



PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Instituto de Ciências Exatas e Informática

Curso de Engenharia de Computação

UNIDADE LÓGICA E ARITMÉTICA VHDL

Alunos:

Alberto Magno Machado

Ana Luiza Diniz Santos

Pablo Marcelino Las-Cazas

Belo Horizonte

Abril de 2025

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Instituto de Ciências Exatas e Informática

Curso de Engenharia de Computação

UNIDADE LÓGICA E ARITMÉTICA

Trabalho desenvolvido para a matéria de
Sistemas Reconfiguráveis do curso de
Engenharia de Computação da PUC Minas.

Professor: Francisco Garcia

Belo Horizonte

Abril de 2025

Sumário

1	Introdução	4
2	Requisitos do Projeto	5
2.1	Requisitos Funcionais	5
3	Desenvolvimento	7
3.1	Implementação VHDL	7
3.2	Organização dos Arquivos do Projeto	9
4	Testes e Resultados	11
4.1	Operações Aritméticas	11
4.2	Operações Lógicas	12
4.3	Rotações e Deslocamentos	13
4.4	Bypass	15
5	Conclusão	16

Lista de Figuras

1	ULA	4
2	Operação ADD (<code>op_sel = 0000</code>): Soma sem carry-in	11
3	Operação ADDC (<code>op_sel = 0001</code>): Soma com carry-in	11
4	Operação ADDC (<code>op_sel = 0001</code>): Soma sem carry-in	11
5	Operação SUB (<code>op_sel = 0010</code>): Subtração sem carry-in	11
6	Operação SUBC (<code>op_sel = 0011</code>): Subtração com carry-in	12
7	Operação SUBC (<code>op_sel = 0011</code>): Subtração sem carry-in	12
8	Operação AND (<code>op_sel = 0100</code>)	12
9	Operação OR (<code>op_sel = 0101</code>)	12
10	Operação XOR (<code>op_sel = 0110</code>)	12
11	Operação NOT (<code>op_sel = 0111</code>)	13
12	Operação RL (<code>op_sel = 1000</code>): Rotação para a esquerda	13
13	Operação RR (<code>op_sel = 1001</code>): Rotação para a direita	13
14	Operação RLC (<code>op_sel = 1010</code>): Rotação à esquerda via carry	13
15	Operação RLC (<code>op_sel = 1010</code>): Rotação à esquerda sem carry	13
16	Operação RRC (<code>op_sel = 1011</code>): Rotação à direita via carry	14
17	Operação RRC (<code>op_sel = 1011</code>): Rotação à direita sem carry	14
18	Operação SLL (<code>op_sel = 1100</code>): Deslocamento lógico à esquerda	14
19	Operação SRL (<code>op_sel = 1101</code>): Deslocamento lógico à direita	14
20	Operação SRA (<code>op_sel = 1110</code>): Deslocamento aritmético à direita	14
21	Operação PASS_B (<code>op_sel = 1111</code>): Passagem direta de <code>b_in</code>	15

Lista de Tabelas

1	Operações aritméticas	5
2	Operações lógicas	5
3	Rotações e deslocamentos	6
4	Operação de bypass	6

1 Introdução

A arquitetura de sistemas digitais modernos exige componentes versáteis e eficientes para o processamento de dados. Dentre esses componentes, a Unidade Lógica e Aritmética (ALU – Arithmetic and Logic Unit) desempenha um papel central na execução de operações fundamentais como somas, subtrações, operações lógicas e manipulações de bits.

