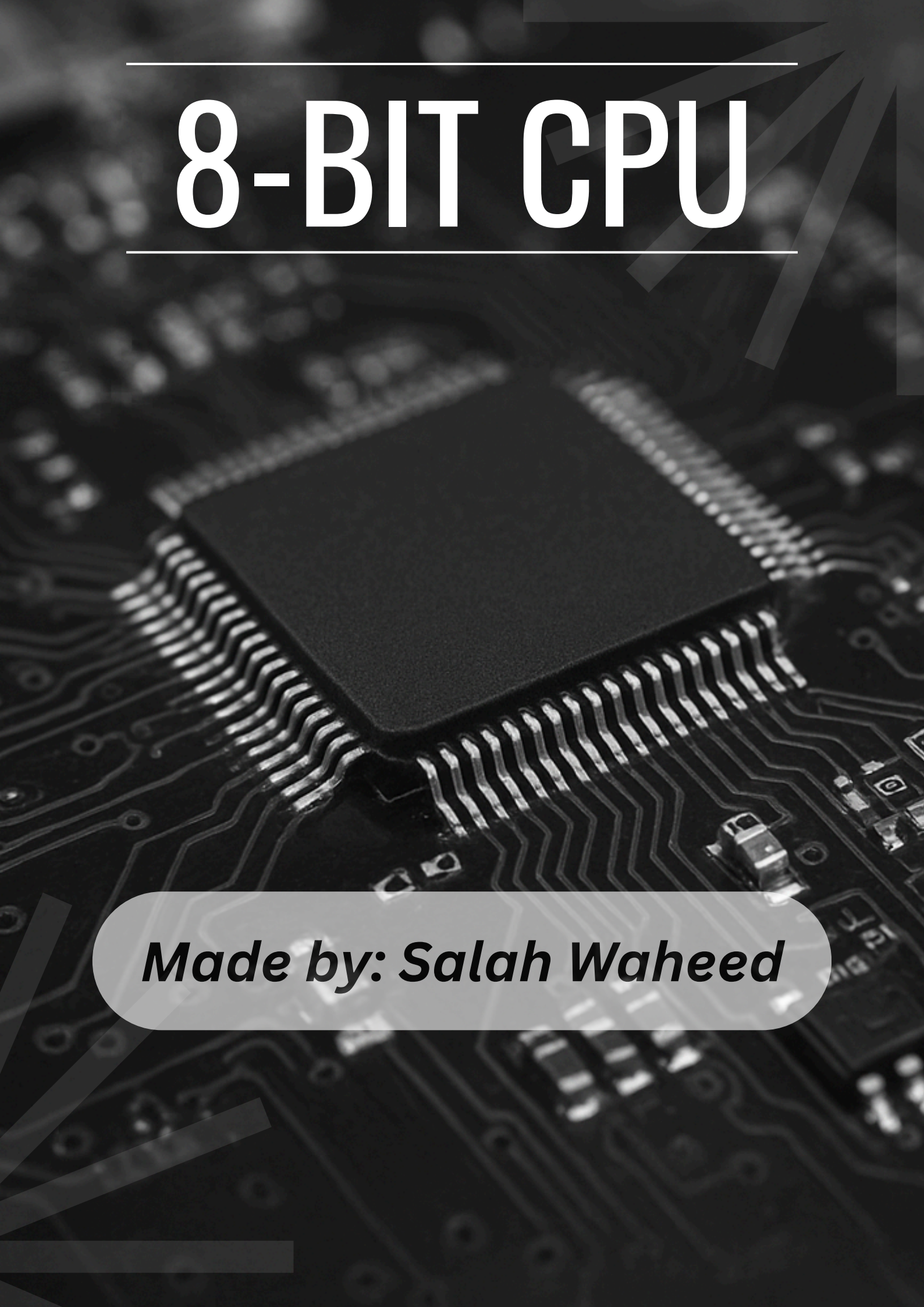
8-BIT CPU



Made by: Salah Waheed

Table of Contents:

Abstract:

This project presents the design and implementation of a custom 8-bit Central Processing Unit (CPU) using Verilog HDL. The CPU is structured in a highly modular fashion and integrates several well-defined components: a Control Unit (CU), Program Counter (PC), Instruction Register (IR), Instruction Memory, Register File (REG), Arithmetic Logic Unit (ALU), Status Register, and a unifying Datapath. Each of these modules is designed to perform a specific role within the instruction cycle, and together they form a cohesive system capable of executing a simplified instruction set.

