UFRJ
UNIVERSIDADE FEDERAL
DO RIO DE JANEIRO

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

RELATÓRIO DE SISTEMAS DIGITAIS
ULA (UNIDADE LÓGICA-ARITMÉTICA)

DHAYSE DE LIMA TITO DRE: 120019062
JÚLIA PAOLI DE ANDRADE DRE: 120022049
LÍGIA CALINA BUENO BONIFÁCIO DRE: 122046065
TURMA EL2

Rio de Janeiro-RJ
2023

SUMÁRIO

1 INTRODUÇÃO

A crescente demanda por sistemas computacionais cada vez mais eficientes e rápidos tem impulsionado a busca por soluções otimizadas de hardware. Nesse contexto, a ULA, acrônimo para Unidade Lógico Aritmética, desempenha um papel crucial, sendo responsável por executar operações lógicas e aritméticas em números representados em circuitos lógicos.

Este trabalho tem como objetivo apresentar o desenvolvimento de uma ULA utilizando a linguagem VHDL (VHSIC Hardware Description Language) e sua implementação em uma FPGA (Field-Programmable Gate Array). A VHDL é uma linguagem de descrição de hardware amplamente utilizada para modelar e simular circuitos digitais complexos. Por sua vez, a FPGA oferece um ambiente flexível para a implementação de circuitos personalizados, permitindo que o projeto da ULA seja adaptado às necessidades específicas.

A construção de uma ULA envolve a combinação de dois fundamentos principais: o controle de fluxo de dados e a implementação de circuitos para operações lógicas e aritméticas. O controle de fluxo de dados é responsável por direcionar o caminho que os dados percorrerão durante as operações, garantindo a correta execução das instruções. Já a implementação dos circuitos envolve o desenvolvimento de estruturas lógicas que executarão as operações desejadas, como adição, subtração, multiplicação, entre outras.

A utilização da linguagem VHDL proporciona uma abordagem estruturada para descrever o comportamento e a funcionalidade da ULA, permitindo uma representação clara e concisa das operações a serem realizadas. Além disso, a FPGA possibilita a configuração do hardware em campo, tornando o projeto da ULA reconfigurável e facilitando possíveis ajustes e melhorias.

Ao longo deste trabalho, serão apresentados os principais conceitos teóricos relacionados à construção de uma ULA, bem como a metodologia utilizada para o desenvolvimento do projeto em VHDL. Serão exploradas as etapas de modelagem, simulação e síntese, culminando na implementação final da ULA na FPGA. Espera-se que essa abordagem proporcione um melhor entendimento sobre a construção de uma ULA e suas aplicações práticas, demonstrando a importância dessa unidade no contexto de sistemas digitais avançados.

2 IMPLEMENTAÇÃO

Nesta secção discorreremos sobre o código em VHDL utilizado para a implementação do trabalho. Todos os códigos estão disponíveis na íntegra no apêndice deste relatório.

Para simplificar a explicação, dividimos o texto em 2 grandes tópicos: “ALU” e “Interface com o usuário”. No primeiro falaremos sobre a ALU e suas funções, já no segundo falaremos da parte da lógica referente a coleta e entrega de dados na interação com o usuário, além da máquina de estados usada para isso.

2.1 ALU

Decidimos definir nossa ULA utilizando 10 arquivos. Isto inclui um para cada função lógica ou aritmética que fizemos e um para criarmos a entidade da ULA. Neste último importarmos como componentes todas as entidades criadas nos outros arquivos, para que elas pudessem todas serem usadas como funções da ULA.

Além das funções e da ULA me si, também foi criado um arquivo para implementar o somador de 1 bit, que serviu de base para a lógica aritmética.

Como podemos observar a entidade ModuleULA possui como entrada dois vetores de 4 bits, que serão os números utilizados nas operações, e um vetor de 3 bits que servirá de seleção. Sua única saída é o resultado da operação escolhida.

```
32 entity ModuleALU is
33     Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
34            B : in  STD_LOGIC_VECTOR (3 downto 0);
35            Z : out  STD_LOGIC_VECTOR (3 downto 0);
36            S : in  STD_LOGIC_VECTOR (2 downto 0));
37 end ModuleALU;
```

Após importar os componentes usados e criarmos os sinais necessários, calculamos todos os resultados possíveis.

```
94                 MY_AND: ModuleAND port map (A,B,Z0);
95                 MY_OR: ModuleOR port map (A,B,Z1);
96                 MY_NOT: ModuleNOT port map (A,Z2);
97                 MY_XOR: ModuleXOR port map (A,B,Z3);
98                 MY_SOMADOR_MODULE_4BIT: MY_SOMADOR_4BIT port map
                   ↪  (A,B,'0',Z4,Cout);
```

```
99                  MY_SUBTRATOR_MODULE_4BIT: MY_SUBTRATOR_4BIT port map
                    ↪ (A,B,Z5,Bout);
100                 MY_MULTIPLICADOR_MODULE: MY_MULTIPLICADOR port map
                    ↪ (A,B,Z6);
101                 MY_C2_MODULE : MY_C2 port map (A,Z7);
```

Em seguida criamos um *process* com uma sequência de *if's* para decidir qual resultado atribuiremos para Z conforme o valor de S.

Agora veremos como cada função logica ou aritmética foi implementada em seu próprio arquivo.

2.1.1 Função AND

A primeira função lógica a ser implementada é a operação AND. Esta função é realizada através de uma operação bit a bit entre o número A e o número B, ambos vetores de 4 bits. O código a seguir ilustra a implementação dessa função:
A lógica da função está abaixo e seu código referenciado no Apêndice B.1.

$Z = A\dot{B}$

Nesse código, o operador “AND” é utilizado para realizar a operação AND entre os bits correspondentes de A e B. O resultado será armazenado na variável “Z”, também um vetor de 4 bits.

2.1.2 Função OR

A função lógica explicada nessa seção é a operação OR. A implementação foi baseada em uma operação bit a bit entre o vetor A e B, ambos com 4 bits. Sendo o resultado armazenado num vetor "Z", também com 4 bits.

A lógica da função está abaixo e seu código referenciado no Apêndice B.2.

$Z = A + B$

2.1.3 Função NOT

A implementação lógica da função NOT, como podem ver, é bem simples. Consiste em somente uma entrada e uma saída, ambas de 4 bits. Nessa arquitetura a saída é definida como um not bit a bit da entrada.
A lógica da função está abaixo e seu código referenciado no Apêndice B.3.

$Z = A'$

2.1.4 Função XOR

Para a função XOR, a implementação é semelhante. Os números A e B são submetidos à operação XOR bit a bit, como ilustrado no código a seguir:

A lógica da função está abaixo e seu código referenciado no Apêndice B.2.

$Z = A \oplus B$

2.1.5 Somador de 1 bit

A implementação de um somador 1 bit foi necessárias para a implementação posterior do somador de 4 bits. Para isso, foram utilizados 4 sinais de entrada: X, Y e C_{In}; e as saída: Z e C_{Out}.

Essa implementação segue a lógica básica de um *Full Adder*. Seu código está referenciado no Apêndice C.1.

$Z = C_{in} \oplus X \oplus Y$

$C_{Out} = X.Y + C_{in}.Y + C_{in}.X$ ou $C_{Out} = XY + C_{in}(X \oplus Y)$

2.1.6 Somador de 4 bits

Nessa implementação foi utilizado, para simplificação, 4 vezes o somador de 1 bit. De modo que o C_{Out} foi passado para o C_{In} do bit seguinte até o penúltimo bit.

Seu código está referenciado no Apêndice C.2. Os vetores de 4 bits de entrada são representados por X e Y e de saída Z. Além dos *carrys* de 1 bit C_{In} e C_{Out}.

2.1.7 Subtrator de 4 bits

Para implementação da lógica do subtrator, realizamos, na realidade, uma soma de X com o complemento de 2 de Y. Para isso, criamos o complemento de 1 de Y (C1y) e, em seguida, conectamos X e C1y no somador de 4 bits com C_{In} = '1'. Isso significa:

$$Z + C_{out} = X + (C1y + 1)$$

$$Z + C_{out} = X + C2y$$

Além disso, também definimos Bout como not C_{Out}. Seu código está referenciado no Apêndice C.3.

2.1.8 Multiplicador de 4 bits

Nesta seção, apresentaremos a implementação da operação de multiplicação na Unidade Lógica e Aritmética. Seu código está referenciado no Apêndice C.4, referente à entidade *MY MULTIPLICADOR*, ilustra como essa operação é realizada.

Nesse código, a entidade *MY MULTIPLICADOR* define as portas de entrada X e Y, que são vetores de 4 bits, e a porta de saída Z, também um vetor de 4 bits. A arquitetura Behavioral contém a implementação da multiplicação, onde são utilizados componentes chamados *MY SOMADOR 1BIT*, que representam somadores de 1 bit, segundo as seguintes equações:

$$S + C_{OutParcial} = (X * Y) + C_{in}$$

Para o caso de 3 Bits ou mais:

$$Z + C_{Out} = (S_1 + C_{OutParcial}) + (S_2 + C_{OutParcial'})$$

Cada bloco dentro da implementação realiza uma multiplicação parcial, onde bits correspondentes dos vetores X e Y são multiplicados e passam pelos somadores 1-bit para obter os resultados parciais e os carry-outs correspondentes. Os resultados finais da multiplicação são atribuídos ao vetor de saída Z.

2.1.9 Complemento de 2

A implementação dessa lógica é bem simples. Existe um vetor de entrada "X"e um vetor de saída "Z". Para simplificação desta foi utilizado a lógica do somador de 4 bits com a entrada X do somador sendo "NOT X".

Assim, o vetor Z é dado pela função abaixo e seu código referenciado no Apêndice C.5.

$$Z + C_{Out} = X' + < 0001 >$$

2.2 INTERFACE COM O USUÁRIO

Para que seja possível a utilização da ULA, criamos uma interface com o usuário. Ela é dada no código como a entidade ModuloInterface, a qual possui as seguintes portas de entrada e saída:

```
32 entity ModuloInterface is
33     port (
34         clk : in std_logic;
35         BOTAO_A : in std_logic;
```

```
36          BOTAO_B : in std_logic;
37          BOTAO_S : in std_logic;
38          BOTAO_RST : in std_logic;
39          SW : in std_logic_vector (3 downto 0);
40          LED_A : out std_logic;
41          LED_B : out std_logic;
42          LED_S : out std_logic;
43          LEDS: out std_logic_vector (3 downto 0)
44      );
45  end ModuloInterface;
```

- clk: sinal de clock utilizado para sincronização dos elementos.
- BOTAO A, BOTAO B, BOTAO S, BOTAO RST: sinais de entrada que representam os botões de controle A, B, S e Reset, respectivamente.
- SW: vetor de sinal de 4 bits que representa os valores dos interruptores.
- LED A, LED B, LED S: sinais de saída que controlam os LEDs indicadores para as variáveis A, B e S, respectivamente.
- LEDS: vetor de sinal de 4 bits que controla os LEDs indicadores gerais.

