PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Engenharia de Computação

Gabriel Luís Pinto Cecconello
Paula Cristina Talim Gonçalves
Rafael Vicente Souza e Paula

# UNIDADE LÓGICA ARITIMÉTICA EM VHDL

Belo Horizonte
2025

Gabriel Luís Pinto Cecconello

Paula Cristina Talim Gonçalves

Rafael Vicente Souza e Paula

# UNIDADE LÓGICA ARITIMÉTICA EM VHDL

Trabalho apresentado à disciplina de Sistemas Reconfiguráveis da Pontifícia Universidade Católica de Minas Gerais como requisito parcial para aprovação na disciplina.

Orientador: Prof. Francisco Garcia

Belo Horizonte

2025

Gabriel Luís Pinto Cecconello

Paula Cristina Talim Gonçalves

Rafael Vicente Souza e Paula

# UNIDADE LÓGICA ARITIMÉTICA EM VHDL

Trabalho apresentado à disciplina de Sistemas Reconfiguráveis da Pontifícia Universidade Católica de Minas Gerais como requisito parcial para aprovação na disciplina.

---

Prof. Francisco Garcia - PUC Minas (Orientador)

Belo Horizonte, 29 de abril de 2025.

## RESUMO

Este trabalho apresenta o projeto e a implementação de uma Unidade Lógica e Aritmética (ULA) de 8 bits, utilizando a linguagem de descrição de hardware VHDL. A ULA foi desenvolvida para executar 16 operações distintas, englobando operações aritméticas, lógicas, de rotação e de deslocamento, de acordo com uma seleção de operação de 4 bits. O projeto foi elaborado exclusivamente com código concorrente, garantindo a natureza totalmente combinacional do circuito, sem a utilização de latches ou flip-flops. Para a simulação e validação do funcionamento da ULA, foi utilizado o ambiente de desenvolvimento Quartus II, versão 9.1sp2. Foram realizados casos de teste específicos para cada operação, comprovando que a ULA projetada atende corretamente às especificações, com a correta geração dos sinais de saída — resultado (r), carry/borrow (c_out), sinal de zero (z) e overflow (v) — conforme as funções exigidas. A metodologia adotada permitiu a análise do comportamento do circuito de forma prática e eficiente, servindo de base para o aprofundamento no desenvolvimento de projetos digitais em VHDL.



**Palavras-chave:** ULA; VHDL; projeto digital; simulação; Quartus II.

## ABSTRACT

This work presents the design and implementation of an 8-bit Arithmetic and Logic Unit (ALU) using the Hardware Description Language (HDL) VHDL. The ALU was developed to perform 16 distinct operations, covering arithmetic, logical, rotation, and shifting operations, according to a 4-bit operation selection. The design was exclusively created with concurrent code, ensuring the fully combinational nature of the circuit, without the use of latches or flip-flops. For simulation and validation of the ALU functionality, the Quartus II development environment, version 9.1sp2, was used. Specific test cases were performed for each operation, confirming that the designed ALU correctly meets the specifications, generating the correct output signals — result (r), carry/borrow (c_out), zero flag (z), and overflow (v) — as required by the functions. The methodology adopted enabled practical and efficient analysis of the circuit's behavior, serving as a foundation for further development in digital design projects using VHDL.



**Keywords:** ALU; VHDL; digital design; simulation; Quartus II.

## LISTA DE QUADROS

## LISTA DE FIGURAS

## LISTA DE TABELAS

**SUMÁRIO**

# 1 INTRODUÇÃO

A linguagem VHSIC-HDL, ou apenas VDHL, tem sua sigla originária da expressão “Very High Speed Integrated Circuit Hardware Description Language” ou, traduzindo, “Linguagem de Descrição de Hardware de Circuito Integrado de Altíssima Velocidade” (HEXTSEL, 2003). Apesar de haver muitas semelhanças com linguagens de programação, como a linguagem C, ela foi criada com o objetivo de descrever uma estrutura de um circuito integrado, indicando as interligações dos componentes eletrônicos ou especificando a função dos elementos (HEXTSEL, 2003).

Sendo assim, a fim de explorar os recursos da linguagem VHDL, foi proposto um projeto para desenvolver uma Unidade Lógica Aritmética (ULA). Nesse sentido, a ULA é um componente pertencente à Unidade Central de Processamento (Central Processing Unit ou CPU) responsável por executar operações lógicas, aritméticas, de rotação e de deslocamento em um processador computacional (UNICAMP, 2016). Esse componente possui 3 entradas, sendo 2 para entrada dos dados para realizar a operação e 1 para determinar a operação a ser executada, e 1 saída, que é o resultado da operação (UNICAMP, 2016).

Dito isso, o projeto tem como objetivo desenvolver uma ULA com a linguagem VHDL, uma vez que o componente executa diversas operações, consequentemente, sendo necessário a utilização de várias estruturas disponíveis em VHDL para se descrever o hardware.

# 2 DESENVOLVIMENTO

## 2.1 Metodologia

Para a produção do projeto, foram realizadas algumas definições antes de se começar a desenvolver o programa. Nesse sentido, primeiramente, foram estipuladas as variáveis de entradas do sistema, as quais estão descritas no Quadro 1.

Quadro 1 – Variáveis de entradas do sistema.

| Nome da variável | Tipo de dado que armazena | Funcionalidade |
|---|---|---|
| **a_in [7...0]** | Array de 8 bits | Entrada de dados “a”. |
| **b_in [7...0]** | Array de 8 bits | Entrada de dados “b”, utilizada em operações |

| | | |
|---|---|---|
| | | envolvendo dois operandos. |
| **c_in** | 1 bit | Entrada “carry”, usada para algumas operações aritméticas e de rotação. |
| **op [3...0]** | Array de 4 bits | defini a operação a ser executada. |

Fonte: Autoria própria (2025).

Após a definição das variáveis de entrada do sistema, foram definidas as variáveis de saída, sendo descritas no Quadro 2.

Quadro 2 - Variáveis de saída do sistema.

| Nome da variável | Tipo de dados que armazena | Funcionalidade |
|---|---|---|
| **r [7...0]** | Array de 8 bits | Armazena o resultado da operação |
| **c_out** | 1 bit | Saída de carry/borrow. |
| **z** | 1 bit | Saída de zero. Sinaliza (z = 1) quando o resultado da operação é zero |
| **v** | 1 bit | Saída de overflow. Sinaliza (v = 1) quando há um overflow nas operações de soma e subtração. |

Fonte: Autoria própria (2025).

Por fim, foram estabelecidas as operações que a ULA executaria. A seguir está o Quadro 3, com as operações selecionadas. Nele contém o identificador da operação, representado pela variável de entrada “op [3...0]”, o Mnemônico, a operação executada e as possíveis saídas esperadas, considerando que a saída “z” sempre será 1, se o resultado for igual a zero, e 0, caso contrário.

Quadro 3 - Operações da ULA.

| **op [3...0]** | **Mnemônico** | **Operação** | **r** | **c_out** | **v** |
|---|---|---|---|---|---|
| 0000 | ADD | Soma sem carry in | a_in + b_in | 1, se houver carry no bit mais significativo | 1, se ocorreu overflow |
| 0001 | ADDC | Soma com carry in | a_in + b_in + c_in | 1, se houver carry no bit mais significativo | 1, se ocorreu overflow |
| 0010 | SUB | Subtração sem carry in | a_in - b_in | 1, se houver borrow no bit mais significativo | 1, se ocorreu overflow |
| 0011 | SUBC | Subtração com carry in: | a_in - b_in – c_in | 1, se houver borrow no bit mais significativo | 1, se ocorreu overflow |
| 0100 | AND | AND lógico bit a bit | a_in AND b_in | 0 | não interessa |
| 0101 | OR | OR lógico bit a bit | a_in OR b_in | 0 | não interessa |
| 0110 | XOR | XOR lógico bit a bit | a_in XOR b_in | 0 | não interessa |
| 0111 | NOT | Complemento (inverte todos os bits) | NOT a_in | 0 | não interessa |
| 1000 | RL | Rotação para esquerda | a_in [6...0] concatenado com a_in [7] | a_in [7] | não interessa |

| | | | | | |
|---|---|---|---|---|---|
| 1001 | RR | Rotação para direita | a_in [0] concatenado com a_in [7...1] | a_in [0] | não interessa |
| 1010 | RLC | Rotação para esquerda através do carry | a [6...0] concatenado com c_in | a_in [7] | não interessa |
| 1011 | RRC | Rotação para direita através do carry | c_in concatenado com a_in [7...1] | a_in [0] | não interessa |
| 1100 | SLL | Deslocamento lógico para esquerda | a_in [6...0] concatenado com 0 | a_in [7] | não interessa |
| 1101 | SRL | Deslocamento lógico para direita | 0 concatenado com a_in [7...1] | a_in [0] | não interessa |
| 1110 | SRA | Deslocamento aritmético para direita | a_in [7] concatenado com a_in [7...1] | a_in [0] | não interessa |
| 1111 | PASS_B | By-pass B | b_in | 0 | não interessa |

Fonte: Autoria própria (2025).

Desse modo, com a definição das operações e das variáveis de entrada e de saída, foi possível desenvolver a ULA. Para isso, foi utilizado o software Quartus na versão 9.1sp2, o qual fornece suporte para linguagem VHDL.

### *2.1.1 Desenvolvimento da ULA em VHDL*

Nesta seção, será apresentada e explicada a implementação da Unidade Lógica e Aritmética (ULA) utilizando a linguagem VHDL. O código desenvolvido descreve o funcionamento interno da ULA, detalhando como são realizadas as operações aritméticas e lógicas sobre dois operandos de 8 bits, além do controle de sinais como o carry/borrow, o overflow e o zero. Cada parte do código será analisada a fim de evidenciar seu papel na execução das instruções definidas.

Inicialmente, foram definidas as bibliotecas que auxiliaram na execução do código e nossa entidade ALU. Foram incluídas para esse projeto duas bibliotecas padrões do IEEE, para manipulações de sinais lógicos e vetores. Já na entidade, foi definido os sinais de entrada e de saída, com base nos Quadros 1 e 2. Na Figura 1 exibe o trecho inicial do programa.

Figura 1 - Bibliotecas utilizadas e entidade definida no programa.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY ALU IS
    PORT (
        a_in, b_in: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        c_in: IN STD_LOGIC;
        op: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        r: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        c_out, z, v: OUT STD_LOGIC
    );
END ENTITY;
```

Fonte: Autoria própria (2025).

Posteriormente, foi definido a arquitetura e a implementação. Na arquitetura foi estabelecido um sinal auxiliar “whole” com 9 bits que é utilizada para armazenar o resultado completo de operações aritméticas, permitindo a captura de carry-out quando necessário.

Na implementação, primeiramente, foi definido a operação que será realizada com base no valor de “op” usando a estrutura de WITH...SELECT. Em cada caso dessa construção, concatena-se um “0” no início de “a_in” e “b_in” para formar vetores de 9 bits, permitindo detectar overflow/carry para as operações aritméticas. A geração de saídas foi desenvolvida conforme as informações no Quadro 2. Na Figura 2 é exibido a arquitetura e implementação do sistema.

Figura 2 - Arquitetura e implementação da ULA.

