UNIVERSIDADE FEDERAL DO PARANÁ

ELOIZA ARANDA BRUSCHI GRR20212721
MARIA PAULA BASTOS GRR20212762

RELATÓRIO LABORATÓRIO 2: PROJETOS COM CÓDIGO CONCORRENTE

CURITIBA
2024

ELOIZA ARANDA BRUSCHI GRR20212721
MARIA PAULA BASTOS GRR20212762

RELATÓRIO LABORATÓRIO 2: PROJETOS COM CÓDIGO CONCORRENTE

Relatório apresentada à disciplina de Microeletrônica, Setor de Tecnologia da Universidade Federal do Paraná, como atividade avaliativa.
Professora: Dra. Sibilla Batista da Luz França.

CURITIBA
2024

1 INTRODUÇÃO

Este trabalho apresenta a implementação de quatro projetos distintos no âmbito da disciplina de Microeletrônica I, utilizando a ferramenta ISE e a placa de desenvolvimento Nexys 2. O primeiro projeto consiste na criação de um decodificador BCD para um display de 7 segmentos, onde a ênfase recai sobre a simulação e a implementação prática no kit. O segundo projeto aborda o cálculo do peso de Hamming de um vetor de bits, adaptando o design do primeiro para atender à especificação de entradas e saídas do hardware disponível. O terceiro projeto foca na ordenação de bits em um vetor, utilizando conceitos de contagem e manipulação de sinais. Por fim, o quarto projeto implementa uma mini Unidade Lógica Aritmética (ULA) que realiza operações aritméticas definidas por um código de operação.

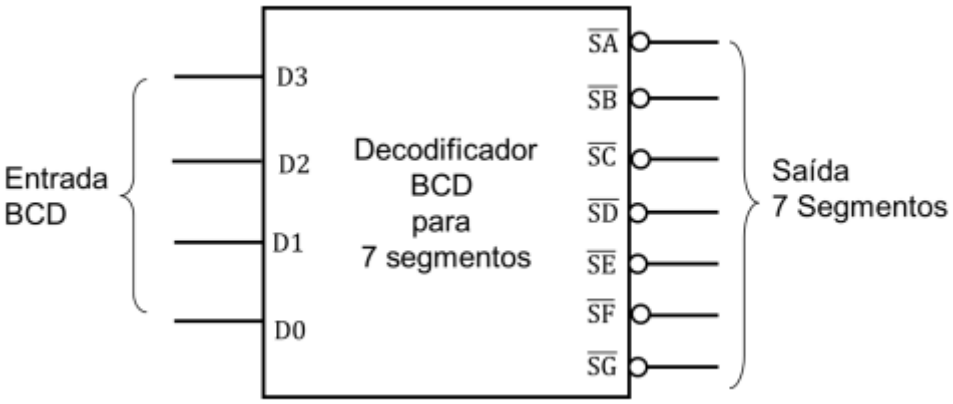
1.1 VHDL

Neste trabalho, utilizamos a plataforma ISE 14.7 da Xilinx para implementar o VHDL (VHSIC Hardware Description Language), uma linguagem de descrição de hardware (HDL) que possibilita a representação de sistemas eletrônicos digitais em diversos níveis de abstração, como RTL e gate-level. Essa flexibilidade torna a modelagem e simulação de circuitos complexos mais acessível. Além disso, o VHDL suporta processos concorrentes e sequenciais, essenciais para a representação de sistemas digitais com múltiplos sinais e eventos simultâneos.

2 PROJETO 1- DECODIFICADOR BCD PARA 7 SEGMENTOS

Utilizou-se a ferramenta ISE para implementar o que foi solicitado no projeto 1, ou seja, um decodificador BCD para 7 segmentos.

FIGURA 1 – ESQUEMÁTICO CONCEITUAL DO PROJETO 1



FONTE: ‘LAB 1’, Sibilla França (20??).

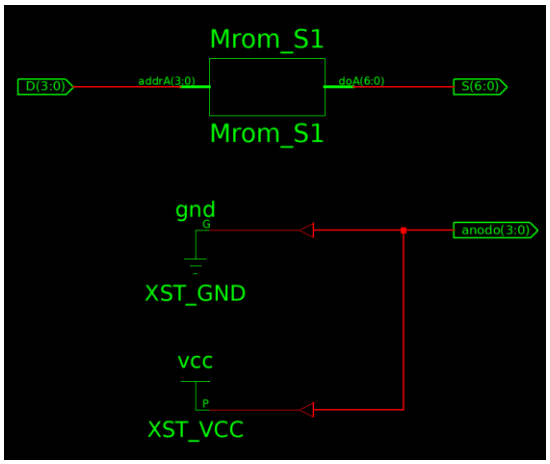
O primeiro projeto consiste na implementação de um decodificador BCD (Binary-Coded Decimal) para um display de 7 segmentos, uma tarefa fundamental em circuitos digitais. Este projeto envolve a programação em VHDL, utilizando apenas código concorrente, nesse caso foi utilizado o código WHEN.

