ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT 1 REPORT

CRN : 30307

LECTURER : Kadir ÖZLEM

GROUP MEMBERS:

150200723 : Semih TUTUŞ

150220731 : Elif Ceren GÖNEN

SUMMER 2025

Contents

1 INTRODUCTION

This project implements the fundamental building blocks of a computer system using Verilog HDL. We designed a hierarchical system containing:

- Basic 16-bit and 32-bit registers with multiple operation modes
- Specialized register files (Instruction and Data registers)
- A complete register file system with general-purpose and scratch registers
- A fully-functional 32-bit ALU with flag generation
- System integration with multiplexed data paths

All components were designed for maximum reusability and verified through exhaustive simulation.

2 MATERIALS AND METHODS

2.1 16-Bit Register

The 16-bit register serves as the foundation for more complex components. Its key features include:

Enable	**FunSel**	**Operation**	**Q^+**
0	Φ	Retain value	Q
1	00	Decrement	Q - 1
1	01	Increment	Q + 1
1	10	Load	I
1	11	Clear	0

Table 1: 16-bit Register Control Logic

Key Design Considerations:

- Implemented with positive-edge triggered D flip-flops
- Asynchronous reset for initialization
- Enable signal acts as clock gating control

2.2 32-Bit Register

The 32-bit register extends functionality with advanced operations:

Enable	**Funsel**	**Operation**	**Q$^+$**
0	Φ	Retain value	Q
1	000	Decrement	Q - 1
1	001	Increment	Q + 1
1	010	Load	I
1	011	Clear	0
1	100	Load Lower 8 bits	Q (31-8) ← Clear
			Q (7-0) ← I (7-0)
1	101	Load Lower 16 bits	Q (31-16) ← Clear
			Q (15-0) ← I (15-0)
1	110	8-bit Left Shift	Q (31-8) ← Q (23-0)
			Q (7-0) ← I (7-0)
1	111	16-bit Sign Extend	Q (31-16) ← Sign Extend (I (15))
			Q (15-0) ← I (15-0)

Table 2: 32-bit Register Advanced Operations

Special Features:

- Partial writes without read-modify-write cycles
- Sign extension unit for arithmetic operations
- Multi-function shifter implementation

2.3 Instruction Register

The instruction register handles 8-bit input data, assembling it into 16-bit instructions. It features byte-selectable writing (MSB/LSB) controlled by the LH signal and only updates on clock edges when enabled.

L'H	**Write**	**Operation**	**IR$^+$**
Φ	0	Retain Value	IR
0	1	Load LSB	IR (7-0) ← I
1	1	Load MSB	IR (15-8) ← I

Table 3: IR Loading Modes

2.4 Data Register

This specialized register processes 8-bit input into 32-bit output with multiple transformation modes including sign/zero extension and bit shifting operations, all controlled through function selection inputs.

Enable	**Funsel**	**Operation**	**DR$^+$**
0	Φ	Retain Value	DR
1	00	8-bit Sign Extend	DR (31-8) $\leftarrow$ Sign Extend (I (7))
			DR (7-0) $\leftarrow$ I (7-0)
1	01	8-bit Extend	DR (31-8) $\leftarrow$ Clear
			DR (7-0) $\leftarrow$ I (7-0)
1	10	8-bit Left Shift	DR (31-8) $\leftarrow$ DR (23-0)
			DR (7-0) $\leftarrow$ I (7-0)
1	11	8-bit Right Shift	DR (23-0) $\leftarrow$ DR (31-8)
			DR (31-24) $\leftarrow$ I (7-0)

Table 4: IR Loading Modes

2.5 Register File

Our register file implementation combines eight 32-bit registers (4 general purpose + 4 scratch) with dual output ports. The design includes complete bypass logic and supports simultaneous read/write operations.

The Register File consists of 8 registers (4 general purpose R1-R4 and 4 scratch S1-S4) with the following organization:

OutASel	**OutA**	**OutBSel**	**OutB**
000	R1	000	R1
001	R2	001	R2
010	R3	010	R3
011	R4	011	R4
100	S1	100	S1
101	S2	101	S2
110	S3	110	S3
111	S4	111	S4

Table 5: IR Loading Modes

Register operations are controlled by:

FunSel	Operation	R_x^+
000	Decrement	R_x - 1
001	Increment	R_x + 1
010	Load	I
011	Clear	0
100	8-bit Clear	R_x (31-8) ← Clear
		R_x (7-0) ← I (7-0)
101	16-bit Clear	R_x (31-16) ← Clear
		R_x (15-0) ← I (15-0)
110	8-bit Left Shift	R_x (31-8) ← R_x (23-0)
		R_x (7-0) ← I (7-0)
111	16-bit Sign Extend	R_x (31-16) ← Sign Extend (I (15))
		R_x (15-0) ← I (15-0)

Table 6: RF Function Selection (FunSel)

Register selection uses two 4-bit control signals:

RegSel	Enabled Registers	RegSel	Enabled Registers
0000	None	1000	R1
0001	R4	1001	R1,R4
0010	R3	1010	R1,R3
0011	R3,R4	1011	R1,R3,R4
0100	R2	1100	R1,R2
0101	R2,R4	1101	R1,R2,R4
0110	R2,R3	1110	R1,R2,R3
0111	R2,R3,R4	1111	All

Table 7: RF Register Selection (RegSel)

Register scratch selection uses two 4-bit control signals:

SrcSel	Enabled Registers	SrcSel	Enabled Registers
0000	None	1000	S1
0001	S4	1001	S1,S4
0010	S3	1010	S1,S3
0011	S3,S4	1011	S1,S3,S4
0100	S2	1100	S1,S2
0101	S2,S4	1101	S1,S2,S4
0110	S2,S3	1110	S1,S2,S3
0111	S2,S3,S4	1111	All

Table 8: RF Scratch Register Selection (SrcSel)

2.6 Address Register File

The address register file contains three special-purpose 16-bit registers (PC, AR, SP) with dedicated increment/decrement functionality.

It provides two independent output ports with full bypass capability:

OutACSel	OutC	OutDSel	OutD
00	PC	00	PC
01	SP	01	SP
10	AR	10	AR
11	AR	11	AR

Table 9: ARF Output Selection

ARF operations are controlled by:

FunSel	Operation	R_x^+
00	Decrement	R_x - 1
01	Increment	R_x + 1
10	Load	I
11	Clear	0

Table 10: ARF Function Selection (FunSel)

Register selection uses a 3-bit control signal:

RegSel	Enabled Registers	RegSel	Enabled Registers
000	None	100	PC
001	AR	101	PC, AR
010	SP	110	PC, SP
011	SP, AR	111	All

Table 11: ARF Register Selection (RegSel)

2.7 Arithmetic Logic Unit

Our ALU implements a comprehensive set of arithmetic and logical operations on 32-bit operands. It generates four condition flags (Zero, Carry, Negative, Overflow) that update synchronously with the operation results.

- **Flag Definitions**:
  - Z (Zero): Set when ALUOut = 0
  - C (Carry): Set when operation produces carry
  - N (Negative): Set when MSB of result is 1
  - O (Overflow): Set when signed arithmetic overflow occurs
- **Flag Order**: Z is MSB (bit 3), O is LSB (bit 0) of flag register
- **Write Flag (WF)**: Flags update only when WF=1

The ALU performs 32 operations on 32-bit inputs with flag generation:

16-bit Operations						32-bit Operations					
FunSel	**Operation**	**Z**	**C**	**N**	**O**	**FunSel**	**Operation**	**Z**	**C**	**N**	**O**
00000	A	+	-	+	-	10000	A	+	-	+	-
00001	B	+	-	+	-	10001	B	+	-	+	-
00010	NOT A	+	-	+	-	10010	NOT A	+	-	+	-
00011	NOT B	+	-	+	-	10011	NOT B	+	-	+	-
00100	A + B	+	+	+	+	10100	A + B	+	+	+	+
00101	A + B + Carry	+	+	+	+	10101	A + B + Carry	+	+	+	+
00110	A - B	+	+	+	+	10110	A - B	+	+	+	+
00111	A AND B	+	-	+	-	10111	A AND B	+	-	+	-
01000	A OR B	+	-	+	-	11000	A OR B	+	-	+	-
01001	A XOR B	+	-	+	-	11001	A XOR B	+	-	+	-
01010	A NAND B	+	-	+	-	11010	A NAND B	+	-	+	-
01011	LSL A	+	+	+	-	11011	LSL A	+	+	+	-
01100	LSR A	+	+	+	-	11100	LSR A	+	+	+	-
01101	ASR A	+	-	-	-	11101	ASR A	+	-	-	-
01110	CSL A	+	+	+	-	11110	CSL A	+	+	+	-
01111	CSR A	+	+	+	-	11111	CSR A	+	+	+	-

Table 12: ALU Operations and Status Flags

2.8 Arithmetic Logic Unit System

The complete system integrates all components through a carefully designed bus structure with multiplexed inputs/outputs. The control logic coordinates data movement between all units while maintaining synchronous operation.

The complete ALU system integrates all components through multiplexers:

- **Data Paths**:
  - 32-bit main data bus
  - 16-bit address bus
  - 8-bit immediate data path
- **Control Signals**:
  - MuxASel, MuxBSel: 2-bit selectors
  - MuxCSel: 2-bit byte selector
  - MuxDSel: 1-bit output selector

Mux	Select	Output Connection
MuxA	00	ALUOut
	01	ARF OutC
	10	DROut
	11	IROut[7:0]
MuxB	00	ALUOut
	01	ARF OutD
	10	DROut
	11	IROut[7:0]
MuxC	00	ALUOut[7:0]
	01	ALUOut[15:8]
	10	ALUOut[23:16]
	11	ALUOut[31:24]
MuxD	0	RF OutA
	1	ARF OutC

