UNIVERSIDADE FEDERAL DO PARANÁ
SETOR DE TECNOLOGIAS
CURSO DE ENGENHARIA ELÉTRICA

LUCAS FILUS RAMOS - GRR20202598
VITORIA MARIANA DA ROCHA GASINO - GRR20202564

TE351 DA – LABORATÓRIO 2

CURITIBA
2024

UNIVERSIDADE FEDERAL DO PARANÁ
SETOR DE TECNOLOGIA
CURSO DE ENGENHARIA ELÉTRICA

Lucas Filus Ramos
Vitoria Mariana da Rocha Gasino

TE 351 DA – LABORATÓRIO 2

Relatório técnico apresentado à disciplina Microeletrônica do Curso de Engenharia Elétrica do Setor de Tecnologia da Universidade Federal do Paraná, como parcial para completamento da disciplina.

Orientador: Sibilla Batista da Luz Franca.

CURITIBA

2024

Sumário

1. INTRODUÇÃO

O relatório a seguir trata de um projeto de quatro partes, sendo que na primeira delas se recebe um número binário na entrada e exibe sua representação decimal em um display de sete segmentos. Um decodificador é utilizado para converter o número binário em um formato adequado para exibição em um display de sete segmentos.

A segunda parte por sua vez determina o peso de Hamming de um vetor de comprimento genérico. O peso de Hamming é a quantidade de bits diferentes de zero em um vetor binário. Neste projeto, o circuito conta o número de bits "1" no vetor binário para calcular o peso de Hamming.

Na terceira parte do projeto se classifica um vetor binário, colocando os bits "1" na posição mais significativa. Em outras palavras, o circuito reorganiza o vetor de forma que os bits "1" fiquem no início do vetor, seguidos pelos bits "0".

Por fim, a parte trata de um projeto que realiza cálculos matemáticos utilizando funções condicionais (WHEN, SELECT e GENERATE). Ele pode executar diversas operações aritméticas, dependendo da configuração do circuito.

Todos esses projetos foram desenvolvidos utilizando apenas funções condicionais, o que os torna compatíveis com o kit NEXYS 2 e adequados para implementação em hardware.

2. PROJETO 1 – Display de 7 segmentos

2.1. Desenvolvimento do código

Após análise da tabela verdade para cada valor em decimal de 0 a 9 e sabendo que a cada posição do display irá ascender com o nível lógico baixo, temos o seguinte código.

FIGURA 1 – Código da parte 1.

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


entity code_pt1 is
    Port ( d : in  STD_LOGIC_VECTOR(3 downto 0);
           a : out  STD_LOGIC_VECTOR(3 downto 0);
           y : out  STD_LOGIC_VECTOR(6 downto 0));

end code_pt1;

architecture Behavioral of code_pt1 is

begin

   a <= "0111";

   y <= "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
    "0000100" when d="1001" else
    "1111111";