A implementação prática foi realizada no kit Nexys 2, onde os switches foram utilizados como entradas do circuito e um display de 7 segmentos serviu como saída. Foi necessário mapear o ânodo do display selecionado

2.1 ESQUEMÁTICOS RTL

Esquemáticos RTL são maneiras de visualizar a entrada e saída do projeto de forma ilustrada. O esquemático RTL na figura abaixo obtido após a síntese mostra a arquitetura lógica do circuito, confirmando a implementação da lógica de decodificação BCD.

FIGURA 2 – ESQUEMÁTICO RTL DO PROJETO 1



FONTE: As autoras (2024).

Apresentando a tabela verdade em anodo comum na tabela abaixo, que vai ser uma informação importante para a análise dos resultados da simulação:

Tabela Verdade para Display de 7 Segmentos (Anodo Comum)

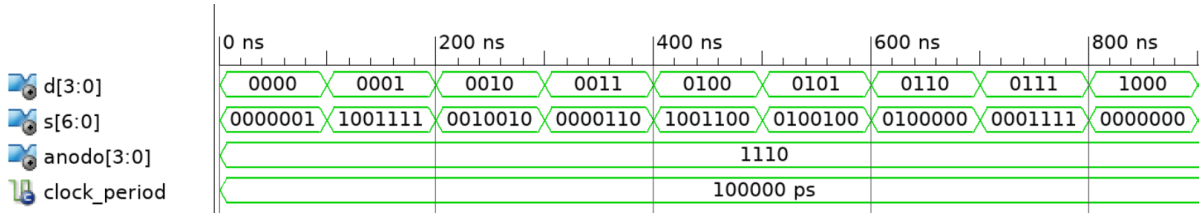
Dígito	a	b	c	d	e	f	g	Representação binária
0	0	0	0	0	0	0	1	`1111110`
1	1	0	0	1	1	1	1	`0110000`
2	0	0	1	0	0	1	0	`1101101`
3	0	0	0	0	1	1	0	`1111001`
4	1	0	0	1	1	0	0	`0110011`
5	0	1	0	0	1	0	0	`1011011`
6	0	1	0	0	0	0	0	`1011111`
7	0	0	0	1	1	1	1	`1110000`
8	0	0	0	0	0	0	0	`1111111`
9	0	0	0	0	1	0	0	`1111011`

2.2 SIMULAÇÕES

.

Uma simulação comportamental (Anexo 2) foi realizada para verificar o funcionamento do decodificador. O *test bench* aplicou todos os valores BCD (0 a 9) como entrada (D), sequencialmente. Os resultados da simulação demonstraram a correta conversão para cada valor de entrada.

FIGURA 3 – SIMULAÇÃO DO PROJETO 1



FONTE: As autoras (2024).

2.3 CÓDIGO COMENTADO

O código comentado é dividido em duas partes, no código de fato da placa física e no código para simular, com as mudanças que seriam feitas se fosse possível testar na placa.

2.3.1 VHDL Module

O código VHDL Module, do projeto 1, está apresentado sem alterações no Anexo 1 deste presente trabalho.

O código VHDL (Anexo 1) define uma entidade dec_bcd_sete_seg com uma entrada D (4 bits, representando o valor BCD) e saídas S (7 bits, representando os segmentos do display) e anodo (4 bits, para controle de ânodo do display). A arquitetura Behavioral usa a instrução WHEN...ELSE para definir a lógica de decodificação: para cada valor possível de D, o valor correspondente de S é atribuído assim como a variável anodo é definida de forma constante no código.

2.3.2 VHDL Test Bench

O código VHDL Test Bench, do projeto 1, está apresentado sem alterações no Anexo 2 deste presente trabalho.

3 PROJETO 2- PESO DE HAMMING

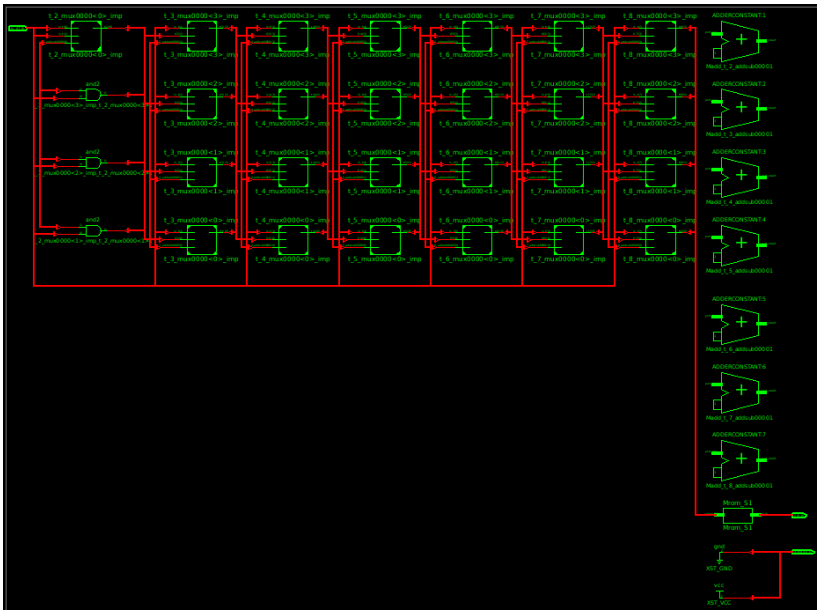
O segundo projeto teve como objetivo a criação de um circuito capaz de determinar o peso de Hamming de um vetor de comprimento genérico. O peso de Hamming é definido como o número de bits '1' presentes em um vetor binário, sendo uma métrica importante em várias aplicações de teoria da informação e codificação.

