



Universidade Federal do Rio de Janeiro
Escola Politécnica
Departamento de Engenharia Eletrônica e de Computação

UNIDADE LÓGICA ARITMÉTICA “ULA” PROGRAMADO EM VHDL

Grupo:

Caio Peixoto Galdino

Raiane de Almeida M. Nascimento

Thomas Cardoso de Miranda

2022/2

1. Introdução.

A unidade lógica e aritmética (ULA) ou em inglês Arithmetic Logic Unit (ALU) é um circuito digital que realiza operações lógicas e aritméticas. A ULA é uma peça fundamental da unidade central de processamento (CPU), e até dos mais simples microprocessadores. É na verdade, uma "grande calculadora eletrônica" do tipo desenvolvido durante a II Guerra Mundial, e sua tecnologia já estava disponível quando os primeiros computadores modernos foram construídos.

A ULA executa as principais operações lógicas e aritméticas do computador. Ela soma, subtrai, divide, determina se um número é positivo ou negativo ou se é zero. Além de executar funções aritméticas, uma ULA deve ser capaz de determinar se uma quantidade é menor ou maior que outra e quando quantidades são iguais. A ULA pode executar funções lógicas com letras e com números.

2. Objetivo.

Este trabalho visa mostrar a implementação da Unidade Lógico Aritmética em VHDL, demonstrando os códigos utilizados e os possíveis problemas para apresentação da saída de dados na Spartan-3A/3AN FPGA Starter, que foi a placa utilizada para mostrar os resultados obtidos. Será apresentado no decorrer deste trabalho a forma de implementação de todo o circuito, toda a estrutura de código utilizado, além de comentários sobre o funcionamento e porquê do código utilizado.

3. Metodologia.

Todo o projeto foi desenvolvido na linguagem VHDL através do software da Xilinx

4. Unidade Lógica Aritmética

Uma representação simplificada de uma ULA está apresentada na Figura abaixo, onde temos A e B que são os dados sobre os quais será realizada uma operação. Como há várias opções para as operações, há uma seleção de qual deve ser a operação, indicada pela entrada S. O resultado da operação é apresentado na saída da ULA.

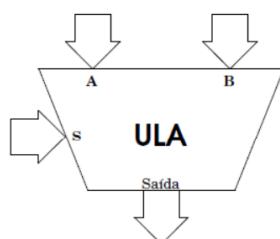


Figura 1 - Esquema simplificado de uma ULA

4.1 Especificações.

A seguir serão apresentadas essas partes, bem como a suas funções e como elas foram devidamente desenvolvidas nesse projeto. Além disso, serão mostrados outros componentes como multiplexadores e Decodificadores, visto que sem eles algumas funcionalidades da ULA proposta no projeto não poderiam ser implementadas, como o acendimento de LED em um display, que representa o resultado de uma operação realizada pela unidade.

Nossa ULA fará as seguintes operações:

- Soma
- Subtração em Complemento de 2
- Troca de sinal
- Incremento +1
- Deslocamento para Esquerda
- Deslocamento para Direita
- AND
- XOR