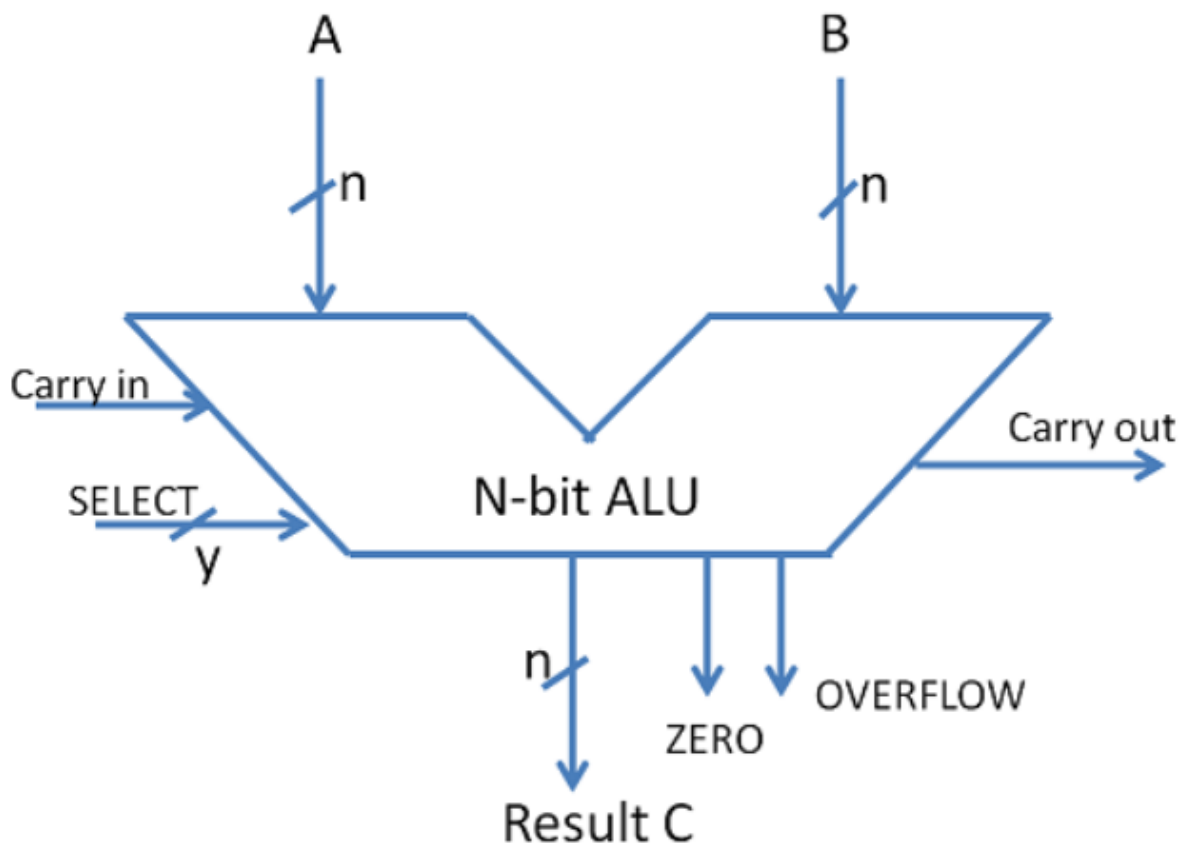


Figura 1: ULA

Este trabalho tem como objetivo o desenvolvimento de uma ALU de 8 bits utilizando a linguagem de descrição de hardware VHDL, respeitando os princípios de design digital combinacional. O projeto foi implementado no ambiente Quartus II 9.1sp2, sendo exclusivamente composto por código concorrente, sem a utilização de elementos sequenciais como flip-flops ou latches. A ALU implementada é capaz de realizar 16 operações distintas, incluindo operações aritméticas com e sem carry, operações lógicas bit a bit, rotações e deslocamentos.

O relatório está organizado da seguinte forma: na seção 2 são apresentados os requisitos do projeto; a seção 3 descreve o desenvolvimento da ALU, incluindo decisões de projeto e trechos de código relevantes; a seção 4 detalha os testes realizados e os resultados obtidos; por fim, a seção 5 apresenta as considerações finais.

2 Requisitos do Projeto

O presente trabalho tem como finalidade o desenvolvimento de uma Unidade Lógica e Aritmética (ALU) descrita em VHDL, operando com palavras de 8 bits e implementada de forma totalmente combinacional. O projeto deve ser implementado e simulado utilizando exclusivamente a versão 9.1sp2 do software Quartus II.

A seguir, são apresentados os requisitos funcionais e técnicos que a ALU deve atender.

2.1 Requisitos Funcionais

A ALU deve ser capaz de realizar 16 operações distintas, conforme o valor de entrada de seleção de operação (`op_sel[3..0]`), agrupadas em:

Operações Aritméticas

op_sel	Mnemônico	Descrição
0000	ADD	Soma sem carry-in: $r_out = a + b$
0001	ADDC	Soma com carry-in: $r_out = a + b + c_in$
0010	SUB	Subtração sem carry-in: $r_out = a - b$
0011	SUBC	Subtração com carry-in: $r_out = a - b - c_in$

Tabela 1: Operações aritméticas

Operações Lógicas

op_sel	Mnemônico	Descrição
0100	AND	AND lógico bit a bit: $r_out = a \wedge b$
0101	OR	OR lógico bit a bit: $r_out = a \vee b$
0110	XOR	XOR lógico bit a bit: $r_out = a \oplus b$
0111	NOT	Complemento bit a bit: $r_out = \sim a$

Tabela 2: Operações lógicas

Rotações e Deslocamentos

op_sel	Mnemônico	Descrição
1000	RL	Rotação à esquerda: $r_out = a[6..0], a[7]$
1001	RR	Rotação à direita: $r_out = a[0], a[7..1]$
1010	RLC	Rotação à esquerda via carry: $r_out = a[6..0], c_in$
1011	RRC	Rotação à direita via carry: $r_out = c_in, a[7..1]$
1100	SLL	Deslocamento lógico à esquerda: $r_out = a[6..0], 0$
1101	SRL	Deslocamento lógico à direita: $r_out = 0, a[7..1]$
1110	SRA	Deslocamento aritmético à direita: $r_out = a[7], a[7..1]$

Tabela 3: Rotações e deslocamentos

Bypass

op_sel	Mnemônico	Descrição
1111	PASS_B	Passa o valor de b_in diretamente: $r_out = b$

Tabela 4: Operação de bypass

3 Desenvolvimento

A ALU foi implementada utilizando a linguagem VHDL, respeitando a estrutura combinacional exigida no projeto, sem o uso de elementos sequenciais como *flip-flops* ou *latches*.