end Behavioral;
```

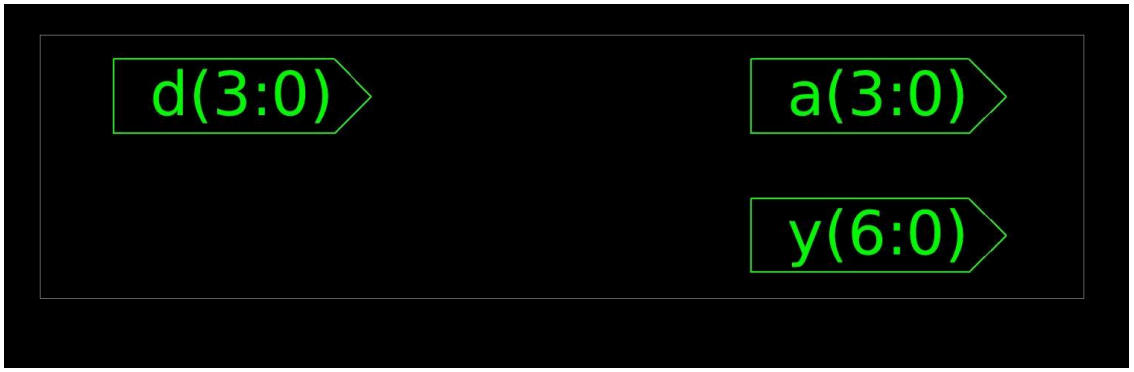
FONTE: Os autores (2024)

Neste código a varável ‘x’ é um vetor genérico de tamanho N será a entrada binária, a varável ‘a’ serve para definir o display que ficará funcional, e a varável ‘y’ define um vetor de tamanho 7 onde cada posição representa um led do display.

Na sequência, além de ‘a’ receber um valor que irá ascender somente o primeiro display é enviado para a saída ‘y’ o valor baseado na entrada ‘x’ após a análise da tabela verdade.

Além disso, também é possível verificar o esquemático do circuito e o desing summary após fazer a síntese, como ilustrado na figura 2.

FIGURA 2 – Circuito gerado em VHDL - Projeto 1



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	4	4656	0%
Number of 4 input LUTs	7	9312	0%
Number of bonded IOBs	15	232	6%

FONTE: Os autores (2024)

2.2. Implementação do Test Bench e resultados

Para avaliar o funcionamento do circuito implementado acima, o seguinte esquemático de Test Bench foi preparado.

FIGURA 5 – Testbench gerado para o Projeto 1

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

ENTITY teste IS
END teste;

ARCHITECTURE behavior OF teste IS

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

    COMPONENT code_pt1
    PORT(
         d : IN  std_logic_vector(3 downto 0);
         a : OUT  std_logic_vector(3 downto 0);
         y : OUT  std_logic_vector(6 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal d : std_logic_vector(3 downto 0) := (others => '0');

   --Outputs
   signal a : std_logic_vector(3 downto 0);
   signal y : std_logic_vector(6 downto 0);

BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: code_pt1 PORT MAP (
          d => d,
          a => a,
          y => y
        );

   -- Stimulus process
   stim_proc: process
   begin
      d<="0000"; -- 0
         wait for 10 ns;

         d<="0001"; --1
         wait for 10 ns;

         d<="0010"; --2
         wait for 10 ns;

         d<="0011"; --3
         wait for 10 ns;

         d<="0100"; --4
         wait for 10 ns;

         d<="0101"; --5
         wait for 10 ns;

         d<="0110"; --6
         wait for 10 ns;

         d<="0111"; --7
         wait for 10 ns;

         d<="1000"; --8
         wait for 10 ns;

         d<="1001"; --9
         wait for 10 ns;

         d<="1010"; --10
         wait for 10 ns;

         d<="1011"; --11
      wait;
   end process;

END;
```

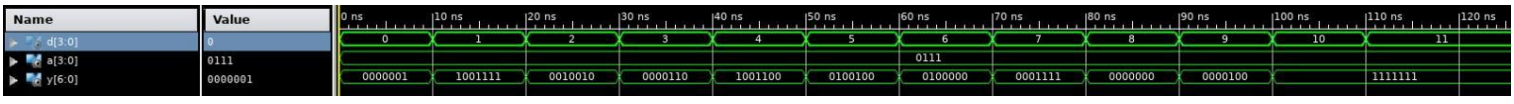
FONTE: Os autores (2024)

Primeiramente, são declarados os sinais que serão utilizados na simulação, juntamente com seus valores iniciais. Posteriormente, é feito o mapeamento entre esses sinais e as portas do componente.

Para conseguir avaliar melhor os resultados variando a entrada ‘x’, o mesmo foi definido como 1, 2, 3, 4, 5, 6, 7, 8, 9,10 e 11.

Após a conclusão da codificação do Test Bench foi possível realizar a simulação do tipo Behaviour onde é verificado se o design atende aos requisitos funcionais. Após o teste é validado o comportamento exatamente conforme o esperado em termos de entradas e saídas. Os resultados são exibidos na figura 3.

FIGURA 3 – Simulação Behaviour do Projeto 1.



FONTE: Os autores (2024)

Sendo possível avaliar que o processo de seleção de saída ocorreu como o esperado considerando a simulação de Test Bench codificada, observando o valor de ‘y’ correspondendo

a cada valor em decimal para o display de sete segmentos, além disso também é valido que para valores acima de 9 a saída se torna “1111111”, o que não ascenderá nenhum led.

3. PROJETO 2 – Peso de Hamming

3.1. Desenvolvimento do código

Para o Projeto 2, foi desenvolvido uma lógica utilizando um generate para efetuar o processamento do peso de Hamming, o código .vhd está exemplificado na figura 4.