Para a implementação prática no kit Nexys 2, foi definido N=8, utilizando switches como entradas do circuito e um display de 7 segmentos para mostrar o valor da saída. Para esta parte, foi reutilizado o decodificador BCD desenvolvido no projeto anterior, promovendo uma integração eficiente entre os dois projetos.

3.1 ESQUEMÁTICOS RTL

O esquemático RTL do projeto 2 está apresentado na figura abaixo, e mostra que as entradas e saídas do projeto. Ele apresenta os 8 bits de entrada dos switches (in_t), a lógica de cálculo do peso de Hamming (que utiliza um contador), a conversão do resultado para BCD, a interface com o decodificador BCD (do Projeto 1), e, finalmente, a saída para o display de 7 segmentos (S). A saída anodo do decodificador, como no projeto anterior,

FIGURA 4 – ESQUEMÁTICO RTL DO PROJETO 2

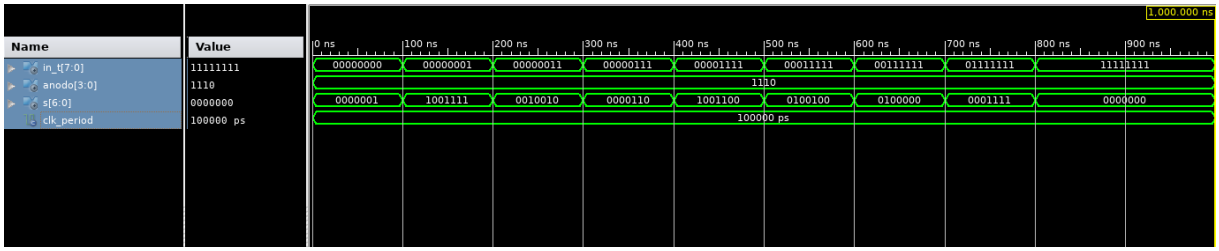


FONTE: As autoras (2024).

3.2 SIMULAÇÕES

O *test bench* (Anexo 4), como descrito no relatório, testa o circuito com diferentes vetores de entrada. Sua função é simular o comportamento do circuito Peso_Hamming em várias condições para verificar sua funcionalidade. O *test bench* aplica diversos valores de entrada (in_t), variando de "00000000" (todos zeros) até "11111111" (todos uns), e observa o resultado na saída (S). Quando comparada com a tabela verdade dos 7 segmentos verificamos que o funcionamento é o desejado, sendo correspondente ao número esperado de uns apresentados no display.

FIGURA 5 – SIMULAÇÃO DO PROJETO 2



FONTE: As autoras (2024).

3.3 CÓDIGO

Nessa seção será apresentado não apenas o código do VHDL Module, mas também o código realizado para que a simulação seja possível.

3.3.1 VHDL Module

O código VHDL Module, do projeto 2, está apresentado sem alterações no Anexo 3 deste presente trabalho.

O código VHDL implementa um contador concorrente para determinar o peso de Hamming de um vetor binário. Um laço for generate itera simultaneamente sobre cada bit do vetor de entrada (in_t). Para cada bit, se o valor for '1', um contador interno (t) é incrementado. Após a iteração, o valor final do contador (final_t) representa o peso de Hamming. Este valor é então convertido para um vetor BCD de 4 bits (D) utilizando a função to_unsigned, e este vetor BCD é usado como entrada para um decodificador BCD (implementado separadamente) para acionar um display de 7 segmentos, com o sinal anodo. A saída final (S) representa o padrão de ativação dos segmentos do display de 7 segmentos, exibindo o peso de Hamming calculado.

3.3.2 VHDL Test Bench

O código VHDL do Test Bench, referente ao projeto 2, é apresentado sem modificações no Anexo 4 deste trabalho.