Table 13: ALU System Multiplexer Configuration

3 RESULTS

3.1 16-Bit Register

- Successfully implemented all 4 operations (retain, decrement, increment, load, clear)
- Verified through exhaustive simulation of all 2-bit FunSel combinations
- Correct behavior observed:
  - Decrement operation properly handles underflow (0x0000 → 0xFFFF)
  - Clear operation resets all bits synchronously

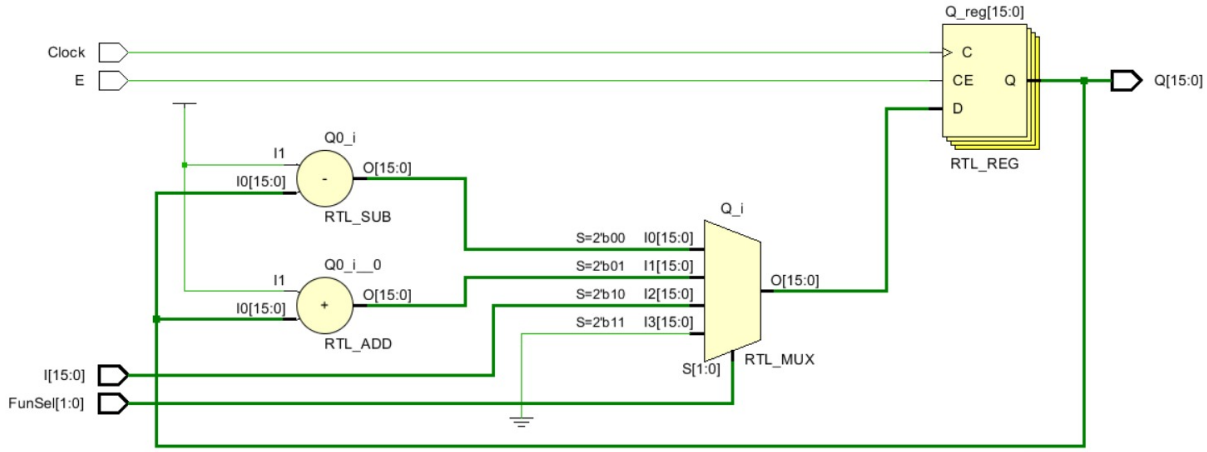

Figure 1: 16-Bit Register Elaboraion Diagram

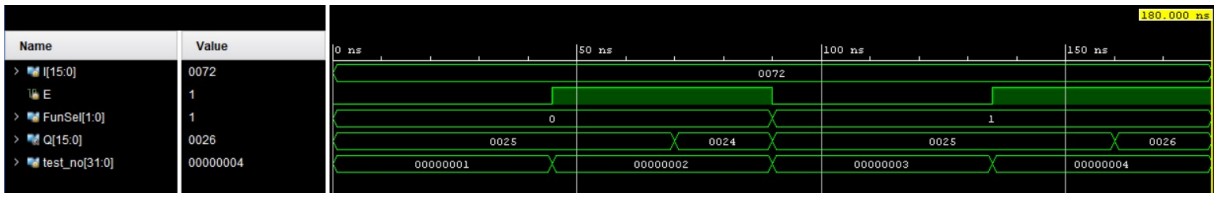

Figure 2: 16-Bit Register Simulation

3.2 32-Bit Register

- Implemented all 8 operations including advanced features:
  - Partial loading (8-bit and 16-bit) with automatic upper-bit clearing
  - Shift operations with correct bit rotation
  - Sign extension working for both positive and negative values
- Simulation verified:
  - 8-bit left shift correctly moves bits [23:0] to [31:8]
  - Sign extension properly replicates MSB of input

