

Projetos de Circuitos Digitais em VHDL e FPGA

Cap. 2 - Introdução ao VHDL

Prof. Erivelton Geraldo Nepomuceno

Engenharia Elétrica
UFSJ - Universidade Federal de São João del-Rei

13 de março de 2019

Objetivo

Objetivo Principal

Apresentar um resumo da linguagem VHDL.

Objetivos Específicos

- Apresentar os fundamentos para os circuitos combinacionais e sequenciais.
- Mostrar exemplos introdutórios.

Introdução

- A estrutura do código VHDL consiste em três partes:

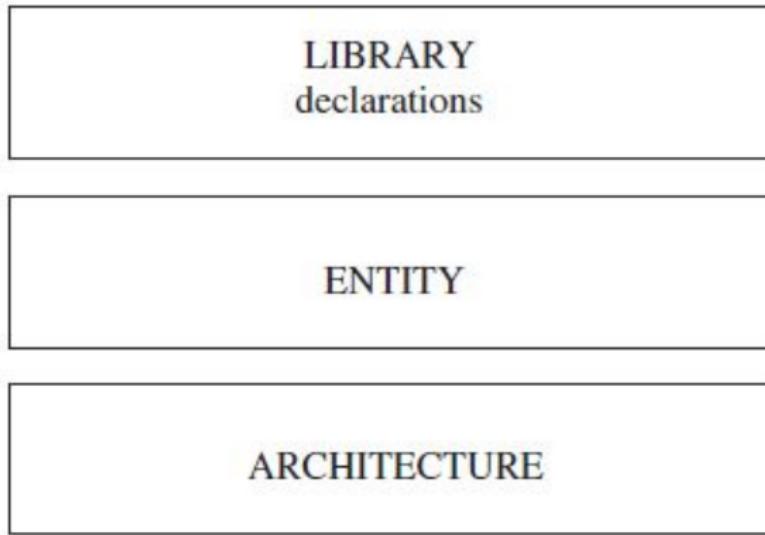


Figura 1: Estrutura de um código VHDL básico.

Introdução - Declarações de Bibliotecas e Pacotes

- A primeira parte do código deve conter uma lista com todas as bibliotecas e pacotes que o compilador necessitará para processar o projeto.
- As bibliotecas std (*standard*) e work são disponibilizadas automaticamente.
- A biblioteca std:
 - Contém definições para os tipos de dados básicos (BIT, BOOLEAN, BIT_VECTOR, INTEGER, ...)
- A biblioteca work indica simplesmente o diretório onde estão armazenados os arquivos do projeto.

Introdução - Declarações de Bibliotecas e Pacotes

- As bibliotecas da IEEE contém várias utilidades.
 - std_logic_1164: especifica o STD_LOGIC (8 níveis) e STD_ULOGIC (9 níveis) sistemas lógicos de valores múltiplos.
 - std_logic_arith: especifica os tipos de dados com sinais e sem sinais, além dos dados relacionados a operações aritméticas e de comparação.
 - std_logic_signed: contém funções que permitem operações com STD_LOGIC_VECTOR a serem executados como se os dados fossem do tipo SIGNED.
 - std_logic_unsigned: contém funções que permitem operações com STD_LOGIC_VECTOR a serem executados como se os dados fossem do tipo UNSIGNED.

Introdução - Declarações de Bibliotecas e Pacotes

- A declaração é feita da seguinte forma:

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

Figura 2: Exemplo de declaração de bibliotecas e pacotes.

Introdução - Declarações de Bibliotecas e Pacotes

Observações:

- O VHDL não é sensível ao tipo de letra (maiúscula ou minúscula).
- Em VHDL, os valores de um bit são escritos com uma par de aspas simples ($x <=' 0'$), ao passo que multibits são escritos com aspas duplas ($y <=" 0010"$).
- São utilizados dois traços (--) para escrever comentários.

Introdução - Entidade (ENTITY)

- A entidade contém duas seções de código denominadas de PORT (declarar todas as entradas e saídas do projeto) e GENERIC (declarar variáveis genéricas / globais).

```
ENTITY entity_name IS
    GENERIC (constant_name: constant_type := constant_value;
              constant_name: constant_type := constant_value;
              ...);
    PORT (port_name: signal_mode signal_type;
          port_name: signal_mode signal_type;
          ...);
    [declarative part]
    [BEGIN]
        [statement part]
    END entity_name;
```

Figura 3: Exemplo de entidade.