FIGURA 4 – Código .vhd Projeto 2

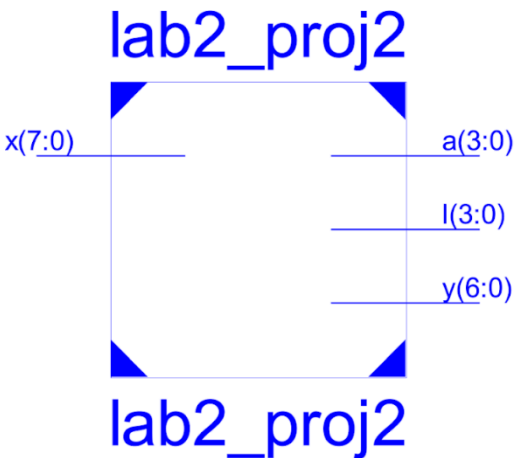
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_Std.ALL;

entity lab2_proj2 is
GENERIC(N: POSITIVE := 8);
PORT (x: in STD_LOGIC_VECTOR(N-1 downto 0);
a : out  STD_LOGIC_VECTOR(3 downto 0); -- a -> qual digito ascende
l : out STD_LOGIC_VECTOR(3 downto 0);
y : out  STD_LOGIC_VECTOR(6 downto 0)); -- y -> valor que sera aceso

end lab2_proj2;
architecture Behavioral of lab2_proj2 is

TYPE type1 is array (0 to N) of integer range 0 to 8;
signal cont : type1;
signal ultimo:  integer range 0 to 8;
signal d : std_logic_vector(3 downto 0);

begin
--CONTA HAMMING
cont(0)<=0;
gen1: for i in 1 to N generate
cont(i)<=cont(i-1)+1 when x(i-1)='1' else
cont(i-1);
end generate;
ultimo <= cont(N);

a <= "0111";
d <=std_logic_vector(to_unsigned(ultimo, 4));
l<=d;

y <= "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
 "0000100" when d="1001" else
 "1111111";

end Behavioral;
```

FONTE: Os autores (2024)

O código apresenta um generate que realiza um loop em ‘x’, valor de entrada, com um sinal contador (cont), um vetor de números inteiros de tamanho 0 a 8. Uma vez que o valor de entrada é ‘1’, o sinal contador envia a soma da sua posição antecessora para a atual (onde na posição 0 o valor é 0 e não é lido no loop, por isso se inicia em 1). Quando o sinal de entrada é lido e enviado o valor do número de ‘1’ para o contador, o sinal ‘ultimo’ receberá esse valor.

Por fim o sinal ‘ultimo’ é convertido para um std_logic_vector para assim ser possível utilizar a primeira parte do projeto, e enviar esse valor para o display de sete segmentos. A seguir é possível analisar o esquemático do circuito e o desing summary.

FIGURA 5 – Circuito gerado em VHDL - Projeto 2



Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	26	9,312	1%	
Number of occupied Slices	14	4,656	1%	
Number of Slices containing only related logic	14	14	100%	
Number of Slices containing unrelated logic	0	14	0%	
Total Number of 4 input LUTs	26	9,312	1%	
Number of bonded IOBs	23	232	9%	
Average Fanout of Non-Clock Nets	3.41			

FONTE: Os autores (2024)

3.2. Implementação do Test Bench e resultados

Para avaliar o funcionamento do circuito implementado acima, o seguinte esquemático de Test Bench foi preparado.

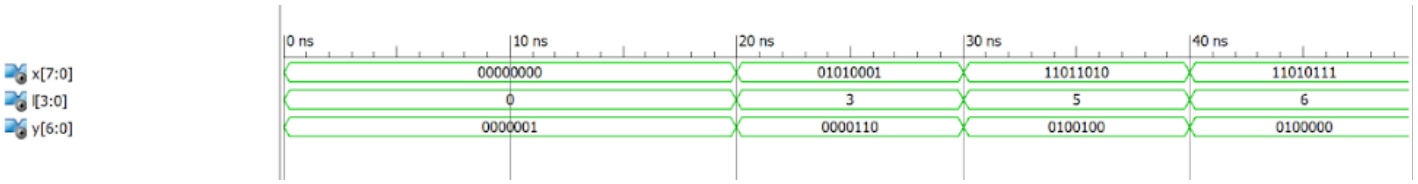
FIGURA 6 – Testbench gerado para o Projeto 2

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY teste2 IS
END teste2;

ARCHITECTURE behavior OF teste2 IS
    COMPONENT lab2_proj2
    PORT(
         x : IN  std_logic_vector(7 downto 0);
         a : OUT  std_logic_vector(3 downto 0);
         l : OUT  std_logic_vector(3 downto 0);
         y : OUT  std_logic_vector(6 downto 0)
        );
    END COMPONENT;
   signal x : std_logic_vector(7 downto 0) := (others => '0');
   signal a : std_logic_vector(3 downto 0);
   signal l : std_logic_vector(3 downto 0);
   signal y : std_logic_vector(6 downto 0);

BEGIN
   uut: lab2_proj2 PORT MAP (
          x => x,
          a => a,
          l => l,
          y => y
        );
   stim_proc: process
   begin
      x<="00000000";
      wait for 10 ns;
      x<="00000000";
      wait for 10 ns;
      x<="01010001";
      wait for 10 ns;
      x<="11011010";
      wait for 10 ns;
      x<="11010111";
      wait for 10 ns;
      x<="00001111";


      wait;
   end process;

END;
```

