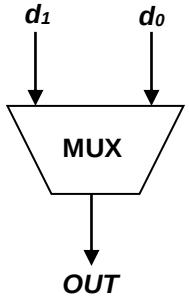
INTRODUÇÃO AO VHDL

Introdução: Projetos de sistemas digitais podem ser realizados em níveis de abstração. O diagrama a seguir mostra os níveis de abstração no projeto e o nível mais baixo nesta hierarquia mostra o ***nível do transistor*** onde os transistores como componentes discretos são conectados juntos para criar o circuito. O nível seguinte ao nível de abstração é o nível das portas lógicas e daí são estas conectadas para criar os circuitos combinatórios, como somadores, multiplexadores, decodificadores e outros e os circuitos seqüênciais como os Flip-Flops. Neste nível as equações são descritas por exepressões booleans e definidas pela sua tabela da verdade. Neste nível é possível a craição de grandes circuitos e até parte de um microprocessador. A metodologia de projeto diz que é muito mais fácil solucionar problemas hierarquicamente do que o problema interiro como um todo. Os blocos combinacionais e seqüenciais são utilizados no nível de fluxo de dados e unidade de controle de um microprocessador. Neste nível são possíveis a transferência dos registros internos e a sua movimentação no fluxo de dados a fim de solucionar o problema. O fluxo de dados contém componentes como ULA, registrador de arquivos, multiplexadores, registradores e outros para realizar uma ou mais operações conforme dita a instrução. Finalmente o nível mais alto hierarquico é o nível comportamental, onde descreve-se o circuito pelo seu comportamento lógico através de uma linguagem de descrição de hardware denominada de VHDL (Very Hardware Description Language).

Nível Comportamental: Um exemplo de descrição no nível comportamental é um circuito Multipleador para 02 entradas d_0 e d_1 e 01 saída ***out_bit***. A sintaxe correta para descrição da linguagem do hardware será:




```
ENTITY Multiplexer IS PORT (
d0,d1,s : IN BIT;
y : OUT BIT);
END multiplexer;

Architecture Behavioral OF Multiplexer IS
 Begin
Process (s,d0,d1)
Begin
y <= d0 WHEN s = '0' ELSE d1;
END Process;
END Behavioral;
```

INTRODUÇÃO AO VHDL

O VHDL é uma linguagem de descrição de hardware o qual serve para a modelagem de sistemas digitais. Igualmente outras linguagens utilizadas por computador, uma vez descrito o programa, um compilador trasnforma a linguagem fonte em linguagem objeto ou código de máquina. Na linguagem VHDL o compilador é um sintetizador que transforma o código fonte VHDL para uma descrição atual do hardware o qual implementa o código. Da descrição é gerada a netlist, dispositivo físico que realiza o código fonte e uma simulação funcional e temporal pode ser realizada para a correção de problemas no circuito.

VHDL PARA PORTA LÓGICA NAND DE 02 ENTRADAS

A implementação da linguagem VHDL para a porta NAND de 02 entradas serve também como template básico para todos os códigos VHDL.
As linhas iniciais com 02 hyfens são comentários. As declarações **Library** e **Use** especificam que a biblioteca **IEEE** é necessária e que todos os componentes da biblioteca podem ser utilizadas. Estas 02 declarações são eqüivalentes para a linha preprocessador "#include" em C++.
Cada componente definedo em VHDL, se é uma simples porta NAND ou um complexo microprocessador, tem 02 partes : primeira entidade e a segunda a arquitetura. A entidade é similar a uma declaração de função em C++ e serve como a interface entre o componente e o lado externo. Cada entidade deve ter um único nome; no examplo, o nome NAND2gate é usedo. A entidade contém uma lista, o qual, como uma lista de parametros, especifica o dado a ser passado dentro e fora do componente. No examplo, existem 02 sinais de entrada chamado x e y do tipo std_logic e um sinal de saída chamado de F do mesmo tipo. O tipo std_logic é como um bit mas contém valores adicionais além de apenas 0 e 1. A arquitetura é uma definição de compoenente e contém o codigo que realiza a operação do componente. Para cada arquitetura se necessita especificar seu nome e qual entidade ela é; no examplo, o nome é Dataflow e ela é para a entidade NAND2 gate. É possível para uma entidade ter mais do que uma arquitetura uma vez uma entidade pode ser implementada em mais do que um modo. Dentro do corpo da arquitetura, pode-se ter uma ou mais declarações concorrentes. Diferentes declarações em C++ onde elas são executadas em ordem sequencial, declaração concorrente no corpo da arquitetura são executadas em paralelo. Assim, a ordenação desta declaração é irrelevante. O símbolo "<=" é usedo na declaração designação do sinal. Apenas como uma declaração regular de designação, a expressão do lado direitp é avaliada primeiro e o resultado é designado para o sinal do lado esquerdo. O operador nand é um operador pronto.

Princípios de\Projeto Lógico Digital

A seguir uma lista de operadores lógicos é apresentada.

A.1.5 Operadores dos Dados

O VHDL listado a seguir são operadores prontos

EXEMPLOS DE OPERADORES LÓGICOS

Operadores Lógicos	**Operação**	**Exemplo**
AND	AND	a AND b
OR	OR	a OR b
NOT	NOT	NOT a
NAND	NAND	a NAND b
NOR	NOR	a NOR b
XOR	XOR	a XOR b
XNOR	XNOR	a XNOR b
Operadores Aritméticos		
+	Adição	a + b
-	Subtração	a - b
*	Multiplicação	a * b
/	Divisão	a / b

MOD	Módulo	a MOD b
REM	Resto	a REM b
**	Exponenciação	a EXP b
&	Concatenação	‘a' & ‘b’
ABS	Absoluto	
Operadores Relacionais		
=	Igual	
/=	Diferente	
<	Menor	
<=	Menor Iugal	
>	Maior	
>=	Maior Igual	
Operadores Deslocamento		
sll	Deslocamento lógico a esquerda	
srl	Deslocamento lógico a direita	
sla	Deslocamento aritmético a esquerda	
sra	Deslocamento lógico a direita	
rol	Giro a esquerda	
Ror	Giro a Direita	

1.6 ENTIDADE

Uma declaração **ENTIDADE** declara a interface externa ou usuário do módulo similar à declaração de uma função. Ela especifica o nome da entidade e sua interface. A interface consiste dos sinais a serem passados dentro ou fora da entidade.
Sintaxe :

ENTITY nome-da-entidade **IS**
PORT (listagem-dos-nomes-dos -portos-e-tipos);
END nome-da-entidade;

Exemplo:

```
ENTITY Siren IS PORT (
M: IN BIT;
D: IN BIT;
V: IN BIT;
S: OUT BIT);
END Siren;
```

1.7 ARQUITETURA

O corpo ARQUITETURA define a implementação atual da funcionalidade da entidade. Ela é similar a definição ou implementação da função. A sintaxe para a arquitetura varia e depende sobre o modelo (dataflow, behavioral, ou structural) a ser utilizado.

Sintaxe para o modelo dataflow :

ARCHITECTURE nome-arquitetura **OF** nome-entidade **IS**
signal-declarations;

BEGIN
concurrent-statements;
END nome-arquitetura;

As declarações concorrentes são executadas concorrentemente.

Exemplo:

ARCHITECTURE Siren_Dataflow OF Siren IS
SIGNAL term_1: BIT;

BEGIN
term_1 <= D OR V;
S <= term_1 AND M;
END Siren_Dataflow;

Sintaxe para o modelo behavioral:

ARCHITECTURE nome-arquitetura **OF** nome-entidade **IS**
declarações dos sinais;
definições das funções;
definições dos procedimentos;

BEGIN

Blocos-processo;
Declarações-concorrentes;
END nome-arquitetura;

Declarações dentro do bloco-processo são executadas sequencialmente. Entretanto, o proprio bloco-processo é uma declaração concorrente.

Exemplo:

ARCHITECTURE *Siren_Behavioral* ***OF*** *Siren* ***IS***
SIGNAL *term_1: BIT;*
BEGIN
PROCESS *(D, V, M)*
BEGIN
term_1 <= D OR V;
S <= term_1 AND M;
END PROCESS;
END *Siren_Behavioral;*

Sintaxe para o modelo estrutural

ARCHITECTURE nome-arquitetura **OF** nome-entidade **IS**
declarações-componentes;
declarações-sinais;

BEGIN
nome-instance : declaração-PORT MAP;
declaração-concorrente;
END nome-arquitetura;

Para cada declaração usada de componente, deve exisitir uma entidade correspondente e arquitetura para aquele componente.
A declaração PORT MAP são declarações concorrentes.