Introdução - Arquitetura (ARCHITECTURE)

- A arquitetura contém o código propriamente dito. Sua sintaxe é mostrada a seguir:

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

Figura 4: Exemplo básico de como escrever a seção relacionada a arquitetura do projeto.

Exemplo 01

- Multiplexador com Buffer de três estados.

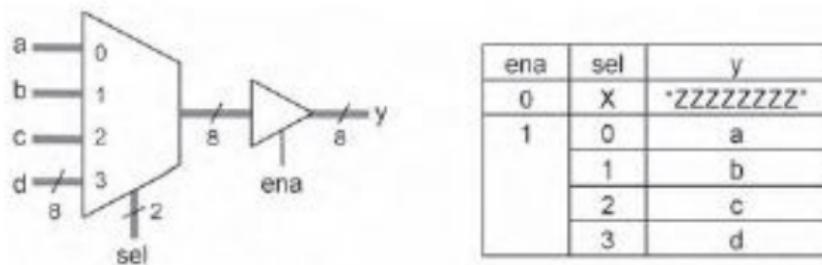


Figura 5: Multiplexador.

Exemplo 01

- Multiplexador com Buffer de três estados.

```
1 Library declarations {  
2 LIBRARY ieee;  
3 USE ieee.std_logic_1164.all;  
4 -----  
5 Entity buffered_mux IS  
6 PORT (a, b, c, d: IN STD_LOGIC_VECTOR(7 DOWNTO 0);  
7 sel: IN INTEGER RANGE 0 TO 3;  
8 ena: IN STD_LOGIC;  
9 y: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));  
10 END buffered_mux;  
11 -----  
12 Architecture myarch OF buffered_mux IS  
13 SIGNAL x: STD_LOGIC_VECTOR(7 DOWNTO 0);  
14 BEGIN  
15 x <= a WHEN sel=0 ELSE --Mux  
16 b WHEN sel=1 ELSE  
17 c WHEN sel=2 ELSE  
18 d;  
19 y <= x WHEN ena='1' ELSE --Tristate buffer  
20 (OTHERS => 'Z');  
21 END myarch;  
22 -----
```

Figura 6: Código fonte.

Tipos de dados predefinidos

Para programar eficientemente, é importante entender as “dimensões” com que os dados podem ser expressos.

Exemplos:

- $a \leq '1'$; – Escalar
- $b \leq FALSE$; – Escalar
- $c \leq "0011"$; – 1D
- $d \leq "ZZZZZZZZ"$; – 1D
- $e \leq -35$; – 1D
- $f \leq '?'$; – 1D
- $g \leq ("0011", "0000", "ZZZZ")$; – 1D x 1D
- $h \leq (5, 255, 0)$; – 1D x 1D
- $i \leq "vhdl"$; – 1D x 1D
- $j \leq (('0', '0', '1'), ('1', '1', '1'), ('Z', 'Z', 'Z'))$; – 2D

Tipos predefinidos sintetizáveis

- BOOLEAN: Valores TRUE e FALSE
- BIT e BIT_VECTOR: Valores '0' e '1'
- STD_LOGIC e STD_LOGIC_VECTOR: Nove Valores ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H' e '-')
- INTEGER, NATURAL e POSITIVE: Valores padrão de $-2^{31} + 1$, 0 ou 1 a $+2^{31} - 1$
- CHARACTER e STRING: Valores do código ASCII estendido
- UNSIGNED e SIGNED: Vetores com valores baseados em STD_LOGIC
- Outros tipos (ponto fixo, ponto flutuante)

Erros Clássicos Cometidos em Assinalamentos

Para assinalar o valor de um objeto a outro, os objetos precisam ser do mesmo tipo.

- **Erro 1:** Os tipos não são iguais.
- **Erro 2:** Os tamanhos não são iguais.
- **Erro 3:** O valor ou a sua representação é inválido (Ex.: BIT não aceita o valor 'Z'; um inteiro não pode ser representado com aspas).
- **Erro 4:** A indexação está incorreta (Ex.: uso incorreto de parênteses).
- **Erro 5:** O operador de assinalamento está incorreto (Ex.: para sinais é " $<=$ ", para variáveis e constantes é " $:=$ ").

