COMPUTER SYSTEM ARCHITECTURE

LAB REPORT #03

“Custom Processor”

Prepared by:
SHAZIL 532263

Presented To:
LE USAMA SHAUKAT

NUST CEME
DEPT. COMPUTER ENGINEERING

TABLE OF CONTENT

Lab#2 Custom Processor Design (Part 2)

Objective

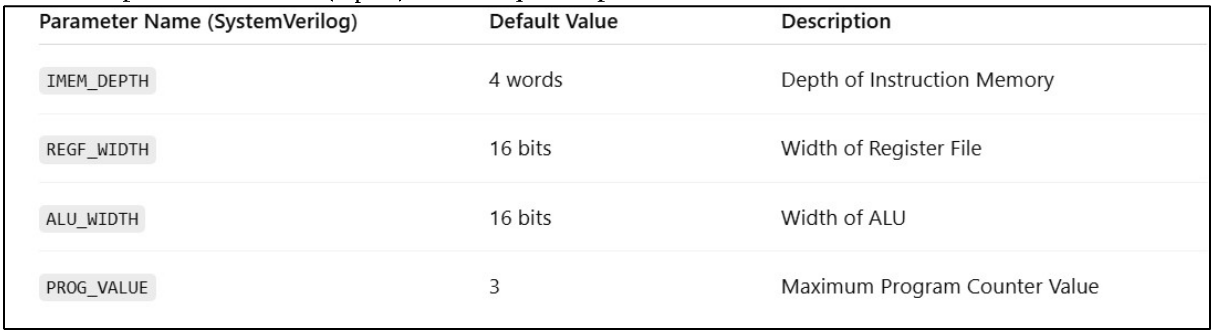
The purpose of this lab is to sequentially design and implement a custom microprocessor.

Tools Required

- System Verilog
- Xilinx Vivado

Lab Task Guidelines

- The implementation must follow a **modular approach**. Each task should be written as a **separate module** and instantiated in the top-level file.
- The design must be **parameterized**, with all parameters defined and passed from the top-level module (top.sv). The required parameters are listed below:



Parameter Name (SystemVerilog)	Default Value	Description
`IMEM_DEPTH`	4 words	Depth of Instruction Memory
`REGF_WIDTH`	16 bits	Width of Register File
`ALU_WIDTH`	16 bits	Width of ALU
`PROG_VALUE`	3	Maximum Program Counter Value

Introduction to Processor Design

To design a simple microprocessor, the most essential foundation is the **Instruction Set Architecture (ISA)**. An ISA defines the collection of instructions that a processor can interpret and execute. In simple terms, it serves as the agreement between **software** (programs) and **hardware** (the processor).

Core Components of a Processor

- **Register File**
  Registers provide very fast data access but can only store a limited amount of information. They act as temporary storage locations, holding values the processor needs immediately for ongoing operations.
- **Control Unit (CU)**
  The control unit acts like the **orchestrator** of the processor. It fetches instructions from memory, decodes them to determine the required action, and then coordinates other units (such as the ALU) to carry out the operation.
- **Arithmetic Logic Unit (ALU)**
  The ALU is the computational powerhouse of the processor. It performs arithmetic tasks (like addition or subtraction) and logical operations (such as AND, OR, NOT, and comparisons). All calculations directed by the control unit are executed here.

- **Memory**
  While technically external to the processor, memory (e.g., RAM or storage) is essential for its functioning. Instructions and data are fetched from memory, processed within the registers and ALU, and the results may be written back to memory afterward.

