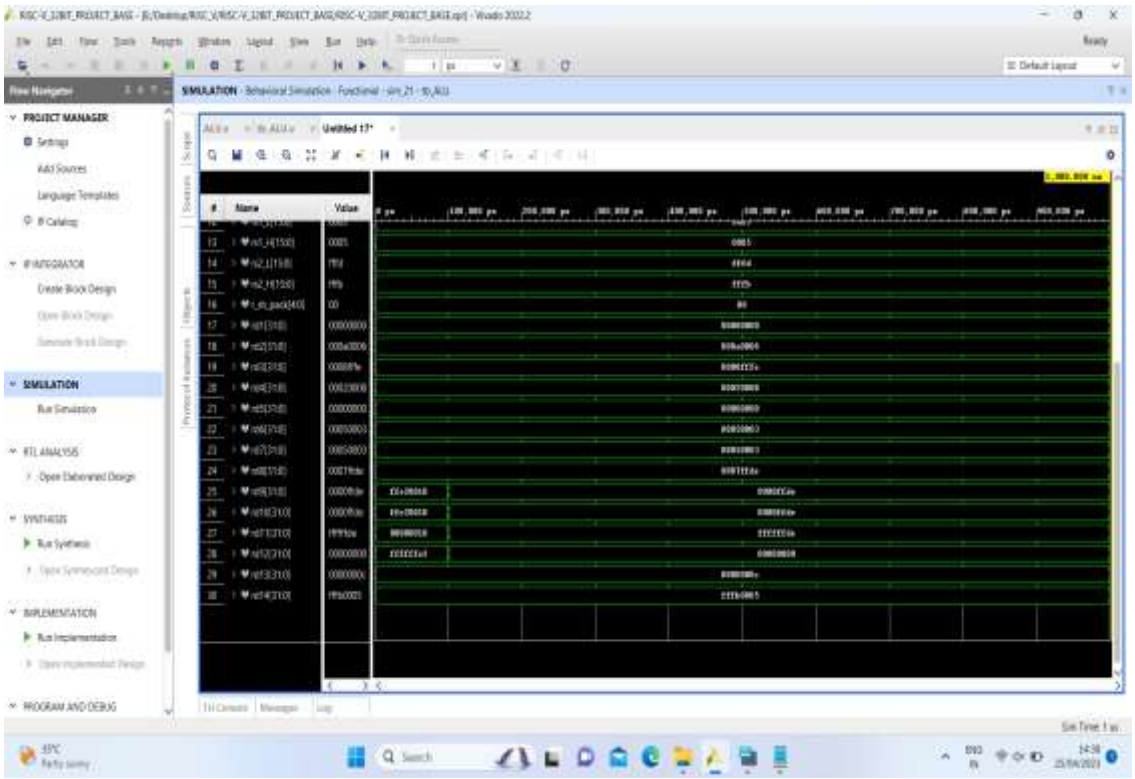


Figure. 4. 32-BIT RISC-V ALU-2

B. Power Estimation of MULC16-32 and MAC-MULC16-32

A comparison of power estimation results of 32bit multiplication instructions MULC16-32 (without MAC unit) and MAC-MULC16-32 (with MAC unit) are illustrated in TABLE I.