Exemplo:

```
ARCHITECTURE Siren_Structural OF Siren IS
COMPONENT myOR PORT (
in1, in2: IN BIT;
out1: OUT BIT);
END COMPONENT;
SIGNAL term1: BIT;
BEGIN
U0: myOR PORT MAP (D, V, term1);
S <= term1 AND M;
END Siren_Structural;
```

1.8 PACKAGE

Um **PACKAGE** proporciona um mecanismo para agrupar juntas e dividir declarações que são utilizadas por várias unidades entidades.

Um package próprio inclui uma declaração e, opcionalmente, um corpo. A declaração package e corpo são usualmente armazenadas juntas num arquivo separado do resto das unidades de projeto. O nome do arquivo dado por este deve ser o mesmo nome do arquivo package. A fim de completar o projeto para sintetizar corretamente usando MAX+PLUS II, deve-se primeiro sintetizar o package comoa uma unidade separada. Depois pode-se sintetizar a unidade que utiliza o package.

Declaração PACKAGE e BODY

A declaração **PACKAGE** contém declarações que podem ser divididas entre diferentes unidades entidades. Ela proporciona a interface, que são, itens que são visíveis para as outras unidades entidades. O opcional **PACKAGE BODY** contém as implementações das funções e procedimentos que são declarados na declaração PACKAGE.

Sintaxe para a declaração **PACKAGE** :

PACKAGE package-name IS
type-declarations;

subtype-declarations;
signal-declarations;
variable-declarations;
constant-declarations;
component-declarations;
function-declarations;
procedure-declarations;
END package-name;

Sintaxe para **PACKAGE body** :

PACKAGE BODY nome-package IS
definições-funções; -- para funções declaradas na declaração package
definições-procedimentos; -- para procedimentos declarados na declaração package
END package-name;

Exemplo:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
PACKAGE my_package IS
SUBTYPE bit4 IS std_logic_vector(3 DOWNTO 0);
FUNCTION Shiftright (input: IN bit4) RETURN bit4; -- declare a function
SIGNAL mysignal: bit4; -- a global signal
END my_package;

PACKAGE BODY my_package IS
-- implementation of the Shiftright function
FUNCTION Shiftright (input: IN bit4) RETURN bit4 IS
BEGIN
RETURN '0' & input(3 DOWNTO 1);
END shiftright;
END my_package;
```

Usando um PACKAGE

To use a package, you simply include a LIBRARY and USE statement for that package. Before synthesizing the
module that uses the package, you need to first synthesize the package by itself as a top-level entity.

Syntax:

LIBRARY WORK;
USE WORK.package-name.ALL;

Exemplo:

```
LIBRARY work;
USE work.my_package.ALL;
ENTITY test_package IS PORT (
```

```
x: IN bit4;
z: OUT bit4);
END test_package;
ARCHITECTURE Behavioral OF test_package IS
BEGIN
mysignal <= x;
z <= Shiftright(mysignal);
END Behavioral;
```

2. Dataflow Model Concurrent Statements

Concurrent statements used in the dataflow model are executed concurrently. Hence, the ordering of these
statements does not affect the resulting output.

2.1 Concurrent Signal Assignment

Assigns a value or the result of evaluating an expression to a signal. This statement is executed whenever a
signal in its expression changes value. However, the actual assignment of the value to the signal takes place after a
certain delay and not instantaneously as for variable assignments.

Syntax:
signal <= expression;

Exemplo:

```
y <= '1';
z <= y AND (NOT x);
```

2.2 Conditional Signal Assignment

Selects one of several different values to assign to a signal based on different conditions. This statement is
executed whenever a signal in any one of the value or condition changes.
Syntax:
signal <= value1 WHEN condition ELSE
value2 WHEN condition ELSE
…
value3;

Exemplo:

```
z <= in0 WHEN sel = "00" ELSE
in1 WHEN sel = "01" ELSE
in2 WHEN sel = "10" ELSE
in3;
```

2.3 Selected Signal Assignment

Selects one of several different values to assign to a signal based on the value of a select expression. This
statement is executed whenever a signal in the expression or any one of the value changes.
Syntax:
WITH expression SELECT
signal <= value1 WHEN choice1,
value2 WHEN choice2 | choice3,
…
value4 WHEN choice4;

All possible choices for the expression must be given. The keyword OTHERS can be used to denote all remaining
choices.

Exemplo:

WITH sel SELECT
z <= in0 WHEN "00",
in1 WHEN "01",
in2 WHEN "10",
in3 WHEN OTHERS;

Dataflow Model Example

-- outputs a 1 if the 4-bit input is a prime number, 0 otherwise
ENTITY Prime IS PORT (
number: IN BIT_VECTOR(3 DOWNTO 0);
yes: OUT BIT);
END Prime;
ARCHITECTURE Prime_Dataflow OF Prime IS
BEGIN
WITH number SELECT
yes <= '1' WHEN "0001" | "0010",
'1' WHEN "0011" | "0101" | "0111" | "1011" | "1101",
'0' WHEN OTHERS;
END Prime_Dataflow;

3. Behavioral Model Sequential Statements

The behavioral model allows statements to be executed sequentially just like in a regular computer program.
Sequential statements include many of the standard constructs such as variable assignments, if-then-else, and loops.

3.1 PROCESS

The PROCESS block contains statements that are executed sequentially. However, the PROCESS statement itself is

a concurrent statement. Multiple process blocks in an architecture will be executed simultaneously. These process
blocks can be combined together with other concurrent statements.

Syntax:
process-name: PROCESS (sensitivity-list)
variable-declarations;
BEGIN
sequential-statements;
END PROCESS process-name;

A lista-sensibilidade é uma lista de sinais separada por vírgulas no qual o processo é sensível para. Em outraspalavras, sempre que um sinal troca o valor na lista, o processo será executado, isto é, todas as delcarações listadas na ordem sequencial. Depois da última declaração tem sido executada, o processo será suspenso até a próxima vez que um sinal na lista sensibilidade altera o valor antes dele é novamente executado.

Exemplo:

PROCESS (D, V, M)
BEGIN
term_1 <= D OR V;
S <= term_1 AND M;
END PROCESS;

3.2 Sequential Signal Assignment

Designa o valor para um sinal. Esta declaração é apenas como sua concorrente counterpart exceto que ela é executada sequencialmente, isto é, somente quando a execução alcança ela.

Sintaxe :

signal <= expression;

Exemplo:

y <= '1';
z <= y AND (NOT x);

3.3 Variable Assignment

Designa um valor ou o resultado da avaliação de uma expressão para o valor da variável. Sempre designada a variável instataneamente sempre que esta declaração é executada.
Variáveis são somente delcaradas dentro do bloco processo.

Sintaxe :
signal := expression;

Exemplo:

y := '1';
yn := NOT y;

3.4 WAIT

When a process has a sensitivity list, the process always suspends after executing the last statement. An alternative to using a sensitivity list to suspend a process is to use a WAIT statement, which must also be the first statement in a process1.
Syntax2:
WAIT UNTIL condition;

Exemplo:

-- suspend until a rising clock edge
WAIT UNTIL clock'EVENT AND clock = '1';

3.5 IF THEN ELSE

Syntax:
IF condition THEN
sequential-statements1;
ELSE
sequential-statements2;
END IF;
IF condition1 THEN
sequential-statements1;
ELSIF condition2 THEN
sequential-statements2;
…
ELSE
sequential-statements3;
1 This is only a MAX+PLUS II restriction.
2 There are three different formats of the WAIT statement, however,

MAX+PLUS II only supports one.
END IF;

Exemplo:

IF count /= 10 THEN -- not equal
count := count + 1;
ELSE
count := 0;
END IF;

3.6 CASE

Syntax:

CASE expression IS
WHEN choices => sequential-statements;
WHEN choices => sequential-statements;
…
WHEN OTHERS => sequential-statements;
END CASE;

Exemplo:

CASE sel IS
WHEN "00" => z <= in0;
WHEN "01" => z <= in1;
WHEN "10" => z <= in2;
WHEN OTHERS => z <= in3;
END CASE;

3.7 NULL

The NULL statement does not perform any actions.
Syntax:
NULL;

3.8 FOR

Syntax:
FOR identifier IN start [TO | DOWNTO] stop LOOP
sequential-statements;
END LOOP;

Loop statements must have locally static bounds3. The identifier is implicitly declared, so no explicit declaration
of the variable is needed.

Example:

sum := 0;
FOR count IN 1 TO 10 LOOP
sum := sum + count;
3 This is only a MAX+PLUS II restriction.
END LOOP;

3.9 WHILE4

Syntax:
WHILE condition LOOP
sequential-statements;

END LOOP;

3.10 LOOP4

Syntax:
LOOP
sequential-statements;
EXIT WHEN condition;
END LOOP;

3.11 EXIT4

The EXIT statement can only be used inside a loop. It causes execution to jump out of the innermost loop and is
usually used in conjunction with the LOOP statement.
Syntax:
EXIT WHEN condition;

3.12 NEXT

The NEXT statement can only be used inside a loop. It causes execution to skip to the end of the current iteration
and continue with the beginning of the next iteration. It is usually used in conjunction with the FOR statement.

Syntax:
NEXT WHEN condition;
Example:

sum := 0;
FOR count IN 1 TO 10 LOOP
NEXT WHEN count = 3;
sum := sum + count;
END LOOP;

3.13 FUNÇÃO

Syntax for function declaration:
4 Not supported by MAX+PLUS II.

FUNCTION function-name (parameter-list) RETURN return-type;
Syntax for function definition:
FUNCTION function-name (parameter-list) RETURN return-type IS
BEGIN
sequential-statements;
END function-name;
Syntax for function call:
function-name (actuals);
Parameters in the parameter-list can be either signals or variables of mode IN only.

Exemplo:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY test_function IS PORT (
x: IN std_logic_vector(3 DOWNTO 0);
z: OUT std_logic_vector(3 DOWNTO 0));
END test_function;

ARCHITECTURE Behavioral OF test_function IS
SUBTYPE bit4 IS std_logic_vector(3 DOWNTO 0);
FUNCTION Shiftright (input: IN bit4) RETURN bit4 IS
BEGIN
RETURN '0' & input(3 DOWNTO 1);
END shiftright;

SIGNAL mysignal: bit4;
BEGIN
        PROCESS
                BEGIN
                        mysignal <= x;
                        z <= Shiftright(mysignal);
        END PROCESS;
END Behavioral;
```

