Design and Implementation of a SIMD-Based RISC Processor for Parallel Data Processing

Konapala Neehar Phani, Konduru Praveen Karthik, Taduvai Satvik Gupta
BL.EN.U4EAC21033, BL.EN.U4EAC21035, BL.EN.U4EAC21075

***Abstract*-- This project presents the design and implementation of a single-instruction, multiple-data (SIMD) Reduced Instruction Set Computing (RISC) processor. The system supports vectorized operations, enabling efficient parallel processing for data-intensive tasks. The implementation includes core modules such as an instruction fetch unit, decode unit, SIMD ALU, register file, and control unit, along with a comprehensive testbench to validate functionality. Results demonstrate the processor's capability to perform various arithmetic and logical operations efficiently across multiple data streams simultaneously.**

Key words: **RISC, SIMD, ALU, Parallel Processing**

I. INTRODUCTION

Modern computing applications demand significant computational power to handle data-intensive tasks, ranging from image and video processing to scientific simulations and cryptographic algorithms. Traditional single-core processors often face limitations in efficiently executing these workloads due to their sequential execution nature. To address this, parallel processing architectures have emerged as a solution, enabling simultaneous data computations to enhance performance.

SIMD (Single Instruction, Multiple Data) is a parallel processing architecture that executes a single instruction on multiple data elements simultaneously. This approach is particularly effective for applications requiring repetitive computations on large datasets, such as matrix operations, signal processing, and neural network computations. By dividing data into smaller chunks and processing them concurrently, SIMD significantly boosts processing speed and resource utilization.

RISC (Reduced Instruction Set Computing) complements SIMD by focusing on a simplified instruction set designed for efficient hardware implementation. The RISC architecture minimizes complexity in control logic, leading to faster instruction decoding and execution cycles. This simplicity makes it easier to implement advanced features like SIMD without compromising performance.

This project integrates the principles of SIMD and RISC to design a highly efficient processor capable of executing vectorized operations on 128-bit wide data. The processor's architecture consists of several modular components, including: Fetch, Decode, SIMD ALU, Register Files, Control Unit.

The modular design of the processor facilitates easy verification and potential scalability to wider data widths or more complex instruction sets. By leveraging SIMD for parallelism and RISC for simplicity, the processor achieves a balanced trade-off between computational power and architectural complexity.

This design was implemented and verified using a comprehensive simulation framework. The processor's functionality

was validated through a series of test cases, demonstrating its capability to execute parallel operations efficiently. The project underscores the significance of SIMD and RISC principles in modern processor design, paving the way for scalable, high-performance architectures in next-generation computing.

II. Methodology

The system consists of the following key components:

1. **Instruction Fetch Unit:**
   Fetches instructions sequentially from a memory block. Employs a 32-bit program counter to track instruction flow.
2. **Instruction Decode Unit:**
   Decodes 32-bit instructions into operation code (opcode), source registers, and destination registers.
3. **Register File:**
   A bank of 32 registers, each 128 bits wide. Supports concurrent read and write operations based on decoded instruction fields.
4. **SIMD ALU:**
   Performs arithmetic and logical operations in parallel on four 32-bit elements of the 128-bit vectors. Supported operations include ADD, SUB, MUL, AND, OR, XOR, NAND, NOR, XNOR, DIV, EQ, GT, LT.
5. **Control Unit:**
   Orchestrates the execution flow by integrating the fetch, decode, register file, and ALU modules.
6. **Testbench:**
   Validates the processor using predefined instructions and vectorized data. Monitors the output for correctness and performance analysis.

Table.1: SIMD RISC System Architecture Overview

Component	Inputs	Outputs	Description / Notes	Next Step
Instruction Fetch	clk, reset	instruction [31:0]	Fetches instructions	Instruction Decode
Instruction Decode	instruction [31:0]	opcode [3:0], src1 [4:0],	Decodes the instruction and extracts key parts	Register File
		src2 [4:0], dest [4:0]		
Register File	clk, reset, src1, src2, dest, write_enable, write_data [31:0]	read_data1 [31:0], read_data2 [31:0]	Reads data from registers based on input sources	SIMD ALU
SIMD ALU	clk, reset, opcode [3:0], operand1 [127:0], operand2 [127:0]	result [127:0]	Performs arithmetic/logical operations; supports ADD, SUB, MUL, AND, OR, XOR, NAND, NOR, XNOR, DIV, EQ, GT, LT	Control Unit
Control Unit	-	Updates pc (Program Counter)	Manages interaction among all components	-

III. Implementation

The design was implemented using Verilog HDL, adhering to modular programming principles. The modules were interconnected to ensure seamless data flow and synchronization. Key implementation features include:

1. **Vectorized Operations:**

The SIMD ALU splits 128-bit operands into four 32-bit chunks, executing operations in parallel for improved efficiency.

2. **Instruction Memory Initialization:**

Instructions are loaded using a .mem file to simplify testing and scalability.

3. **Clock and Reset Mechanisms:**

Ensures proper synchronization across modules, with a 50% duty cycle clock for consistent timing.

4. **Pipeline Execution:**

Instruction execution is streamlined through sequential fetch, decode, and execute stages.