Exemplo 02

Considere as seguintes definições:

SIGNAL a: BIT;

SIGNAL b: BIT_VECTOR(7 DOWNTO 0);

SIGNAL c: BIT_VECTOR(1 TO 16);

SIGNAL d: STD_LOGIC;

SIGNAL e: STD_LOGIC_VECTOR(7 DOWNTO 0);

SIGNAL f: STD_LOGIC_VECTOR(1 TO 16);

SIGNAL g: INTEGER RANGE -35 TO 35;

SIGNAL h: INTEGER RANGE 0 TO 255;

SIGNAL i: NATURAL RANGE 0 TO 255;

Exemplo 02

- Qual é a dimensão de cada um desses sinais e quantos bits o compilador atribuirá aos mesmos?
- Dentre os assinalamentos a seguir, todos são ilegais. Quais são os tipos dos erros?

$a \leq 'Z'$

$f(5 \text{ TO } 9) \leq e(7 \text{ DOWNTO } 2)$

$d \leq a$

$b \leq '1111000'$

$e(5 \text{ DOWNTO } 1) \leq c(8 \text{ DOWNTO } 3)$

$d := 'Z'$

$c(16) \leq d(0)$

Há três tipos de objetos em VHDL: CONSTANT, SIGNAL e VARIABLE.

- CONSTANT

```
CONSTANT constant_name: constant_type := constant_value;
```

```
CONSTANT number_of_bits: INTEGER:=16;
CONSTANT mask: BIT_VECTOR(31 DOWNTO 0):=(OTHERS=>'1');
```

Figura 7: Sintaxe de como declarar constante.

- SIGNAL: Definem I/Os e fios internos do circuito.

```
SIGNAL signal_name: signal_type [range] [:= initial_value];
```

```
SIGNAL seconds: INTEGER RANGE 0 TO 59;  
SIGNAL enable: BIT;  
SIGNAL my_data: STD_LOGIC_VECTOR(1 TO 8) := "00001111";
```

Figura 8: Sintaxe de como declarar sinais.

- VARIABLE: Variáveis só podem ser declaradas e usadas em código sequencial. Tais objetos representam somente informação local, ao contrário dos sinais. Por outro lado, sua atualização é imediata.

```
VARIABLE variable_name: variable_type [range] [:= initial_value];
```

```
VARIABLE seconds: INTEGER RANGE 0 TO 59;  
VARIABLE enable: BIT;  
VARIABLE my_data: STD_LOGIC_VECTOR(1 TO 8) := "00001111";
```

Figura 9: Sintaxe de como declarar variáveis.

Exemplo 03

- Criar um contador de 0 a 9.

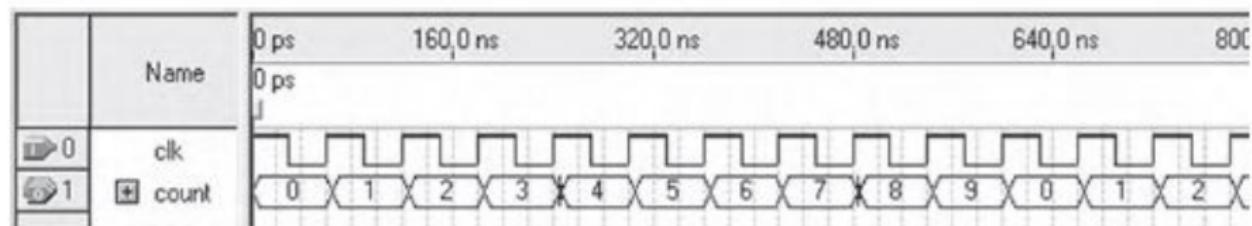


Figura 10: Resultados da simulação de um contador de 0 a 9.

