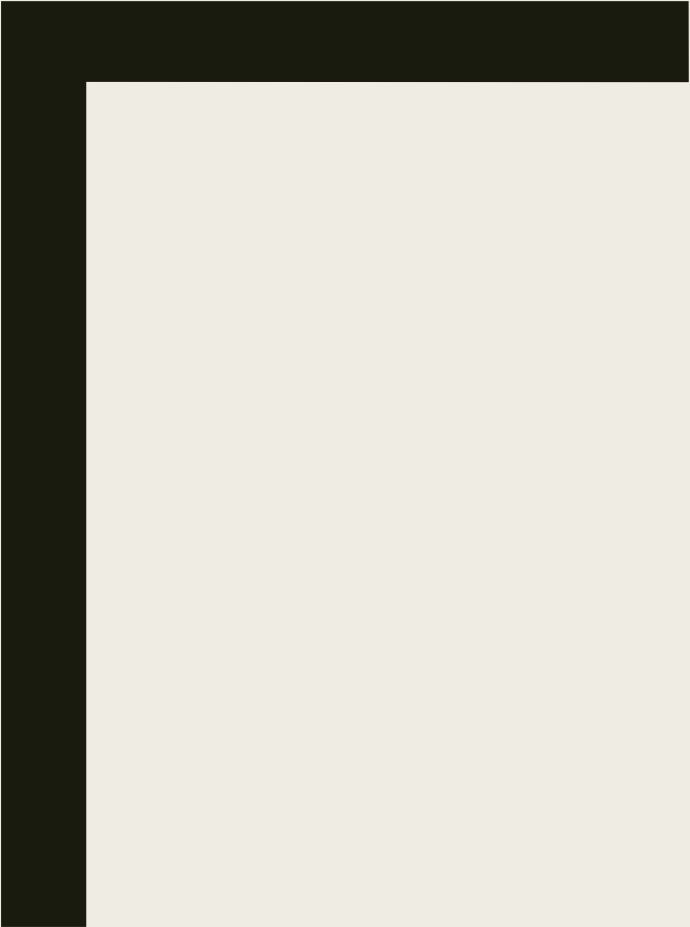




VHDL

Prof. Maurício A Dias
Laboratório de Lógica Digital



Introdução

VHDL é uma linguagem para descrever sistemas digitais utilizada universalmente.

Origem:

VHDL é proveniente de *VHSIC Hardware Description Language*, no contexto do programa americano “Very High Speed Integrated Circuits” (VHSIC), iniciado em 1980.

Vantagens

- a) facilidade de atualização dos projetos
- b) diferentes alternativas de implementação, permitindo vários níveis de abstração
- c) verificação do comportamento do sistema digital, através de simulação
- d) redução do tempo e custo do projeto
- e) eliminação de erros de baixo nível do projeto

Desvantagens

- a) dificuldade para otimização no hardware gerado
- b) necessidade de treinamento para lidar com a linguagem

Características

A linguagem VHDL permite particionar o sistema em diferentes níveis de abstração, quais sejam:

- nível de sistema,
- nível de transferência entre registradores (RT level),
- nível lógico e
- nível de circuito.

Permite três diferentes domínios de descrição:

- comportamental,
- estrutural e
- físico.

Níveis de abstração e descrição

Nível de sistema:

descrição comportamental: *algoritmos*

descrição estrutural: *processadores e memórias*

descrição física: *boards e chips*

Nível RT:

descrição comportamental: *transferências entre registradores*

descrição estrutural: *registradores, unidades funcionais e multiplexadores*

descrição física: *chips e módulos*

Nível Lógico:

descrição comportamental: *equações booleanas*

descrição estrutural: *gates e flip-flops*

descrição física: *módulos e células*

Nível de Circuito:

descrição comportamental: *funções de transferência*

descrição estrutural: *transistores e conexões*

descrição física: *células e segmentos do circuito*

COMENTÁRIOS E NOTAÇÕES NA LINGUAGEM VHDL

Os comentários em VHDL ocorrem após dois traços “- -”.

Os caracteres maiúsculos e minúsculos não tem distinção em VHDL.

Os nomes de variáveis devem iniciar-se com letras alfabéticas, sendo possível utilizar também *dígitos numéricos* e “_”.

O caracter “_” não pode ser usado duplicado, e nem no final de um nome.

ESTRUTURA DE UM PROGRAMA VHDL

- A estrutura básica de um programa em VHDL é composta de três elementos:

library (biblioteca)

entity

architecture

LIBRARY



As primeiras informações contidas num programa VHDL é a declaração das bibliotecas *library (ies)* usada no projeto.

Várias funções e tipos básicos são armazenados em bibliotecas. A biblioteca “IEEE” é sempre incluída.

Ex:

Library IEEE;

Use IEEE.std_logic_1164.all;

Use IEEE.std_logic_unsigned.all;

Observações:

1. a declaração Library IEEE é usada para definir a biblioteca IEEE;
2. a declaração `use IEEE.std_logic_1164.all` é necessária para usar os dados correspondentes à lógica padrão da biblioteca; e
3. a declaração `use IEEE.std_logic_unsigned.all` é necessária para realizar a aritmética não sinalizada.