TABLE I

Power consumption	
MULC16-32	MAC- MULC16-32
24.103 W	6.718

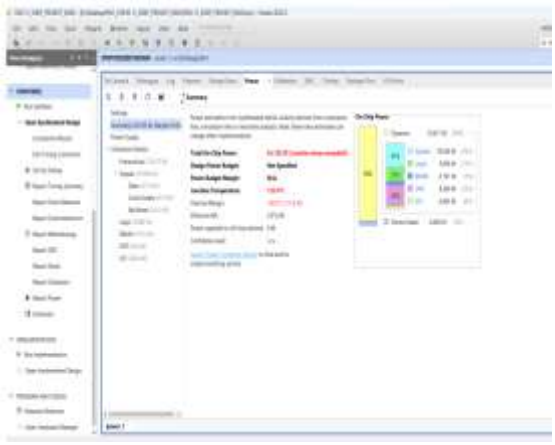


Figure:5 Power Estimation Result of MAC- MULC16-32

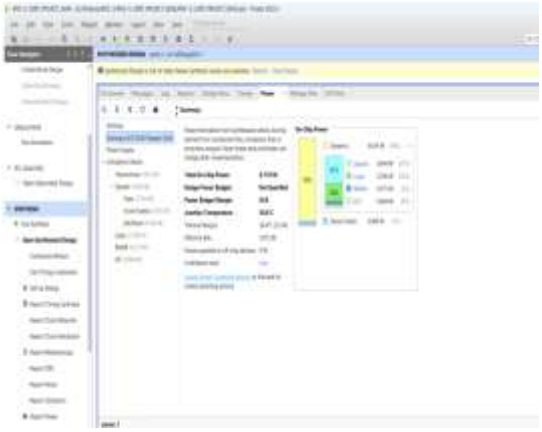


Figure:6 Power Estimation Result of MULC16-32

C. FPGA Implementation Result

The work is implemented on Basys 3 FPGA board and obtained the output via ChipScope. The coding is done in Vivado 2022.2 Software using Verilog HDL(Hardware D-scription Language) on Basys3 Artix-7 FPGA. The final output obtained after FPGA implementation is shown in Figure 7 and Figure 8. The final output shows the FPGA implementation of the 32-bit RISC- V ISA extension (including modified instruction MAC-MULC16-32) .