A entidade utiliza dois componentes: ModuleALU e redutorClock. Referenciados, respectivamente, nos Apêndices D.1 e D.2.

O componente ModuleALU representa a ALU e realiza as operações lógicas e aritméticas com base nos sinais de entrada A, B e S, produzindo o resultado Z, como já explicado na seção 2.1. O componente redutorClock é responsável pela redução da frequência do sinal de clock para sincronização adequada dos elementos. Ele será melhor explicado na seção 2.2.2.

2.2.1 Máquina de estados

Foi utilizado um sistema de Máquina de Estados para realizar o controle do fluxo de dados da nossa ULA. Ela está representada de forma simplificada pela imagem a seguir.

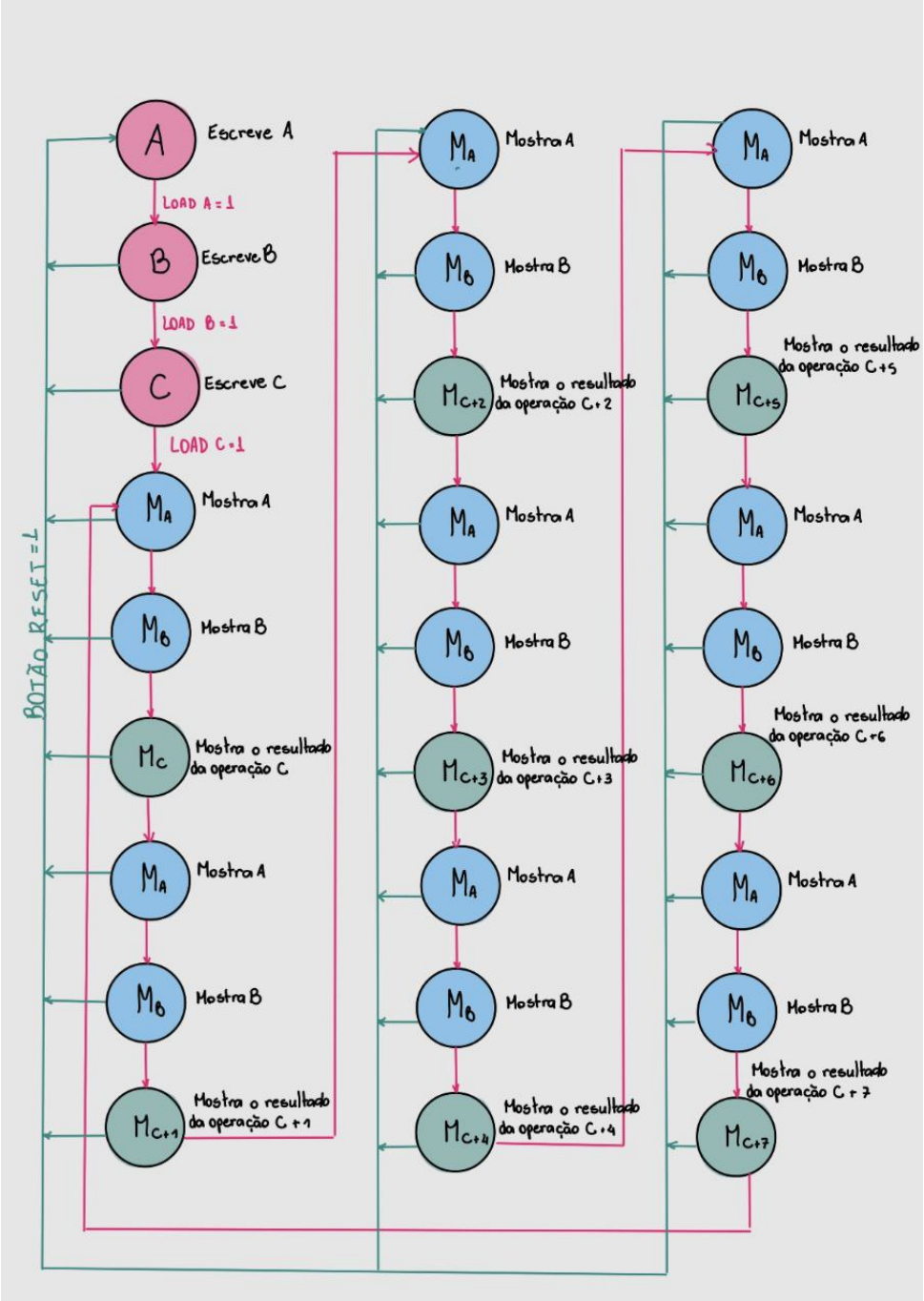

Figura 1 – Implementação em máquina de estados. **A notação "C+1"refere-se a ideia de próximo estado, e não da soma literal, uma vez que no estado referente a ultima operação, ele em seguida retornará para o estado da primeira.

O código utiliza o tipo enumerado estado para definir os estados possíveis do sistema. Os sinais estadoAtual e estadoAux são utilizados para controlar a transição entre os estados.

```
63      type estado is (E0,E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,E12,E13,E14,
        ↪  E15,E16,E17,E18,E19,E20,E21,E22,E23,E24,E25,E26,e27);
64          -- criando os estados
```

```
65      signal estadoAtual, estadoAux: estado := E0; -- criando o estado
        ↪  oficial e uma auxiliar
66      signal A,B,S_inicial: std_logic_vector (3 downto 0); -- sinais do
        ↪  numero A, B e a selecao S recebida nos SW
67      signal Z0,Z1,Z2,Z3,Z4,Z5,Z6,Z7 : std_logic_vector (3 downto 0); --
        ↪  sinais de saida
68      signal S_int : integer;
69           signal clk_novo : std_logic;
70
```

Cada uma das operações S possui um número em binário referente a ela - indo de 000 a 111 - para que seja possível identificar qual o Estado de operação.

```
71  begin
72      BLOCO_CLK : redutorClock port map (clk,clk_novo); -- mandamos o clk
        ↪  da placo e pegamos um reduzido que eh o q vamos usar
73      BLOCO_ALU0: ModuleALU port map(A,B,Z0,"000"); -- conectando as
        ↪  entradas e saidas da ALU para S = 0
74      BLOCO_ALU1: ModuleALU port map(A,B,Z1,"001"); -- conectando as
        ↪  entradas e saidas da ALU para S = 1
75      BLOCO_ALU2: ModuleALU port map(A,B,Z2,"010"); -- conectando as
        ↪  entradas e saidas da ALU para S = 2
76      BLOCO_ALU3: ModuleALU port map(A,B,Z3,"011"); -- conectando as
        ↪  entradas e saidas da ALU para S = 3
77      BLOCO_ALU4: ModuleALU port map(A,B,Z4,"100"); -- conectando as
        ↪  entradas e saidas da ALU para S = 4
78      BLOCO_ALU5: ModuleALU port map(A,B,Z5,"101"); -- conectando as
        ↪  entradas e saidas da ALU para S = 5
79      BLOCO_ALU6: ModuleALU port map(A,B,Z6,"110"); -- conectando as
        ↪  entradas e saidas da ALU para S = 6
80      BLOCO_ALU7: ModuleALU port map(A,B,Z7,"111"); -- conectando as
        ↪  entradas e saidas da ALU para S = 7
81
```

A lógica do código é definida pelos processos. O primeiro processo, sensível aos sinais clk novo e estadoAux, atualiza o estado atual (estadoAtual) com base no estado auxiliar (estadoAux) quando ocorre uma borda de subida do sinal de clock (clk novo).

```
83      process(clk_novo, estadoAux) -- logica do estado auxiliar
84      begin
```

```
85          if(clk_novo'event and clk_novo='1') then
86              estadoAtual <= estadoAux;  -- alteramos o estado no clock
87          end if;
88      end process;
89
```

O segundo processo, sensível aos sinais BOTAO A, BOTAO B, BOTAO S, BOTAO RST, SW e estadoAtual, implementa a lógica principal da interface com o usuário. Ele controla os LEDs indicadores, captura os valores dos interruptores e realiza a transição entre os estados com base nos botões pressionados.

Cada estado é definido por um bloco *when* no qual são realizadas as ações correspondentes. Por exemplo, no estado E0, o LED A é aceso e se o botão BOTAO A for pressionado, o valor dos interruptores é atribuído à variável A, o LED A é apagado e o estado é alterado para E1.

```
90      process(BOTAO_A,BOTAO_B, BOTAO_S,BOTAO_RST, SW,estadoAtual) -- agora
        ↪  a logica da maquina de estados
91      begin
92                  report "Iniciou";
93          if (BOTAO_RST = '1') then -- caso tenhamos RST, voltamos para o
            ↪  primeiro estado
94              estadoAux <= E0;
95              LEDS <= "0000";   -- apaga os LEDS
96                                  report "Resetou";
97          else                    -- resto da logica acontece se nao
            ↪  tivermos o reset
98              case estadoAtual is
99                  when E0 =>       -- E0 -> pegar o numero A
                    ↪  ---------------------------------------
100                     LED_A <= '1';      -- acende o led q sinaliza q
                        ↪  estamos pegando o primeiro num
101                     LED_B <= '0';
102                     LED_S <= '0';
103
104                     if(BOTAO_A = '1') then  -- gravamos oq esta nos
                        ↪  switchs quando o botao é apertado
105                         A <= SW;
106                         LED_A <= '0';      -- apaga o led
107                         estadoAux<=E1;     -- mudamos de estado
108                     else
```

```
109                     estadoAux<=E0;  -- se o batao nao for apertado
                        ↪  cont no mesmo estado
110                 end if;
111             when E1 => -- E1 -> pegar o numero B
                ↪  ---------------------------------------
112                 LED_B <= '1';       -- acende o led q sinaliza q
                    ↪  estamos pegando o segundo num
113                 LED_A <= '0';
114                 LED_S <= '0';
115
116                 if(BOTAO_B = '1') then  -- gravamos oq esta nos
                    ↪  switchs quando o botao é apertado
117                     B <= SW;
118                     LED_B <= '0';       -- apaga o led
119                     estadoAux<=E2;      -- mudamos de estado
120                 else
121                     estadoAux<=E1;  -- se o batao nao for apertado
                        ↪  cont no mesmo estado
122                 end if;
```

Essa lógica é repetida para os demais estados, controlando os LEDs indicadores e capturando os valores dos interruptores para as variáveis B e S.

Em resumo, o código implementa a lógica da interface com o usuário para a ALU, permitindo ao usuário inserir os valores de A, B e S por meio de botões e interruptores, e visualizar o resultado por meio dos LEDs indicadores.

