ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT 1 REPORT

CRN : 22596

LECTURER : Assoc. Prof. Dr. Gökhan İnce

GROUP MEMBERS:

150220030 : Ahmet Emir Arslan (Group Representative)

150220067 : Baran Turhan

SPRING 2025

Contents

1 INTRODUCTION

In this project, we have built-simulated a basic computer that can make arithmetic and logical operations. Furthermore, results of these operations can be stored into a memory or registers while operations are run sequentially. We used Verilog hardware description language to built our modules and Xilinx Vivado Design Suite to run our tests.

2 MATERIALS AND METHODS

This project consists of multiple Verilog modules that form the fundamental components of a simple computer architecture. The modules are developed in accordance with the specifications given in the Project 1 Homework File and tested using the provided simulation files in Vivado Design Suite.

2.1 List of Modules Implemented

The following modules were implemented:

- **Register16bit.v**: 16-bit register module that supports increment, decrement, load and clear functions based on control inputs.

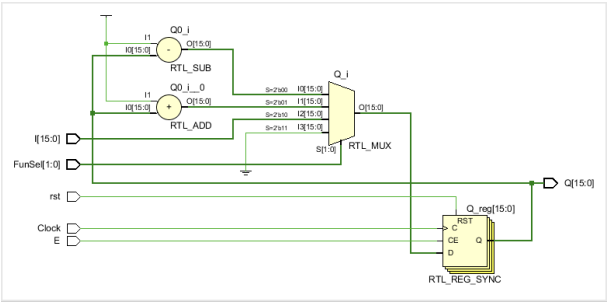

Figure 1: 16-bit Register

- **Register32bit.v**: 32-bit register module that handles both full and partial data loading, clearing, and increment/decrement operations.

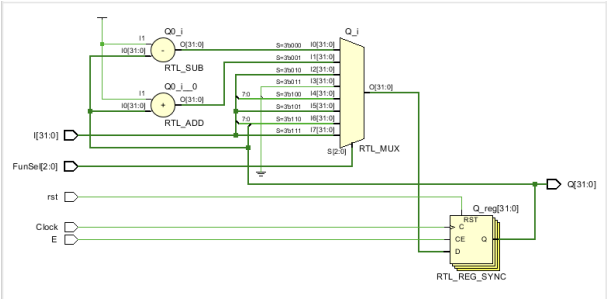

Figure 2: 32-bit Register

- **DataRegister.v**: Module used for holding data with configurable operations.

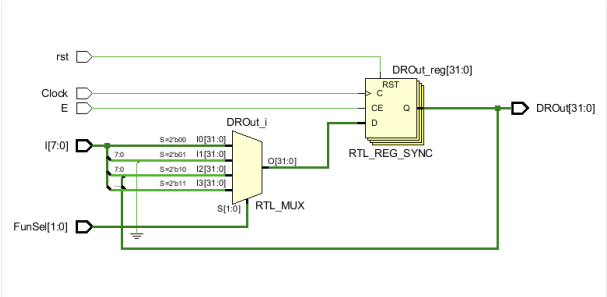

Figure 3: Data Register

- **InstructionRegister.v**: Specialized register for instruction holding with parallel load and reset capabilities.

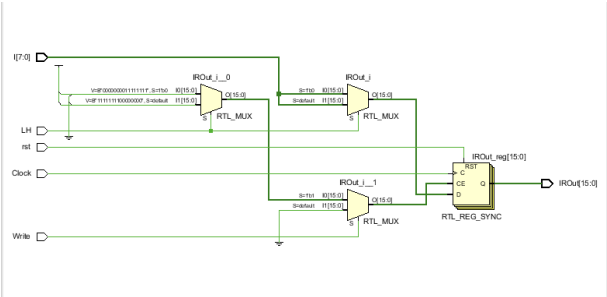

Figure 4: Instruction Register

- **RegisterFile.v**: Holds general-purpose registers including R1-R4 and S1-S4, supporting multiple read/write operations.

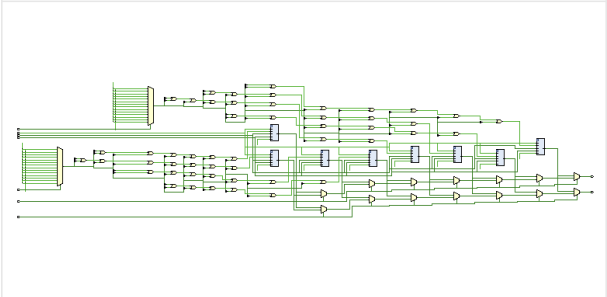

Figure 5: Register File

- **AddressRegisterFile.v**: Maintains address registers and provides access to memory mapping.

