ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION PROJECT 2 REPORT

CRN	:	22599
LECTURER	:	Prof. Dr. Mustafa Ersel Kamaşak

GROUP MEMBERS:

150230084	:	Mehmet Fuat Karakaya
150220049	:	Furkan Kalay

SPRING 2025

Contents

1 INTRODUCTION [10 points]

In this project, we designed a hardwired control unit for an Arithmetic Logic Unit (ALU) system that was developed in a previous project. The primary goal was to implement a control unit capable of fetching, decoding, and executing a specific set of instructions as defined in the project documentation. The control unit is responsible for generating the appropriate control signals at precise clock cycles to manage data flow and operations within the CPU architecture. This report details the design of the control unit, the instruction set architecture, timing analysis including clock cycles per instruction, and the verification of the system.

The task distribution is as follows:

- Furkan Kalay: Implementing Instructions and Debugging of the Control Unit
- Mehmet Fuat Karakaya: Implementing Instructions and Debugging of the Control Unit

2 MATERIALS AND METHODS [40 points]

2.1 Materials

- Verilog HDL for hardware description and implementation.
- A simulation environment (e.g., ModelSim or a similar Verilog simulator) for testing and debugging.
- The pre-designed Arithmetic Logic Unit System from Project 1, which includes:
  - Arithmetic Logic Unit (ALU)
  - Register File (RF)
  - Address Register File (ARF)
  - Data Register (DR)
  - Instruction Register (IR)
  - Memory Unit (MEM)
- Multiplexers (MUX) for data path selection.

2.2 Methods

The design of the hardwired control unit was approached by first understanding the instruction formats and the operations required for each instruction. A state machine was then designed to manage the sequence of operations for instruction fetch, decode, and execution.

2.2.1 Control Unit Design

The control unit is implemented as a hardwired state machine within the `CPUSystem.v` module. The state of the control unit is managed by a 12-bit register `T`, which acts as a cycle counter. This register is initialized to `12'b000000000001` and shifts left in each clock cycle (`T <= {T[10:0], T[11]}`), effectively stepping through different time states. An instruction's execution is completed when the `T_Reset` signal is asserted, which resets `T` for the next instruction fetch.

Instruction Fetch: As per the project specification, instructions are 16 bits wide but the memory (RAM) has an 8-bit data output. Therefore, fetching a single instruction requires two clock cycles:

- **Cycle 1 (T = `12'b...001`):** The LSB of the instruction (IR[7:0]) is loaded from the memory location pointed to by the Program Counter (PC). The PC is then incremented.
- **Cycle 2 (T = `12'b...010`):** The MSB of the instruction (IR[15:8]) is loaded from the new memory location pointed to by PC (which was incremented in the previous cycle). The PC is incremented again.

After these two cycles, the complete 16-bit instruction is available in the Instruction Register (IR).

Instruction Decode and Execution: Execution begins from the third clock cycle onwards (when `T` becomes `12'b...100`). The 6-bit opcode, extracted from IR[15:10], determines the specific sequence of control signals to be generated. A large `case` statement based on the `Opcode` and further nested `case` statements based on the `T` state value dictate the operations in each execution cycle. Control signals for MUX selection, register enables, memory read/write, and ALU functions are asserted according to the current instruction and state.

The control unit generates signals such as `RF_FunSel`, `ALU_FunSel`, `Mem_WR`, `Mem_CS`, `IR_Write`, MUX select signals, etc., to orchestrate the datapath components of the `ArithmeticLogicUnitSystem`.

2.2.2 Instruction Formats and Opcodes

Instruction Formats: Two main instruction formats are supported: one with an address reference and one without.

The instructions with an address reference have the format shown in Figure 1.

OPCODE (6-bit)	RSEL (2-bit)	ADDRESS (8-bit)

Figure 1: Instructions with an address reference.

The instructions without an address reference have the format shown in Figure 2.

OPCODE (6-bit)	DSTREG(3-bit)	SREG1 (3-bit)	SREG2 (3-bit)	0

Figure 2: Instructions without an address reference.

Opcodes, Symbols, Descriptions, and Clock Cycles: The following table details the instructions supported by the CPU, their opcodes, symbolic representation, a brief description, and the total number of clock cycles required for their execution (including the 2 fetch cycles). The clock cycles are derived from the control unit logic in `CPUSystem.v`.

