IMPLEMENTAÇÃO E SIMULAÇÃO DE UMA UNIDADE LÓGICA E ARITMÉTICA DE 16 BITS COM AS OPERAÇÕES DE MULTIPLICAÇÃO, DESLOCAMENTO E ROTAÇÃO DE BITS EM VHDL

VHDL IMPLEMENTATION AND SIMULATION OF A 16-BIT LOGICAL AND ARITHMETIC UNIT TO MULTIPLICATION, BIT ROTATION AND SHIFTING

ANDRESSA FERREIRA DE OLIVEIRA, ELIAN MIGUEL DE OLIVEIRA, MAURO HEMERLY GAZZANI, KÁTIA LOPES SILVA

RESUMO

Este trabalho apresenta a implementação em VHDL de uma ULA(Unidade Lógica e Aritmética) de 16 bits com as 11 operações lógicas e aritméticas, além das operações de multiplicação, deslocamento e rotação de bits. Os resultados das simulações de casos de testes foram gerados utilizando *test benches* específicas da linguagem VHDL. O modelo da ULA foi projetado possibilitando que o módulo possa ser utilizado como componente para o projeto de ULA 16, 32, 64 bits ou mais. Os resultados da simulação comprovam que este projeto foi executado com sucesso conforme o esperado.

Palavras-chave: ULA; LDH;Bancada de Teste

ABSTRACT

This work presents the VHDL implementation of a 16-bit ALU (Arithmetic Logic Unit) with 11 arithmetic and logic operations, in addition to the bit multiplication, shift and rotation operations. The test case simulation results were generated using VHDL language-specific testbenches. The ALU model was designed so that the module can

be used as a component for the design of ALU 16, 32, 64 bits or more. The simulation results prove that this project was successfully executed as expected.

Keywords: ALU; HDL; Testbench

INTRODUÇÃO

A VHDL (VHSIC *Hardware Description Language*) tornou-se uma ferramenta essencial para projetistas no mundo do projeto digital. VHDL permite a síntese de circuito, bem como simulação de circuito. O primeiro é a tradução de um código-fonte em uma estrutura de hardware que implementa a funcionalidade pretendida, enquanto o último é um procedimento de teste para garantir que tal a funcionalidade é de fato alcançada pelo circuito sintetizado (PEDRONI, 2010).

O código em linguagens de descrição de *hardware*(HDL *Hardware Description Language*) descreve o comportamento ou estrutura de um circuito eletrônico, a partir do qual um circuito físico compatível pode ser inferido por um compilador. Suas principais aplicações incluem a síntese de circuitos digitais em CPLD/FPGA (Dispositivos lógicos programáveis complexos/Matriz de portas programáveis em campo) chips e layout / geração de máscara para fabricação de ASIC (Circuito Integrado de Aplicação Específica) (VAHID,2008).

Em computação, uma Unidade Lógica e Aritmética(ULA) é um circuito digital que realiza operações aritméticas e lógicas inteiras operações. A ULA é um bloco básico da unidade central de processamento de um computador, e até mesmo os microprocessadores mais simples que realizam o trabalho de manutenção dos temporizadores (TAIB et al., 2020).

Uma ULA pode ser descrita uma função digital de lógica combinatória com múltiplas operações. Ele pode realizar uma série de operações aritméticas básicas e um conjunto de operações lógicas. A ULA tem uma série de linhas de seleção para selecionar uma determinada operação na unidade (SARANGI et al.,2020).

Taib e outros (2020) propuseram um modelo de uma ULA de 16 bits usando VHDL. O software Altera Quartus II foi usado como a ferramenta para criar a

operação projetada de multiplicação e divisão. Os resultados da simulação mostraram que o projeto da ULA proposto executa com sucesso as operações de multiplicação e divisão com 16 bits.

Singh (2010) desenvolveu uma dissertação relacionada com o projeto e implementação FPGA de um módulo de Aritmética de 16 bits, que utiliza algoritmos de Matemática Védica. Para a multiplicação aritmética, várias técnicas de multiplicação védica, como UrdhvaTiryakbhyam, Nikhilam e Anurupye, foram exaustivamente analisadas. Também foi discutido o algoritmo de Karatsuba para multiplicação. Foi descoberto que UrdhvaTiryakbhyam Sutra é o Sutra (Algoritmo) mais eficiente, dando atraso mínimo para a multiplicação de todos os tipos de números.

O algoritmo da multiplicação implementado neste trabalho na ULA 16 bits é baseado na Matemática Védica, proposta por Singh (2010) que permite realizar as operações do sutra Urdhva-Tiryagabhyam de forma paralela, o que confere um aumento da velocidade das operações de multiplicação da ULA em comparação com o algoritmo de que é utilizado atualmente nos computadores (SINGH,2010).

A ULA é um projeto que tem por objetivo de desenvolver algoritmos adequados para alcançar uma utilização eficiente do *hardware* disponível. Este trabalho apresenta a implementação de uma ULA de 16 bits com as 11 operações lógicas e aritméticas, além das operações de multiplicação e rotação de bits usando VHDL.

MATERIAL E MÉTODOS

Uma alternativa à entrada esquemática de um circuito digital em um sistema de projeto auxiliado por computador é utilizar a técnica de projeto de dispositivos lógicos programáveis (PLD *Programmable Logic Device*) com uma ferramenta de projeto baseado em texto ou em HDL. Exemplos de HDLs: AHDL, e os padrões VHDL e Verilog.

Neste trabalho, o projeto de uma ULA típica é realizado em quatro etapas. Primeiramente, o projeto da unidade aritmética da ULA de 16 bits é realizado; em segundo lugar, o projeto da seção lógica é considerado, em seguida o projeto das

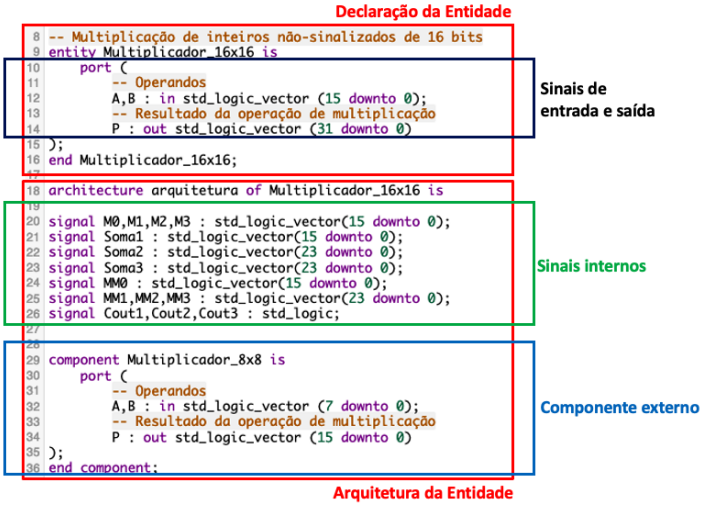
operações de multiplicação de inteiros não sinalizados e sinalizados, e, finalmente, o projeto das operações de rotação e deslocamento de bits.