Exemplo 03

```
1 -----
2 ENTITY counter IS
3     PORT (clk: IN BIT;
4             count: OUT INTEGER RANGE 0 TO 9);
5 END counter;
6 -----
7 ARCHITECTURE counter OF counter IS
8 BEGIN
9     PROCESS (clk)
10        VARIABLE temp: INTEGER RANGE 0 TO 10;
11    BEGIN
12        IF (clk'EVENT AND clk='1') THEN
13            temp := temp+1;
14            IF (temp=10) THEN
15                temp := 0;
16            END IF;
17        END IF;
18        count <= temp;
19    END PROCESS;
20 END counter;
21 -----
```

Figura 11: Código fonte de um contador de 0 a 9.

Tipos de Dados Definidos pelo Usuário

A linguagem VHDL também permite que o usuário crie os seus próprios tipos de dados.

- Tipo baseados em inteiros.

```
TYPE type_name IS RANGE type_range;
```

```
TYPE bus_size IS RANGE 8 TO 64;  
TYPE temperature IS RANGE -5 TO 120;
```

Figura 12: Sintaxe de como declarar tipos baseados em inteiros.

Tipos de Dados Definidos pelo Usuário

- Tipo enumerados : empregados principalmente no projeto de máquinas de estados.

```
TYPE type_name IS (state_names);
```

```
TYPE machine_state IS (idle, forward, backward);  
TYPE counter IS (zero, one, two, three);
```

Figura 13: Sintaxe de como declarar tipos enumerados.

Tipos de Dados Definidos pelo Usuário

- Tipos baseados em arranjos: permitem a criação de conjuntos de dados multidimensionais.

```
TYPE type_name IS ARRAY (array_range) OF data_type;
```

1D arrays

```
TYPE vector IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;           --1x8 array
```

```
TYPE BIT_VECTOR IS ARRAY (NATURAL RANGE <>) OF BIT;      --unconstrained array
```

1D×1D

```
TYPE array1D1D IS ARRAY (1 TO 4) OF BIT_VECTOR(7 DOWNTO 0);    --4x8 array
```

```
TYPE vector_array IS ARRAY (1 TO 4) OF vector;                 --4x8 array
```

2D array

```
TYPE array2D IS ARRAY (1 TO 8, 1 TO 16) OF STD_LOGIC;          --8x16 array
```

Figura 14: Sintaxe de como declarar tipos baseados em arranjos.

Tipos de Dados Definidos pelo Usuário

- Subtipos (SUBTYPE): a vantagem de criar um subtipo em vez de um novo tipo é que valores de objetos do subtipo podem ser passados diretamente a objetos do tipo e vice-versa, o que não é permitido fazer entre dois tipos de dados diferentes.

Operadores

Operadores Lógicos.

- NOT
- AND
- NAND
- OR
- NOR
- XOR
- XNOR

Operadores

Operadores Aritméticos.

- Adição (+)
- Subtração (-)
- Multiplicação (*)
- Divisão (/)
- Exponenciação (**)
- Valor absoluto (ABS)
- Resto de a/b com o sinal de a (REM)
- Resto de a/b com o sinal de b (MOD)

Operadores

Operadores de Comparação.

- Igual (=)
- Diferente (/=)
- Maior (>)
- Menor (<)
- Maior ou igual (\geq)
- Menor ou igual (\leq)

Operadores de Deslocamento.

- SLL (shift left logical): Os dados são deslocados para a esquerda, com '0's nas posições vazias.
- SRL (shift right logical): Os dados são deslocados para a direita, com '0's nas posições vazias.
- SLA (shift left arithmetic): Dados deslocados à esquerda, com bit extremo direito nas posições vazias.
- SRA (shift right arithmetic): Dados deslocados à direita, com bit extremo esquerdo nas posições vazias.
- ROL (rotate left): Deslocamento circular para a esquerda.
- ROR (rotate right): Deslocamento circular para a direita.

Operadores

Operadores de Deslocamento. Exemplo:

$a \leq "11001";$

$x \leq a \text{ SLL } 2; \quad \text{-- } x \leq "00100"$

$y \leq a \text{ SLA } 2; \quad \text{-- } y \leq "00111"$

$z \leq a \text{ SLL } -3; \quad \text{-- } z \leq "00011"$

Operadores

- Operador de Concatenação (&). Exemplo:

k: CONSTANT BIT_VECTOR(1 to 4) := "1100";

x <= 'Z' & k(2 to 3) & "1111";

-- x <= "Z1011111"

y <= "Z10" & "1111";

-- y <= "Z1011111"

z <= ('Z' & "10" & "1111");

-- z <= "Z1011111"

Operadores de Atribuição.

