Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Coordenação de Engenharia Elétrica (COELT)
Campus Apucarana

Relatório do laboratório 05

Maria Eduarda Pedroso (Líder) **Matrícula: 2150336**

Sistemas Digitais (SICO5A)

5º Experimento: VHDL - Introdução DATA 27/09/2022

Resumo

Nessa atividade todas as simulações e análises foram feitas no simulador EDAplaydround e também com auxílio dos conteúdos disponibilizados pelo professor e sua ajuda em aula. O intuito deste relatório é fixar e analisar códigos desenvolvidos na linguagem VHDL, sendo o primeiro um circuito lógico, foram analisados 3 códigos.

Após essa análise conseguimos sair com um maior entendimento de como funciona a linguagem e sua sintaxe e samantica além de uma parte do que se pode fazer.

Por fim, simulamos a expansão de células com duas entradas e um módulo adaptável ALM para as condições descritas nas figuras contidas no PDF disponibilizado pelo professor. Como foi realizada apenas análise e prototipagem no software Logisim não houve cálculos nem valores que seriam da parte prática do experimento.

Objetivos

- Aprender a utilizar o simulador EDAplaydround (https://www.edaplayground.com)
- Simular e analisar e a linguagem VDHL para circuitos lógicos e memória RAM.

Materiais e equipamentos

Os materiais utilizados foram todos códigos e funções no simulador EDA, sendo estes explicados sua utilidade no relatório.

Procedimentos e análise

VHDL é uma sigla que contém uma sigla: VHSIC Hardware Description Language, sendo VHSIC abreviatura para Very High Speed Integrated Circuits. Como o nome diz, VHDL é uma linguagem de descrição de hardware. Guarde bem essa sigla, HDL, que abrevia Hardware Description Language.

Diferentemente de C, um programa em VHDL descreve uma estrutura, ao invés de definir ou prescrever uma computação. Neste sentido, um programa em C é uma “receita de bolo” com ingredientes (variáveis) e modo de preparar (algoritmo). Em VHDL um modelo estrutural pode descrever a interligação de componentes, ou pode especificar sua função sem mencionar a implementação daquela função.

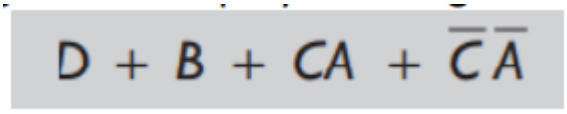
A partir da estrutura, um compilador como ghdl gera um simulador para o circuito descrito pelo código fonte. VHDL também permite descrever a função ou o comportamento de um circuito, sem que seja necessário definir sua estrutura.

Um sintetizador é um compilador que gera uma descrição de baixo nível – portas lógicas ou transistores – para que um circuito integrado seja fabricado a partir da descrição.

Após essa breve descrição podemos começar a explicar detalhadamente os códigos cedidos pelo professor e produzidos ou alterados pela aluna.

O primeiro código é a representação de um circuito de 7 segmentos ilustrado abaixo:

Figura 1 – Display de 7 segmentos.




```
entity SEGLOGIC is
port (A, B, C,D: in bit; SEGa: out bit);
end entity SEGLOGIC;
architecture LogicFunction of
SEGLOGIC is
begin
SEGa <= (A and C) or (notA
and not C) or B or D;

end architecture LogicFunction;
```

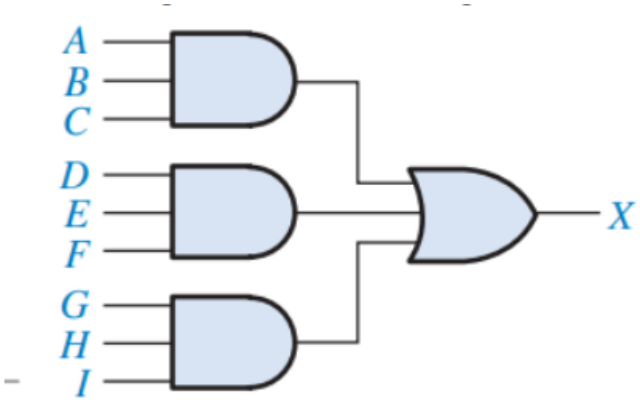
(a) (b)

Figura 1 – Display de 7 segmentos (a) Construção; (b) – Código.

Como podemos observar temos uma entidade chamada SEGLOGIC no qual tem as entradas A, B, C, D em bit (0 e 1) e uma saída SEGa também em bit, na arquitetura LogicFunction temos como realmente será o funcionamento do display sendo que a saída SEGa vai receber o resultado total das portas A e C sendo essa a representação de uma porta and, not A e Not C uma porta and com às duas entradas negadas e esses resultados todos em uma porta or juntamente com a soma das entradas B e D. Esse código não é muito complexo, mas nos dá uma introdução bem concreta de como é uma estrutura de um código VHDL.

Após a simulação desse primeiro código, entendemos e adquirimos conhecimento para implementar a figura 2(a) cedida pelo professor.