A ULA é implementada em linguagem VHDL. A ULA é projetada para realizar 11 operações que incluem lógicas e operações aritméticas, além das operações de multiplicação de inteiros e rotação e deslocamento de bits.

Para realizar as simulações será utilizado o EDA Playground (EDA Playground, 2021) que é um recurso de aprendizagem online de acesso gratuito onde o público pode praticar suas habilidades de codificação e compartilhar trechos de código com outras pessoas na comunidade. O objetivo é acelerar o aprendizado de design e desenvolvimento de banco de teste com compartilhamento de código mais fácil e com acesso simples a ferramentas e bibliotecas EDA.

A descrição de qualquer circuito lógico por meio da linguagem VHDL é constituída por duas estruturas sintáticas com finalidades específicas: a descrição da entidade e a arquitetura correspondente que descreve o funcionamento da entidade de acordo com o circuito em questão. A figura 1 ilustra o código da entidade de projeto de um circuito multiplicador de 16 bits, onde a entidade descreve os seus sinais de entrada e saída, e a arquitetura descreve o funcionamento do circuito, bem como os seus sinais internos e os componentes externos conectados.

Figura 1- Estrutura de um programa em VHDL

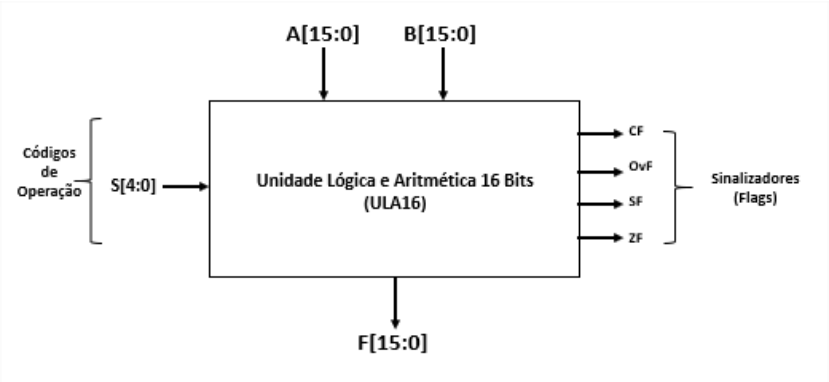


Fonte: Autores

A figura 2 mostra uma visão geral da ULA de 16 Bits do projeto deste trabalho. Os dezesseis bits da entrada A e os dezesseis bits da entrada B são os operandos da ULA. As entradas de seleção de 4 bits S[4:0] formam o código de operação que especifica a operação a ser realizada (operações aritméticas, lógicas, deslocamento e rotação de bits, multiplicação de inteiros não sinalizados e sinalizados).

Os sinalizadores (*flags*) são bits de saída que servem para identificar situações ou estados resultantes da última operação realizada pela ULA referente às operações aritméticas de adição e subtração. A unidade aritmética é constituída de 4 bits sinalizadores em sua saída (segundo nomenclatura dos processadores da plataforma de *hardware* Intel), sendo 2 para detecção de erro (*Overflow*) no resultado na última operação aritmética realizada e 2 para detecção de sinal e nulidade da última operação, conforme mostrado na figura 2. Desta forma tem-se: CF: *flag* de *carry* - sinaliza a ocorrência de *overflow* em operação aritmética de inteiros não-sinalizados 1:*carry*; 0: no *carry*. OvF: *flag* de *overflow* - sinaliza a ocorrência de overflow em operação aritmética de inteiros sinalizados 1:*overflow*; 0: *not overflow*. SF: *flag* de sinal - sinaliza se a operação aritmética resulta em um valor 0:*positive*; 1:*negative*. ZF: *flag* de *zero* - sinaliza se a operação aritmética resulta em um valor 1:*zero*; 0:*no zero*.

Figura 2- Visão geral da ULA 16 Bits



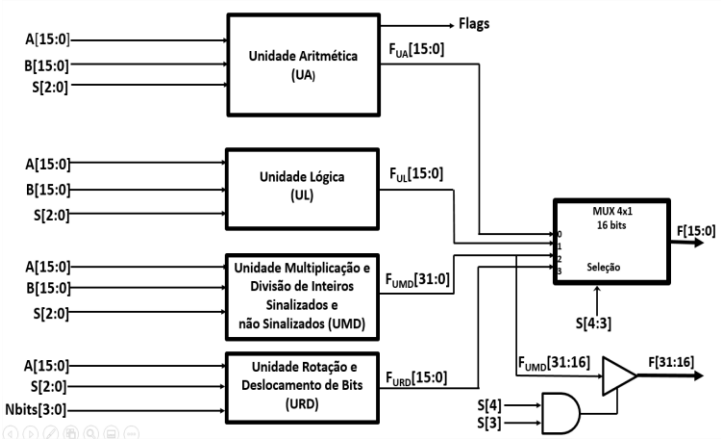
Fonte: Oliveira e Oliveira (2021)

A Figura 3 mostra o diagrama de blocos da ULA de 16 bits. A ULA é implementada em linguagem VHDL e é projetada para realizar 11 operações que incluem lógicas e operações aritméticas, além das operações de multiplicação,

deslocamento e rotação de bits. Para realizar as simulações é utilizado o EDA Playground (que é um recurso de aprendizagem online de acesso gratuito. Esta ferramenta oferece acesso prático imediato para simular pequenas quantidades de SystemVerilog, Verilog, VHDL, C ++, SystemC e outros HDLs. É utilizado ainda neste trabalho o Logisim, para criação e simulação.

O projeto da ULA foi implementado de forma modular, desta forma tem-se os seguintes módulos: Unidade Aritmética (UA) que realiza as operações aritméticas, a Unidade Lógica (UL) que realiza as operações lógicas, Unidade de Multiplicação de Inteiros Não Sinalizados e Sinalizados e Divisão (UMD) e a Unidade Rotação e Deslocamento de Bits (URD), como mostrado na figura 3.

Figura 3- Unidades funcionais da ULA de 16 Bits



Fonte: Autores

A Tabela 1 mostra os códigos de operação S_4 (S[4]) e S_3 (S[3]) que distinguem a unidade funcional da ULA. Os códigos S_2 (S[2]), S_1 (S[1]) e S_0 (S[0]) especificam a operação que será realizada pela respectiva unidade funcional definida anteriormente. Desta forma, com três códigos de operação, é possível especificar até oito operações por unidade funcional.