4 PROJETO 3- ORDENADOR BINÁRIO COM GENERATE

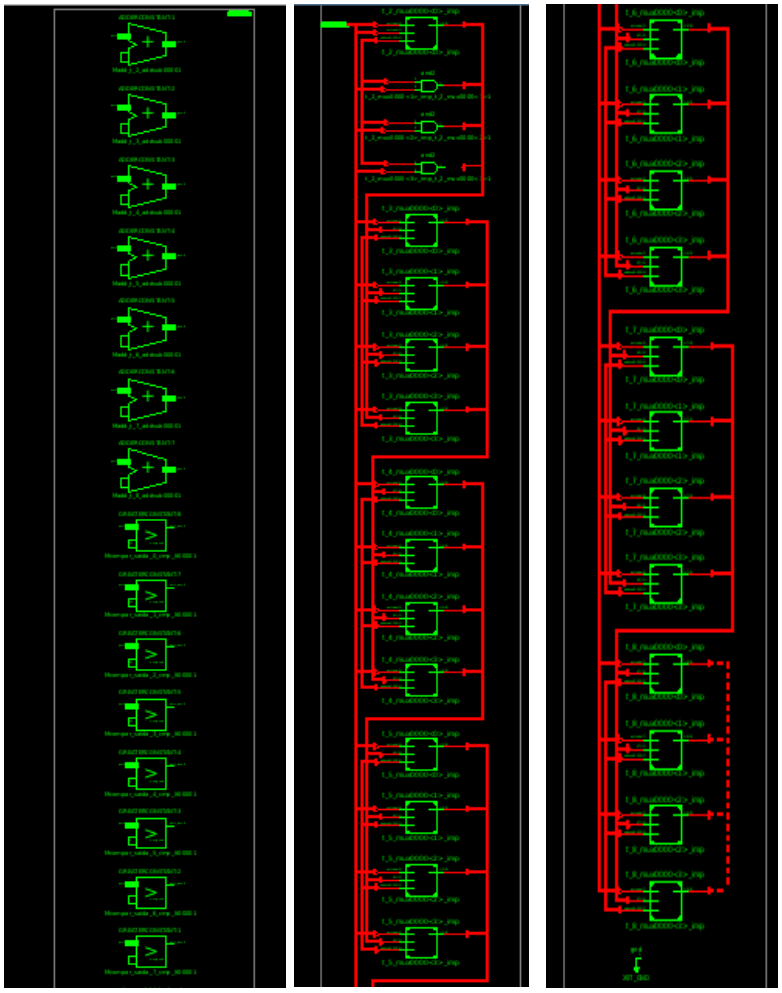
Este projeto tem como objetivo projetar e implementar um circuito que ordena os bits de um vetor binário de comprimento genérico, colocando todos os bits '1' à esquerda, seguidos pelos bits '0'. A ordenação é feita da esquerda para a direita. A implementação utiliza exclusivamente código concorrente em VHDL.

Para a implementação no kit Nexys 2, o tamanho do vetor foi definido como N=8. Os 8 switches da placa foram utilizados para definir a entrada (entrada), e os 8 LEDs da placa foram usados para representar a saída ordenada (saida).

4.1 ESQUEMÁTICOS RTL

O esquemático RTL do projeto 2 está apresentado na figura abaixo, e mostra que as entradas e saídas do projeto.

FIGURA 6 – ESQUEMÁTICO RTL DO PROJETO 3

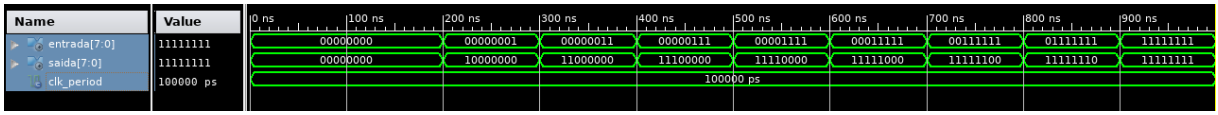


FONTE: As autoras (2024).

4.2 SIMULAÇÕES

O *test bench* (Anexo 6) simula o circuito ordenador_bin, aplicando uma sequência de vetores de entrada (entrada) ao componente em teste (uut). Para cada vetor de entrada, o *test bench* espera um tempo definido por clk_period antes de aplicar o próximo vetor. Após a aplicação de cada vetor, o *test bench* observa a saída (saida) gerada pelo circuito, verificando que a quantidade de uns é colocada na esquerda da onda sendo o comportamento esperado.

FIGURA 7 – SIMULAÇÃO DO PROJETO 3



FONTE: As autoras (2024).

4.3 CÓDIGO

Nessa seção será apresentado não apenas o código do VHDL Module, mas também o código realizado para que a simulação seja possível.

4.3.1 VHDL Module

O código VHDL Module, do projeto 3, está apresentado sem alterações no Anexo 5 deste presente trabalho.

O código VHDL do módulo principal (Anexo 5) implementa um circuito que ordena um vetor binário de entrada (entrada) colocando todos os bits '1' à esquerda, seguidos pelos bits '0'. Ele primeiro calcula o peso de Hamming do vetor de entrada usando um laço for generate para contagem concorrente dos bits '1'. Em seguida, outro laço for generate gera a saída (saida) atribuindo '1' aos primeiros final_t bits (onde final_t é o peso de Hamming) e '0' aos bits restantes.

4.3.2 VHDL Test Bench

O código VHDL do Test Bench, referente ao projeto 3, é apresentado sem modificações no Anexo 6 deste trabalho.

5 PROJETO 4- CIRCUITO ARITMÉTICO COM STD_LOGIC

Projetar e implementar em VHDL um circuito aritmético (mini ULA) capaz de realizar diferentes operações aritméticas (adição, subtração etc.) com base em um código de operação (opcode). O circuito deve ser implementado utilizando exclusivamente código concorrente em VHDL, com o número de bits das entradas e saídas definido por um parâmetro genérico.