ENTITY

- O **entity** define a interface(port) do projeto, através dos pinos de entrada (**in**) e saída (**out**) e o tipo do sinal correspondente, no seguinte formato:

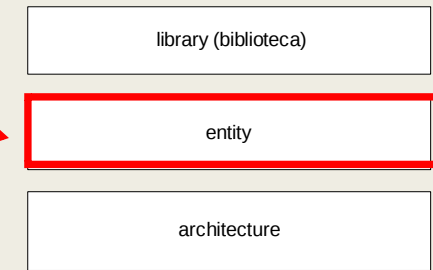
```
entity nome_da_entity is
  port (
    Declaração dos pinos
  );
end [nome_da_entity] ;
```

Exemplo:

```
entity COMPARA is
  port ( A,B: in std_logic; C: out std_logic);
end COMPARA;
```



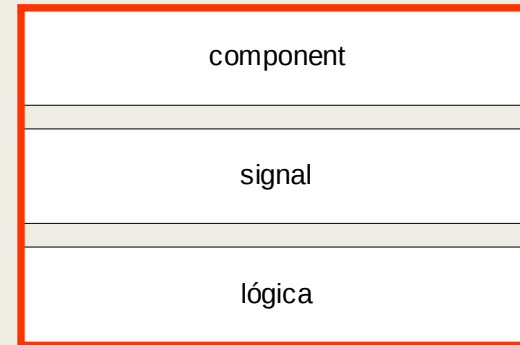
*Interfaces definidas através do exemplo de **entity**.*



ARCHITECTURE

■ A **architecture** define a lógica do circuito e pode ser composta dos seguintes elementos:

- a) component
- b) signal
- c) lógica

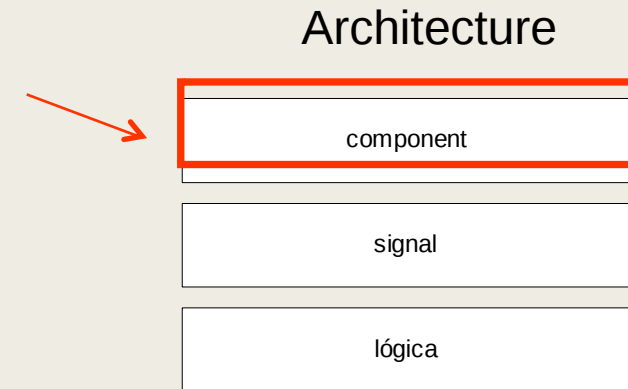


sendo **component** e **signal** declarações de componentes e sinais intermediários opcionais.

O formato para a descrição da arquitetura é o seguinte:

```
Architecture nome_da_architecture of nome_da_entity is
    Declarações opcionais (component e signal)
begin
end [nome_da_architecture];
```

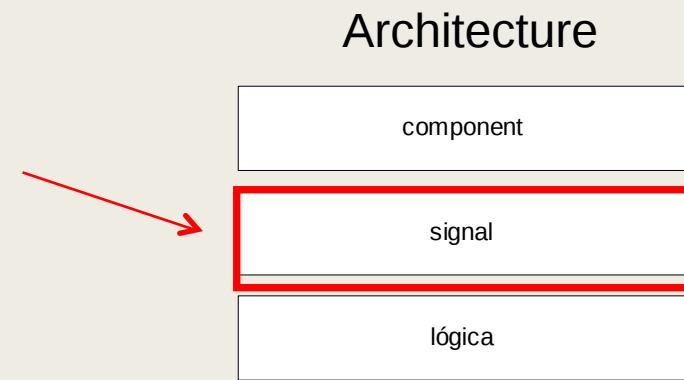
COMPONENT



- Declaração do componente que deve ser projetado através de um outro programa VHDL, ou outra forma de projeto.

```
Component nome_do_componente
port (
    Clk : in std_logic;
    Rst : in std_logic;
    Din : in std_logic;
    Dout : out std_logic
);
end component;
```

SIGNAL



- O **signal** pode ser declarado em entity, architecture ou em package, e serve para a comunicação entre os módulos.

sintaxe:

```
signal identificador (es) : tipo [restrição]  
[:=expressão];
```

Exemplos:

```
signal cont : integer range 50 downto 1;  
signal ground : bit := ´0´;  
signal bus : bit_vector;
```

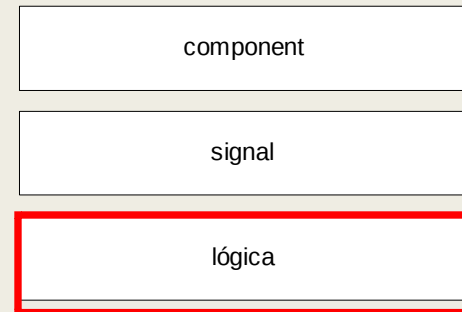
Lógica: Descrição Comportamental

Usa o comando *process*, com o formato:

```
Process ( lista de sensibilidade ) begin  
    descrição lógica  
end process;
```

A *lista de sensibilidade* corresponde aos sinais que devem alterar a saída do circuito, e é composta de todos os sinais de entrada para os circuitos combinatórios.

Para os registradores assíncronos, a lista seria composta do *clock* e do *reset*; e para os registradores síncronos, do *clock*.

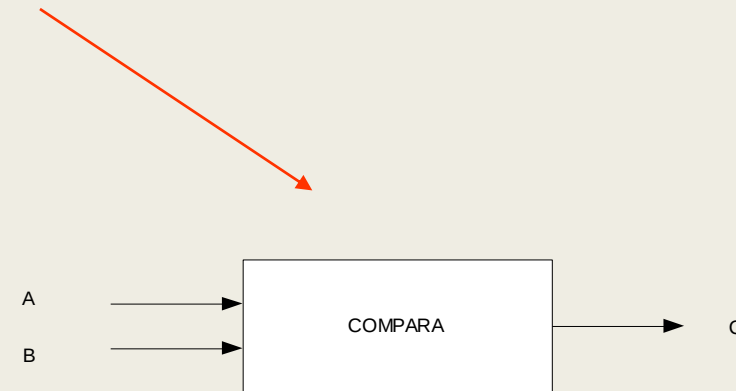


Exemplo de arquitetura, com descrição comportamental

Architecture COMPORTAMENTO of COMPARA is

```
begin
  process (A,B)
    begin
      if(A=B) then C<='1';
      else C<='0';
      end if;
    end process;
end COMPORTAMENTO;
```

Lista de sensibilidade

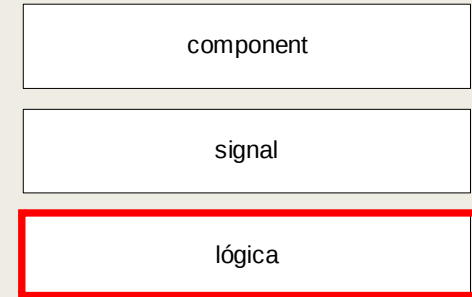


O comando `process (A,B)` indica que os sinais `A` e `B` formam a *lista de sensibilidade*.