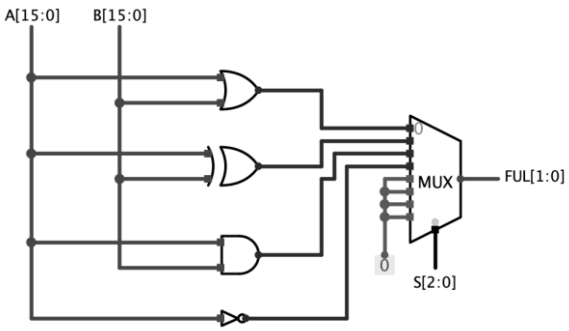
Tabela 1- Tabela de funções da ULA 16 bits

Código de Operação					Saída	Função	Unidade Funcional
S_4	S_3	S_2	S_1	S_0			
0	0	0	0	0	F = A	Transfere A	Aritmética
0	0	0	0	1	F = A + 1	Incrementa A	
0	0	0	1	0	F = A + B	Adição	
0	0	0	1	1	F = A + B + 1	Adição com Vai-um	
0	0	1	0	0	F = A − B − 1	Subtração com Vem-um	
0	0	1	0	1	F = A − B	Subtração	
0	0	1	1	0	F = A − 1	Decrementa A	
0	0	1	1	1	F = A	Transfere A	
0	1	0	0	0	F = A ∨ B	Ou (Or)	Lógica
0	1	0	0	1	F = A ⊕ B	Ou-exclusivo (Xor)	
0	1	0	1	0	F = A ∧ B	E (And)	
0	1	0	1	1	F = $\bar{A}$	Não (Not)	
1	0	0	0	0	F = << A << n	Rotação de n bits à esquerda	Rotação e Deslocamento de Bits
1	0	0	0	1	F = >> A >> n	Rotação de n bits à direita	
1	0	0	1	0	F = A <<< n	Deslocamento lógico de n bits à esquerda	
1	0	0	1	1	F = A >>> n	Deslocamento lógico de n bits à direita	
1	0	1	0	0	F = A >> n	Deslocamento aritmético de n bits à direita	
1	1	0	0	0	F = A * B	Multiplicação Inteiros Não Sinalizados	Multiplicação e Divisão de Inteiros Sinalizados e Não Sinalizados
1	1	0	0	1	F = A * B	Multiplicação Inteiros Sinalizados	

Fonte: Autores

As operações lógicas podem ser obtidas por meio de operações AND, OR e NOT (complemento), podendo ser mais conveniente empregar apenas um circuito lógico com essas operações. Para as operações há três entradasS[2:0] disponíveis de seleção do MUX. Estas três linhas de seleções podem selecionar dentre oito operações lógicas, e desta forma a função OU exclusivo (XOR) para o circuito lógico ser projetado nesta parte (Figura 4).

Figura 4- Circuito da Unidade lógica (UL)

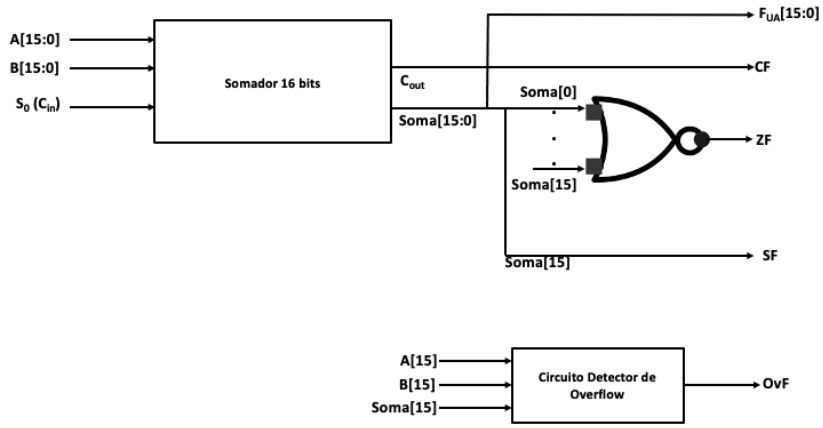


Fonte: Autores

O componente básico da unidade aritmética de uma ULA é um somador completo de 16 bits conforme mostrado na figura 5. Este é construído com vários

circuitos somadores completos conectados em cascata. Quando as entradas de dados A e B são manipuladas no somador de 16 bits, é possível obter diferentes tipos de operações aritméticas. O *carry* de entrada S_0 (S[0]) entra no circuito somador completo de 16 bits na posição de bit menos significativa. O *carry* de saída C_{out} está na saída do circuito somador completo na posição de bit mais significativa.

Figura 2-Circuito da Unidade Aritmética

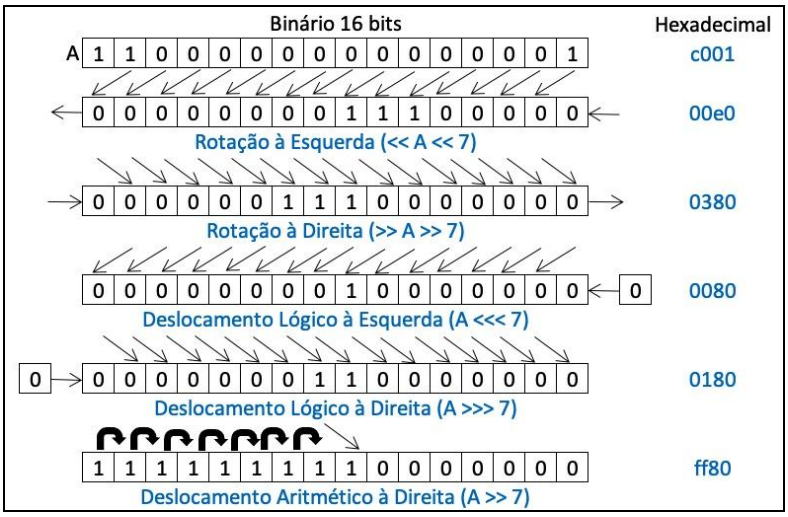


Fonte: Autores

A rotação de bits ou deslocamento circular é uma operação semelhante ao deslocamento, exceto que os bits que saem em uma extremidade são colocados de volta na outra extremidade. Na rotação para a esquerda, os bits que saem na extremidade esquerda são colocados de volta na extremidade direita. Na rotação à direita, os bits que saem na extremidade direita são colocados de volta na extremidade esquerda (CAVANAGH,2010).

As operações de rotação e deslocamento podem ser usadas para manipular o mapa de bits de imagens digitais em grupos de bits que devem ser deslocados para a esquerda e para a direita para reposicionar as imagens na tela. Uma outra aplicação é a criptografia de dados em que o algoritmo de criptografia envolve a mudança dos bits. Finalmente, as operações de esquerda e direita, podem ser usadas ao realizar multiplicação ou divisão rápida com inteiros muito longos, quando estas operações são uma potência de 2.A figura 53 mostra o método de rotação e deslocamento de bits para a entrada A= $C001_{16.}$