3.14 PROCEDURE

Syntax for procedure declaration:
PROCEDURE procedure -name (parameter-list);
Syntax for procedure definition:
PROCEDURE procedure-name (parameter-list) IS
BEGIN
sequential-statements;
END procedure-name;
Syntax for procedure call:
procedure -name (actuals);
Parameters in the parameter-list are variables of modes IN, OUT, or INOUT.

Example:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY test_procedure IS PORT (
x: IN std_logic_vector(3 DOWNTO 0);
z: OUT std_logic_vector(3 DOWNTO 0));
END test_procedure;

ARCHITECTURE Behavioral OF test_procedure IS
SUBTYPE bit4 IS std_logic_vector(3 DOWNTO 0);
PROCEDURE Shiftright (input: IN bit4; output: OUT bit4) IS
```

```
BEGIN
        output := '0' & input(3 DOWNTO 1);
END shiftright;
        BEGIN
                PROCESS
                VARIABLE mysignal: bit4;
                BEGIN
                        Shiftright(x, mysignal);
                        z <= mysignal;
                END PROCESS;
END Behavioral;
```

3.15 Behavioral Model Example

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY bcd IS PORT (
I: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
Segs: OUT std_logic_vector (1 TO 7));
END bcd;

ARCHITECTURE Behavioral OF bcd IS
BEGIN
        PROCESS(I)
                BEGIN
                CASE I IS
                                WHEN "0000" => Segs <= "1111110";
                                WHEN "0001" => Segs <= "0110000";
                                WHEN "0010" => Segs <= "1101101";
                                WHEN "0011" => Segs <= "1111001";
                                WHEN "0100" => Segs <= "0110011";
                                WHEN "0101" => Segs <= "1011011";
                                WHEN "0110" => Segs <= "1011111";
                                WHEN "0111" => Segs <= "1110000";
                                WHEN "1000" => Segs <= "1111111";
                                WHEN "1001" => Segs <= "1110011";
                                WHEN OTHERS => Segs <= "0000000";
                END CASE;
        END PROCESS;
END Behavioral;
```

A.4 Structural Model Statements

The structural model allows the manual connection of several components together using signals. All components used must first be defined with their respective ENTITY and ARCHITECTURE sections, which can be in the
same file or they can be in separate files.
In the topmost module, each component used in the netlist is first declared using the COMPONENT statement. The declared components are then instantiated with the actual components in the circuit using the PORT MAP statement.

SIGNALs are then used to connect the components together according to the netlist.

4.1 COMPONENT Declaration

Declares the name and the interface of a component that is used in the circuit description. For each component declaration used, there must be a corresponding entity and architecture for that component. The declaration name
and the interface must match exactly the name and interface that is specified in the entity section for that component.

Syntax:
COMPONENT component-name IS
PORT (list-of-port-names-and-types);
END COMPONENT;
Example:
COMPONENT half_adder IS PORT (
xi, yi, cin: IN BIT;
cout, si: OUT BIT);
END COMPONENT;

4.2 PORT MAP

The PORT MAP statement instantiates a declared component with an actual component in the circuit by
specifying how the connections to this instance of the component are to be made.
Syntax:
label: component-name PORT MAP (association-list);
The association-list can be specified using either the *positional* or *named* method.
Example (positional association):
SIGNAL x0, x1, y0, y1, c0, c1, c2, s0, s1: BIT;
U1: half_adder PORT MAP (x0, y0, c0, c1, s0);
U2: half_adder PORT MAP (x1, y1, c1, c2, s1);
Example (named association):
SIGNAL x0, x1, y0, y1, c0, c1, c2, s0, s1: BIT;
U1: half_adder PORT MAP (cout=>c1, si=>s0, cin=>c0, xi=>x0, yi=>y0);
U2: half_adder PORT MAP (cin=>c1, xi=>x1, yi=>y1, cout=>c2, si=>s1);

4.3 OPEN

The OPEN keyword is used in the PORT MAP association-list to signify that that particular port is not connected or
used.
Example:
U1: half_adder PORT MAP (x0, y0, c0, OPEN, s0);

4.4 GENERATE

The GENERATE statement works like a macro expansion. It provides a simple way to duplicate similar
components.

Syntax:
label: FOR identifier IN start [TO | DOWNTO] stop GENERATE
port-map-statements;
END GENERATE label;

Exemplo:

```
-- using a FOR-GENERATE statement to generate four instances of the full adder
-- component for a 4-bit adder
ENTITY Adder4 IS PORT (
        Cin: IN BIT;
        A, B: IN BIT_VECTOR(3 DOWNTO 0);
        Cout: OUT BIT;
        SUM: OUT BIT_VECTOR(3 DOWNTO 0));
END Adder4;

ARCHITECTURE Structural OF Adder4 IS
        COMPONENT FA PORT (
        ci, xi, yi: IN BIT;
        co, si: OUT BIT);
END COMPONENT;

        SIGNAL Carryv: BIT_VECTOR(4 DOWNTO 0);
                BEGIN
                Carryv(0) <= Cin;
                Adder: FOR k IN 3 DOWNTO 0 GENERATE
                FullAdder: FA PORT MAP (Carryv(k), A(k), B(k), Carryv(k+1), SUM(k));
                END GENERATE Adder;
                Cout <= Carryv(4);
END Structural;
```

4.5 Structural Model Example

This example is based on the following circuit:
D
M
V S

```
-- declare and define the 2-input OR gate
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
        ENTITY myOR IS PORT (
        in1, in2: IN STD_LOGIC;
        out1: OUT STD_LOGIC);
END myOR;

ARCHITECTURE OR_Dataflow OF myOR IS
        BEGIN
        out1 <= in1 OR in2;
END OR_Dataflow;
```

```
-- topmost module for the siren
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
        ENTITY Siren IS PORT (
        M: IN STD_LOGIC;
        D: IN STD_LOGIC;
        V: IN STD_LOGIC;
        S: OUT STD_LOGIC);
END Siren;

ARCHITECTURE Siren_Structural OF Siren IS
-- declaration of the needed OR gate
        COMPONENT myOR PORT (
        in1, in2: IN STD_LOGIC;
        out1: OUT STD_LOGIC);
END COMPONENT;
-- signal for connecting the output of the OR gate
-- with the input to the AND gate
        SIGNAL term1: STD_LOGIC;
                BEGIN
                        U0: myOR PORT MAP (D, V, term1);
                        S <= term1 AND M;
-- note how we can have both PORT MAP and signal assignment statements
        END Siren_Structural;
```

5. Conversion Routines

5.1 CONV_INTEGER()

Converts a std_logic_vector type to an integer;
Requires:

LIBRARY ieee;
USE ieee.std_logic_unsigned.ALL;
Syntax:
CONV_INTEGER(std_logic_vector)

Exemplo:

```
LIBRARY ieee;
USE ieee.std_logic_unsigned.ALL;
SIGNAL four_bit: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL n: INTEGER;
n := CONV_INTEGER(four_bit);
```

5.2 CONV_STD_LOGIC_VECTOR(,)

Converts an integer type to a std_logic_vector type.

