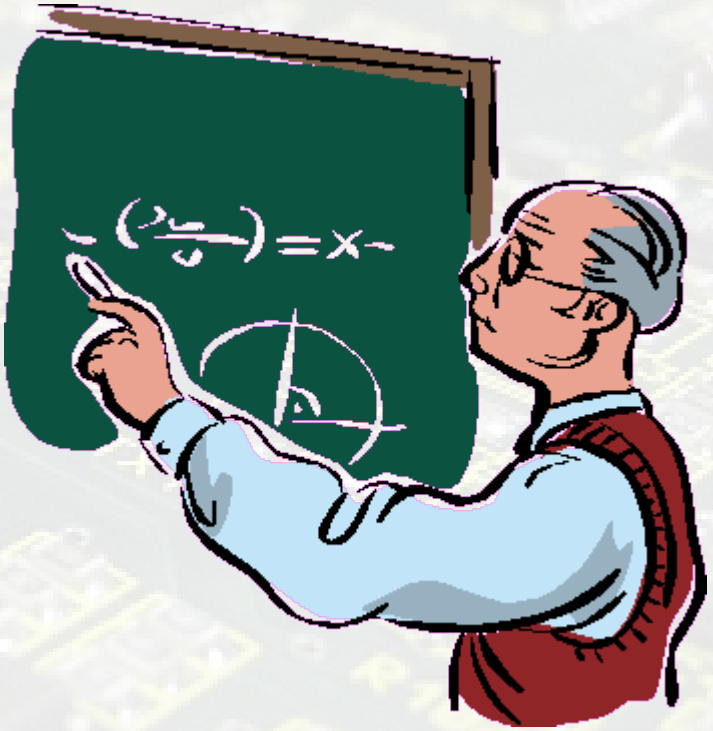
# Sistemas Digitais

## Aulas práticas de laboratório

Parametrização e Hierarquia no VHDL

Prof. Dr. Eng. Rafael Luiz Cancian

# Conteúdo



# Projeto Parametrizado

- Reuso de projeto é um dos maiores objetivos ao desenvolver código VHDL.
- Idealmente queremos projetar alguns módulos comuns que possam ser compartilhados por muitas aplicações.
- VHDL suporta projeto parametrizado de muitas formas, incluindo
  - passagem de parâmetros para a entidade;
  - extração de atributos de objetos;
  - sobrecarga de operadores;
  - *unconstrained arrays*;
  - Construções de linguagem, como *generic*, *for* (*generate* e *loop*)

# Parametrização - Generics

- Um projeto parametrizado precisa de um mecanismo para especificar parâmetros.
  - *generics, array attribute, unconstrained array*.
- Generics são “constantes simbólicas” passadas para a declaração da entidade.
- Para usar um generic, precisamos substituir a declaração constante no código orieginal pela declaração genérics na declaração da entidade.

# Parametrização - Generics

```
library ieee;
use ieee.std_logic_1164.all;
entity reduced_xor is
    generic(WIDTH: natural);  -- generic declaration
    port(
        a: in std_logic_vector(WIDTH-1 downto 0);
        y: out std_logic
    );
end reduced_xor;

architecture loop_linear_arch of reduced_xor is
    signal tmp: std_logic_vector(WIDTH-1 downto 0);
begin
    process(a,tmp)
    begin
        tmp(0) <= a(0);  -- boundary bit
        for i in 1 to (WIDTH-1) loop
            tmp(i) <= a(i) xor tmp(i-1);
        end loop;
    end process;
    y <= tmp(WIDTH-1);
end loop_linear_arch;
```

# Parametrização – Array Attributes

- Um atributo provê informação sobre um objeto, como um tipo de dados ou um sinal.
- Atributos são acessados utilizando ‘ após o nome do objeto, seguido pelo atributo desejado. Alguns atributos utilizados na parametrição são:
  - *s‘left* , *s‘right* : os limites da esquerda e da direita de *s*;
  - *s‘low* , *s‘high* : os limites inferior e superior de *s*;
  - *s‘length* : o comprimento do índice de *s*;
  - *s‘range* : a faixa do índice de *s*;

# Parametrização – Array Attributes

```
signal s1: std_logic_vector(31 downto 0);
signal s2: std_logic_vector(8 to 15);
```