```
ARCHITECTURE arch OF ALU IS
    SIGNAL whole: STD_LOGIC_VECTOR(8 DOWNTO 0);
BEGIN
    WITH op SELECT
        whole <= ('0' & a_in) + ('0' & b_in) WHEN "0000", -- ADD
                 ('0' & a_in) + ('0' & b_in) + ("00000000" & c_in) WHEN "0001", -- ADDC
                 ('0' & a_in) - ('0' & b_in) WHEN "0010", -- SUB
                 ('0' & a_in) - ('0' & b_in) - ("00000000" & c_in) WHEN "0011", -- SUBC
                 '0' & (a_in AND b_in) WHEN "0100", -- AND
                 '0' & (a_in OR b_in) WHEN "0101", -- OR
                 '0' & (a_in XOR b_in) WHEN "0110", -- XOR
                 '0' & NOT a_in WHEN "0111", -- NOT
                 a_in(7) & a_in(6 DOWNTO 0) & a_in(7) WHEN "1000", -- RL
                 a_in(0) & a_in(0) & a_in(7 DOWNTO 1) WHEN "1001", -- RR
                 a_in(7) & a_in(6 DOWNTO 0) & c_in WHEN "1010", -- RLC
                 a_in(0) & c_in & a_in(7 DOWNTO 1) WHEN "1011", -- RRC
                 a_in(7) & a_in(6 DOWNTO 0) & '0' WHEN "1100", -- SLL
                 a_in(0) & '0' & a_in(7 DOWNTO 1) WHEN "1101", -- SRL
                 a_in(0) & a_in(7) & a_in(7 DOWNTO 1) WHEN "1110", -- SRA
                 '0' & b_in WHEN "1111", -- PASS_B
                 (others => '0') WHEN OTHERS;

    r <= whole(7 DOWNTO 0);
    c_out <= whole(8);
    z <= '1' WHEN whole(7 DOWNTO 0) = "00000000" ELSE '0';
```

Fonte: Autoria própria (2025).

Em seguida, foi definido o cálculo da flag de overflow, que nesse caso é calculada apenas para casos de operações aritméticas (ADD, ADDC, SUB, SUBC), baseando-se nos sinais dos operandos e do resultado. A lógica dessa parte do código é baseada em detectar as seguintes situações: soma de dois positivos resultando em um negativo, soma de dois negativos resultando em um positivo, um positivo menos um negativo resultando em um negativo, e um negativo menos um positivo resultando em um positivo. Na Figura 3 mostra a implementação do cálculo de overflow do programa.

Figura 3 - Cálculo de overflow no programa.

```
    WITH op SELECT
        v <= (a_in(7) AND b_in(7) AND NOT whole(7)) OR
             (NOT a_in(7) AND NOT b_in(7) AND whole(7)) WHEN "0000", -- ADD
             (a_in(7) AND b_in(7) AND NOT whole(7)) OR
             (NOT a_in(7) AND NOT b_in(7) AND whole(7)) WHEN "0001", -- ADDC
             (NOT a_in(7) AND b_in(7) AND whole(7)) OR
             (a_in(7) AND NOT b_in(7) AND NOT whole(7)) WHEN "0010", -- SUB
             (NOT a_in(7) AND b_in(7) AND whole(7)) OR
             (a_in(7) AND NOT b_in(7) AND NOT whole(7)) WHEN "0011", -- SUBC
             '0' WHEN OTHERS;

END arch;
```

Fonte: Autoria própria (2025).

Na Figura 4, é exibido a versão final do código, incluindo todas as partes explicadas anteriormente.

Figura 4 - Código fonte da ULA com a linguagem VHDL.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY ALU IS
    PORT (
        a_in, b_in: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        c_in: IN STD_LOGIC;
        op: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        r: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        c_out, z, v: OUT STD_LOGIC
    );
END ENTITY;

ARCHITECTURE arch OF ALU IS
    SIGNAL whole: STD_LOGIC_VECTOR(8 DOWNTO 0);
BEGIN
    WITH op SELECT
        whole <= ('0' & a_in) + ('0' & b_in) WHEN "0000", -- ADD
                 ('0' & a_in) + ('0' & b_in) + ("00000000" & c_in) WHEN "0001", -- ADDC
                 ('0' & a_in) - ('0' & b_in) WHEN "0010", -- SUB
                 ('0' & a_in) - ('0' & b_in) - ("00000000" & c_in) WHEN "0011", -- SUBC
                 '0' & (a_in AND b_in) WHEN "0100", -- AND
                 '0' & (a_in OR b_in) WHEN "0101", -- OR
                 '0' & (a_in XOR b_in) WHEN "0110", -- XOR
                 '0' & NOT a_in WHEN "0111", -- NOT
                 a_in(7) & a_in(6 DOWNTO 0) & a_in(7) WHEN "1000", -- RL
                 a_in(0) & a_in(0) & a_in(7 DOWNTO 1) WHEN "1001", -- RR
                 a_in(7) & a_in(6 DOWNTO 0) & c_in WHEN "1010", -- RLC
                 a_in(0) & c_in & a_in(7 DOWNTO 1) WHEN "1011", -- RRC
                 a_in(7) & a_in(6 DOWNTO 0) & '0' WHEN "1100", -- SLL
                 a_in(0) & '0' & a_in(7 DOWNTO 1) WHEN "1101", -- SRL
                 a_in(0) & a_in(7) & a_in(7 DOWNTO 1) WHEN "1110", -- SRA
                 '0' & b_in WHEN "1111", -- PASS_B
                 (others => '0') WHEN OTHERS;

    r <= whole(7 DOWNTO 0);
    c_out <= whole(8);
    z <= '1' WHEN whole(7 DOWNTO 0) = "00000000" ELSE '0';

    WITH op SELECT
        v <= (a_in(7) AND b_in(7) AND NOT whole(7)) OR
             (NOT a_in(7) AND NOT b_in(7) AND whole(7)) WHEN "0000", -- ADD
             (a_in(7) AND b_in(7) AND NOT whole(7)) OR
             (NOT a_in(7) AND NOT b_in(7) AND whole(7)) WHEN "0001", -- ADDC
             (NOT a_in(7) AND b_in(7) AND whole(7)) OR
             (a_in(7) AND NOT b_in(7) AND NOT whole(7)) WHEN "0010", -- SUB
             (NOT a_in(7) AND b_in(7) AND whole(7)) OR
             (a_in(7) AND NOT b_in(7) AND NOT whole(7)) WHEN "0011", -- SUBC
             '0' WHEN OTHERS;

END arch;
```

Fonte: Autoria própria (2025).

Vale ressaltar que o programa foi desenvolvido utilizando exclusivamente código corrente, sem o auxílio de latches e flip-flops. Além disso, todas as entradas e saídas foram do tipo STD_LOGIC ou STD_LOGIC_VECTOR.

## 2.2 Testes, Resultados e Análises

Após o desenvolvimento da ULA na linguagem VHDL, foi realizada uma bateria de testes. Estes procuram estipular valores de entrada e verificar se os valores de saída condizem com o esperado de acordo com a operação, sendo que a cada 10ns

um novo conjunto de valores de entradas é posto para que seja possível se testar todas as saídas possíveis.

Nesse sentido, primeiramente, foram realizados os casos de teste da operação ADD, ou seja, com a variável de entrada “op” sendo igual a 0000. Assim, foram produzidos os casos de testes, os quais são descritos no Quadro 4.

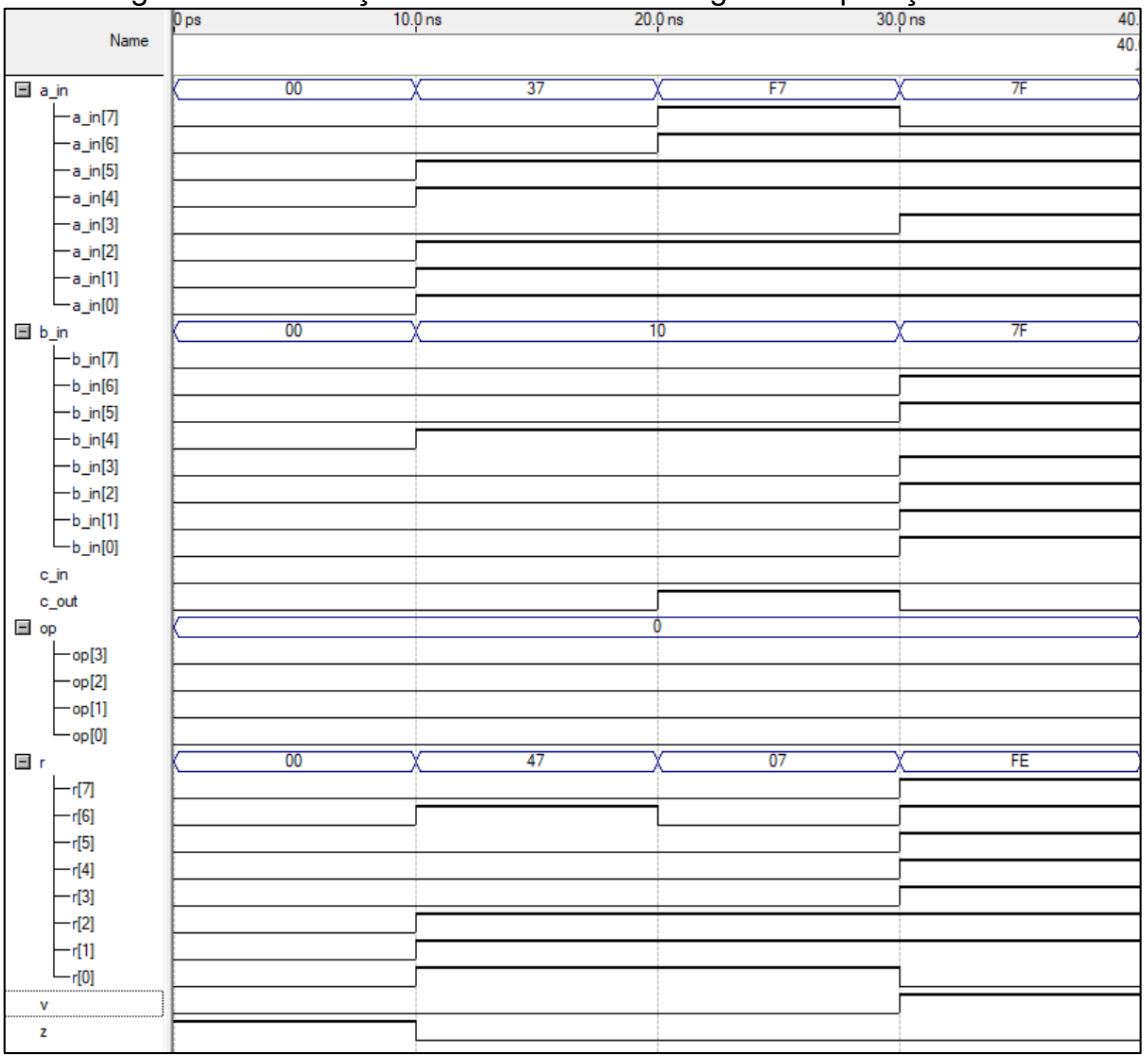
Quadro 4 - Casos de teste para ADD.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **0ps – 10ns** | 0x00 | 0x00 | DC | 0x00 | 0 | 1 | 0 |
| **10ns - 20ns** | 0x37 | 0x10 | DC | 0x47 | 0 | 0 | 0 |
| **20ns - 30ns** | 0xF7 | 0x10 | DC | 0x07 | 1 | 0 | 0 |
| **30ns - 40ns** | 0x7F | 0x7F | DC | 0xFE | 0 | 0 | 1 |

Fonte: Autoria própria (2025).