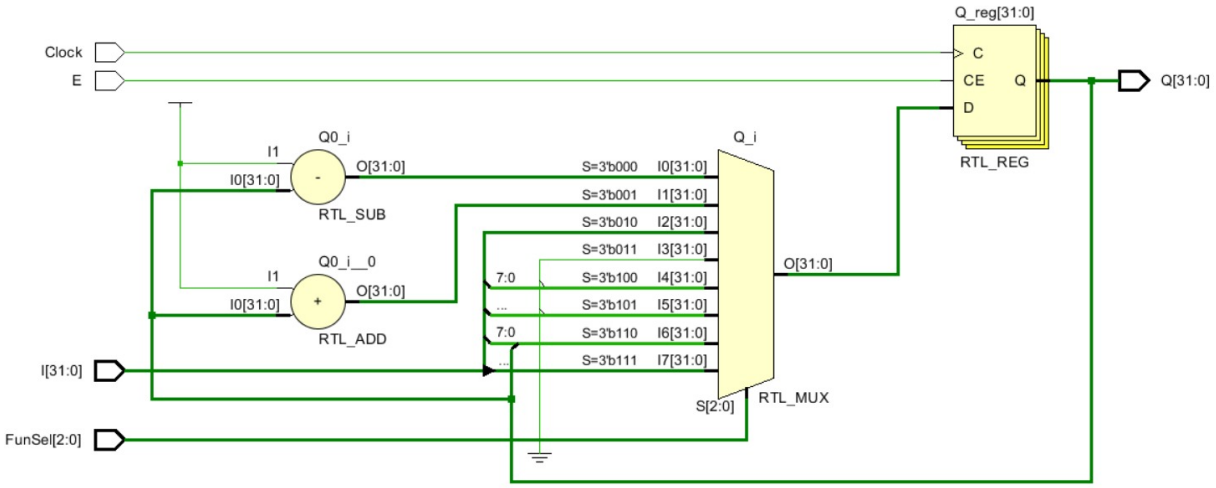

Figure 3: 32-Bit Register Elaboration Diagram

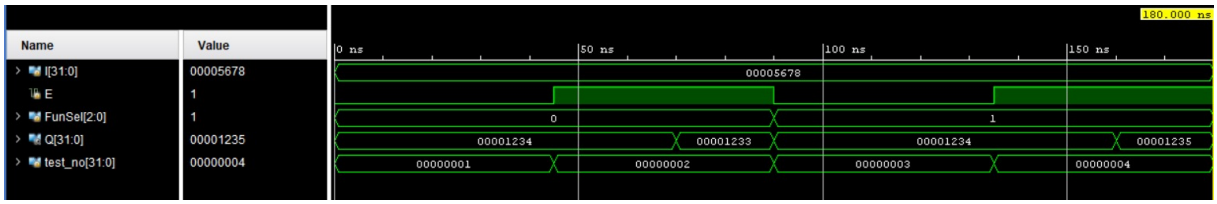

Figure 4: 32-Bit Register Simulation

3.3 Instruction Register

- Byte-wise loading functioning correctly:
  - LH=0 loads lower byte while preserving upper byte
  - LH=1 loads upper byte while preserving lower byte
- Simulation shows correct assembly of 16-bit instructions from two 8-bit writes

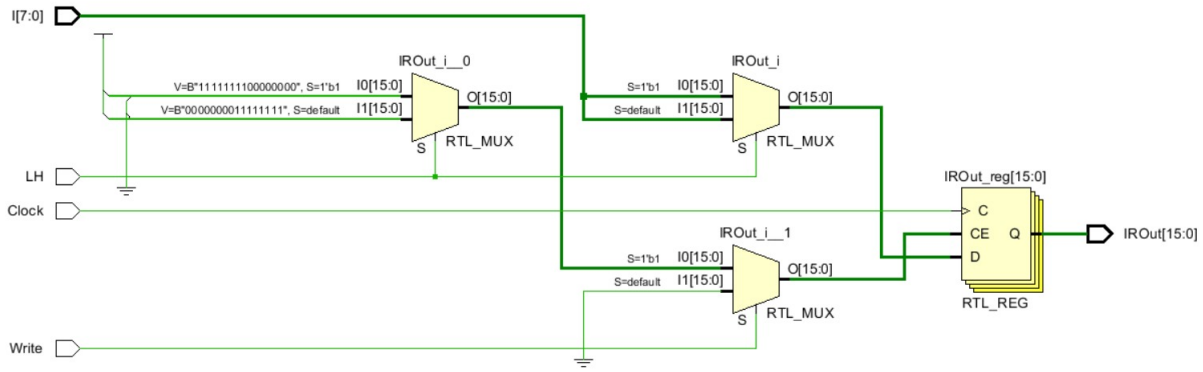

Figure 5: Instruction Register Elaboration Diagram

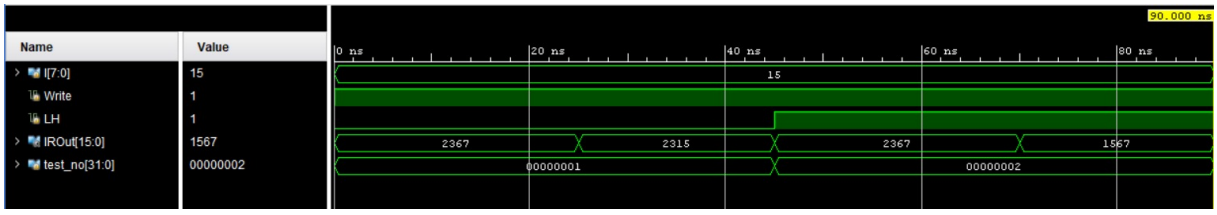

Figure 6: Instruction Register Simulation

3.4 Data Register

- Multi-cycle loading verified:
  - 4 consecutive writes properly assemble 32-bit value
  - Intermediate states show correct partial values
- All control signals validated through edge-case testing