Table 1: OPCODE field, symbols, descriptions, and clock cycles.

Opcode (Hex)	Symbol	Description	Clock Cycles
0x00	BRA	PC $\leftarrow$ VALUE	3
0x01	BNE	If Z=0 THEN PC $\leftarrow$ VALUE	3
0x02	BEQ	If Z=1 THEN PC $\leftarrow$ VALUE	3
0x03	POPL	SP $\leftarrow$ SP+1, Rx $\leftarrow$ M[SP] (16-bit)	6
0x04	PSHL	M[SP] $\leftarrow$ Rx, SP $\leftarrow$ SP-1 (16-bit)	4
0x05	POPH	SP $\leftarrow$ SP+1, Rx $\leftarrow$ M[SP] (32-bit)	8
0x06	PSHH	M[SP] $\leftarrow$ Rx, SP $\leftarrow$ SP-1 (32-bit)	6
0x07	CALL	M[SP] $\leftarrow$ PC, SP $\leftarrow$ SP-1, PC $\leftarrow$ VALUE (16-bit)	5
0x08	RET	SP $\leftarrow$ SP+1, PC $\leftarrow$ M[SP] (16-bit)	6
0x09	INC	DSTREG $\leftarrow$ SREG1 + 1	5
0x0A	DEC	DSTREG $\leftarrow$ SREG1 - 1	5
0x0B	LSL	DSTREG $\leftarrow$ LSL SREG1	3
0x0C	LSR	DSTREG $\leftarrow$ LSR SREG1	3

Continued on next page

Table 1 – continued from previous page

Opcode (Hex)	Symbol	Description	Clock Cycles
0x0D	ASR	DSTREG ← ASR SREG1	3
0x0E	CSL	DSTREG ← CSL SREG1	3
0x0F	CSR	DSTREG ← CSR SREG1	3
0x10	NOT	DSTREG ← NOT SREG1	3
0x11	AND	DSTREG ← SREG1 AND SREG2	3 or 4*
0x12	ORR	DSTREG ← SREG1 OR SREG2	3 or 4*
0x13	XOR	DSTREG ← SREG1 XOR SREG2	3 or 4*
0x14	NAND	DSTREG ← SREG1 NAND SREG2	3 or 4*
0x15	ADD	DSTREG ← SREG1 + SREG2	3 or 4*
0x16	ADC	DSTREG ← SREG1 + SREG2 + CARRY	3 or 4*
0x17	SUB	DSTREG ← SREG1 - SREG2	3 or 4*
0x18	MOV	DSTREG ← SREG1	3
0x19	MOVL	Rx[7:0] ← IMMEDIATE (8-bit)	4
0x1A	MOVSH	Rx[31:8] ← Rx[23:0] ≪ 8, Rx[7:0] ← IMMEDIATE (8-bit)	4
0x1B	LDARL	DSTREG ← M[AR] (16-bit)	5
0x1C	LDARH	DSTREG ← M[AR] (32-bit)	7
0x1D	STAR	M[AR] ← SREG1 (SREG1: ARF 16-bit, RF 32-bit)	5 (ARF) or 7 (RF)
0x1E	LDAL	Rx ← M[ADDRESS] (16-bit)	6
0x1F	LDAH	Rx ← M[ADDRESS] (32-bit)	8
0x20	STA	M[ADDRESS] ← Rx (32-bit)	7
0x21	LDDRL	DR ← M[AR] (16-bit)	4
0x22	LDDRH	DR ← M[AR] (32-bit)	6
0x23	STDR	DSTREG ← DR	3
0x24	STRIM	M[AR+OFFSET] ← Rx (32-bit)	8

* For instructions 0x11-0x17 (AND, ORR, XOR, NAND, ADD, ADC, SUB): 4 cycles if both SREG1 and SREG2 are from ARF (requiring one to be temporarily moved to a scratch register); 3 cycles otherwise.

The detailed schematic of the CPU system is shown in Figure 3.