3.1 Implementação VHDL

A seguir, apresenta-se a declaração da entidade `alu`, incluindo as portas de entrada e saída:

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  ENTITY alu IS
6      PORT (
7          -- Entradas
8          a_in : IN STD_LOGIC_VECTOR(7 DOWNTO 0);    -- Operando A
9          b_in : IN STD_LOGIC_VECTOR(7 DOWNTO 0);    -- Operando B
10         c_in : IN STD_LOGIC;                        -- Carry-in
11         op_sel : IN STD_LOGIC_VECTOR(3 DOWNTO 0);  -- Operacao Selecionada
12
13         -- Saidas
14         r_out : OUT STD_LOGIC_VECTOR(7 DOWNTO 0); -- Resultado
15         c_out : OUT STD_LOGIC;                    -- Carry/Borrow
16         z_out : OUT STD_LOGIC;                    -- Indicador de Zero
17         v_out : OUT STD_LOGIC;                    -- Indicador de Overflow
18     );
19 END ENTITY;
```

Listing 1: Declaração da entidade `alu`

A seguir, são apresentados os sinais internos utilizados na arquitetura:

```

1  ARCHITECTURE arch OF alu IS
2      SIGNAL temp_soma : STD_LOGIC_VECTOR(8 DOWNTO 0); -- Resultado da soma
3      SIGNAL temp_sub  : STD_LOGIC_VECTOR(8 DOWNTO 0); -- Resultado da subtracao
4      SIGNAL temp_r     : STD_LOGIC_VECTOR(7 DOWNTO 0); -- Resultado final
```

Listing 2: Declaração dos sinais internos

A implementação das operações de soma e subtração com e sem carry pode ser vista abaixo:

```

1  -- Calculo da Soma (ADD e ADDC)
2  temp_soma <=
3  STD_LOGIC_VECTOR(unsigned('0' & a_in) + unsigned('0' & b_in)) WHEN op_sel = "0000" ELSE
4  STD_LOGIC_VECTOR(unsigned('0' & a_in) + unsigned('0' & b_in) + 1) WHEN op_sel = "0001"
5  ELSE
6  (others => '0');
7
8  -- Calculo da Subtracao (SUB e SUBC)
9  temp_sub <=
10 STD_LOGIC_VECTOR(unsigned('0' & a_in) - unsigned('0' & b_in)) WHEN op_sel = "0010" ELSE
11 STD_LOGIC_VECTOR(unsigned('0' & a_in) - unsigned('0' & b_in) - 1) WHEN op_sel = "0011"
    ELSE
    (others => '0');

```

Listing 3: Cálculo de soma e subtração

As 16 operações são selecionadas por meio da diretiva `with...select`, conforme segue:

```

1  WITH op_sel SELECT
2  temp_r <=
3      temp_soma(7 DOWNT0 0) WHEN "0000", -- ADD
4      temp_soma(7 DOWNT0 0) WHEN "0001", -- ADDC
5      temp_sub(7 DOWNT0 0) WHEN "0010", -- SUB
6      temp_sub(7 DOWNT0 0) WHEN "0011", -- SUBC
7
8      a_in AND b_in WHEN "0100", -- AND
9      a_in OR b_in WHEN "0101", -- OR
10     a_in XOR b_in WHEN "0110", -- XOR
11     NOT a_in WHEN "0111", -- NOT
12
13     a_in(6 DOWNT0 0) & a_in(7) WHEN "1000", -- RL
14     a_in(0) & a_in(7 DOWNT0 1) WHEN "1001", -- RR
15     a_in(6 DOWNT0 0) & c_in WHEN "1010", -- RLC
16     c_in & a_in(7 DOWNT0 1) WHEN "1011", -- RRC
17     a_in(6 DOWNT0 0) & '0' WHEN "1100", -- SLL
18     '0' & a_in(7 DOWNT0 1) WHEN "1101", -- SRL
19     a_in(7) & a_in(7 DOWNT0 1) WHEN "1110", -- SRA
20     b_in WHEN "1111"; -- PASS_B

```

Listing 4: Bloco de seleção de operações da ALU

A lógica de atribuição do sinal `c_out` considera o tipo de operação realizada:

```

1  -- Saida de Carry/Borrow
2  c_out <=
3      temp_soma(8) WHEN (op_sel = "0000" OR op_sel = "0001") ELSE -- Carry da ADD/ADDC
4      NOT temp_sub(8) WHEN (op_sel = "0010" OR op_sel = "0011") ELSE -- Borrow da SUB/SUBC
5      a_in(7) WHEN (op_sel = "1000" OR op_sel = "1010" OR op_sel = "1100") ELSE -- Rotacao/
        Deslocamento a esquerda
6      a_in(0) WHEN (op_sel = "1001" OR op_sel = "1011" OR op_sel = "1101" OR op_sel = "1110")
        ELSE -- Rotacao/Deslocamento a direita
7      '0';

```

Listing 5: Atribuição de c_out

O sinal z_out é ativado quando o resultado da operação é igual a zero:

```

1  -- Saida de Zero
2  z_out <= '1' WHEN temp_r = "00000000" ELSE '0'; -- Recebe 1 quando o resultado final e
        zero, caso contrario, recebe 0

```

Listing 6: Atribuição de z_out

A lógica de v_out detecta overflow nas operações aritméticas:

```

1  -- Saida de Overflow
2  v_out <=
3      -- Overflow na Soma
4      ((NOT a_in(7) AND NOT b_in(7) AND temp_r(7)) OR (a_in(7) AND b_in(7) AND NOT temp_r(7)))
5      WHEN (op_sel = "0000" OR op_sel = "0001") ELSE
6      -- Overflow na Subtracao
7      ((a_in(7) AND NOT b_in(7) AND NOT temp_r(7)) OR (NOT a_in(7) AND b_in(7) AND temp_r(7)))
8      WHEN (op_sel = "0010" OR op_sel = "0011") ELSE
9      '0';

```

Listing 7: Atribuição de v_out

A saída principal da ALU é atribuída com o valor de temp_r:

```

1  -- Atribuicao final da saida
2  r_out <= temp_r;
3  END arch;

```

Listing 8: Atribuição final da saída r_out

3.2 Organização dos Arquivos do Projeto

O projeto foi salvo no diretório E1_ALU/ALU, contendo os seguintes arquivos principais:

- **alu.vhd**: código-fonte VHDL da ALU, com estrutura combinacional.
- **alu.vwf**: arquivo de forma de onda (Waveform File), utilizado para simulação no Quartus II.
- **Relatórios de compilação**: arquivos com extensão **.rpt**, **.summary**, **.sof**, **.pof**, entre outros, gerados automaticamente pelo ambiente de desenvolvimento Quartus II durante a síntese e análise do projeto.
- **alu.sim.rpt**: relatório contendo os resultados das simulações funcionais realizadas sobre o circuito.
- **alu.done**: arquivo que indica a compilação bem-sucedida do projeto.

4 Testes e Resultados

Após a implementação da ALU, foram realizados testes para cada uma das 16 operações especificadas. As simulações foram feitas utilizando o editor de formas de onda (VWF) do Quartus II, com diferentes valores de entrada para os sinais **a_in**, **b_in**, **c_in** e **op_sel**. Cada teste resultou em uma imagem capturada da simulação, localizada na pasta **Testes/** do projeto.

A seguir, são apresentados os resultados obtidos em cada operação, com a respectiva análise.

4.1 Operações Aritméticas

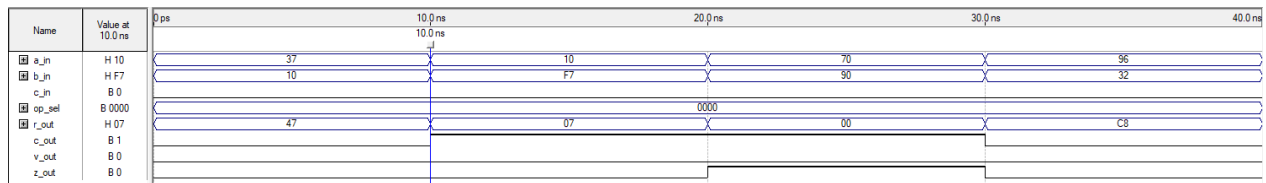


Figura 2: Operação ADD (**op_sel** = 0000): Soma sem carry-in

Nesta simulação, observa-se que o valor de **r_out** corresponde corretamente à soma de **a_in** e **b_in**. O sinal **c_out** indica a ocorrência (ou não) de carry, e **v_out** sinaliza overflow quando aplicável.