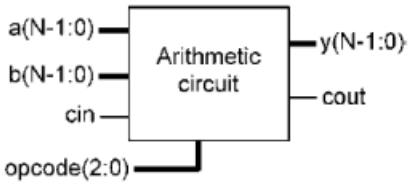


Figura 1(a)

Tipo	Operação	Opcode
Unsigned	y = a + b	000
	y = a - b	001
	y = - a + b	010
	y = a + b + cin	011
Signed	y = a + b	100
	y = a - b	101
	y = - a + b	110
	y = a + b + cin	111

Figura 1(b)

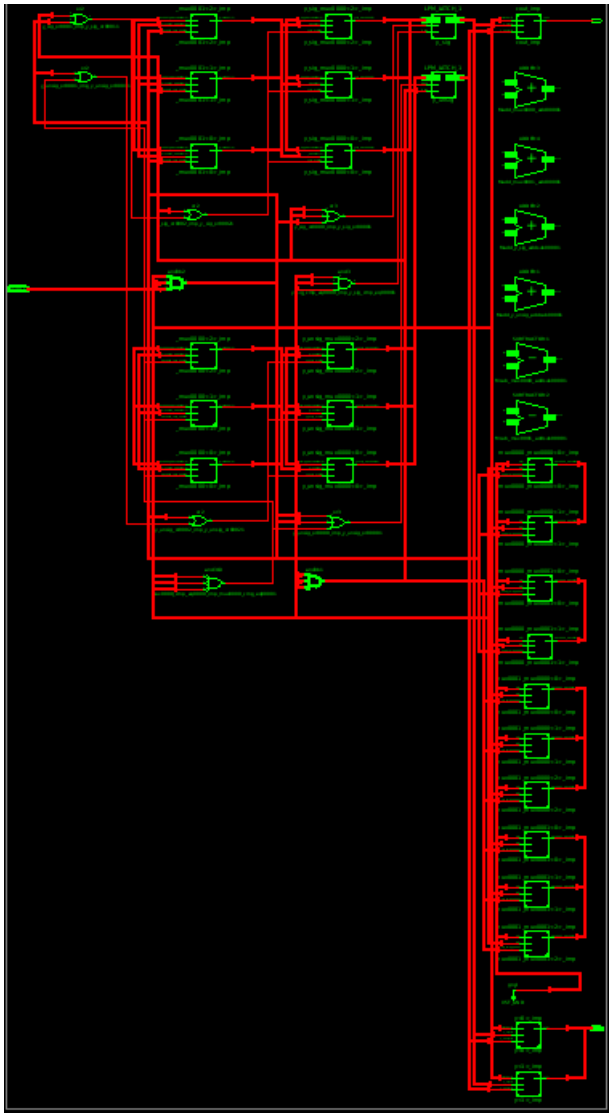
Figura 3 - Mini ULA.

O circuito aritmético utiliza um multiplexor para selecionar a operação aritmética a ser realizada, com base no código de operação (opcode). As operações suportadas são adição, subtração e adição com carry-in, tanto para números sem sinal (Unsigned) quanto para números com sinal (Signed). A seleção da operação é feita através de uma estrutura CASE (ou WHEN...ELSE implícita) no código VHDL.

5.1 ESQUEMÁTICOS RTL

O esquemático RTL do projeto 4 está apresentado na figura abaixo, e mostra que as entradas e saídas do projeto. O diagrama RTL mostra a arquitetura do circuito. Ele ilustra o bloco "Arithmetic circuit", as entradas (a, b, cin, opcode), e as saídas (y, cout).

FIGURA 8 – ESQUEMÁTICO RTL DO PROJETO 4

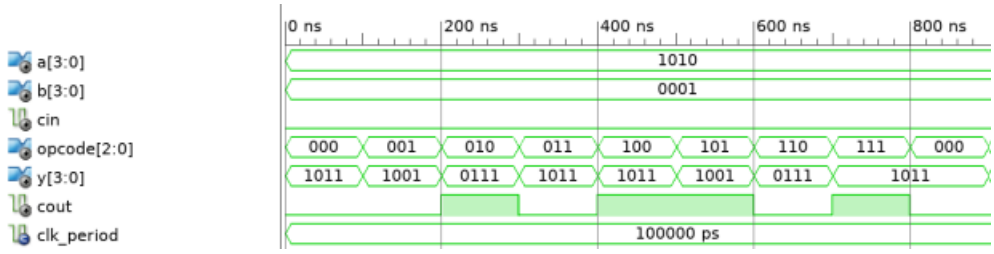


FONTE: As autoras (2024).

5.2 SIMULAÇÕES

Para a simulação (Anexo 8), o tamanho do vetor foi definido como N=4. O *test bench* aplicou diferentes combinações de entradas (a, b, cin, opcode) para testar todas as operações em diversos cenários.

FIGURA 9 – SIMULAÇÃO DO PROJETO 2



FONTE: As autoras (2024).

5.3 CÓDIGO

Nessa seção será apresentado não apenas o código do VHDL Module, mas também o código realizado para que a simulação seja possível.

5.3.1 VHDL Module

O código VHDL Module, do projeto 4, está apresentado sem alterações no Anexo 3 deste presente trabalho.