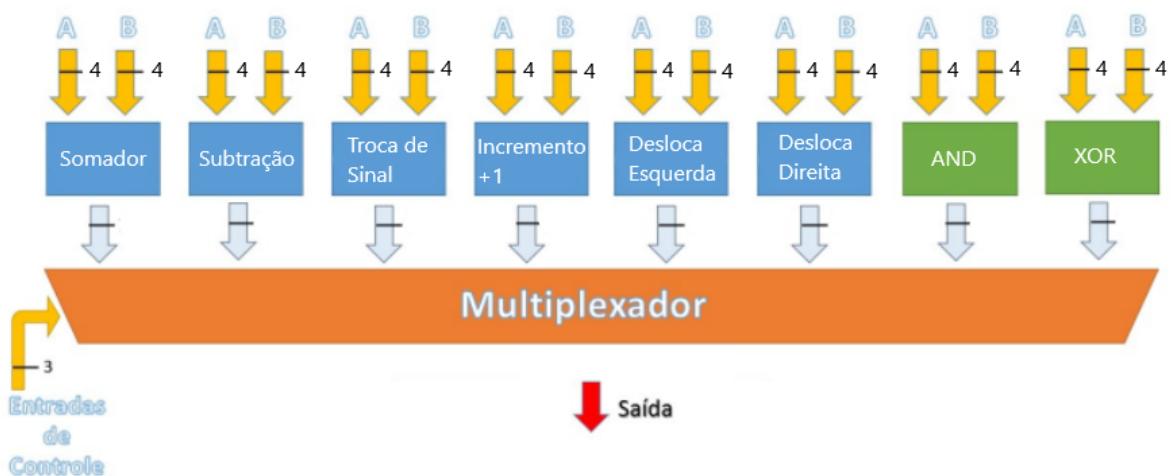


Figura 2: Diagrama de blocos da ULA

Pode-se visualizar que como entrada da ULA temos as chaves A e B, onde cada uma possui 4 bits. A notação de vetor de bits foi utilizada para simplificar o esquemático. Além das entradas A e B, pode-se verificar também a “Entrada de Controle” que faz a seleção de operação desejada na ULA, e está também possui 4 bits.

A entrada de controle possui a mesma lógica utilizada em multiplexadores. Cada bloco lógico realiza a sua operação e envia ao multiplexador sua saída. Este multiplexador possui 8 entradas, mas cada entrada é um vetor de bits, como se pode ver na figura 2. Assim como já configurado para as entradas, é muito mais interessante trabalhar com

vetores em VHDL pois simplifica bastante o circuito.

5. Desenvolvimento

Utilizando os conceitos aprendidos em sala de aula foram montados os programas das 8 operações. O primeiro passo foi escolher as operações adicionais que seriam realizadas e definir a identificação de estado de cada operação. Desta forma, obtivemos a tabela abaixo:

X	Y	Z	Operações
0	0	0	Soma
0	0	1	Subtração em CPL2
0	1	0	Incremento +1
0	1	1	Troca de Sinal
1	0	0	Desloca Esquerda
1	0	1	Desloca Direita
1	1	0	XOR
1	1	1	AND

5.1.1 Somador

Este módulo tem a função de realizar as operações de soma, dado um vetor de bits. Ele será composto de uma unidade básica responsável pela soma de dois bits (1 bit da entrada A e 1 bit da entrada B) e de uma unidade maior que reúne várias unidades básicas. Esta última tem a capacidade de realizar a soma de uma cadeia de vários bits.

Logo, este bloco está dividido em duas sources:

- **Somador Completo de 1 bit:**

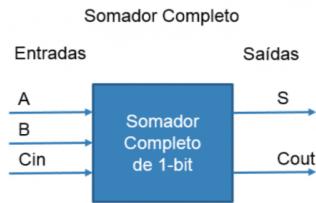


Figura 3: Esquema Módulo Somador

A	B	Cin	Saída	COut
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabela 1 – Formato de uma soma binária

```

1  library ieee;
2  use IEEE.STD_LOGIC_1164.all;
3
4  entity somador_completo_1bit is
5      port (
6          A: in std_logic;
7          B: in std_logic;
8          Cin: in std_logic;
9          Cout: out std_logic;
L0         Z: out std_logic
L1     );
L2  end somador_completo_1bit;
L3
L4  architecture logica of somador_completo_1bit is
L5      begin
L6          Z <= A XOR B XOR Cin;
L7          Cout <= (A AND B) OR (cin AND (A XOR B));
L8      end logica;

```

Lógica somador de 1-bit em VHDL

Acima implementamos um Somador Completo de 1 bit com base nos nossos conhecimentos de Sistemas Digitais, usando lógicas de ANDs e XORs.

- Somador Completo 4 bits

Por fim, há a junção desses módulos somadores, para formar um somador completo de n-bits, em nosso projeto o número total de bits é 4