Figura 2 – Circuito lógico e seu respectivo código.




```
entity testbench is
-- empty
end testbench;

architecture tb of testbench is

-- DUT component
component or_gate is
        port (a,b,c,d,e,f,g,h,i:in bit; q: out bit);
end component;

signal a_in, b_in, c_in, d_in, e_in, f_in, g_in, h_in, i_in,
q_out: bit;

begin
```

```
  -- Connect DUT
  DUT: or_gate port map(a_in, b_in, c_in, d_in, e_in, f_in,
g_in, h_in, i_in, q_out);

  process
  begin
    a_in <= '0';
    b_in <= '0';
    c_in <= '0';
    d_in <= '0';
    e_in <= '0';
    f_in <= '0';
    g_in <= '0';
    h_in <= '0';
    i_in <= '0';
    wait for 1 ns;

    assert(q_out='0') report "1";
    assert(q_out='1') report "0";

    assert false report "Teste terminou";

    -- Clear inputs
    a_in <= '0';
    b_in <= '0';
    c_in <= '0';
    d_in <= '0';
    e_in <= '0';
    f_in <= '0';
    g_in <= '0';
    h_in <= '0';
    i_in <= '0';

    wait;
  end process;
end tb;

entity or_gate is
        port (a,b,c,d,e,f,g,h,i:in bit; q: out bit);
end entity or_gate;

architecture rtl of or_gate is
begin
        process(a, b, c, d, e, f, g, h, i) is
    begin
      q <=(a and b and c) or (d and e and f) or (g and h and i);
    end process;
end architecture rtl;
```

(a) (b)

Figura 2 – Circuito lógico e seu respectivo código. (a) Circuito lógico; (b) – Código.

Como podemos ver temos uma entidade chamada testbench que possui entradas A, B, C, D, E, F, G, H, I para bits (0 e 1) e saídas Q para bits (0 e 1), na arquitetura LogicFunction conhecemos o display. O funcionamento real e a saída Q receber o resultado total das portas A, B, C, D, E, F, G, H, I que é uma representação de uma porta AND.

Nesse exemplo conseguimos verificar a funcionalidade em VHDL pela verificação: assert(q_out='0') report "1";
assert(q_out='1') report "0";

A qual retorna 1 caso a soma de todos os valores de entrada são 0, e 0 caso algum dos valores seja 1.

Após a simulação desse segundo código, seguimos para a implementação da figura 3(a) cedida pelo professor.

Figura 3 – Circuito lógico e seu respectivo código.

$$Y = A\overline{B}C + \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C} + \overline{A}BC$$

```
entity testbench is
-- empty
end testbench;

architecture tb of testbench is

-- DUT component
component or_gate is
        port (a,b,c:in bit; q: out bit);
end component;

signal a_in, b_in, c_in, q_out: bit;

begin

  -- Connect DUT
  DUT: or_gate port map(a_in, b_in, c_in, q_out);

  process
  begin
    a_in <= '0';
    b_in <= '0';
    c_in <= '0';
    wait for 1 ns;

    assert(q_out='0') report "1";
    assert(q_out='1') report "0";

    assert false report "Teste terminou";

    -- Clear inputs
    a_in <= '0';
    b_in <= '0';
    c_in <= '0';
```

```
   wait;
  end process;
end tb;

entity or_gate is
        port (a,b,c:in bit; q: out bit);
end entity or_gate;

architecture rtl of or_gate is
begin
        process(a, b, c) is
   begin
     q <=(A and not B and C) or (not A and not B and C) or
(A and not B and not C) or (not A and B and C);
   end process;
end architecture rtl;
```

(a) (b)

Figura 3 – Circuito lógico e seu respectivo código. (a) Circuito lógico; (b) – Código.

Como podemos ver temos uma entidade chamada testbench que possui entradas A, B, C para bits (0 e 1) e saídas Q para bits (0 e 1), na arquitetura LogicFunction conhecemos o display.

Nesse exemplo conseguimos verificar a funcionalidade em VHDL pela verificação:

```
assert(q_out='0') report "1";
assert(q_out='1') report "0";
```

Já nesse exemplo temos a figura 3 (a) como circuito lógico a qual é constituída da soma de vários AND e OR, para esse caso temos como retorno “0” para o resultado da figura 1, aplicando como entrada valores 0 para A, B e C.

Por fim, precisamos analisar um código descrito na figura 4 que emula uma memória ram.