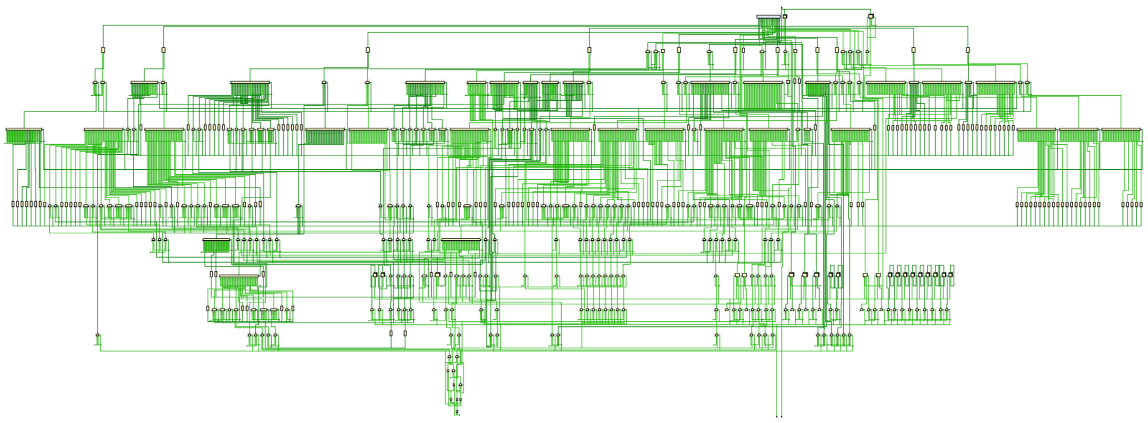

Figure 3: Schematic of CPU System including the Control Unit (Conceptual).

The symbolic diagram of the ALU system used as a component is shown in Figure 4, and its detailed schematic is provided in Figure 5.

