Computer System Architecture

Project Report

Course Instructor: Dr. Muhammad Yasin

Lab Engineer: Malaika Awais

Project Title:

Pipelined Processor with Floating Point Accelerator.

Group Members:

Wahab Sohail	**418159**
Talha Razzaq	**418013**
Ahmad Zeeshan	**403866**

Degree/ Syndicate: 44 /A

Submission Date:
4-6-2024

Executive Summary:

In this project, we have developed a five-stage pipelined processor with proper hazard handling, including stalls and forwarding, capable of executing R-type, I-type, and J-type instructions. Additionally, we integrated a floating-point accelerator as a coprocessor, capable of performing addition, subtraction, multiplication, and division, and implemented the entire system on an FPGA. Our systematic design methodology included simulation, synthesis, and hardware implementation, incorporating advanced hazard detection and mitigation techniques for efficient instruction execution.

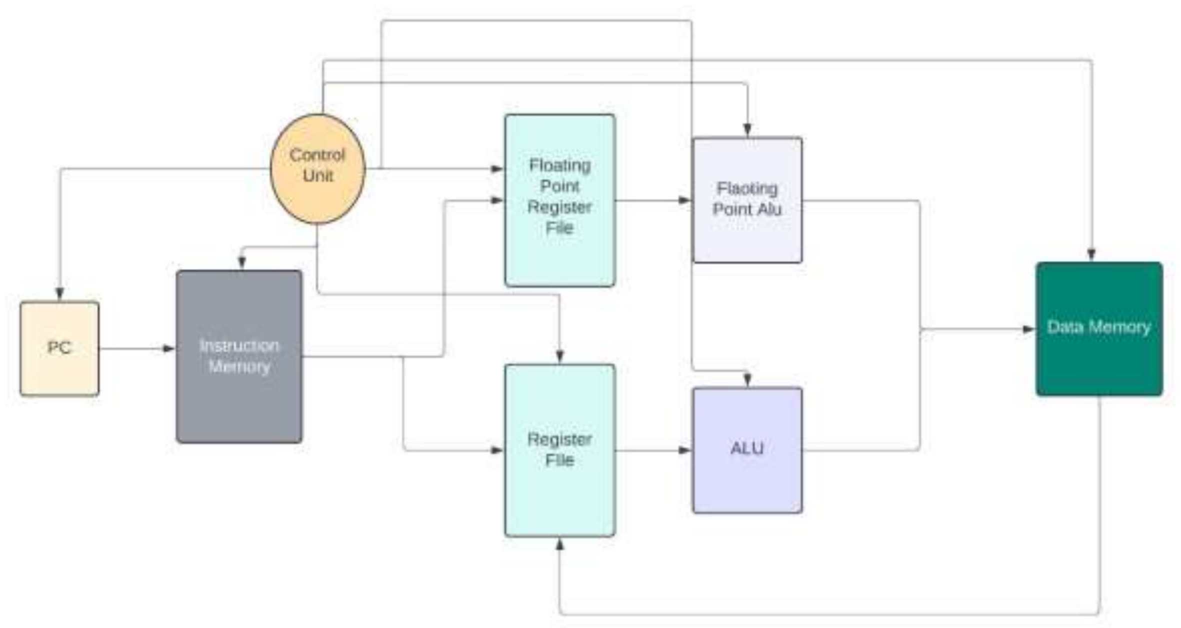
Modules:

- PC
- Instruction_Memory
- Control_Unit_With_Mux
- IF_ID_Register
- Hazard_Detection_Unit
- Forwarding_Unit
- ID_EX_Register
- EX_MEM_Register
- MEM_WB_Register
- Sign_Extension
- Register_File
- ALU_Control
- ALU
- FP_ALU
- Data_Memory
- SevenSegmentDisplay
- TopModule

Modules Description:

The PC (Program Counter) module maintains the address of the current instruction and increments it at every clock cycle's positive edge, ensuring the sequential flow of instructions. Instructions are fetched from the Instruction Memory module and transferred to the IF_ID_Register for decoding. The Control_Unit_With_Mux module generates control signals necessary for instruction execution and selects appropriate paths through multiplexers based on the instruction type. Once decoded, instructions and operands are stored in the ID_EX_Register module. The ALU (Arithmetic Logic Unit) module performs computations on the operands after passing through the ID_EX_Register. Additionally, the Hazard_Detection_Unit identifies hazards in instruction execution, while the Forwarding_Unit resolves data forwarding to mitigate hazards efficiently. Results from ALU operations are stored temporarily in the EX_MEM_Register before being passed to the appropriate destination based on the instruction type. The MEM_WB_Register module holds data momentarily before writing it back to the Register_File. Sign_Extension extends the sign of immediate values, while the ALU_Control module determines the operation to be performed by the ALU. The FP_ALU module specializes in floating-point arithmetic operations. Data is stored and retrieved from the Data_Memory module, and outputs are displayed via the SevenSegmentDisplay. Finally, the TopModule coordinates the overall functionality of the processor, ensuring seamless operation.

Block Diagram:



Floating Point Implementation:

The register file for the floating-point unit is an extension of the general-purpose register file. Floating-point address calculation for the register file follows a specific formula: address = x + 32, where 'x' represents any general-purpose register address. These registers are identified based on their opcode, which we have designated as 17. All calculations are performed in accordance with the IEEE 754 format.

Overall Processor Simulation:

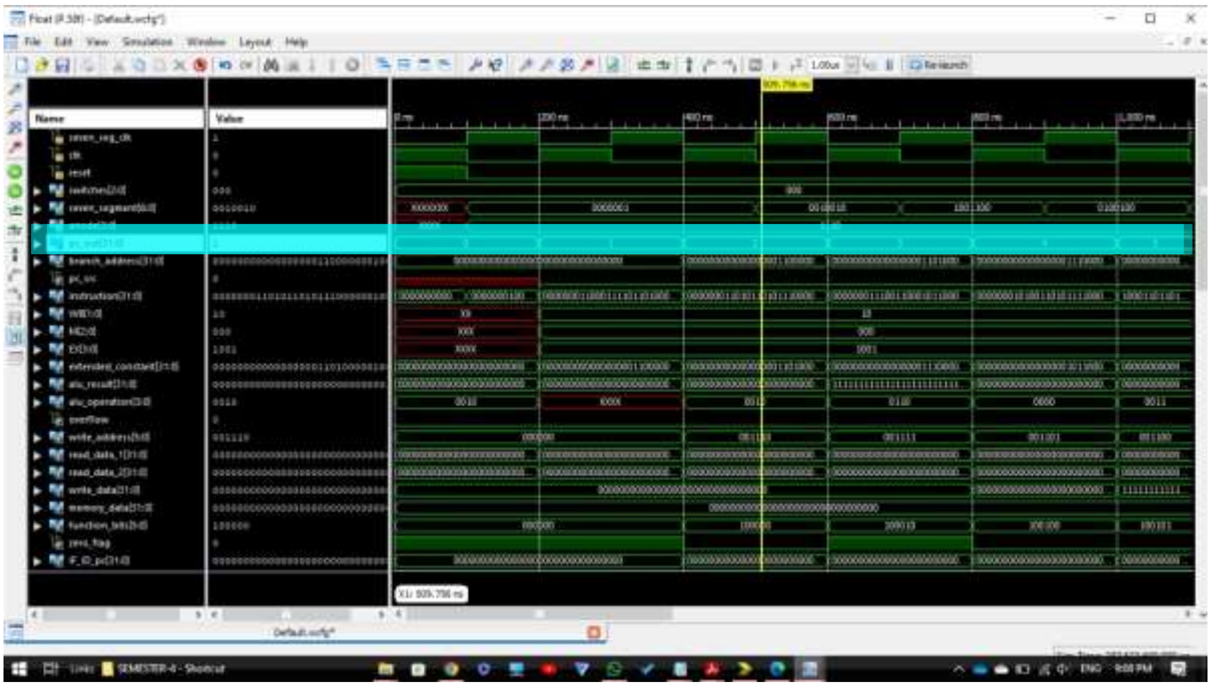


Figure 1: This figure depicts the simulation of the pipelined processor, highlighting the execution of the PC. As observed, the PC increments after every positive clock cycle, fetching the next instruction as the pipeline executes instructions sequentially.