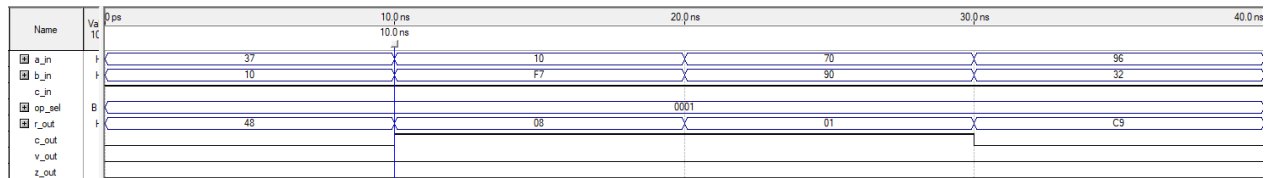


Figura 3: Operação ADDC (**op_sel** = 0001): Soma com carry-in

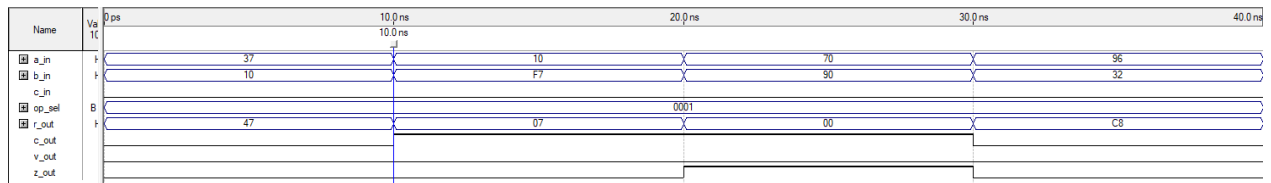


Figura 4: Operação ADDC (**op_sel** = 0001): Soma sem carry-in

Neste caso, o bit de carry-in (**c_in**) é adicionado ao resultado, afetando tanto o valor de saída quanto o **c_out** final.

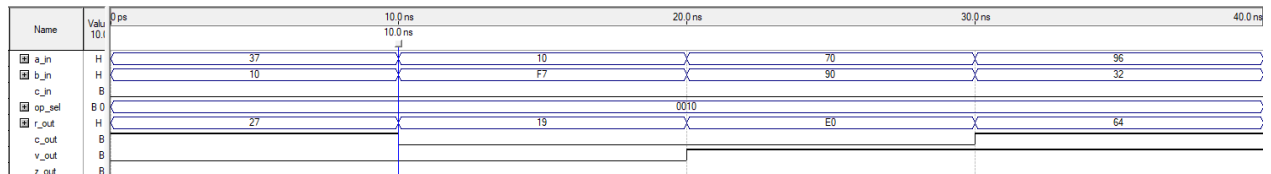


Figura 5: Operação SUB (**op_sel** = 0010): Subtração sem carry-in

A subtração simples de **a_in** menos **b_in** é realizada, e o **c_out** representa o borrow.

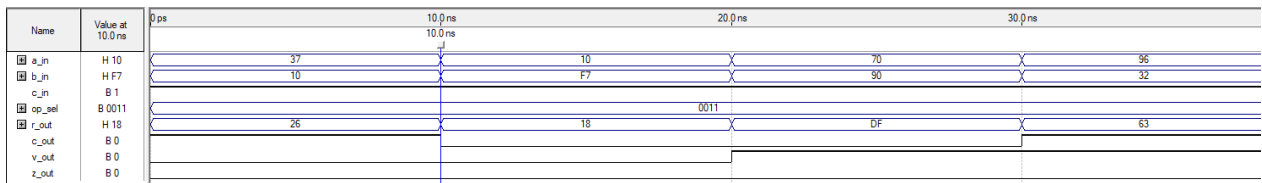


Figura 6: Operação SUBC (op_sel = 0011): Subtração com carry-in

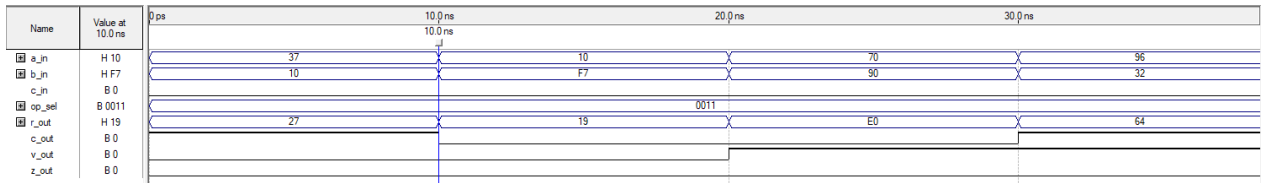


Figura 7: Operação SUBC (op_sel = 0011): Subtração sem carry-in

A subtração considera o carry-in como um decrementador adicional, afetando o resultado final e os sinais auxiliares.

4.2 Operações Lógicas

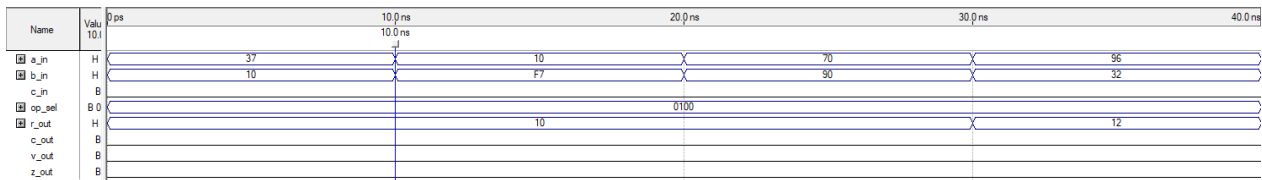


Figura 8: Operação AND (op_sel = 0100)

A saída corresponde ao resultado do AND bit a bit entre os operandos a_in e b_in. O sinal z_out indica se o resultado foi nulo.

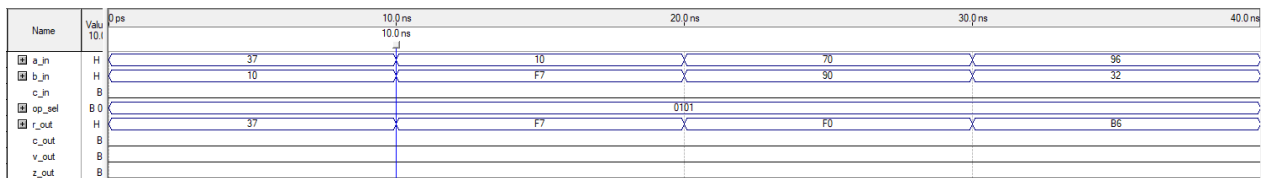


Figura 9: Operação OR (op_sel = 0101)

Operação OR bit a bit entre os dois operandos, útil para ativar bits em paralelo.

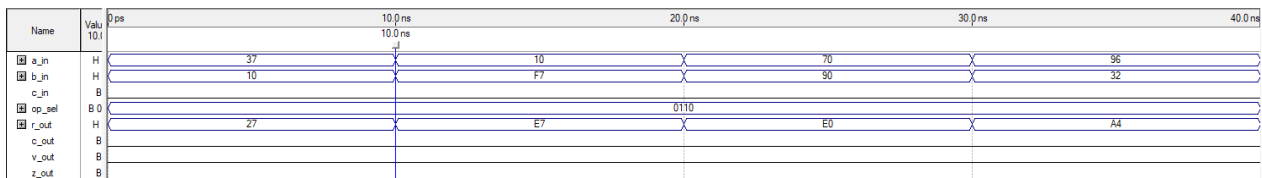


Figura 10: Operação XOR (op_sel = 0110)

Realiza o XOR bit a bit, com o sinal de zero sendo ativado se todos os bits da saída forem '0'.

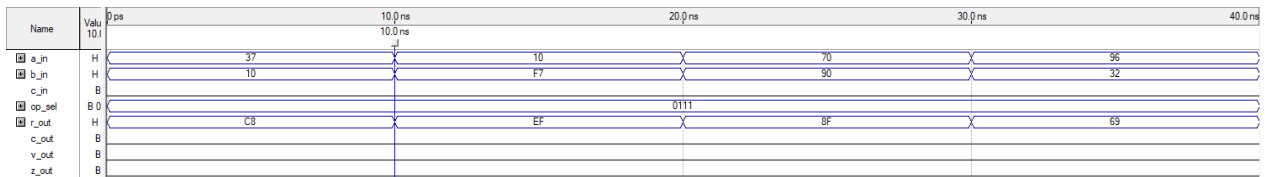