IV. Results

The proposed SIMD-based RISC processor was implemented and tested using modular Verilog components, and its functionality was validated through simulation. The results highlight the processor's ability to execute parallel vectorized operations efficiently, confirming its suitability for applications requiring high-throughput computation. The following sections detail the results and analysis of the processor's performance and functionality.

1. Functionality Testing of Each Module

Instruction Fetch Unit:

- The instruction fetch unit successfully retrieved sequential instructions from the memory during the simulation. The program counter (PC) was updated correctly with each clock cycle, ensuring a continuous flow of instructions.
- Validation was performed by monitoring the instruction output signal for correctness.

Instruction Decode Unit:

- The decode unit correctly parsed the 32-bit instructions into opcode, source, and destination registers. This parsing ensured accurate control signals for downstream modules.

Example: For the instruction 32'b0000_00001_00010_00011 (ADD operation), the decoded signals were:

  - Opcode: 0000
  - Source Registers: 00001 (R1), 00010 (R2)
  - Destination Register: 00011 (R3)

Table.2: Opcode and Operations

Opcode	Operation
0000	ADD
0001	SUB
0010	MUL
0011	AND
0100	OR
0101	XOR
0110	NAND
0111	NOR
1000	XNOR
1001	DIV
1010	EQ
1011	GT
1100	LT

SIMD ALU:

- The SIMD ALU executed arithmetic and logical operations on 128-bit vector operands. The operations were performed on four 32-bit elements in parallel, demonstrating the processor's SIMD capabilities.
- The Implemented Operations in this SIMD processor are: ADD, SUB, MUL, AND, OR, XOR, NAND, NOR, XNOR, DIV, EQ, GT, LT.
- Test case results for few vectorized operations:

Addition (ADD):

- Inputs: R1 = {20, 15, 10, 5}, R2 = {12, 9, 6, 3}
- Output: R3 = {32, 24, 16, 8}

Subtraction (SUB):

- Inputs: R1 = {20, 15, 10, 5}, R2 = {12, 9, 6, 3}
- Output: R4 = {8, 6, 4, 2}

Bitwise AND (AND):

- Inputs: R1 = {20, 15, 10, 5}, R2 = {12, 9, 6, 3}
- Output: R6 = {4, 9, 2, 1}

Operation	Input1	Input2	result	Designation Register
ADD	{20, 15, 10, 5}	{12, 9, 6, 3}	{32, 24, 16, 8}	R3
SUB	{20, 15, 10, 5}	{12, 9, 6, 3}	{8, 6, 4, 2}	R4
MUL	{20, 15, 10, 5}	{12, 9, 6, 3}	{240,135,60,15}	R5
AND	{20, 15, 10, 5}	{12, 9, 6, 3}	{4, 9, 2, 1}	R6
OR	{20, 15, 10, 5}	{12, 9, 6, 3}	{28, 15, 14, 7}	R7
XOR	{20, 15, 10, 5}	{12, 9, 6, 3}	{24, 6, 12, 6}	R8

Table.3: Example Output

Register File:

- The register file provided high-speed access to data and successfully stored results from the ALU.
- Validation was performed by monitoring the read_data1 and read_data2 signals during instruction execution and confirming the correctness of write_data to destination registers.

Control Unit:

- The control unit orchestrated the operation of all modules, ensuring proper sequencing of fetch, decode, execute, and write-back stages.
- The integration of the control unit demonstrated seamless operation, with no delays or data hazards during simulation.

The current design of the SIMD-based RISC processor has several limitations that provide avenues for improvement. It lacks advanced pipelining stages, which are crucial for enhancing instruction throughput and minimizing execution delays. Additionally, the instruction set is limited to six basic operations, making it less versatile compared to modern processors with extensive instruction capabilities. Another notable limitation is the absence of support for floating-point arithmetic, restricting its use in applications requiring high precision.

Future enhancements can address these limitations by extending the SIMD width to process larger vectors, enabling higher computational efficiency. Incorporating pipelining and hazard detection mechanisms can significantly improve throughput and streamline execution. Expanding the instruction set to include more complex operations, such as division and transcendental functions like sine and cosine, would increase the processor's versatility.

Furthermore, implementing floating-point arithmetic units would enable the processor to handle a broader range of workloads, making it more suitable for diverse computational tasks.