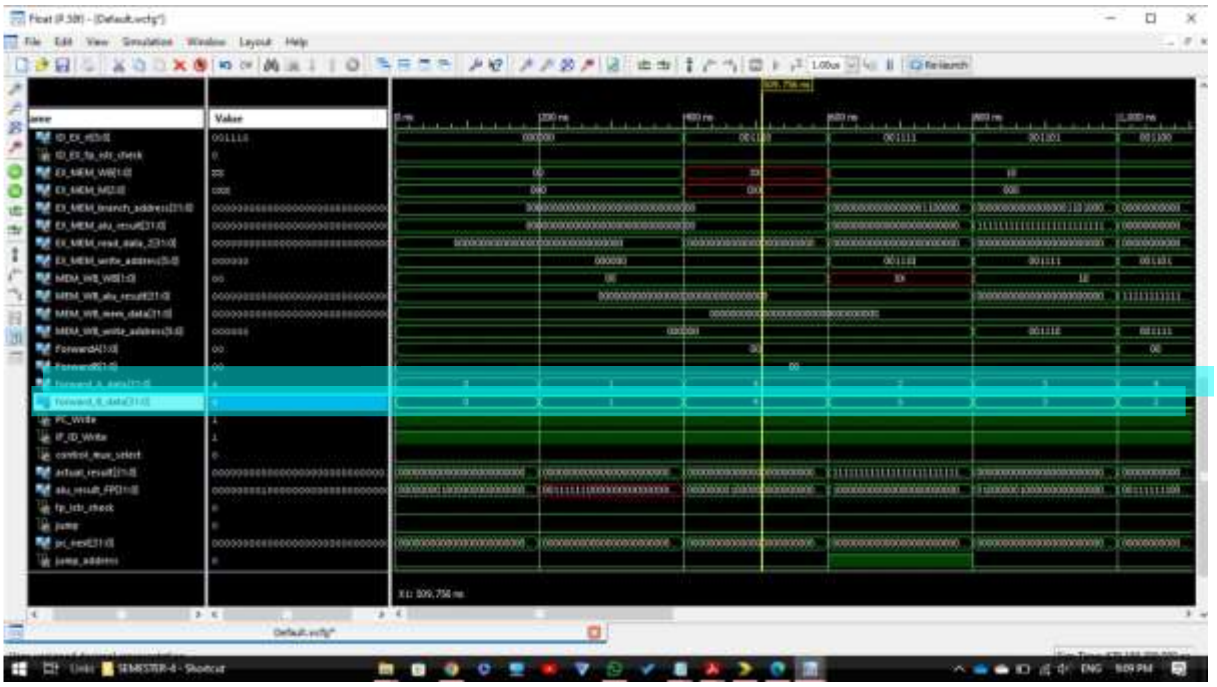


Figure 2: This figure illustrates the operand fetching stage after the decode stage. In the highlighted region, it is evident that operand A is 4 and operand B is also 4, which are then forwarded to the ALU for computation of the ALU result.

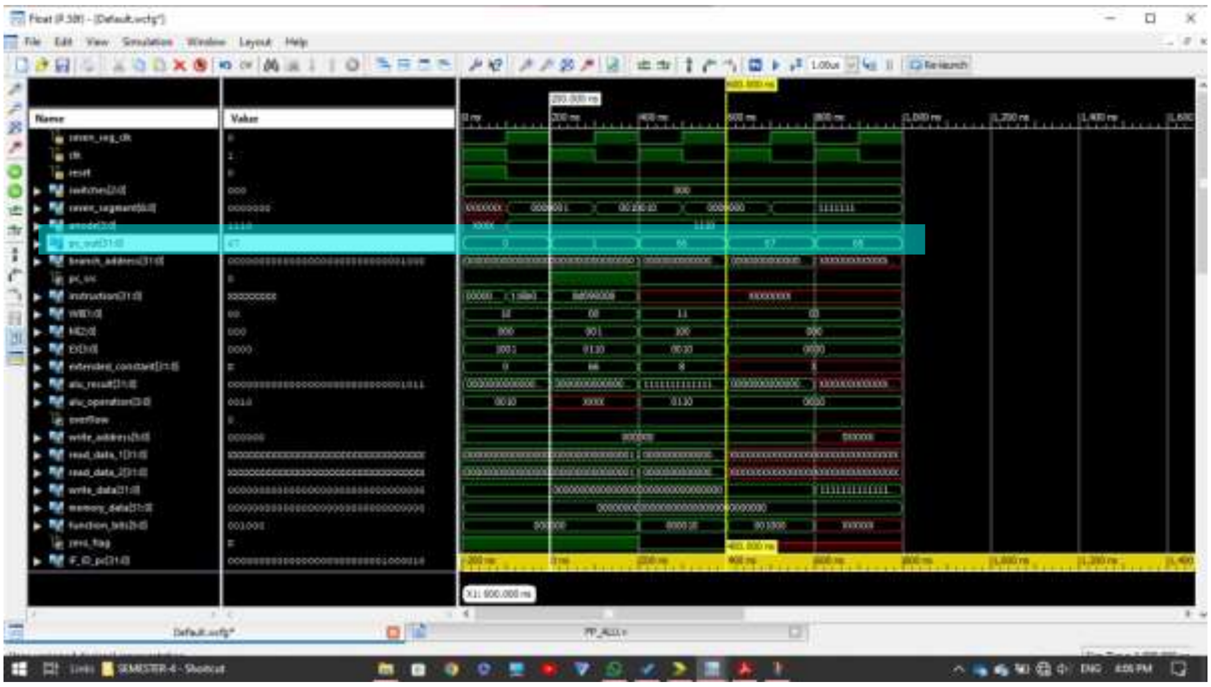


Figure 3:Figure shows the execution of the branch instruction in which the PC value is jumped to 66 which is the branch address.