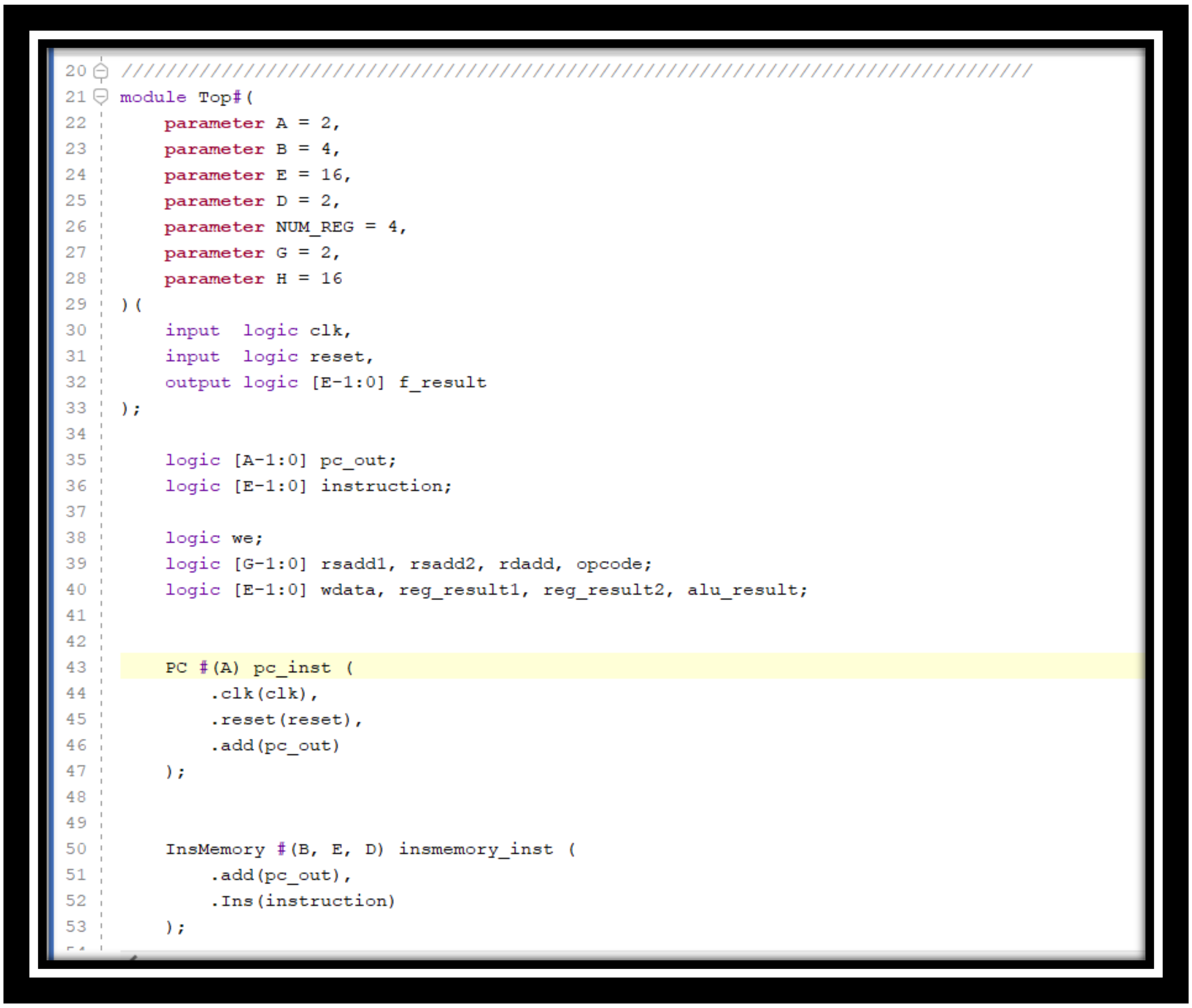
Softwares used:

- System Verilog
- Xilinx Vivado

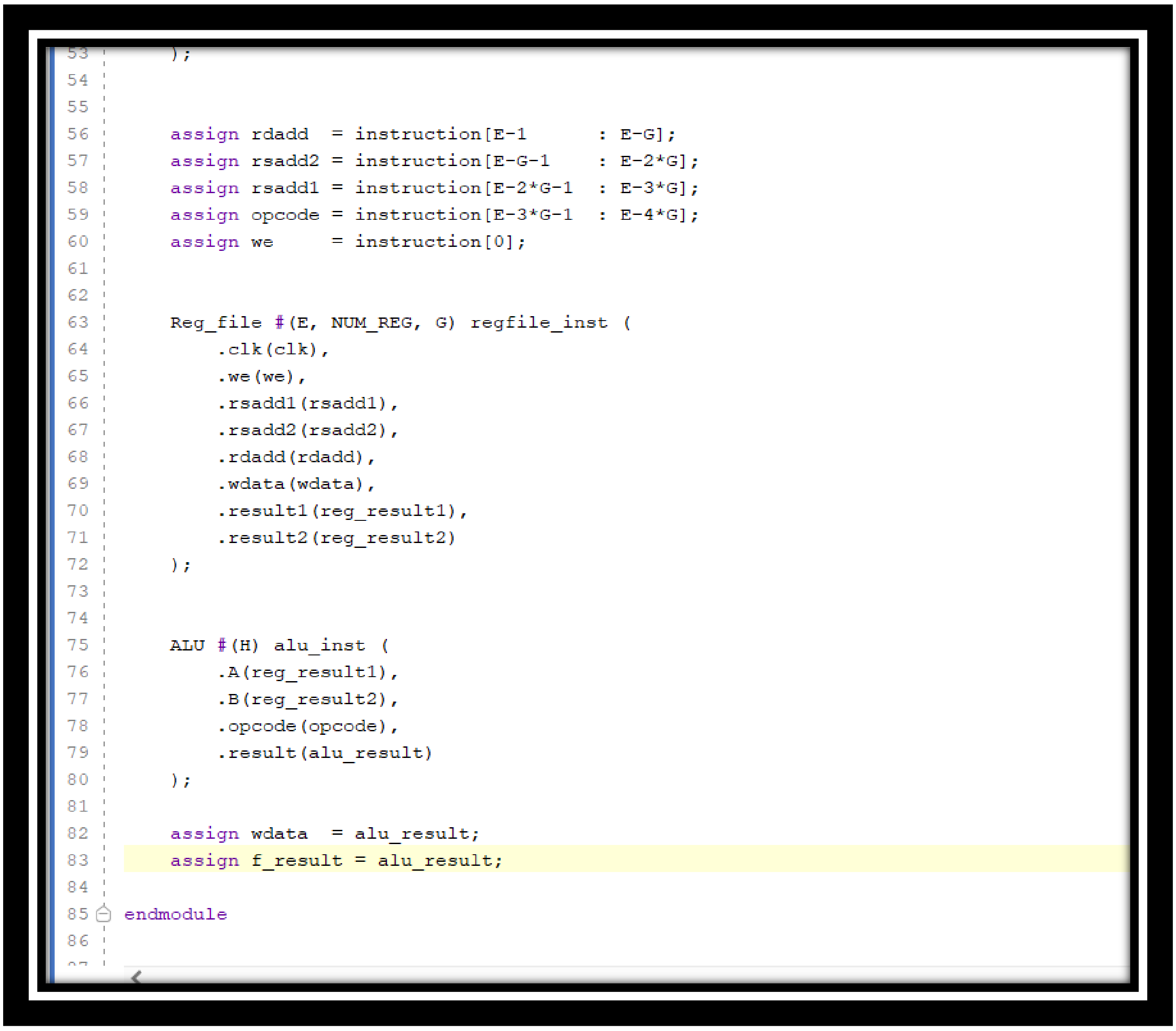
Lab Task 5:

Implement the Top Module while instantiating all the modules which we had made in last lab and do test them through testbench.

Code:

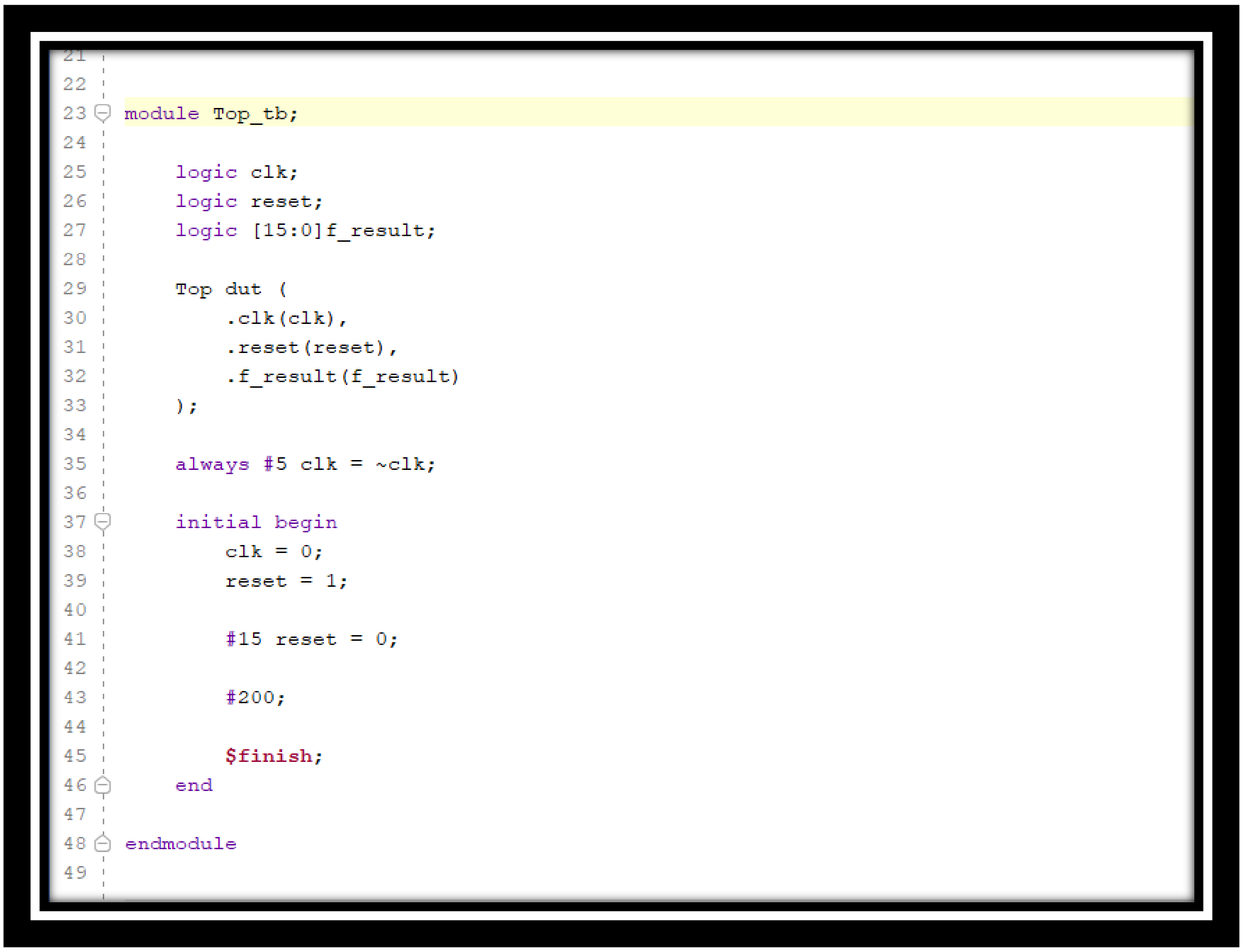


```
/////////////////////////////////////////////////////////////////////////////////
module Top#(
    parameter A = 2,
    parameter B = 4,
    parameter E = 16,
    parameter D = 2,
    parameter NUM_REG = 4,
    parameter G = 2,
    parameter H = 16
)(
    input  logic clk,
    input  logic reset,
    output logic [E-1:0] f_result
);

    logic [A-1:0] pc_out;
    logic [E-1:0] instruction;

    logic we;
    logic [G-1:0] rsadd1, rsadd2, rdadd, opcode;
    logic [E-1:0] wdata, reg_result1, reg_result2, alu_result;


    PC #(A) pc_inst (
        .clk(clk),
        .reset(reset),
        .add(pc_out)
    );


    InsMemory #(B, E, D) insmemory_inst (
        .add(pc_out),
        .Ins(instruction)
    );
```