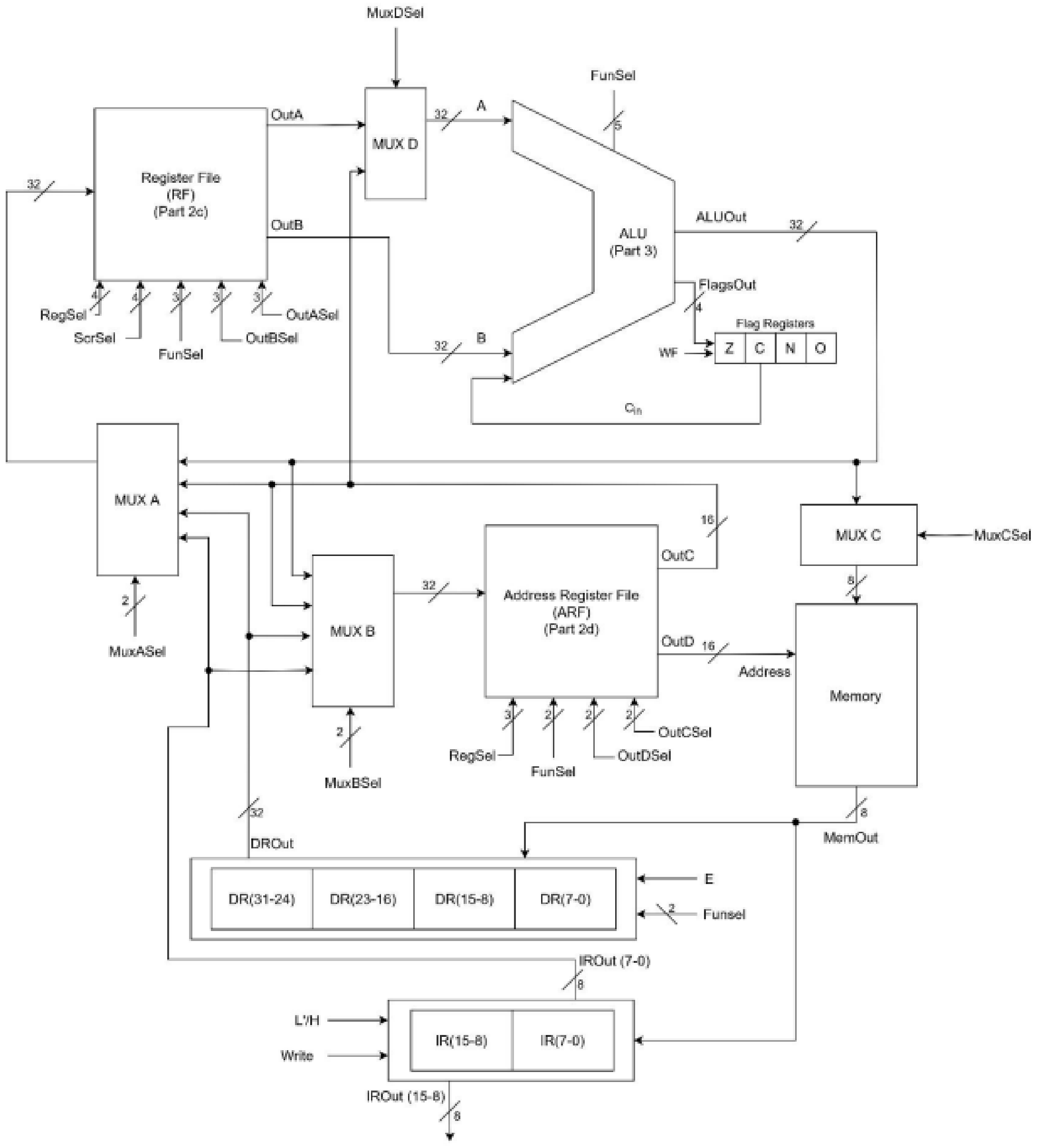

Figure 4: Symbolic representation of the Arithmetic Logic Unit System (from Project 1).

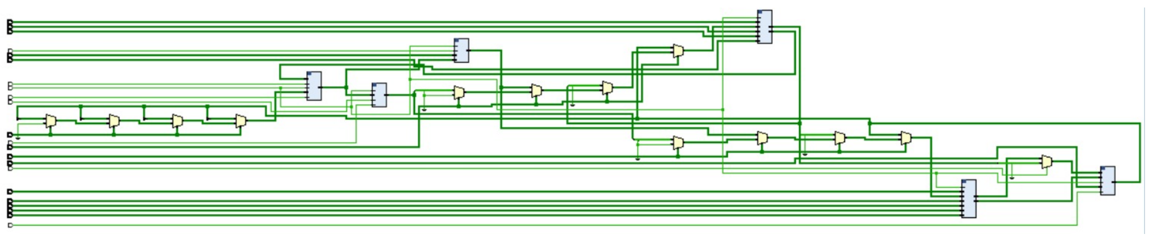

Figure 5: ALU System schematic (from Project 1).

Memory Module

The system includes a memory module `MEM` that is accessed through an address, typically provided by the ARF (e.g., PC for instruction fetch, AR or SP for data access). Data is written to or read from memory depending on the control signals `Mem_WR` (Write Enable) and `Mem_CS` (Chip Select). The 8-bit memory output is then used as input to the Instruction Register (IR) during fetch, or the Data Register (DR) during data load operations, depending on the control context.

Clock Synchronization

All sequential components in this system, including registers and the control unit's state register `T`, are synchronized using a single clock input `Clock`. This ensures that data flow, memory operations, and register updates occur in a coordinated manner, cycle by cycle. The `Reset` signal is used to initialize the system to a known state.

3 RESULTS [15 points]

The `CPUSystem.v` module, encompassing the hardwired control unit and the integrated ALU system, was tested and debugged using the provided simulation environments. It is reported that all tests passed successfully. Behavioral simulation results verified that each component and instruction functioned correctly under various data input scenarios.

Module-Level Verification

Each module within the ALU system and the control unit's logic for each instruction were conceptually verified by tracing control signals and data paths for expected behavior. The cycle-by-cycle operation, as defined by the `T` states in the control unit, was analyzed to ensure correct sequencing. For example, the fetch and decode stages (first two clock cycles, T=`...001` and T=`...010`) correctly load the LSB and MSB of the instruction into the IR, and increment the PC appropriately. Subsequent execution cycles for each opcode correctly manipulate data and registers as per the instruction definitions.The simulation console output confirms the success of the tests.

For CPUSystemSimulation.v, there were 0 Tests Failed and 25 Tests Passed.

For CPUSystemSimulation_Factorial.v, there were 0 Tests Failed and 26 Tests Passed.

