INF01 118

Técnicas Digitais para Computação

VHDL

UFRGS

Linguagem de Descrição de Hardware

- Linguagens de descrição de hardware (HDL, do inglês, *Hardware Description Language*) fornecem um meio eficiente para o projeto auxiliado por computador (CAD, do inglês, *Computer-Aided Design*) de sistemas lógicos digitais.
- Uma das linguagens utilizadas é denominada VHDL, onde V significa VHSIC, uma abreviação para a área de trabalho conhecida como *Very High-Speed Integrated Circuits*.
- VHDL foi desenvolvida pelo Departamento de Defesa Americano na década de 1970.
- Obejtivo:
  - Modelagem e simulação de sistemas digitais
  - Sintese de sistemas digitais usando ferramentas de CAD

Sintaxe

- Cada arquivo VHDL possui a descrição de um módulo ou unidade lógica que pode ser tão complexa quanto se queira. Para a descrição de um módulo são necessárias duas definições:
- a interface do módulo, ou seja, os sinais de entrada e saída que provem a comunicação com o mundo externo. Esta parte é denominada de entidade (entity).
- a arquitetura do módulo, ou seja, a descrição da lógica que contem e define o módulo. Esta parte é denominada da declaração da arquitetura (architecture).

Sintaxe

```
ENTITY <nome> IS
PORT(
<nome_pino> : <direção do pino>  <tipo do pino e tamanho>;
<nome_pino> : <direção do pino>  <tipo do pino e tamanho>;
<nome_pino> : <direção do pino>  <tipo do pino e tamanho>);
END <nome>;
ARQUITECTURE <nome_arq> OF <nome> IS
--<declarações dos componentes>
COMPONENT <nome do componente>
PORT(
<nome_pino> : <direção do pino>  <tipo do pino e tamanho>;
<nome_pino> : <direção do pino>  <tipo do pino e tamanho>);
END COMPONENT;
--<declaração dos sinais internos>
signal <nome do sinal> :  <tipo do sinal e tamanho>;
signal <nome do sinal> :  <tipo do sinal e tamanho>;
BEGIN
--<instanciação dos componentes>
<nome da instanciação> : <nome do componente>
PORT MAP(<nome do sinal da interface do componente> => <nome do sinal interno usado na
      arquitetura, ...);
--descrição do funcionamento da arquitetura
END;
```


Paralelismo e Concorrência

- Importante: Tudo que estiver descrito na arquitetura da um circuito funciona de **maneira paralela** como em uma "montagem no proto-board", ou seja, não interessa a ordem de descrição, ao menos que se use comandos específicos do VHDL para inserir ordem, ou seja, sequenciamento.
- Mas tem uma maneira de descrever operações em sequencia ..... usando a estrutura **process ( )....**

Léxico

-- : comentário

'A', '0', '1' : caracter

"00011", X " 0A", O "135" : exemplo de strings.

Funções lógicas básicas: not, and, or, xor, nand, nor, xnor

Estrutura para Sequenciamento:

```
PROCESS (<lista de sensibilidade dos sinais>)
BEGIN
...
END PROCESS;
```

clock e reset (circuito sequencial)

todos os sinais usados no process (circuito combinacional)

Léxico

Funções de condição usada dentro do **process ( ):**

```
IF ... THEN
…
ELSE
…
END IF;
```

```
IF … THEN

ELSIF … THEN

ELSE

END IF;
```

```
CASE … IS
WHEN “00” => … ;
WHEN “00” => … ;
WHEN “00” => … ;
WHEN others => … ;
END CASE;
```


Léxico

Funções de condição usada fora do process:

```
G <=  ‘1’ when (x=’0’ and y=’0’) else
      ‘1’ when (x=’0’ and y=’1’) else
      ‘0’ when (x=’1’ and y=’0’) else
      ‘1’;
```

(fio):

```
G <= A;
```

Concatenação:

```
H <= A & B;
```

LAÇO:

Usado fora do processo:

```
For k in 0 to 7 generate
…
end generate;
```

usado dentro do processo:

```
For k in 0 to 7 loop
…
end loop;
```


Tipos de Dados

BIT ou BIT_VECTOR :

pode assumir valor de 0 e 1, e vetores de 0s e 1s.

Ex:

signal enta : bit; -- pode assumir o valor 0 ou 1

signal entx : bit_vector(5 downto 0); -- vetor de 6 bits pode assumir valores de 000000 até 111111

Tipos de Dados

STD ou STD_LOGIC_VECTOR: pode assumir valores de

```
` U`,  -- uninitialized
` X`,  -- strong 0 or 1 (= unknown)
` 0`,  -- strong 0
` 1`,  -- strong 1
` Z`,  -- high impedance
` W`,  -- weak 0 or 1 (= unknown)
` L`,  -- weak 0
` H`,  -- weak 1
` -`,  -- don`t care)
```

Ex: signal entb: std_logic;

signal entc : std_logic_vector(7 downto 0) ;

Neste caso é necessário utilizar uma biblioteca, inserir no começo do arquivo VHDL.

```
LIBRARY IEEE;
use IEEE.std_logic_1164.all;
```


Tipo de Dados

INTEGER : pode assumir valor inteiro variando conforme declarado no range.

Ex: signal x : integer range 0 to 256;

signal y : integer range -127 to 128;

VARIABLE: descreve-se como sinais mas não são sintetizados como sinais

Ex: variable a : integer range 0 to 10;

Begin

A =: 2;

CONSTANT: descreve-se como sinais mas não são sintetizados como sinais

Ex: constant b : std_logic_vector(1 downto 0) := “00”;

Tipos de Dados

SIGNAL or UNSIGNED: usado para possibilitar operações aritméticas com sinais std_logic.