Sendo os valores de entrada e a de saída “r” na base hexadecimal, DC; “don’t care”, um valor que não influencia na operação e as demais saídas na base binária. Nesse sentindo, essas observações serão validas para os próximos casos de teste que serão apresentados nesse trabalho. Assim, foi realizada uma simulação de forma de onda digital com as entradas de dados estabelecidas no Quadro 4 e suas respectivas faixas de tempo. A Figura 5 mostra as formas de ondas obtidas.

Figura 5 – Simulação em forma de onda digital da operação ADD.




Fonte: Autoria própria (2025).

Sendo as ondas em nível baixo equivalendo ao bit 0 e as ondas em nível alto equivalendo ao bit 1. Foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 1.

Tabela 1 - Resultados do teste da operação ADD.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 0ps – 10ns | 0x00 | 0 | 1 | 0 |
| 10ns - 20ns | 0x47 | 0 | 0 | 0 |
| 20ns - 30ns | 0x07 | 1 | 0 | 0 |
| 30ns - 40ns | 0xFE | 0 | 0 | 1 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 1, e os valores esperados no cenário de teste, representado no Quadro 4, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da operação

ADD da ULA está correta. Posteriormente, foram elaborados os testes para operação ADDC, logo, com a variável “op” equivalendo a 0001. Os casos de teste estão listados no Quadro 5.

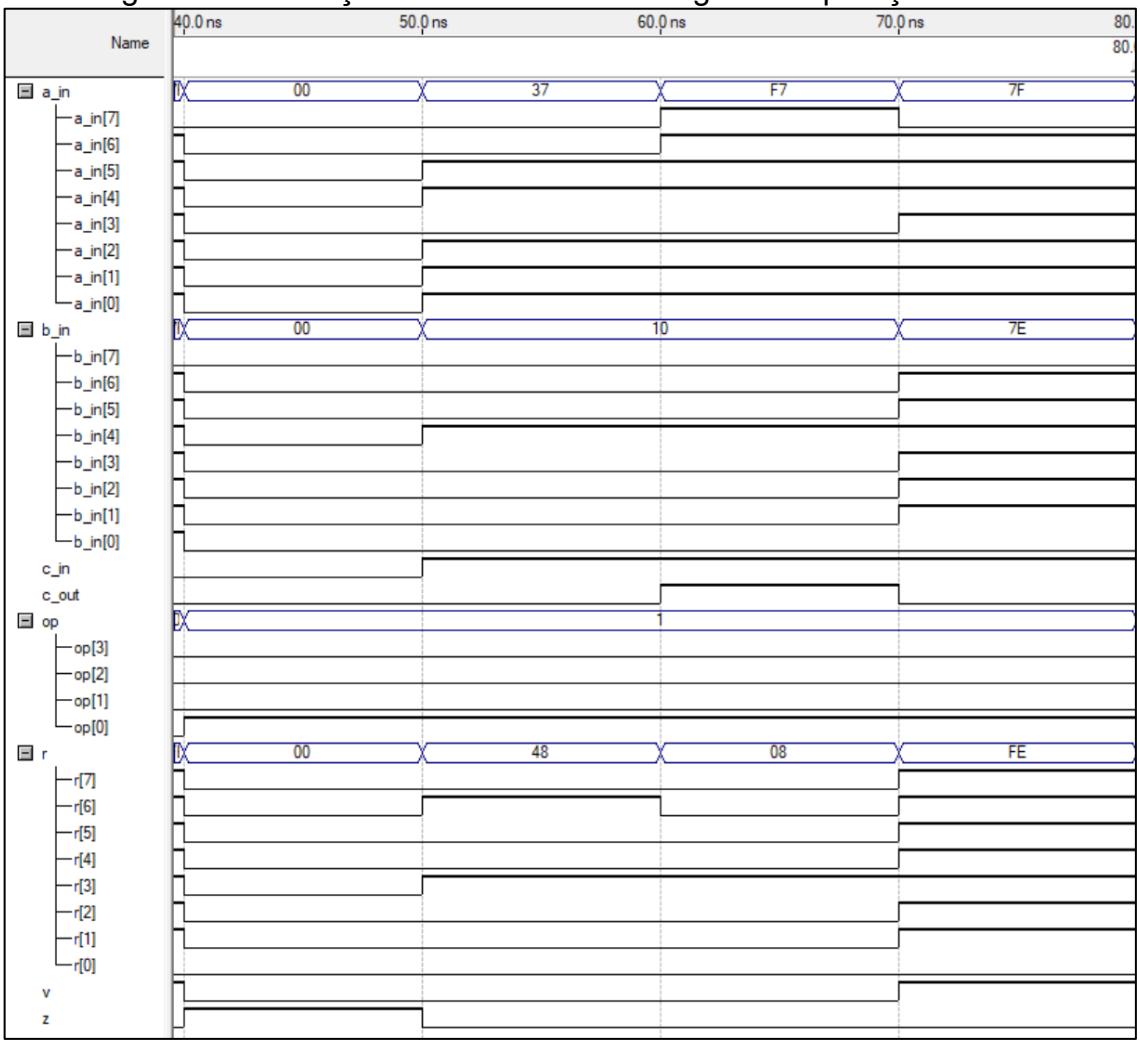
Quadro 5 - Casos de teste para ADDC.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **40ns – 50ns** | 0x00 | 0x00 | 0 | 0x00 | 0 | 1 | 0 |
| **50ns - 60ns** | 0x37 | 0x10 | 1 | 0x48 | 0 | 0 | 0 |
| **60ns - 70ns** | 0xF7 | 0x10 | 1 | 0x08 | 1 | 0 | 0 |
| **70ns - 80ns** | 0x7F | 0x7E | 1 | 0xFE | 0 | 0 | 1 |

Fonte: Autoria própria (2025).

Com “c_in” na base binária. A Figura 6 mostra os resultados obtidos com os cenários de testes descritos no Quadro 5.

Figura 6 - Simulação em forma de onda digital da operação ADDC.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 2.

Tabela 2 - Resultados do teste da operação ADDC.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 40ns – 50ns | 0x00 | 0 | 1 | 0 |
| 50ns - 60ns | 0x48 | 0 | 0 | 0 |
| 60ns - 70ns | 0x08 | 1 | 0 | 0 |
| 70ns - 80ns | 0xFE | 0 | 0 | 1 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 2, e os valores esperados no cenário de teste, representado no Quadro 5, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da operação

ADDC da ULA está correta. Para os casos de teste da operação SUB, variável “op” equivalendo a 0010, estão descritos no Quadro 6.

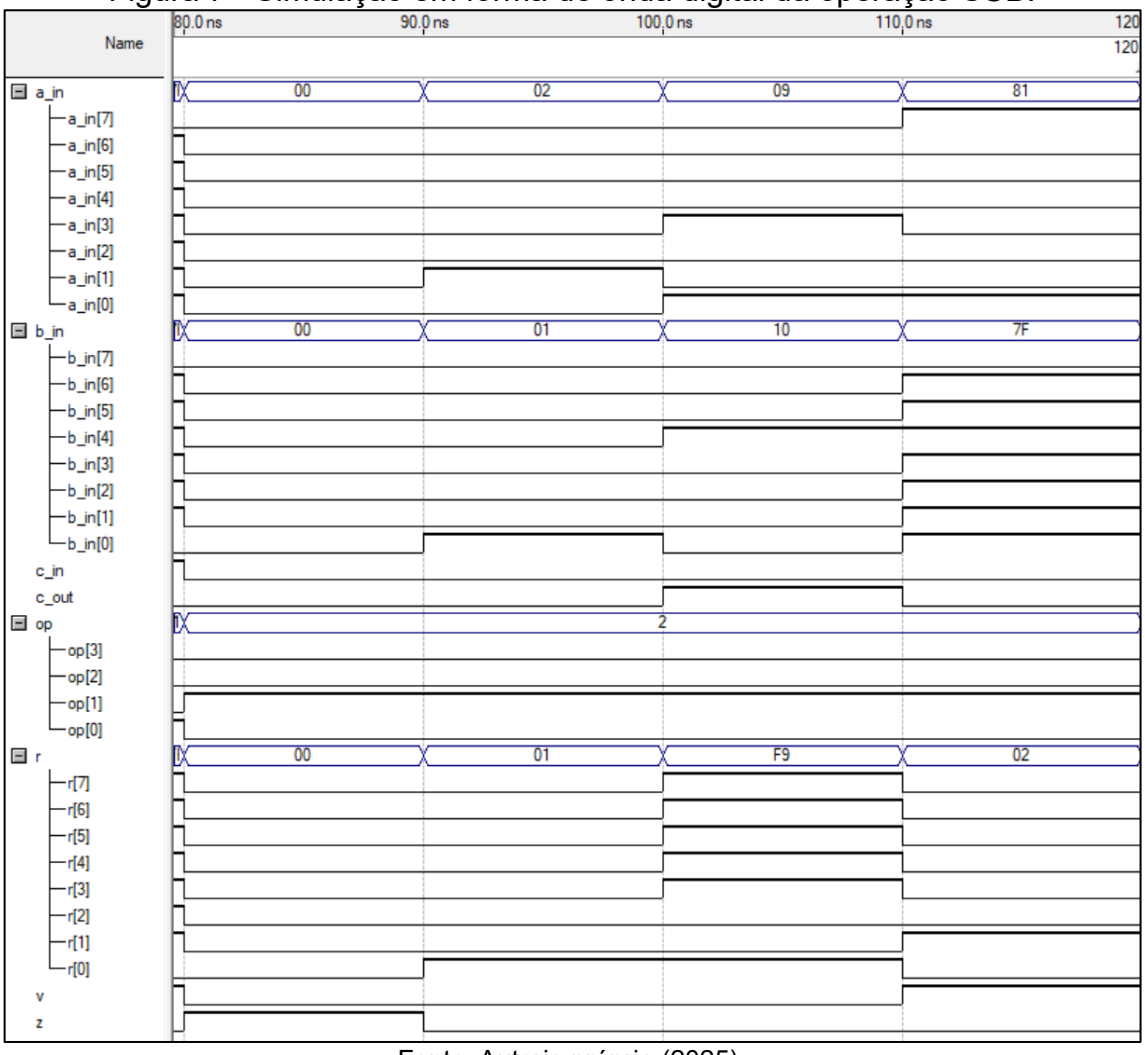
Quadro 6 - Casos de teste para SUB.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **80ns – 90ns** | 0x00 | 0x00 | DC | 0x00 | 0 | 1 | 0 |
| **90ns - 100ns** | 0x02 | 0x01 | DC | 0x01 | 0 | 0 | 0 |
| **100ns – 110ns** | 0x09 | 0x10 | DC | 0xF9 | 1 | 0 | 0 |
| **110ns – 120ns** | 0x81 | 0x7F | DC | 0x02 | 0 | 0 | 1 |

Fonte: Autoria própria (2025).

A Figura 7 mostra os resultados obtidos com os cenários de testes descritos no Quadro 6.

Figura 7 - Simulação em forma de onda digital da operação SUB.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 3.