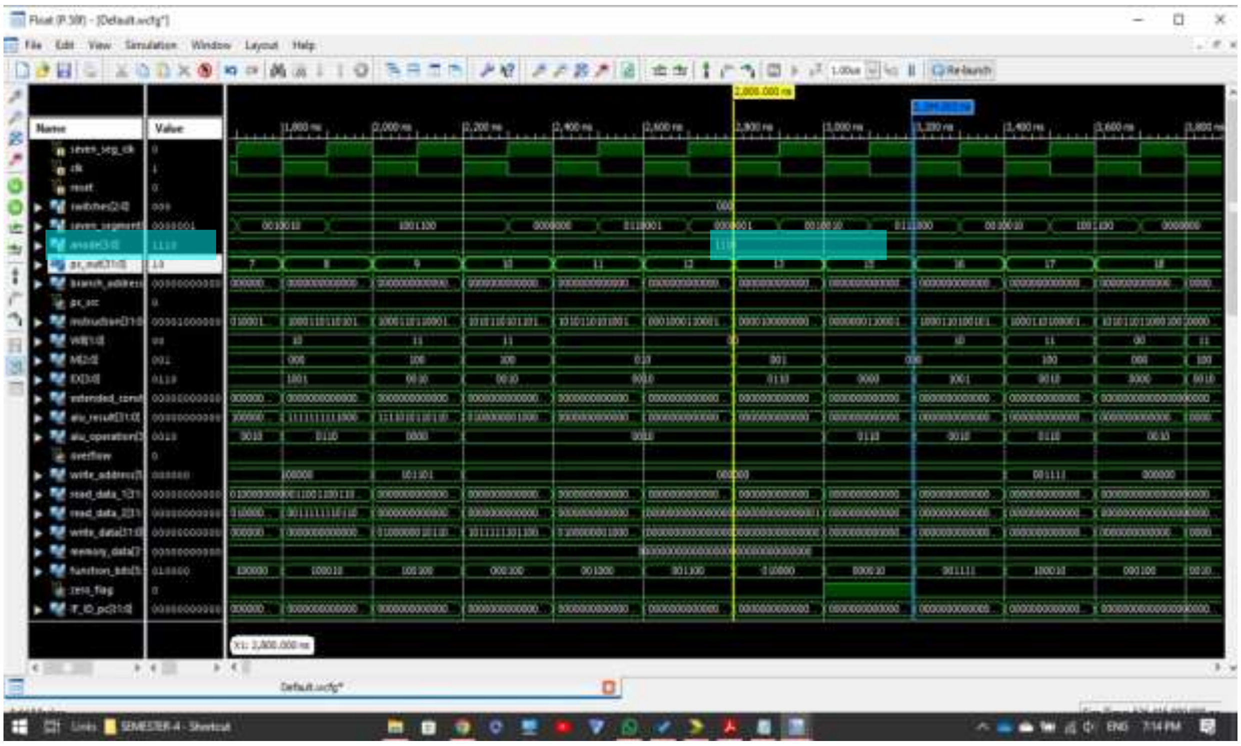


Figure 4:This figure shows the execution of the jump instruction in which the PC is jumped to 15th instruction.

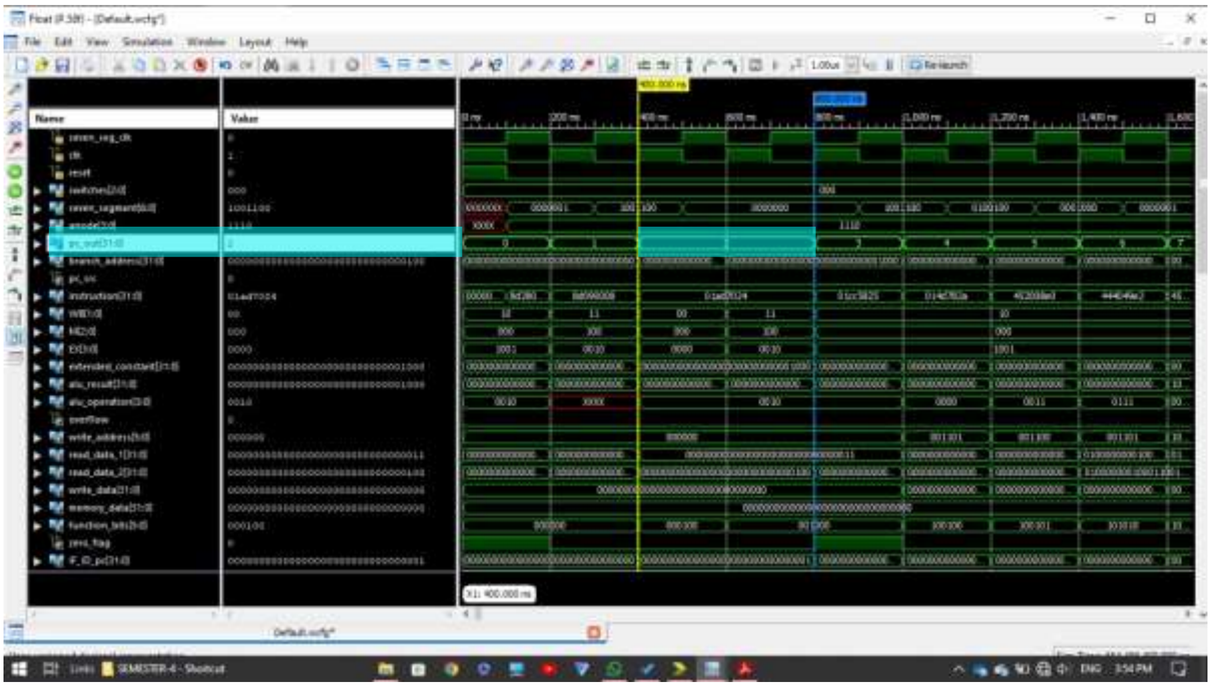


Figure 5:Figure shows that due to the dependency of the instructions stall has been used in the second cycle for the proper execution of the instruction.

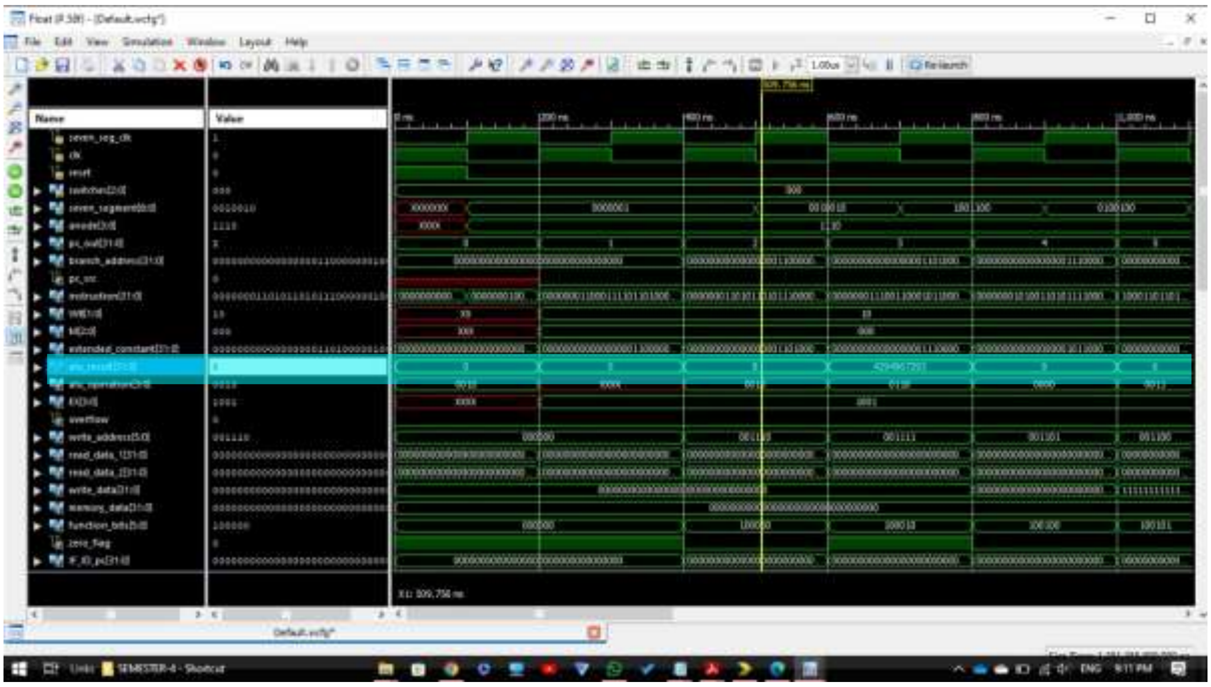


Figure 6: This figure shows the computation of the ALU result from the fetched operands. Initially, the operands were 4 and 4, and the instruction was addition. Therefore, the result is 8, as seen in the highlighted region. In the next cycle, the operands 2 and 5 were processed with a subtraction instruction, yielding a result of -3. Since the result is signed, it is displayed as an unsigned decimal, which is correct. Subsequently, the operands 3 and 3 have been multiplied resulting in 9.