Requires:

LIBRARY ieee;
USE ieee.std_logic_arith.ALL;

Sintaxe:

CONV_STD_LOGIC_VECTOR (integer, number_of_bits)

Exemplo:

```
LIBRARY ieee;
USE ieee.std_logic_arith.ALL;
SIGNAL four_bit: std_logic_vector(3 DOWNTO 0);
SIGNAL n: INTEGER;
four_bit := CONV_STD_LOGIC_VECTOR(n, 4);
```

PORTA AND DE 02 ENTRADAS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
        ENTITY AND2gate IS PORT (
        x: IN STD_LOGIC;
        y: IN STD_LOGIC;
        f: OUT STD_LOGIC);
END AND2gate;

ARCHITECTURE Dataflow OF AND2gate IS

        BEGIN
        f <= x AND y;
END Dataflow;
```

PORTA OR DE 02 ENTRADAS

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

ENTITY OR2gate IS PORT (
        x: IN STD_LOGIC;
        y: IN STD_LOGIC;
        f: OUT STD_LOGIC);
END OR2gate;

ARCHITECTURE Dataflow OF OR2gate IS
        BEGIN
        f <= x OR y;
END Dataflow;
```

PORTA NOT

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
        ENTITY NAND2gate IS PORT (
        x: IN STD_LOGIC;
        f: OUT STD_LOGIC);
END NOTgate;

ARCHITECTURE Dataflow OF NOTgate IS
        BEGIN
        f <=  NOT x;
END Dataflow;
```

PORTA NAND DE 02 ENTRADAS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
        ENTITY NAND2gate IS PORT (
        x: IN STD_LOGIC;
        y: IN STD_LOGIC;
        f: OUT STD_LOGIC);
END NAND2gate;

ARCHITECTURE Dataflow OF NAND2gate IS
        BEGIN
        f <= x NAND y;
END Dataflow;
```

PORTA NOR DE 02 ENTRADAS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
        ENTITY NOR2gate IS PORT (
        x: IN STD_LOGIC;
        y: IN STD_LOGIC;
        f: OUT STD_LOGIC);
END NOR2gate;

ARCHITECTURE Dataflow OF NAND2gate IS
        BEGIN
        f <= x NOR y;
END Dataflow;
```

PORTA XOR DE 02 ENTRADAS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
        ENTITY XOR2gate IS PORT (
```

```
        x: IN STD_LOGIC;
        y: IN STD_LOGIC;
        f: OUT STD_LOGIC);
END XOR2gate;

ARCHITECTURE Dataflow OF XOR2gate IS
        BEGIN
        f <= x XOR y;
END Dataflow;
```

PORTA XNOR DE 02 ENTRADAS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
        ENTITY XNOR2gate IS PORT (
        x: IN STD_LOGIC;
        y: IN STD_LOGIC;
        f: OUT STD_LOGIC);
END XNOR2gate;

ARCHITECTURE Dataflow OF XNOR2gate IS
        BEGIN
        f <= x XNOR y;
END Dataflow;
```

PORTA NOR DE 03 ENTRADAS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
        ENTITY NOR3gate IS PORT (
        x: IN STD_LOGIC;
        y: IN STD_LOGIC;
        z: IN STD_LOGIC;
        f: OUT STD_LOGIC);
END NOR3gate;

ARCHITECTURE Dataflow OF NOR3gate IS
        SIGNAL xory, xoryorz : STD_LOGIC;
                BEGIN
                xory <= x OR y;
                xoryorz <= xory OR z;
                f <= NOT xoryorz;
END Dataflow;
```

PORTA NAND DE 03 ENTRADAS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
        ENTITY NAND3gate IS PORT (
                x: IN STD_LOGIC;
                y: IN STD_LOGIC;
                z: IN STD_LOGIC;
                f: OUT STD_LOGIC);
        END NAND3gate;

ARCHITECTURE Dataflow OF NAND3gate IS
        SIGNAL xandy,xandyandz:STD_LOGIC;
                BEGIN
                xandy <= x AND y;
                xandyandz <= xandy AND z;
                f <= NOT xandyandz;
END Dataflow;
```

PORTA XOR DE 03 ENTRADAS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
        ENTITY XOR3gate IS PORT (
        x: IN STD_LOGIC;
        y: IN STD_LOGIC;
        z: IN STD_LOGIC;
        f: OUT STD_LOGIC);
        END XOR3gate;

ARCHITECTURE Dataflow OF XOR3gate IS
        SIGNAL xxory,xxoryxorz:STD_LOGIC;
                BEGIN
                xxory <= x XOR y;
                xxoryxorz <= xxory XOR z;
                f <= xxoryxorz;
END Dataflow;
```

PORTA XNOR DE 03 ENTRADAS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
        ENTITY XNOR3gate IS PORT (
        x: IN STD_LOGIC;
        y: IN STD_LOGIC;
        z: IN STD_LOGIC;
        f: OUT STD_LOGIC);
        END XNOR3gate;

ARCHITECTURE Dataflow OF XNOR3gate IS
```

```
        SIGNAL xxory,xxoryxorz:STD_LOGIC;
                BEGIN
                xxory <= x XOR y;
                xxoryxorz <= xxory XOR z;
                f <= NOT xxoryxorz;
END Dataflow;
```

As funções booleanas podem ser programadas como a seguir :

S = (A B' C) + (A B C') + (A B C).

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
        ENTITY Siren IS PORT (
        A: IN STD_LOGIC;
        B: IN STD_LOGIC;
        C: IN STD_LOGIC;
        S: OUT STD_LOGIC);
        END Siren;

ARCHITECTURE Dataflow OF Siren IS
        SIGNAL term_1, term_2, term_3: STD_LOGIC;
                BEGIN
                term_1 <= A AND (NOT B) AND C;
                term_2 <= A AND B AND (NOT C);
                term_3 <= A AND B AND C;
                S <= term_1 OR term_2 OR term_3;
END Dataflow;
```

1. AULA_1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY PORTA_E_02_ENTRADAS IS
        Port(a, b : IN BIT;
                c: OUT BIT);
END PORTA_E_02_ENTRADAS;

ARCHITECTURE Structural OF PORTA_E_02_ENTRADAS IS
COMPONENT AND2 PORT(x, y: IN BIT;
Z : OUT BIT);
END COMPONENT;

SIGNAL X1 : BIT;
BEGIN
        G1 : AND2 port map(a,b,X1);
END Structural;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY PORTA_E_02_ENTRADAS IS
        Port(a, b : IN BIT;
                c: OUT BIT);
END PORTA_E_02_ENTRADAS;

ARCHITECTURE Structural OF PORTA_E_02_ENTRADAS IS
BEGIN
        C <= a AND b;
END Structural;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY PORTA_NAO IS
        Port(a : IN BIT;
                b: OUT BIT);
END PORTA_NAO;

ARCHITECTURE Structural OF PORTA_NAO IS
BEGIN
        b <= NOT a;
END Structural;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY PORTA_OU_02_ENTRADAS IS
        Port(a, b : IN BIT;
                c: OUT BIT);
END PORTA_OU_02_ENTRADAS;