Tabela 3 - Resultados do teste da operação SUB.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 80ns – 90ns | 0x00 | 0 | 1 | 0 |
| 90ns - 100ns | 0x01 | 0 | 0 | 0 |
| 100ns – 110ns | 0xF9 | 1 | 0 | 0 |
| 110ns – 120ns | 0x02 | 0 | 0 | 1 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 3, e os valores esperados no cenário de teste, representado no Quadro 6, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da operação SUB da ULA está correta. Para os casos de teste da operação SUBC, variável “op” equivalendo a 0011, estão descritos no Quadro 7.

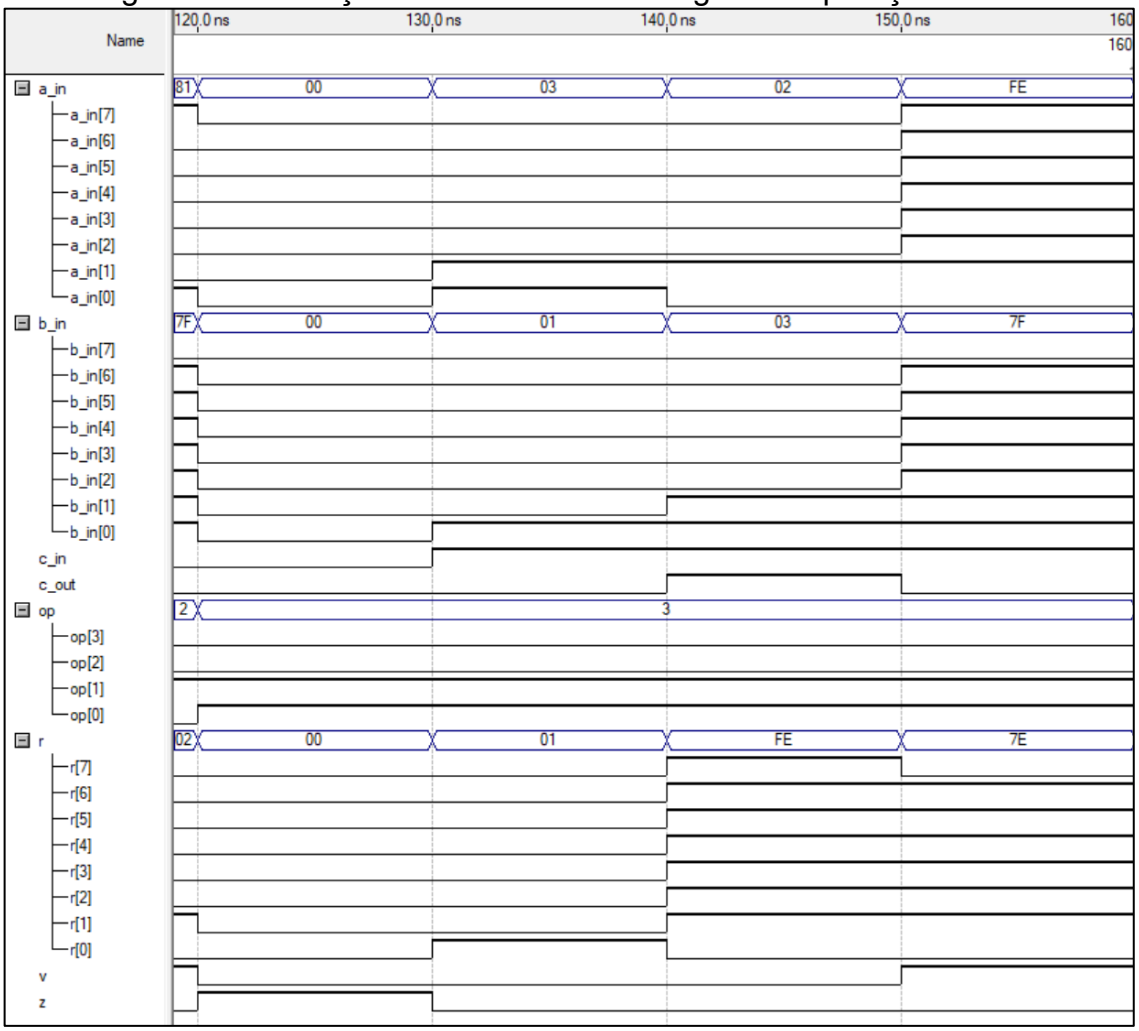
Quadro 7 - Casos de teste para SUBC.

| Tempo | Variáveis de Entrada | | | Resultados esperados nas variáveis de saída | | | |
|---|---|---|---|---|---|---|---|
| | a_in | b_in | c_in | r | c_out | z | v |
| **120ns – 130ns** | 0x00 | 0x00 | 0 | 0x00 | 0 | 1 | 0 |
| **130ns - 140ns** | 0x03 | 0x01 | 1 | 0x01 | 0 | 0 | 0 |
| **140ns - 150ns** | 0x02 | 0x03 | 1 | 0xFE | 1 | 0 | 0 |
| **150ns - 160ns** | 0xFE | 0x7F | 1 | 0x7E | 0 | 0 | 1 |

Fonte: Autoria própria (2025).

A Figura 8 mostra os resultados obtidos com os cenários de testes descritos no Quadro 7.

Figura 8 - Simulação em forma de onda digital da operação SUBC.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 4.

Tabela 4 - Resultados do teste da operação SUBC.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 120ns – 130ns | 0x00 | 0 | 1 | 0 |
| 130ns - 140ns | 0x01 | 0 | 0 | 0 |
| 140ns - 150ns | 0xFE | 1 | 0 | 0 |
| 150ns - 160ns | 0x7E | 0 | 0 | 1 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 4, e os valores esperados no cenário de teste, representado no Quadro 7, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da operação

SUBC da ULA está correta. Para os casos de teste da operação AND, variável “op” equivalendo a 1000, estão descritos no Quadro 8.

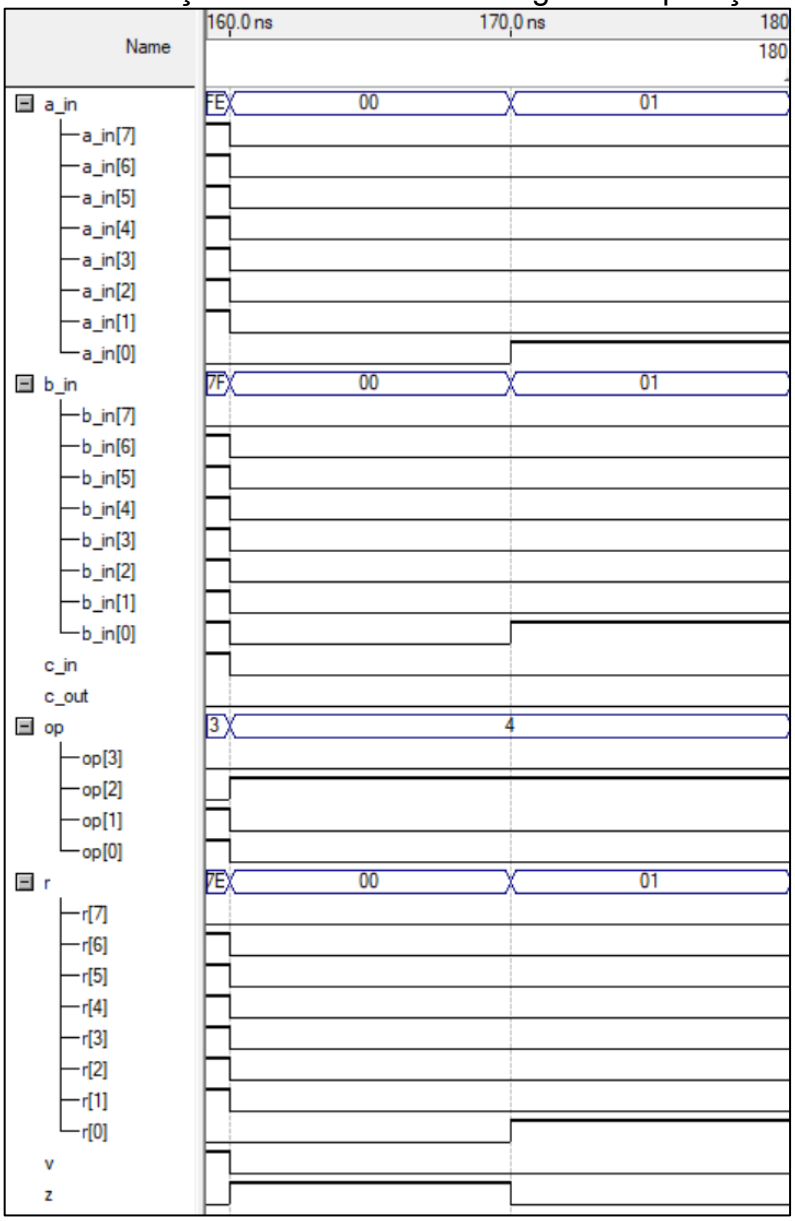
Quadro 8 - Casos de teste para AND.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **160ns – 170ns** | 0x00 | 0x00 | DC | 0x00 | 0 | 1 | 0 |
| **170ns - 180ns** | 0x01 | 0x01 | DC | 0x01 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

A Figura 9 mostra os resultados obtidos com os cenários de testes descritos no Quadro 8.

Figura 9 - Simulação em forma de onda digital da operação AND.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 5.

Tabela 5 - Resultados do teste da operação AND.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 160ns – 170ns | 0x00 | 0 | 1 | 0 |
| 170ns - 180ns | 0x01 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 5, e os valores esperados no cenário de teste, representado no Quadro 8, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da operação

AND da ULA está correta. Para os casos de teste da operação OR, variável “op” equivalendo a 0101, estão descritos no Quadro 9.

