**MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA TECHNICAL UNIVERSITY OF MOLDOVA**

**FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS**

Computer Architecture

***Laboratory work 3: Exercises in Logisim***

Elaborated: st.gr. FAF-213, Sirghi Tudor

Verified: asist.univ., Vladislav Voitcovschi

Chis, ina˘u, 2023

# Introduction

Logisim is a free and open-source digital design and simulation tool that allows users to create and test digital circuits using a variety of logic gates and other digital components. With its user-friendly interface and powerful simulation capabilities, Logisim is an ideal tool for students, hobbyists, and professionals who want to learn about digital circuit design or develop new digital systems.

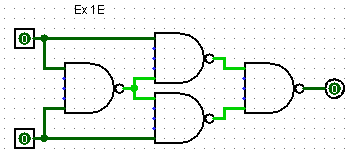
To work with Logisim, you first need to download and install the software on your computer. Once installed, you can start a new project and begin designing your digital circuit by dragging and dropping digital components, such as logic gates, onto the design canvas. You can then connect the input and output lines of the digital components to create the desired logic function.

After designing the digital circuit, you can simulate its behavior by applying input signals and observing the output signals. Logisim provides a range of simulation tools and analysis options that allow you to test and refine your circuit's functionality. You can also save and export your circuit designs in various file formats, such as PNG, SVG, and Verilog.

Logisim supports a wide range of digital components, including logic gates, flip-flops, multiplexers, decoders, and counters, as well as more advanced features such as subcircuits and hierarchical design. By learning how to work with Logisim, you can gain valuable insight into the principles of digital circuit design and develop the skills necessary to create complex digital systems.

**Easy**

**Ex 1E**

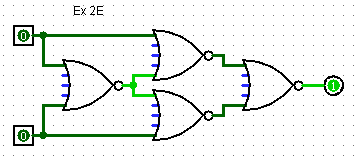


The XOR (exclusive OR) gate is a logical gate that produces an output of "1" only when the two input values are different from each other. If the inputs are the same, the output is "0". The XOR gate can be represented by the following truth table:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A XOR B** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

In this exercise I implemented and XOR Gate with 2 inputs using only NAND Gates. I used 2 inputs and 4 NAND Gates.

**Ex 2E**



The XNOR logical gate is a digital logic gate that performs an exclusive-NOR (XNOR) operation on its inputs. It produces a true output (logic 1) only when both of its inputs are the same (both 1 or both 0), and a false output (logic 0) otherwise.

The XNOR gate is symbolized with an enclosed circle with an "X" inside, and it has two input terminals and one output terminal. The Boolean expression for an XNOR gate with inputs A and B is:

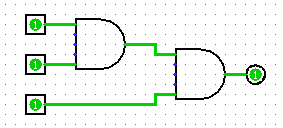
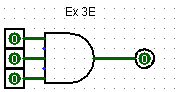
A XNOR B = (A AND B) OR (NOT A AND NOT B)

The truth table for an XNOR gate is:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A XNOR B** |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

In this exercise I implemented and XNOR Gate with 2 inputs using only NOR Gates. I used 2 inputs and 4 NOR Gates.

**Ex 3E**



The AND logical gate is a digital logic gate that performs a logical conjunction operation on its inputs. It produces a true output (logic 1) only when both of its inputs are true (logic 1), and a false output (logic 0) otherwise.

The AND gate is symbolized with an inverted "V" and it has two input terminals and one output terminal. The Boolean expression for an AND gate with inputs A and B is:

A AND B

The truth table for an AND gate is:

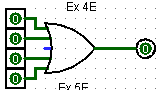
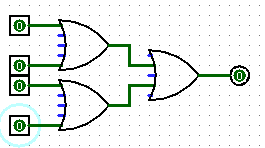
|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A AND B** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

For 3 inputs truth table will be:

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **C** | **A AND B AND C** |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

In this exercise I implemented and AND Gate with 3 inputs using only AND Gates. I used 3 inputs and an AND Gate for first example, and for second example I used 3 inputs and 2 AND Gates. The output for both is the same.

**Ex 4E**

**** ****

The OR logical gate is a digital logic gate that performs a logical disjunction operation on its inputs. It produces a true output (logic 1) if at least one of its inputs is true (logic 1), and a false output (logic 0) only when both inputs are false (logic 0).