A saída `C` será igual a `1` caso as entradas `A` e `B` sejam iguais, e `C` será igual a `0`, caso contrário.

Lógica: Descrição Estrutural

Architecture



Para a descrição estrutural é feita a associação dos pinos do componente com os sinais usados no projeto.

Exemplo:

```
U0: nome_do_componente
  port map (
    Clk => clk_top;
    Rst => rst_top;
    Din => din_top;
    Dout => dout_top
  );
```

No exemplo, **U0** é um label.

Exemplo de arquitetura, usando descrição estrutural

architecture ESTRUTURA of COMPARA is

component XOR_Gate

port (I0, I1: in std_logic; O: out std_logic);

end component;

component NOT_Gate

port (I0: in std_logic; O: out std_logic);

end component;

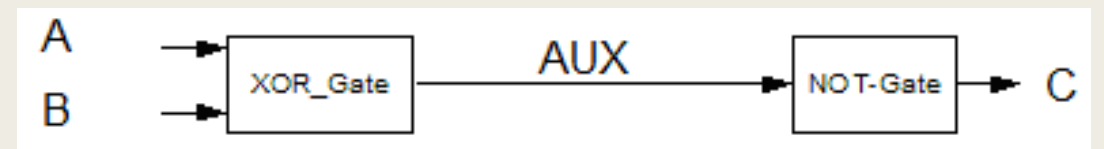
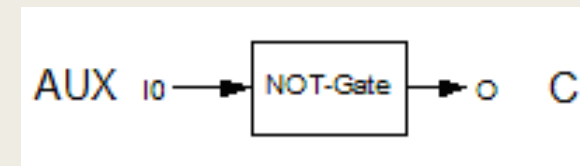
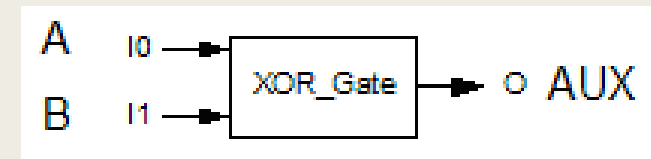
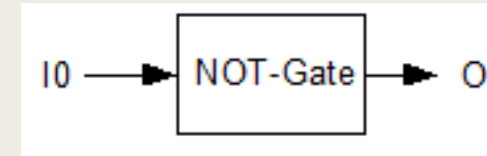
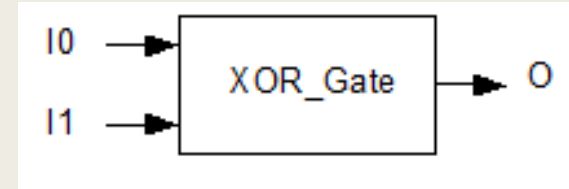
signal AUX: std_logic;

begin

U0: XOR_Gate port map (I0=>A, I1=>B, O=>AUX);

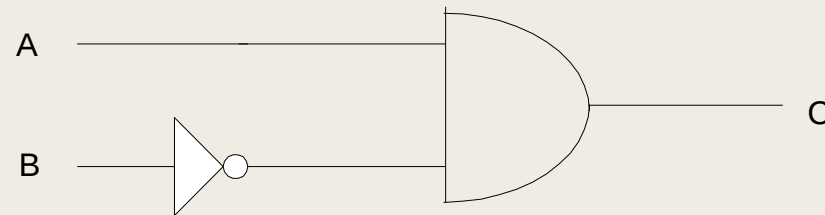
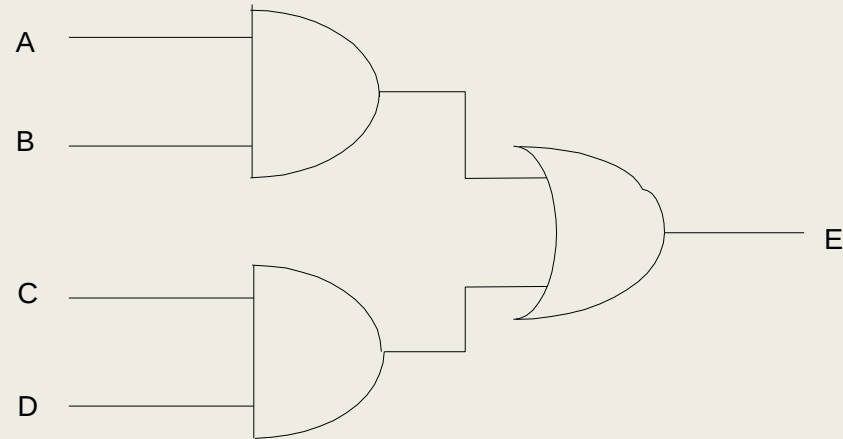
U1: NOT_Gate port map (I0=>AUX, O=>C);

end ESTRUTURA;



Exercício

Definir *Entity* e *Architecture*, usando descrição comportamental e estrutural, para os circuitos:



Tipos de dados pré-definidos

Bit

Assume valores *'0'* e *'1'* .

Bit não tem relação com o tipo boolean.

As vezes, o bit *'1'* deve ser explícito - *bit('1')* , quando confunde-se com *caractere '1'* .

Bit_vector

Designa um conjunto de bits.

Exemplo: "001100" ou x"00FF" .

Boolean

Assume valores *true* e *false*.

É útil para descrições onde um sinal só pode assumir dois valores.

Tipos de dados (cont.)

Real

Sempre ocorre um ponto decimal num valor real.