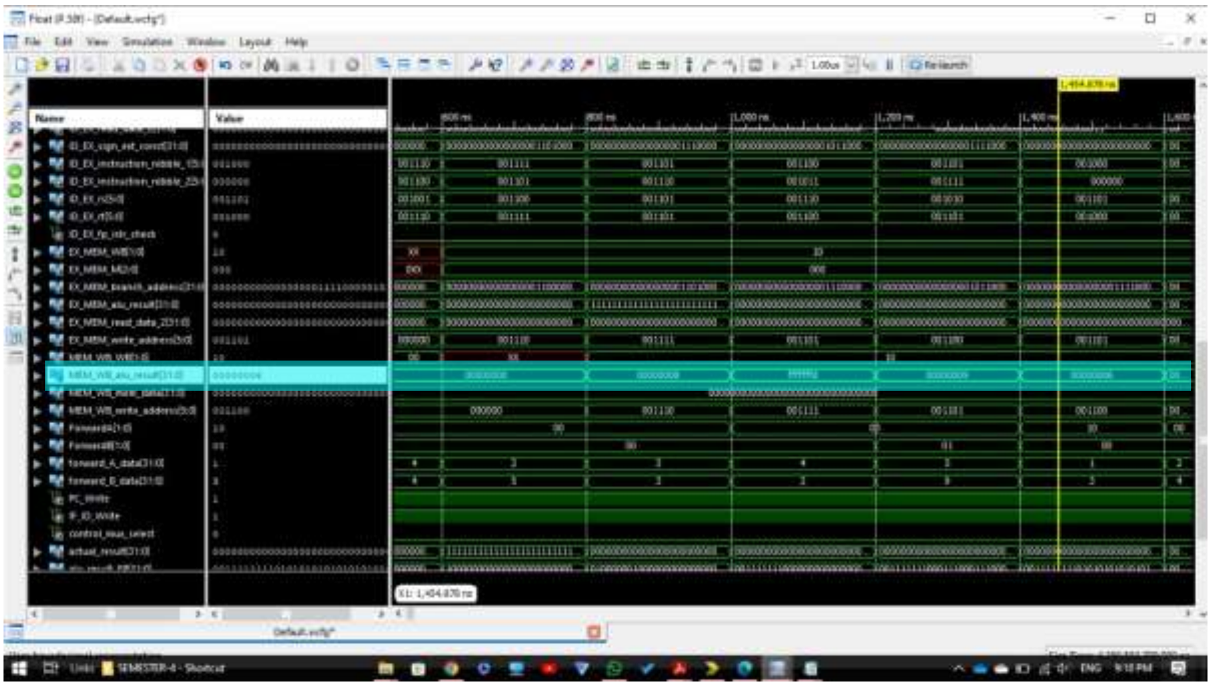


Figure 7: This figure depicts the behaviour that now after every new coming cycle we would get the result of the instruction. As seen from the previous figures that the results have been fetched.

Floating Point Instructions Simulation:

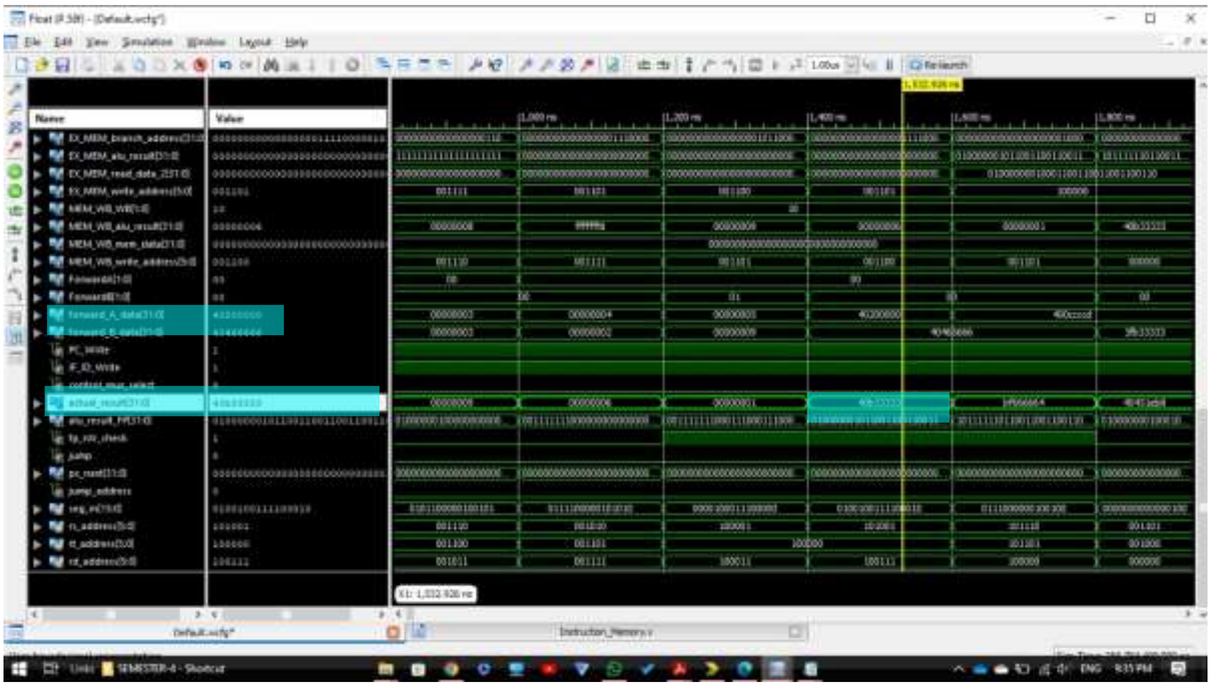


Figure 8: The highlighted portion of the figure shows the result of the floating-point addition. The operands, 2.5 and 3.1, are displayed in hexadecimal. After the addition, the computed result is 5.6, which is represented as 40b33333 in hexadecimal. The result is in normalized form following the IEEE 754 format.

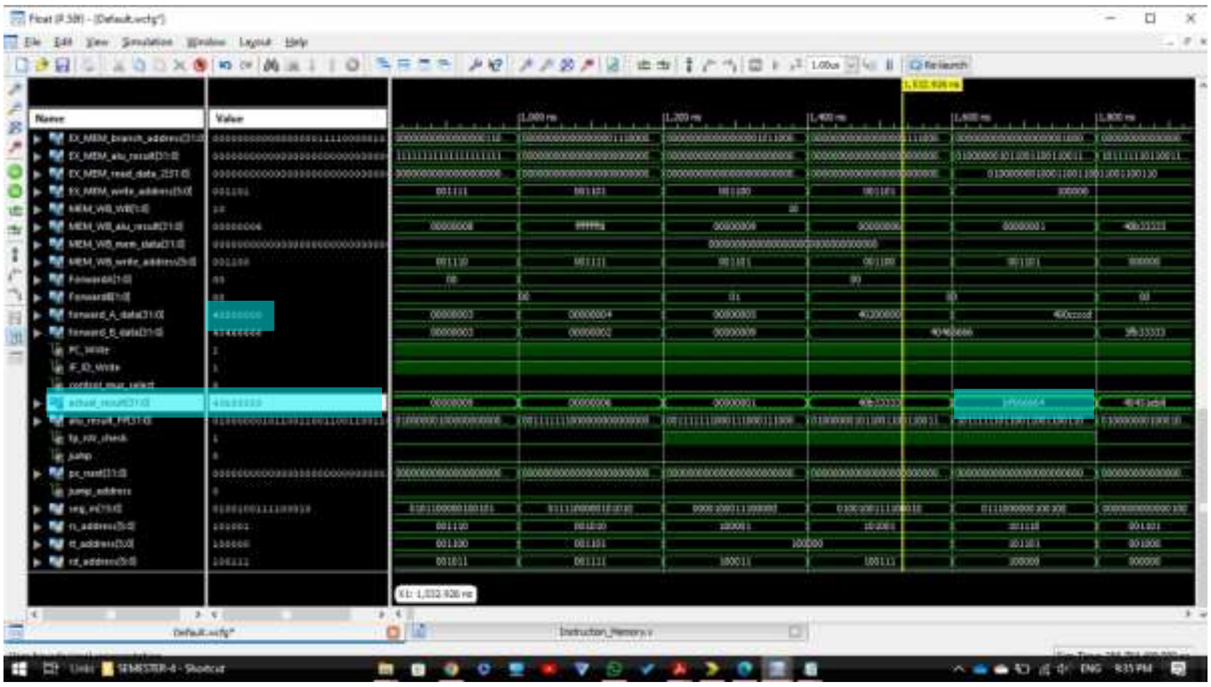


Figure 9: The highlighted portion of the figure shows the result of the floating-point subtraction. The operands, 2.5 and 3.1, are displayed in hexadecimal. After the subtraction, the computed result is -0.6, which is represented as bf666664 in hexadecimal. The result is in normalized form following the IEEE 754 format.