Conversões entre tipos de dados:

```
Library IEEE;
Use use IEEE.std_logic_1164.all
Library IEEE;
Use use IEEE.std_logic_arith.all
```

from	to	function
lv	bv	to_bitvector(x)
bv	lv	to_stdlogicvector(x)

from	to	function
lv	un	unsigned(x)
un	lv	std_logic_vector(x)
un	in	conv_integer(x)
in	un	conv_unsigned(x,len)
in	lv	conv_std_logic_vector(x,len)

onde: lv=std_logic_vector, bv=bit_vector, un=signed/ unsigned, in=integer.

Direção dos pinos

IN, OUT, INOUT

Ex:

```
PORT(
A, B, C : in std_logic_vector (7 downto 0);
X       : out std_logic;
Y       : out std_logic_vector(15 downto 0));
```


Operadores

not, and, or, xor, nand, nor, xnor

= (igual), /= (diferente), >(maior), < (menor), >=, <=

& (concatenação)

+, -. *, / , ** (exponencial)

Exemplo 1

```
ENTITY example1 IS
PORT (
 x1, x2, x3 : IN BIT ;
f : OUT BIT ) ;
END example1 ;
ARCHITECTURE LogicFunc OF example1 IS
BEGIN
f <= (x1 AND x2) OR (NOT x2 AND x3) ;
END LogicFunc ;
```


Exemplo 2

```
ENTITY example2 IS
PORT (
x1, x2, x3, x4 : IN BIT ;
f, g : OUT BIT ) ;
END example2 ;
ARCHITECTURE LogicFunc OF example2 IS
BEGIN
f <= (x1 AND x3) OR (NOT x3 AND x2) ;
g <= (NOT x3 OR x1) AND (NOT x3 OR x4) ;
END LogicFunc ;
```


Exemplo 3

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY shift4 IS
PORT (
R : IN STD_LOGIC_VECTOR (3 DOWNTO 0) ;
Clock : IN STD_LOGIC ;
L, w : IN STD_LOGIC ;
Q : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift4 ;

ARCHITECTURE Behavior OF shift4 IS
BEGIN
PROCESS
BEGIN
WAIT UNTIL Clock'EVENT AND Clock = '1' ;
      IF L = '1' THEN
            Q <= R ;
      ELSE
            Q(0) <= Q(1) ;
            Q(1) <= Q(2);
            Q(2) <= Q(3) ;
            Q(3) <= w ;
END IF ;
END PROCESS ;
END Behavior ;
```

\ ou if (clock'event and clock='1') then

Exemplo 4

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY upcount IS
 PORT (
Clock, Resetn, E : IN STD_LOGIC ;
Q : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)) ;
END upcount ;
ARCHITECTURE Behavior OF upcount IS
SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
BEGIN
PROCESS ( Clock, Resetn )
BEGIN
IF Resetn = '0' THEN
      Count <= "0000" ;
ELSIF (Clock'EVENT AND Clock = '1') THEN
      IF E = '1' THEN
            Count <= Count + 1 ;
      ELSE
            Count <= Count ;
END IF ;
END IF ;
END PROCESS ;
Q <= Count ;
END Behavior ;
```

Exemplo 5

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY maq1 IS
PORT (
Clock, Resetn, w : IN STD_LOGIC ;
z : OUT STD_LOGIC ) ;
END maq1 ;

ARCHITECTURE Behavior OF maq1 IS
TYPE State_type IS (A, B, C) ;
SIGNAL y : State_type ;
BEGIN

PROCESS ( Resetn, Clock )
BEGIN
 IF Resetn = '0' THEN
    y <= A ;
 ELSIF (Clock'EVENT AND Clock = '1')  THEN
   CASE y IS
   WHEN A =>
            IF w = '0' THEN
             y <= A ;
            ELSE y <= B ;
            END IF ;
WHEN B =>
            IF w = '0' THEN
            y <= A ;
            ELSE
            y <= C ;
            END IF ;
WHEN C =>
            IF w = '0' THEN
            y <= A ;
            ELSE y <= C ;
            END IF ;
END CASE ;
END IF ;
END PROCESS ;

z <= '1' WHEN y = C ELSE '0' ;

END Behavior ;
```


Exemplo 6

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

ENTITY maq2 IS
PORT (
Clock, Resetn, w : IN STD_LOGIC ;
z : OUT STD_LOGIC ) ;
END maq2 ;

ARCHITECTURE Behavior OF maq2 IS
TYPE State_type IS (A, B) ;
SIGNAL y : State_type ;

BEGIN

PROCESS ( Resetn, Clock )
BEGIN
IF Resetn = '0' THEN
    y <= A ;
ELSIF (Clock'EVENT AND Clock = '1') THEN
CASE y IS
WHEN A =>
            IF w = '0' THEN
                        y <= A ;
            ELSE
                        y <= B ;
            END IF ;
WHEN B =>
            IF w = '0' THEN
                        y <= A ;
            ELSE
                        y <= B ;
            END IF ;
 END CASE ;
END IF ;
END PROCESS ;

PROCESS ( y, w )
BEGIN
CASE y IS
            WHEN A => z <= '0' ;
            WHEN B => z <= w ;
END CASE ;
END PROCESS ;
END Behavior ;
```