Estrutura básica de código

Comentários: são precedidos de duas hífens consecutivas (--) e terminam no final da linha.

Exemplo:

```
-- Isso é um comentário
```

Declaração de bibliotecas: Lista de bibliotecas necessárias para o projeto.

Exemplo:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

Entidade: Lista com as especificações de todas as portas de entrada e saída (I/O). A sintaxe simplificada é mostrada abaixo.

```
ENTITY entity_name IS
      PORT (port_name: signal_mode signal_type;
             port_name: signal_mode signal_type;
             …);
END entity_name;
```

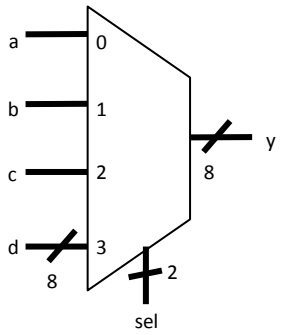
Signal mode: IN, OUT, INOUT, BUFFER.
Signal type: BIT, BIT_VECTOR, INTEGER, STD_LOGIC, STD_LOGIC_VECTOR, BOOLEAN, etc**.**

Arquitetura: Contém o código VHDL.

```
ARCHITECTURE architecture_name OF entity_name IS
      [declarative part]
BEGIN
      (code)
END architecture_name;
```

Declarative part: TYPE, COMPONENT, SIGNAL, etc.
Code: Concorrente ou sequencial.

Exemplo: 4x8 mux




```
------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------------------------------
ENTITY mux_4x8 IS
      PORT (a, b, c, d:  IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      port_name: signal_mode signal_type;
      …);
END entity_name;
------------------------------------------------------------
ARCHITECTURE myarch OF mux_4x8 IS
      -- SIGNAL x: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
      y <= a WHEN sel=0 ELSE
           b WHEN sel=1 ELSE
           c WHEN sel=2 ELSE
           d;

END myarch;
------------------------------------------------------------
```

Bibliotecas / Pacotes

std / standard: Tipos BOOLEAN, BIT, BIT_VECTOR, INTEGER, NATURAL, POSITIVE, REAL, TIME, etc.
iee / std_logic_1164: Tipos STD_ULOGIC e STD_LOGIC.

Tipos pré-definidos

Pre-defined data types	Library/Package of origin	Synthesizable values
BIT, BIT_VECTOR	std / standard	'0', '1'
BOOLEAN	std / standard	TRUE, FALSE
INTEGER	std / standard	$-(2^{31}-1)$ à $+(2^{31}-1)$
NATURAL	std / standard	0 à $(2^{31}-1)$
POSITIVE	std / standard	1 à $(2^{31}-1)$
CHARACTER	std / standard	8bit ASCII
STRING	std / standard	String of characters
STD_ULOGIC, STD_ULOGIC_VECTOR STD_LOGIC, STD_LOGIC_VECTOR	ieee / std_logic_1164	Input: '0', '1', 'L', 'H' Output: '0', '1', 'L', 'H', 'Z', '-', 'X'

Operadores

Operadores de atribuição: "<=" para sinais, ":=" para variáveis ou valores iniciais.

Operadores de concatenação: "&", ",","OTHERS =>"
Exemplo:

```
k : CONSTANT BIT_VECTOR(1 TO 4) := "1100";
x: <= ('Z', k(2 TO 3), "11111");    -- result: "Z1011111"
y <= 'Z' & k(2 TO 3) & "11111";     -- result: "Z1011111"
```

Operadores lógicos: NOT, AND, NAND, OR, NOR, XOR, XNOR.
Suportam: BOOLEAN, BIT, BIT_VECTOR, STD_(U)LOGIC, STD_(U)LOGIC_VECTOR

Operadores aritméticos: +, -, *, /, **, ABS, REM, MOD
Suportam: INTEGER, NATURAL, POSITIVE

Operadores de SHIFT: SLL, SRL, SLA, SRA, ROL, ROR
Suportam: BIT_VECTOR

Operadores de comparação: =, /=, >, <, >=, <=
Suportam: BOOLEAN, BIT, BIT_VECTOR,INTEGER, NATURAL, POSITIVE,CHARACTER, STRING

Componente

É um pedaço *convencional* de código dividido em 3 partes:
Código do componente: Em um arquivo separado (exige a clausula USE) ou no mesmo arquivo do código principal.
Declaração do componente: Em um package, architecture, generate ou block.
Instanciação do componente: Na arquitetura do código principal.

```
COMPONENT component_name IS
      PORT (port_name: signal_mode signal_type;
            port_name: signal_mode signal_type;
            …);
END COMPONENT;
```

```
label: [COMPONENT] component_name PORT MAP (port list);
```

```
------- Component declaration: ------------------------------------
COMPONENT nand_gate IS
      PORT (a, b: IN BIT; c: OUT BIT);
END COMPONENT;

------- Component instantiation: ----------------------------------
nand1: nand_gate PORT MAP (x, y, z);           --positional mapping
nand2: nand_gate PORT MAP (a=>x, b=>y, c=>z);  --nominal mapping
```

Receita de bolo de teste

Para compilar e testar o código desenvolvido, em qualquer máquina do laboratório executar a sequência abaixo.

vi mux_2_1.vhdl
ghdl -a mux_2_1.vhdl
ghdl -e teste
ghdl -r teste --vcd=teste.vcd
gtkwave teste.vcd

O mux_2_1.vhdl usado como exemplo está abaixo.

mux_2_1.vhdl

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

entity inversor is
  port( input: in std_logic;
        output: out std_logic );
end inversor;

architecture estrutura_inversor of inversor is
begin
  output <= not input after 5 ns;
end estrutura_inversor;

-----------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity e2 is
  port( i1, i2: in std_logic;
        output: out std_logic );
end e2;

architecture estrutura_e2 of e2 is
begin
  output <= i1 and i2 after 8 ns;
end estrutura_e2;

-----------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity ou2 is
  port( i1, i2: in std_logic;
        output: out std_logic );
end ou2;

architecture estrutura_ou2 of ou2 is
begin
  output <= i1 or i2 after 7 ns;
end estrutura_ou2;
```

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

entity mux21 is
  port( a, b, s: in std_logic;
        y: out std_logic );
end mux21;

architecture estrutura_mux21 of mux21 is
  component c1
    port( input: in std_logic;
          output: out std_logic );
  end component;
  component c2
    port( i1, i2: in std_logic;
          output: out std_logic );
  end component;
  for comp1: c1 use entity work.inversor(estrutura_inversor);
  for comp2: c2 use entity work.e2(estrutura_e2);
  for comp3: c2 use entity work.e2(estrutura_e2);
  for comp4: c2 use entity work.ou2(estrutura_ou2);
  signal n1, n2, n3: std_logic;
begin
  comp1: c1 port map (s, n1);
  comp2: c2 port map (a, n1, n2);
  comp3: c2 port map (b, s, n3);
  comp4: c2 port map (n2, n3, y);
end estrutura_mux21;

-----------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity teste is
end teste;

architecture arq_teste of teste is
  component c1
    port( a, b, s: in std_logic;
          y: out std_logic );
  end component;
  for comp1: c1 use entity work.mux21(estrutura_mux21);
  signal ent0, ent1, sel, saida: std_logic;
begin
  comp1: c1 port map (ent0, ent1, sel, saida);
  ent0 <= '1';
  ent1 <= '0';
  sel  <= '0', '1' after 30 ns, '0' after 60 ns, '1' after 90 ns;
end arq_teste;
```