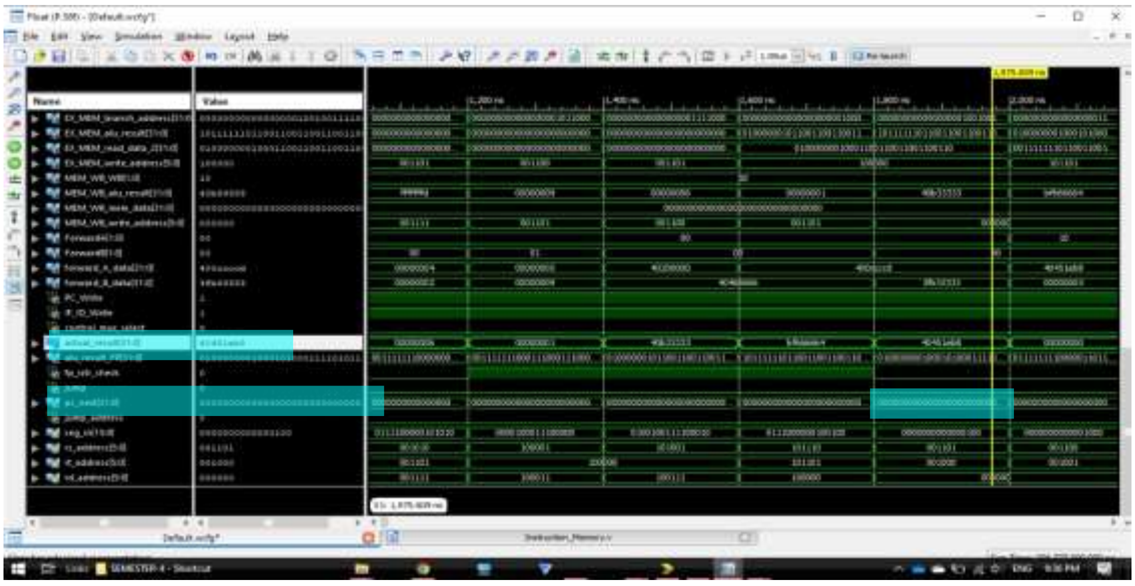


Figure 10: The highlighted portion of the figure shows the result of the floating-point multiplication. The operands, 2.2 and 1.4, are displayed in hexadecimal. After the multiplication, the computed result is 3.08, which is represented as 40451eb8 in hexadecimal. The result is in normalized form following the IEEE 754 format.

FPGA Outputs:

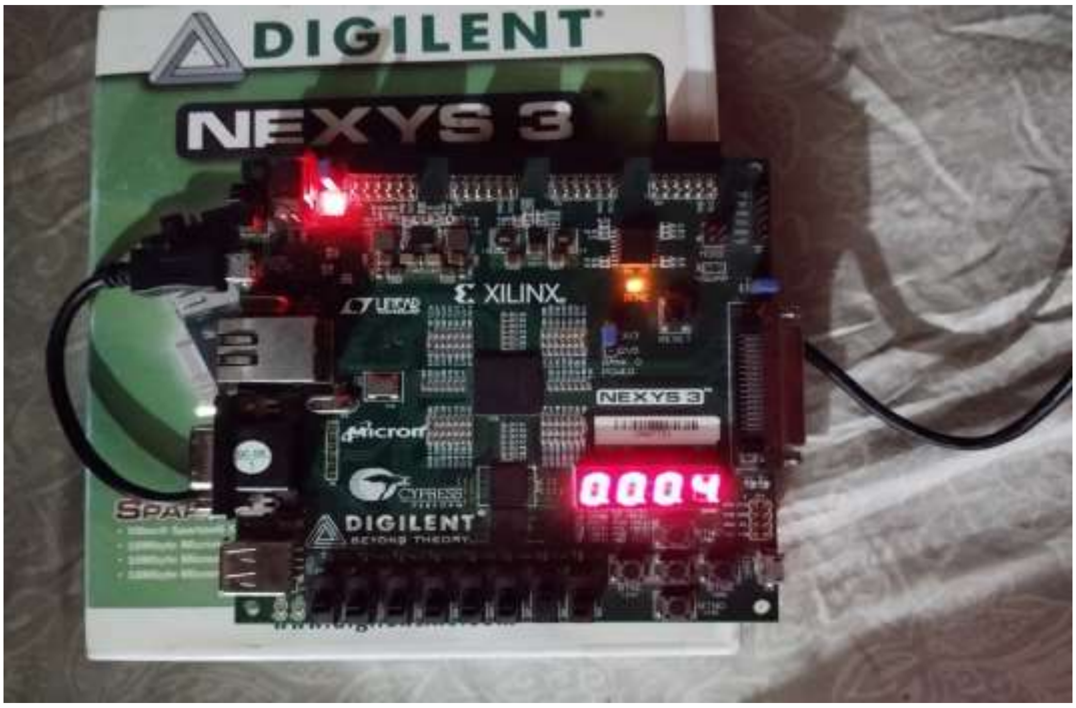


Figure 11: The value of PC is 4, indicating that the fourth instruction is now ready to be executed.

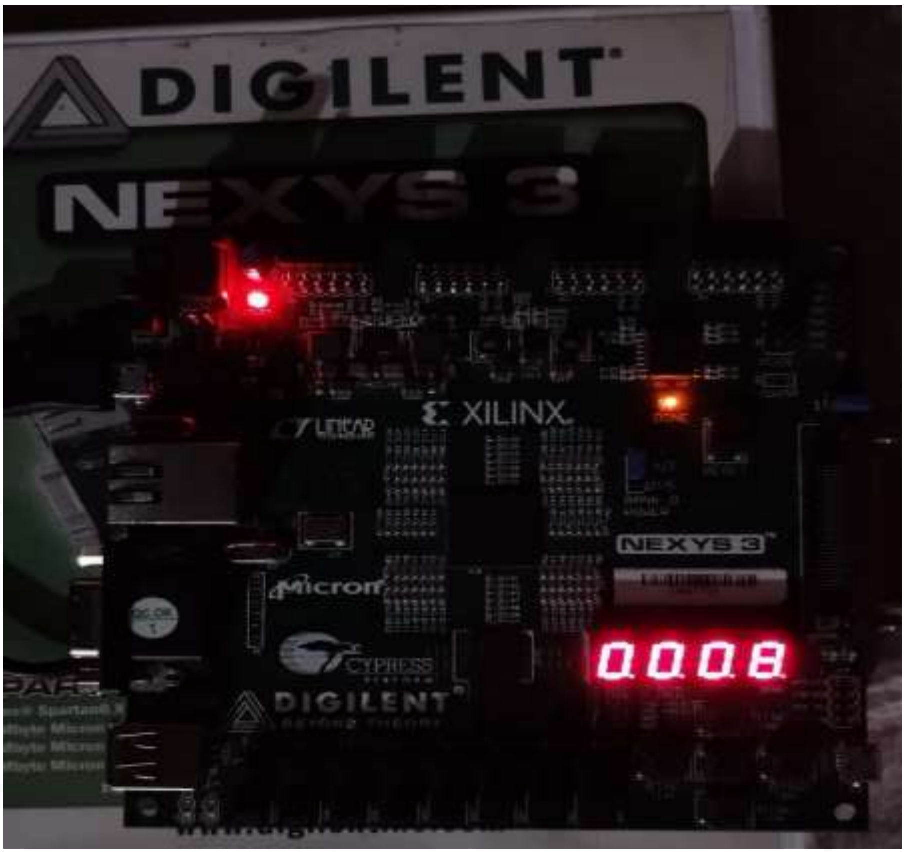


Figure 12: This figure shows the computation of the ALU result of the first instruction in which the operands were 4 and 4 and the result as seen is 8.