The OR gate is symbolized with a curved "V" shape, and it has two input terminals and one output terminal. The Boolean expression for an OR gate with inputs A and B is:

A OR B

The truth table for an OR gate is:

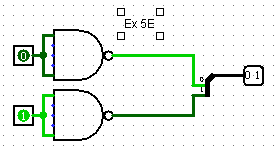
|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A OR B** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

For 4 inputs truth table wiil be:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **A OR B OR C OR D** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

In this exercise I implemented and OR Gate with 4 inputs using only OR Gates. I used 4 inputs and an OR Gate for first example, and for second example I used 4 inputs and 3 OR Gates. The output for both is the same.

**Ex 5E**



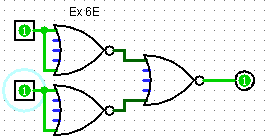
Inverting NOT gates are single input device which have an output level that is normally at logic level “1” and goes “LOW” to a logic level “0” when its single input is at logic level “1”, in other words it “inverts”

The truth table for NOT Gate is:

|  |  |
| --- | --- |
| **A** | **NOT A** |
| 0 | 1 |
| 1 | 0 |

In this exercise I implemented and NOT Gate with 2 inputs using only NAND Gates. I used 2 inputs, a splitter and a NAND Gate.

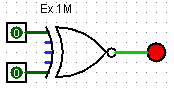
**Ex 6E**



The information about AND Gate can see at Ex 3E.

In this exercise I implemented and AND Gate with 2 inputs using only OR Gates. I used 2 inputs and 2 NOR Gates.

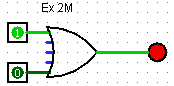
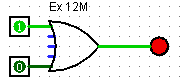
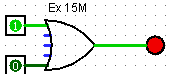
**Medium:**

**Ex 1M**  
****

To solve this exercise I used XNOR Gate. The truth table for XNOR Gate is:

|  |  |  |
| --- | --- | --- |
| A | B | Output |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

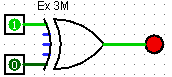
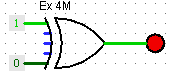
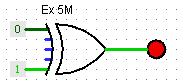
**Ex 2M 12M 15M**

To solve this exercises I used OR Gate. The truth table for OR Gate is:

|  |  |  |
| --- | --- | --- |
| A | B | Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

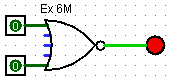
.  **Ex 3M 4M 5M**

To solve this exercises I used XOR Gates. The truth table for XOR Gate is:

|  |  |  |
| --- | --- | --- |
| A | B | Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

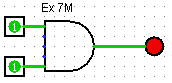
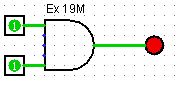
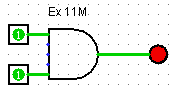
**Ex 6M**

****

To solve this exercises I used NOR Gates. The truth table for NOR Gate is:

|  |  |  |
| --- | --- | --- |
| A | B | Output |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

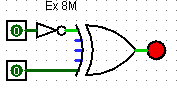
**Ex 7M 11M 19M**

To solve this exercises I used AND Gate. The truth table for AND Gate is:

|  |  |  |
| --- | --- | --- |
| A | B | A AND B |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Ex 8M**

****

To solve this exercises I used XOR Gate and a NOT Gate. The truth table for this circuit is:

|  |  |  |
| --- | --- | --- |
| A | B | Output |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Ex 9M**



To solve this exercises I used a NOT Gate. The truth table for NOT Gate is:

|  |  |
| --- | --- |
| A | B |
| 0 | 1 |
| 1 | 0 |

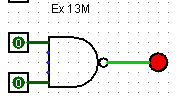
**Ex 10M**



To solve this exercises I used only 1 input. The truth table for this circuit is:

|  |  |
| --- | --- |
| A | B |
| 0 | 0 |
| 1 | 1 |

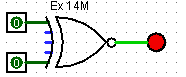
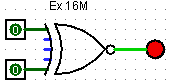
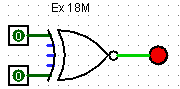
**Ex 13M**



To solve this exercises I used NAND Gate. The truth table for NAND Gate is:

|  |  |  |
| --- | --- | --- |
| A | B | Output |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