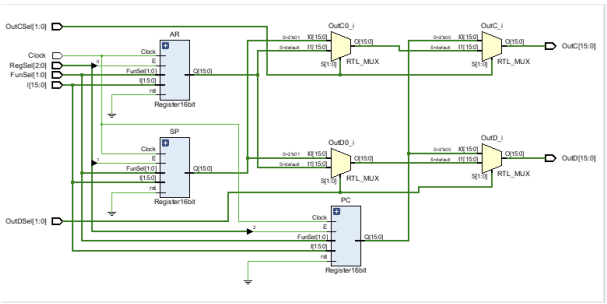

Figure 6: Address Register File

- **ArithmeticLogicUnit.v**: Core ALU that supports arithmetic and logical operations based on FunSel input.

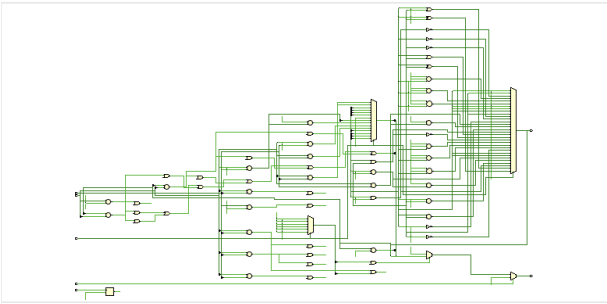

Figure 7: Arithmetic Logic Unit

- **ArithmeticLogicUnitSystem.v**: Combines ALU, RegisterFile, and other subsystems into a functional unit.

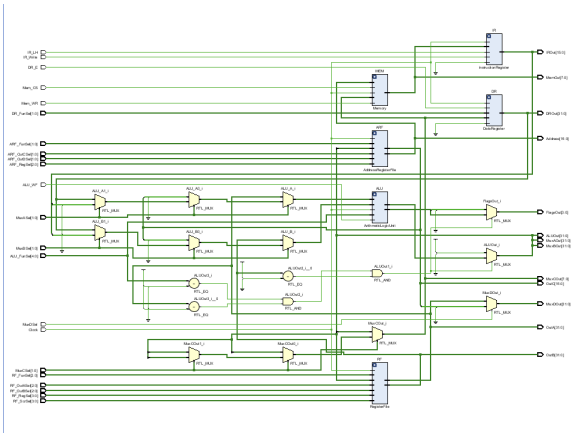

Figure 8: Arithmetic Logic Unit System

2.2 Control Inputs and Functions

Each module was designed with specific control inputs which dictate its operation:

- **E (Enable)**: Activates the module for operation.
- **rst (Reset)**: Clears the register or memory.
- **FunSel**: Selects the function type, e.g., increment, decrement, load, etc.
- **I (Input)**: Input data fed into modules like registers.
- **RegSel / ScrSel / OutASel / OutBSel**: Register selectors used within RegisterFile.

2.3 Task Distribution

- Baran Turhan implemented:
  - Register16bit.v
  - Register32bit.v
  - DataRegister.v
  - InstructionRegister.v
  - RegisterFile.v
- Ahmet Emir Arslan implemented:
  - AddressRegisterFile.v
  - ArithmeticLogicUnit.v

- ArithmeticLogicUnitSystem.v

- Both members collaboratively tested and debugged the system.

3 RESULTS

All modules were compiled and simulated successfully using the provided simulation files. Below are notable outcomes:

- Registers correctly respond to enable and reset signals.
- ALU performs arithmetic and logical operations accurately according to the function selector.
- RegisterFile supports correct read/write behavior across multiple registers.
- System passed simulation outputs without any syntax or logic errors.
- **Register16bit.v**