Figura 11: Operação NOT (op_sel = 0111)

Complementa todos os bits de **a_in**, ignorando **b_in** e **c_in**.

4.3 Rotações e Deslocamentos

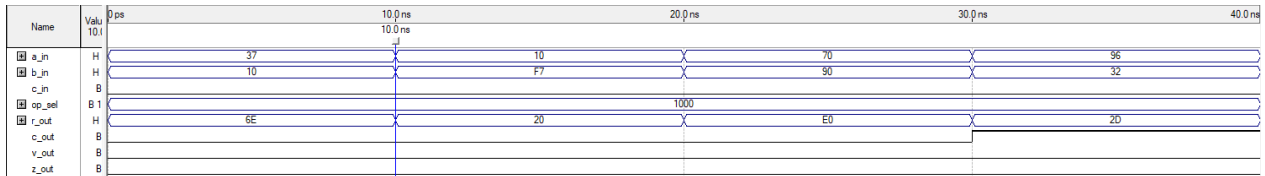


Figura 12: Operação RL (op_sel = 1000): Rotação para a esquerda

Rotaciona os bits de **a_in** para a esquerda, com o bit mais significativo indo para a posição menos significativa.

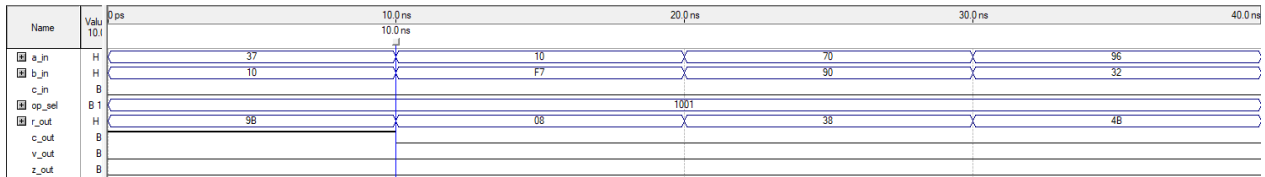


Figura 13: Operação RR (op_sel = 1001): Rotação para a direita

Rotação simples para a direita, levando o LSB para o MSB.

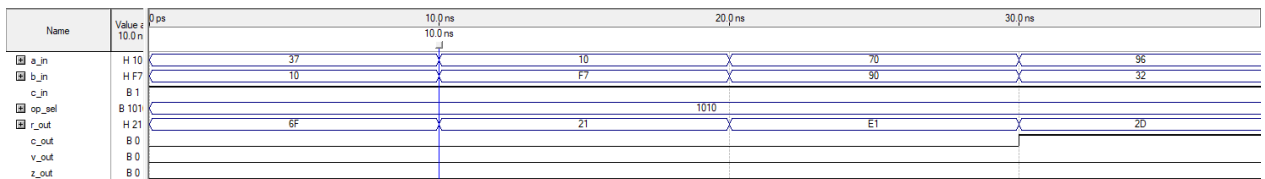


Figura 14: Operação RLC (op_sel = 1010): Rotação à esquerda via carry

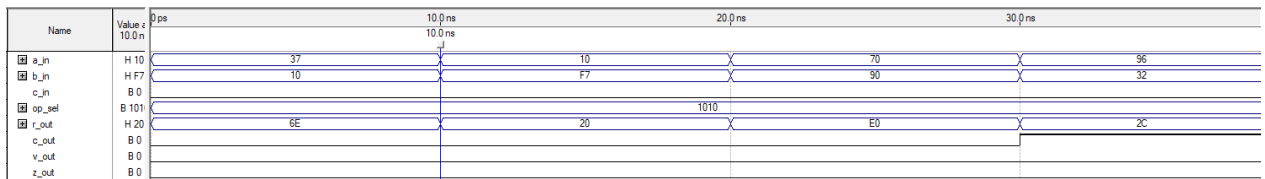


Figura 15: Operação RLC (op_sel = 1010): Rotação à esquerda sem carry

Aqui, o bit de carry-in é inserido como LSB, e o MSB de **a_in** torna-se o **c_out**.

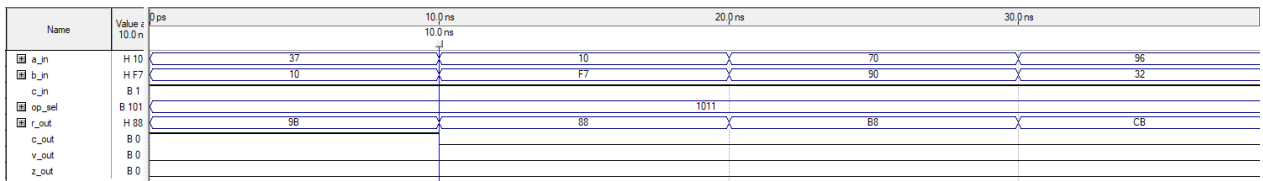


Figura 16: Operação RRC (op_sel = 1011): Rotação à direita via carry

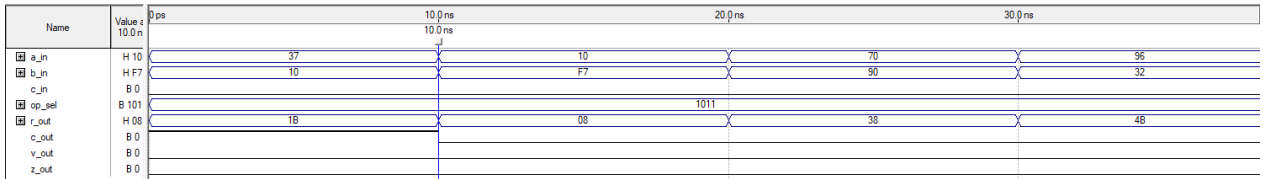


Figura 17: Operação RRC (op_sel = 1011): Rotação à direita sem carry

Similar à RLC, mas rotacionando para a direita. O carry-in ocupa o MSB, e o LSB vira c_out.

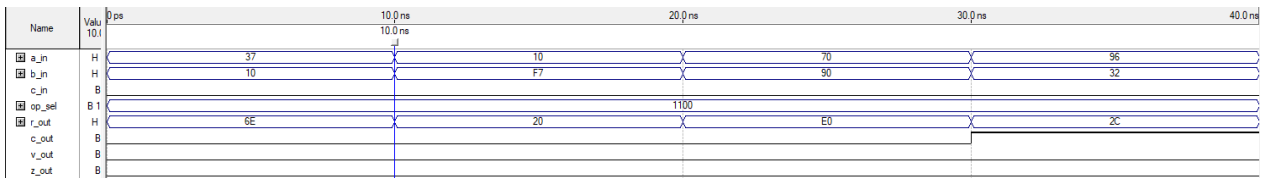


Figura 18: Operação SLL (op_sel = 1100): Deslocamento lógico à esquerda

Desloca os bits à esquerda e insere zero no LSB.

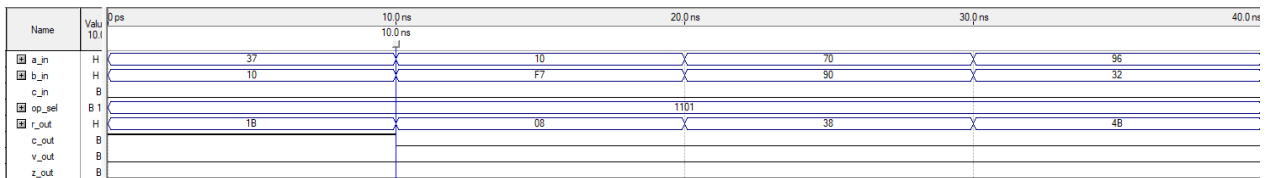


Figura 19: Operação SRL (op_sel = 1101): Deslocamento lógico à direita

Desloca os bits à direita, preenchendo o MSB com zero.

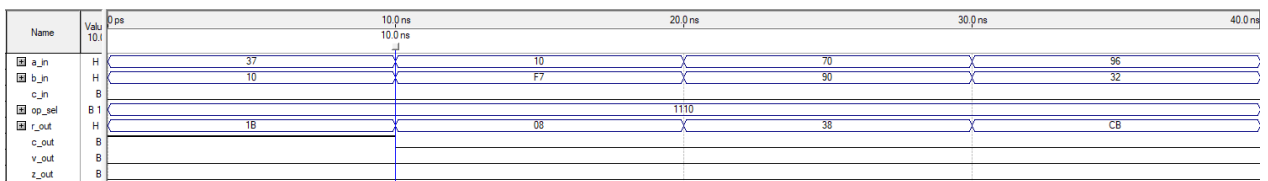


Figura 20: Operação SRA (op_sel = 1110): Deslocamento aritmético à direita