Exemplos: -1.0 / +2.35 / 37.0 / -1.5E+23

Integer

Representam valores inteiros.

Exemplos: +1 / 1232 / -1234

Character

A linguagem VHDL não é “case sensitive”, exceto para caracteres.

Os caracteres devem ser explicitados entre aspas: ‘a’, ‘x’, ‘0’, ‘1’, ...

Para o caractere ‘1’ a declaração deve ser explícita - `character('1')`, pois caso contrário confunde-se com o bit ‘1’.

String

Este tipo designa um conjunto de caracteres.

Physical, range

Physical

Representa uma medida física como: voltagem, capacitância, tempo, comprimento.

Tipos pré-definidos: *ps*, *ns*, *um*, *ms*, *sec*, ~~*min*~~, *hr*
micrômetro

Range

Define o intervalo de utilização.

sintaxe: *range* *valor_baixo* *to* *valor_alto*
 range *valor_alto* ~~*downto*~~ *valor_baixo*

Exemplos:

integer range 1 to 10

real range 1.0 to 10.0

Declaração sem *range* declara todo o intervalo.

Declaração *range*<> : declaração postergada do intervalo

Tipos definidos

Tipos definidos pelo usuário

*O usuário pode criar tipos de dados através do comando **type**.*

Exemplos:

```
type logic_level is ( '0', '1', 'X', 'Z' )
```

```
type octal is ( '0', '1', '2', '3', '4', '5', '6', '7' )
```

ARRAYS

Coleção de elementos de mesmo tipo.

type word is array (31 downto 0) of bit;

type transform is array (1 to 4) of real;

type register_bank is array (byte range 0 to 132) of integer;

Array sem definição de tamanho.

type vector is array (integer range <>) of real;

Exemplos de arrays pré-definidos.

type string is array (positive range <>) of character;

type bit_vector is array (natural range <>) of bit;

Atribuição de um array: posicional ou por nome.

type a is array (1 to 4) of character;

posicional: ('f', 'o', 'o', 'd')

por nome: (1 => 'f', 3 => 'o', 4 => 'd', 2 => 'o')

CONSTANTES

As constantes tem valores fixos e são usadas somente para leitura.

Consiste de um nome, do tipo, e de um valor (opcional, com declaração posterior).

Sintaxe:

```
constant identificador : tipo [ := expressão ];
```

Exemplo:

```
constant gnd : real := 0.0;
```

As constantes podem ser declaradas em qualquer parte, porém é aconselhável declarar constantes frequentemente utilizadas em um *package*

STANDARD LOGIC

Os valores fixos definidos no `std_logic` são:

<i>Valores</i>	<i>significado</i>
'0'	0
'1'	1
'X'	indefinido forçado
'Z'	alta impedância
'U'	não inicializado
'L'	0 fraco
'H'	1 fraco
'W'	indefinido
'_'	irrelevante

VARIÁVEIS

As variáveis podem ter seus valores alterados durante a execução do programa e são usadas para leitura e escrita.

sintaxe:

```
variable identificador (es) : tipo [restrição] [ :=expressão];
```

exemplos:

```
variable índice : integer range 1 to 50 := 50;
```

```
variable ciclo_de_máquina : time range 10 ns to 50 ns := 10ns;
```

```
variable memória : bit_vector (0 to 7)
```

```
variable x, y : integer;
```

Para a associação de um valor a uma variável sempre se usa o operador :=

Ex:

```
var := var + 1;
```

EXPRESSÕES

Expressões são fórmulas que realizam operações sobre objetos de mesmo tipo.
As operações possíveis são as seguintes:

tipos	operações
lógicas	and, or, nand, nor, xor, not
relacionais	=, /=, <, <=, >, >=
aritméticas	- (unária), abs
aritméticas	+, -
aritméticas	*, /
aritméticas	mod, rem, **
junção	&

Observações

- a) As operações lógicas são realizadas sobre tipos `bit` e `boolean`.
- b) Os operadores aritméticos trabalham sobre inteiros e reais.
- c) Todo tipo físico pode ser multiplicado/dividido por inteiro ou ponto flutuante.
- d) A concatenação é aplicável sobre caracteres, strings, bits, vetores de bits e arrays.

Exemplos:	"ABC" & "xyz"	resulta em:	"ABCxyz"
	"1001" & "0011"	resulta em:	"10010011"

TEMPORIZAÇÃO

ATRASSO

$A \leq B + C$ after 5.0 ns;

$D \leq A + E$; (D recebe o valor antigo de A !!)

ESPERA

$x \leq y$;

$y \leq x$;

wait on clock;

COMANDOS SEQUENCIAIS (1)

ATRIBUIÇÕES

Atribuição de variáveis

$A := B;$

As variáveis não passam valores para fora do processo no qual foram declaradas, ou sejam, as atribuições são locais.

As atribuições são sequenciais, ou seja, a ordem das mesmas são importantes.

Comandos Sequenciais (2)

Atribuição de sinais (para a atribuição de valor inicial em sinais, usa-se o operador `:=`, enquanto que para a atribuição de valores no código da arquitetura, usa-se o operador `<=`).