2.2.2 Redutor de Clock

Para realizar as mudanças de estado, utilizamos o clock interno da placa. Porém, este tem frequência de 50MHz, sendo assim muito rápido e impróprio para interação com o usuário. Como o enunciado do trabalho pedia um período de 2 segundo na exibição de cada valor, produzimos um “novo” clock através do original da placa. Nessa

entidade, a entrada é o clock da placa e a saída é o clock com período de 2 segundos. Produzimos o último com a ajuda de um contador que vai até 100 milhões e soma um toda vez que o clock da placa dá um pulso. Definimos o clock de saída como zero ou um dependendo do valor do contador.

O código dessa implementação está referenciado no Apêndice D.2.

3 SIMULAÇÕES

Através da simulação, buscamos validar o funcionamento correto da ALU e garantir a precisão dos resultados obtidos. Nesta seção, apresentaremos os detalhes da simulação, incluindo os resultados obtidos. Analisaremos os gráficos e valores gerados durante as simulações, visando verificar o desempenho e a coerência da ALU em relação às operações realizadas.

3.0.1 Função AND

Ao aplicar a função AND nos números binários "1010"e "1110", o resultado esperado é "1010". A simulação confirmou que a saída da função AND foi de fato "1010", indicando que a saída é alta (1) apenas quando ambos os bits correspondentes nas entradas são 1. Foram colocados também os valores "1010"e "1100", resultando em "1000". Isto foi feito para mostrar o respeito da função ao pulso de clock.

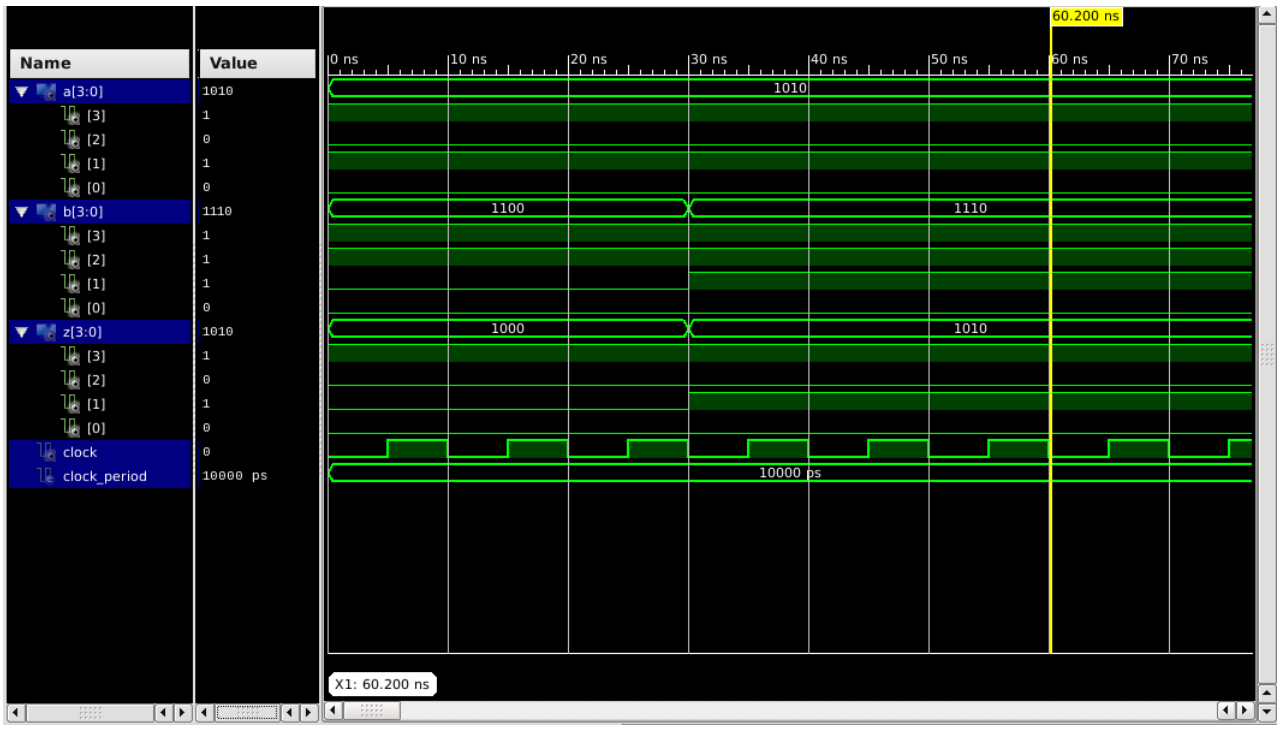

Figura 2 – Imagem referente a simulação da função AND na ALU.

3.0.2 Função OR

Aplicando a função OR nos números binários "1010"e "1110", esperamos obter o resultado "1110". A simulação validou essa expectativa, mostrando que a saída da função OR foi de fato "1100", indicando que a saída é alta (1) quando pelo menos um dos bits correspondentes nas entradas é 1. Foram colocados também os valores "1010"e "1100", resultando em "1110". Isto foi feito para mostrar o respeito da função ao pulso de clock.

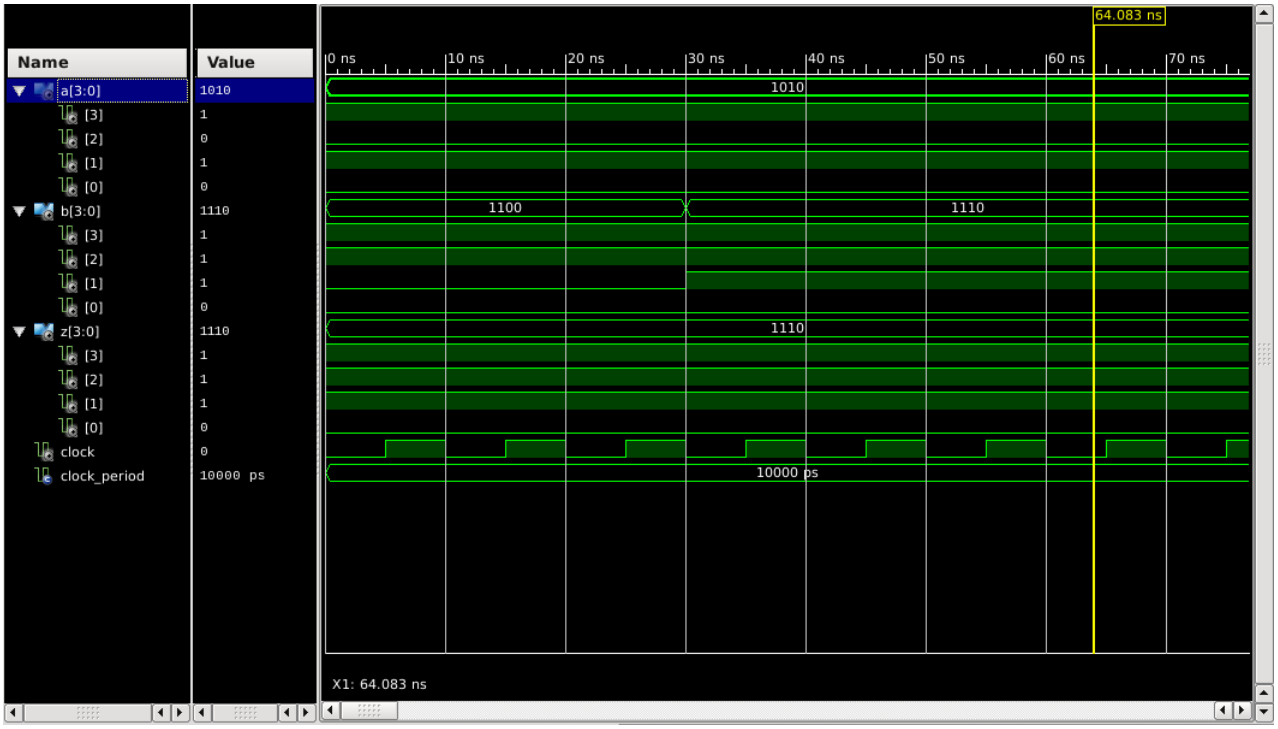

Figura 3 – Imagem referente a simulação da função OR na ALU.

3.0.3 Função NOT

A função NOT foi aplicada ao número binário "1100". O resultado esperado é o inverso do bit de entrada, ou seja, "0011". A simulação confirmou essa previsão, mostrando que a saída da função NOT foi de fato "0011", invertendo os bits do número de entrada. Foi colocado também o valor "1010", resultando em "0101". Isto foi feito para mostrar o respeito da função ao pulso de clock.

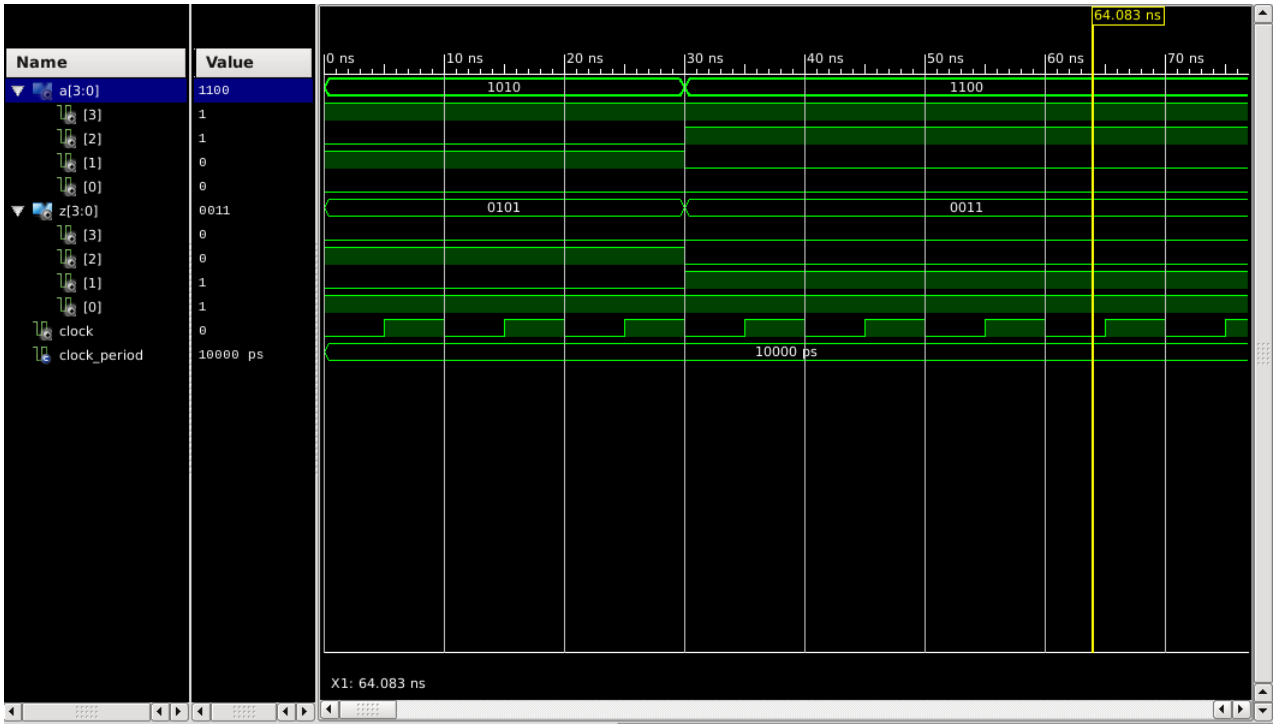

Figura 4 – Imagem referente a simulação da função NOT na ALU.