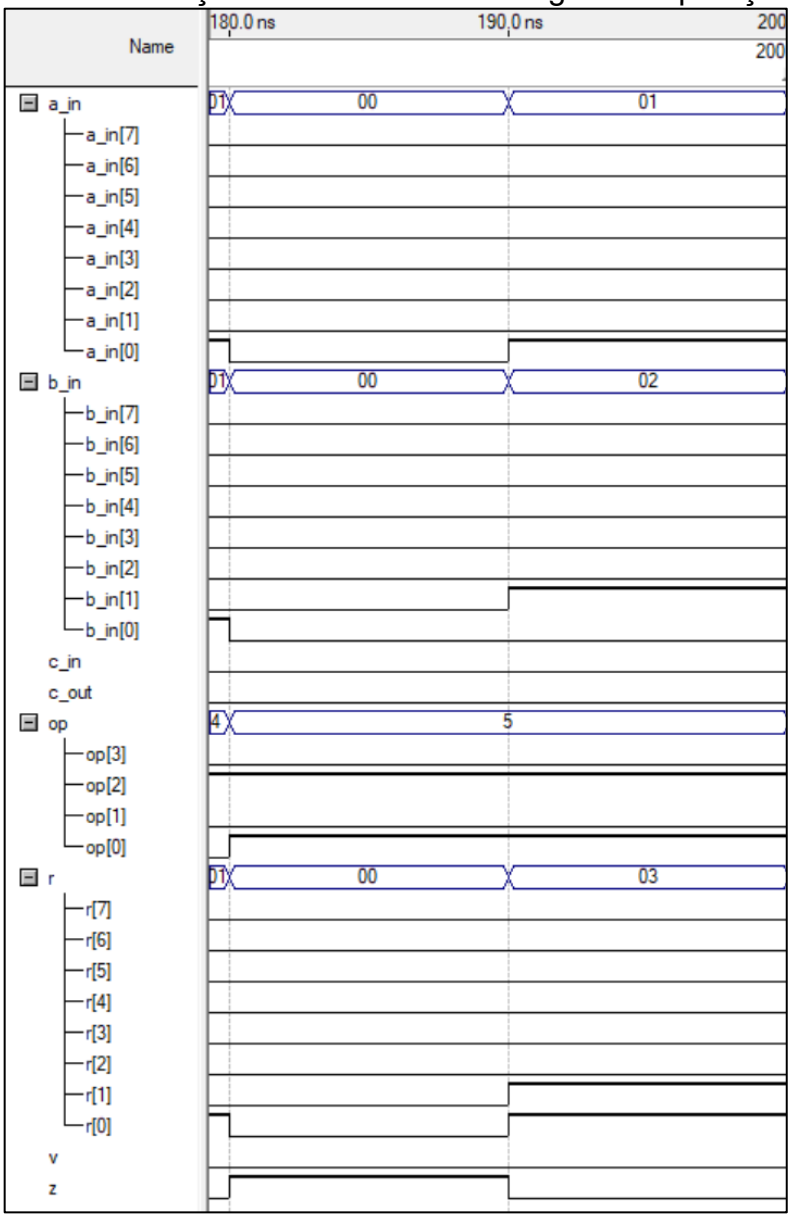
Quadro 9 - Casos de teste para OR.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **180ns – 190ns** | 0x00 | 0x00 | DC | 0x00 | 0 | 1 | 0 |
| **190ns - 200ns** | 0x01 | 0x02 | DC | 0x03 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

A Figura 10 mostra os resultados obtidos com os cenários de testes descritos no Quadro 9.

Figura 10 - Simulação em forma de onda digital da operação OR.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 6.

Tabela 6 - Resultados do teste da operação OR.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 180ns – 190ns | 0x00 | 0 | 1 | 0 |
| 190ns - 200ns | 0x03 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 6, e os valores esperados no cenário de teste, representado no Quadro 9, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da operação

OR da ULA está correta. Para os casos de teste da operação XOR, variável “op” equivalendo a 0110, estão descritos no Quadro 10.

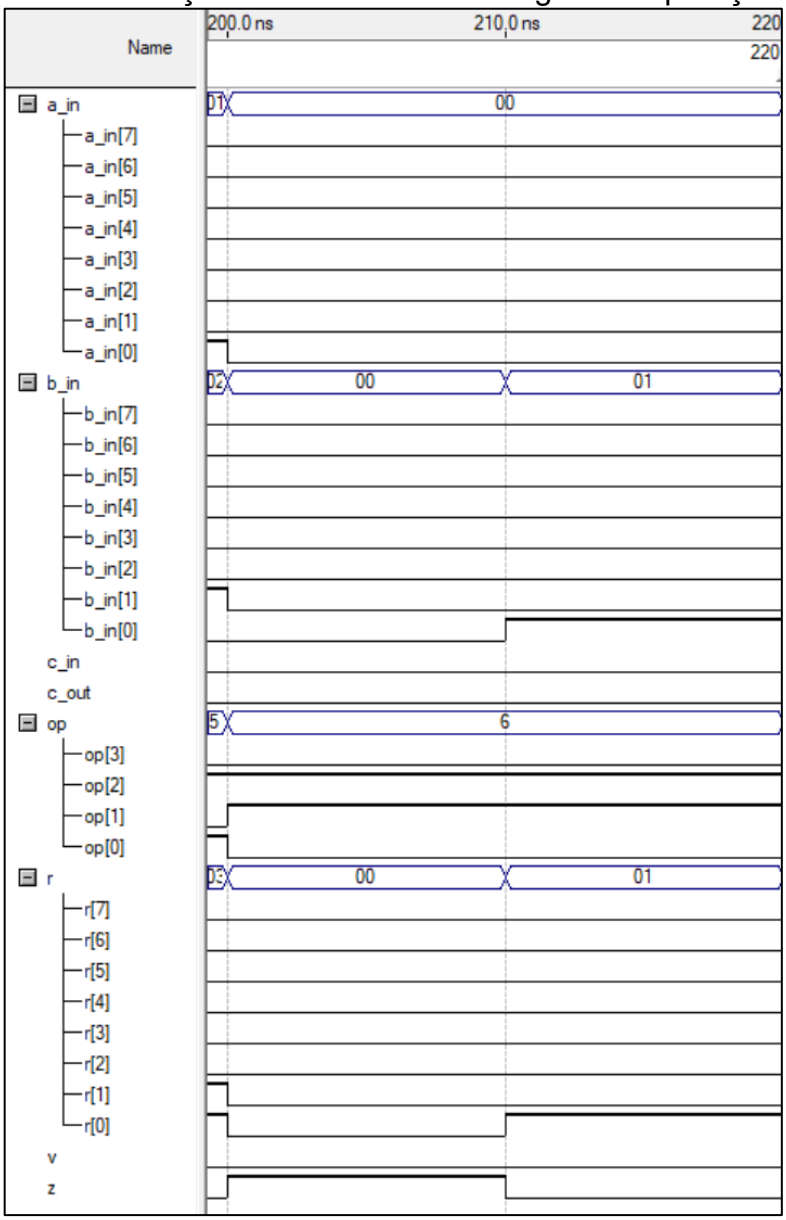
Quadro 10 - Casos de teste para XOR.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **200ns – 210ns** | 0x00 | 0x00 | DC | 0x00 | 0 | 1 | 0 |
| **210ns - 220ns** | 0x00 | 0x01 | DC | 0x01 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

A Figura 11 mostra os resultados obtidos com os cenários de testes descritos no Quadro 10.

Figura 11 - Simulação em forma de onda digital da operação XOR.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 7.

Tabela 7 - Resultados do teste da operação XOR.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 200ns – 210ns | 0x00 | 0 | 1 | 0 |
| 210ns - 220ns | 0x01 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 7, e os valores esperados no cenário de teste, representado no Quadro 10, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da operação

XOR da ULA está correta. Para os casos de teste da operação NOT, variável “op” equivalendo a 0111, estão descritos no Quadro 11.

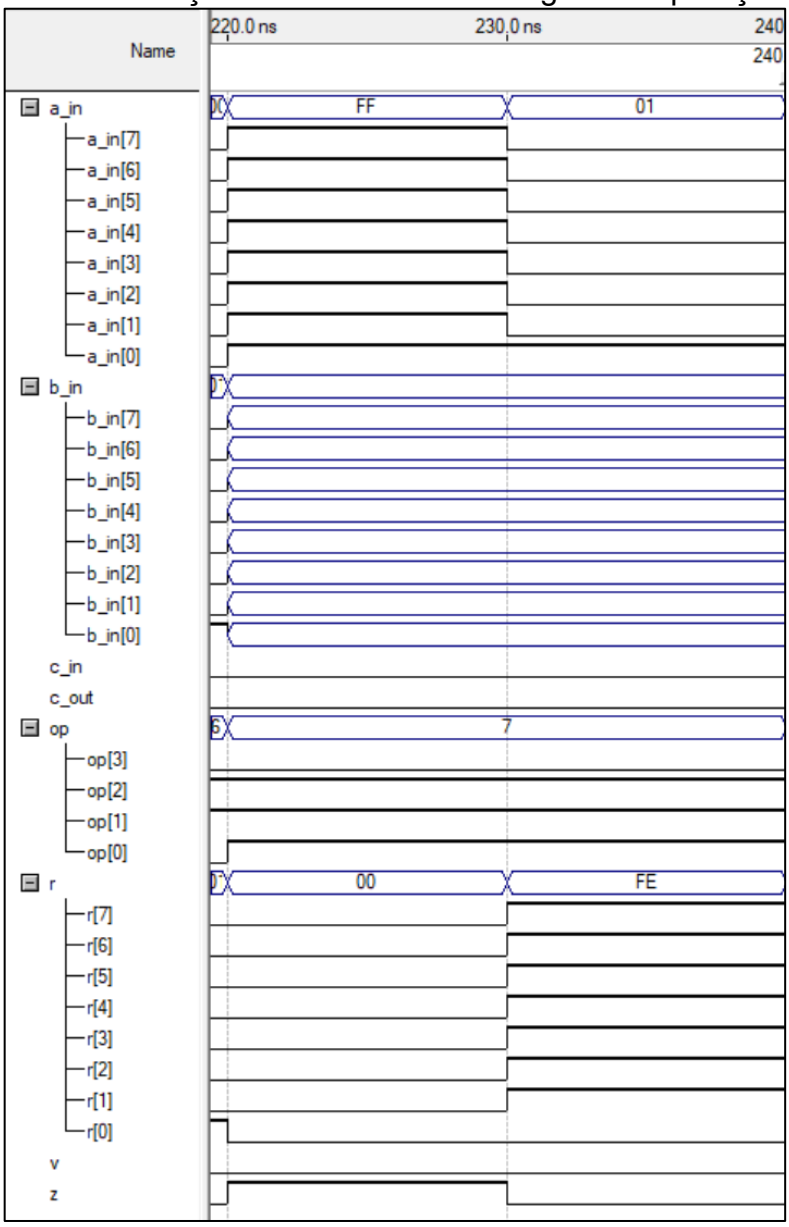
Quadro 11 - Casos de teste para NOT.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **220ns – 230ns** | 0x00 | DC | DC | 0x00 | 0 | 1 | 0 |
| **230ns - 240ns** | 0x01 | DC | DC | 0x03 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

A Figura 12 mostra os resultados obtidos com os cenários de testes descritos no Quadro 11.

Figura 12 - Simulação em forma de onda digital da operação NOT.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 8.

Tabela 8 - Resultados do teste da operação NOT.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 220ns – 230ns | 0x00 | 0 | 1 | 0 |
| 230ns - 240ns | 0x03 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 8, e os valores esperados no cenário de teste, representado no Quadro 11, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da operação

NOT da ULA está correta. Para os casos de teste da operação RL, variável “op” equivalendo a 1000, estão descritos no Quadro 12.