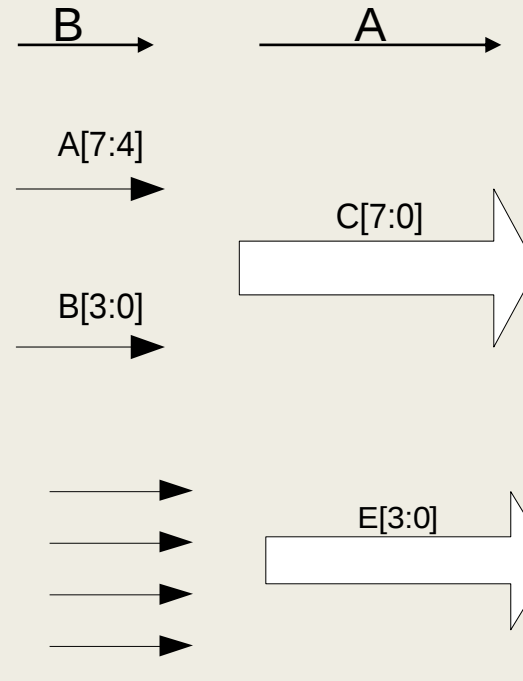
Exemplos:

`signal clk: BIT := '0';`

`A <= B;`

`C <= A (7 downto 4) & B (3 downto 0);`

`E (3 downto 0) <= D & D & D & D;`



Comandos Sequenciais (3)

Atribuição de expressões lógicas

*Quando as operações tem mesma prioridade, deve-se usar os parênteses para indicar prioridade.
A operação **not** tem maior prioridade.*

Exemplos:

a) expressão simples:

A <= B and C;

b) várias operações com prioridades diferentes:

A <= (B and C) or (D and E);

*c) operação **not** que tem maior prioridade:*

A <= not B and C;

Comandos Sequenciais (4)

Atribuição de sinais de saída

Um sinal de saída não pode ser usado como entrada, numa realimentação.

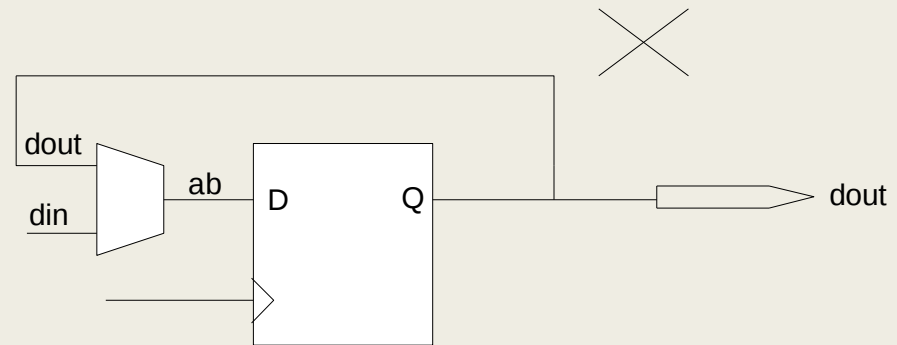
Um exemplo, onde dout é definida como saída.

Entity is

dout : out std_logic;

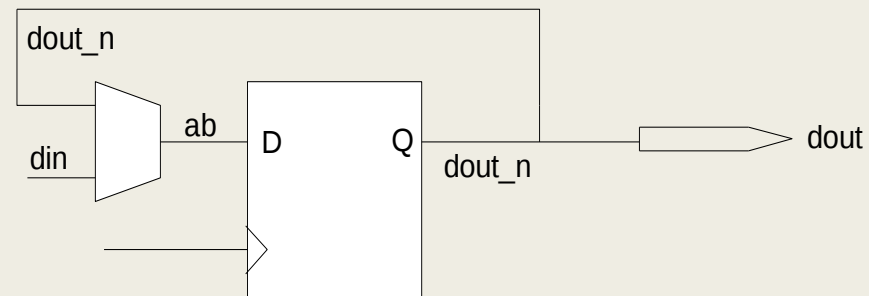
.....

ab <= dout and din;



Para solucionar o problema acima, é necessário definir um sinal intermediário (**dout_n**), diferenciando-o do sinal de saída (**dout**), conforme

ab <= dout_n and din;



Resumo dos operadores de atribuição

operador	significado	exemplo
<code><=</code>	Atribuição de sinal	<code>Aux <= '0'</code>
<code>:=</code>	Atribuição de variável	<code>A := '1'</code>
<code>:=</code>	Inicialização de constantes, sinais e variáveis	<code>Signal aux : bit := '0'</code>
<code>=></code>	Atribuição de valores únicos em vetores	<code>Vetor <= (0=> '0')</code>
<code>=></code>	Atribuição de vários valores em vetores junto com a cláusula <code>others</code>	<code>Vetor <= (0 => '0' , others => '1')</code>

COMANDO IF

O comando IF segue o seguinte formato:

```
if condition then
    sequence_of_statements
{ elsif condition then
    sequence_of_statements }
[ else
    sequence_of_statements ]
end if ;
```

Exemplos de comando if

1) teste de borda de subida:

if clock'event and clock='1' then ...

2) teste de borda de descida:

if clock'event and clock='0' then ...

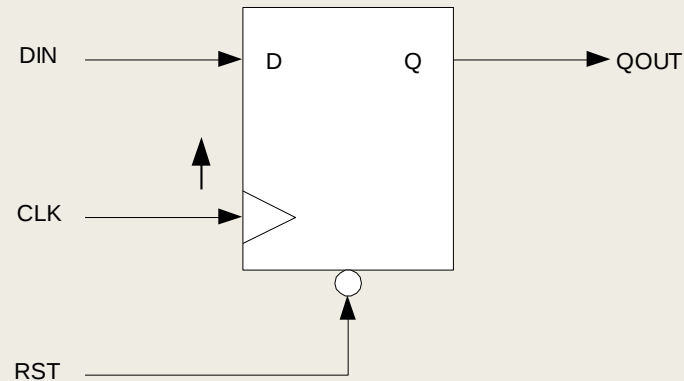
3) if (x) then T := A; end if;
 if (y) then T := B; end if;
 if (z) then T := C; end if;

é equivalente a:

if (z) then T:= C;
elseif (y) then T := B;
elseif (x) then T := A;
end if;

Exercício

- Descrever através de um comando process o flip-flop sensível à borda de subida:

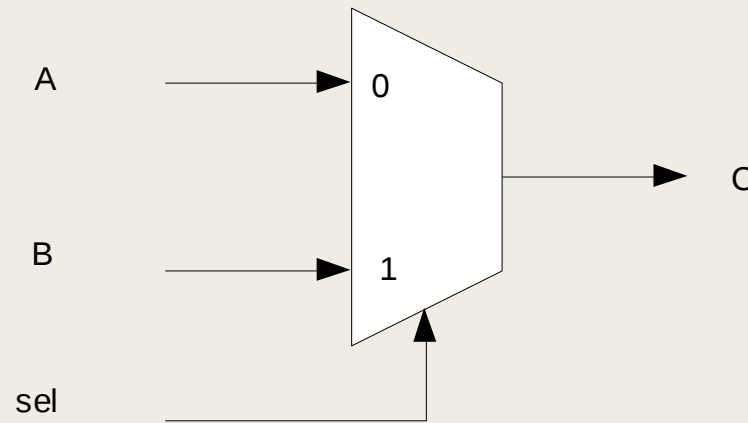


Solução:

```
Process (CLK, RST) begin
    If (RST = '0' ) then QOUT <= '0' ;
    elsif (CLK'event and CLK = '1' ) then QOUT <= DIN;
    end if;
end process;
```

Exercício

- Desenvolver o comando process para o seletor (ou multiplexador)



Solução:

```
Process (A,B,SEL) begin
    If SEL = '0' then C <= A;
    else C <= B;
    end if;
end process;
```

COMANDO CASE

É utilizado basicamente para decodificação.

```
case element_colour is
```

```
    when red =>  
        statements for red;
```

```
    when green | blue =>  
        statements for green or blue;
```

```
    when orange to turquoise =>  
        statements for these colours;
```

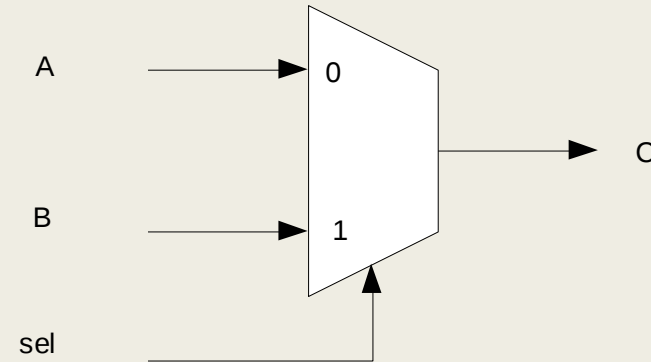
```
end case;
```

Exercício

Codificar o process do exercício anterior utilizando “case” .

Solução:

```
Process (A,B,SEL) begin  
  case SEL is  
    when '0' => C <= A;  
    when '1' => C <= B;  
    when others => C <= 'X' ;  
  end case;  
end process;
```



COMANDO NULL

O comando **NULL** serve para indicar “não faça nada” em uma condição de **case**.

Exemplo:

```
case controller_command is
  when forward => engage_motor_forward;
  when reverse => engage_motor_reverse;
  when idle => null;
end case;
```

EXERCICIO A RESOLVER

1) Qual das sentenças abaixo está incorreta:

variable A,B,C,D: bit_vector (3 downto 0);

variable E,F,G: bit_vector (1downto 0);

variable H, I, J, K: bit;

a) $A \leq B \text{ xor } C \text{ and } D$;

b) $H \leq I \text{ and } J \text{ or } K$;

c) $H \leq I \text{ or } F$;

d) $H \leq A(3) \text{ or } I$;

Exercício a resolver

Qual das sentenças abaixo está correta:

signal A,B,C, D,E: in bit;

signal OU: out bit_vector (3 downto 0);

variable T: in integer;

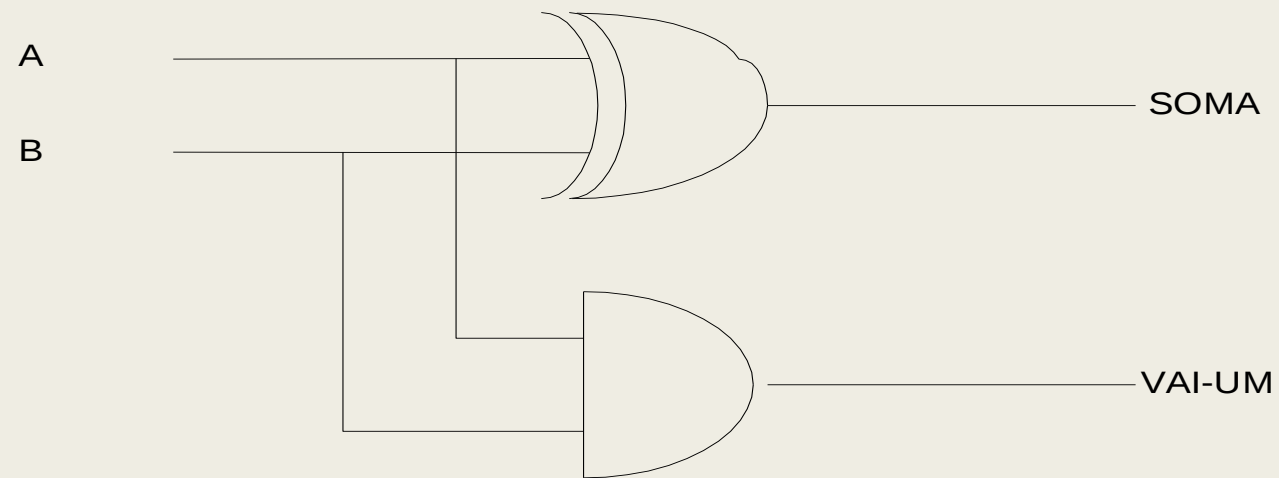
a) *T := A and B;*

b) *E := not T;*

c) *T:= integer (B or C);*

Exercício

- Desenvolver um programa em VHDL para o circuito meio-somador.