- \leq Atribuição de valores a sinais.
- $:=$ Atribuição de valores a variáveis, constantes ou valores iniciais de sinais.
- $=>$ Atribuição de valores a elementos individuais de conjuntos de bits.

Atributos

Podem ser divididos em:

- Atributos de código: tem a finalidade de permitir a construção de códigos flexíveis e monitorar eventos. Tais atributos são precedidos por um "tick"(').
- Atributos de síntese: tem a finalidade de comunicar com o software a fim de definir ou modificar opções de síntese. Nesses atributos, o tick não é utilizado.

Atributos

Atributos para tipos escalares	Atributos para arranjos	Atributos de sinais
t'LEFT	t'LEFT[(N)]	s'DELAYED[(t)]
t'RIGHT	t'RIGHT[(N)]	s'STABLE[(t)]
t'LOW	t'LOW[(N)]	s'QUIET[(t)]
t'HIGH	t'HIGH[(N)]	s'TRANSACTION
t'ASCENDING	t'RANGE[(N)]	s'EVENT
t'IMAGE(X)	t'REVERSE_RANGE[(N)]	s'ACTIVE
t'VALUE(X)	t'LENGTH[(N)]	s'LAST_EVENT
t'POS(X)	t'ASCENDING[(N)]	s'LAST_ACTIVE
t'VAL(X)	t'ELEMENT	s'LAST_VALUE
t'SUCC(X)		s'DRIVING
t'PRED(X)		s'DRIVING_VALUE
t'LEFTOF(X)	Atributos de entidades nomeadas	
t'RIGHTOF(X)	E'SIMPLE_NAME	
t'BASE	E'INSTANCE_NAME	
O'SUBTYPE	E'PATH_NAME	

Código Concorrente × Código Sequencial

- Código Concorrente - Adequado para a construção de circuitos combinacionais.

Suas instruções são:

- WHEN, SELECT e GENERATE.

- Código Sequencial - Pode implementar circuitos sequenciais quanto combinacionais.

Suas instruções são:

- IF, CASE, LOOP e WAIT (essas instruções só podem ser usadas em unidades sequenciais (Ex.: PROCESS, FUNCTION ou PROCEDURE)).

Código Concorrente (WHEN, SELECT, GENERATE)

• WHEN / SELECT

(WHEN-ELSE)

```
assignment WHEN conditions ELSE  
assignment WHEN conditions ELSE  
...;
```

```
x <= a WHEN sel=0 ELSE  
      b WHEN sel=1 ELSE  
      c;
```

(WITH-SELECT-WHEN)

```
WITH identifier SELECT  
  assignment WHEN conditions ELSE  
  assignment WHEN conditions ELSE  
...;
```

```
WITH sel SELECT  
  x <= a WHEN 0,  
      b WHEN 1,  
      c WHEN OTHERS;
```

Figura 15: Sintaxe da instrução WHEN.

Código Concorrente (WHEN, SELECT, GENERATE)

- GENERATE: equivalente a LOOP.

```
label: FOR identifier IN range GENERATE
      [declarations]
      BEGIN]
      (concurrent assignments)
END GENERATE [label];
```

```
G1: FOR i IN 0 TO M GENERATE
    b(i) <= a(M-i);
END GENERATE G1;
```

Figura 16: Sintaxe da instrução GENERATE.

Código Sequencial (PROCESS, IF, CASE, LOOP, WAIT)

- PROCESS: Permite a construção de código sequencial na região do código principal.
- Apenas as instruções IF, CASE, LOOP e WAIT são permitidas.

```
[label:] PROCESS (sensitivity list)
[declarative part]
BEGIN
  (sequential code)
END PROCESS [label];
```

```
PROCESS (clk)
BEGIN
  IF clk'EVENT AND clk='1' THEN
    q <= d;
  END IF;
END PROCESS;
```

Figura 17: Sintaxe da instrução PROCESS.

