UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

DOCUMENTATION WORK

-ALU-

Structure of Computer Systems

Name: Nechita Florina-Elena

Group: 30434_2

Coordinator: Mădălin Neagu

1. Introduction

1.1 Context

The goal of this project is to design, implement, and document the development of an Arithmetic-Logic Unit (ALU). The ALU is a crucial component in any computing system, responsible for performing a wide range of arithmetic and logical operations. This documentation outlines the design and implementation details of the ALU, along with its key objectives, functionalities, and the methods for testing its correctness. The ALU supports operations including addition, subtraction in Two's complement (C2), increment, decrement, logical AND, logical OR, logical NOT. Furthermore, the ALU incorporates an accumulator for one input operand and the result, as well as an additional circuit dedicated to multiplication and division operations.

1.2 Objectives

The main objectives of this project are as follows:

- Design an ALU capable of performing essential arithmetic operations, logical operations, and rotation operations.
- Include logical operations such as AND, OR, and NOT.
- Utilize an accumulator to store one input operand and the result.
- Develop an additional circuit for performing multiplication and division operations.

1.3 Specifications

The ALU will be simulated using the Xilinx IDE and then programmed onto a Nexys A7 board. The size of operands, memory requirements, and instruction encoding details are considered during the implementation.

2. Operations

2.1 Addition and Subtraction in C2

The ALU is designed to execute addition and subtraction operations using the Two's Complement representation. It can handle both signed and unsigned numbers.

2.2 Logical Operations (AND, OR, NOT)

Logical operations such as AND, OR, and NOT are supported, providing bitwise manipulation capabilities for data processing.

2.3 Operand Handling

An accumulator is used to store one of the input operands and the result of the operations. This facilitates the execution of multi-step operations and simplifies the data handling process.

2.4 Multiplication and Division Circuit

In addition to basic arithmetic and logical operations, the ALU is equipped with a supplementary circuit dedicated to multiplication and division. This specialized circuit ensures accurate and efficient processing of these complex mathematical operations.

3. Design and implementation

3.1 Architecture

The design of the Arithmetic-Logic Unit (ALU) is based on a modular architecture that allows for efficient execution of various operations. The architecture includes the following key components:

3.1.1 Arithmetic Unit

Responsible for addition, subtraction, multiplication, and division operations.
Utilizes Two's Complement representation for signed numbers in addition and subtraction.
Implements efficient algorithms for multiplication and division.

3.1.2 Logical Unit

Handles logical operations such as AND, OR, NOT.
Provides support for data manipulation and bitwise operations.

3.1.3 Accumulator

Stores one input operand and the result of operations.
Facilitates multi-step operations by retaining intermediate values.

3.2 Circuit Design

The ALU's circuit design is optimized for performance and efficiency. It includes the following elements:

3.2.1 Arithmetic Circuit

Consists of full adders for addition and subtraction.
Multiplier and divider circuits for multiplication and division operations.
Appropriate control logic for operation selection.

3.2.2 Logical Circuit

Logical gates for AND, OR, and NOT operations.
Shift registers for left and right rotation.

3.2.3 Accumulator Circuit

Registers for storing input operands and results.
Control logic for data transfer between registers and the arithmetic or logical units.

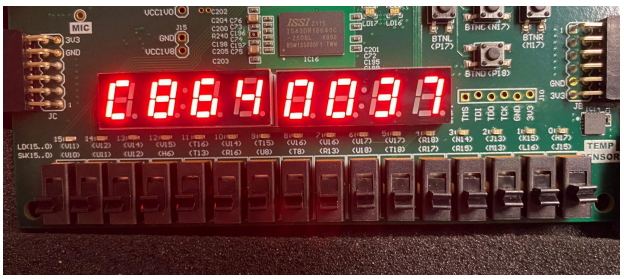
4. ALU Operation Codes

The ALUOp input is a 3-bit vector defining the operation to be performed by the ALU. The following operation codes are supported:

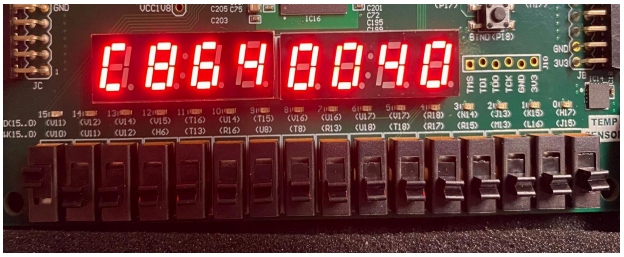
NOT (000): Bitwise NOT of operand A.
AND (001): Bitwise AND of operands A and B.
OR (010): Bitwise OR of operands A and B.
XOR (011): Bitwise XOR of operands A and B.
ADDITION (100): Addition of operands A and B.
SUBTRACTION (101): Subtraction of operand B from A.
MULTIPLICATION (110): Multiplication of operands A and B.
DIVISION (111): Division of operand A by B.