ARCHITECTURE Structural OF PORTA_OU_02_ENTRADAS IS
BEGIN
        C <= a OR b;
END Structural;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY PORTA_XOR IS
        Port(a, b : IN BIT;
                x: OUT BIT);
END PORTA_XOR;


ARCHITECTURE Structural OF PORTA_XOR IS
BEGIN
        x <=  (a AND (NOT b)) OR ((NOT a) AND b);
END Structural;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY PORTA_XNOR IS
        Port(a, b : IN BIT;
                x: OUT BIT);
END PORTA_XNOR;

ARCHITECTURE Structural OF PORTA_XNOR IS
BEGIN
        x <=  (a AND b) OR ((NOT a) AND (NOT b));
END Structural;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY PORTA_NE_02_ENTRADAS IS
        Port(a, b : IN BIT;
                x: OUT BIT);
END PORTA_NE_02_ENTRADAS;

ARCHITECTURE Structural OF PORTA_NE_02_ENTRADAS IS
BEGIN
        x <= NOT (a AND b);
END Structural;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY PORTA_NE_03_ENTRADAS IS
        Port(a, b,c : IN BIT;
                x: OUT BIT);
END PORTA_NE_03_ENTRADAS;

ARCHITECTURE Structural OF PORTA_NE_03_ENTRADAS IS
BEGIN
        x <= NOT (a AND b AND c);
END Structural;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY PORTA_NOU_02_ENTRADAS IS
        Port(a, b : IN BIT;
                c: OUT BIT);
END PORTA_NOU_02_ENTRADAS;

ARCHITECTURE Structural OF PORTA_NOU_02_ENTRADAS IS
BEGIN
        C <= NOT (a OR b);
END Structural;