3.0.4 Função XOR

Para a função XOR, aplicamos a operação nos números binários "1010"e "1110". O resultado esperado é "0100". A simulação validou essa expectativa, mostrando que a saída da função XOR foi de fato "0100", indicando que a saída é alta (1) quando os bits correspondentes nas entradas são diferentes. Foram colocados também os valores "1010"e "1100", resultando em "0110". Isto foi feito para mostrar o respeito da função ao pulso de clock.

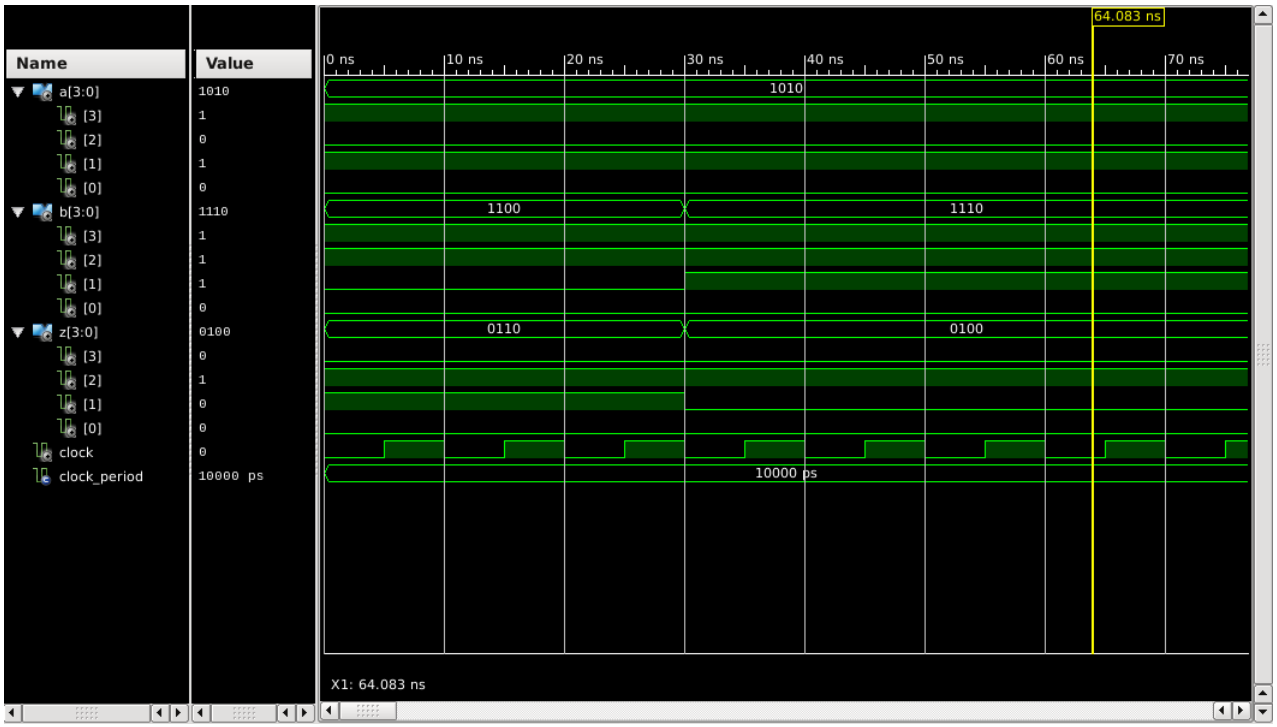

Figura 5 – Imagem referente a simulação da função XOR na ALU.

3.0.5 Função SOMADOR

Na simulação do somador com os números binários "1010"e "1100", esperamos obter o resultado da soma correta, que é "0110", pois não é possível escrever o resultado completo em apenas 4 bits. Além disso, esperamos que o bit de carry seja ativado (CIN = 1). A simulação confirmou que a saída do somador foi de fato "0110"e que o bit de carry foi ativado corretamente quando necessário. Foram colocados também os valores "1010"e "1110", resultando em "1000". Isto foi feito para mostrar o respeito da função ao pulso de clock.

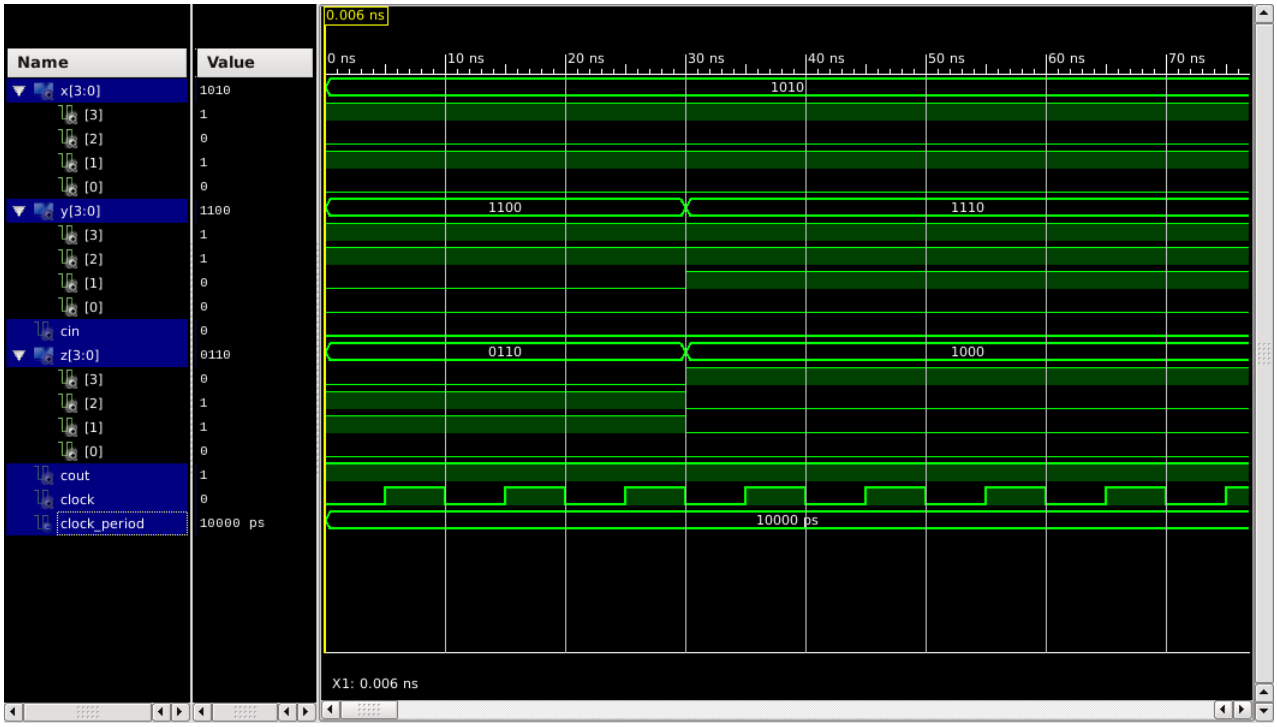

Figura 6 – Imagem referente a simulação da função SOMADOR na ALU.

3.0.6 Função SUBTRATOR

Ao simular o subtrator com os números binários "1010"e "1110", esperamos obter o resultado da subtração correta, que é "1100". Vale ressaltar que é feito o complemento de 2 do número Y que está sendo subtraído. Também esperamos que o bit de borrow de saída seja ativado (BOUT=1). A simulação confirmou que a saída do subtrator foi de fato "1110"e que o bit de borrow foi ativado corretamente quando necessário. Foram colocados também os valores "1010"e "1100", resultando em "1110". Isto foi feito para mostrar o respeito da função ao pulso de clock.

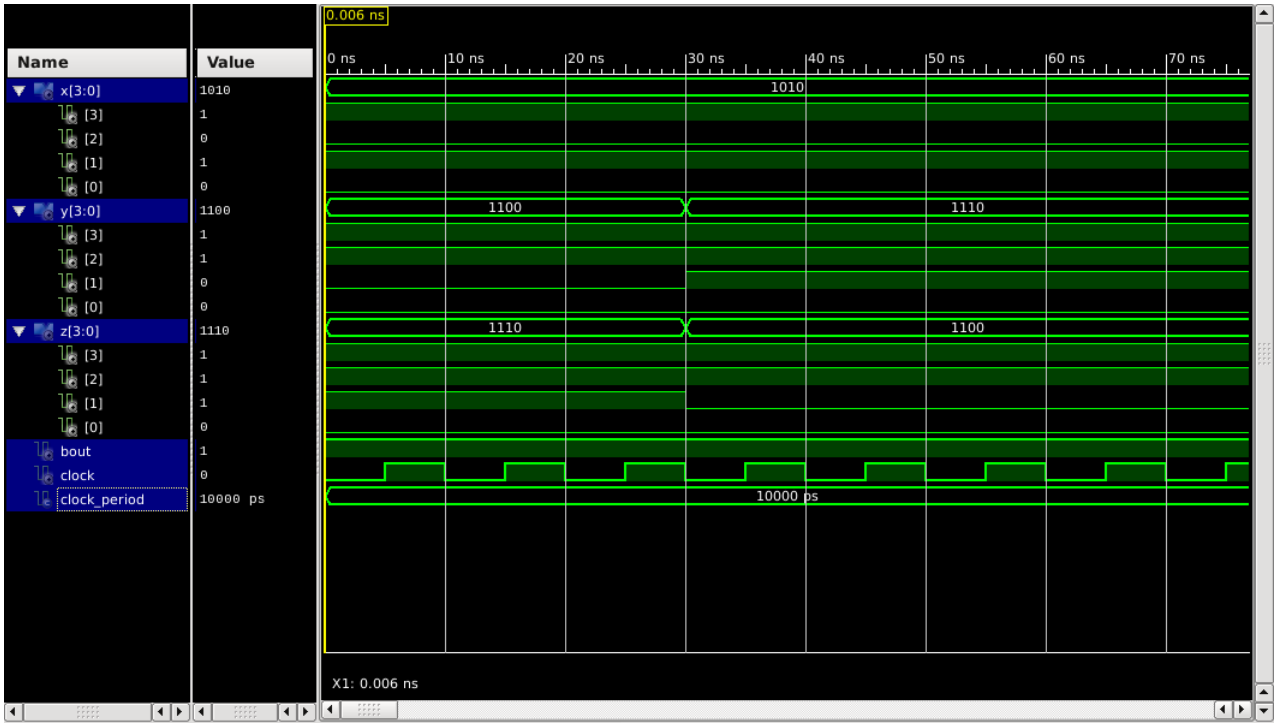

Figura 7 – Imagem referente a simulação da função SUBTRATOR na ALU.