O código VHDL (Anexo 7) define uma entidade Circuito_aritmetico com um parâmetro genérico N (número de bits das entradas a e b e da saída y), uma entrada cin (carry-in), e uma entrada opcode (3 bits) para selecionar a operação. A saída y representa o resultado da operação aritmética, e cout representa o carry-out. A lógica do circuito utiliza sinais auxiliares (a_sig, b_sig, a_unsig, b_unsig, y_sig, y_unsig) para trabalhar com números com e sem sinal. Um bloco CASE (ou WHEN...ELSE implícito) seleciona a operação a ser executada com base no valor de opcode, realizando a operação aritmética correspondente, considerando também se os números são com ou sem sinal. A saída (y e cout) é então gerada.

5.3.2 VHDL Test Bench

O código VHDL do Test Bench, referente ao projeto 4, é apresentado sem modificações no Anexo 8 deste trabalho.

O *test bench* (Anexo 8) instancia o componente Circuito_aritmetico e, no processo stim_proc, aplica sequencialmente todos os oito códigos de operação (opcode, de "000" a "111") à entrada do componente, mantendo os valores de entrada a e b constantes ("1010" e "0001", respectivamente) e cin em '0'. Após aplicar cada código de operação, o *test bench* espera um período (clk_period), antes de passar para o próximo código de operação. A saída (y e cout) do componente é observada para cada opcode

CONCLUSÃO

Este trabalho apresentou a implementação de quatro projetos distintos em VHDL, utilizando a ferramenta ISE e a placa de desenvolvimento Nexys 2, demonstrando a aplicação prática da linguagem de descrição de hardware na concepção de circuitos digitais. Os projetos abrangeram diferentes níveis de complexidade e funcionalidades, permitindo uma compreensão abrangente das capacidades do VHDL.

O Projeto 1, um decodificador BCD para display de 7 segmentos, demonstrou a implementação de lógica combinacional utilizando código concorrente. O Projeto 2, o cálculo do peso de Hamming, expandiu a aplicação do código concorrente, utilizando um laço for generate para processamento paralelo eficiente. O Projeto 3, a ordenação de bits, mostrou a capacidade do VHDL para implementar algoritmos mais complexos, utilizando conceitos de contagem e manipulação de sinais, novamente com código concorrente. Finalmente, o Projeto 4, a mini ULA, demonstrou a implementação de um circuito aritmético com capacidade de realizar diferentes operações aritméticas (adição e subtração com e sem carry-in, com e sem sinal), utilizando um bloco CASE (ou WHEN...ELSE implícito) para seleção da operação e mostrando o uso de genéricos para flexibilidade no número de bits.

REFERÊNCIAS

PEDRONI, V. A. Circuit Design with VHDL. 2 ed. MIT Press, 2010.

ANEXO 1 – VHDL MODULE – PROJETO 1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity dec_bcd_sete_seg is
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           S : out STD_LOGIC_VECTOR (6 downto 0);
           anodo: out STD_logic_vector (3 downto 0));
end dec_bcd_sete_seg;

architecture Behavioral of dec_bcd_sete_seg is

begin
    anodo <= "1110";

    S <=  "0000001" WHEN D="0000" ELSE
       "1001111" WHEN D="0001" ELSE
       "0010010" WHEN D="0010" ELSE
       "0000110" WHEN D="0011" ELSE
       "1001100" WHEN D="0100" ELSE
       "0100100" WHEN D="0101" ELSE
       "0100000" WHEN D="0110" ELSE
       "0001111" WHEN D="0111" ELSE
       "0000000" WHEN D="1000" ELSE
       "0001100" WHEN D="1001" ELSE
       "1111111" ;


end Behavioral;
```

ANEXO 2 – VHDL TEST BENCH – PROJETO 1

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY dec_bcd_sete_seg_tb IS
END dec_bcd_sete_seg_tb;

ARCHITECTURE behavior OF dec_bcd_sete_seg_tb IS
    COMPONENT dec_bcd_sete_seg
    PORT(
         D : IN  std_logic_vector(3 downto 0);
         S : OUT std_logic_vector(6 downto 0);
         anodo : OUT std_logic_vector(3 downto 0)
        );
    END COMPONENT;

    -- Inputs
    signal D : std_logic_vector(3 downto 0) := (others => '0');
    -- Outputs
    signal S : std_logic_vector(6 downto 0)


 signal anodo : std_logic_vector(3 downto 0);

constant clock_period : time := 0.1 us;

BEGIN

    uut: dec_bcd_sete_seg PORT MAP (
          D => D,
          S => S,
          anodo => anodo
        );

    stimulus_process : process
    begin
        D <= "0000";  -- 0
        wait for clock_period;

        D <= "0001";  -- 1
        wait for clock_period;
```

```
        D <= "0010";  -- 2
        wait for clock_period;

        D <= "0011";  -- 3
        wait for clock_period;

        D <= "0100";  -- 4
        wait for clock_period;

        D <= "0101";  -- 5
        wait for clock_period;

        D <= "0110";  -- 6
        wait for clock_period;

        D <= "0111";  -- 7
        wait for clock_period;

        D <= "1000";  -- 8
        wait for clock_period;

        D <= "1001";  -- 9
        wait for clock_period;

        wait;
    end process;

END behavior;
```