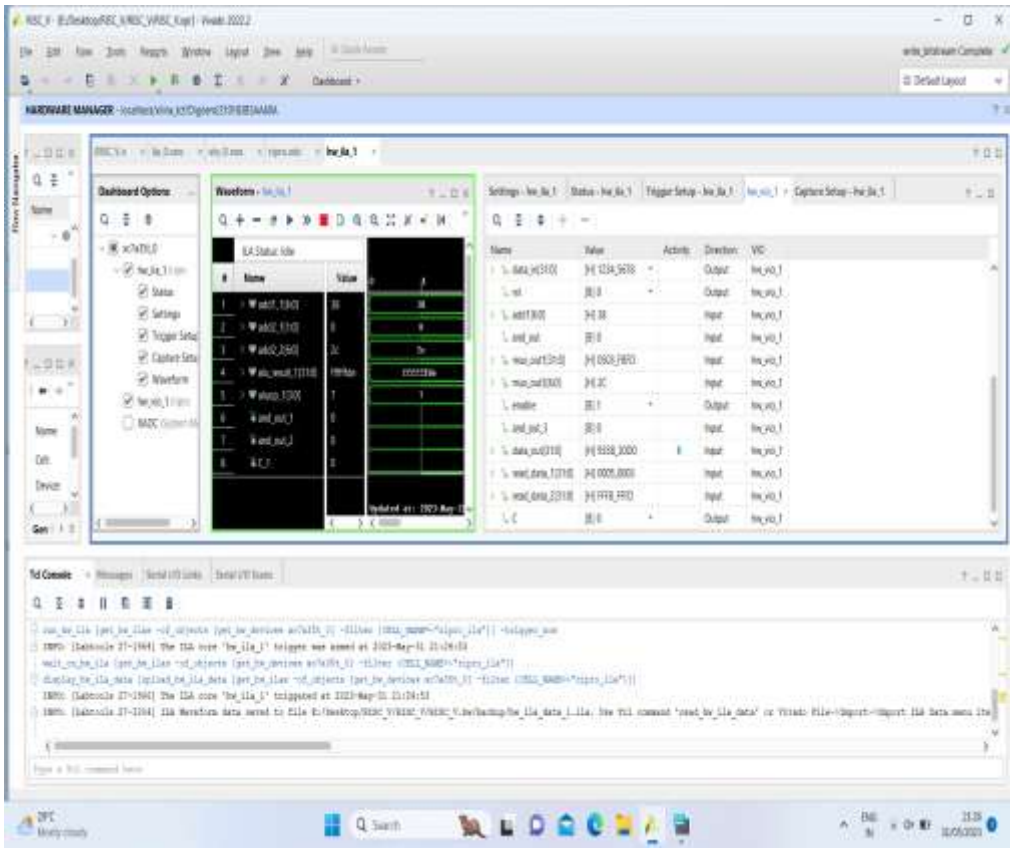


Figure 7: Hardware implementation results of the RISC-V ISA on the Basys FPGA board

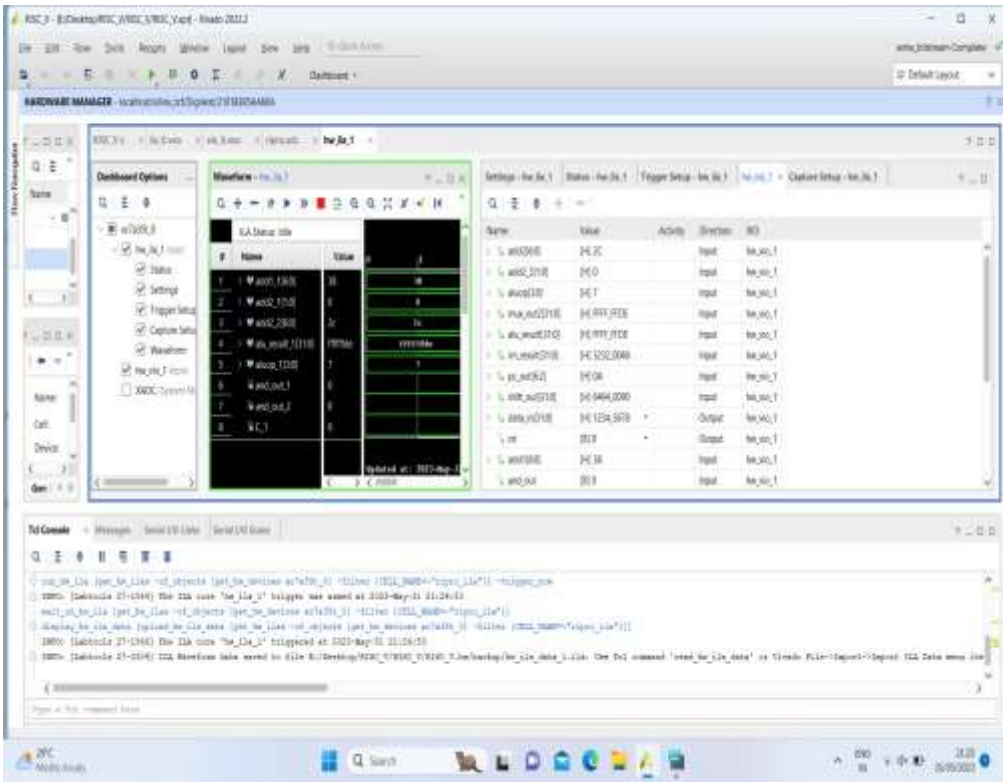


Figure 8: Hardware implementation results of the RISC-V ISA on the Basys FPGA Board (Cont.)

5. CONCLUSION AND FUTURE SCOPE

This work presents the design and implementation of a single-cycle, 32-bit RISC-V processor incorporating an extended instruction set comprising 14 application-specific instructions for complex arithmetic operations. To improve computational efficiency and reduce energy consumption, the power-intensive MULC16-32 instruction is modified by integrating a dedicated Multiply-Accumulate (MAC) unit, resulting in the MAC-MULC16-32 instruction. The dedicated MAC architecture facilitates efficient execution of multiply-accumulate operations, thereby enhancing the processor's capability for DSP-oriented computations. The implementation of an FIR filter using the proposed MAC-enabled processor further demonstrates its effectiveness in executing computationally intensive DSP operations. The results establish the potential of the proposed architecture as a compact and application-specific processing solution for embedded systems and low-power DSP applications.