An excerpt of the typical console output is shown below:

```
CPUSystem Simulation Started
[PASS] Test No: 1, Component: R2, Actual Value: 0x77777777, Expected Value: 0x77777777
[PASS] Test No: 1, Component: R2, Actual Value: 0x00000000, Expected Value: 0x00000000
[PASS] Test No: 2, Component: R1, Actual Value: 0x00000001, Expected Value: 0x00000001
...
[PASS] Test No: 8, Component: AR, Actual Value: 0x00000012, Expected Value: 0x00000012
CPUSystem Simulation Finished
0 Test Failed
25 Test Passed
...
CPUSystem Simulation Started
[PASS] Test No: 70, Component: IROut, Actual Value: 0x00000052, Expected Value: 0x00000052
[PASS] Test No: 70, Component: PC, Actual Value: 0x00000052, Expected Value: 0x00000052
...
[PASS] Test No: 70, Component: MEM[SP], Actual Value: 0x0000002e, Expected Value: 0x0000002e
CPUSystem Simulation Finished
0 Test Failed
26 Test Passed
```

Clock Cycle Analysis for Example Code Snippet

The project documentation provides a simple example code snippet. We analyze its execution to determine the total clock cycles required. The iteration count is determined by R1, which is set to 6 by the `MOVL R1, 0x00` and `MOVSH R1, 0x06` instructions. The loop (from `LDARH R4` to `BNE LABEL`) executes 6 times.

The instructions and their cycle counts are:

- `BRA 0x00` (Opcode 0x00): 3 cycles.
- `MOVL R1, 0x00` (Opcode 0x19): 4 cycles.
- `MOVSH R1, 0x06` (Opcode 0x1A): 4 cycles. (R1 is now 6)
- `MOVL R2, 0x00` (Opcode 0x19): 4 cycles.
- `MOVL R3, 0xB0` (Opcode 0x19): 4 cycles.
- `MOV AR, R3` (Opcode 0x18): 3 cycles.

Loop (executes 6 times):

- `LDARH R4` (Opcode 0x1C): 7 cycles.
- `ADD R2, R2, R4` (Opcode 0x15): 3 cycles.

- `INC AR, AR` (Opcode 0x09): 5 cycles.
- `DEC R1, R1` (Opcode 0x0A): 5 cycles.
- `BNE LABEL` (Opcode 0x01): 3 cycles.

Total cycles per loop iteration = $7 + 3 + 5 + 5 + 3 = 23$ cycles. Total for 6 iterations = $6 \times 23 = 138$ cycles.

After loop:

- `INC AR, AR` (Opcode 0x09): 5 cycles.
- `STAR R2` (Opcode 0x1D): 7 cycles.

Total clock cycles for the snippet = (BRA) + (Setup) + (Loop) + (Post-loop) Total = $3 + (4 + 4 + 4 + 4 + 3) + 138 + (5 + 7)$ Total = $3 + 19 + 138 + 12 =$ **172 cycles**.

Behavioral Simulation Visuals

To further demonstrate the correct functionality of each Instruction, waveform outputs from behavioral simulations were captured. Below are selected simulation snapshots for the `CPUSystemSimulation.v` and `CPUSystemSimulation_Factorial.v` test files:

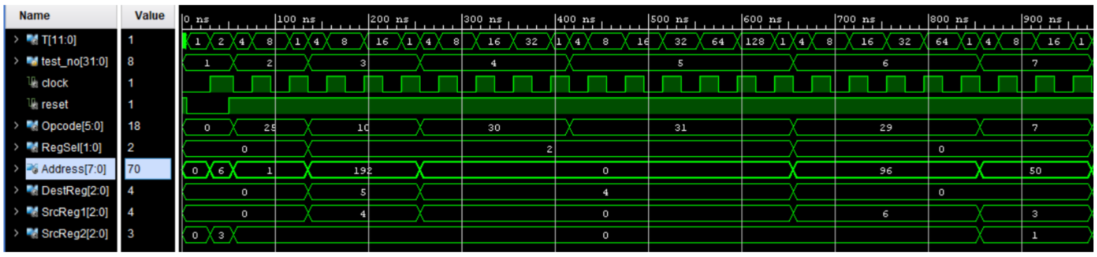

Figure 6: Simulation output of CPUSystem Simulation.