**Ex 14M 16M 18M**

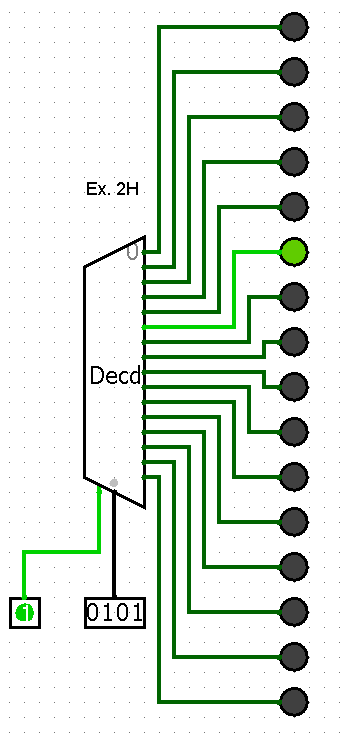
To solve this exercises I used XNOR Gates. The truth table for XNOR Gate is:

|  |  |  |
| --- | --- | --- |
| A | B | Output |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**HARD**

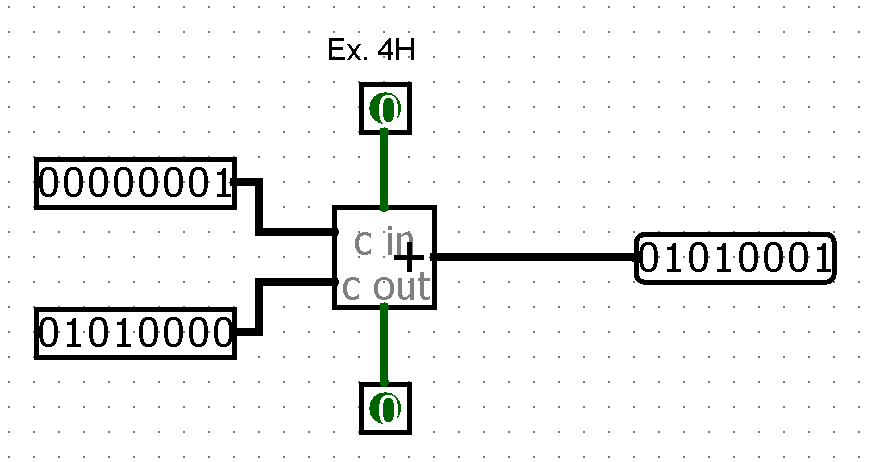
**Ex 2H**

A decoder is a combinational logic circuit that is used to change the code into a set of signals. It is the reverse process of an encoder. A decoder circuit takes multiple inputs and gives multiple outputs. A decoder circuit takes binary data of ‘n’ inputs into ‘2^n’ unique output. In addition to input pins, the decoder has a enable pin. This enables the pin when negated, to make the circuit inactive. in this article, we discuss 3 to 8 line Decoder and demultiplexer.



**Ex 4H**

A full adder adds binary numbers and accounts for values carried in as well as out. A one-bit full adder adds three one-bit numbers, often written as A, B, and Cin; A and B are the operands, and Cin is a bit carried in (in theory from a past addition). The full-adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. binary numbers.



**Ex 6H**

This component divides two values coming in via the west inputs and outputs the quotient on the east output. The component is designed so that it can be cascaded with other dividers to provide support a dividend with more bits than is possible with a single divider: The upper input provides the upper bitWidth bits of the dividend (if it is specified at all), and the rem bits provide the remainder, which can be fed as the upper input into another divider.

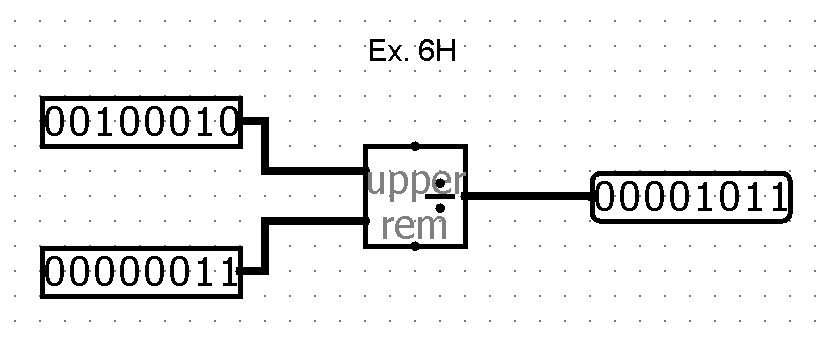
If the divisor is 0, then no division is performed (i.e., the divisor is assumed to be 1).

The divider essentially performs unsigned division. That is, the remainder will always be between 0 and divisor-1. The quotient will always be an integer so that

*quotient \* divisor + remainder = dividend.*

If, however, the quotient does not fit into bitWidth bits, then only the lower bitWidth bits will be reported. The component does not provide any method for accessing the upper bitWidth bits.