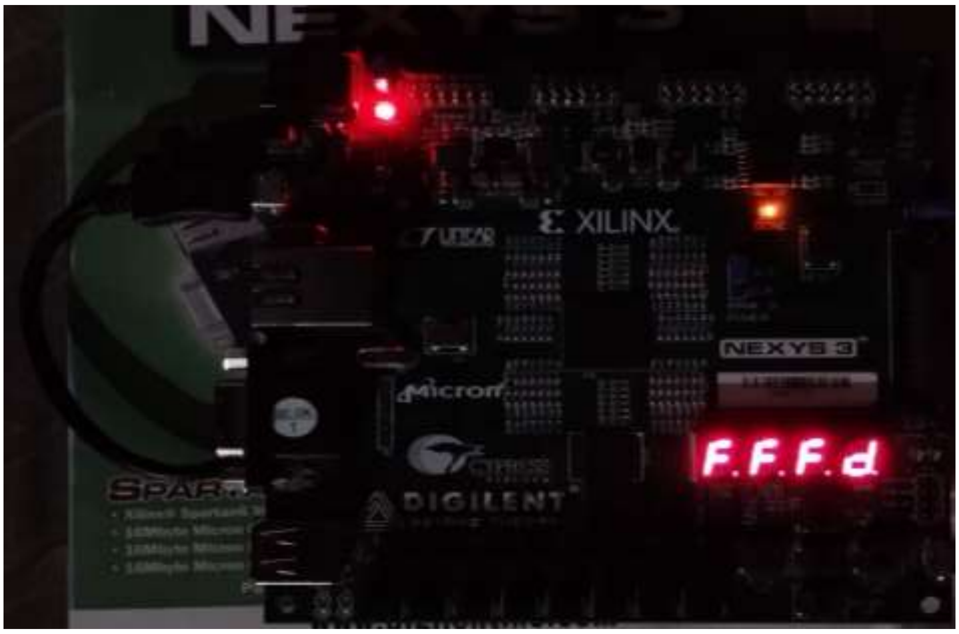


Figure13: Figure shows the result of the next instruction that was of subtraction between 2 and 5 and the result is -3 as seen in the hex representation of the first 16 bits.

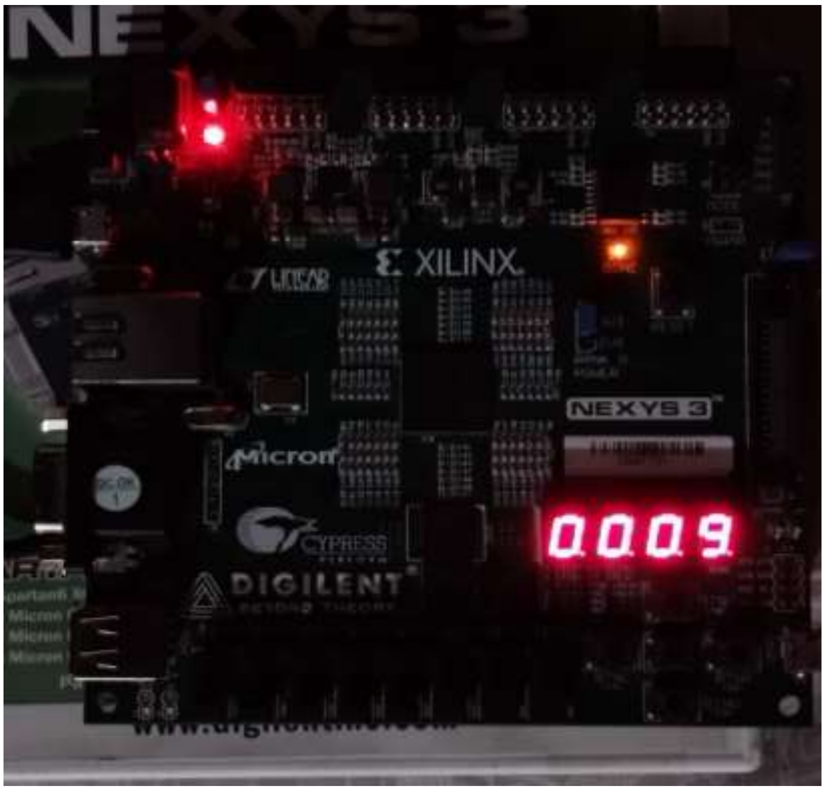


Figure 14: Figure shows the results of the multiplication instruction between 3 and 3.

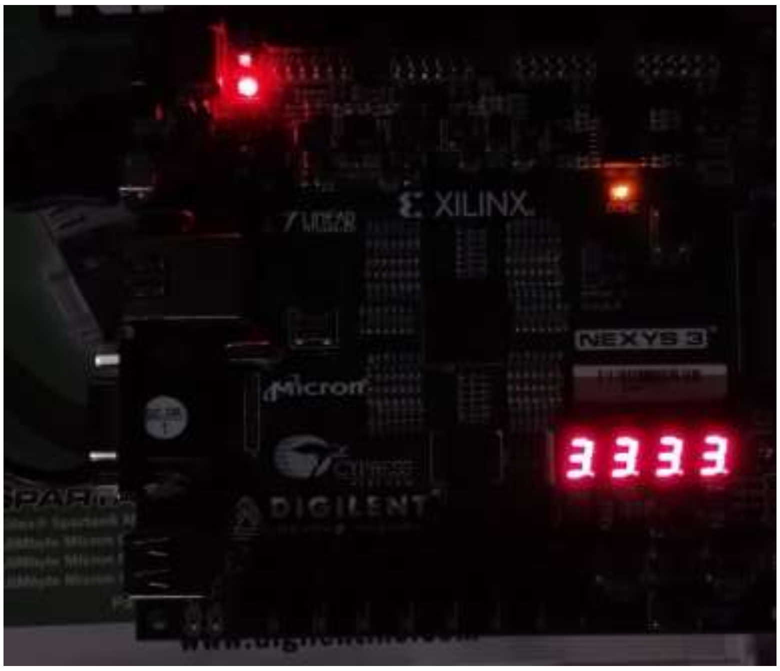


Figure 11:Figure shows the execution of the floating-point addition instruction, and the result shows the first 16 bits hexadecimal representation of the answer.

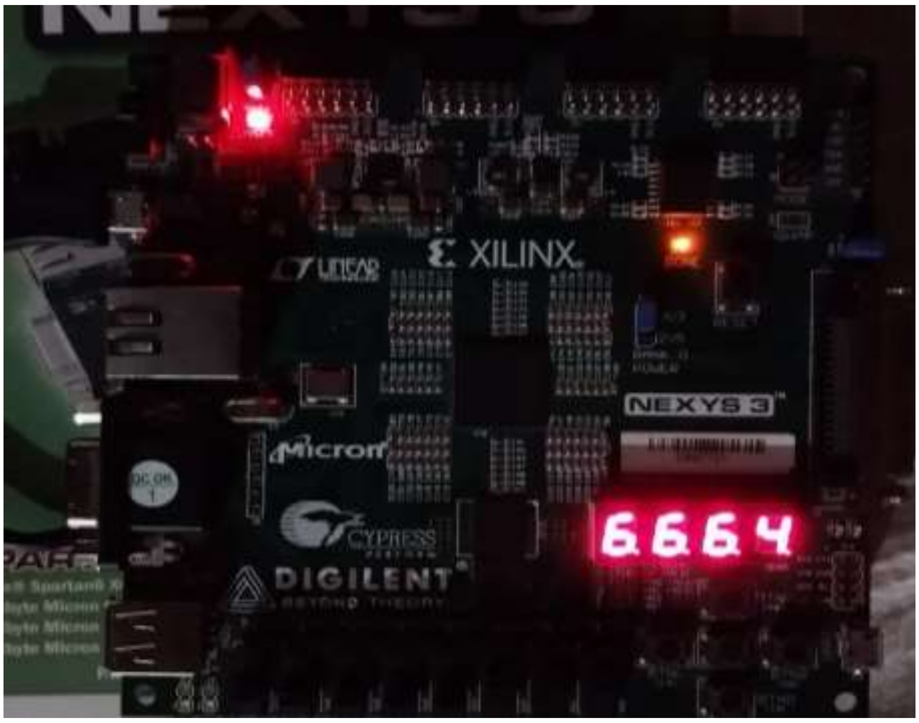


Figure 12: Figure shows the execution of the floating-point subtraction instruction, and the result shows the first 16 bits hexadecimal representation of the answer.