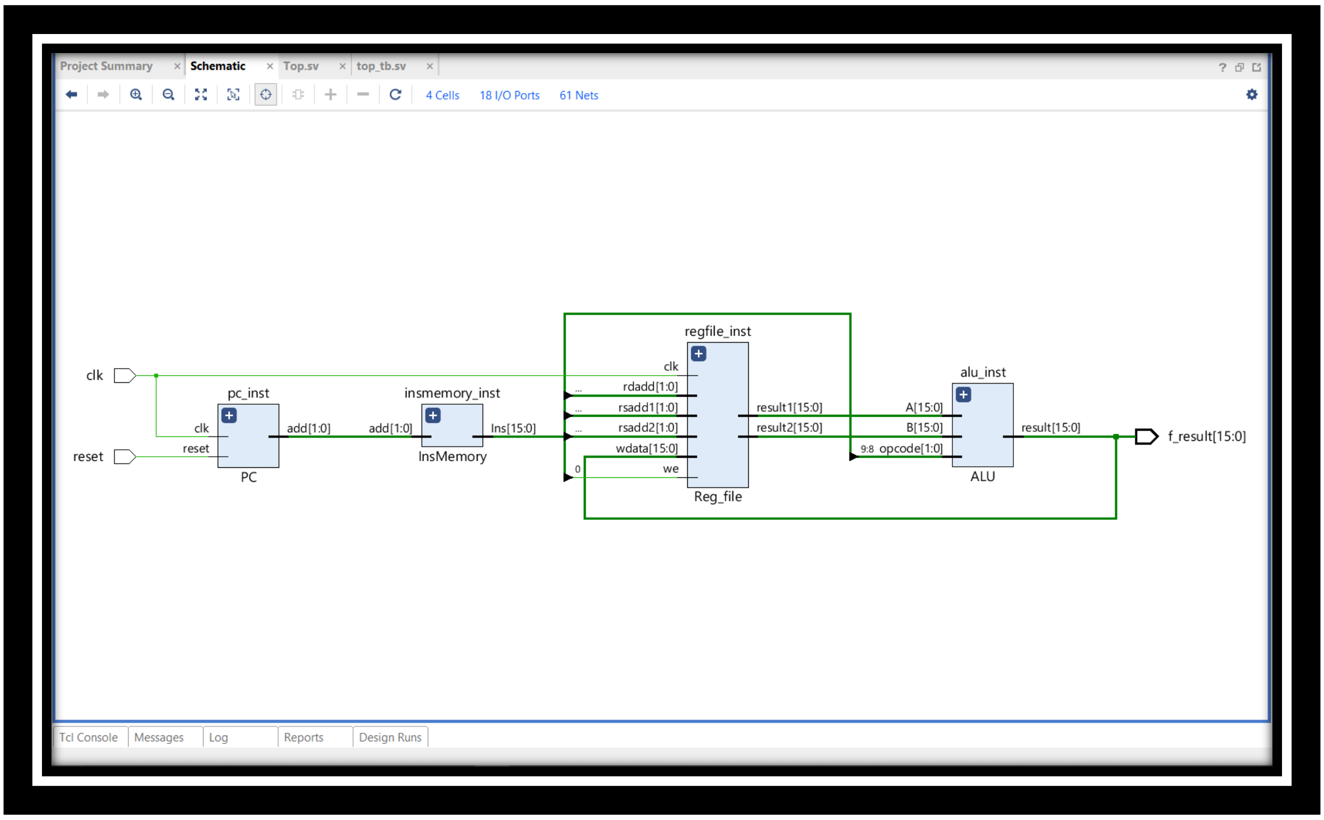


```
    );


    assign rdadd  = instruction[E-1      : E-G];
    assign rsadd2 = instruction[E-G-1    : E-2*G];
    assign rsadd1 = instruction[E-2*G-1  : E-3*G];
    assign opcode = instruction[E-3*G-1  : E-4*G];
    assign we     = instruction[0];


    Reg_file #(E, NUM_REG, G) regfile_inst (
        .clk(clk),
        .we(we),
        .rsadd1(rsadd1),
        .rsadd2(rsadd2),
        .rdadd(rdadd),
        .wdata(wdata),
        .result1(reg_result1),
        .result2(reg_result2)
    );


    ALU #(H) alu_inst (
        .A(reg_result1),
        .B(reg_result2),
        .opcode(opcode),
        .result(alu_result)
    );

    assign wdata  = alu_result;
    assign f_result = alu_result;

endmodule