ANEXO 3 – VHDL MODULE – PROJETO 2

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity Peso_Hamming is
    generic (N : integer := 8);
    Port (
        in_t : in  STD_LOGIC_VECTOR(N-1 downto 0);
               anodo: out STD_logic_vector (3 downto 0);
               S : out  STD_LOGIC_VECTOR (6 downto 0)
    );
end Peso_Hamming;
architecture Behavioral of Peso_Hamming is
    type int_vector is array (0 to N) of integer range 0 to N+1;
    signal t : int_vector;
       signal D : STD_logic_vector (3 downto 0);
    signal final_t : integer range 0 to N+1;
begin
    t(0) <= 0;
    gen: for i in 1 to N generate
        t(i) <= t(i-1) + 1 when in_t(i-1) = '1' else t(i-1);
    end generate;
    final_t <= t(N);

    D <= STD_LOGIC_VECTOR(to_unsigned(final_t, 4));
       anodo <= "1110";
       S <=  "0000001" WHEN D="0000" ELSE
        "1001111" WHEN D="0001" ELSE
        "0010010" WHEN D="0010" ELSE
        "0000110" WHEN D="0011" ELSE
        "1001100" WHEN D="0100" ELSE
        "0100100" WHEN D="0101" ELSE
        "0100000" WHEN D="0110" ELSE
        "0001111" WHEN D="0111" ELSE
        "0000000" WHEN D="1000" ELSE
        "0001100" WHEN D="1001" ELSE
        "1111111" ;
```

```
end Behavioral;
```

ANEXO 4 – VHDL TEST BENCH – PROJETO 2

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb IS
END tb;

ARCHITECTURE behavior OF tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Peso_Hamming
    PORT(
         in_t : IN  std_logic_vector(7 downto 0);
         anodo : OUT  std_logic_vector(3 downto 0);
         S : OUT  std_logic_vector(6 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal in_t : std_logic_vector(7 downto 0) := (others => '0');

      --Outputs
   signal anodo : std_logic_vector(3 downto 0);
   signal S : std_logic_vector(6 downto 0);
   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name

   constant clk_period : time := 0.1 us;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: Peso_Hamming PORT MAP (
          in_t => in_t,
          anodo => anodo,
          S => S
```

```
    );
-- Stimulus process
stim_proc: process
begin
   -- Nenhum bit 1, peso de Hamming deve ser 0
    in_t <= "00000000";
    wait for clk_period;

    -- Um bit 1, peso de Hamming deve ser 1
    in_t <= "00000001";
    wait for clk_period;

    -- Dois bits 1, peso de Hamming deve ser 2
    in_t <= "00000011";
    wait for clk_period;

    -- Três bits 1, peso de Hamming deve ser 3
    in_t <= "00000111";
    wait for clk_period;

    -- Quatro bits 1, peso de Hamming deve ser 4
    in_t <= "00001111";
    wait for clk_period;

    -- Cinco bits 1, peso de Hamming deve ser 5
    in_t <= "00011111";
    wait for clk_period;

    -- Seis bits 1, peso de Hamming deve ser 6
    in_t <= "00111111";
    wait for clk_period;

    -- Sete bits 1, peso de Hamming deve ser 7
    in_t <= "01111111";
    wait for clk_period;

    -- Oito bits 1, peso de Hamming deve ser 8
    in_t <= "11111111";
    wait for clk_period
```

```
        wait;
    end process;

END;
```

ANEXO 5 – VHDL MODULE – PROJETO 2

```
        library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ordenador_bin is
       generic (N : integer := 8);
    Port ( entrada : in  STD_LOGIC_VECTOR (N-1 downto 0);
           saida : out  STD_LOGIC_VECTOR (N-1 downto 0) := (others => '0'));
end ordenador_bin;

architecture Behavioral of ordenador_bin is
        type int_vector is array (0 to N) of integer range 0 to N+1;
    signal t : int_vector;
    signal final_t : integer range 0 to N+1;

begin
       t(0) <= 0;
   gen: for i in 1 to N generate
        t(i) <= t(i-1) + 1 when entrada(i-1) = '1' else t(i-1);
    end generate;

    final_t <= t(N);

        gene: for i in N-1 downto 0 generate
       saida(N-i-1) <= '1' when i < final_t else '0';
    end generate;

end Behavioral;
```