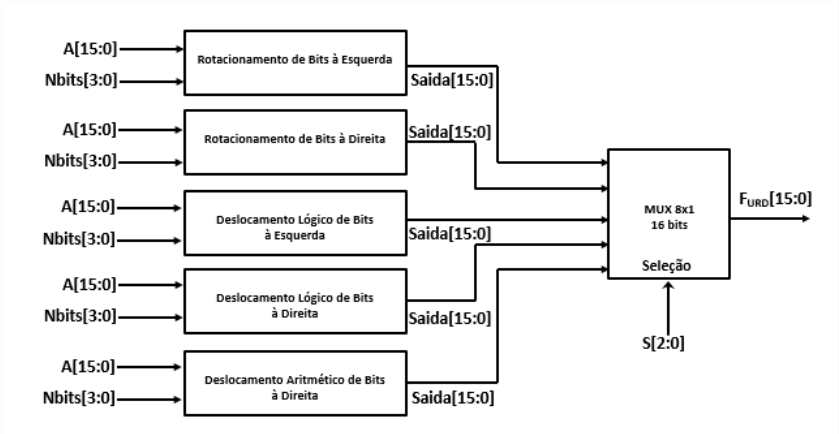
Figura 6 –Exemplificação de Rotação e Deslocamento de Bits



Fonte: Autores

O diagrama em blocos da Unidade Rotação e Deslocamento de Bits (URD) é mostrado na figura 7. Cinco operações: Rotacionamento de Bits à esquerda, Rotacionamento de Bits à direita, Deslocamento Lógico de Bits à esquerda, Deslocamento Lógico de Bits à direita, Deslocamento Aritmético de Bits à direita são implementadas e para cada uma delas tem-se como entrada A[15:0] e Nbits[3:0] que é a quantidade de bits que se deseja deslocar ou rotacionar, conforme a operação escolhida. Uma vez tendo a saída, o MUX 8x1 realiza a seleção, conforme os valores de S[2:0], e obtém a saída F_{URD}.

Figura 7 - Unidade Rotção e Deslocamento de Bits (URD)



Fonte: Autores

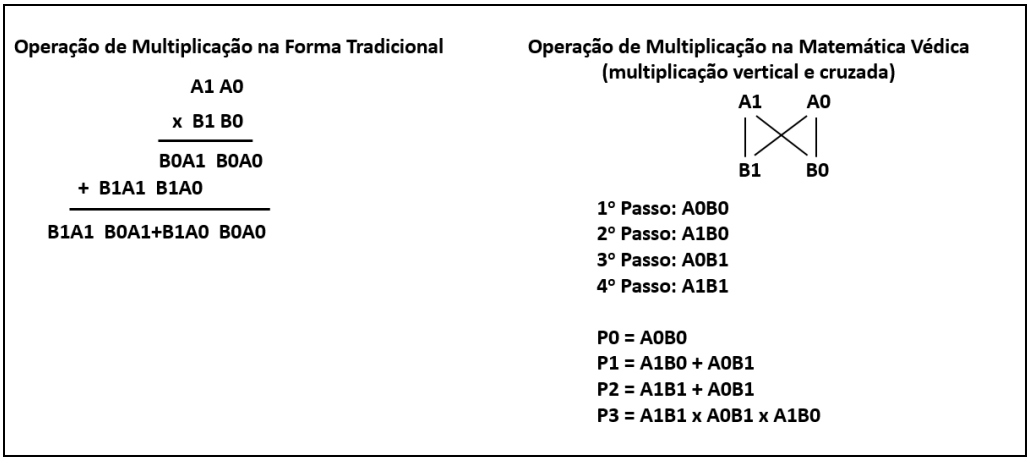
O método de multiplicação baseado na Matemática Védica é mostrado na figura 8. O sutra Urdhva-Tiryagabhyam (verticalmente e transversalmente) utilizado no método é semelhante ao algoritmo convencional da multiplicação utilizado atualmente. A multiplicação é realizada em quatro passos: Primeiro passo: multiplicação vertical A0B0; Segundo passo: multiplicação transversal: A1B0;Terceiro passo: multiplicação transversal: A0B1; Quarto passo: multiplicação vertical A1B1.

Primeiramente P0 = A0B0, em seguida são realizadas as somas parciais resultando nos produtos:

P1 = A1B0 + A0B1, P2 = A1B1 + A0B1, P3 = A1B1 x A0B1 x A1B0.

O algoritmo da multiplicação implementado na ULA 16 bits é baseado na Matemática Védica que permite realizar as operações do sutra Urdhva-Tiryagabhyam de forma paralela, o que confere um aumento da velocidade das operações de multiplicação da ULA em comparação com o algoritmo de que é utilizado atualmente nos computadores.

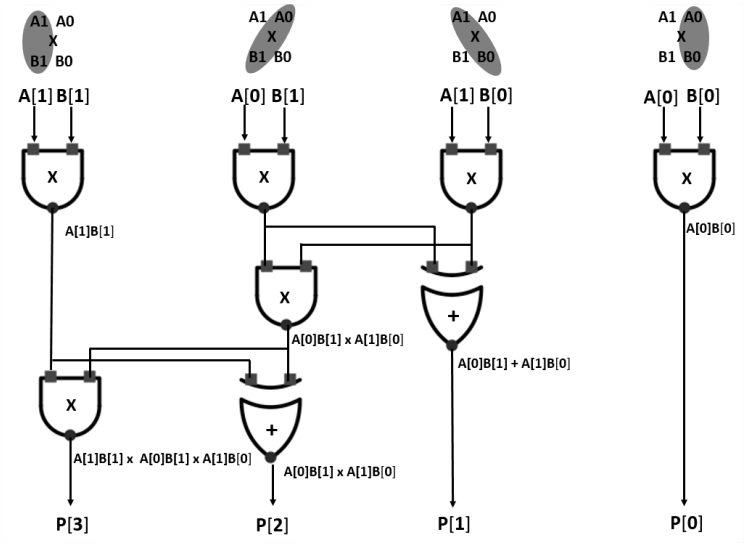
Figura 8 - Método de Multiplicação baseado na Matemática Védica



Fonte: Autores

A figura 9 mostra o circuito Multiplicador 2x2 para inteiros não sinalizados.

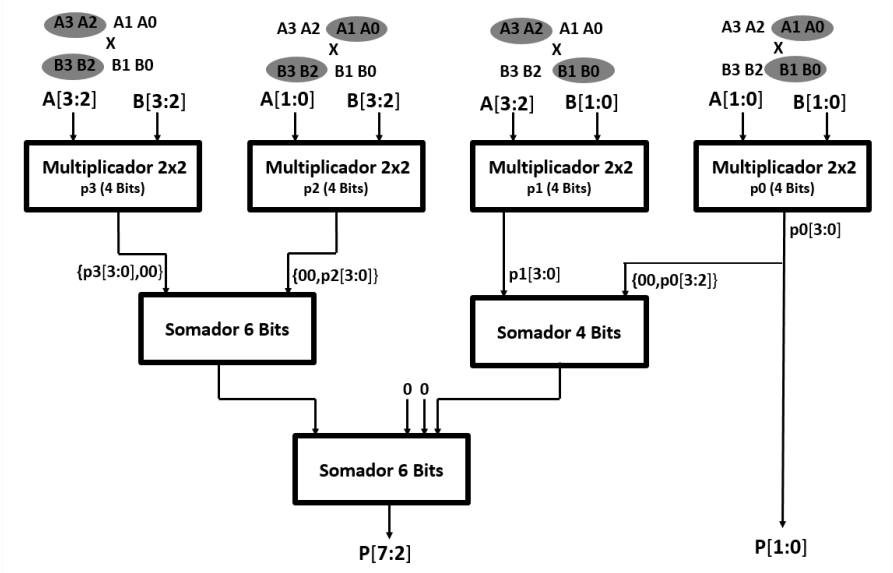
Figura 9 - Circuito Multiplicador 2x2 para inteiros não sinalizados



Fonte: Modificada de Singh, 2010

O método de multiplicação utilizado na ULA empregando os circuitos multiplicadores 4x4 não sinalizados é mostrado na figura 10. Pode-se notar que este mesmo utiliza o Multiplicador 2x2, bem como um somador de 4 Bits. O resultado final é obtido pelo somador de 6 Bits.