FONTE: Os autores (2024)

Com este código de test bench foi possível validar o código de hamming, como mostrado na figura abaixo.

FIGURA 7 – Simulação Behaviour do Projeto 2.



FONTE: Os autores (2024)

Na simulação, o valor do número de ‘1’ em ‘x’ é enviado de maneira correta para ‘l’, que por sua vez é transformado em um vetor binário ‘y’ responsável por acender de maneira correta os leds do display de sete segmentos, validando o circuito.

4. PROJETO 3 – Ordenador Binário

4.1. Desenvolvimento do código

Para o Projeto 3, foi desenvolvido uma lógica utilizando um dois generates para efetuar o processamento do peso de Hamming, como no projeto 2, e um segundo para ordenar os ‘1’ do vetor, o código .vhd está exemplificado na figura 8.

FIGURA 8 – Código .vhd Projeto 3

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

entity lab2_proj2 is
   GENERIC(N: POSITIVE := 8);
   PORT (x: in STD_LOGIC_VECTOR(N-1 downto 0);     -- x -> entrada
         a: out STD_LOGIC_VECTOR(3 downto 0);
         y : out  STD_LOGIC_VECTOR(6 downto 0);
         l : out STD_LOGIc_VECTOR(3 downto 0); -- valor de 1s
         led: out  STD_LOGIC_VECTOR(N-1 downto 0)); -- y -> valor que sera aceso
end lab2_proj2;

architecture Behavioral of lab2_proj2 is
   TYPE type1 is array (0 to N) of integer range 0 to 8;
   signal cont : type1;
   signal ordenador : std_logic_vector(0 to N-1);
   signal ultimo : integer range 0 to 8;
   signal d : std_logic_vector(3 downto 0);
begin
   a <= "0111";
   --CONTA HAMMING
   cont(0)<=0;
   gen1: for i in 1 to N generate
      cont(i)<=cont(i-1)+1 when x(i-1)='1' else
               cont(i-1);
   end generate gen1;
   ultimo<=cont(N);
   d<=std_logic_vector(to_unsigned(ultimo,4));
   l<=d;
   -- Decodificador
   y <= "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
        "0000100" when d="1001" else
        "1111111";
   --ORDENA
   gen2: for i in 0 to N-1 generate
      ordenador(i)<= '1' when i< cont(N) else
                     '0';
   end generate gen2;
   led <= ordenador;