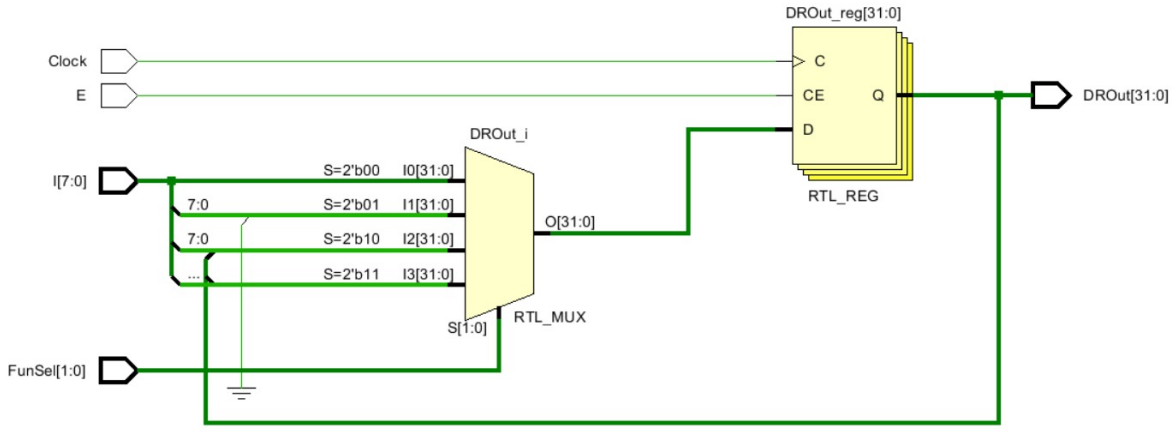

Figure 7: Data Register Elaboration Diagram

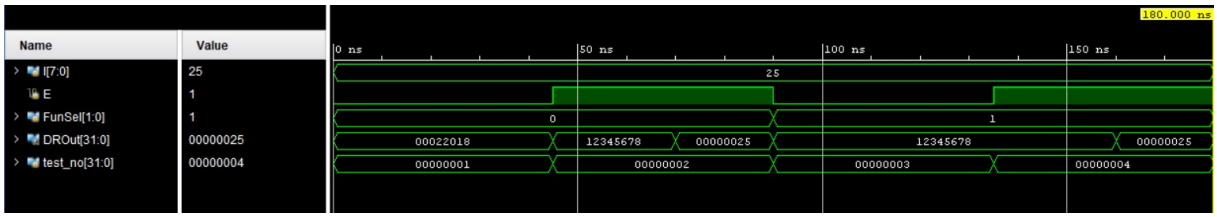

Figure 8: Data Register Simulation

3.5 Register File

- Dual-port access confirmed:
  - Simultaneous reads from OutA and OutB to different registers
  - Write-through behavior when reading and writing same register
- All 256 register combinations tested (16 RegSel × 16 ScrSel)
- Bypass logic correctly handles write/read conflicts