- s1'left = 31; s1'right = 0;
- s1'low = 0; s1'high = 31;
- s1'length = 32;
- s1'range = 31 downto 0
- s1'reverse_range = 0 to 31

- s2'left = 8; s2'right = 15;
- s2'low = 8; s2'high = 15;
- s2'length = 8;
- s2'range = 8 to 15
- s2'reverse_range = 15 downto 8

```
architecture attr_arch of reduced_xor is
   signal tmp: std_logic_vector(a'length-1 downto 0);
begin
   process(a,tmp)
   begin
      tmp(0) <= a(0);
      for i in 1 to (a'length-1) loop
         tmp(i) <= a(i) xor tmp(i-1);
      end loop;
   end process;
   y <= tmp(a'length-1);
end attr_arch;
```

# Parametrização – Array

- Sinais normalmente usam tipo de dados *array*, como *std_logic_vector, unsigned* ou *signed.*
- O projeto parametrizado utiliza descrições sem referências fixas de tamanho. Exemplos:

*s <= “00000000”;* *prefira:* *s <= (others=>’0’);*

*s <= “00000101”;* *prefira:* *s <= (0,2=>’1’, others=>’0’);*

*s <= ‘1’ when (a=(a’range=>’0’)) else …* para verificar se a tem todos os bits ‘0’

```
constant zero: std_logic_vector(width-1 downto 0) := (others=>’0’);
…
s <= ‘1’ when a=zero eles ...
```

```
signal src: std_logic_vector(7 downto 0);
…
dest <= src(6 downto 0);   prefira:   dest <= src(src’left-1 downto src’right+1);
```

# Parametrização – For Generate

- O comando concorrente *for generate* é muito útil para gerar uma quantidade parametrizável de hardware.
- Exemplo:

# Parametrização – For Generate

```
entity adder is
    generic( N: positive );
    port(    a, b: in std_logic_vector(N-1 downto 0);
             result: out std_logic_vector(N-1 downto 0);
             ovf, cout: out std_logic );
end entity;
architecture arch1 of adder is
    component fulladder1bit is
        port(    cin, a, b: in std_logic;
                 sum: out std_logic;
                 cout: out std_logic );
    end component;
    signal carry: std_logic_vector(N downto 0);
begin
    gera: for i in result'range generate
        fa: fulladder1bit port map (carry(i), a(i), b(i), result(i), carry(i+1));
    end generate;
    carry(0) <= '0';
    cout <= carry(N);
    ovf <= carry(N) xor carry(N-1);
end architecture;
```

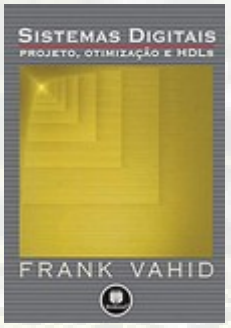
# Parametrização – If Generate

# Parametrização – If Generate

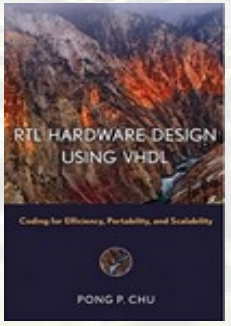
```
entity addersubtractor is
        generic(    N: positive;
                    isAdder: boolean;
                    isSubtractor: boolean );
        port( op: in std_logic;
                a, b: in std_logic_vector(N-1 downto 0);
...
        signal operandB: std_logic_vector(N-1 downto 0);
begin
        gera: for i in result'range generate
                fa: fulladder1bit port map (carry(i), a(i), operandB(i), result(i), carry(i+1));
        end generate;
        generateAdder: if isAdder and not isSubtractor generate
                carry(0) <= '0';
                operandB <= b;
        end generate;
        generateSubtractor: if not isAdder and isSubtractor generate
                carry(0) <= '1';
                operandB <= not b;
        end generate;
        generateBoth: if isAdder and isSubtractor generate
                carry(0) <= op;
                operandB <= b when op='0' else not b;
        end generate;
        ...
```

# Referências Bibliográficas


- Vahid, Frank. Sistemas Digitais: projeto, otimização e HDLs. Porto Alegre: Bookman, 2008. ISBN 978-85-7780-190-9




- Chu, Pomg P. RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability. Wiley-Interscience, 2006.




- Pedroni, Volnei. Circuit Design with VHDL. The MIT Press, 3th edition, 2020.