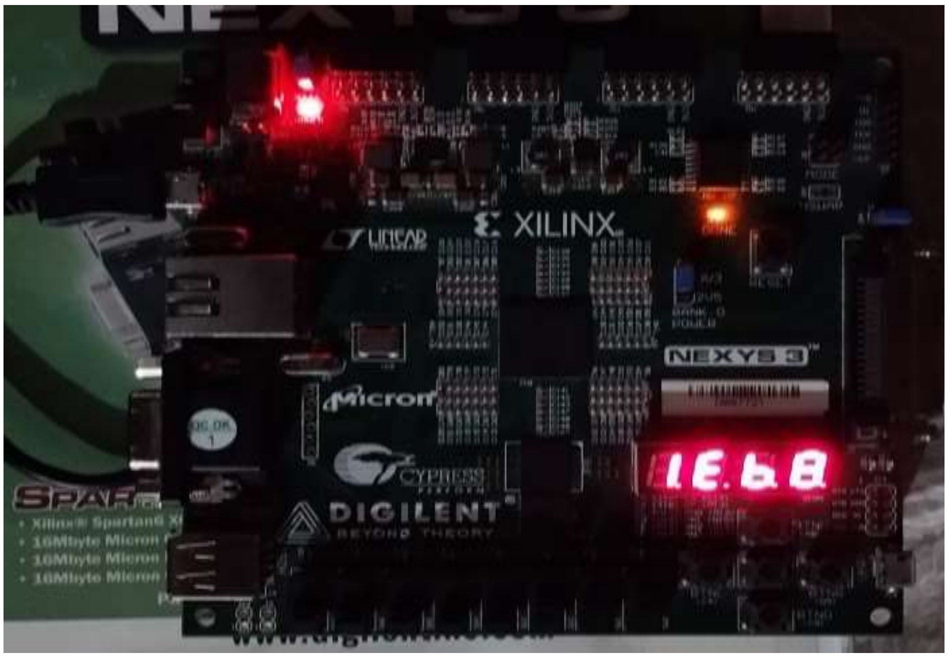


Figure 17: Figure shows the execution of the floating-point multiplication instruction, and the result shows the first 16 bits hexadecimal representation of the answer.

RTL Schematics:

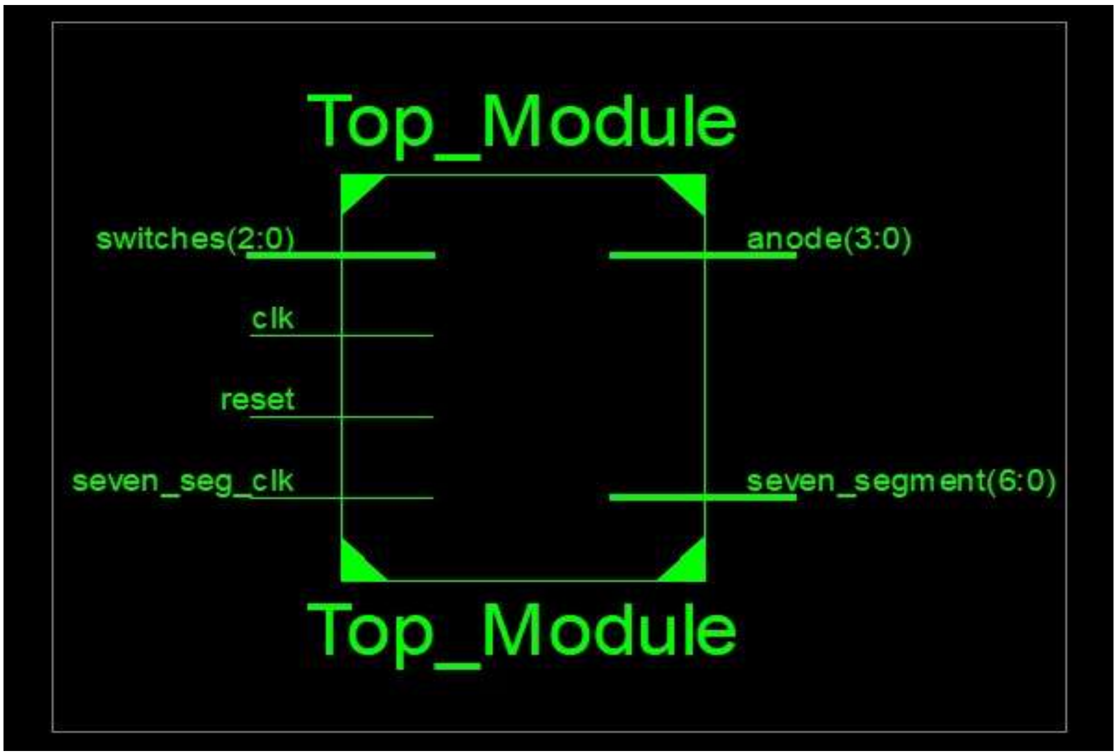


Figure 18: Figure shows the RTL schematics of the top module where all the modules have been called. As seen in the figure that clk, reset, seven_seg_clk are one bit input signals while there is a 3-bit switch that selects the output we want to display on the FPGA. Also the outputs are 7-bit seven segment signal and a 4 bit anode.