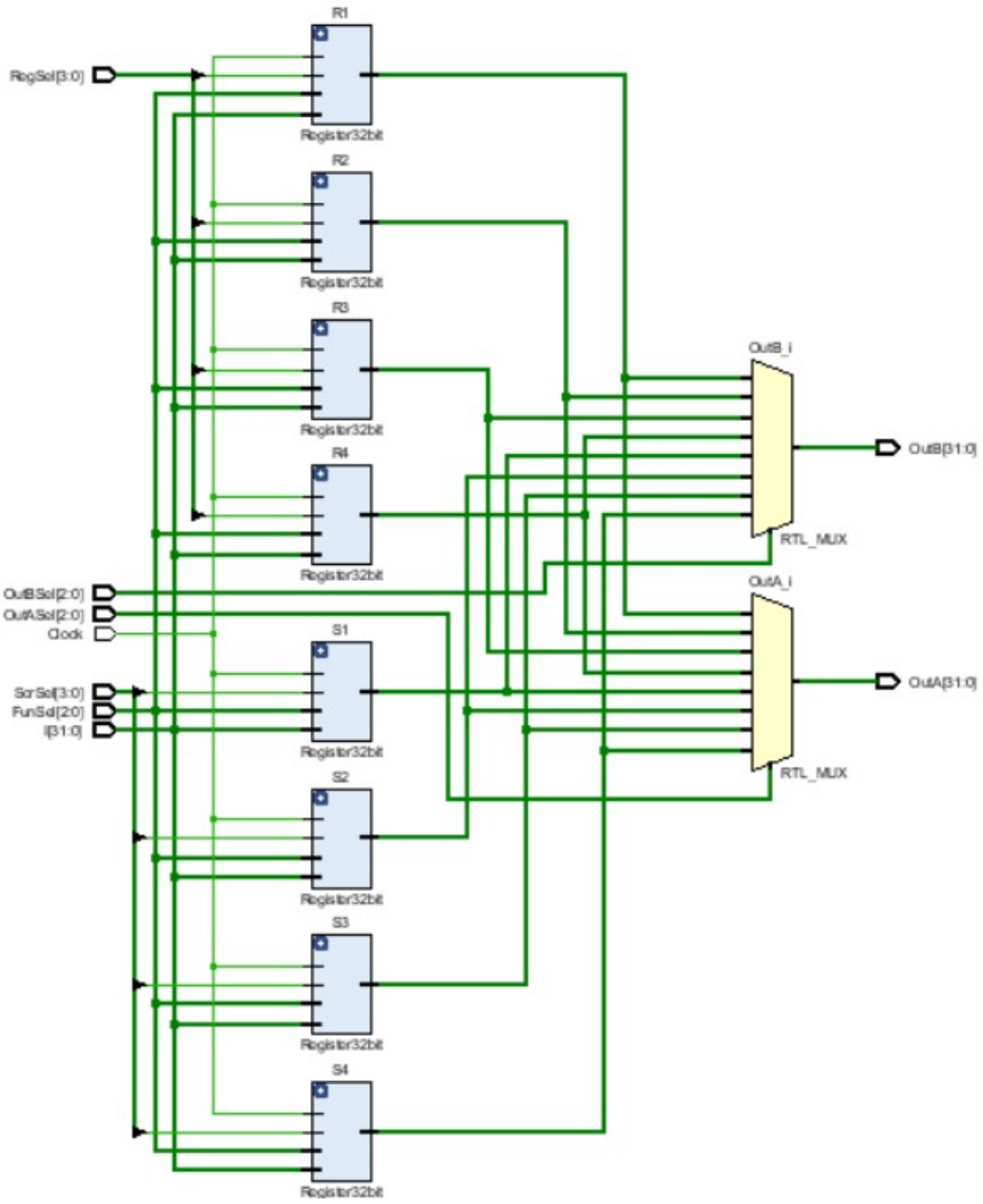

Figure 9: Register File Elaboration Diagram

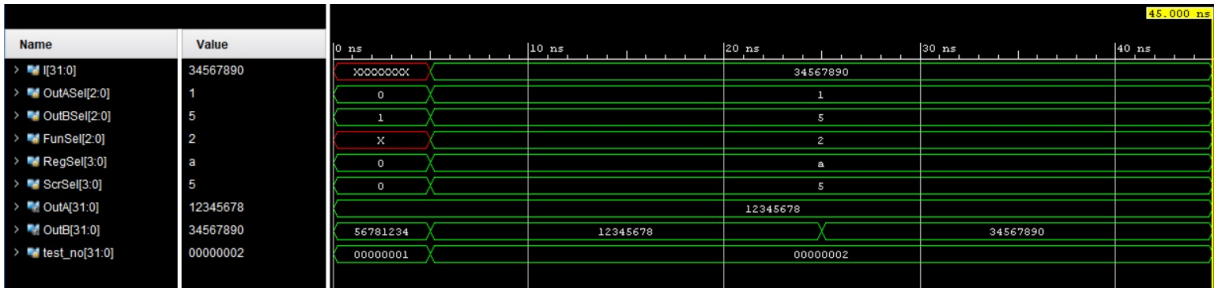

Figure 10: Register File Simulation

3.6 Address Register File

- Special registers functioning:
  - PC shows correct increment behavior
  - SP handles stack operations (push/pop)
  - AR maintains address values
- Output multiplexers verified for all OutCSel/OutDSel combinations