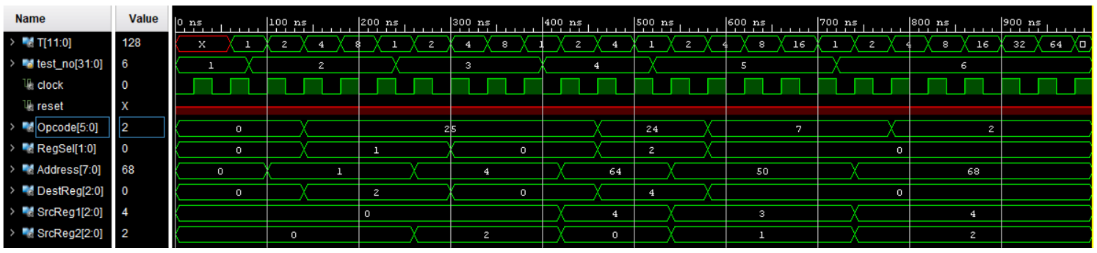

Figure 7: Simulation output of CPUSystem Simulation implementing Factorial Operation.

The combination of text-based simulation logs and waveform results confirm that all Instructions operate as intended, both individually in `CPUSystemSimulation` and subsequently in the `CPUSystemSimulation_Factorial` tests.

4 DISCUSSION [25 points]

The design of the hardwired control unit for the given ALU system was successfully completed. The control unit utilizes a state-based approach, managed by the `T` cycle counter, to generate control signals for instruction fetch, decode, and execution phases.

Design Choices and Complexity: A hardwired control unit, as implemented, generates control signals directly from the instruction bits (opcode) and the current state (`T`). This approach can lead to fast operation since the logic is combinational (after state and instruction are latched). However, the design complexity increases significantly with the number of instructions and the complexity of each instruction's micro-operations. The Verilog code for the control unit, particularly the nested `case` statements, reflects this complexity. Any modification or addition of instructions would require redesigning parts of this logic.

Performance and Timing: The clock cycle counts for each instruction (Table 1) vary based on the complexity of the operation, especially the number of memory accesses or sequential internal operations required. Instructions involving 32-bit memory accesses (e.g., `POPH`, `PSHH`, `LDAH`, `STA`) naturally take more cycles than simple ALU operations or 16-bit transfers. The two-cycle instruction fetch is a fixed overhead for every instruction due to the 8-bit memory interface.

Control Signal Generation: The `CPUSystem.v` module demonstrates how control signals are asserted at specific `T` states for each opcode. For instance, memory access instructions carefully manage `Mem_CS`, `Mem_WR`, `ARF_OutDSel` (for memory addressing), and `DR_E`/`DR_FunSel` (for data staging through the Data Register). ALU operations set `ALU_FunSel`, select operands via MUXes (`MuxDSel`, `MuxASel`, `MuxBSel`), and route results.

Debugging and Verification Challenges: Debugging a hardwired control unit can be challenging due to the direct mapping of instruction bits to control signals. Any error in this mapping can lead to incorrect behavior that might be hard to trace. Thorough simulation with comprehensive test cases, covering all instructions and operand combinations, is crucial, as indicated by the successful simulation results.

5 CONCLUSION [10 points]

This project involved the design and implementation of a hardwired control unit for a previously developed ALU system. The control unit successfully orchestrates the fetch-decode-execute cycle for a defined set of instructions, managing data paths and component operations through precisely timed control signals.

The analysis shows that instruction execution times vary, with an overhead of two clock cycles for instruction fetch due to the memory interface. The performance of the system, in terms of clock cycles per instruction, is documented and provides a basis for understanding the CPU's operational speed. The example code snippet analysis demonstrated how these individual instruction timings contribute to overall program execution time.

The hardwired approach offers speed but at the cost of flexibility and design complexity, especially as the instruction set grows. The successful simulation results confirm the functional correctness of the designed control unit and its integration with the ALU system, meeting the project objectives. Future work could explore microprogrammed control for enhanced flexibility or optimizations in the control logic for specific instructions.

REFERENCES

[1] BLG222E Project2 Documentation, Istanbul Technical University, 2024-2025.