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

```
ENTITY PORTA_NOU_03_ENTRADAS IS
        Port(a, b,c : IN BIT;
                x: OUT BIT);
END PORTA_NOU_03_ENTRADAS;

ARCHITECTURE Structural OF PORTA_NOU_03_ENTRADAS IS
BEGIN
        x <= NOT (a OR b OR c);
END Structural;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY PARIDADE_PAR_03_VAR IS
        Port(a, b, c : IN BIT;
                x: OUT BIT);
END PARIDADE_PAR_03_VAR;

ARCHITECTURE Structural OF PARIDADE_PAR_03_VAR IS
BEGIN
        x <=  a XOR b XOR c;
END Structural;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY PARIDADE_IMPAR_03_VAR IS
        Port(a, b, c : IN BIT;
                x: OUT BIT);
END PARIDADE_IMPAR_03_VAR;

ARCHITECTURE Structural OF PARIDADE_IMPAR_03_VAR IS
BEGIN
        x <=  NOT (a XOR b XOR c);
END Structural;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY funcao_booleana IS PORT(
A, B, C: IN BIT;
F: OUT BIT);
END funcao_booleana;

ENTITY inv IS
        Port(i : IN BIT;
                o: OUT BIT);
END inv;
ARCHITECTURE Structural OF inv IS
BEGIN
        o <= NOT i ;
END Structural;
```

```
ENTITY myand2 IS
        Port(i1, i2 : IN BIT;
                o: OUT BIT);
END myand2;
ARCHITECTURE Structural OF myand2 IS
BEGIN
        o <=  i1 AND i2;
END Structural;

ENTITY myor2 IS
        Port(i1, i2 : IN BIT;
                o: OUT BIT);
END myor2;
ARCHITECTURE Structural OF myor2 IS
BEGIN
        o <=  i1 OR i2;
END Structural;

ARCHITECTURE Structural OF funcao_booleana IS
COMPONENT inv PORT(
i: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT myand2 PORT(
i1, i2: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT myor2 PORT(
i1, i2: IN BIT;
o: OUT BIT);
END COMPONENT;

SIGNAL g,h,i,j,k,l,m,n,o,p,q : BIT;
BEGIN
U1: inv port map(A,g);
U2: inv port map(B,h);
U3: inv port map(C,i);
U4: myand2 port map(A, h, j);
U5: myand2 port map(b, g, k);
U6: myand2 port map(i, h, l);
U7: myand2 port map(A, B, m);
U8: myand2 port map(j, C, n);
U9: myand2 port map(m, C, o);
U10: myor2 port map(n, k, p);
U11: myor2 port map(l, o, q);
U12: myor2 port map(p, q, F);
END Structural;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY funcao_booleana_conjuntiva IS PORT(
A, B, C: IN BIT;
F: OUT BIT);
END funcao_booleana_conjuntiva;

ENTITY inv IS
        Port(i : IN BIT;
                o: OUT BIT);
END inv;
ARCHITECTURE Structural OF inv IS
BEGIN
        o <= NOT i ;
END Structural;

ENTITY myand3 IS
        Port(i1, i2, i3 : IN BIT;
                o: OUT BIT);
END myand3;
ARCHITECTURE Structural OF myand3 IS
BEGIN
        o <=  i1 AND i2 AND i3;
END Structural;

ENTITY myor2 IS
        Port(i1, i2 : IN BIT;
                o: OUT BIT);
END myor2;
ARCHITECTURE Structural OF myor2 IS
BEGIN
        o <=  i1 OR i2;
END Structural;

ARCHITECTURE Structural OF funcao_booleana_conjuntiva IS
COMPONENT inv PORT(
i: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT myor2 PORT(
i1, i2: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT myand3 PORT(
i1, i2, i3: IN BIT;
o: OUT BIT);
END COMPONENT;
```

```
SIGNAL g,h,i,j,k,l,m,n : BIT;
BEGIN
U1: inv port map(A,g);
U2: inv port map(B,h);
U3: inv port map(C,i);
U4: myor2 port map(A, h, j);
U5: myor2 port map(h, g, k);
U6: myor2 port map(B, g, l);
U7: myor2 port map(j,C, m);
U8: myor2 port map(l, i, n);
U9: myand3 port map(m, k, n, F);
END Structural;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY funcao_booleana_conjuntiva_dataflow IS PORT(
A, B, C: IN BIT;
F: OUT BIT);
END funcao_booleana_conjuntiva_dataflow;

ARCHITECTURE DATAFLOW OF funcao_booleana_conjuntiva_dataflow  IS
        BEGIN
                F <= (A OR (NOT B) OR C) AND ((NOT A) OR (NOT B)) AND ((NOT A) OR B OR
(NOT C));
END DATAFLOW;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY funcao_booleana_conjuntiva_dataflow_1 IS PORT(
A, B, C: IN BIT;
F: OUT BIT);
END funcao_booleana_conjuntiva_dataflow_1;

ARCHITECTURE DATAFLOW OF funcao_booleana_conjuntiva_dataflow_1 IS
        BEGIN
                F <= NOT(A OR B OR C) OR NOT(NOT A OR B) OR NOT(A OR NOT B OR C) OR
NOT(NOT B OR NOT C);
END DATAFLOW;

-- A 4-to-1 8-bit multiplexer
LIBRARY ieee;
USE IEEE.std_logic_1164.all;

ENTITY Multiplex_4_x_1 IS
PORT(S: IN std_logic_vector(1 downto 0); -- select lines
D0, D1, D2, D3: IN std_logic_vector(7 downto 0); -- data bus input
Y: OUT std_logic_vector(7 downto 0)); -- data bus output
END Multiplex_4_x_1;
```

```
-- using a process statement

ARCHITECTURE Behavioral OF Multiplex_4_x_1 IS
BEGIN
PROCESS (S,D0,D1,D2,D3)
BEGIN
CASE S IS
WHEN "00" => Y <= D0;
WHEN "01" => Y <= D1;
WHEN "10" => Y <= D2;
WHEN "11" => Y <= D3;
WHEN OTHERS => Y <= (OTHERS => 'U'); -- 8-bit vector of U
END CASE;
END PROCESS;
END Behavioral;

-- Um 8 para 1 8-bit multiplexador
LIBRARY ieee;
USE IEEE.std_logic_1164.all;

ENTITY Multiplex_8_x_1 IS
PORT(
 s2,s1,s0 : IN BIT;  --linhas de seleção
 d0,d1,d2,d3,d4,d5,d6,d7 :IN BIT;  -- canais de entrada
Y: OUT BIT); -- saída
END Multiplex_8_x_1;

ARCHITECTURE Dataflow OF Multiplex_8_x_1 IS
        SIGNAL S : bit_vector(2 downto 0);
BEGIN
        S <= (S2 & S1 & S0);
        WITH S SELECT
        Y <=
                d0 WHEN "000",
                d1 WHEN "001",
                d2 WHEN "010",
                d3 WHEN "011",
                d4 WHEN "100",
                d5 WHEN "101",
                d6 WHEN "110",
                d7 WHEN "111";

END Dataflow;

LIBRARY ieee;
USE IEEE.std_logic_1164.all;
ENTITY Multiplex_8_x_1_dataflow IS
PORT(S : IN std_logic_vector(2 downto 0); -- select lines
d0,d1,d2,d3,d4,d5,d6,d7: IN std_logic_vector(0 downto 0);  -- data bus input
Y: OUT std_logic_vector(0 downto 0)); -- data bus output
```

```
END Multiplex_8_x_1_dataflow;

-- using a process statement

ARCHITECTURE Dataflow OF Multiplex_8_x_1_dataflow IS
BEGIN

        WITH S SELECT
        Y <=
                d0 WHEN "000",
                d1 WHEN "001",
                d2 WHEN "010",
                d3 WHEN "011",
                d4 WHEN "100",
                d5 WHEN "101",
                d6 WHEN "110",
                d7 WHEN "111",
                (OTHERS => 'U') WHEN OTHERS; -- 8-bit vector of U

END Dataflow;

-- MULTIPLEX DE 8 CANAIS
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY mux_8_x_1_estrutural IS PORT(
s2, s1, s0: IN BIT; -- variaveis de seleção
d0,d1,d2,d3,d4,d5,d6,d7 : IN BIT; -- canais de entrada
E : IN BIT; -- enable
F: OUT BIT); -- saída booleana
END mux_8_x_1_estrutural;

ENTITY inv IS
        Port(i : IN BIT;
                o: OUT BIT);
END inv;
ARCHITECTURE Structural OF inv IS
BEGIN
        o <= NOT i ;
END Structural;

ENTITY myand5 IS
        Port(i1, i2, i3, i4, i5 : IN BIT;
                o: OUT BIT);
END myand5;
ARCHITECTURE Structural OF myand5 IS
BEGIN
        o <=  i1 AND i2 AND i3 AND i4 AND i5;
END Structural;

ENTITY myor8 IS
```

```
        Port(i1, i2, i3, i4, i5, i6, i7,i8 : IN BIT;
                o: OUT BIT);
END myor8;
ARCHITECTURE Structural OF myor8 IS
BEGIN
        o <=  i1 OR i2 OR i3 OR i4 OR i5 OR i6 OR i7 OR i8;
END Structural;

ARCHITECTURE Structural OF mux_8_x_1_estrutural IS
COMPONENT inv PORT(
i: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT myand5 PORT(
i1, i2, i3, i4, i5: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT myor8 PORT(
i1, i2, i3, i4, i5, i6, i7, i8: IN BIT;
o: OUT BIT);
END COMPONENT;

SIGNAL g,h,i,j,k,l,m,n,o,p,q : BIT;
BEGIN
U1: inv port map(s2,g);
U2: inv port map(s1,h);
U3: inv port map(s0,i);
U4: myand5 port map(g, h, i, E, d0, j);
U5: myand5 port map(g, h, s0, E, d1, k);
U6: myand5 port map(g, s1, i, E, d2, l);
U7: myand5 port map(g, s1, s0, E, d3, m);
U8: myand5 port map(s2, h, i, E, d4, n);
U9: myand5 port map(s2, h, s0, E, d5, o);
U10: myand5 port map(s2, s1, i, E, d6, p);
U11: myand5 port map(s2, s1, s0, E, d7, q);