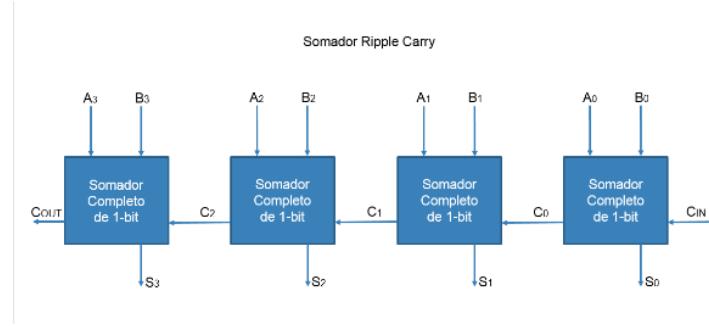


Figura 4: Esquema Somador de n -bits

```

1  library ieee;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY somador_completo_4bits IS
5    PORT (
6      A: in std_logic_vector(3 downto 0);
7      B: in std_logic_vector(3 downto 0);
8      Cin: in std_logic;
9      Cout, Overflow: out std_logic;
10     Z: out std_logic_vector(3 downto 0)
11   );
12 END somador_completo_4bits;
13
14 ARCHITECTURE logica OF somador_completo_4bits IS
15   COMPONENT somador_completo_1bit
16   PORT (
17     A, B, cin: in std_logic;
18     Cout, z: out std_logic
19   );
20   END COMPONENT;
21
22 SIGNAL C : std_logic_vector(3 downto 0);
23
24 BEGIN
25
26   S0: somador_completo_1bit PORT MAP(A => A(0), B => B(0), Cin => Cin, Cout => C(0), Z => Z(0));
27   S1: somador_completo_1bit PORT MAP(A => A(1), B => B(1), Cin => C(0), Cout => C(1), Z => Z(1));
28   S2: somador_completo_1bit PORT MAP(A => A(2), B => B(2), Cin => C(1), Cout => C(2), Z => Z(2));
29   S3: somador_completo_1bit PORT MAP(A => A(3), B => B(3), Cin => C(2), Cout => C(3), Z => Z(3));
30
31   Cout <= C(3);
32   overflow <= (C(3) XOR C(2));
33
34 END logica;

```

Lógica somador de 4-bit em VHDL

Acima mostramos, que fizemos uso do FA 1 bit em cascata para conseguirmos um FA de 4 bits. A source anterior é utilizada como componente e definimos, como sabemos de Sistemas Digitais, o overflow e o carry de saída Cout. Neste trabalho eles não são saídas da ULA, mas podem ser utilizados em implementações futuras.

5.1.2 Subtrator

Este módulo tem a função de realizar as operações de subtração, dado um vetor de bits. Ele será composto pela mesma unidade básica responsável pela operação de soma. Este módulo é uma combinação da utilização de dois módulos, somador e complemento a 2. Utilizando como princípio a operação na forma: $A + (-B)$.

Neste blocos utilizamos o Full Adder de 4 bits para realizar uma soma com o segundo número em complemento de 2: Invertemos Y bit a bit, somamos 1 a este valor por meio do Carry de entrada e somamos este valor a X, obtendo nosso Z final.

5.1.3 Troca de Sinal

Este módulo tem a função de realizar a operação de troca de sinal em complemento a 2, dado um vetor de bits. Ele será composto pela mesma unidade básica responsável pela operação de soma. Este módulo é, teoricamente, uma combinação da utilização de duas funções NOT e incremento +1.

```
1 library ieee;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 ENTITY troca_de_sinal IS
5     PORT (
6         A: in std_logic_vector(3 downto 0);
7         Z: out std_logic_vector(3 downto 0)
8     );
9 END troca_de_sinal;
10
11 ARCHITECTURE logica OF troca_de_sinal IS
12     COMPONENT somador_completo_4bits
13         PORT (
14             A, B: in std_logic_vector(3 downto 0);
15             Cin: in std_logic;
16             Cout: out std_logic;
17             Z: out std_logic_vector(3 downto 0)
18         );
19     END COMPONENT;
20
21     SIGNAL S: std_logic_vector(3 downto 0);
22     SIGNAL Um : std_logic := '1';
23
24 BEGIN
25     S(0) <= NOT A(0);
26     S(1) <= NOT A(1);
27     S(2) <= NOT A(2);
28     S(3) <= NOT A(3);
29
30     R: somador_completo_4bits PORT MAP(A => S, B => "0000", Cin => '1', Z => Z);
31 
```

Lógica troca de sinal em VHDL

5.1.4 Incremento +1

Este módulo tem a função de realizar a operação de incremento +1, dado um vetor de bits. Ele será composto também pela mesma unidade básica responsável pela operação de soma. Este módulo é um caso específico da operação de soma, em o operando Y é por padrão um vetor de zeros e temos um Cin=1 do módulo dos bits menos significativos.

5.1.5 Desloca Esquerda

Este módulo tem a função de realizar a operação Deslocamento para Esquerda, dado um vetor de bits. Esta operação é equivalente a multiplicar um número por 2.

```

1  Library ieee;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY deslocar_esquerda IS
5    PORT (
6      A: in std_logic_vector(3 downto 0);
7      Z: out std_logic_vector(3 downto 0)
8    );
9  END deslocar_esquerda;
10
11 ARCHITECTURE logica OF deslocar_esquerda IS
12 BEGIN
13   Z(1) <= A(0);
14   Z(2) <= A(1);
15   Z(3) <= A(2);
16   Z(0) <= A(3);
17 END logica;