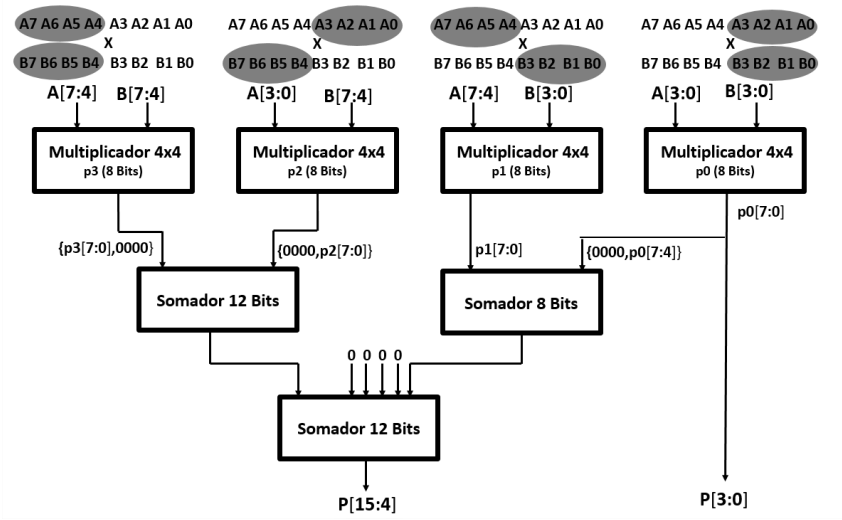
Figura 10- Multiplicador 4x4 (Não sinalizado)



Fonte: Modificada de Singh, 2010

Baseado na reutilização de módulos, o Multiplicador 8x8 emprega o multiplicador 4x4, que por sua vez utiliza o multiplicador 2x2, conforme mostrado na figura 11. Pode-se notar além do multiplicador 4x4, o método utiliza os somadores de 12 e 8 Bits. O resultado final é obtido pelo somador de 12 Bits.

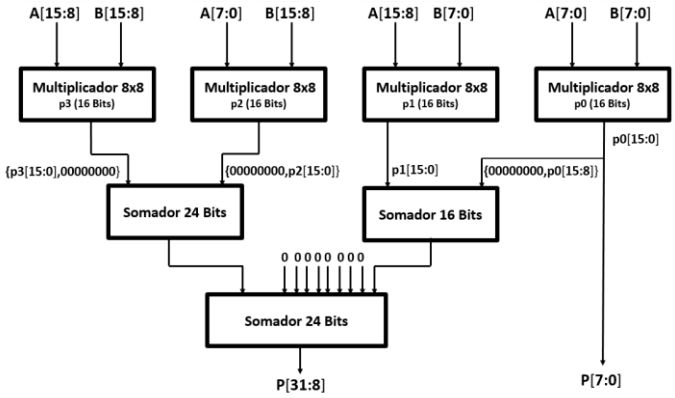
Figura 11- Multiplicador 8x8 (Não sinalizado)



Fonte: Modificada de Singh, 2010

Finalmente, o método de multiplicação utilizado na ULA empregando os circuitos multiplicadores 16x16 não sinalizado é mostrado na figura 12. Pode-se notar que este utiliza o multiplicador 8x8, bem como os somadores de 16 e 24 Bits. O resultado final é obtido pelo somador de 24 Bits.

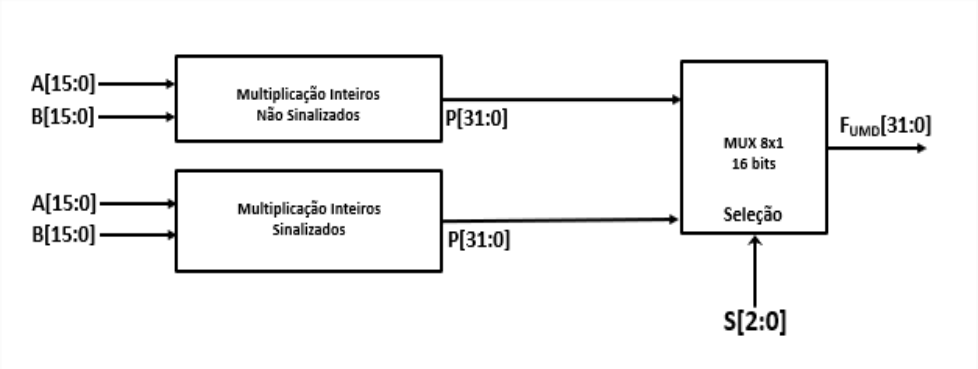
Figura 12- Multiplicador 16x16 (Não sinalizado)



Fonte: Modificada de Singh, 2010

A figura 13 mostra uma visão geral da Unidade de Multiplicação e Divisão de Inteiros Sinalizados e não sinalizados (UMD). Para cada entrada A[15:0] e B[15:0] , a multiplicação é realizada para inteiros sinalizados e não sinalizados e o MUX 8x1 de 16 Bits seleciona a saída conforme os valores de S[2:0].

Figura 13- Unidade Multiplicação e Divisão de Inteiros Sinalizados e Não Sinalizados (UMD)



Fonte: Autores

A implementação da ULA 16 bits foi organizada nas entidades funcionais que a compõem. Cada entidade é representada com seu nome utilizado no código VHDL. Pode-se notar que o projeto é totalmente modular e cada entidade pode ser reutilizada, simulada e verificada individualmente conforme necessário.