The CPU supports fundamental arithmetic operations (ADD, SUB, MUL, DIV), logical operations (AND, OR, XOR), and bitwise shifts/rotates (LSL, LSR, ROL, ROR). In addition, it implements control-flow instructions such as unconditional jumps (JMP) and data movement instructions such as load immediate (LDI). This set of operations allows the CPU to execute basic computational tasks while maintaining a relatively small design footprint, making it both educational and practical for FPGA-based experiments.

Verification of the CPU was carried out through an extensive Verilog testbench. Simulation waveforms were analyzed to confirm that instructions were correctly fetched, decoded, and executed, with proper updates to the register file and status flags. The test programs included sequences of arithmetic, logical, and control-flow operations to validate both individual modules and their integration within the datapath.

The modular nature of this CPU makes it highly scalable and extensible. It provides a strong foundation for enhancements such as conditional branching instructions (e.g., JZ, JC, JNZ), stack operations (PUSH, POP), subroutine call/return mechanisms, and memory-mapped I/O for peripheral communication. These potential extensions highlight the adaptability of the design and its suitability for teaching, experimentation, and prototyping in digital systems and computer architecture courses.

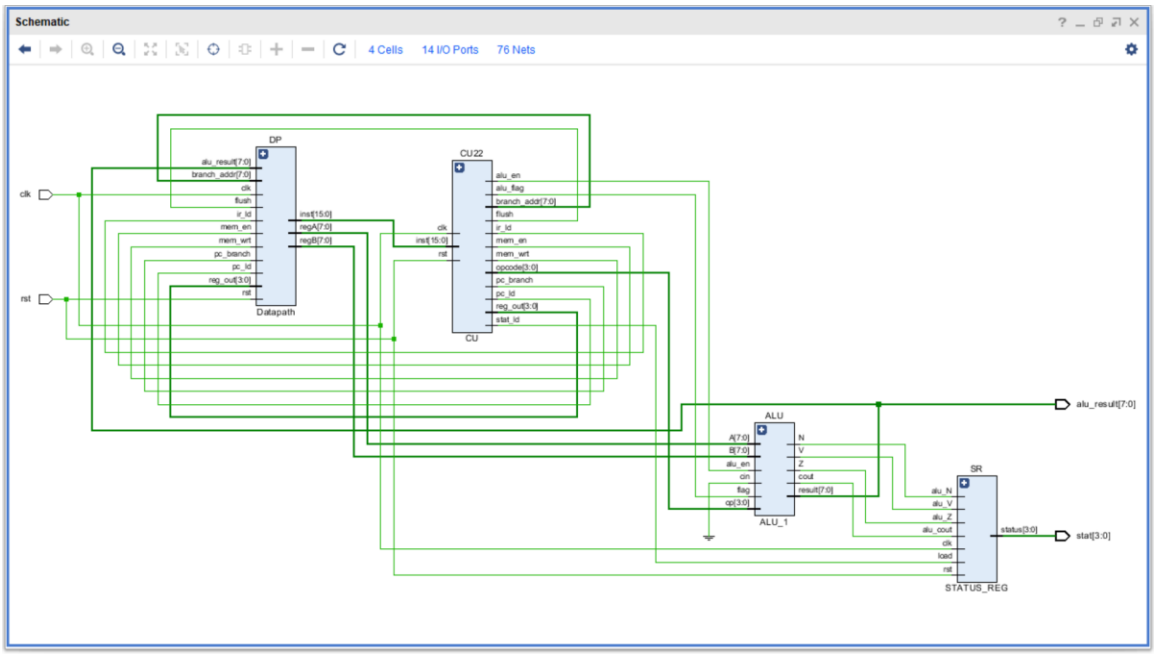
Introduction:

The Central Processing Unit (CPU) is the brain of a digital system, responsible for fetching, decoding, and executing instructions. In this project, we design a simplified 8-bit CPU to demonstrate the fundamental principles of instruction execution in hardware. The design emphasizes modularity, with each block implemented as a separate Verilog module.

Instruction flow in the CPU:
1. The Program Counter (PC) Points to the next address of the instruction to be executed.
2. The Instruction Memory (InstMem) provides the instruction at that address.
3. The Instruction Register (IR) stores the fetched instruction.
4. The Control Unit (CU) decodes the instruction and generates control signals.
5. Operands are read from the Register File (REG).
6. The Arithmetic Logic Unit (ALU) executes arithmetic/logical operations.
7. The Status Register updates flags (C, V, N, Z).
8. Results are written back to the Register File.