```

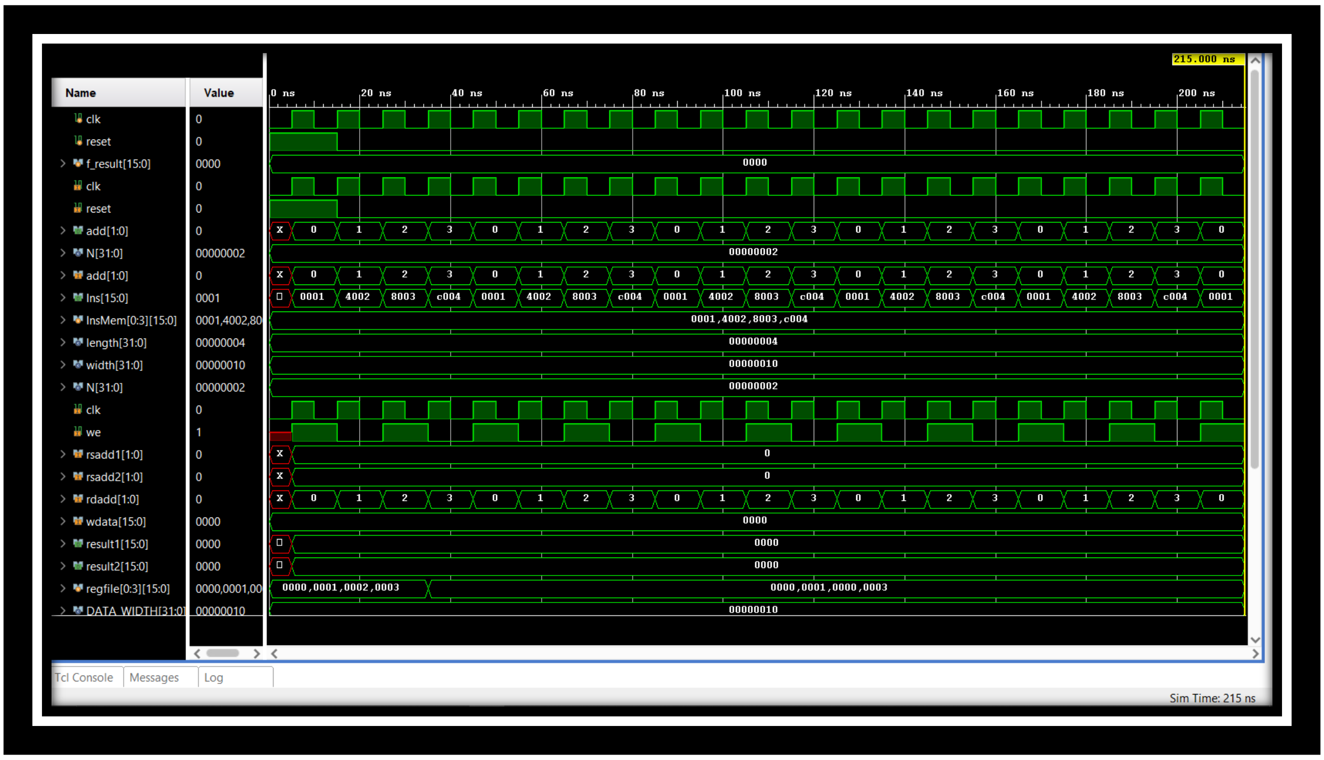
TESTBENCH CODE:


```
module Top_tb;

    logic clk;
    logic reset;
    logic [15:0]f_result;

    Top dut (
        .clk(clk),
        .reset(reset),
        .f_result(f_result)
    );

    always #5 clk = ~clk;

    initial begin
        clk = 0;
        reset = 1;

        #15 reset = 0;

        #200;

        $finish;
    end

endmodule
```

SCHEMATICS:



SIMULATIONS:



CONCLUSION:

The top module was successfully implemented by instantiating all previously designed modules. The integration verified correct connectivity between ALU, register memory, instruction memory, and program counter. The testbench confirmed that data flow and control signals worked as expected. The modular approach simplified debugging and ensured reusability of individual components. This task demonstrated the importance of hierarchical design in building complex digital systems. It also reinforced practical skills in Verilog coding, simulation, and verification using testbenches.