end Behavioral;
```

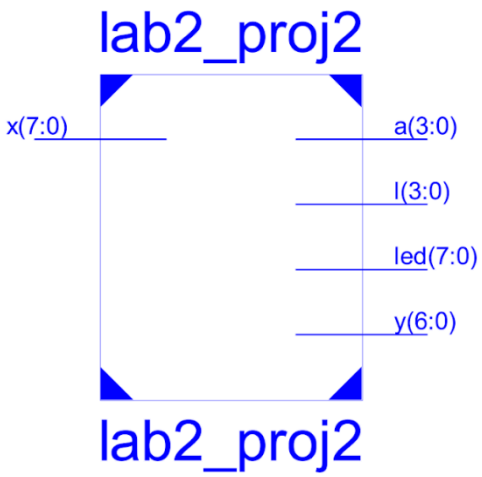
FONTE: Os autores (2024)

O código mostrado acima se assemelha muito ao do projeto 2, porém apresenta um segundo generate que realiza um loop que verifica o número de ‘1’s (gerado pelo primeiro generate) e se o valor de ‘i’ (varável de incrementação para o loop) for menor que este número de ‘1’s, o valor enviado para a variável ‘ordenador’ na devida posição ‘i’ será ‘1’, caso contrário o valor enviado será ‘0’. Desta maneira o vetor passa a estar ordenado, e é enviado para a variável ‘led’ que se tornará responsável por definir o valor alto ou baixo dos leds da placa física.

Neste projeto também foi realizada a passagem do número de ‘1’s para o display de sete segmentos como mostrado no projeto 2.

A seguir é possível analisar o esquemático do circuito e o desing summary.

FIGURA 9 – Circuito gerado em VHDL - Projeto 3



Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	37	9,312	1%	
Number of occupied Slices	19	4,656	1%	
Number of Slices containing only related logic	19	19	100%	
Number of Slices containing unrelated logic	0	19	0%	
Total Number of 4 input LUTs	37	9,312	1%	
Number of bonded IOBs	31	232	13%	
Average Fanout of Non-Clock Nets	3.52			

FONTE: Os autores (2024)

4.2. Implementação do Test Bench e resultados

Para avaliar o funcionamento do circuito implementado acima, o seguinte esquemático de Test Bench foi preparado.

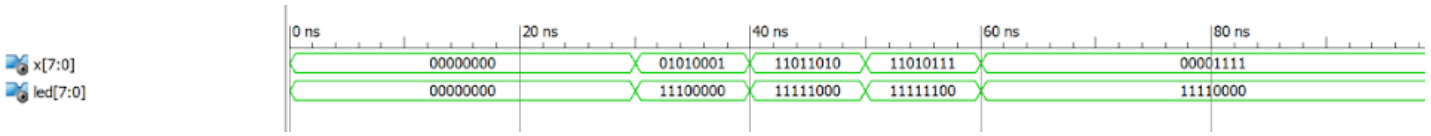
FIGURA 10 – Testbench gerado para o Projeto 3

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

ENTITY teste IS
END teste;

ARCHITECTURE behavior OF teste IS
    COMPONENT lab2_proj2
    PORT(
         x : IN  std_logic_vector(7 downto 0);
         a : OUT  std_logic_vector(3 downto 0);
         y : OUT  std_logic_vector(6 downto 0);
         led : OUT  std_logic_vector(7 downto 0)
        );
    END COMPONENT;
   signal x : std_logic_vector(7 downto 0) := (others => '0');
   signal a : std_logic_vector(3 downto 0);
   signal y : std_logic_vector(6 downto 0);
   signal led : std_logic_vector(7 downto 0);

BEGIN
   uut: lab2_proj2 PORT MAP (
          x => x,
          a => a,
          y => y,
          led => led
        );
   stim_proc: process
   begin
      wait for 10 ns;
      x<="00000000";
      wait for 10 ns;
      x<="00000000";
      wait for 10 ns;
      x<="01010001";
      wait for 10 ns;
      x<="11011010";
      wait for 10 ns;
      x<="11010111";
      wait for 10 ns;
      x<="00001111";
      wait;
   end process;

END;
```

FONTE: Os autores (2024)

Com este código de test bench foi possível validar a ordenação, como mostrado na figura abaixo.

FIGURA 11 – Simulação Behaviour do Projeto 3.



FONTE: Os autores (2024)

Na simulação, é possível observar que independente da entrada o valor de saída ‘led’ será a entrada ordenada, colocando os ‘1’s nas posições mais significativas.

5. PROJETO 4 – Circuito Aritmético com STD_LOGIC

5.1. Desenvolvimento do código

Para o Projeto 3, foi desenvolvido uma “mini ULA”, capaz de realizar operações de somas e subtrações, considerando ou não o valor de entrada como Signed ou Unsigned, e o valor de carry in. O código abaixo implementa essa condição.