For the hardcoded values of A = C8 (hex), 200 (decimal) and B = 64 (hex), 100 (decimal):

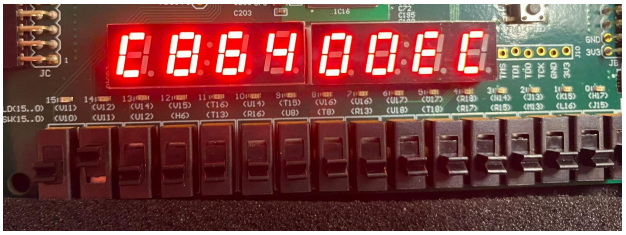
NOT (000):
Input: A = C8
Expected Result: ~C8 = 37
~C8 means taking the bitwise NOT of each corresponding bit in the binary representation of C8.
C8 in binary is 11001000, and the bitwise NOT of each bit gives 00110111 in binary.
In hexadecimal, 00110111 is 37. So, the correct result is 37 in hex, or 55 in decimal.



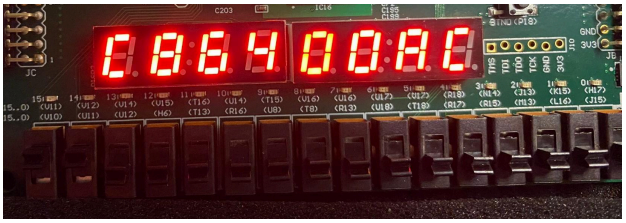
AND (001):
Input: A = C8, B = 64
Expected Result: C8 AND 64 = 40
C8 AND 64 means taking the bitwise AND of each corresponding bit in the binary representations of C8 and 64.
C8 in binary is 11001000, 64 in binary is 01000000, and the AND of each bit gives 01000000 in binary.
In hexadecimal, 01000000 is 40. So, the correct result is 40 in hex, or 64 in decimal.



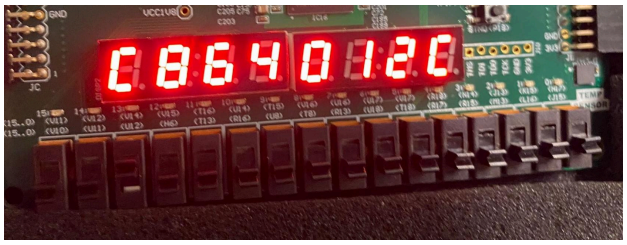
OR (010):
Input: A = C8, B = 64
Expected Result: C8 OR 64 = EC
C8 OR 64 means taking the bitwise OR of each corresponding bit in the binary representations of C8 and 64.
C8 in binary is 11001000, 64 in binary is 01000000, and the OR of each bit gives 11001000 in binary.
In hexadecimal, 11001000 is EC. So, the correct result is EC in hex, or 236 in decimal.



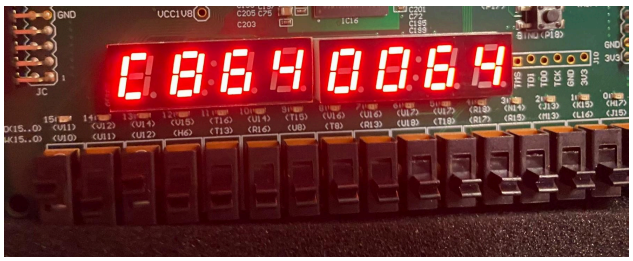
XOR (011):
Input: A = C8, B = 64
Expected Result: C8 XOR 64 = AC
C8 XOR 64 means taking the bitwise XOR of each corresponding bit in the binary representations of C8 and 64.
C8 in binary is 11001000, 64 in binary is 01000000, and the XOR of each bit gives 10001000 in binary.
In hexadecimal, 10001000 is AC. So, the correct result is AC in hex, or 172 in decimal.