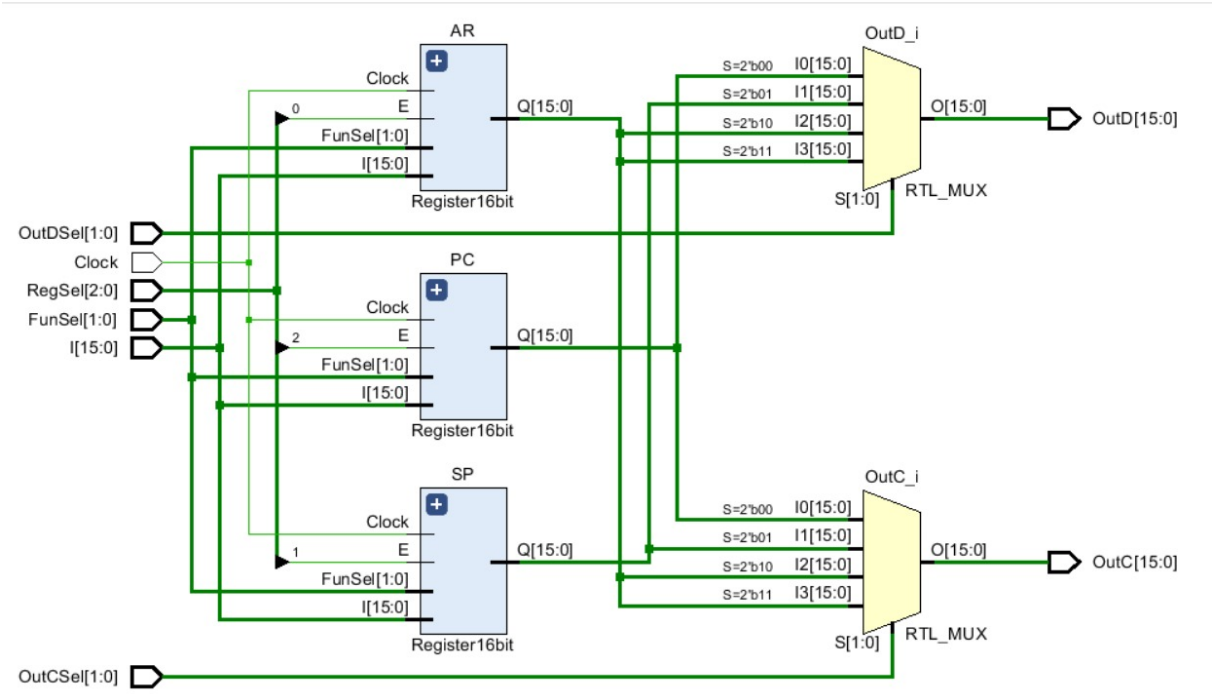

Figure 11: Address Register File Elaboration Diagram

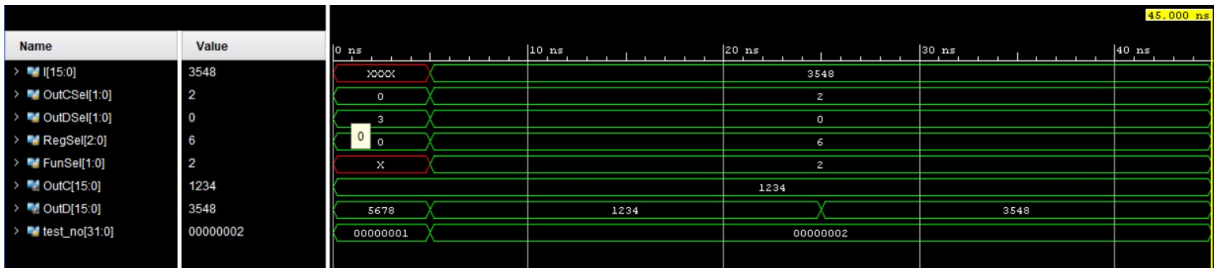

Figure 12: Address Register File Simulation

3.7 Arithmetic Logic Unit

- All 32 operations implemented and verified:
  - Arithmetic operations produce correct 2's complement results
  - Logical operations show proper bitwise behavior
  - Shift operations (logical, arithmetic, circular) working as specified
- Flag generation confirmed:
  - Zero flag sets when result equals zero
  - Carry flag detects overflow in arithmetic operations
  - Negative flag matches MSB of result
  - Overflow flag triggers on signed arithmetic exceptions

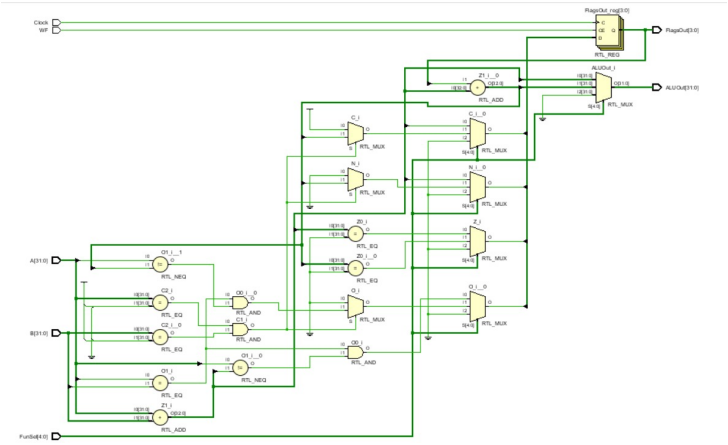

Figure 13: Arithmetic Logic Unit Elaboration Diagram

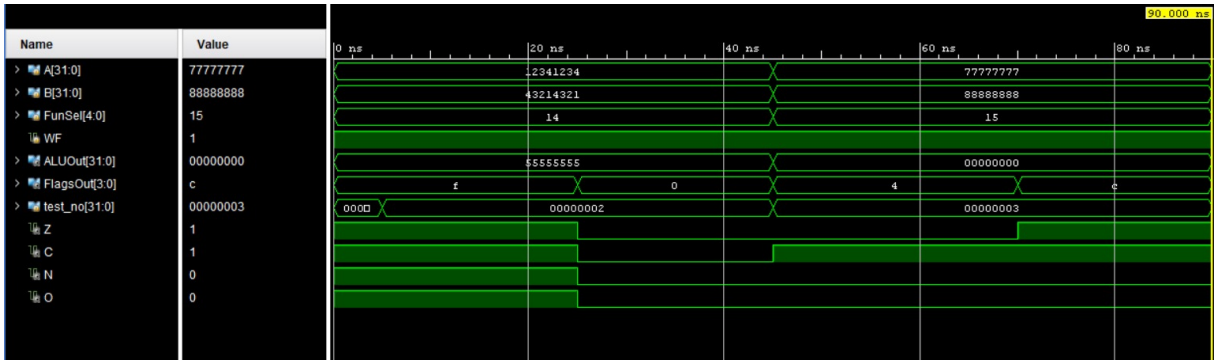

Figure 14: Arithmetic Logic Unit Simulation

3.8 Arithmetic Logic Unit System

- Multiplexers correctly route all data paths:
  - MuxA/MuxB show proper input selection
  - MuxC successfully extracts all byte positions
  - MuxD switches between RF and ARF outputs