```

Lógica Desloca Esquerda em VHDL

5.1.6 Desloca Direita

Este módulo tem a função de realizar a operação Deslocamento para Direita, dado um vetor de bits. Esta operação é equivalente a dividir de forma inteira um número por 2.

```

1  Library ieee;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY deslocar_direita IS
5    PORT (
6      A: in std_logic_vector(3 downto 0);
7      Z: out std_logic_vector(3 downto 0)
8    );
9  END deslocar_direita;
10
11 ARCHITECTURE logica OF deslocar_direita IS
12 BEGIN
13   Z(3) <= A(0);
14   Z(0) <= A(1);
15   Z(1) <= A(2);
16   Z(2) <= A(3);
17 END logica;

```

Lógica Desloca Direita em VHDL

5.1.7 Multiplicador bit a bit (AND)

A multiplicação, pensando primeiramente em apenas um bit, se trata apenas de uma porta AND, ou seja:

A	B	A AND B	A x B
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Tabela 1 – Formato de uma multiplicação bit a bit

Em nosso código inicialmente fizemos a multiplicação de todos os bits de A com os bits de B utilizando portas AND. Depois apenas somamos os bits de acordo com seu valor relativo na multiplicação, utilizando somadores já mencionados acima, e obtivemos o resultado.

```

1 library ieee;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 ENTITY and_4bits IS
5   PORT (
6     A,B: in std_logic_vector(3 downto 0);
7     Z: out std_logic_vector(3 downto 0)
8   );
9 END and_4bits;
10
11 ARCHITECTURE logica OF and_4bits IS
12 BEGIN
13   Z(0) <= A(0) AND B(0);
14   Z(1) <= A(1) AND B(1);
15   Z(2) <= A(2) AND B(2);
16   Z(3) <= A(3) AND B(3);
17 END logica;
```

Lógica porta AND em VHDL

5.1.8 Ou exclusivo bit a bit (XOR)

Função que executa XOR bit a bit.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 2 – Formato de um XOR

Em nosso código fizemos dos bits de A com os bits de B utilizando portas XOR.

```

1  library ieee;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY xor_4bits IS
5    PORT (
6      A,B: in std_logic_vector(3 downto 0);
7      Z: out std_logic_vector(3 downto 0)
8    );
9  END xor_4bits;
10
11 ARCHITECTURE logica OF xor_4bits IS
12 BEGIN
13   Z(0) <= A(0) XOR B(0);
14   Z(1) <= A(1) XOR B(1);
15   Z(2) <= A(2) XOR B(2);
16   Z(3) <= A(3) XOR B(3);
17 END logica;