FIGURA 12 – Código .vhd Projeto 3

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

entity lab3_proj4 is
    GENERIC (N: POSITIVE := 2);
    Port (    opcode : in  std_logic_vector(2 downto 0);  -- Unsi:  y = a + b 000 , y = a - b 001 , y = - a + b 010  , y = a + b + cin 011
               -- Sig : y = a + b 100 , y = a - b 101 , y = - a + b 110 , y = a + b + cin 111
     a, b: in std_logic_vector(N-1 downto 0);
     cin: in std_logic;
              cout: out std_logic;
              y: out std_logic_vector(N-1 downto 0));

end entity;

architecture Behavioral of lab3_proj4 is
    signal somador_n: std_logic_vector(N downto 0);

begin
    somador_n <=  std_logic_vector('0' & unsigned(a)+unsigned(b)) when opcode = "000" else
                  std_logic_vector('0' & unsigned(a)-unsigned(b)) when opcode = "001" else
                  std_logic_vector('0' & unsigned(b)-unsigned(a)) when opcode = "010" else
                  std_logic_vector('0' &  unsigned(a)+unsigned(b)+ ('0' & cin)) when opcode = "011" else

                  std_logic_vector(a(N-1) & signed(a)+signed(b)) when opcode = "100" else
                  std_logic_vector(a(N-1) & signed(a)-signed(b)) when opcode = "101" else
                  std_logic_vector(a(N-1) & signed(b)-signed(a)) when opcode = "110" else
                  std_logic_vector(a(N-1) & signed(a)+signed(b) +  ('0' & cin)) when opcode = "111" else
                  "000";

    y<= somador_n(N-1 downto 0);
    cout<=somador_n(N);

end Behavioral;
```

FONTE: Os autores (2024)

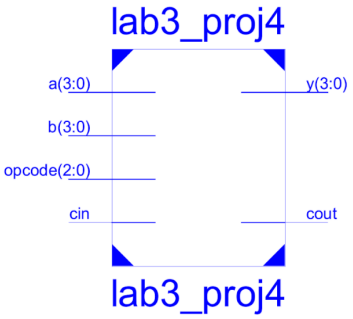
O código mostrado acima apresenta 8 operações aritméticas das entradas ‘a’ e ‘b’ (ambas um vetor de dois bits) e o resultado de saída ‘y’ irá receber uma das operações conforme a escolha por meio de um valor binário da entrada ‘opcode’. As operações sâo definidas pelo valor do vetor de entrada ‘opcode’,seguindo a tabela 1.

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

Tabela 1

A seguir é possível analisar o esquemático do circuito e o desing summary.

FIGURA 13 – Circuito gerado em VHDL - Projeto 4



FONTE: Os autores (2024)

FIGURA 13 – Sumário do números de LUTs utilizadas- Projeto 4

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	17	9,312	1%	
Number of occupied Slices	9	4,656	1%	
Number of Slices containing only related logic	9	9	100%	
Number of Slices containing unrelated logic	0	9	0%	
Total Number of 4 input LUTs	17	9,312	1%	
Number of bonded IOBs	11	232	4%	
Average Fanout of Non-Clock Nets	2.60			

FONTE: Os autores (2024)

5.2. Implementação do Test Bench e resultados

Para avaliar o funcionamento do circuito implementado acima, o seguinte esquemático de Test Bench foi preparado.

FIGURA 14 – Testbench gerado para o Projeto 3

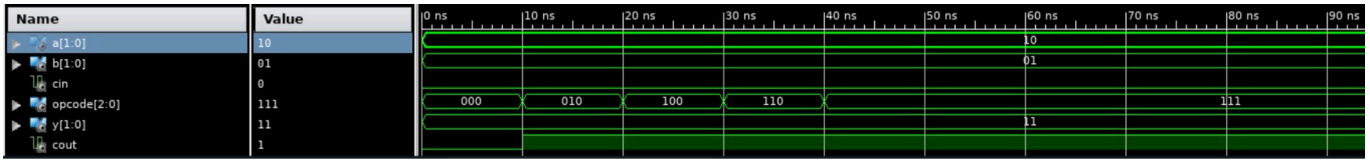
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY testeinicial IS
END testeinicial;

ARCHITECTURE behavior OF testeinicial IS


    COMPONENT lab3_proj4
    PORT(
         a : IN  std_logic_vector(1 downto 0);
         b : IN  std_logic_vector(1 downto 0);
         cin : IN  std_logic;
         opcode : IN  std_logic_vector(2 downto 0);
         y : OUT  std_logic_vector(1 downto 0);
         cout : OUT  std_logic
        );
    END COMPONENT;

   --Inputs
   signal a : std_logic_vector(1 downto 0) := (others => '0');
   signal b : std_logic_vector(1 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(1 downto 0);
   signal cout : std_logic;

BEGIN
   uut: lab3_proj4 PORT MAP (
          a => a,
          b => b,
          cin => cin,
          opcode => opcode,
          y => y,
          cout => cout
        );
```

```
BEGIN
   uut: lab3_proj4 PORT MAP (
          a => a,
          b => b,
          cin => cin,
          opcode => opcode,
          y => y,
          cout => cout
        );

   stim_proc: process
   begin
      a<= "10";
      b<= "01";
      opcode <= "000";
      wait for 10 ns;
      opcode <= "010";
      wait for 10 ns;
      opcode <= "100";
      wait for 10 ns;
      opcode <= "110";
      wait for 10 ns;
      opcode <= "111";
      wait for 100 ns;
      wait;
   end process;

END;
```