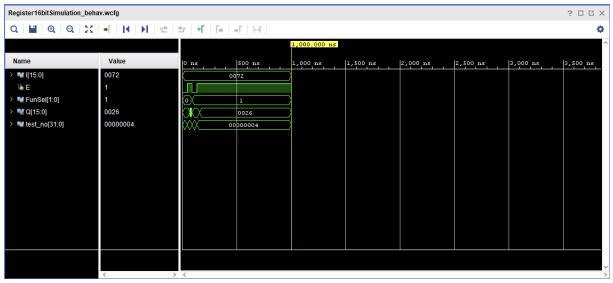

Figure 9: 16-bit Register Results

- **Register32bit.v**

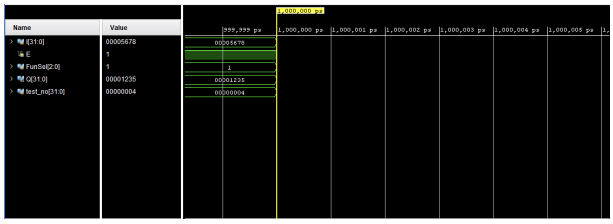

Figure 10: 32-bit Register Results

- **DataRegister.v**

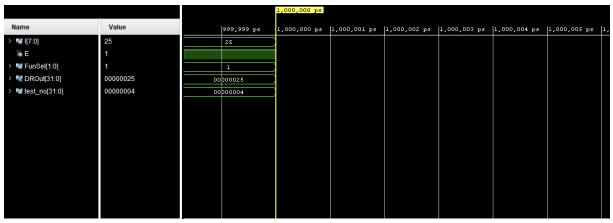

Figure 11: Data Register Results

- **InstructionRegister.v**

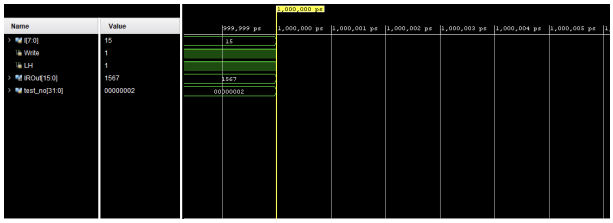

Figure 12: Instruction Register Results

- **RegisterFile.v**

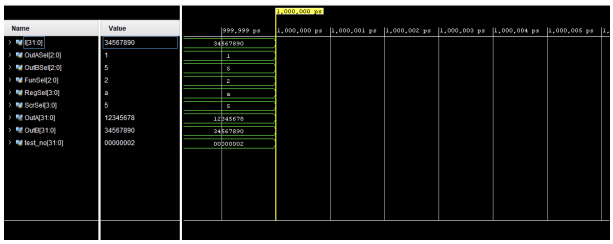

Figure 13: Register File Results

- **AddressRegisterFile.v**

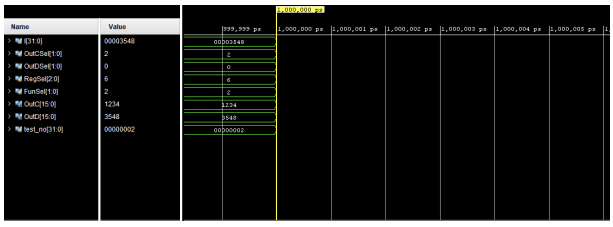

Figure 14: Address Register File Results

- **ArithmeticLogicUnit.v**

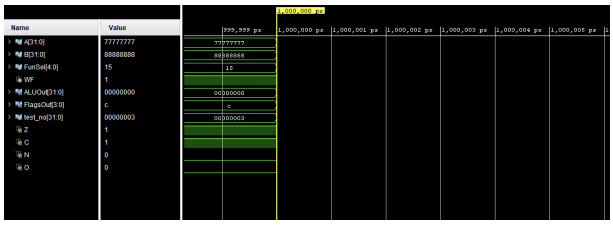

Figure 15: Arithmetic Logic Unit Results

- **ArithmeticLogicUnitSystem.v**

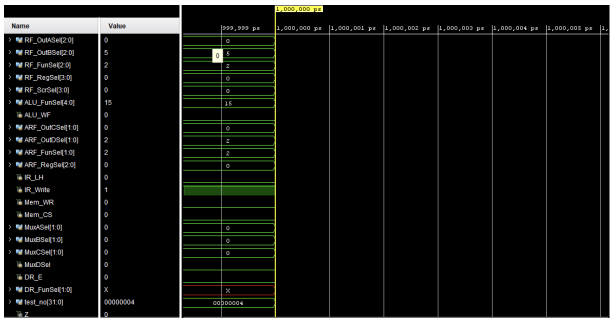

Figure 16: Arithmetic Logic Unit System Results

4 DISCUSSION

The project highlighted key aspects of digital system design:

- Designing modular and reusable components simplifies integration and testing.
- Verilog requires precise control logic, and simulation is critical in catching edge cases.
- Understanding hardware logic such as enable signals, clocked vs combinational behavior was crucial.
- Group collaboration accelerated debugging and ensured comprehensive testing.

We faced initial issues with the simulation compatibility which were resolved by strictly following the expected module interface and naming conventions.

5 CONCLUSION

This project deepened our understanding of computer architecture and digital logic. We:

- Gained hands-on experience in Verilog and Vivado.
- Learned how to debug complex digital systems.
- Realized the importance of clean module interfaces and documentation.

Though some modules were initially challenging, the final system functioned as expected under simulation. The clear structure of responsibilities also helped manage the workload efficiently.