3.0.7 Função MULTIPLICADOR

No caso do multiplicador com os números binários "1010"e "1100", esperamos obter o resultado correto da multiplicação, que é "1111000", porém, como temos somente 4 bits para representar este resultado, a saída é "1000", os quais referem-se aos 4 bits menos significativos. A simulação validou essa expectativa, mostrando que a saída do multiplicador foi de fato "1000", correspondendo ao resultado esperado. Foram colocados também os valores "1010"e "1110", resultando em "1100". Isto foi feito para mostrar o respeito da função ao pulso de clock.

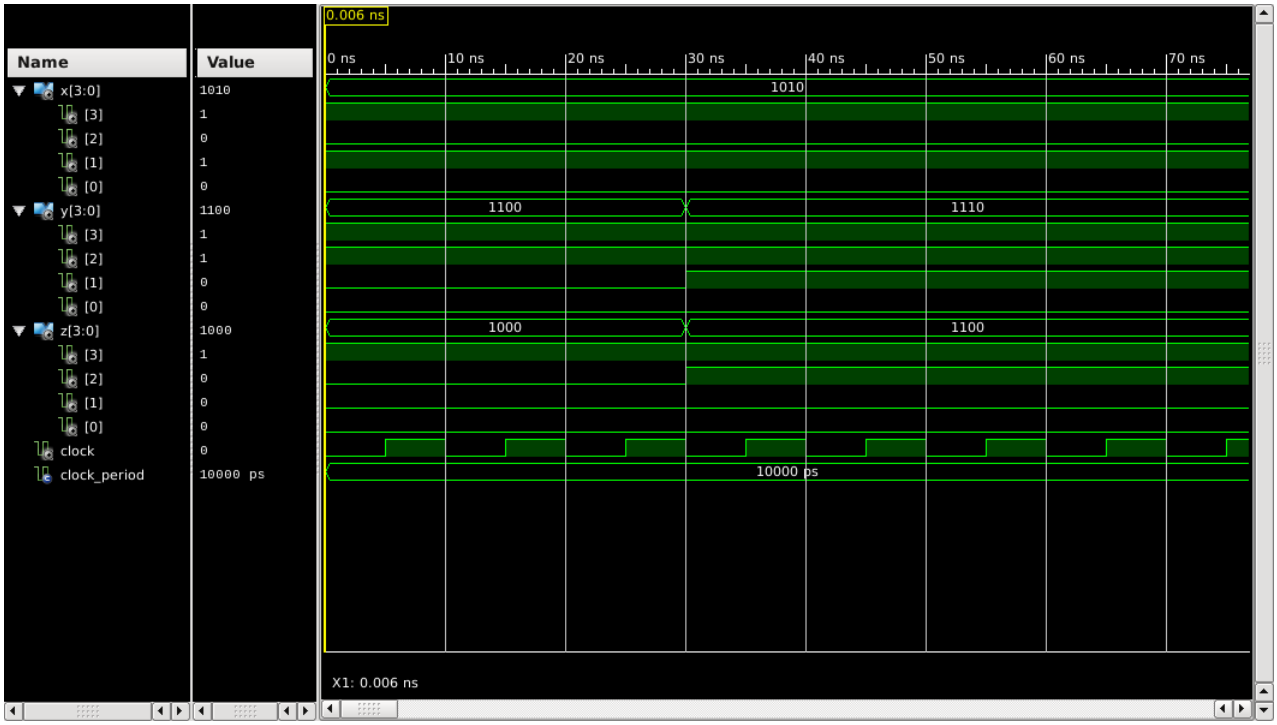

Figura 8 – Imagem referente a simulação da função MULTIPLICADOR na ALU.

3.0.8 Função COMPLEMENTO DE 2

Na simulação do complemento de 2 com o número binário "1010", esperamos obter o complemento de 2 correto, que é "0110". A simulação confirmou que a saída do complemento de 2 foi de fato "0110", invertendo os bits de entrada e adicionando 1 ao resultado. Foram colocados também os valores "1100", resultando em "0100". Isto foi feito para mostrar o respeito da função ao pulso de clock.

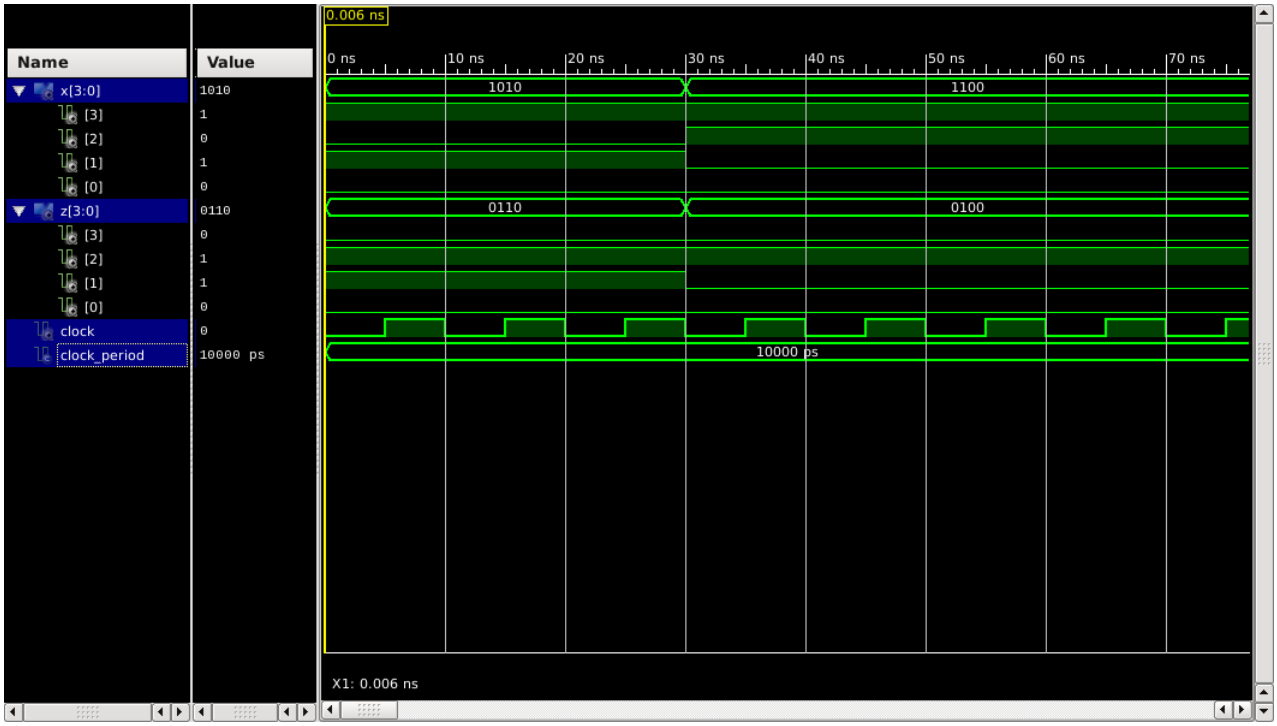

Figura 9 – Imagem referente a simulação da função Complemento de 2 na ALU.

Essa análise detalhada dos resultados obtidos durante a simulação das funções lógicas e aritméticas demonstra a validade e a precisão da implementação da ALU em relação às operações realizadas.

4 CONCLUSÃO

Ao longo deste trabalho, dedicamos nossos esforços à criação de uma ULA (Unidade Lógica-Aritmética). Nosso objetivo era projetar e implementar uma ULA capaz de realizar as principais operações lógicas e aritméticas utilizando os conhecimentos adquiridos nas aulas teóricas da disciplina.

Durante o processo de desenvolvimento, enfrentamos alguns desafios e tomamos decisões que foram de suma importância para o projeto, principalmente no operador aritmético multiplicador e na máquina de estados.

A implementação da máquina de estados apresentou um tamanho considerável e poderia ter sido otimizada de forma mais eficiente. No entanto, ao tentarmos implementações com um número menor de estados, não obtivemos os resultados esperados. Portanto, optamos por utilizar uma máquina com 27 estados, o que nos permitiu alcançar os resultados desejados.

Realizamos a modelagem e simulação de cada operador da ULA pelo software disponibilizado pelo professor. Implementamos as operações lógicas: AND, OR, XOR, NOT e as operações aritméticas SOMA, SUBTRAÇÃO, MULTIPLICAÇÃO E COMPLEMENTO DE 2. Além disso, implementamos a interface com o usuário usufruindo dos botões e LEDs que a placa possui. Por fim, para que fosse possível a implementação conforme os requisitos do trabalho foi feita uma máquina de estados.

Ao final desse trabalho, nossa ULA se mostrou capaz de realizar as operações solicitadas de forma precisa e eficiente. No entanto, reconhecemos que possuem pontos de melhorias, principalmente na máquina de estados.