Código Sequencial (PROCESS, IF, CASE, LOOP, WAIT)

- IF:

```
-----  
IF conditions THEN  
    assignments;  
ELSIF conditions THEN  
    assignments; ...  
ELSE  
    assignments;  
END IF;
```

```
-----  
IF (x=a AND y=b) THEN  
    output <= '0';  
ELSIF (x=a AND y=c) THEN  
    output <= '1';  
ELSE  
    output <= 'Z';  
END IF;
```

Figura 18: Sintaxe da instrução IF.

Código Sequencial (PROCESS, IF, CASE, LOOP, WAIT)

- WAIT: seu principal uso é na construção de formas de onda para simulação (*testbenches*)

(WAIT UNTIL) ----- WAIT UNTIL signal_condition;	WAIT UNTIL clk'EVENT AND clk='1'; -----
(WAIT ON) ----- WAIT ON signal1 [, signal2, ...];	WAIT ON clk, rst; -----
(WAIT FOR) ----- WAIT FOR time;	WAIT FOR 5 ns;

Figura 19: Sintaxe da instrução WAIT.

Código Sequencial (PROCESS, IF, CASE, LOOP, WAIT)

- CASE: instrução muito utilizada para a criação de tabelas verdade.

```
CASE identifier IS
    WHEN value => assignments;
    WHEN value => assignments;
    ...
END CASE;
```

```
CASE sel IS
    WHEN 0 => y<=a;
    WHEN 1 => y<=b;
    WHEN OTHERS => y<=c;
END CASE;
```

Figura 20: Sintaxe da instrução CASE.

Código Sequencial (PROCESS, IF, CASE, LOOP, WAIT)

- LOOP: permite a criação de múltiplas instâncias das mesmas atribuições.

(FOR-LOOP)

```
[label:] FOR identifier IN range LOOP  
    (sequential statements)  
END LOOP [label];
```

(WHILE-LOOP)

```
[label:] WHILE condition LOOP  
    (sequential statements)  
END LOOP [label];
```

```
FOR i IN x'RANGE LOOP  
    x(i) <= a(M-i) AND b(i);  
END LOOP;
```

```
WHILE i < M LOOP  
    ...  
END LOOP;
```

Figura 21: Sintaxe da instrução LOOP.

Código Sequencial (PROCESS, IF, CASE, LOOP, WAIT)

(LOOP with EXIT)

```
... LOOP  
...  
[label:] EXIT [loop_label]  
[WHEN condition];  
...  
END LOOP;
```

(LOOP with NEXT)

```
... LOOP  
...  
[label:] NEXT [loop_label]  
[WHEN condition];  
...  
END LOOP;
```

```
temp := 0;  
FOR i IN N-1 DOWNTO 0 LOOP  
    EXIT WHEN x(i)='1';  
    temp := temp +1;  
END LOOP;
```

```
temp := 0;  
FOR i IN N-1 DOWNTO 0 LOOP  
    NEXT WHEN x(i)='1';  
    temp := temp +1;  
END LOOP;
```

Figura 22: Sintaxe da instrução LOOP.

Exemplo 04

- Contagem de zeros à esquerda.

```
1 -----
2 ENTITY leading_zeros IS
3     GENERIC (N: INTEGER:=8);
4     PORT (x: IN BIT_VECTOR(N-1 DOWNTO 0);
5             y: OUT INTEGER RANGE 0 TO N);
6 END leading_zeros;
7 -----
8 ARCHITECTURE behavioral OF leading_zeros IS
9 BEGIN
10    PROCESS (x)
11        VARIABLE temp: INTEGER RANGE 0 TO N;
12    BEGIN
13        temp := 0;
14        FOR i IN x'RANGE LOOP
15            EXIT WHEN x(i)='1';
16            temp := temp + 1;
17        END LOOP;
18        y <= temp;
19    END PROCESS;
20 END behavioral;
21 -----
```

Figura 23: Código fonte de um contador de zeros à esquerda.

Instrução Auxiliar - ASSERT

- ASSERT: Tem a função de detectar se certas condições foram ou não atendidas.

```
ASSERT condition  
[REPORT "message"]  
[SEVERITY severity_level];
```

Figura 24: Sintaxe da instrução ASSERT.

```
ASSERT a'LENGTH = b'LENGTH  
REPORT "Error: vectors do not have same length!"  
SEVERITY failure;
```

Figura 25: Exemplo do uso da instrução ASSERT.