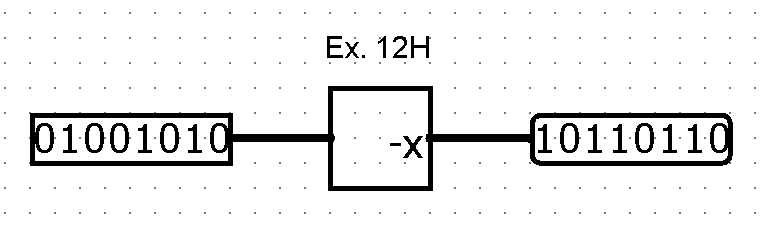
If either of the operands contains some floating bits or some error bits, then the component's outputs will be either entirely floating or entirely error values.



**Ex 12H**

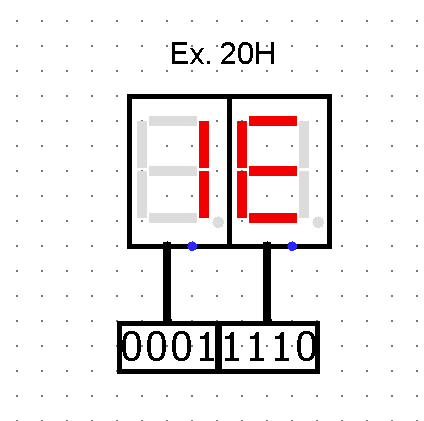
Computes the two's-complement negation of the input. This negation is performed by maintaining all the lower-order bits up to the lowest-order 1, and complementing all bits above that.

If the value to be negated happens to be the least negative value, then its negation (which cannot be represented in two's-complement form), is still the least negative value.



**Ex 20H**

Using a seven-segment display, shows the hexadecimal digit corresponding to the four-bit input. If any of the inputs are not 0/1 (either floating or error), then the display shows a dash ('-'). A separate one-bit input controls the display of the decimal point.



**Requirements:**

**Exercise 1E:** Implement a XOR gate with 2 inputs using NAND gates.

**Exercise 2E:** Implement a XNOR gate with 2 inputs using NOR gates.

**Exercise 3E:** Implement an AND gate with 3 inputs using AND gates.

**Exercise 4E:** Implement an OR gate with 4 inputs using OR gates.

**Exercise 5E:** Implement a NOT gate 2 inputs using NAND gates.

**Exercise 6E:** Implement an AND gate with 2 inputs using NOR gates.

**Exercise 1M:** Implement a logic circuit that will activate the output if 2 inputs are equal.

**Exercise 2M:** Implement a logic circuit that will activate the output if at least one of the 2 inputs is 1.

**Exercise 3M:** Implement a logic circuit that will activate the output if 2 inputs are different.

**Exercise 4M:** Implement a logic circuit that will activate the output if one of the 2 inputs is 1 and the other one is 0.

**Exercise 5M:** Implement a logic circuit that will activate the output if one of two inputs is 0 and the other one is 1.

**Exercise 6M:** Implement a logic circuit that will activate the output if both inputs are 0.

**Exercise 7M:** Implement a logic circuit that will activate the output if both inputs are 1.

**Exercise 8M:** Implement a logic circuit that will activate the output if the input is negated.

**Exercise 9M:** Implement a logic circuit that will activate the output if the input is null.

**Exercise 10M:** Implement a logic circuit that will activate the output if the input is not null.

**Exercise 11M:** Implement a logic circuit that will activate the output if both inputs are 1, but not when both are equal to 0.

**Exercise 12M:** Implement a logic circuit that will activate the output if at least one of the 2 inputs is 1, but not when both are 0.

**Exercise 13M:** Implement a logic circuit that will activate the output if at least one of the 2 inputs is 0, but not when both are equal to 1

**Exercise 14M:** Implement a logic circuit that will activate the output if both inputs are 0 or both are 1, but not when one of the is 0 and the other is 1.

**Exercise 15M:** Implement a logic circuit that will activate the output if both inputs are 1 and at least one of the is 0.

**Exercise 16M:** Implement a logic circuit that will activate the output if both inputs are 0 pr both are 1, but not is one of them is 1 and the other is 0.

**Exercise 17M:** Implement a logic circuit that will activate the output if both inputs are 0 and at least one of them is 1.

**Exercise 18M:** Implement a logic circuit that will activate the output if both inputs are 1 or both are 0, but not is one of the is 0 and the other is 1.

**Exercise 19M:** Implement a logic circuit that will activate the output if both inputs are 1 and 0 does not exist between them.