This CPU supports a reduced instruction set:
- Arithmetic: ADD, SUB, MUL, DIV
- Logical: AND, OR, XOR
- Shifts and Rotates: LSL, LSR, ROL, ROR
- Control: JMP (unconditional branch)
- Data Movement: LDI (Load Immediate)

CPU Block Diagram and Components:



The CPU design is broken down into distinct components, each performing a specific function:

1. Control Unit (CU):

The Control Unit (CU) orchestrates the CPU by generating the control signals required for each instruction. It implements a finite state machine (FSM) with three stages: FETCH, DECODE, and EXECUTE.

- FETCH: Loads the next instruction from memory into the Instruction Register (IR).
- DECODE: Extracts the opcode and register fields from the instruction.
- EXECUTE: Activates the appropriate datapath signals to carry out the operation.

The CU also manages control-flow instructions like JMP by asserting the branch and flush signals, ensuring that the CPU jumps to the correct program address without executing stale instructions.

2. Program Counter (PC):

The Program Counter (PC) is responsible for keeping track of the next instruction address. It increments sequentially after each instruction fetch, unless a jump instruction is encountered. In that case, the PC is loaded with the branch address, effectively redirecting program execution.

The PC has three modes:
- Reset: Clears the PC to zero.
- Increment: Adds one to the PC during normal execution.
- Branch: Loads a new address during JMP.

This simple yet effective design allows the CPU to support sequential instruction flow and jumps.

3. Instruction Register (IR):

The Instruction Register (IR) holds the instruction fetched from memory until it is decoded and executed. It acts as a buffer between the Instruction Memory and the Control Unit. The IR is cleared during reset or when a flush signal is asserted (for example, after a JMP instruction). By isolating the fetched instruction, the IR ensures that decoding and execution stages always work with a stable copy of the instruction.

4. Instruction Memory (InstMem):

Instruction Memory stores the program that the CPU executes. In this project, the memory is initialized with test instructions covering arithmetic, logical, load immediate, and jump operations. It is modeled as a simple array of 16-bit words, with the address provided by the Program Counter. For synthesis on hardware, this module could be implemented as ROM or block RAM.

5. Register File (REG):

The Register File provides fast storage for operands and results. It contains 16 general-purpose registers, each 8 bits wide. The REG supports two asynchronous read ports and one synchronous write port, enabling the CPU to fetch two operands and store one result in a single cycle. Register zero and one are typically initialized with test values using the LDI instruction at the start of the program.

6. Full Adder Unit (FUA):

The Full Adder Unit (FUA) is a fundamental component inside the Arithmetic Logic Unit (ALU). It is designed to perform both addition and subtraction operations efficiently. Instead of building two separate circuits for addition and subtraction, the FUA cleverly reuses the same logic with the help of operand inversion and carry control.

- Addition**:** The operands `A` and `B` are added directly, along with the carry-in.
- Subtraction**:** Operand `B` is inverted and the carry-in is complemented, effectively performing `A - B` using two's complement arithmetic.

This modular approach reduces hardware complexity and allows the ALU to support arithmetic operations without duplicating resources

7. Arithmetic Logic Unit (ALU):

The ALU performs all arithmetic and logical operations. It takes two 8-bit operands as inputs, along with a control signal (opcode) from the CU, and produces an 8-bit result. Supported operations include ADD, SUB, MUL, DIV, AND, OR, XOR, shifts, and rotates. The ALU also generates four status flags:
• Carry (C): Indicates carry-out from addition or borrow in subtraction.
• Overflow (V): Indicates signed arithmetic overflow.
• Negative (N): Set if the result is negative.
• Zero (Z): Set if the result equals zero.

These flags are stored in the Status Register for later use by conditional instructions (future extension).

8. Status Register:

The Status Register holds the condition flags {C, V, N, Z}. It is updated after every arithmetic or logical operation if the CU asserts the status load signal. By storing these flags, the CPU can support conditional branches in future versions (e.g., jump if zero, jump if carry).

9. Datapath:

The Datapath ties all components together. It connects the Control Unit, PC, IR, Instruction Memory, Register File, ALU, and Status Register. A write-back multiplexer ensures that either ALU results or immediate values from instructions can be stored in the Register File. The datapath executes instructions in a synchronized sequence under the direction of the CU.

10. CPU (Top Module):

The CPU top-level module is the backbone of the design. It integrates all the building blocks together:

- Control Unit (CU): Generates control signals to manage the instruction cycle (fetch, decode, execute).
- Datapath: Connects the Program Counter (PC), Instruction Register (IR), Register File, and handles operand flow.
- ALU: Executes arithmetic and logic operations based on the opcode.
- Status Register: Stores status flags (Carry, Overflow, Negative, Zero) for future conditional instructions.