Apêndices

APÊNDICE A – CÓDIGO ULA

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 -- Uncomment the following library declaration if using
24 -- arithmetic functions with Signed or Unsigned values
25 --use IEEE.NUMERIC_STD.ALL;
26
27 -- Uncomment the following library declaration if instantiating
28 -- any Xilinx primitives in this code.
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity ModuleALU is
33     Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
34            B : in  STD_LOGIC_VECTOR (3 downto 0);
35            Z : out  STD_LOGIC_VECTOR (3 downto 0);
36            S : in  STD_LOGIC_VECTOR (2 downto 0));
37 end ModuleALU;
38
39 architecture Behavioral of ModuleALU is
40
41         component ModuleAND is
42                 Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
43                        B : in  STD_LOGIC_VECTOR (3 downto 0);
44                        Z : out  STD_LOGIC_VECTOR (3 downto
                          ↪  0));
45         end component;
46
47         component ModuleOR is
48                 Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
49                        B : in  STD_LOGIC_VECTOR (3 downto 0);
50                        Z : out  STD_LOGIC_VECTOR (3 downto
                          ↪  0));
51         end component;
52
53         component ModuleNOT is
54                 Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
```

```
55                                  Z : out  STD_LOGIC_VECTOR (3 downto
                                    ↪  0));
56          end component;
57
58          component ModuleXOR is
59                  Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
60                                  B : in  STD_LOGIC_VECTOR (3 downto 0);
61                                  Z : out  STD_LOGIC_VECTOR (3 downto
                                    ↪  0));
62          end component;
63
64           component MY_SOMADOR_4BIT is
65      Port ( X : in  STD_LOGIC_VECTOR (3 downto 0);
66             Y : in  STD_LOGIC_VECTOR (3 downto 0);
67             Cin : in  STD_LOGIC;
68             Z : out   STD_LOGIC_VECTOR (3 downto 0);
69             Cout : out  STD_LOGIC);
70          end component;
71
72          component MY_SUBTRATOR_4BIT is
73      Port ( X : in  STD_LOGIC_VECTOR (3 downto 0);
74             Y : in  STD_LOGIC_VECTOR (3 downto 0);
75             Z : out  STD_LOGIC_VECTOR (3 downto 0);
76             Bout : out  STD_LOGIC);
77          end component;
78
79           component MY_MULTIPLICADOR is
80      Port ( X : in  STD_LOGIC_VECTOR (3 downto 0);
81             Y : in  STD_LOGIC_VECTOR (3 downto 0);
82             Z : out  STD_LOGIC_VECTOR (3 downto 0));
83          end component;
84
85          component MY_C2 is
86      Port ( X : in  STD_LOGIC_VECTOR (3 downto 0);
87             Z : out  STD_LOGIC_VECTOR (3 downto 0));
88          end component;
89
90          signal Z0,Z1,Z2,Z3,Z4,Z5,Z6,Z7: STD_LOGIC_VECTOR (3 downto 0);
91          signal Cout, Bout: STD_LOGIC;
```

```
92
93          begin
94                  MY_AND: ModuleAND port map (A,B,Z0);
95                  MY_OR: ModuleOR port map (A,B,Z1);
96                  MY_NOT: ModuleNOT port map (A,Z2);
97                  MY_XOR: ModuleXOR port map (A,B,Z3);
98                  MY_SOMADOR_MODULE_4BIT: MY_SOMADOR_4BIT port map
                    ↪  (A,B,'0',Z4,Cout);
99                  MY_SUBTRATOR_MODULE_4BIT: MY_SUBTRATOR_4BIT port map
                    ↪  (A,B,Z5,Bout);
100                 MY_MULTIPLICADOR_MODULE: MY_MULTIPLICADOR port map
                    ↪  (A,B,Z6);
101                 MY_C2_MODULE : MY_C2 port map (A,Z7);
102
103                 process
104                 begin
105                         if (S = "000") then
106                                 Z <= Z0;
107                         end if;
108                         if (S = "001") then
109                                 Z <= Z1;
110                         end if;
111                         if (S = "010") then
112                                 Z <= Z2;
113                         end if;
114                         if (S = "011") then
115                                 Z <= Z3;
116                         end if;
117                         if (S = "100") then
118                                 Z <= Z4;
119                         end if;
120                         if (S = "101") then
121                                 Z <= Z5;
122                         end if;
123                         if (S = "110") then
124                                 Z <= Z6;
125                         end if;
126                         if (S = "111") then
127                                 Z <= Z7;
```

```
128                         end if;
129                 end process;
130
131 end Behavioral;
```

APÊNDICE B – CÓDIGOS OPERADORES LÓGICOS

B.1 MODULE AND

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 -- Uncomment the following library declaration if using
24 -- arithmetic functions with Signed or Unsigned values
25 --use IEEE.NUMERIC_STD.ALL;
26
27 -- Uncomment the following library declaration if instantiating
28 -- any Xilinx primitives in this code.
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity ModuleAND is
33     Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
34            B : in  STD_LOGIC_VECTOR (3 downto 0);
35            Z : out  STD_LOGIC_VECTOR (3 downto 0));
36 end ModuleAND;
37
38 architecture Behavioral of ModuleAND is
39
40 begin
41         Z <= A AND B;
42
43 end Behavioral;
```

B.2 MODULE OR

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 -- Uncomment the following library declaration if using
24 -- arithmetic functions with Signed or Unsigned values
25 --use IEEE.NUMERIC_STD.ALL;
26
27 -- Uncomment the following library declaration if instantiating
28 -- any Xilinx primitives in this code.
```

```
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity ModuleOR is
33     Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
34            B : in  STD_LOGIC_VECTOR (3 downto 0);
35            Z : out  STD_LOGIC_VECTOR (3 downto 0));
36 end ModuleOR;
37
38 architecture Behavioral of ModuleOR is
39
40 begin
41
42         Z <= A OR B;
43
44 end Behavioral;
```

B.3 MODULE NOT

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 -- Uncomment the following library declaration if using
24 -- arithmetic functions with Signed or Unsigned values
25 --use IEEE.NUMERIC_STD.ALL;
26
27 -- Uncomment the following library declaration if instantiating
28 -- any Xilinx primitives in this code.
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity ModuleNOT is
33     Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
34            Z : out  STD_LOGIC_VECTOR (3 downto 0));
35 end ModuleNOT;
36
37 architecture Behavioral of ModuleNOT is
38
39 begin
40
```

```
41          Z <= NOT A;
42
43  end Behavioral;
```

B.4 MODULE XOR

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity ModuleXOR is
33      Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
34             B : in  STD_LOGIC_VECTOR (3 downto 0);
35             Z : out  STD_LOGIC_VECTOR (3 downto 0));
36  end ModuleXOR;
37
38  architecture Behavioral of ModuleXOR is
39
40  begin
41
42          Z <= A XOR B;
43
44  end Behavioral;
```

APÊNDICE C – CÓDIGOS OPERADORES ARITMÉTICOS

C.1 SOMADOR 1 BIT

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 -- Uncomment the following library declaration if using
24 -- arithmetic functions with Signed or Unsigned values
25 --use IEEE.NUMERIC_STD.ALL;
26
27 -- Uncomment the following library declaration if instantiating
28 -- any Xilinx primitives in this code.
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity MY_SOMADOR_1BIT is
33     Port ( X : in  STD_LOGIC;
34            Y : in   STD_LOGIC;
35            Cin : in  STD_LOGIC;
36            Z : out   STD_LOGIC;
37            Cout : out  STD_LOGIC);
38 end MY_SOMADOR_1BIT;
39
40 architecture Behavioral of MY_SOMADOR_1BIT is
41
42         signal S0,S1,S2,S3: STD_LOGIC;
43
44         begin
45                 S0 <= Cin xor Y;
46                 Z <= S0 xor X;
47                 S1 <= Cin and Y;
48                 S2 <= Cin and X;
49                 S3 <= X and Y;
50                 Cout <= S1 or S2 or S3;
51
52 end Behavioral;
```

C.2 SOMADOR 4 BITS

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 -- Uncomment the following library declaration if using
24 -- arithmetic functions with Signed or Unsigned values
25 --use IEEE.NUMERIC_STD.ALL;
26
27 -- Uncomment the following library declaration if instantiating
28 -- any Xilinx primitives in this code.
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32
33 entity MY_SOMADOR_4BIT is
34     Port ( X : in  STD_LOGIC_VECTOR (3 downto 0);
35            Y : in  STD_LOGIC_VECTOR (3 downto 0);
36            Cin : in  STD_LOGIC;
37            Z : out   STD_LOGIC_VECTOR (3 downto 0);
38            Cout : out  STD_LOGIC);
39 end MY_SOMADOR_4BIT;
40
41
42 architecture Behavioral of MY_SOMADOR_4BIT is
43
44         component MY_SOMADOR_1BIT is
45                 Port ( X : in  STD_LOGIC;
46            Y : in   STD_LOGIC;
47            Cin : in  STD_LOGIC;
48            Z : out   STD_LOGIC;
49            Cout : out  STD_LOGIC);
50         end component;
51
52         signal C0,C1,C2 :STD_LOGIC;
53
54         begin
55
56         BLOCO1: MY_SOMADOR_1BIT port map(X(0),Y(0),Cin,Z(0),C0);
57         BlOCO2: MY_SOMADOR_1BIT port map(X(1),Y(1),C0,Z(1),C1);
```

```
58          BlOCO3: MY_SOMADOR_1BIT port map(X(2),Y(2),C1,Z(2),C2);
59          BlOCO4: MY_SOMADOR_1BIT port map(X(3),Y(3),C2,Z(3),Cout);
60
61  end Behavioral;
```

C.3 SUBTRATOR

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity MY_SUBTRATOR_4BIT is
33      Port ( X : in  STD_LOGIC_VECTOR (3 downto 0);
34             Y : in  STD_LOGIC_VECTOR (3 downto 0);
35             Z : out  STD_LOGIC_VECTOR (3 downto 0);
36             Bout : out  STD_LOGIC);
37  end MY_SUBTRATOR_4BIT;
38
39  architecture Behavioral of MY_SUBTRATOR_4BIT is
40
41          component MY_SOMADOR_4BIT is
42                  Port ( X : in  STD_LOGIC_VECTOR (3 downto 0);
43             Y : in  STD_LOGIC_VECTOR (3 downto 0);
44             Cin : in  STD_LOGIC;
45             Z : out   STD_LOGIC_VECTOR (3 downto 0);
46             Cout : out  STD_LOGIC);
47          end component;
48
49          signal C1y:  STD_LOGIC_VECTOR (3 downto 0);
50          signal Cout: STD_LOGIC;
51
52          begin
```

```
53                  C1y <= not Y;
54                  MY_BLOCO: MY_SOMADOR_4BIT port map(X,C1y,'1',Z,Cout);
55                  Bout <= not Cout;
56
57
58  end Behavioral;
```

C.4 MULTIPLICADOR

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity MY_MULTIPLICADOR is
33      Port ( X : in  STD_LOGIC_VECTOR (3 downto 0);
34             Y : in  STD_LOGIC_VECTOR (3 downto 0);
35             Z : out  STD_LOGIC_VECTOR (3 downto 0));
36  end MY_MULTIPLICADOR;
37
38  architecture Behavioral of MY_MULTIPLICADOR is
39
40          component MY_SOMADOR_1BIT is
41                  Port ( X : in  STD_LOGIC;
42             Y : in   STD_LOGIC;
43             Cin : in  STD_LOGIC;
44             Z : out   STD_LOGIC;
45             Cout : out  STD_LOGIC);
46          end component;
47
48          signal C0,C1,C2,C3,C4,C5,C12,S1,S2,S3: STD_LOGIC;
49          signal A0,A1,A2,A3,A4,A5,A6,A7,A8: STD_LOGIC;
50
```