FONTE: Os autores (2024)

Com este teste bench foi possível validar as operações, como mostrado na figura abaixo.

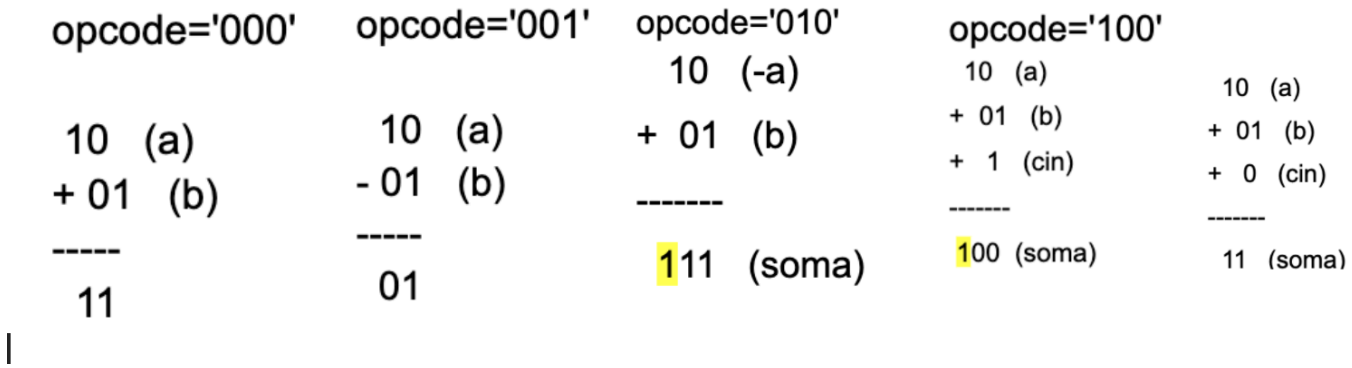
FIGURA 15 – Simulação Behaviour do Projeto 3.



FONTE: Os autores (2024)

Para fins de validação dos resultados, foram elaborada as somas e subtrações, na imagem abaixo.

FIGURA 16 – Resultados esperados das operações.



FONTE: Os autores (2024)

Os bits assinalados em amarelo representam a subida('1') do carry-out(cout). Comparando os resultados da imagem 16 com os obtidos com o testebench(imagem 15), podemos observar que os resultados esperados foram satisfeitos.

6. CONCLUSÃO

Este relatório apresentou quatro projetos desenvolvidos e efetivos em suas respectivas funcionalidades. Após implementação e teste nos kits NEXYS 2, os resultados foram consistentes e satisfatórios, alinhando-se com os objetivos planejados para cada projeto.

Portanto, foi comprovado que os quatro projetos funcionaram conforme o esperado e atenderam aos requisitos de desempenho quando implementados na placa kit NEXYS 2. Esses resultados validam a eficácia do uso de funções condicionais na elaboração de circuitos digitais para aplicações diversas.