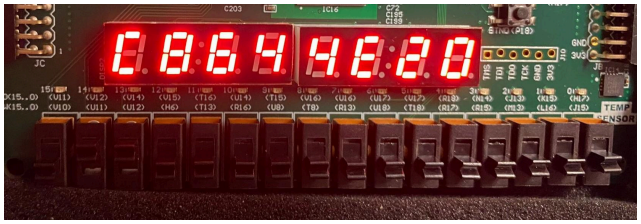
ADDITION (100):
Input: A = C8, B = 64
Expected Result: C8 + 64 = 12C
C8 + 64 is the sum of the decimal values. C8 in hex is 200, and 64 in hex is 100. The sum is 300.
In hexadecimal, 300 is 12C. So, the correct result is 12C in hex, or 300 in decimal.



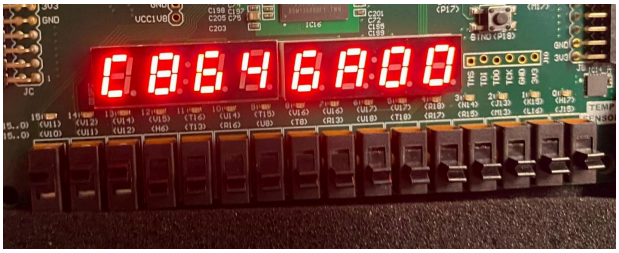
SUBTRACTION (101):
Input: A = C8, B = 64
Expected Result: C8 - 64 = 64
C8 - 64 is the difference of the decimal values. The difference is 200 - 100 = 100.
In hexadecimal, 100 is 64. So, the correct result is 64 in hex, or 100 in decimal.



MULTIPLICATION (110):
Input: A = C8, B = 64
Expected Result: C8 * 64 = 4E20
C8 * 64 is the product of the decimal values. The product is 200 * 100 = 20000.
In hexadecimal, 20000 is 4E20. So, the correct result is 4E20 in hex, or 20000 in decimal.

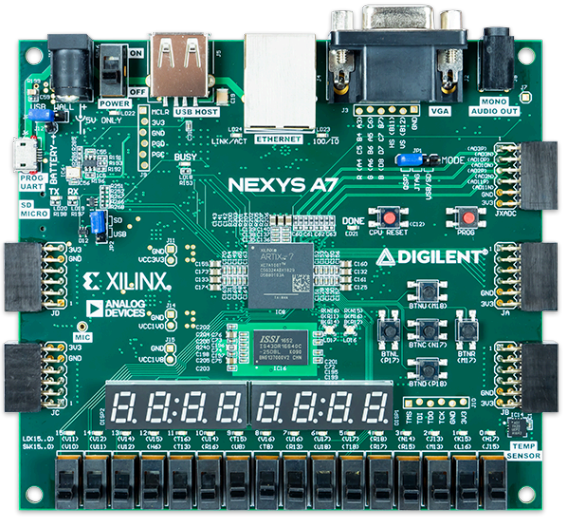


DIVISION (111):
Input: A = C8, B = 64
Expected Result: C8 / 64 = 2
C8 / 64 is the division of the decimal values. The division is 200 / 100 = 2.
In hexadecimal, 2 is 2. So, the correct result is 2 in hex, or 2 in decimal.



5. Implementation on Nexys A7 FPGA

The ALU is designed to be programmed onto the Nexys A7 FPGA board. Specific pins, clock configurations, and resource utilization considerations are detailed in the hardware implementation section (the Constraints).



6. Future Work

While the current implementation meets the specified requirements, future iterations could explore additional features or optimizations. Addressing any identified limitations and considering advancements in FPGA technology would be valuable for future development.

7. Conclusion

In summary, the development of the Arithmetic-Logic Unit (ALU) stands as a pivotal achievement in computer systems design. This project focused on creating a versatile ALU capable of executing essential arithmetic, logical, and rotational operations with efficiency and scalability.

The modular architecture, comprising an Arithmetic Unit, Logical Unit, and Accumulator, ensures optimal performance. Key features include Two's Complement representation for signed numbers, logical operations (AND, OR, NOT), and a dedicated circuit for multiplication and division. The ALU, simulated using Xilinx IDE and programmed onto Nexys A7 FPGA, undergoes rigorous testing to guarantee correctness.

Looking forward, this ALU design serves as a robust foundation for potential enhancements and optimizations. Future iterations may explore additional features while keeping pace with advancements in FPGA technology. This project significantly contributes to the evolution of computer systems, promising continued innovation in the field.