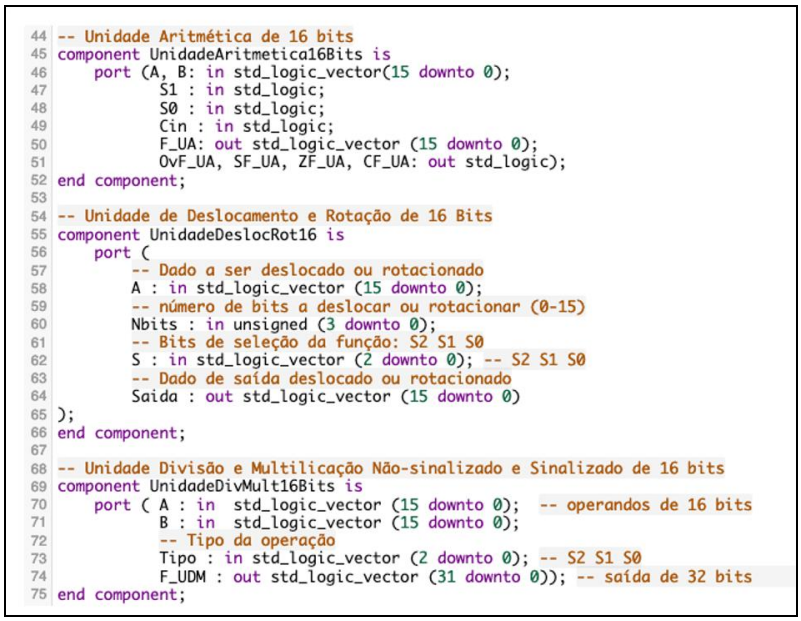
O código da entidade ULA16Bits é mostrado nas figuras 14e15.

Figura 14- Código VHDL da Entidade ULA16bits - Parte 1

```
-- Unidade Lógica e Aritmética de 16 bits
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ULA16Bits is
    port (  S2  : in  std_logic;  -- Códigos de operação: S2 S1 S0 Cin
            S1  : in  std_logic;
            S0  : in  std_logic;
           Cin  : in  std_logic;
            A   : in  std_logic_vector (15 downto 0);  -- operando de 16 bits
            B   : in  std_logic_vector (15 downto 0);  -- operando de 16 bits
          OvF, SF, ZF, CF: out std_logic;  --- sinalizadores (Flags)
            F   : out std_logic_vector (15 downto 0)); -- saída de 16 bits
end ULA16Bits;

architecture arquitetura of ULA16Bits is
-- Multiplexador 4x1 de 16 bits
component MUX4x1 is
    port ( S   : in  std_logic_vector (1 downto 0); -- entrada de seleção - 2 bits
           X   : in  std_logic_vector (15 downto 0); -- entrada 16 bits
           Y   : in  std_logic_vector (15 downto 0); -- entrada 16 bits
           Z   : in  std_logic_vector (15 downto 0); -- entrada 16 bits
           W   : in  std_logic_vector (15 downto 0); -- entrada 16 bits
           F   : out std_logic_vector (15 downto 0));  -- saída 16 bits
end component;

-- Unidade Lógica de 16 bits
component UnidadeLogica16Bits is
    port ( A : in  std_logic_vector (15 downto 0);  -- entrada de 16 bits
           B : in  std_logic_vector (15 downto 0);  -- entrada de 16 bits
           -- Tipo da operação lógica
           Tipo : in std_logic_vector (2 downto 0); -- S2 S1 S0
           F_UL : out std_logic_vector (3 downto 0)); -- saída de 16 bits
end component;
```

Fonte: Autores

Figura 15- Código VHDL da Entidade ULA16bits - Parte 2



```
-- Unidade Aritmética de 16 bits
component UnidadeAritmetica16Bits is
    port (A, B: in std_logic_vector(15 downto 0);
          S1 : in std_logic;
          S0 : in std_logic;
          Cin : in std_logic;
          F_UA: out std_logic_vector (15 downto 0);
          OvF_UA, SF_UA, ZF_UA, CF_UA: out std_logic);
end component;

-- Unidade de Deslocamento e Rotação de 16 Bits
component UnidadeDeslocRot16 is
    port (
        -- Dado a ser deslocado ou rotacionado
        A : in std_logic_vector (15 downto 0);
        -- número de bits a deslocar ou rotacionar (0-15)
        Nbits : in unsigned (3 downto 0);
        -- Bits de seleção da função: S2 S1 S0
        S : in std_logic_vector (2 downto 0); -- S2 S1 S0
        -- Dado de saída deslocado ou rotacionado
        Saida : out std_logic_vector (15 downto 0)
);
end component;

-- Unidade Divisão e Multilicação Não-sinalizado e Sinalizado de 16 bits
component UnidadeDivMult16Bits is
    port ( A : in  std_logic_vector (15 downto 0);  -- operandos de 16 bits
           B : in  std_logic_vector (15 downto 0);
           -- Tipo da operação
           Tipo : in std_logic_vector (2 downto 0); -- S2 S1 S0
           F_UDM : out std_logic_vector (31 downto 0)); -- saída de 32 bits
end component;
```

Fonte: Autores

RESULTADOS E DISCUSSÃO

Neste projeto os casos de teste foram implementados por meio da entidade *testbench*, e realizou-se uma simulação funcional manual onde os atrasos internos do DUT não são considerados e a saída é verificada manualmente, normalmente por inspeção visual das formas de onda.

A ULA implementada neste trabalho realiza operações aritméticas, lógicas e disponibiliza os *flags* que foram afetados ou não no caso de *carry*, *overflow*, sinal e zero/um, além de realizar as operações de deslocamento e rotação de bits e a multiplicação. Desta forma os casos de teste foram divididos em: casos de teste da UA (oito casos) e UL (quatro casos) e casos de teste de *flags* (8 casos), caso de teste de deslocamento e rotação de bits (5 casos) e multiplicação (onze casos).

A tabela 2 mostra para cada caso de teste e as condições, ou seja, valores de entrada (A e B em binário) e a saída F esperada em binário e hexa para cada um dos oito casos da Unidade Aritmética.

Caso de teste	Entradas	Saída esperada
Transfere A F = A	S2 = 0;S1 = 0;S0 = 0 A = 0xABCD; B = 0x1111	F_UA = 0xABCD
Incrementa A F = A + 1	S2 = 0;S1 = 0;S0 = 1 A = 0xABCD; B = 0x1111	F_UA = 0xABCE
Adição F = A + B	S2 = 0;S1 = 1;S0 = 0 A = 0xABCD; B = 0x1111	F_UA = 0xBCDE
Adição com Vai-um F = A + B + 1	S2 = 0;S1 = 1;S0 = 1 A = 0xABCD; B = 0x1111	F_UA = 0xBCDF
Subtração com Vem-um F = A - B - 1	S2 = 1;S1 = 0;S0 = 0 A = 0xABCD; B = 0x1111	F_UA = 0x9ABB
Subtração F = A - B	S2 = 1;S1 = 0;S0 = 1 A = 0xABCD; B = 0x1111	F_UA = 0x9ABC
Decrementa A F = A - 1	S2 = 1;S1 = 1;S0 = 0 A = 0xABCD; B = 0x1111	F_UA = 0xABCC
Transfere A F = A	S2 = 1;S1 = 1;S0 = 1 A = 0xABCD; B = 0x1111	F_UA = 0xABCD

Tabela 2 - Casos de Testes – Unidade Aritmética

Fonte: Autores

Os casos de testes especificados acima foram simulados utilizando as *testbenches*. A figura 17 mostra o código VHDL da *TestBench* da UA.