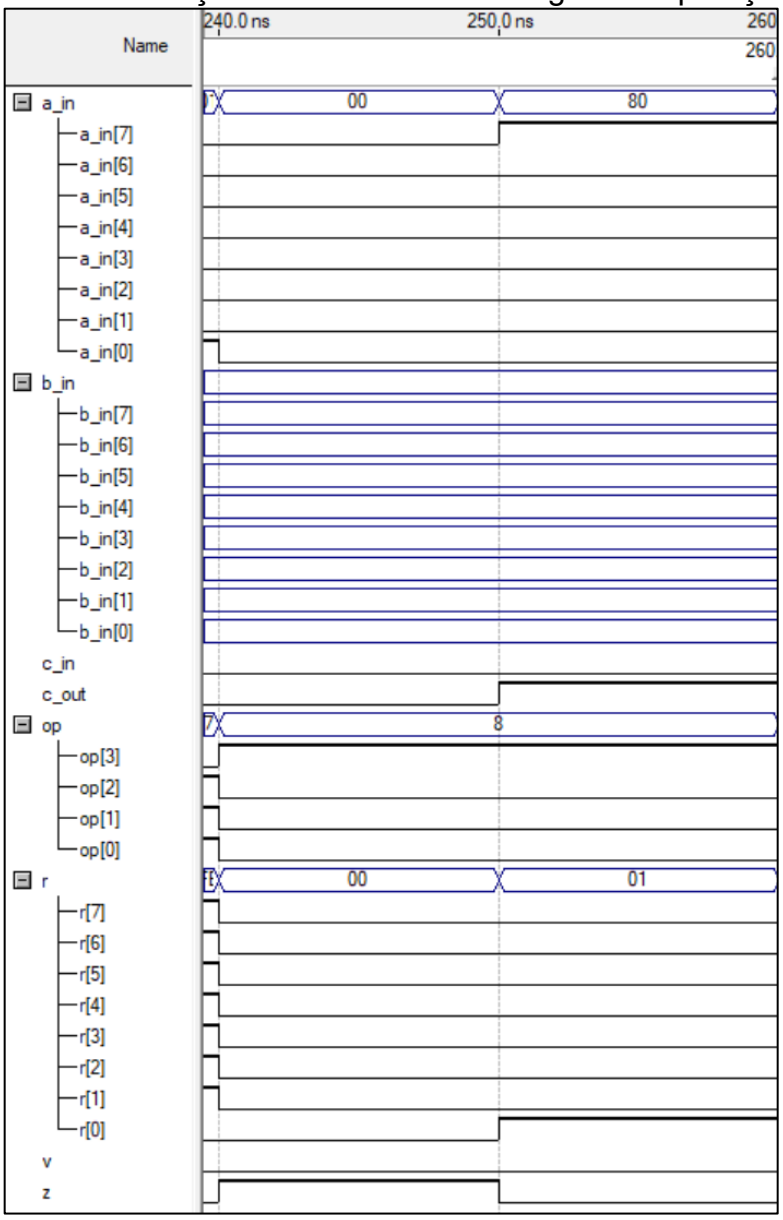
Quadro 12 - Casos de teste para RL.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **240ns – 250ns** | 0x00 | DC | DC | 0x00 | 0 | 1 | 0 |
| **250ns – 260ns** | 0x80 | DC | DC | 0x01 | 1 | 0 | 0 |

Fonte: Autoria própria (2025).

A Figura 13 mostra os resultados obtidos com os cenários de testes descritos no Quadro 12.

Figura 13 - Simulação em forma de onda digital da operação RL.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 9.

Tabela 9 - Resultados do teste da operação RL.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 240ns – 250ns | 0x00 | 0 | 1 | 0 |
| 250ns – 260ns | 0x01 | 1 | 0 | 0 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 9, e os valores esperados no cenário de teste, representado no Quadro 12, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da operação

RL da ULA está correta. Para os casos de teste da operação RR, variável “op” equivalendo a 1001, estão descritos no Quadro 13.

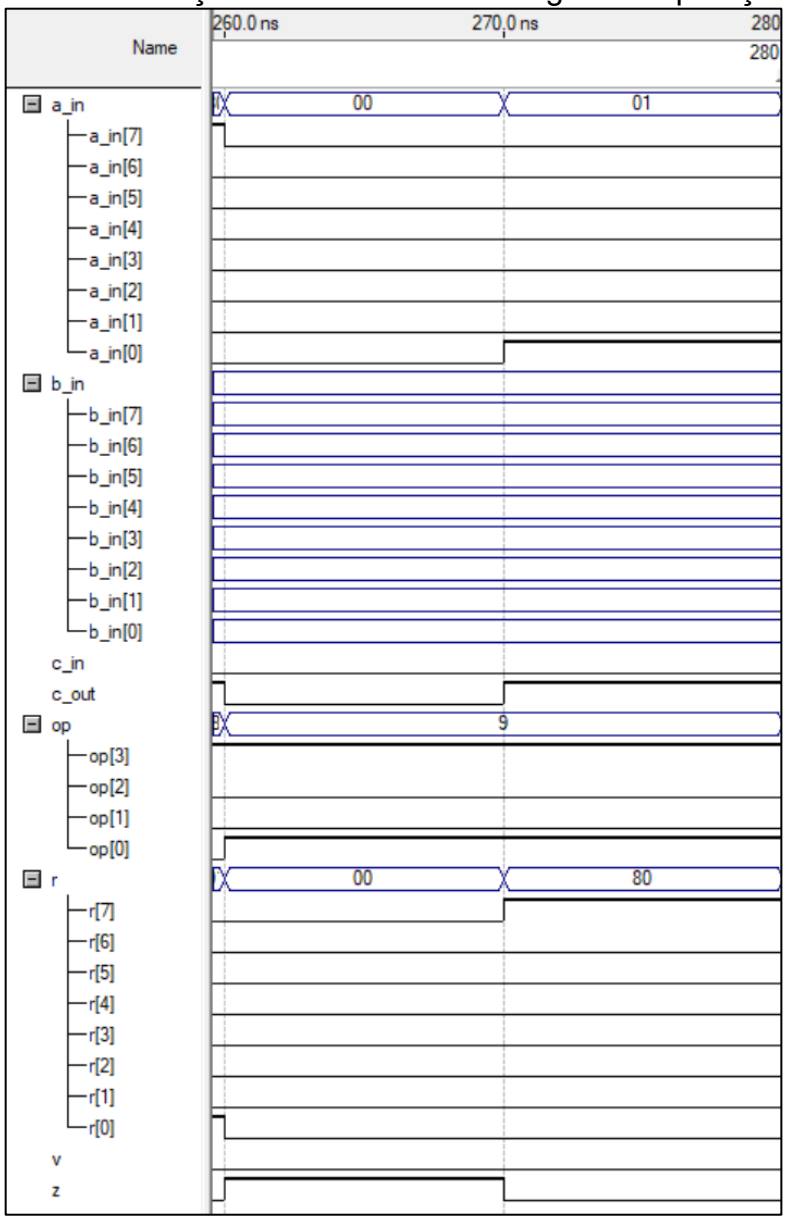
Quadro 13 - Casos de teste para RR.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **260ns – 270ns** | 0x00 | DC | DC | 0x00 | 0 | 1 | 0 |
| **270ns – 280ns** | 0x01 | DC | DC | 0x80 | 1 | 0 | 0 |

Fonte: Autoria própria (2025).

A Figura 14 mostra os resultados obtidos com os cenários de testes descritos no Quadro 13.

Figura 14 - Simulação em forma de onda digital da operação RR.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 10.

Tabela 10 - Resultados do teste da operação RR.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 260ns – 270ns | 0x00 | 0 | 1 | 0 |
| 270ns – 280ns | 0x80 | 1 | 0 | 0 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 10, e os valores esperados no cenário de teste, representado no Quadro 13, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da

operação RR da ULA está correta. Para os casos de teste da operação RLC, variável “op” equivalendo a 1010, estão descritos no Quadro 14.

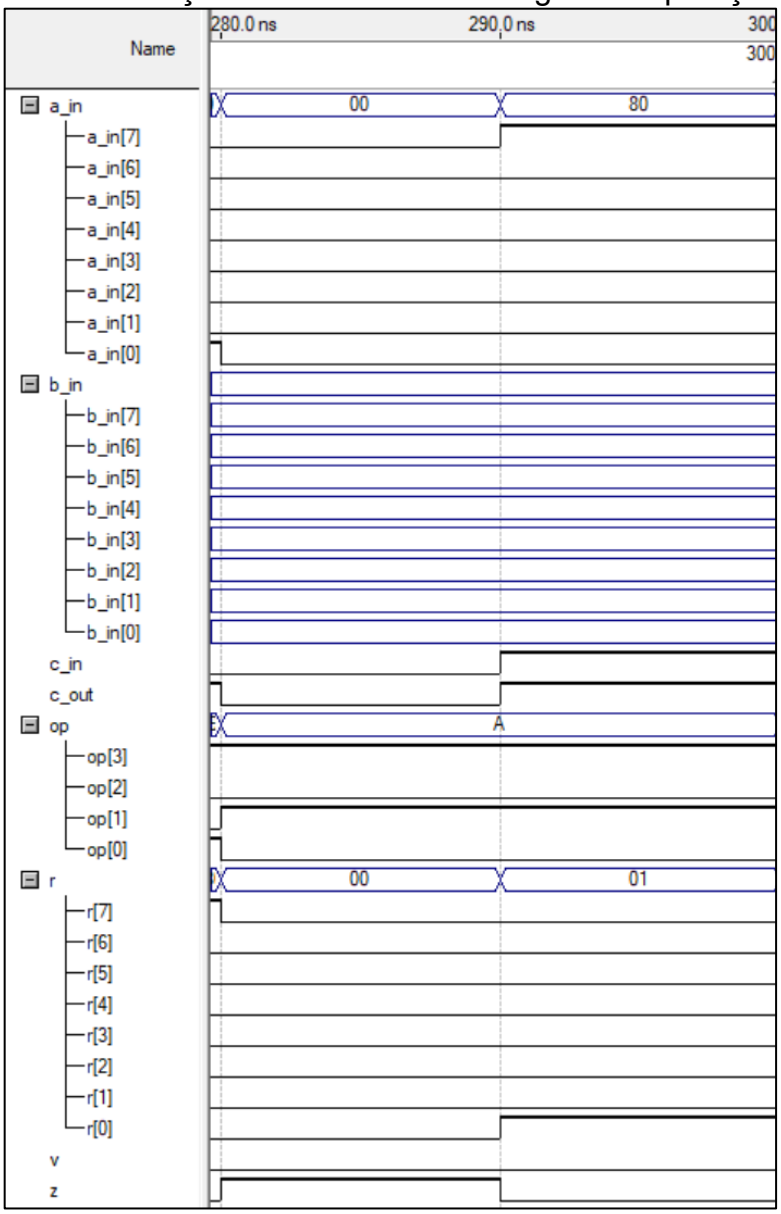
Quadro 14 - Casos de teste para RLC.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **280ns – 290ns** | 0x00 | DC | 0 | 0x00 | 0 | 1 | 0 |
| **290ns – 300ns** | 0x80 | DC | 1 | 0x01 | 1 | 0 | 0 |

Fonte: Autoria própria (2025).

A Figura 15 mostra os resultados obtidos com os cenários de testes descritos no Quadro 14.

Figura 15 - Simulação em forma de onda digital da operação RLC.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 11.

Tabela 11 - Resultados do teste da operação RLC.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 280ns – 290ns | 0x00 | 0 | 1 | 0 |
| 290ns – 300ns | 0x01 | 1 | 0 | 0 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 11, e os valores esperados no cenário de teste, representado no Quadro 14, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da

operação RLC da ULA está correta. Para os casos de teste da operação RRC, variável “op” equivalendo a 1011, estão descritos no Quadro 15.