Future work will focus on further optimizing the processor architecture in terms of power, performance, and area (PPA). The integration of additional DSP-oriented instructions, such as those supporting convolution, Fast Fourier Transform (FFT), and other signal-processing algorithms, can further enhance its computational capabilities. Further improvements may include pipelining and parallel execution to increase instruction throughput, along with clock-gating and other low-power design techniques to minimize dynamic power consumption. The architecture can also be extended to support more complex DSP applications, including digital audio processing, biomedical signal processing, and real-time sensor data analysis. In addition, comprehensive evaluation using different workloads and benchmarking against existing RISC-V and dedicated DSP architectures will help assess its performance, energy efficiency, and suitability for resource-constrained embedded applications.

REFERENCES

1. Hela Belhadj Amor, Carolynn Bernier , and Zdenek Prikryl, “A RISC-V ISA Extension for Ultra-Low Power IoT Wireless Signal Processing,” IEEE Transactions On Computers,Vol. 71, No. 4, April 2022.
2. Andrew Waterman, Krste Asanovic, “The RISC-V Instruction Set Man-ual,” in Unprivileged ISA, Volume I, June 8, 2019.
3. Jayant Chowdhary, Vivek Garg, Tushar Negi, Shreya Jain, “Realization Of An 8-bit Pipelined Microprocessor in Verilog HDL,” IISTE E-Jounal, Vol 3, No.7, 2012.
4. Nischitha D D, K Sanjana Priya, Shivani D, Yashaswini K L and Dr. C M Patil, “Implementation of 32-bit RISC-V Processor,” International Research Journal of Engineering and Technology (IRJET), Volume: 08 Issue: 05, May 2021.
5. J. Tu and L. Van, “Power-efficient pipelined reconfigurable fixed width Baugh-Wooley multipliers,” IEEE Trans. Comput., vol. 58,no. 10, pp. 1346–1355, Oct. 2009.
6. Mamatha R. Maheshwarappa, Student Member, IEEE, Mark D. J. Bowyer, Member, IEEE and Christopher P. Bridges, Member, IEEE, “Improvements in CPU & FPGA Performance for Small Satellite SDR Applications,”IEEE Trans., 2017.
7. Pallavi S and Swamy T N, “SINGLE CYCLE 64-BIT RISC-V PRO-CESSOR AND IT’S FPGA PROTOTYPE,”JETIR, July 2018.
8. Geun-young Jeong and Ju-sung Park, “Design of 32-bit RISC processor and efficient verification,”7th Korea-Russia International Symposium on Science and Technology, Proceedings KORUS 2003. (IEEE Cat. No.03EX737).
9. Hyeokdong Kwon 1, Hyunjun Kim1, (Member, Ieee), Si-Woo Eum 2, Minjoo Sim2, Hyunji Kim1,

Wai-Kong Lee 3, (Member, Ieee), Zhi Hu4, And Hwajeong Seo 2, "Optimized Implementation of SM4 on AVR Microcontrollers, RISC-V Processors, and ARM Processors,"IEEE Access,2022.

10. P. Davide Schiavone et al., "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet of-Things applications," in Proc. 27th Int. Symp. Power Timing Model. Optim. Simul., 2017, pp. 1–8.
11. C. A. R. A. Melo and E. Barros, "Oolong: A Baseband processor extension to the RISC-V ISA," in Proc. IEEE 27th Int. Conf. Appl.-Specific Syst. Architectures Processors, 2016, pp. 241–242.
12. M. Gautschi et al., "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 25, no. 10, pp. 2700–2713, Oct. 2017.