U12: myor8 port map(j,k,l,m,n,o,p,q,F);
END Structural;

LIBRARY ieee;
USE IEEE.std_logic_1164.all;

ENTITY demultiplex_3_x_8 IS
PORT(E : IN BIT;
 s2,s1,s0 : IN BIT;  --linhas de seleção
 O0,O1,O2,O3,O4,O5,O6,O7 : OUT BIT); -- saída
END demultiplex_3_x_8;

ARCHITECTURE Dataflow OF demultiplex_3_x_8 IS
```

```
        BEGIN
                O0 <= ((not s2) and (not s1)and (not s0))and E;
                O1 <= ((not s2) and (not s1) and s0)and E;
                O2 <= ((not s2) and (s1) and (not s0))and E;
                O3 <= ((not s2) and s1 and s0)and E;
                O4 <= (s2 and (not s1) and (not s0))and E;
                O5 <= (s2 and (not s1) and s0)and E;
                O6 <= (s2 and s1 and (not s0))and E;
                O7 <= (s2 and s1 and s0)and E;
        END Dataflow;

-- DEMULTIPLEX DE 3 POR 8 SAIDAS LÓGICA NEGATIVA
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY demux_3_x_8_neg_estrutural IS PORT(
s2, s1, s0: IN BIT; -- variaveis de seleção
NE : IN BIT; -- canal de entrada
o0,o1,o2,o3,o4,o5,o6,o7 : OUT BIT); -- saídas
END demux_3_x_8_neg_estrutural;

ENTITY inv IS
        Port(i : IN BIT;
                o: OUT BIT);
END inv;
ARCHITECTURE Structural OF inv IS
BEGIN
        o <= NOT i ;
END Structural;

ENTITY mynand4 IS
        Port(i1, i2, i3, i4 : IN BIT;
                o: OUT BIT);
END mynand4;
ARCHITECTURE Structural OF mynand4 IS
BEGIN
        o <=  NOT(i1 AND i2 AND i3 AND i4);
END Structural;

ARCHITECTURE Structural OF demux_3_x_8_neg_estrutural IS
COMPONENT inv PORT(
i: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT mynand4 PORT(
i1, i2, i3, i4: IN BIT;
o: OUT BIT);
END COMPONENT;

SIGNAL g,h,i,j: BIT;
```

```
BEGIN
U1: inv port map(s2,g);
U2: inv port map(s1,h);
U3: inv port map(s0,i);
U4: inv port map(NE,j);
U5: mynand4 port map(g, h, i,j,o0);
U6: mynand4 port map(g, h, s0,j,o1);
U7: mynand4 port map(g, s1,i,j,o2);
U8: mynand4 port map(g, s1,s0,j,o3);
U9: mynand4 port map(s2, h,i,j,o4);
U10: mynand4 port map(s2, h,s0,j,o5);
U11: mynand4 port map(s2,s1,i,j,o6);
U12: mynand4 port map(s2,s1,s0,j,o7);
END Structural;

-- DEMULTIPLEX DE 3 POR 8 SAIDAS LÓGICA POSITIVA
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY demux_3_x_8_pos_estrutural IS PORT(
s2, s1, s0: IN BIT; -- variaveis de seleção
E : IN BIT; -- canal de entrada
o0,o1,o2,o3,o4,o5,o6,o7 : OUT BIT); -- saídas
END demux_3_x_8_pos_estrutural;

ENTITY inv IS
        Port(i : IN BIT;
                o: OUT BIT);
END inv;
ARCHITECTURE Structural OF inv IS
BEGIN
        o <= NOT i ;
END Structural;

ENTITY myand4 IS
        Port(i1, i2, i3, i4 : IN BIT;
                o: OUT BIT);
END myand4;
ARCHITECTURE Structural OF myand4 IS
BEGIN
        o <=  i1 AND i2 AND i3 AND i4;
END Structural;

ARCHITECTURE Structural OF demux_3_x_8_pos_estrutural IS
COMPONENT inv PORT(
i: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT myand4 PORT(
i1, i2, i3, i4: IN BIT;
```

```
o: OUT BIT);
END COMPONENT;

SIGNAL g,h,i: BIT;
BEGIN
U1: inv port map(s2,g);
U2: inv port map(s1,h);
U3: inv port map(s0,i);
U4: myand4 port map(g, h, i,E,o0);
U5: myand4 port map(g, h, s0,E,o1);
U6: myand4 port map(g, s1,i,E,o2);
U7: myand4 port map(g, s1,s0,E,o3);
U8: myand4 port map(s2, h,i,E,o4);
U9: myand4 port map(s2, h,s0,E,o5);
U10: myand4 port map(s2,s1,i,E,o6);
U11: myand4 port map(s2,s1,s0,E,o7);
END Structural;

-- SOMADOR COMPLETO DE 1 BIT - DATAFLOW
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY somador_completo_1bit_dataflow IS PORT (
ci, xi, yi: IN BIT; -- xi e yi são bits de entrada e ci vem um
co, si: OUT BIT); -- si soma e co vai um
END somador_completo_1bit_dataflow;

ARCHITECTURE Dataflow OF somador_completo_1bit_dataflow IS
BEGIN
co <= (xi AND yi) OR (ci AND (xi XOR yi));
si <= xi XOR yi XOR ci;
END Dataflow;

-- SOMADOR COMPLETO DE 1 BIT - MODO ESTRUTURAL
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY somador_completo_1bit_estrutural IS PORT(
x0, y0, ci: IN BIT; -- x0 e y0 = bit de entrada e ci = vem um
co,s0 : OUT BIT); -- s0 = saída e co = vai um
END somador_completo_1bit_estrutural;

ENTITY myand2 IS
        Port(i1, i2 : IN BIT;
                o: OUT BIT);
END myand2;
ARCHITECTURE Structural OF myand2 IS
BEGIN
        o <=  i1 AND i2;
END Structural;

ENTITY myor2 IS
```

```
        Port(i1, i2 : IN BIT;
                o: OUT BIT);
END myor2;
ARCHITECTURE Structural OF myor2 IS
BEGIN
        o <=  i1 OR i2;
END Structural;

ENTITY myxor2 IS
        Port(i1, i2 : IN BIT;
                o: OUT BIT);
END myxor2;
ARCHITECTURE Structural OF myxor2 IS
BEGIN
        o <=  i1 XOR i2;
END Structural;

ARCHITECTURE Structural OF somador_completo_1bit_estrutural IS
COMPONENT myand2 PORT(
i1, i2: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT myor2 PORT(
i1, i2: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT myxor2 PORT(
i1, i2: IN BIT;
o: OUT BIT);
END COMPONENT;

SIGNAL a,b,c: BIT;

BEGIN
U1: myxor2 port map(x0,y0,a);
U2: myand2 port map(x0,y0,b);
U3: myxor2 port map(a,ci,s0);
U4: myand2 port map(a,ci,c);
U5: myor2 port map(b,c,co);
END Structural;


-- SUBTRATOR COMPLETO DE 1 BIT - DATAFLOW
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY subtrator_completo_1bit_dataflow IS PORT (
ci, xi, yi: IN BIT; -- xi e yi são bits de entrada e ci vem um
co, si: OUT BIT); -- si soma e co vai um
```

```
END subtrator_completo_1bit_dataflow;

ARCHITECTURE Dataflow OF subtrator_completo_1bit_dataflow IS
BEGIN
co <= ((NOT xi) AND (ci OR yi)) OR (yi AND ci);
si <= xi XOR yi XOR ci;
END Dataflow;

-- SUBTRATOR COMPLETO DE 1 BIT - MODO ESTRUTURAL
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY subtrator_completo_1bit_estrutural IS PORT(
x0, y0, ci: IN BIT; -- x0 e y0 = bit de entrada e ci = vem um
co,s0 : OUT BIT); -- s0 = saída e co = vai um
END subtrator_completo_1bit_estrutural;

ENTITY myand2 IS
        Port(i1, i2 : IN BIT;
                o: OUT BIT);
END myand2;
ARCHITECTURE Structural OF myand2 IS
BEGIN
        o <=  i1 AND i2;
END Structural;

ENTITY myor2 IS
        Port(i1, i2 : IN BIT;
                o: OUT BIT);
END myor2;
ARCHITECTURE Structural OF myor2 IS
BEGIN
        o <=  i1 OR i2;
END Structural;

ENTITY myxor2 IS
        Port(i1, i2 : IN BIT;
                o: OUT BIT);
END myxor2;
ARCHITECTURE Structural OF myxor2 IS
BEGIN
        o <=  i1 XOR i2;
END Structural;

ENTITY inv IS
        Port(i1 : IN BIT;
                o: OUT BIT);
END inv;
ARCHITECTURE Structural OF inv IS
BEGIN
        o <=  NOT i1;
```

```
END Structural;

ARCHITECTURE Structural OF subtrator_completo_1bit_estrutural IS
COMPONENT myand2 PORT(
i1, i2: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT myor2 PORT(
i1, i2: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT myxor2 PORT(
i1, i2: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT inv PORT(
i1: IN BIT;
o: OUT BIT);
END COMPONENT;

SIGNAL a,b,c,d,e: BIT;

BEGIN
U1: myxor2 port map(x0,y0,a);
U2: myand2 port map(ci,y0,b);
U3: myxor2 port map(a,ci,s0);
U4: myor2 port map(y0,ci,c);
U5: inv port map(x0,d);
U6: myand2 port map(d,c,e);
U7: myor2 port map(b,e,co);
END Structural;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- O seguinte pacote é necessário para os sinais STD_LOGIC_VECTOR
-- A e B podem ser usados como operações aritméticas de numeros não assinalados.
USE ieee.std_logic_unsigned.all;
ENTITY ula_4bit_dataflow IS PORT (
S: IN std_logic_vector(2 downto 0); -- seleção das operações
A, B: IN std_logic_vector(3 downto 0); -- operandos de entrada
F: OUT std_logic_vector(3 downto 0)); -- saída
END ula_4bit_dataflow;
ARCHITECTURE Dataflow OF ula_4bit_dataflow IS
BEGIN
PROCESS(S, A, B)
BEGIN
CASE S IS
```

```
WHEN "000" => -- passagem de A através
F <= A;
WHEN "001" => -- AND
F <= A AND B;
WHEN "010" => -- OR
F <= A OR B;
WHEN "011" => -- NOT A
F <= NOT A;