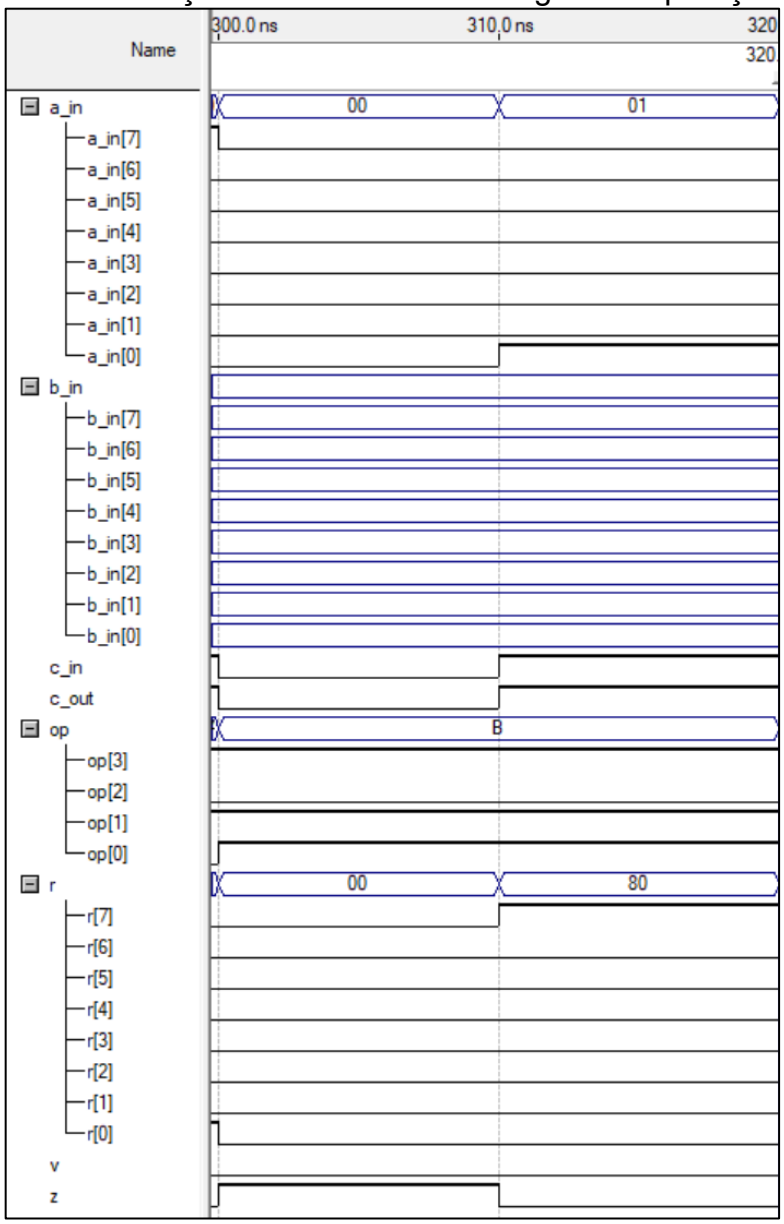
Quadro 15 - Casos de teste para RRC.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **300ns – 310ns** | 0x00 | DC | 0 | 0x00 | 0 | 1 | 0 |
| **310ns – 320ns** | 0x01 | DC | 1 | 0x80 | 1 | 0 | 0 |

Fonte: Autoria própria (2025).

A Figura 16 mostra os resultados obtidos com os cenários de testes descritos no Quadro 15.

Figura 16 - Simulação em forma de onda digital da operação RRC.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 12.

Tabela 12 - Resultados do teste da operação RRC.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 300ns – 310ns | 0x00 | 0 | 1 | 0 |
| 310ns – 320ns | 0x80 | 1 | 0 | 0 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 12, e os valores esperados no cenário de teste, representado no Quadro 15, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da

operação RRC da ULA está correta. Para os casos de teste da operação SLL, variável “op” equivalendo a 1100, estão descritos no Quadro 16.

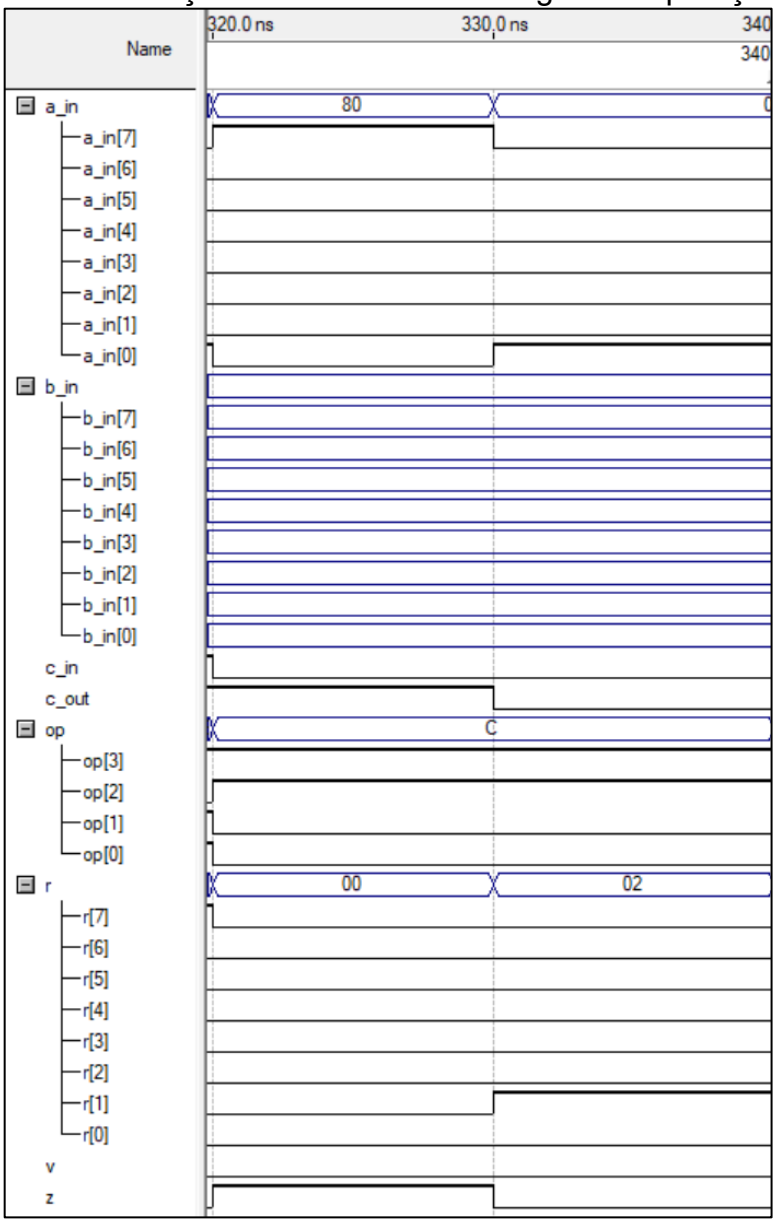
Quadro 16 - Casos de teste para SLL.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **320ns – 330ns** | 0x80 | DC | DC | 0x00 | 1 | 1 | 0 |
| **330ns – 340ns** | 0x01 | DC | DC | 0x02 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

A Figura 17 mostra os resultados obtidos com os cenários de testes descritos no Quadro 16.

Figura 17 - Simulação em forma de onda digital da operação SLL.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 13.

Tabela 13 - Resultados do teste da operação SLL.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 320ns – 330ns | 0x00 | 1 | 1 | 0 |
| 330ns – 340ns | 0x02 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 13, e os valores esperados no cenário de teste, representado no Quadro 16, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da

operação SLL da ULA está correta. Para os casos de teste da operação SRL, variável “op” equivalendo a 1101, estão descritos no Quadro 17.

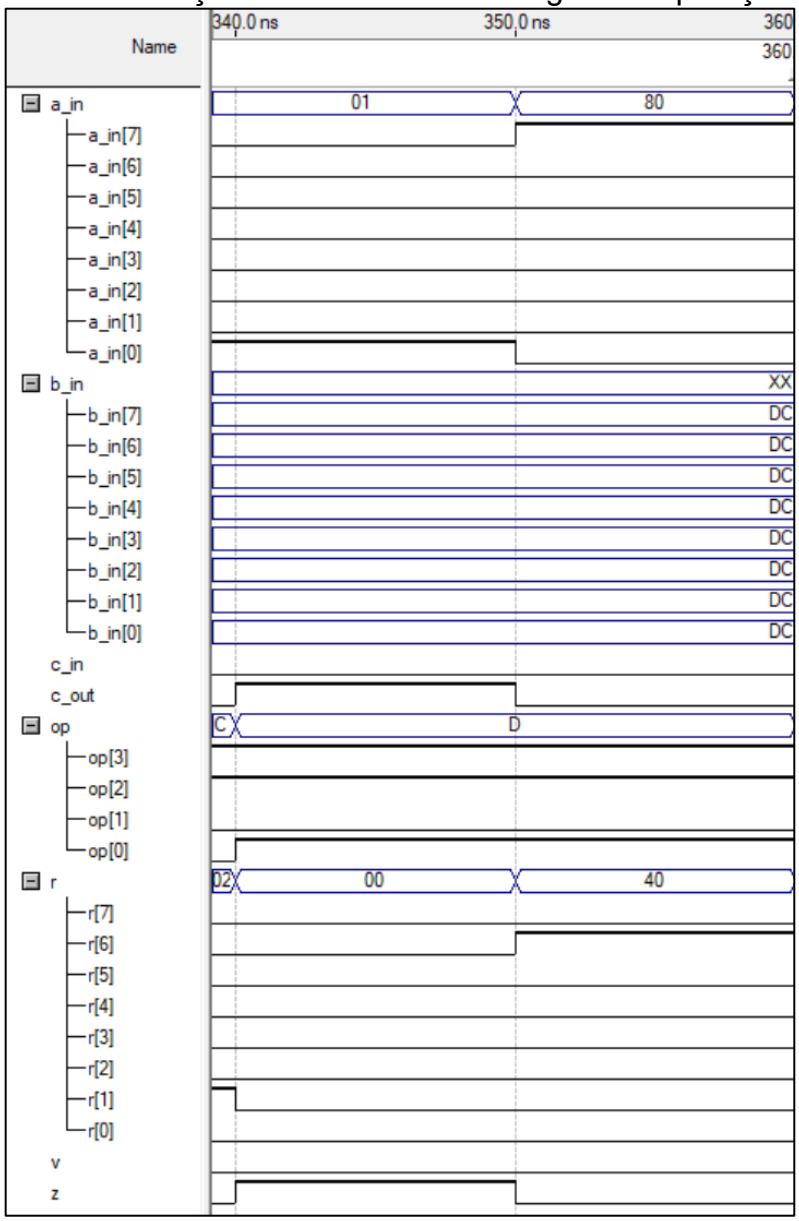
Quadro 17 - Casos de teste para SRL.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **340ns – 350ns** | 0x01 | DC | DC | 0x00 | 1 | 1 | 0 |
| **350ns – 360ns** | 0x80 | DC | DC | 0x40 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

A Figura 18 mostra os resultados obtidos com os cenários de testes descritos no Quadro 17.

Figura 18 - Simulação em forma de onda digital da operação SRL.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 14.

Tabela 14- Resultados do teste da operação SRL.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 340ns – 350ns | 0x00 | 1 | 1 | 0 |
| 350ns – 360ns | 0x40 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 14, e os valores esperados no cenário de teste, representado no Quadro 17, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da

operação SRA da ULA está correta. Para os casos de teste da operação SRA, variável “op” equivalendo a 1110, estão descritos no Quadro 18.