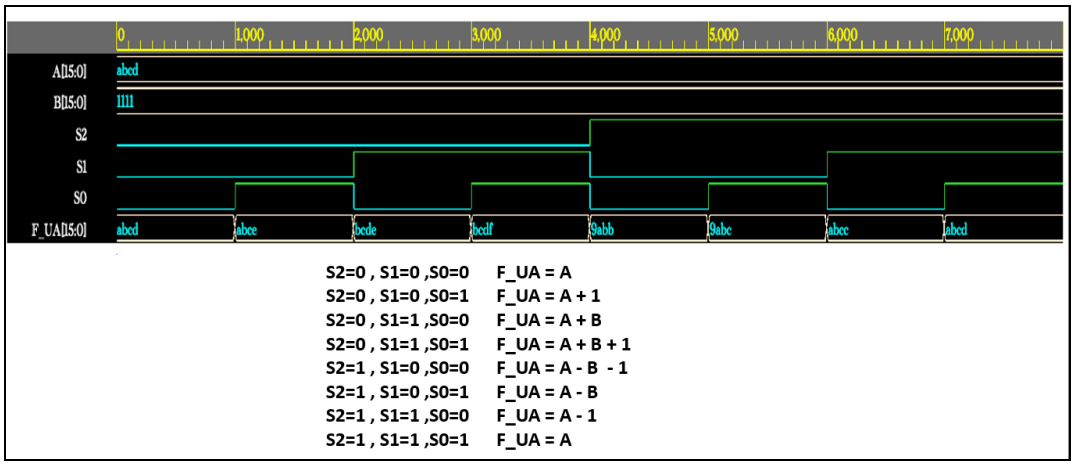
Figura 17- Simulação – *TestBench* – VHDL Casos de Teste – Unidade UA

```
library IEEE;
use IEEE.std_logic_1164.all;

entity testbench is
-- vazio
end testbench;

architecture tb of testbench is
-- DUT component
component UnidadeAritmetica16Bits is
    port (A, B : in std_logic_vector(15 downto 0);
          S2 : in std_logic;
          S1 : in std_logic;
          S0 : in std_logic;
          F_UA : out std_logic_vector (15 downto 0);
          OvF_UA, SF_UA, ZF_UA, CF_UA: out std_logic);
end component;

signal A, B, F_UA: std_logic_vector(15 downto 0);
signal S2, S1, S0 : std_logic;
signal OvF, SF, ZF, CF: std_logic;
constant largura_pulso: time := 1 ns;

begin
  -- Conecta DUT
 DUT: UnidadeAritmetica16Bits
        port map(A,B,S2,S1,S0,F_UA,OvF,SF,ZF,CF);
 process
    begin
      S2 <= '0'; S1 <= '0'; S0 <= '0';   -- 1. F = A
      A <= X"A115";
      B <= X"1000";
      wait for largura_pulso;
      S2 <= '0'; S1 <= '0'; S0 <= '1';   -- 2. F = A +
      A <= X"A115";
      B <= X"5100";
      wait for largura_pulso;

      S2 <= '0'; S1 <= '1'; S0 <= '0';   -- 3. F = A + B
      A <= X"A115";
      B <= X"10AA";
      wait for largura_pulso;

      S2 <= '0'; S1 <= '1'; S0 <= '1';   -- 4. F = A + B + 1
      A <= X"A115";
      B <= X"BF00";
      wait for largura_pulso;

      S2 <= '1'; S1 <= '0'; S0 <= '0';   -- 5. F = A - B - 1
      A <= X"A115";
      B <= X"5FB0";
      wait for largura_pulso;

      S2 <= '1'; S1 <= '0'; S0 <= '1';   -- 6. F = A + B
      A <= X"A115";
      B <= X"0FFF";
      wait for largura_pulso;

      S2 <= '1'; S1 <= '1'; S0 <= '0';   -- 7. F = A - B
      A <= X"A115";
      B <= X"2EE0";
      wait for largura_pulso;

      S2 <= '1'; S1 <= '1'; S0 <= '1';   -- 8. F = A + B
      A <= X"A115";
      B <= X"5EEB";
      wait for largura_pulso;

      wait;    -- suspende indefinidamente o processo
end process;
```

Fonte: Autores

A figura 18 mostra os resultados dos oito casos de testes da unidade aritmética. Os casos de teste foram realizados nos seguintes intervalos:0-1.000ps: F_UA = A;1.000ps-2.000ps: F_UA = A + 1;2.000ps-3.000ps: F_UA = A + B;3.000ps-4.000ps: F_UA = A + B + 1;4.000ps-5.000ps: F_UA = A - B – 1;5.000ps-6.000ps: F_UA = A – B;6.000ps-7.000ps: F_UA = A – 1;7.000ps-8.000ps: F_UA = A.

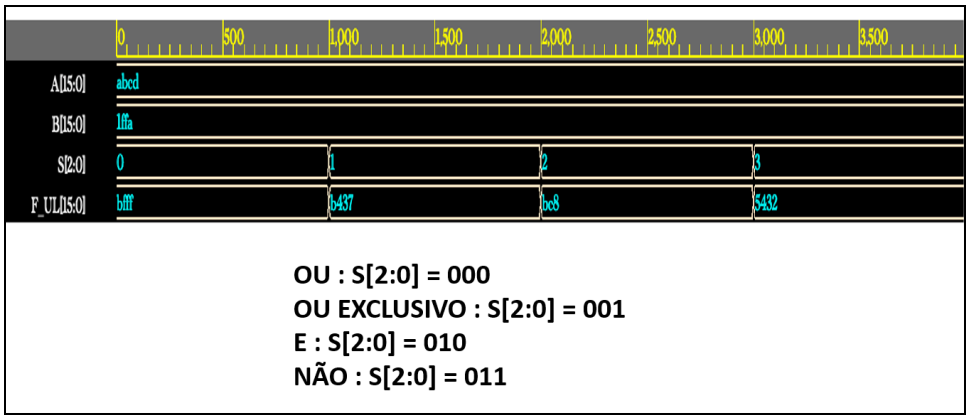
Figura 18- Resultados da Simulação dos Casos de Testes das unidades UA



Fonte: Autores

A figura 19 mostra os resultados dos quatro casos de testes da unidade lógica. Os casos de teste foram realizados nos seguintes intervalos:0-1000ps: F_UL = A or B;1.000ps-2.000ps: F_UL = A xor B;2.000ps-3.000ps: F_UL = A andB;3.000ps-4.000ps: F_UL = not A.