Foram definidas diversas expansões em VHDL 2008.

- Package *fixed_pkg*
- Package *fixed_generic_pkg*
- Package *float_pkg*
- Package *float_generic_pkg*
- Package *fixed_float_types*

São definidos:

TYPE UFIXED IS ARRAY (INTEGER RANGE <>) OF STD_LOGIC;

–Sem sinal

TYPE SFIXED IS ARRAY (INTEGER RANGE <>) OF STD_LOGIC;

–Com sinal

Exemplos:

SIGNAL a: UFIXED(2 DOWNTO -3); –“aaa.aaa”

a <= “100101”; – $1 \times 2^2 + 1 \times 2^{-1} + 1 \times 2^{-3} = 4,625$

SIGNAL b: SFIXED(4 DOWNTO -1); – “bbbb.b”

b <= “100011”; – complemento de 2 = 01110,1 = -14,5

Ponto Flutuante

São definidos:

TYPE FLOAT IS ARRAY (INTEGER RANGE <>) OF STD_LOGIC;

SUBTYPE FLOAT32 IS FLOAT(8 DOWNTO -23); – 32 bits

SUBTYPE FLOAT64 IS FLOAT(11 DOWNTO -52); – 64 bits

SUBTYPE FLOAT128 IS FLOAT(15 DOWNTO -112); – 128 bits

Exemplo 05

- Meio Somador (Código Principal e Testbench)

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity half_adder is
5   port (a, b : in std_logic;
6         sum, carry : out std_logic
7         );
8 end half_adder;
9
10 architecture arch of half_adder is
11 begin
12   sum <= a xor b;
13   carry <= a and b;
14 end arch;
```

Figura 26: Código fonte de um meio somador.

Exemplo 05

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4
5 entity half_adder_simple_tb is
6 end half_adder_simple_tb;
7
8 architecture tb of half_adder_simple_tb is
9   signal a, b : std_logic;
10  signal sum, carry : std_logic;
11 begin
12   UUT : entity work.half_adder port map (a => a, b => b, sum => sum, carry => carry);
13   |   a <= '0', '1' after 20 ns, '0' after 40 ns, '1' after 60 ns;
14   |   b <= '0', '1' after 40 ns;
15 end tb;
```

Figura 27: Testbench de um meio somador.

Exemplo 05

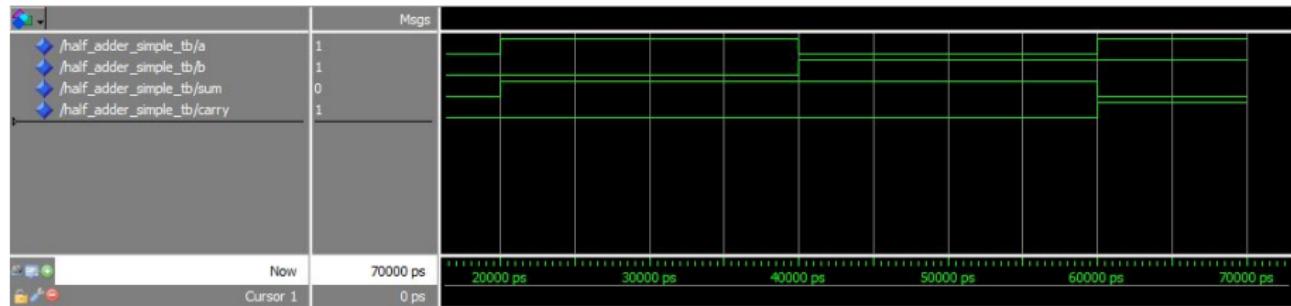


Figura 28: Resposta obtida via testbench.

- PEDRONI, Volnei A. **Digital electronics and design with VHDL.** Morgan Kaufmann, 2008.
- PEDRONI, Volnei A. **Eletônica digital moderna e VHDL.** Rio de Janeiro, RJ: Elsevier, 2010.
- VHDL GUIDE:
<https://vhdlguru.readthedocs.io/en/latest/vhdl/testbench.html>