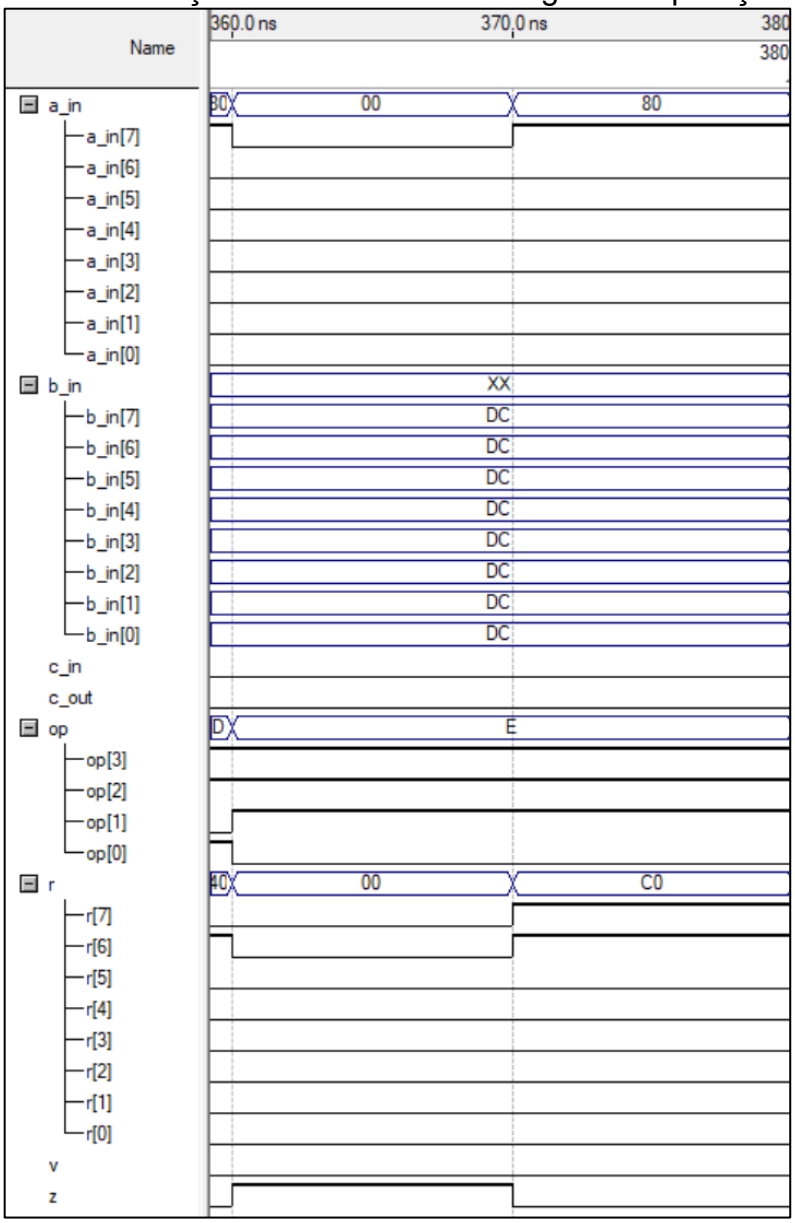
Quadro 18 - Casos de teste para SRA.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **360ns – 370ns** | 0x00 | DC | DC | 0x00 | 0 | 1 | 0 |
| **370ns – 380ns** | 0x80 | DC | DC | 0xC0 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

A Figura 19 mostra os resultados obtidos com os cenários de testes descritos no Quadro 18.

Figura 19 - Simulação em forma de onda digital da operação SRA.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 15.

Tabela 15 - Resultados do teste da operação SRA.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 360ns – 370ns | 0x00 | 0 | 1 | 0 |
| 370ns – 380ns | 0xC0 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 15, e os valores esperados no cenário de teste, representado no Quadro 19, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da

operação SRA da ULA está correta. Por fim, para os casos de teste da operação PASS_B, variável “op” equivalendo a 1111, estão descritos no Quadro 19.

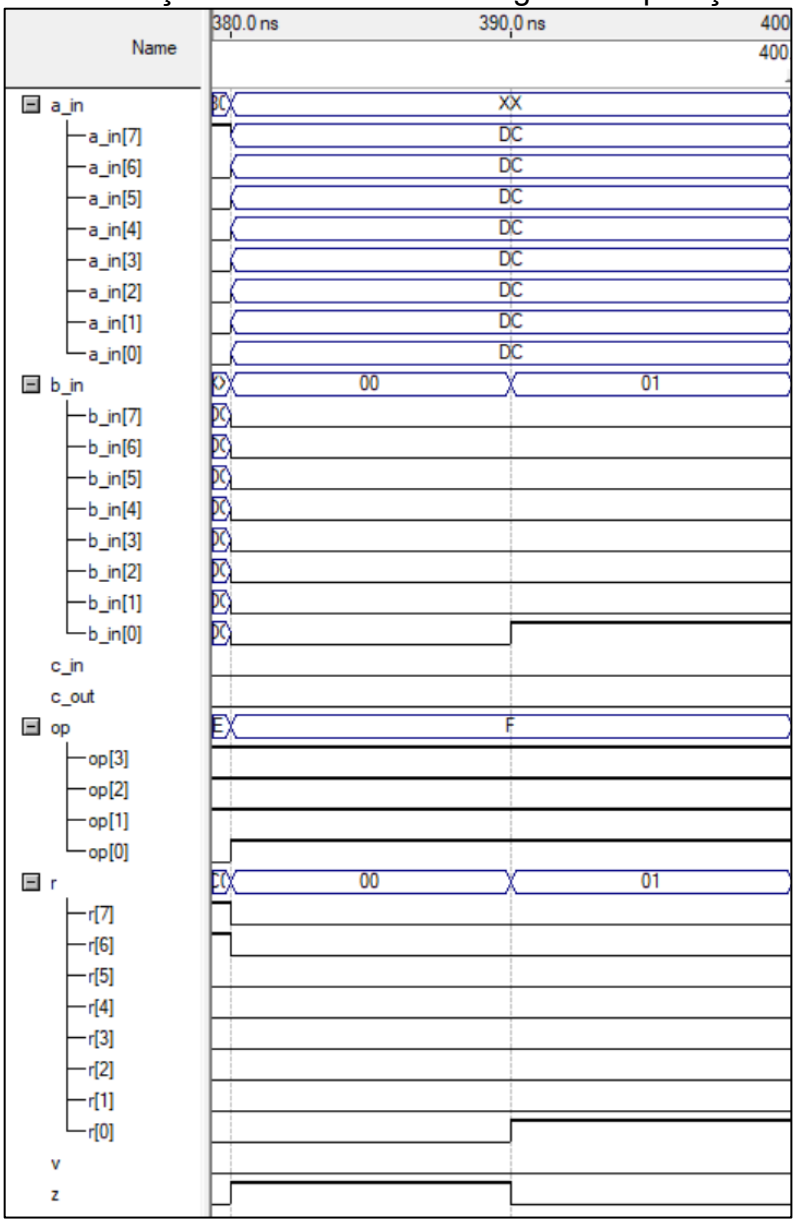
Quadro 19 - Casos de teste para PASS_B.

| **Tempo** | **Variáveis de Entrada** | | | **Resultados esperados nas variáveis de saída** | | | |
|---|---|---|---|---|---|---|---|
| | **a_in** | **b_in** | **c_in** | **r** | **c_out** | **z** | **v** |
| **380ns – 390ns** | DC | 0x00 | DC | 0x00 | 0 | 1 | 0 |
| **390ns – 400ns** | DC | 0x01 | DC | 0x01 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

A Figura 20 mostra os resultados obtidos com os cenários de testes descritos no Quadro 19.

Figura 20 - Simulação em forma de onda digital da operação PASS_B.




Fonte: Autoria própria (2025).

Assim, foi analisado os resultados, valor das variáveis de saída, gerados pela simulação, o que ocasionou nos dados descritos na Tabela 16.

Tabela 16 - Resultados do teste da operação PASS_B.

| Tempo | r (Hexadecimal) | c_out (Binário) | z (Binário) | v (Binário) |
|---|---|---|---|---|
| 380ns – 390ns | 0x00 | 0 | 1 | 0 |
| 390ns – 400ns | 0x01 | 0 | 0 | 0 |

Fonte: Autoria própria (2025).

Comparando os valores obtidos na simulação, listados na Tabela 16, e os valores esperados no cenário de teste, representado no Quadro 19, pode-se observar que dados são equivalentes, desse modo, pode-se afirmar que a execução da

operação PASS_B da ULA está correta. Assim, se finalizou a bateria de testes do projeto com sucesso.

# 3 CONSIDERAÇÕES FINAIS

A construção de uma ULA de 8 bits utilizando a linguagem VHDL representou uma oportunidade enriquecedora para aplicar, na prática, os conteúdos teóricos abordados na disciplina de Sistemas Reconfiguráveis. A opção por utilizar exclusivamente código concorrente contribuiu significativamente para o entendimento da lógica combinacional e do funcionamento de circuitos digitais.

Durante a implementação, foram desenvolvidas diversas estruturas em VHDL para representar as operações lógicas, aritméticas, de deslocamento e rotação que caracterizam o funcionamento de uma ULA. A definição clara das entradas, saídas e operações permitiu uma modelagem precisa e organizada do que representa o hardware.

Os resultados dos testes confirmaram que a ULA implementada apresenta o comportamento esperado, validando a fidelidade do projeto em relação ao seu propósito. A etapa de simulação mostrou-se essencial para garantir o funcionamento correto antes da síntese, reforçando a relevância da verificação lógica no desenvolvimento de sistemas digitais.

Em resumo, o projeto alcançou plenamente seus objetivos, tanto do ponto de vista técnico quanto educacional. Proporcionou uma experiência concreta com ferramentas profissionais de modelagem e de validação de hardware, ao mesmo tempo em que destacou a importância da simulação como uma etapa crítica no processo de desenvolvimento de circuitos digitais. Assim, a atividade contribuiu significativamente para a formação acadêmica, consolidando conhecimentos essenciais em sistemas digitais e reconfiguráveis, com aplicabilidade direta em contextos reais de engenharia.

## REFERÊNCIAS

D'AMORE, Roberto. **VHDL**: descrição e síntese de circuitos digitais. Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora, 2005. ISBN 978-85-216-1452-4.

HEXSEL, Roberto. **VHDL**: descrição e síntese de circuitos digitais. Curitiba: Departamento de Informática, Universidade Federal do Paraná, 2003. Disponível em: <https://www.inf.ufpr.br/hexsel/ci210/vhdl.pdf>. Acesso em: 29 abr. 2025.

INTEL. **Quartus Prime**: ferramentas de desenvolvimento FPGA. Disponível em: <https://www.intel.com.br/content/www/br/pt/products/details/fpga/development-tools/quartus-prime.html>. Acesso em: 29 abr. 2025.

UNIVERSIDADE DE SÃO PAULO. Escola Politécnica. Laboratório de Sistemas Digitais. **Unidade lógica e aritmética (ULA)**. Disponível em: <https://www2.pcs.usp.br/~labdig/pdffiles_2011/ula.pdf>. Acesso em: 29 abr. 2025.

UNIVERSIDADE ESTADUAL DE CAMPINAS. Faculdade de Engenharia Elétrica e de Computação. **Roteiro 3 – EA773**. Disponível em: <https://www.decom.fee.unicamp.br/~cardieri/MaterialEA773/EA773_2S2016_Roteiro3.pdf>. Acesso em: 29 abr. 2025.