The CPU module wires up all these components, ensuring smooth communication and correct sequencing of instructions. It provides the interface between internal components and the external environment, such as testbenches or FPGA hardware.

Simulation Waves:

Simulation was carried out using a Verilog testbench. The instruction memory was preloaded with test programs including LDI, ADD, SUB, MUL, DIV, AND, OR, XOR, shifts, rotates, and JMP. The simulation verified the following:

- LDI correctly loads immediate values into registers.
- Arithmetic instructions produce correct results and update flags.
- Logical and shift/rotate instructions behave as expected.
- JMP flushes the pipeline and redirects execution to the branch address.

Waveforms captured during simulation confirm correct sequencing of control signals, correct operand fetching, and proper register file updates. The testbench also demonstrated that multiple instructions can be chained together to form small programs.

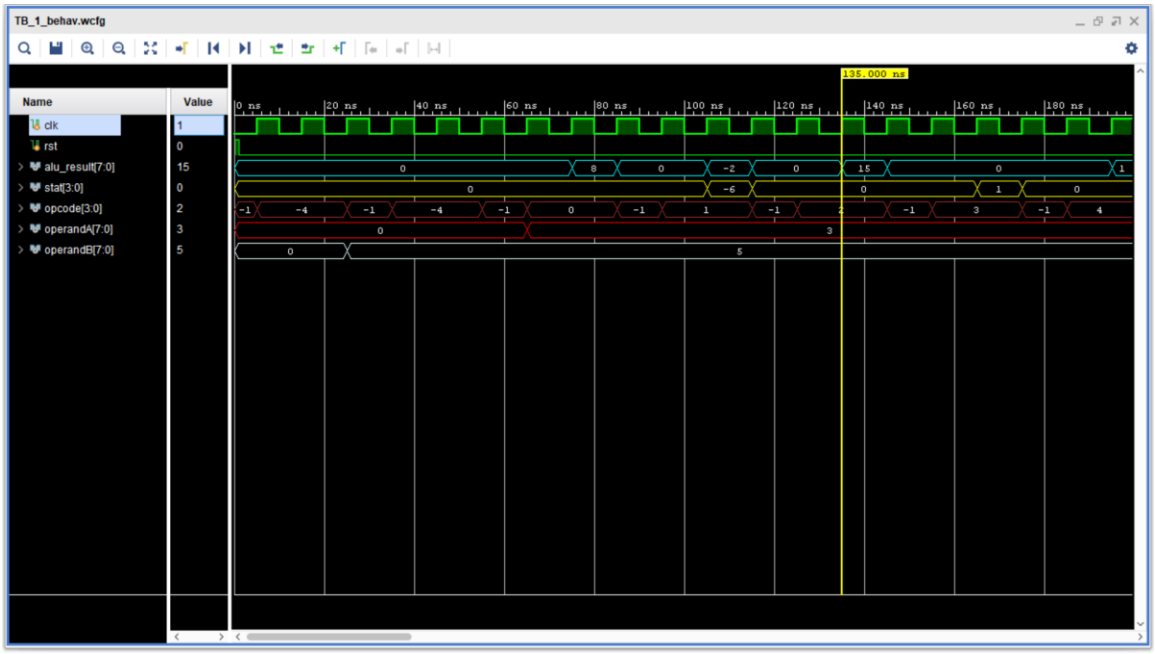