```

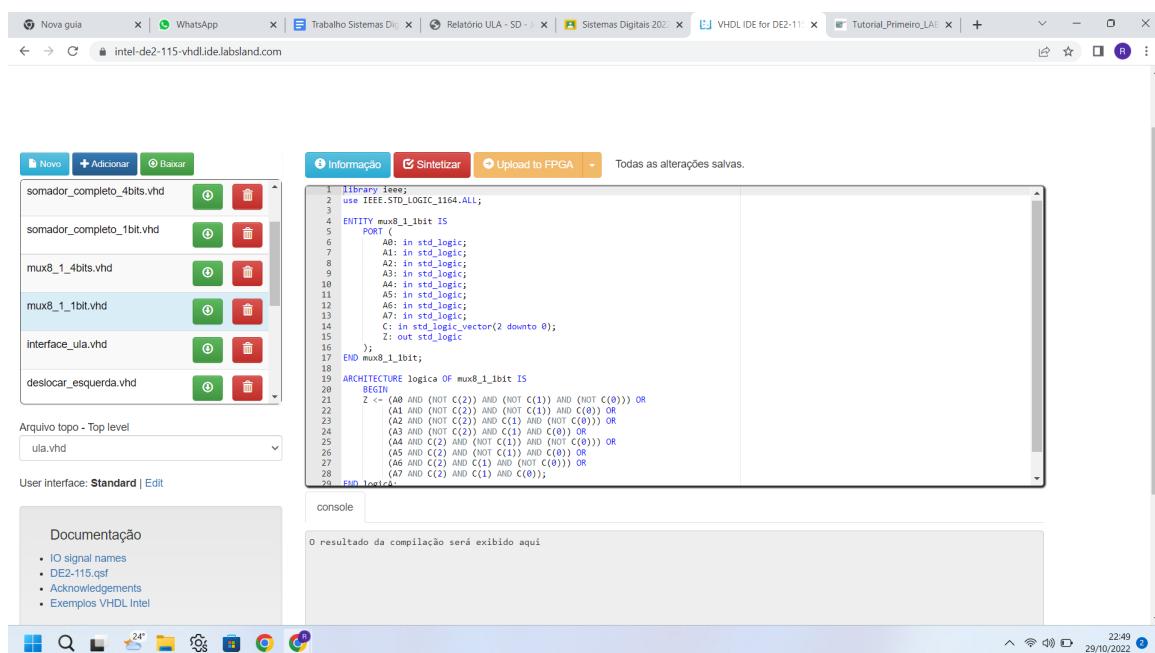
Lógica porta XOR em VHDL

5.2 Exibição dos resultados

5.2.1 Multiplexador

Responsável por escolher qual será a saída relaciona ao código da operação

- 1 bit



- 4 bit

```

library ieee;
use IEEE.STD_LOGIC_1164.all;

entity mux8_1_4bits is
    port (
        AB: in std_logic_vector(3 downto 0);
        A1: in std_logic_vector(3 downto 0);
        A2: in std_logic_vector(3 downto 0);
        A3: in std_logic_vector(3 downto 0);
        A4: in std_logic_vector(3 downto 0);
        A5: in std_logic_vector(3 downto 0);
        A6: in std_logic_vector(3 downto 0);
        A7: in std_logic_vector(3 downto 0);
        C: in std_logic_vector(2 downto 0);
        Z: out std_logic_vector(3 downto 0)
    );
end mux8_1_4bits;

architecture logicA of mux8_1_4bits is
    component mux8_1_1bit is
        port (
            A0: in std_logic;
            A1: in std_logic;
            A2: in std_logic;
            A3: in std_logic;
            A4: in std_logic;
            A5: in std_logic;
            A6: in std_logic;
            A7: in std_logic;
            C: in std_logic;
            Z: out std_logic
        );
    end component;
begin
    M0: mux8_1_1bit port map(
        A0 => AB(0), A1 => A1(0), A2 => A2(0),
        A3 => A3(0), A4 => A4(0), A5 => A5(0),
        A6 => A6(0), A7 => A7(0), C => C, Z => Z(0)
    );
    M1: mux8_1_1bit port map(
        A0 => A0(1), A1 => A1(1), A2 => A2(1),
        A3 => A3(1), A4 => A4(1), A5 => A5(1),
        A6 => A6(1), A7 => A7(1), C => C, Z => Z(1)
    );
    M2: mux8_1_1bit port map(
        A0 => A0(2), A1 => A1(2), A2 => A2(2),
        A3 => A3(2), A4 => A4(2), A5 => A5(2),
        A6 => A6(2), A7 => A7(2), C => C, Z => Z(2)
    );
    M3: mux8_1_1bit port map(
        A0 => A0(3), A1 => A1(3), A2 => A2(3),
        A3 => A3(3), A4 => A4(3), A5 => A5(3),
        A6 => A6(3), A7 => A7(3), C => C, Z => Z(3)
    );
end logicA;