Mantém o bit de sinal (MSB) ao deslocar bits à direita.

4.4 Bypass

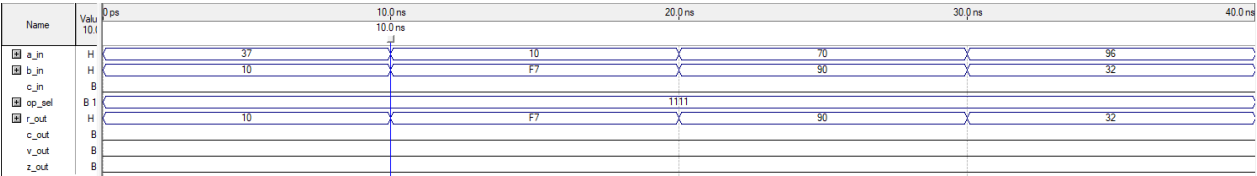


Figura 21: Operação PASS_B (op_sel = 1111): Passagem direta de b_in

Nesta operação, o valor de b_in é passado diretamente para a saída r_out, ignorando a_in e c_in.

5 Conclusão

O desenvolvimento de uma Unidade Lógica e Aritmética (ALU) de 8 bits utilizando a linguagem VHDL permitiu consolidar conhecimentos teóricos e práticos relacionados à lógica digital, à modelagem de hardware e ao uso do ambiente de desenvolvimento Quartus II.

A implementação foi conduzida de forma estruturada, respeitando os requisitos propostos, com a construção de um circuito totalmente combinacional e a aplicação de técnicas de codificação concorrente. A ALU resultante foi capaz de executar corretamente as 16 operações especificadas, incluindo funções aritméticas, lógicas, rotações e deslocamentos, além de sinalizar adequadamente os estados de carry, zero e overflow.

As simulações realizadas confirmaram a conformidade do comportamento funcional do projeto com as especificações fornecidas. O uso de arquivos de forma de onda (`.vwf`) e os relatórios de simulação evidenciaram que todas as operações foram validadas com sucesso.

Referências

Witscad. *Arithmetic Logic Unit Design*. Disponível em: <https://witscad.com/course/computer-architecture/chapter/arithmetic-logic-unit-design>.

Staff. *VHDL References Manual*. Disponível em: <https://staff.fysik.su.se/~silver/digsyst/vhdl.ref.pdf>

Intel Corporation. *Quartus II Handbook, Version 9.1sp2*.