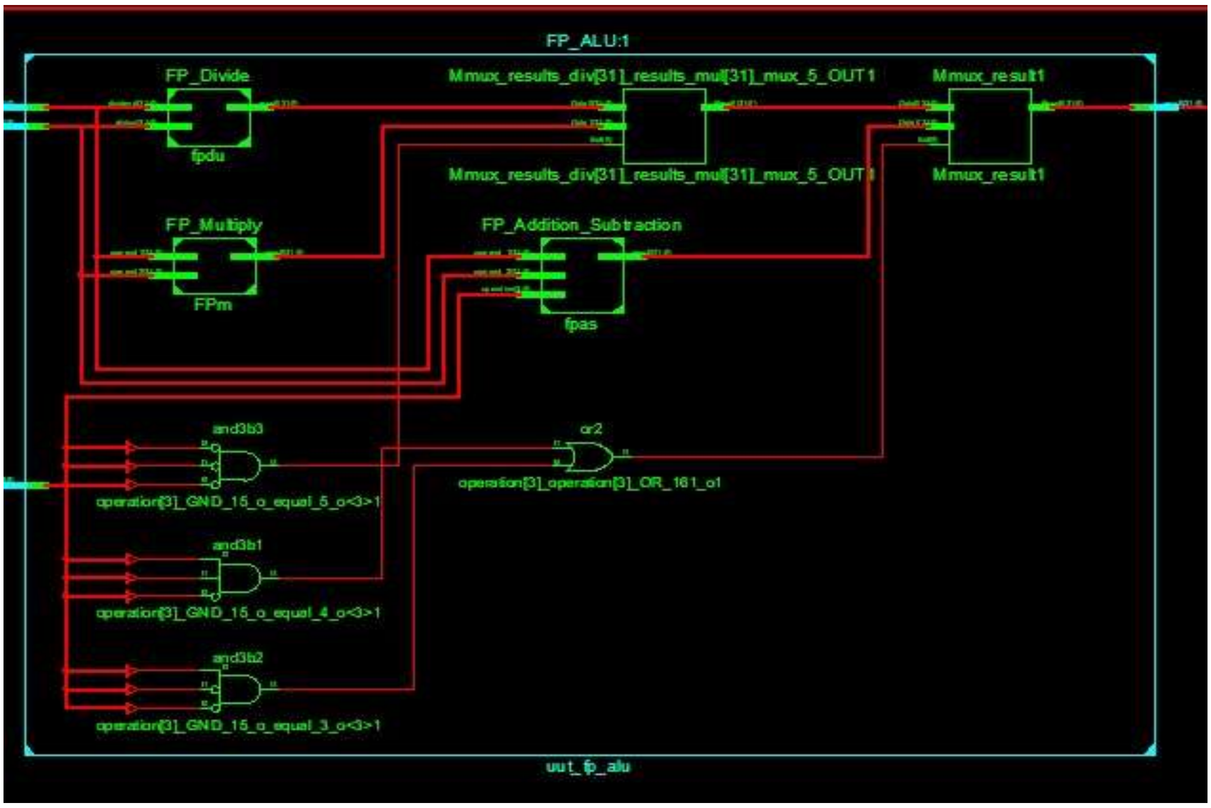
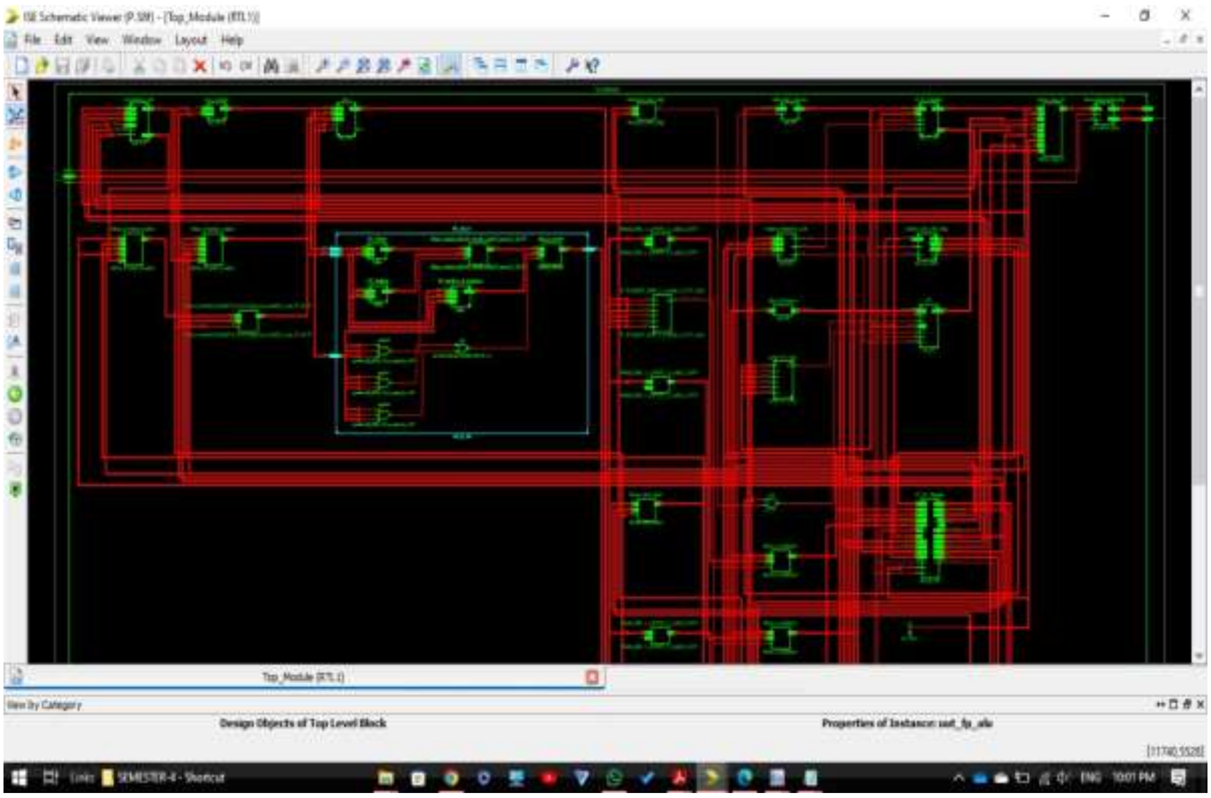
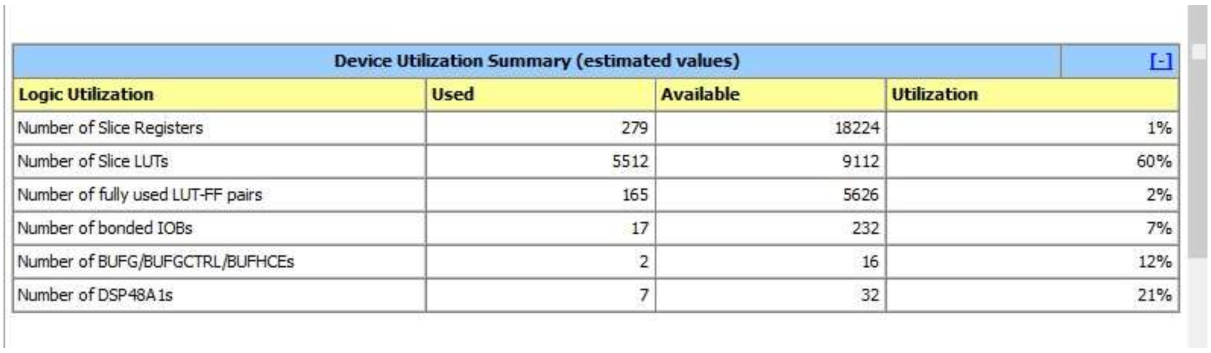

FP_ALU:1
FP_Divide
fpdu
Mmux_results_div[31]_results_mul[31]_mux_5_OUT1
Mmux_result1
FP_Multiply
FPm
FP_Addition_Subtraction
fpas
and3b3
operation[3]_GND_15_o_equal_5_o<3>1
and3b1
operation[3]_GND_15_o_equal_4_o<3>1
and3b2
operation[3]_GND_15_o_equal_3_o<3>1
or2
operation[3]_operation[3]_OR_161_o1
uut_fp_alu

Figure 19: Figure shows the schematics of the Floating Point ALU module in which various sub modules such as Addition, subtraction. Multiplication and division have been implemented and a MUX is used to select the required result based upon the given operation.


ISE Schematic Viewer (P.58f) - [Top_Module (RTL1)]
File Edit View Window Layout Help
Top_Module (RTL1)
View by Category
Design Objects of Top Level Block
Properties of Instance: uut_fp_alu
SEMESTER-4 - Shortcut
ENG
10:01 PM

Device Utilization Summary:



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	279	18224	1%
Number of Slice LUTs	5512	9112	60%
Number of fully used LUT-FF pairs	165	5626	2%
Number of bonded IOBs	17	232	7%
Number of BUFG/BUFGCTRL/BUFHCEs	2	16	12%
Number of DSP48A1s	7	32	21%

Figure 21: This table shows the estimated utilization of various resources on a device. Logic Utilization is at 1%, while Slice LUTs are at 60% utilization.

Conclusions:

In conclusion, we have demonstrated the successful implementation of a five-stage pipelined processor with integrated floating-point capabilities. The processor efficiently handles a variety of instruction types, leveraging a modular design approach that enhances performance and flexibility. Each module, from the PC to the ALU and floating-point unit, played a crucial role in ensuring smooth and accurate instruction execution. The implementation on an FPGA confirmed the practical viability of our design, showcasing its applicability in real-world scenarios. Our approach to floating-point operations, adhering to the IEEE 754 format, ensures precise and reliable computation, essential for applications requiring high numerical accuracy.

GitHub Link:

https://github.com/ahmad292-6000/Pipelined-Processor