WHEN "100" => -- soma
F <= A + B;
WHEN "101" => -- subtrai
F <= A - B;
WHEN "110" => -- incremento
F <= A + 1;
WHEN OTHERS => -- decremento
F <= A - 1;
END CASE;
END PROCESS;
END Dataflow;

LIBRARY ieee;
USE IEEE.std_logic_1164.all;

ENTITY decodificador_3_x_8_neg_dataflow IS
PORT(NE : IN BIT; -- enable
 s2,s1,s0 : IN BIT;  --linhas de seleção
 O0,O1,O2,O3,O4,O5,O6,O7 : OUT BIT); -- saída
END decodificador_3_x_8_neg_dataflow;

ARCHITECTURE Dataflow OF decodificador_3_x_8_neg_dataflow IS
        BEGIN
                        O0 <= NOT((not s2) and (not s1)and (not s0))and (NOT NE);
                        O1 <= NOT((not s2) and (not s1) and s0)and (NOT NE);
                        O2 <= NOT((not s2) and (s1) and (not s0))and (NOT NE);
                        O3 <= NOT((not s2) and s1 and s0)and (NOT NE);
                        O4 <= NOT(s2 and (not s1) and (not s0))and (NOT NE);
                        O5 <= NOT(s2 and (not s1) and s0)and (NOT NE);
                        O6 <= NOT(s2 and s1 and (not s0))and (NOT NE);
                        O7 <= NOT(s2 and s1 and s0)and (NOT NE);
        END Dataflow;

-- DECODIFICADOR DE 3 POR 8 SAIDAS LÓGICA NEGATIVA
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY decodificador_3_x_8_neg_estrutural IS PORT(
s2, s1, s0: IN BIT; -- variaveis de seleção
NE : IN BIT; -- enable
o0,o1,o2,o3,o4,o5,o6,o7 : OUT BIT); -- saídas
END decodificador_3_x_8_neg_estrutural;
```

```
ENTITY inv IS
        Port(i : IN BIT;
                o: OUT BIT);
END inv;
ARCHITECTURE Structural OF inv IS
BEGIN
        o <= NOT i ;
END Structural;

ENTITY mynand4 IS
        Port(i1, i2, i3, i4 : IN BIT;
                o: OUT BIT);
END mynand4;
ARCHITECTURE Structural OF mynand4 IS
BEGIN
        o <=  NOT(i1 AND i2 AND i3 AND i4);
END Structural;

ARCHITECTURE Structural OF decodificador_3_x_8_neg_estrutural IS
COMPONENT inv PORT(
i: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT mynand4 PORT(
i1, i2, i3, i4: IN BIT;
o: OUT BIT);
END COMPONENT;

SIGNAL g,h,i,j: BIT;
BEGIN
U1: inv port map(s2,g);
U2: inv port map(s1,h);
U3: inv port map(s0,i);
U4: inv port map(NE,j);
U5: mynand4 port map(g, h, i,j,o0);
U6: mynand4 port map(g, h, s0,j,o1);
U7: mynand4 port map(g, s1,i,j,o2);
U8: mynand4 port map(g, s1,s0,j,o3);
U9: mynand4 port map(s2, h,i,j,o4);
U10: mynand4 port map(s2, h,s0,j,o5);
U11: mynand4 port map(s2,s1,i,j,o6);
U12: mynand4 port map(s2,s1,s0,j,o7);
END Structural;

LIBRARY ieee;
USE IEEE.std_logic_1164.all;

ENTITY decodificador_3_x_8_pos_dataflow IS
PORT(E : IN BIT;
```

```
 s2,s1,s0 : IN BIT;  --linhas de seleção
 O0,O1,O2,O3,O4,O5,O6,O7 : OUT BIT); -- saída
END decodificador_3_x_8_pos_dataflow;

ARCHITECTURE Dataflow OF decodificador_3_x_8_pos_dataflow IS
        BEGIN
                        O0 <= ((not s2) and (not s1)and (not s0))and E;
                        O1 <= ((not s2) and (not s1) and s0)and E;
                        O2 <= ((not s2) and (s1) and (not s0))and E;
                        O3 <= ((not s2) and s1 and s0)and E;
                        O4 <= (s2 and (not s1) and (not s0))and E;
                        O5 <= (s2 and (not s1) and s0)and E;
                        O6 <= (s2 and s1 and (not s0))and E;
                        O7 <= (s2 and s1 and s0)and E;
        END Dataflow;

-- DECODIFICADOR DE 3 POR 8 SAIDAS LÓGICA POSITIVA
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY decodificador_3_x_8_pos_estrutural IS PORT(
s2, s1, s0: IN BIT; -- variaveis de seleção
E : IN BIT; -- enable
o0,o1,o2,o3,o4,o5,o6,o7 : OUT BIT); -- saídas
END decodificador_3_x_8_pos_estrutural;

ENTITY inv IS
        Port(i : IN BIT;
                o: OUT BIT);
END inv;
ARCHITECTURE Structural OF inv IS
BEGIN
        o <= NOT i ;
END Structural;

ENTITY myand4 IS
        Port(i1, i2, i3, i4 : IN BIT;
                o: OUT BIT);
END myand4;
ARCHITECTURE Structural OF myand4 IS
BEGIN
        o <=  i1 AND i2 AND i3 AND i4;
END Structural;

ARCHITECTURE Structural OF decodificador_3_x_8_pos_estrutural IS
COMPONENT inv PORT(
i: IN BIT;
o: OUT BIT);
END COMPONENT;

COMPONENT myand4 PORT(
```

```
i1, i2, i3, i4: IN BIT;
o: OUT BIT);
END COMPONENT;

SIGNAL g,h,i: BIT;
BEGIN
U1: inv port map(s2,g);
U2: inv port map(s1,h);
U3: inv port map(s0,i);
U4: myand4 port map(g, h, i,E,o0);
U5: myand4 port map(g, h, s0,E,o1);
U6: myand4 port map(g, s1,i,E,o2);
U7: myand4 port map(g, s1,s0,E,o3);
U8: myand4 port map(s2, h,i,E,o4);
U9: myand4 port map(s2, h,s0,E,o5);
U10: myand4 port map(s2,s1,i,E,o6);
U11: myand4 port map(s2,s1,s0,E,o7);
END Structural;

LIBRARY ieee;
USE IEEE.std_logic_1164.all;
ENTITY codificador_3_x_8_neg_dataflow IS
PORT(t1,t2,t3,t4,t5,t6,t7 : IN BIT;  --linhas de entradas
 NO0,NO1,NO2 : OUT BIT); -- saída
END codificador_3_x_8_neg_dataflow;

ARCHITECTURE Dataflow OF codificador_3_x_8_neg_dataflow IS
        BEGIN
                        NO0 <= NOT(t1 or t3 or t5 or t7);
                        NO1 <= NOT(t2 or t3 or t6 or t7);
                        NO2 <= NOT(t4 or t5 or t6 or t7);
        END Dataflow;

-- CODIFICADOR DE 3 POR 8 SAIDA OCTAL
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY codificador_3_x_8_neg_estrutural IS PORT(
t1, t2, t3, t4, t5, t6, t7 : IN BIT; -- variaveis de seleção
No0,No1,No2 : OUT BIT); -- saídas
END codificador_3_x_8_neg_estrutural;

ENTITY mynor4 IS
        Port(i1, i2, i3, i4 : IN BIT;
                o: OUT BIT);
END mynor4;
ARCHITECTURE Structural OF mynor4 IS
BEGIN
        o <=  NOT (i1 OR i2 OR i3 OR i4);
END Structural;
```

```
ARCHITECTURE Structural OF codificador_3_x_8_neg_estrutural IS

COMPONENT mynor4 PORT(
i1, i2, i3, i4: IN BIT;
o: OUT BIT);
END COMPONENT;

BEGIN
U1: mynor4 port map(t1,t3,t5,t7,No0);
U2: mynor4 port map(t2,t3,t6,t7,No1);
U3: mynor4 port map(t4,t5,t6,t7,No2);
END Structural;

LIBRARY ieee;
USE IEEE.std_logic_1164.all;
ENTITY codificador_3_x_8_pos_dataflow IS
PORT(t1,t2,t3,t4,t5,t6,t7 : IN BIT;  --linhas de entradas
 O0,O1,O2 : OUT BIT); -- saída
END codificador_3_x_8_pos_dataflow;

ARCHITECTURE Dataflow OF codificador_3_x_8_pos_dataflow IS
        BEGIN
                        O0 <= t1 or t3 or t5 or t7;
                        O1 <= t2 or t3 or t6 or t7;
                        O2 <= t4 or t5 or t6 or t7;
        END Dataflow;

-- CODIFICADOR DE 3 POR 8 SAIDA OCTAL
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY codificador_3_x_8_pos_estrutural IS PORT(
t1, t2, t3, t4, t5, t6, t7 : IN BIT; -- variaveis de seleção
o0,o1,o2 : OUT BIT); -- saídas
END codificador_3_x_8_pos_estrutural;

ENTITY myor4 IS
        Port(i1, i2, i3, i4 : IN BIT;
                o: OUT BIT);
END myor4;
ARCHITECTURE Structural OF myor4 IS
BEGIN
        o <=  i1 OR i2 OR i3 OR i4;
END Structural;

ARCHITECTURE Structural OF codificador_3_x_8_pos_estrutural IS

COMPONENT myor4 PORT(
i1, i2, i3, i4: IN BIT;
o: OUT BIT);
END COMPONENT;
```

```
BEGIN
U1: myor4 port map(t1,t3,t5,t7,o0);
U2: myor4 port map(t2,t3,t6,t7,o1);
U3: myor4 port map(t4,t5,t6,t7,o2);
END Structural;

LIBRARY ieee;
USE IEEE.std_logic_1164.all;
ENTITY codificador_3_x_8_prioridade_dataflow IS
PORT(t1,t2,t3,t4,t5,t6,t7 : IN BIT;  --linhas de entradas
 O0,O1,O2 : OUT BIT); -- saída
END codificador_3_x_8_prioridade_dataflow;

ARCHITECTURE Dataflow OF codificador_3_x_8_prioridade_dataflow IS
        BEGIN
        O0 <= (t1 AND NOT T2 AND NOT T3 AND NOT T4 AND NOT T5 AND NOT T6 AND
        NOT T7) OR (t3 AND NOT T4 AND NOT T5 AND NOT T6 AND NOT T7) OR (t5 AND
        NOT T6 AND NOT T7) OR t7;
        O1 <= (t2 AND NOT T3 AND NOT T4 AND NOT T5 AND NOT T6 AND NOT T7) OR (t3
        AND NOT T4 AND NOT T5 AND NOT T6 AND NOT T7) OR (t6 AND NOT T7) OR t7;
        O2 <= (t4 AND NOT T5 AND NOT T6 AND NOT T7) OR (t5 AND NOT T6 AND NOT T7)
        OR (t6 AND NOT T7) OR t7;
        END Dataflow;
```