Figure 1: Wave showing the LDI, ADD, SUB, MULTI

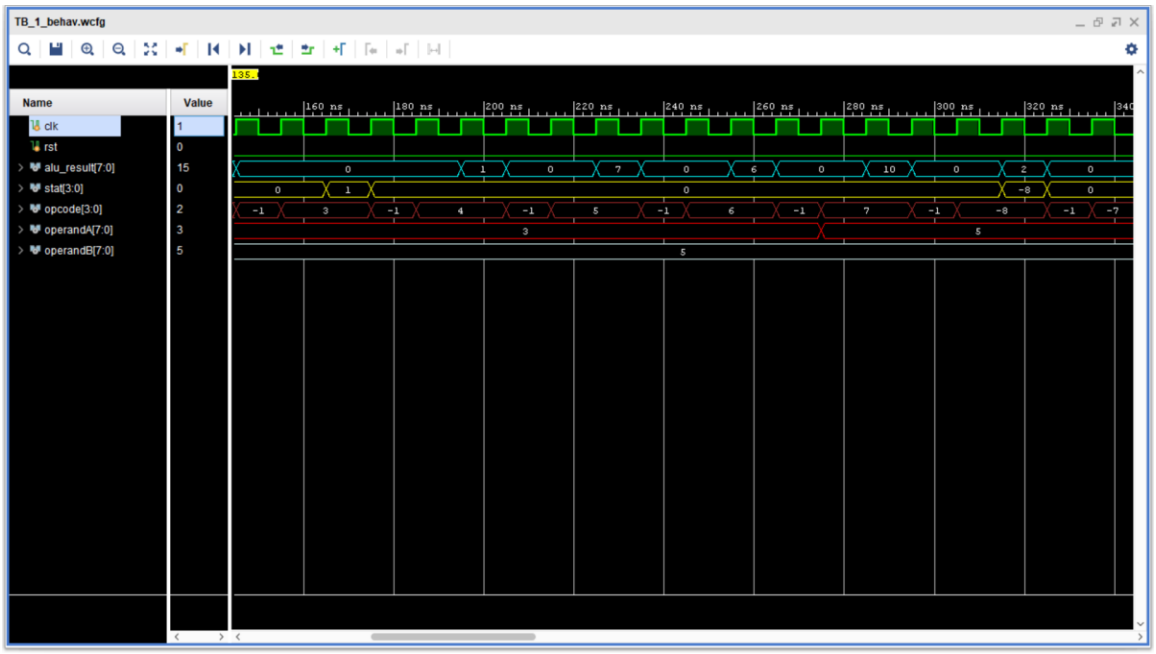


Figure 2: Wave showing the DIV, OR, XOR, AND, LSL

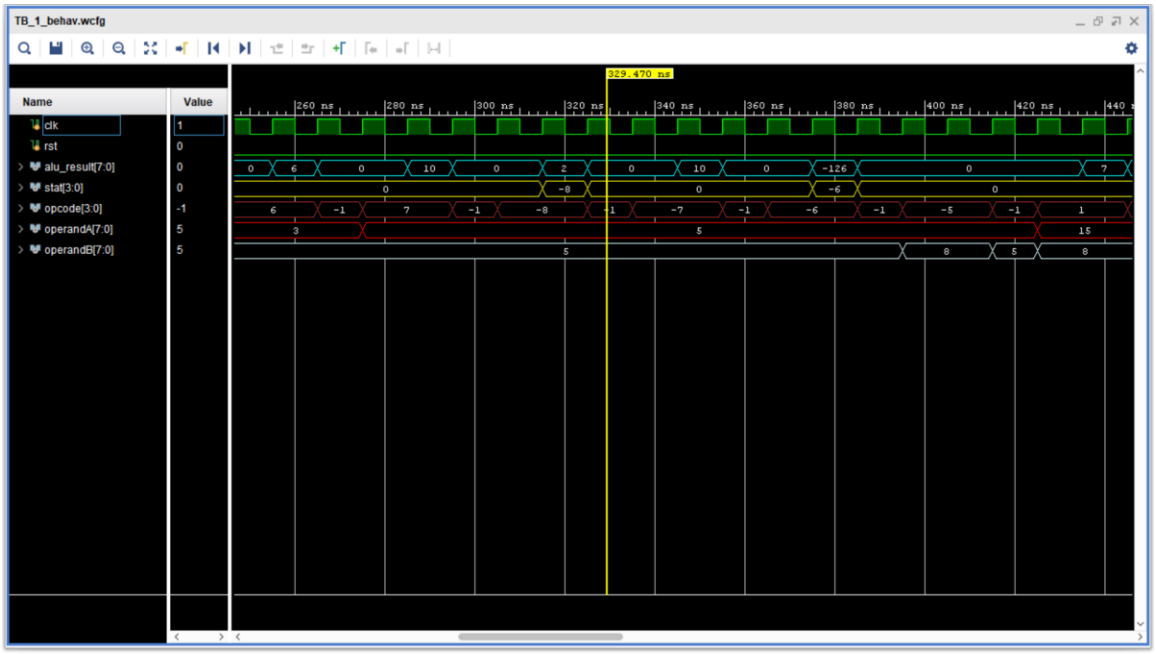


Figure 3: Wave showing the LSR, ROL, ROR, JMP, SUB

Conclusion:

This project successfully designed and implemented a simple 8-bit CPU using Verilog HDL. The modular design made it easier to test and verify each component individually before integrating them. Simulation results validated the correct functioning of the CPU for arithmetic, logical, data movement, and control flow instructions.

Future work can extend the design with:
- Conditional branching based on status flags.
- Stack operations with push and pop instructions.
- Subroutine call and return mechanisms.
- Support for memory-mapped I/O.

This project provides a strong foundation for students and developers interested in CPU design, and demonstrates how digital systems can be constructed incrementally from smaller reusable components.