```
51          begin
52                  -- primeiro bit
53                  Z(0) <= X(0) and Y(0);
54                  -- segundo bit
55                  A0 <= X(1) and Y(0);
56                  A1 <= X(0) and Y(1);
57                  BLOCO1: MY_SOMADOR_1BIT port map (A0,A1, '0' , Z(1), C0);
58                  -- terceiro bit
59                  A2 <= X(2) and Y(0);
60                  A3 <= X(1) and Y(1);
61                  A4 <= X(0) and Y(2);
62                  BLOCO2: MY_SOMADOR_1BIT port map (A2,A3, C0 , S1, C1);
63                  BLOCO3: MY_SOMADOR_1BIT port map (S1 ,A4, '0' , Z(2),
                    ↪  C2);
64                  -- quarto bit
65                  A5 <= X(3) and Y(0);
66                  A6 <= X(2) and Y(1);
67                  A7 <= X(1) and Y(2);
68                  A8 <= X(0) and Y(3);
69                  C12 <= C1 xor C2;
70                  BLOCO4: MY_SOMADOR_1BIT port map (A5,A6, C12 , S2, C3);
71                  BLOCO5: MY_SOMADOR_1BIT port map (S2,A7, '0' , S3, C4);
72                  BLOCO6: MY_SOMADOR_1BIT port map (S3,A8, '0', Z(3), C5);
73
74
75  end Behavioral;
```

C.5 COMPLEMENTO DE 2

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
```

```
31
32 entity MY_C2 is
33     Port ( X : in  STD_LOGIC_VECTOR (3 downto 0);
34            Z : out  STD_LOGIC_VECTOR (3 downto 0));
35 end MY_C2;
36
37 architecture Behavioral of MY_C2 is
38         component MY_SOMADOR_4BIT is
39                  Port ( X : in  STD_LOGIC_VECTOR (3 downto 0);
40            Y : in  STD_LOGIC_VECTOR (3 downto 0);
41            Cin : in  STD_LOGIC;
42            Z : out   STD_LOGIC_VECTOR (3 downto 0);
43            Cout : out  STD_LOGIC);
44         end component;
45
46         signal C1:  STD_LOGIC_VECTOR (3 downto 0);
47         signal Cout: STD_LOGIC;
48
49         begin
50                 C1 <= not X;
51                 MY_BLOCO: MY_SOMADOR_4BIT port map(C1,"0000",'1',Z,Cout);
52
53
54 end Behavioral;
```

APÊNDICE D – CÓDIGO INTERFACE COM USUÁRIO

D.1 MODULE INTERFACE

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 -- Uncomment the following library declaration if using
24 -- arithmetic functions with Signed or Unsigned values
25 --use IEEE.NUMERIC_STD.ALL;
26
27 -- Uncomment the following library declaration if instantiating
28 -- any Xilinx primitives in this code.
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity ModuloInterface is
33     port (
34         clk : in std_logic;
35         BOTAO_A : in std_logic;
36         BOTAO_B : in std_logic;
37         BOTAO_S : in std_logic;
38         BOTAO_RST : in std_logic;
39         SW : in std_logic_vector (3 downto 0);
40         LED_A : out std_logic;
41         LED_B : out std_logic;
42         LED_S : out std_logic;
43         LEDS: out std_logic_vector (3 downto 0)
44     );
45 end ModuloInterface;
46
47 architecture Behavioral of ModuloInterface is
48
49     component ModuleALU is -- pegando a ALU
50                 Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
51            B : in  STD_LOGIC_VECTOR (3 downto 0);
52            Z : out  STD_LOGIC_VECTOR (3 downto 0);
53            S : in  STD_LOGIC_VECTOR (2 downto 0));
54         end component;
55
```

```
56      component redutorClock is -- pegando o clk
57          port (
58              clk_in : in std_logic;
59              clk_out : out std_logic
60          );
61      end component;
62
63      type estado is (E0,E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,E12,E13,E14,
        ↪  E15,E16,E17,E18,E19,E20,E21,E22,E23,E24,E25,E26,e27);
64           -- criando os estados
65      signal estadoAtual, estadoAux: estado := E0; -- criando o estado
        ↪  oficial e uma auxiliar
66      signal A,B,S_inicial: std_logic_vector (3 downto 0); -- sinais do
        ↪  numero A, B e a selecao S recebida nos SW
67      signal Z0,Z1,Z2,Z3,Z4,Z5,Z6,Z7 : std_logic_vector (3 downto 0); --
        ↪  sinais de saida
68      signal S_int : integer;
69           signal clk_novo : std_logic;
70
71  begin
72      BLOCO_CLK : redutorClock port map (clk,clk_novo); -- mandamos o clk
        ↪  da placo e pegamos um reduzido que eh o q vamos usar
73      BLOCO_ALU0: ModuleALU port map(A,B,Z0,"000"); -- conectando as
        ↪  entradas e saidas da ALU para S = 0
74      BLOCO_ALU1: ModuleALU port map(A,B,Z1,"001"); -- conectando as
        ↪  entradas e saidas da ALU para S = 1
75      BLOCO_ALU2: ModuleALU port map(A,B,Z2,"010"); -- conectando as
        ↪  entradas e saidas da ALU para S = 2
76      BLOCO_ALU3: ModuleALU port map(A,B,Z3,"011"); -- conectando as
        ↪  entradas e saidas da ALU para S = 3
77      BLOCO_ALU4: ModuleALU port map(A,B,Z4,"100"); -- conectando as
        ↪  entradas e saidas da ALU para S = 4
78      BLOCO_ALU5: ModuleALU port map(A,B,Z5,"101"); -- conectando as
        ↪  entradas e saidas da ALU para S = 5
79      BLOCO_ALU6: ModuleALU port map(A,B,Z6,"110"); -- conectando as
        ↪  entradas e saidas da ALU para S = 6
80      BLOCO_ALU7: ModuleALU port map(A,B,Z7,"111"); -- conectando as
        ↪  entradas e saidas da ALU para S = 7
81
```

```
82
83      process(clk_novo, estadoAux) -- logica do estado auxiliar
84      begin
85          if(clk_novo'event and clk_novo='1') then
86              estadoAtual <= estadoAux;  -- alteramos o estado no clock
87          end if;
88      end process;
89
90      process(BOTAO_A,BOTAO_B, BOTAO_S,BOTAO_RST, SW,estadoAtual) -- agora
        ↪  a logica da maquina de estados
91      begin
92                  report "Iniciou";
93          if (BOTAO_RST = '1') then -- caso tenhamos RST, voltamos para o
            ↪  primeiro estado
94              estadoAux <= E0;
95              LEDS <= "0000";   -- apaga os LEDS
96                                  report "Resetou";
97          else                    -- resto da logica acontece se nao
            ↪  tivermos o reset
98              case estadoAtual is
99                  when E0 =>       -- E0 -> pegar o numero A
                    ↪  ---------------------------------------
100                     LED_A <= '1';      -- acende o led q sinaliza q
                        ↪  estamos pegando o primeiro num
101                     LED_B <= '0';
102                     LED_S <= '0';
103
104                     if(BOTAO_A = '1') then  -- gravamos oq esta nos
                        ↪  switchs quando o botao é apertado
105                         A <= SW;
106                         LED_A <= '0';      -- apaga o led
107                         estadoAux<=E1;     -- mudamos de estado
108                     else
109                          estadoAux<=E0;  -- se o batao nao for apertado
                             ↪  cont no mesmo estado
110                     end if;
111                 when E1 => -- E1 -> pegar o numero B
                    ↪  ---------------------------------------
```

```
112                     LED_B <= '1';       -- acende o led q sinaliza q
                        ↪ estamos pegando o segundo num
113                     LED_A <= '0';
114                     LED_S <= '0';
115
116                     if(BOTAO_B = '1') then  -- gravamos oq esta nos
                        ↪ switchs quando o botao é apertado
117                         B <= SW;
118                         LED_B <= '0';       -- apaga o led
119                         estadoAux<=E2;      -- mudamos de estado
120                     else
121                          estadoAux<=E1;  -- se o batao nao for apertado
                             ↪ cont no mesmo estado
122                     end if;
123                 when E2 => -- E2 -> pegar a selecao
                    ↪ ---------------------------------------
124                     LED_S <= '1';       -- acende o led q sinaliza q
                        ↪ estamos pegando a selecao
125                     LED_A <= '0';
126                     LED_B <= '0';
127
128                     if(BOTAO_S = '1') then  -- gravamos oq esta nos
                        ↪ switchs quando o botao é apertado
129                         S_inicial <= SW;
130
131                         if(S_inicial = "0000") then -- passando de
                            ↪ binario para inteiro
132                             estadoAux<=E3;
133                         elsif(S_inicial = "0001") then
134                             estadoAux<=E6;
135                         elsif(S_inicial = "0010") then
136                            estadoAux<=E9;
137                         elsif(S_inicial = "0011") then
138                            estadoAux<=E12;
139                         elsif(S_inicial = "0100") then
140                             estadoAux<=E15;
141                         elsif(S_inicial = "0101") then
142                             estadoAux<=E18;
143                         elsif(S_inicial = "0110") then
```

```
144                         estadoAux<=E21;
145                     elsif(S_inicial = "0111") then
146                         estadoAux<=E24;
147                     end if;
148
149                     LED_S <= '0';       -- apaga o led
150                 else
151                      estadoAux<=E2;  -- se o batao nao for apertado
                     ↪  cont no mesmo estado
152                 end if;
153             when E3 =>  -- E3 -> mostra p numero A
                ↪  ---------------------------------------
154
155                 LED_A <= '1';  -- acende o led q sinaliza q estamos
                    ↪  mostrando o primeiro num
156                 LED_B <= '0';
157                 LED_S <= '0';
158
159                 LEDS <= A ;          --- mostra A
160                 estadoAux<=E4;  -- proximo
161             when E4 =>  -- E4 -> mostra p numero B
                ↪  ---------------------------------------
162
163                 LED_A <= '0';  -- acende o led q sinaliza q estamos
                    ↪  mostrando no segundo num
164                 LED_B <= '1';
165                 LED_S <= '0';
166
167                 LEDS <= B ;          --- mostra B
168                 estadoAux<=E5;  -- proximo
169             when E5 =>  -- E5 -> mostra o resultado
                ↪  ---------------------------------------
170                 LED_A <= '0';  -- acende o led q sinaliza q estamos
                    ↪  mostrando o resultado
171                 LED_B <= '0';
172                 LED_S <= '1';
173                 LEDS<=Z0;
174                 estadoAux <= E6;
```

```
175                                 when E6 =>  -- E6 -> mostra p numero A
                                    ↪ ------------------------------------

176
177                     LED_A <= '1';  -- acende o led q sinaliza q estamos
                        ↪ mostrando o primeiro num
178                     LED_B <= '0';
179                     LED_S <= '0';
180
181                     LEDS <= A ;          --- mostra A
182                     estadoAux<=E7;  -- proximo
183                 when E7 =>  -- E7 -> mostra p numero B
                    ↪ ------------------------------------