Fig.1: ModelSim Waveform

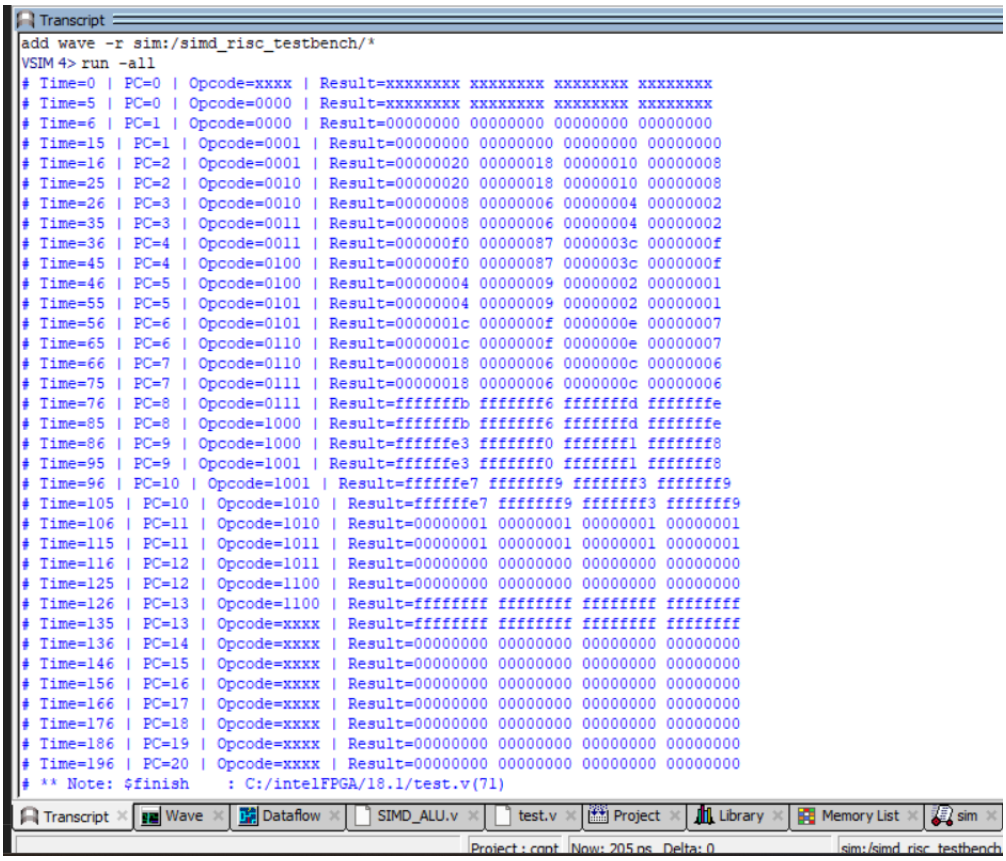

Fig.2: ModelSim Terminal Hexadecimal Output

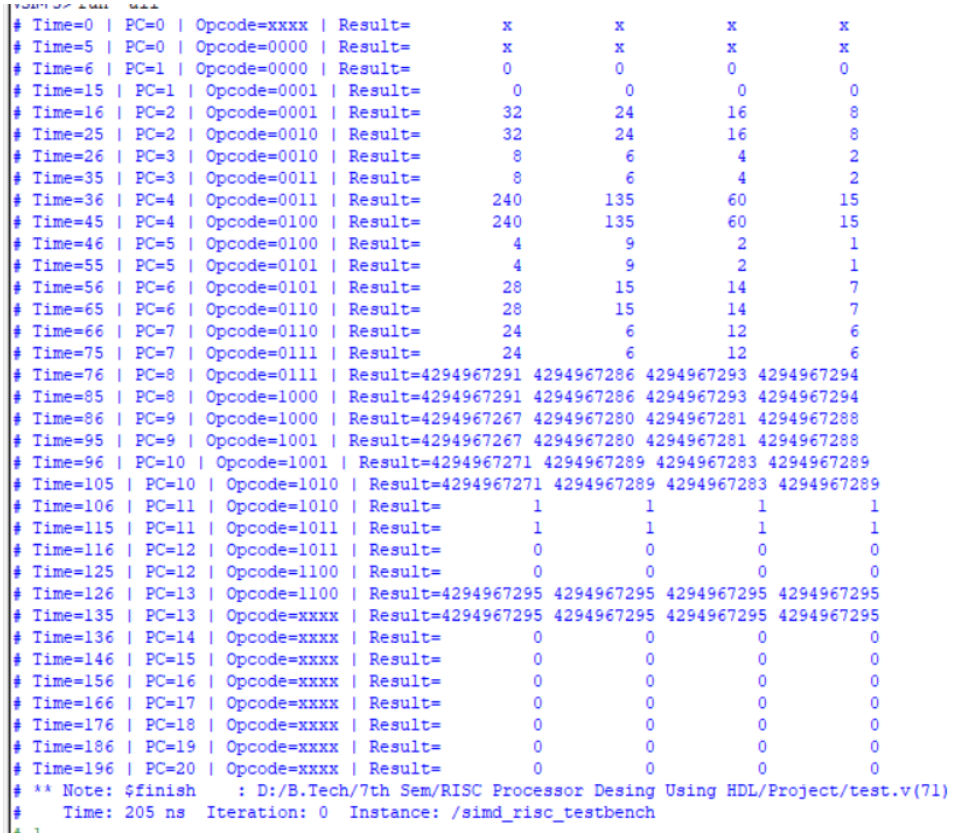

Fig.3: ModelSim Terminal Decimal Output

V. CONCLUSION

The proposed SIMD-based RISC processor was implemented and tested using modular Verilog components, and its functionality was validated through simulating in ModelSim. The results highlight the processor's ability to execute parallel vectorized operations efficiently, confirming its suitability for applications requiring high-throughput computation. The following sections detail the results and analysis of the processor's performance and functionality.