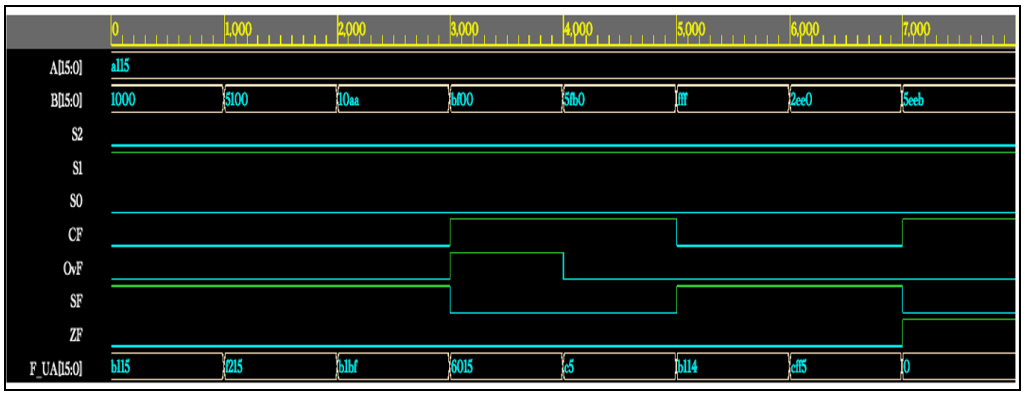
Figura 19- Resultados da Simulação dos Casos de Testes da UL



Fonte: Autores

A figura 20mostra os resultados dos oito casos de testes de *flags*. Os casos de teste foram realizados nos seguintes intervalos:0-1.000ps: F_UA = A + B, CF = 0, OvF = 0, SF = 1, ZF = 0;1.000ps-2.000ps: F_UA = A + B, CF = 0, OvF = 0, SF = 1, ZF = 0;2.000ps-3.000ps: F_UA = A + B, CF = 0, OvF = 0, SF = 1, ZF = 0;3.000ps-4.000ps: F_UA = A + B, CF = 1, OvF = 1, SF = 0, ZF = 0;4.000ps-5.000ps: F_UA = A + B, CF = 1, OvF = 0, SF = 0, ZF = 0; 5.000ps-6.000ps: F_UA = A + B, CF = 0, OvF = 0, SF = 1, ZF = 0; 6.000ps-7.000ps: F_UA = A + B, CF = 0, OvF = 0, SF = 1, ZF = 0; 7.000ps-8.000ps: F_UA = A + B, CF = 1, OvF = 0, SF = 0, ZF = 1.

Figura 20 -Resultados da Simulação dos Casos de Testes de *flags*



Fonte: Autores

CONSIDERAÇÕES FINAIS

A ULA executa operações aritmética e lógica em um ou mais valores quando solicitada pelo processador por meio da unidade de controle que passa os códigos de operação que a ULA deverá realizar.

Neste trabalho, que se propunha a demonstrar o funcionamento da ULA de 16 Bits em linguagem VHDL, o projeto foi dividido em quatro etapas.Primeira Etapa: o projeto da unidade lógica; Segunda Etapa: projeto da unidade aritmética;Terceira Etapa: o projeto da unidade de unidade rotação e deslocamento de Bits;Quarta Etapa: o projeto da unidade multiplicação de inteiros não sinalizados e sinalizados.

A ULA foi projetada com oito operações aritméticas, quatro operações lógicas, cinco operações de deslocamento e rotação de bits, uma operação de multiplicação de inteiros não sinalizados e outra de inteiros sinalizados.

Para a implementação em linguagem VHDL utilizou-se a ferramenta EDA Playground, que é uma ferramenta de ambiente online para simular e sintetizar implementações no nosso caso em linguagem VHDL.

Considerando os resultados da simulação, pode-se observar que a ULA de 16 bits implementada pelo método e código descritos teve o funcionamento previsto para todas as combinações de entrada e códigos de operação com a correspondência exata entre os valores esperados na saída das tabelas dos casos de teste e os valores de saída obtidos na simulação.

Pode-se estender este projeto com outros assuntos não abordados, tais como o projeto e implementação em VHDL do circuito divisor de inteiros não sinalizados e sinalizados para complementar a unidade funcional UDM, a sintetização do código da ULA de 16 bits e teste em um dispositivo lógico programável como FPGA.

REFERÊNCIAS

EDA Playground. Disponível em <https://edaplayground.com/>. Acesso em: 27 de ago. de 2021.

OLIVEIRA, A. F.; OLIVEIRA, E. M.**Implementação de uma unidade lógica e aritmética de 16 bits com as operações de multiplicação, deslocamento e**

rotação de bits em VHDL. 2021. Monografia (Bacharelado em Engenharia Elétrica), Universidade do Estado de Minas Gerais – UEMG, Campus Ituiutaba, 2021.

PEDRONI, Volnei A. **Circuit design and simulation with VHDL.** 2nd ed. Cambridge: MIT Press, 2010.

SARANGI, Saumyakanta et al. **VHDL Implementation of Arithmetic Logic Unit**. International Journal of Engineering Research & Technology (IJERT). Vol. 3 Issue 4, April – 2014. ISSN: 2278-0181. Disponívelem<ijert.org/vhdl-implementation-of-arithmetic-logic-unit >. Acesso em: 27 de ago. 2021.

SINGH, A. **Design and hardware realization of a 16 Bit vedic arithmetic unit**. 2010. 65 p. Dissertação (Mestrado em tecnologia) - Department of Eletronics and Communication Engeneering, Thapar University, Patiala, India, 2010. Disponível em http://tudr.thapar.edu:8080/jspui/bitstream/10266/1109/4/1109.pdf>. Acesso em: 27 de ago. 2021.

TAIB, Muhammad Ikmal Mohd et al. **Design of Multiplication and Division Operation for 16 Bit Arithmetic Logic Unit (ULA)**. Journal Of Electronic Voltage and Application. Vol. 1 No. 2 (2020) 46-54. Disponível em: https: publisher. uthm. edu. my / ojs / index.php / jeva / article / download / 7219 / 3987 /. Acesso em: 27 de ago. 2021.

VAHID, F. **Sistemas digitais: projeto, otimização e HDLs**. Tradução Anatólio Laschuk. Porto Alegre: Artmed, 2008. 560 p.

AUTORES

ANDRESSA FERREIRA DE OLIVEIRA, graduanda do Curso de Engenharia Elétrica na Universidade do Estado de Minas Gerais – UEMG, Unidade Ituiutaba. E-mail: andressa.1501300@discente.uemg.br .
ELIAN MIGUEL DE OLIVEIRA, graduando do Curso de Engenharia Elétrica na Universidade do Estado de Minas Gerais –UEMG, Unidade Ituiutaba. E-mail: andressa.1501300@discente.uemg.br .

MAURO HEMERLY GAZZANI, doutor em Engenharia Elétrica pela Universidade Federal de Uberlândia. Bacharel em Engenharia Elétrica pela Universidade Federal de Uberlândia. Professor do Curso de Graduação em Engenharia Elétrica, da Universidade do Estado de Minas Gerais – UEMG, Unidade Ituiutaba. E-mail: maur.gazzani@uemg.br.

KÁTIA LOPES SILVA, Docteuren Sciences Appliqué es pela Université de Liège. Bacharel em Engenharia Química pela Universidade Federal de Uberlândia. Professor do Curso de Graduação em Engenharia Elétrica, da Universidade do Estado de Minas Gerais – UEMG, Unidade Ituiutaba. E-mail: katia.lopes@uemg.br.