Figura 4 – Código que emula uma memória RAM.

```
 library ieee;
 use ieee.std_logic_1164.all;
 ENTITY shift_register IS
 port
 (
     sys_clk : in std_logic;
     sys_rst : in std_logic;
     dado_entrada : in std_logic;
-- Random Access Memory (RAM) with
-- 1 read/write port

LIBRARY IEEE;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_UNSIGNED.ALL;

-- RAM entity
ENTITY RAM IS
  PORT(
      DATAIN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      ADDRESS : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      -- Write when 0, Read when 1
      W_R : IN STD_LOGIC;
      DATAOUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
      );
END ENTITY;

-- RAM architecture
ARCHITECTURE BEV OF RAM IS

TYPE MEM IS ARRAY (255 DOWNTO 0) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL MEMORY : MEM;
SIGNAL ADDR : INTEGER RANGE 0 TO 255;

BEGIN


  PROCESS(ADDRESS, DATAIN, W_R)
  BEGIN

    ADDR<=CONV_INTEGER(ADDRESS);
    IF(W_R='0')THEN
      MEMORY(ADDR)<=DATAIN;
    ELSIF(W_R='1')THEN
      DATAOUT<=MEMORY(ADDR);
    ELSE
      DATAOUT<="ZZZZZZZZ";
    END IF;
  END PROCESS;

END BEV;
```

Figura 4 – Código que emula uma memória RAM.

Como podemos observar, a estrutura desse código é um pouco mais complexa, temos as livrarias no início do código, essas livrarias são pacotes parecidos com os arquivos .h da linguagem c, após isso temos a entidade SHIFT REGISTER que é nada mais é do que um array de flip-flops conectados em um mesmo clock de forma que a cada pulso de clock a entrada de um flip-flop recebe o valor da saída de seu anterior.

Após isso a próxima entidade é intitulada como RAM que é basicamente o funcionamento descrito da memória RAM com duas entradas DATAIN e ADRESS sendo um array lógico e seu

algarismo mais significativo na posição 7, e uma saída DATAOUT também sendo um array com o algarismo mais significativo na posição 7.

Começamos a arquitetura instanciando um vetor de array de 256 posições, temos um processo que recebe como parâmetro ADRESS, DATAIN W_R e esse processo basicamente pega o endereço em inteiro passando para ADDR e e faz uma comparação para ver se estamos fazendo uma leitura ou escrita, se W_R for 1 ele faz a leitura, já se W_R for 0 temos a escrita.

Uma vez descrito um dispositivo em VHDL, passa-se para a etapa de verificação, cujo objetivo é verificar se ele está funcionando corretamente. Funcionando corretamente significa que ele fornece o resultado esperado para todas as entradas válidas, isto é, que se esperam que aconteçam em funcionamento normal. Para isto, analogamente aos dispositivos físicos, deve-se montar o dispositivo como um componente em uma bancada de testes (testbench), injetar se os sinais adequados para fazer o dispositivo funcionar e verificar se as saídas estão corretas. Isto pode ser feito em VHDL usando um testbench virtual.

O primeiro passo é definir o dispositivo de teste. Ele é descrito por uma entidade sem portas de entrada e saída. Por exemplo, para se testar um dispositivo filtro (mostrado no anexo) pode-se criar a entidade tb_filtro como abaixo.

```
library IEE;
use IFEE.std_logic_1164.ALL;
use IEFE.numeric_std.ALL;
use std.textio.ALL;

entity TB FILTRO is
lend TB_FILTRO;
```

Na descrição da arquitetura, deve-se ter uma instância do dispositivo em teste e os vários sinais para fazê-lo funcionar. Para se criar uma instância, pode-se usar o esquema tradicional de se definir um componente como mostrado abaixo. Esta maneira é a única possível no VHDL87. Não é necessário que a implementação seja sintetizável, pois não será gerado um circuito a partir dela.

```
architecture TESTBENCH1 of TB_FILTRO is

component FILTRO
port (CLK : in std_logic;
RST : in std_logic;
INPUT : out std_logic;
OUTPUT : in std_logic;
 );
end component FILTRO;

constant PERIODO : time := 10 ns;
```

```
signal W_CLK : std_logic := '0'; -- deve ser inicializado
signal W_RST : std_logic;
signal W_INPUT : std_logic;
signal W_OUTPUT : std_logic;
begin
-- geracao do relogio com periodo PERIODO
W_CLK <= not W_CLK after PERIODO/2;
-- criacao de um FILTRO
 DUT : FILTRO port map( CLK => W_CLK,
RST => W_RST,
INPUT => W_INPUT,
OUTPUT => W_OUTPUT);
 --- falta a injecao de sinais
end TESTBENCH1;
```

Maria Eduarda Pedroso

Resultados e Conclusão

A partir dessa prática obtivemos o entendimento de como criar códigos não tão complexos em VHDL e como essa linguagem funciona.

Ao contrário de linguagens de programação como C ou assembly, nas quais os comandos/instruções são executados na ordem em que aparecem no código fonte, em VHDL o cômputo das expressões e as atribuições de valores ocorrem em paralelo, com um efeito que emula a propagação de sinais num circuito. Neste sentido, o código VHDL é declarativo, ao invés de imperativo, como em C. A diferença entre estes estilos, imperativo e declarativo, ficará mais clara no que se segue.

Muitos tópicos dessa prática se relacionam e complementam ao estudo teórico passado pelo professor em aula, como não houve cálculos e sim análise acredito que essa atividade foi algo de suma importância para um melhor entendimento do assunto, visto que é muito raro o aprendizado de tal linguagem.