ANEXO 6 – VHDL TEST BENCH – PROJETO 2

```
--------------------------------------------------------------------------------
------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY ordenador_bin_tb IS
END ordenador_bin_tb
ARCHITECTURE behavior OF ordenador_bin_tb IS
    COMPONENT ordenador_bin
    PORT(
         entrada : IN  std_logic_vector(7 downto 0);
         saida : OUT  std_logic_vector(7 downto 0)
        );
    END COMPONENT;
   signal entrada : std_logic_vector(7 downto 0) := (others => '0');
   signal saida : std_logic_vector(7 downto 0);

   constant clk_period: time := 100 ns;

BEGIN

   uut: ordenador_bin PORT MAP (
          entrada => entrada,
          saida => saida
        );
   stim_proc: process
   begin
      wait for 100 ns;

      -- insert stimulus here
                -- Nenhum bit 1, peso de Hamming deve ser 0
        entrada <= "00000000";
        wait for clk_period;

        -- Um bit 1, peso de Hamming deve ser 1
```

```
        entrada <= "00000001";
        wait for clk_period;

        -- Dois bits 1, peso de Hamming deve ser 2
        entrada <= "00000011";
        wait for clk_period;

        -- Três bits 1, peso de Hamming deve ser 3
        entrada <= "00000111";
        wait for clk_period;

        -- Quatro bits 1, peso de Hamming deve ser 4
        entrada <= "00001111";
        wait for clk_period;

        -- Cinco bits 1, peso de Hamming deve ser 5
        entrada <= "00011111";
        wait for clk_period;

        -- Seis bits 1, peso de Hamming deve ser 6
        entrada <= "00111111";
        wait for clk_period;

        -- Sete bits 1, peso de Hamming deve ser 7
        entrada <= "01111111";
        wait for clk_period;

        -- Oito bits 1, peso de Hamming deve ser 8
        entrada <= "11111111";
        wait for clk_period;

      wait;
   end process;

END;
```

ANEXO 7 – VHDL MODULE – PROJETO 2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity Circuito_aritmetico is
generic(N: integer:=2)
    Port ( a : in  STD_LOGIC_VECTOR (N-1 downto 0);
           b : in  STD_LOGIC_VECTOR (N-1 downto 0);
           cin : in  STD_LOGIC;
           opcode : in  STD_LOGIC_VECTOR (2 downto 0);
           y : out  STD_LOGIC_VECTOR (N-1 downto 0);
           cout : out  STD_LOGIC);
end Circuito_aritmetico;

architecture Behavioral of Circuito_aritmetico is
      signal a_sig , b_sig: signed (N-1 downto 0);
      signal a_unsig, b_unsig: unsigned (N-1 downto 0);
      signal y_sig : signed (N downto 0);
      signal y_unsig : unsigned (N downto 0);

begin
      a_sig <= signed(a);
      b_sig <= signed(b);

      a_unsig <= unsigned(a);
      b_unsig <= unsigned(b);

      -- signed:
      y_sig <=     (a_sig(N-1) & a_sig) + (b_sig(N-1) & b_sig) when
opcode="100"else
                   (a_sig(N-1) & a_sig) - (b_sig(N-1) & b_sig) when
opcode="101"else
                   -(a_sig(N-1) & a_sig) + (b_sig(N-1) & b_sig) when
opcode="110"else
                   (a_sig(N-1) & a_sig) + (b_sig(N-1) & b_sig) + ('0' & cin) when
opcode="111";

      -- unsig:
      y_unsig <=   ('0' & a_unsig) + ('0' & b_unsig) when opcode="000"else
```

```
                                ('0' & a_unsig) - ('0' & b_unsig) when
opcode="001"else
                                ('0' & b_unsig) - ('0' & a_unsig)  when
opcode="010"else
                                ('0' & a_unsig) + ('0' & b_unsig) + ('0' & cin)
when opcode="011";


      y <=   std_logic_vector(y_sig(N-1 downto 0)) when opcode(2)='1' else
                  std_logic_vector(y_unsig(N-1 downto 0));
      cout <=     std_logic(y_sig(N)) when opcode(2)='1' else
                  std_logic(y_unsig(N));




end Behavioral;
```

ANEXO 8 – VHDL TEST BENCH – PROJETO 2

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Circuito_aritmetico_tb IS
END Circuito_aritmetico_tb;

ARCHITECTURE behavior OF Circuito_aritmetico_tb IS


    COMPONENT Circuito_aritmetico
    PORT(
         a : IN  std_logic_vector(3 downto 0);
         b : IN  std_logic_vector(3 downto 0);
         cin : IN  std_logic;
         opcode : IN  std_logic_vector(2 downto 0);
         y : OUT  std_logic_vector(3 downto 0);
         cout : OUT  std_logic
        );
    END COMPONENT;
```

```
    --Inputs
    signal a : std_logic_vector(3 downto 0) := (others => '0');
    signal b : std_logic_vector(3 downto 0) := (others => '0');
    signal cin : std_logic := '0';
    signal opcode : std_logic_vector(2 downto 0) := (others => '0');

        --Outputs
    signal y : std_logic_vector(3 downto 0);
    signal cout : std_logic;

    constant clk_period : time := 100 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
    uut: Circuito_aritmetico PORT MAP (
            a => a,
            b => b,
            cin => cin,
            opcode => opcode,
            y => y,
            cout => cout
          );
   a<= "1010";
   b<= "0001";



    -- Stimulus process
    stim_proc: process
    begin

         opcode <= "000";
         wait for clk_period;

         opcode <= "001";
```

```
        wait for clk_period;

        opcode <= "010";
        wait for clk_period;

        opcode <= "011";
        wait for clk_period;

        opcode <= "100";
        wait for clk_period;

        opcode <= "101";
        wait for clk_period;

        opcode <= "110";
        wait for clk_period;

        opcode <= "111";
        wait for clk_period;

   end process;

END;
```