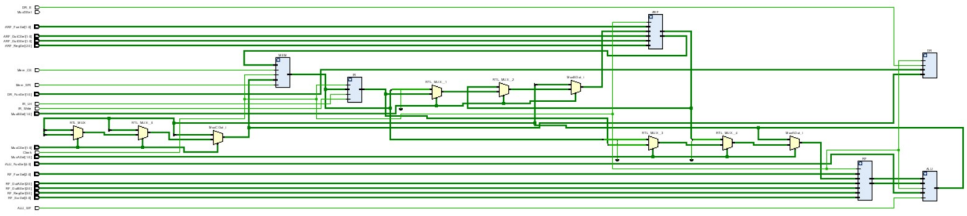

Figure 15: Arithmetic Logic Unit System Elaboration Diagram

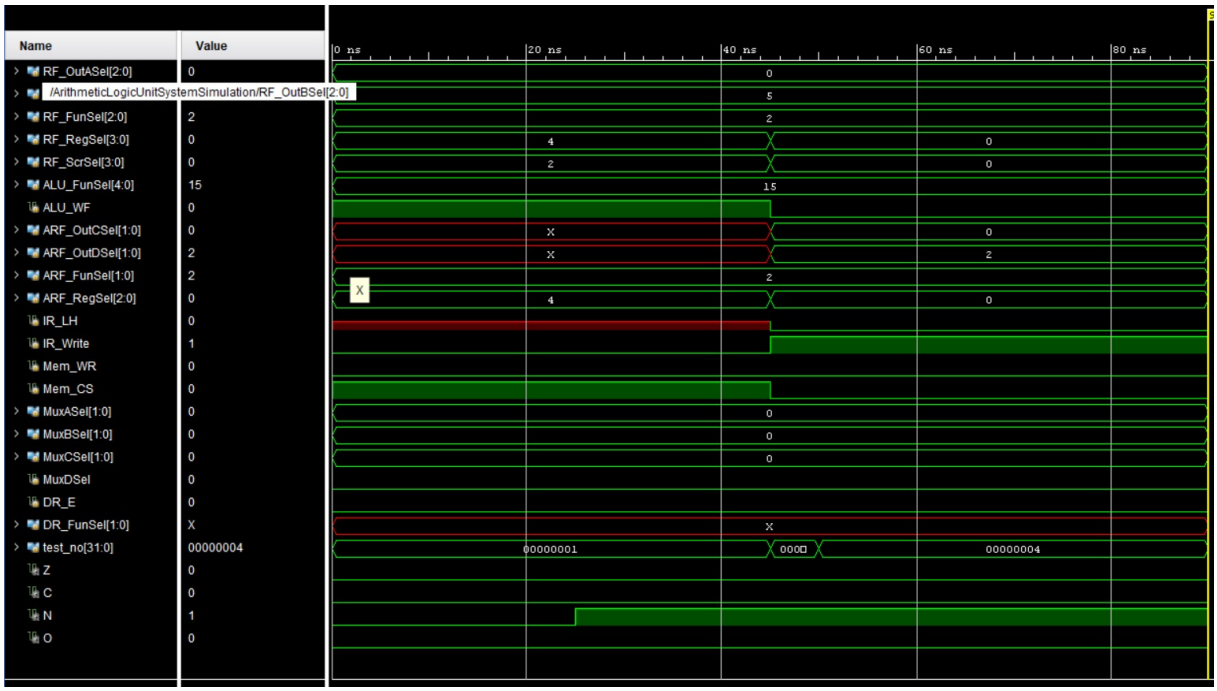

Figure 16: Arithmetic Logic Unit System Simulation

4 DISCUSSION

The project successfully met all design requirements, though several challenges emerged during development. The register file's bypass logic required particular attention to ensure correct behavior during write/read conflicts. ALU flag generation also needed careful implementation to meet timing constraints.

Notably, the decision to implement a unified clocking scheme for the entire system proved beneficial, simplifying synchronization at the cost of some performance flexibility. The modular design approach allowed efficient division of work between team members while maintaining system coherence.

5 CONCLUSION

This project provided valuable hands-on experience with computer organization principles through practical implementation. We successfully designed, implemented, and verified all required components, culminating in a fully functional ALU system. The experience highlighted both the challenges and rewards of hardware design, particularly in areas of timing control and system integration.

REFERENCES

[1] Patterson, D. A., & Hennessy, J. L. (2017). *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann.

[2] Harris, D. M., & Harris, S. L. (2012). *Digital Design and Computer Architecture*. Morgan Kaufmann.