```

```

library ieee;
use IEEE.STD_LOGIC_1164.all;

entity mux8_1_4bits is
    port (
        AB: in std_logic_vector(3 downto 0);
        A1: in std_logic_vector(3 downto 0);
        A2: in std_logic_vector(3 downto 0);
        A3: in std_logic_vector(3 downto 0);
        A4: in std_logic_vector(3 downto 0);
        A5: in std_logic_vector(3 downto 0);
        A6: in std_logic_vector(3 downto 0);
        A7: in std_logic_vector(3 downto 0);
        C: in std_logic_vector(2 downto 0);
        Z: out std_logic_vector(3 downto 0)
    );
end mux8_1_4bits;

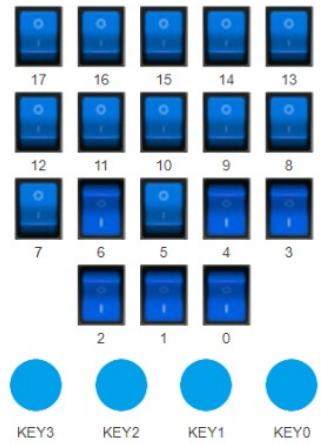
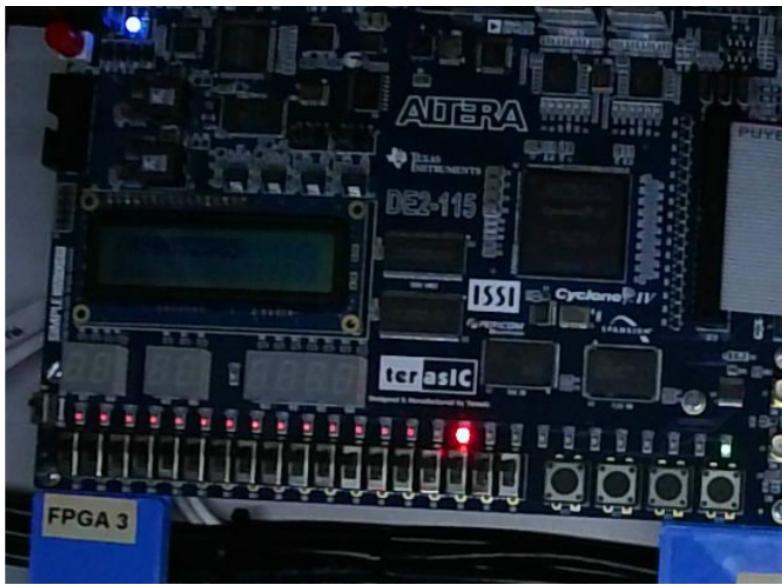
architecture logicA of mux8_1_4bits is
    component mux8_1_1bit is
        port (
            A0: in std_logic;
            A1: in std_logic;
            A2: in std_logic;
            A3: in std_logic;
            A4: in std_logic;
            A5: in std_logic;
            A6: in std_logic;
            A7: in std_logic;
            C: in std_logic;
            Z: out std_logic
        );
    end component;
begin
    M0: mux8_1_1bit port map(
        A0 => AB(0), A1 => A1(0), A2 => A2(0),
        A3 => A3(0), A4 => A4(0), A5 => A5(0),
        A6 => A6(0), A7 => A7(0), C => C, Z => Z(0)
    );
    M1: mux8_1_1bit port map(
        A0 => A0(1), A1 => A1(1), A2 => A2(1),
        A3 => A3(1), A4 => A4(1), A5 => A5(1),
        A6 => A6(1), A7 => A7(1), C => C, Z => Z(1)
    );
    M2: mux8_1_1bit port map(
        A0 => A0(2), A1 => A1(2), A2 => A2(2),
        A3 => A3(2), A4 => A4(2), A5 => A5(2),
        A6 => A6(2), A7 => A7(2), C => C, Z => Z(2)
    );
    M3: mux8_1_1bit port map(
        A0 => A0(3), A1 => A1(3), A2 => A2(3),
        A3 => A3(3), A4 => A4(3), A5 => A5(3),
        A6 => A6(3), A7 => A7(3), C => C, Z => Z(3)
    );
end logicA;

```

A apresentação dos resultados será dada a partir das saídas visuais simples da placa FPGA, no caso, alguns leds representarão o vetor z de saída.

5.2.2 Exemplo de Algumas operações

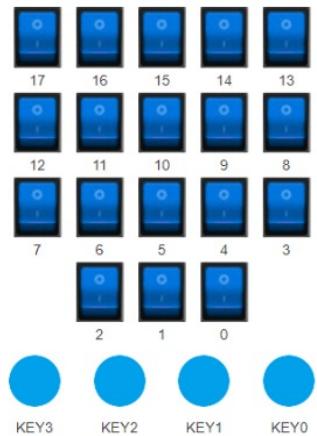
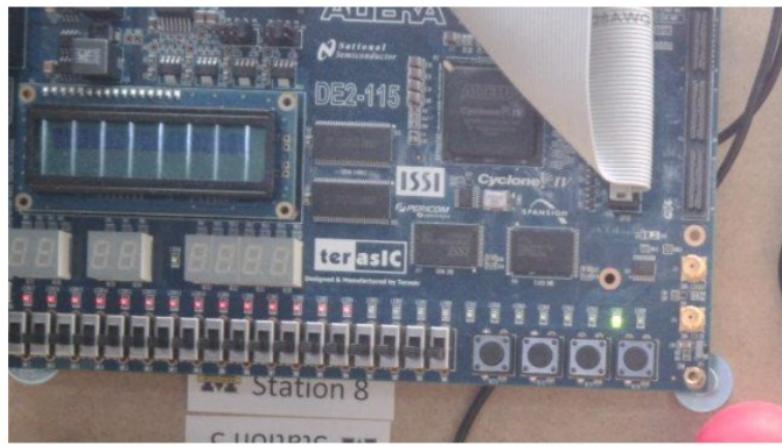
Soma:



A: 1111

B: 0101

Z: 0100

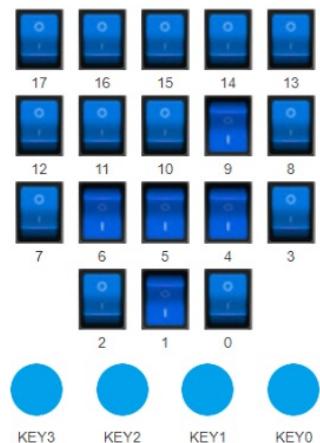
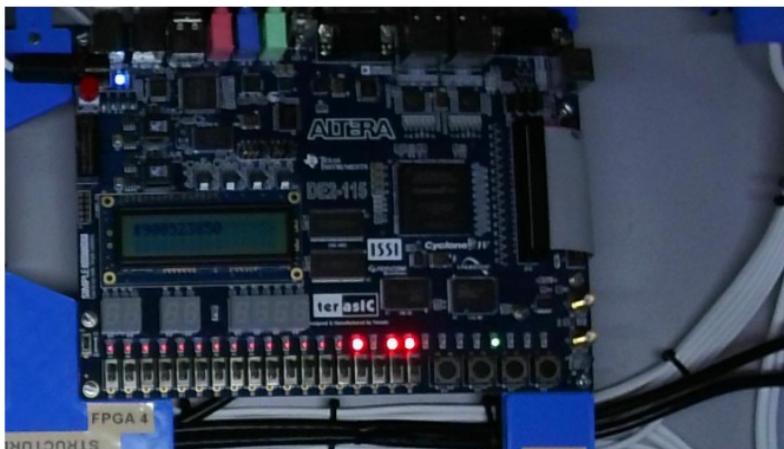


A: 0000

B: 0000

Z: 0000 (Visualização da flag de 0)

Subtração:



A: 0010

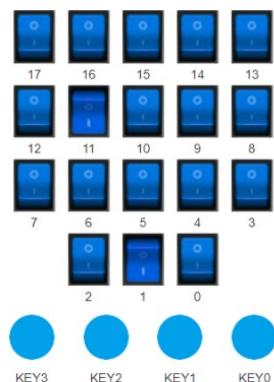
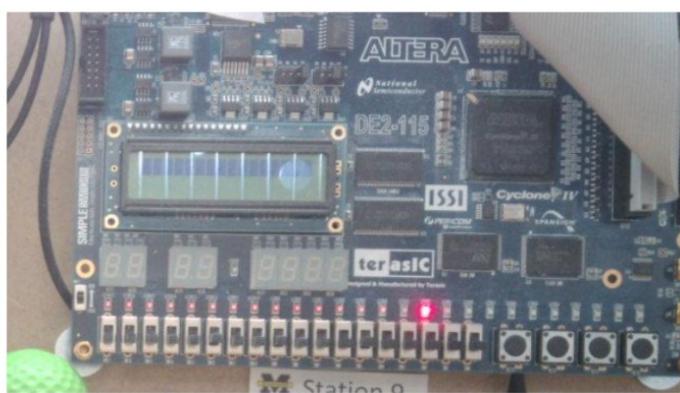
B: 0111

Z: 1011 (Visualização da flag de número negativo)

Deslocamento para Esquerda:



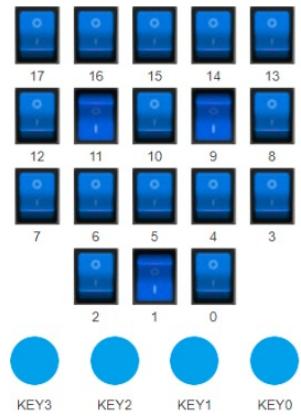
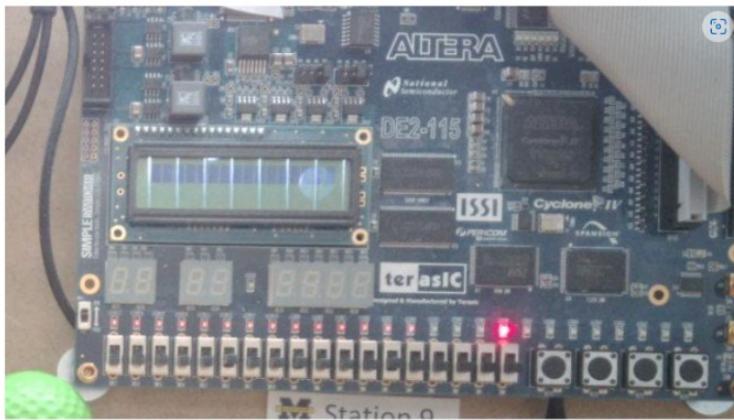
Laboratório Intel FPGA