184
185                     LED_A <= '0';  -- acende o led q sinaliza q estamos
                        ↪ mostrando no segundo num
186                     LED_B <= '1';
187                     LED_S <= '0';
188
189                     LEDS <= B ;          --- mostra B
190                     estadoAux<=E8;  -- proximo
191                 when E8 =>  -- E8 -> mostra o resultado
                    ↪ ------------------------------------
192                     LED_A <= '0';  -- acende o led q sinaliza q estamos
                        ↪ mostrando o resultado
193                     LED_B <= '0';
194                     LED_S <= '1';
195                     LEDS <= Z1;
196                     estadoAux <= E9;
197                                 when E9 =>  -- E9 -> mostra p numero
                                    ↪ A
                                    ↪ ------------------------------------

198
199                     LED_A <= '1';  -- acende o led q sinaliza q estamos
                        ↪ mostrando o primeiro num
200                     LED_B <= '0';
201                     LED_S <= '0';
202
203                     LEDS <= A ;          --- mostra A
204                     estadoAux<=E10;  -- proximo
```

```
205                 when E10 =>  -- E10 -> mostra p numero B
                    ↪ ---------------------------------------
206
207                     LED_A <= '0';  -- acende o led q sinaliza q estamos
                        ↪ mostrando no segundo num
208                     LED_B <= '1';
209                     LED_S <= '0';
210
211                     LEDS <= B ;          --- mostra B
212                     estadoAux<=E11;  -- proximo
213                 when E11 =>  -- E11 -> mostra o resultado
                    ↪ ---------------------------------------
214                     LED_A <= '0';  -- acende o led q sinaliza q estamos
                        ↪ mostrando o resultado
215                     LED_B <= '0';
216                     LED_S <= '1';
217                     LEDS <= Z2;
218                     estadoAux <= E12;
219                                 when E12 =>  -- E12 -> mostra p
                                    ↪ numero A
                                    ↪ ---------------------------------------
220
221                     LED_A <= '1';  -- acende o led q sinaliza q estamos
                        ↪ mostrando o primeiro num
222                     LED_B <= '0';
223                     LED_S <= '0';
224
225                     LEDS <= A ;          --- mostra A
226                     estadoAux<=E13;  -- proximo
227                 when E13 =>  -- E13 -> mostra p numero B
                    ↪ ---------------------------------------
228
229                     LED_A <= '0';  -- acende o led q sinaliza q estamos
                        ↪ mostrando no segundo num
230                     LED_B <= '1';
231                     LED_S <= '0';
232
233                     LEDS <= B ;          --- mostra B
234                     estadoAux<=E14;  -- proximo
```

```
235                 when E14 =>  -- E14 -> mostra o resultado
                    ↪ ---------------------------------------
236                     LED_A <= '0';  -- acende o led q sinaliza q estamos
                        ↪ mostrando o resultado
237                     LED_B <= '0';
238                     LED_S <= '1';
239
240                     LEDS <= Z3;
241                     estadoAux <= E15 ;
242                                         when E15 =>  -- E15 -> mostra p
                                            ↪ numero A
                                            ↪ ---------------------------------------
243
244                     LED_A <= '1';  -- acende o led q sinaliza q estamos
                        ↪ mostrando o primeiro num
245                     LED_B <= '0';
246                     LED_S <= '0';
247
248                     LEDS <= A ;          --- mostra A
249                     estadoAux<=E16;  -- proximo
250                 when E16 =>  -- E16 -> mostra p numero B
                    ↪ ---------------------------------------
251
252                     LED_A <= '0';  -- acende o led q sinaliza q estamos
                        ↪ mostrando no segundo num
253                     LED_B <= '1';
254                     LED_S <= '0';
255
256                     LEDS <= B ;          --- mostra B
257                     estadoAux<=E17;  -- proximo
258                 when E17 =>  -- E17 -> mostra o resultado
                    ↪ ---------------------------------------
259                     LED_A <= '0';  -- acende o led q sinaliza q estamos
                        ↪ mostrando o resultado
260                     LED_B <= '0';
261                     LED_S <= '1';
262                     LEDS<=Z4;
263
264                     estadoAux <= E18;
```

```
265                                                 when E18 =>  -- E18 ->
                                                    ↪ mostra p numero A
                                                    ↪ ------------------------------
266
267                 LED_A <= '1';  -- acende o led q sinaliza q estamos
                    ↪ mostrando o primeiro num
268                 LED_B <= '0';
269                 LED_S <= '0';
270
271                 LEDS <= A ;          --- mostra A
272                 estadoAux<=E19;  -- proximo
273             when E19 =>  -- E19 -> mostra p numero B
                ↪ ---------------------------------------
274
275                 LED_A <= '0';  -- acende o led q sinaliza q estamos
                    ↪ mostrando no segundo num
276                 LED_B <= '1';
277                 LED_S <= '0';
278
279                 LEDS <= B ;          --- mostra B
280                 estadoAux<=E20;  -- proximo
281             when E20 =>  -- E20 -> mostra o resultado
                ↪ ---------------------------------------
282                 LED_A <= '0';  -- acende o led q sinaliza q estamos
                    ↪ mostrando o resultado
283                 LED_B <= '0';
284                 LED_S <= '1';
285                 LEDS<=Z5;
286
287                 estadoAux <= E21;
288                                     when E21 =>  -- E21 -> mostra p
                                        ↪ numero A
                                        ↪ ---------------------------------------
289
290                 LED_A <= '1';  -- acende o led q sinaliza q estamos
                    ↪ mostrando o primeiro num
291                 LED_B <= '0';
292                 LED_S <= '0';
293
```

```
294                     LEDS <= A ;          --- mostra A
295                     estadoAux<=E22;  -- proximo
296                 when E22 =>  -- E22 -> mostra p numero B
                    ↪ ---------------------------------------
297
298                     LED_A <= '0';  -- acende o led q sinaliza q estamos
                        ↪ mostrando no segundo num
299                     LED_B <= '1';
300                     LED_S <= '0';
301
302                     LEDS <= B ;          --- mostra B
303                     estadoAux<=E23;  -- proximo
304                 when E23 =>  -- E23 -> mostra o resultado
                    ↪ ---------------------------------------
305                     LED_A <= '0';  -- acende o led q sinaliza q estamos
                        ↪ mostrando o resultado
306                     LED_B <= '0';
307                     LED_S <= '1';
308                     LEDS<=Z6;
309
310                     estadoAux <= E24;
311                                 when E24 =>  -- E24 -> mostra p
                                    ↪ numero A
                                    ↪ ---------------------------------------
312
313                     LED_A <= '1';  -- acende o led q sinaliza q estamos
                        ↪ mostrando o primeiro num
314                     LED_B <= '0';
315                     LED_S <= '0';
316
317                     LEDS <= A ;          --- mostra A
318                     estadoAux<=E25;  -- proximo
319                 when E25 =>  -- E25 -> mostra p numero B
                    ↪ ---------------------------------------
320
321                     LED_A <= '0';  -- acende o led q sinaliza q estamos
                        ↪ mostrando no segundo num
322                     LED_B <= '1';
323                     LED_S <= '0';
```

```
324
325                     LEDS <= B ;          --- mostra B
326                     estadoAux<=E26;   -- proximo
327                 when E26 =>   -- E26 -> mostra o resultado
                    ↪ ---------------------------------------
328                     LED_A <= '0';  -- acende o led q sinaliza q estamos
                        ↪ mostrando o resultado
329                     LED_B <= '0';
330                     LED_S <= '1';
331                     LEDS<=Z7;
332
333                     estadoAux <=
                        ↪ E3;
334                 when others =>
335                     null;
336
337             end case;
338         end if;
339     end process;
340
341
342 end architecture;
```

D.2 REDUTOR DE CLOCK

```
30 library IEEE;
31 use IEEE.STD_LOGIC_1164.ALL;
32
33 entity redutorClock is
34     port (
35         clk_in : in std_logic;
36         clk_out : out std_logic
37     );
38 end redutorClock;
39
40 architecture arcRedutorClock of redutorClock is
41     signal aux : integer;
42     begin
43         process(clk_in)
44                   begin
```

```
45              if(clk_in'event and clk_in = '1') then
46                  -- o clock interno da placa possui freq de 50MHz.
47                  --Vamos multiplicar seu periodo por
48                  -- 100 milhoes para ter um periodo de 2 segundos
49                  if(aux = 100000000 ) then  -- variavel auxiliar vai de
                    ↪  zero ate
50                                                  --100 milhoes e volta
51                     aux<=0;
52                  else
53                      aux <= aux + 1;
54                  end if;
55
56                  if(aux >= 50000000) then -- metado do tempo ela ta em
                    ↪  high,
57                                              -- metade em low
58                      clk_out <='1';
59                  else
60                      clk_out <='0';
61                  end if;
62              end if;
63          end process;
64  end architecture;
```

APÊNDICE E – CÓDIGO PINAGEM

```
1  NET "SW<0>"          LOC = "V8"    | IOSTANDARD = LVCMOS33 ;
2  NET "SW<1>"          LOC = "U10"   | IOSTANDARD = LVCMOS33 ;
3  NET "SW<2>"          LOC = "U8"    | IOSTANDARD = LVCMOS33 ;
4  NET "SW<3>"          LOC = "T9"    | IOSTANDARD = LVCMOS33 ;
5
6  NET "LEDS<0>"         LOC = "R20"   | IOSTANDARD = LVCMOS33 | DRIVE = 8 |
   ↪  SLEW = SLOW ;
7  NET "LEDS<1>"         LOC = "T19"   | IOSTANDARD = LVCMOS33 | DRIVE = 8 |
   ↪  SLEW = SLOW ;
8  NET "LEDS<2>"         LOC = "U20"   | IOSTANDARD = LVCMOS33 | DRIVE = 8 |
   ↪  SLEW = SLOW ;
9  NET "LEDS<3>"         LOC = "U19"   | IOSTANDARD = LVCMOS33 | DRIVE = 8 |
   ↪  SLEW = SLOW ;
10
11 NET "LED_A"         LOC = "W21"   | IOSTANDARD = LVCMOS33 | DRIVE = 8 |
   ↪  SLEW = SLOW ;
12 NET "LED_B"         LOC = "Y22"   | IOSTANDARD = LVCMOS33 | DRIVE = 8 |
   ↪  SLEW = SLOW ;
13 NET "LED_S"         LOC = "V20"   | IOSTANDARD = LVCMOS33 | DRIVE = 8 |
   ↪  SLEW = SLOW ;
14
15 NET "BOTAO_A"          LOC = "U15"    | IOSTANDARD = LVCMOS33 ;
16 NET "BOTAO_B"          LOC = "T15"    | IOSTANDARD = LVCMOS33 ;
17 NET "BOTAO_S"          LOC = "T16"    | IOSTANDARD = LVCMOS33 ;
18 NET "BOTAO_RST"          LOC = "T14"    | IOSTANDARD = LVCMOS33 ;
19 NET "BOTAO_RST" CLOCK_DEDICATED_ROUTE = FALSE;
20
21 NET "clk"          LOC = "E12"    | IOSTANDARD = LVCMOS33 | PERIOD =
   ↪  20.000;
```