A: 0010

Z: 0100

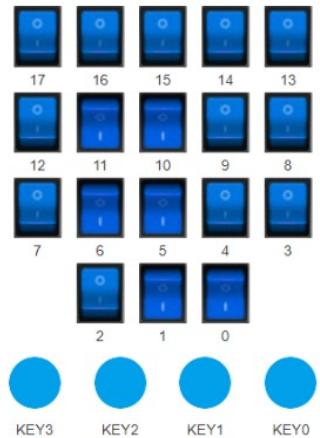
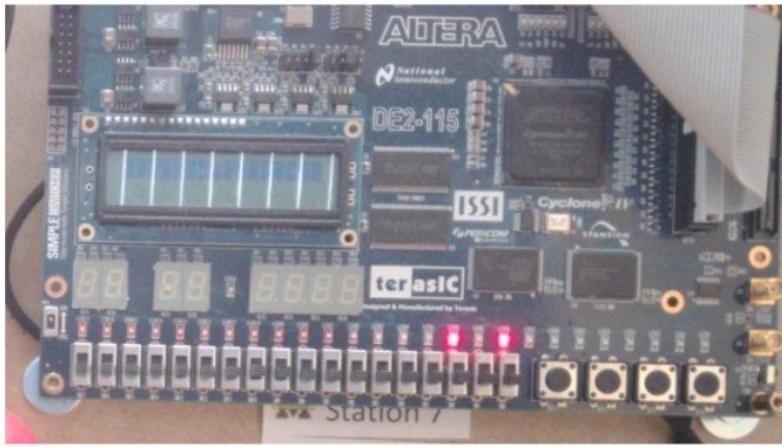
Deslocamento para Direita:



A: 0010

Z: 0001

Operação XOR:

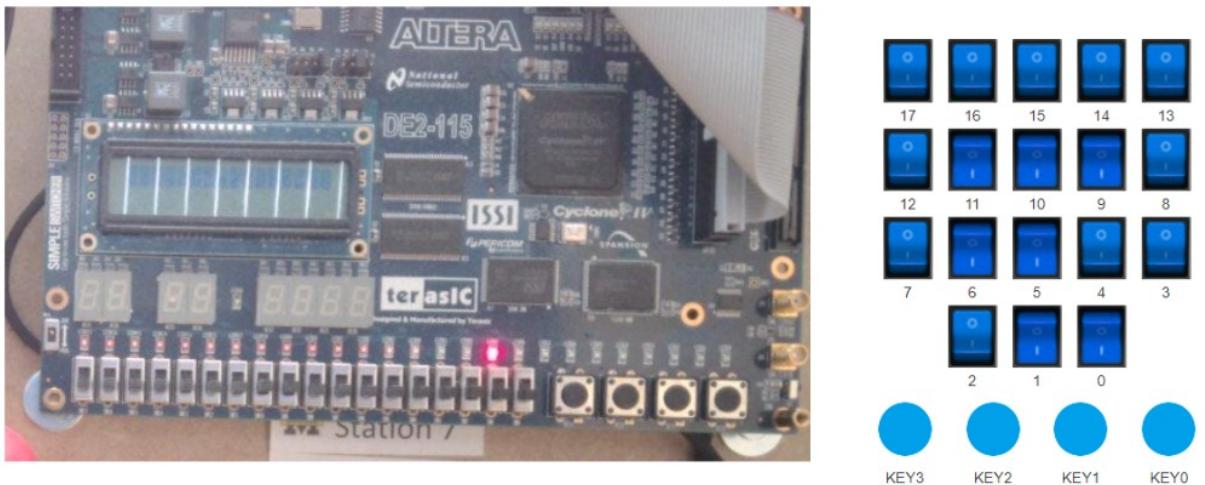


A:0110

B:0011

Z:0101

Operação AND:



A:0110

B:0011

Z:0010

6. Conclusão

Desta forma, foi alcançado o objetivo deste trabalho: desenvolver uma Unidade-Lógico-Aritmética (ULA) de 4 bits com 8 operações distintas que conta com interface com o usuário, e implementada na placa de desenvolvimento em laboratório.

Repositório no GitHub: <https://github.com/Raiane01/Sistemas-Digitais-ULA>

[caiopgaldino/Unidade-Logica-Aritmetica](https://github.com/caiopgaldino/Unidade-Logica-Aritmetica)

[Este repositório é dedicado ao primeiro trabalho de Sistemas Digitais de 2022.1
\(github.com\)](https://github.com/caiopgaldino/Unidade-Logica-Aritmetica)