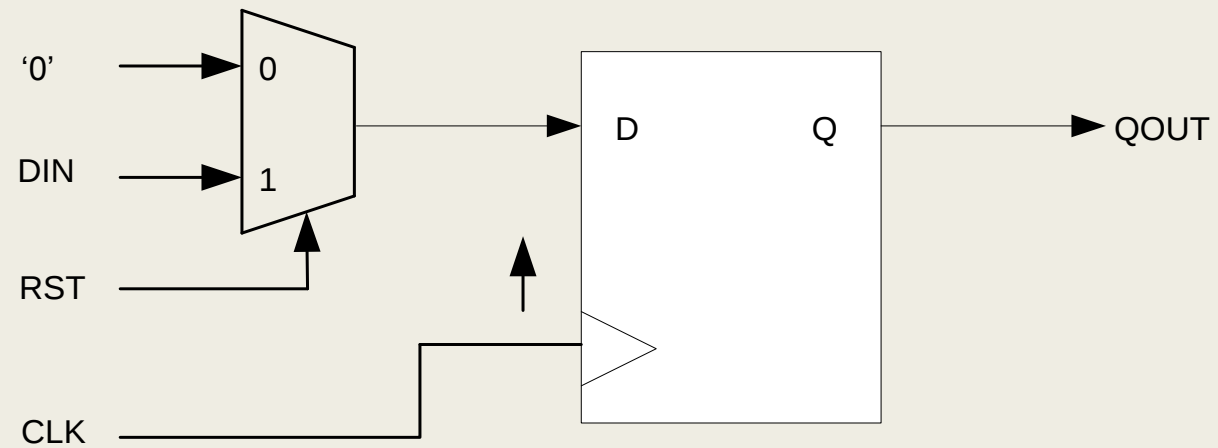
Exercício

- Implementar um circuito decodificador definido pela Tabela:

A	Decode
00	0001
01	0010
10	0100
11	1000

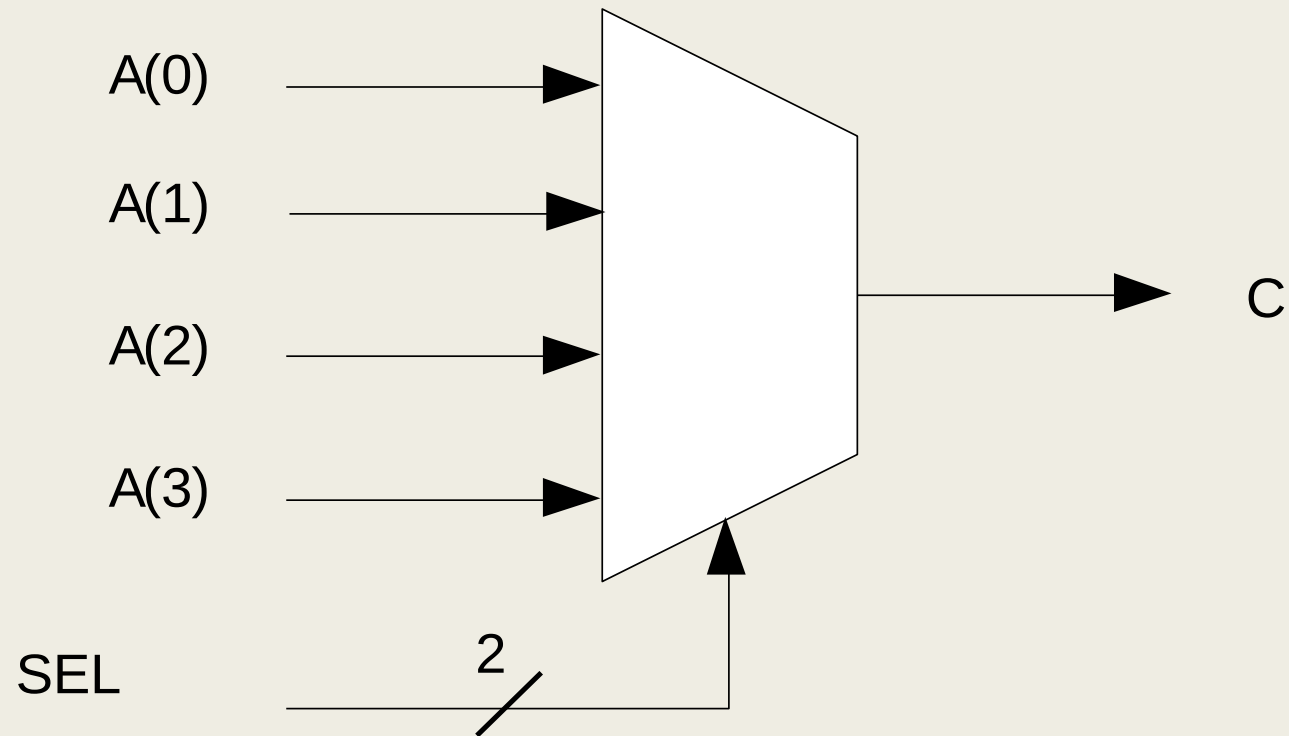
Exercício

- Escrever o comando `process` para o circuito:



Exercício

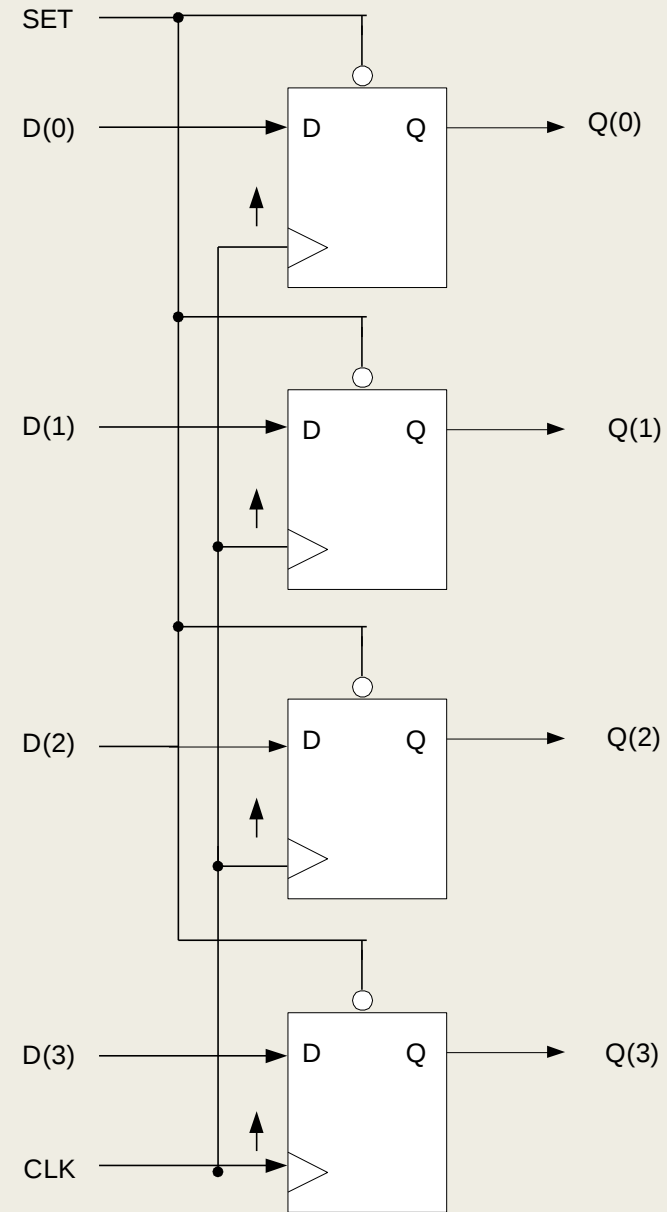
- Escrever o comando process para o circuito multiplexador



Exercício

- Escrever um programa para o circuito da Figura ao lado, cujo funcionamento dos flip-flops é descrito pela Tabela abaixo.

SET	CLK	D	Q
L	X	X	H
H	!	L	L
H	!	H	H



Exemplo 1 de programa VHDL completo

(Circuito lógico combinatório: $EOUT = (AIN \text{ xor } BIN \text{ xor } CIN) \cdot DIN'$)

```
-- Context Clauses
-- Library Clause
library ieee;
-- Use Clause
use ieee.std_logic_1164.all;
-- Entity Declaration
entity FORM1 is
-- lista de entradas e saidas
port (
    AIN : in std_logic;
    BIN : in std_logic;
    CIN : in std_logic;
    DIN : in std_logic;
    EOUT : out std_logic
);
end;
```

continuação ($EOUT = (AIN \text{ xor } BIN \text{ xor } CIN). \text{DIN}'$)

-- Architecture Body

architecture RTL of FORM1 is

-- Declaracao de sinais intermediarios

signal abc: std_logic;

signal Eout_N : std_logic;

begin

-- Processo da saida

process (Eout_N) begin

EOUT <= Eout_N;

end process;

-- Processos intermediarios

process (AIN, BIN, CIN) begin

abc <= AIN xor BIN xor CIN;

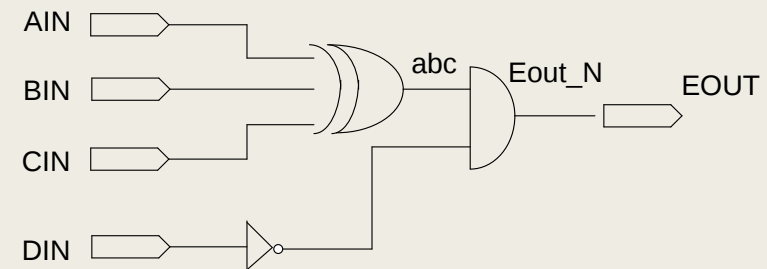
end process;

process (abc, DIN) begin

Eout_N <= abc and (not DIN);

end process;

end;



Exemplo 2 de programa completo

(CIRCUITO SEQUENCIAL: contador binário crescente de 4 bits.)

-- Context Clauses

-- Library Clause

library ieee;

-- Use Clause

use ieee.std_logic_1164.all;

use ieee.std_logic_unsigned.all;

-- Entity Declaration

entity FORM2 is

port (

 LOAD : in std_logic;

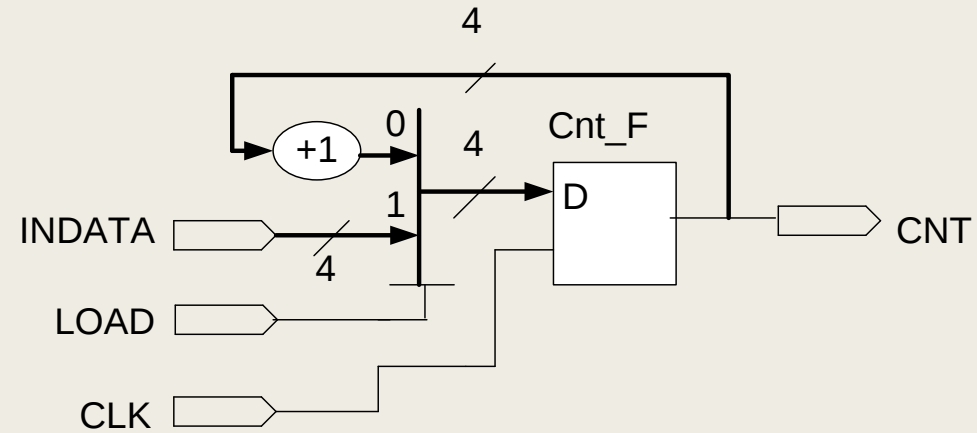
 INDATA : in std_logic_vector (3 downto 0);

 CLK : in std_logic;

 CNT : out std_logic_vector (3 downto 0)

);

end;



(Continuação: contador binário crescente de 4 bits.)

-- Architecture Body

architecture RTL of FORM2 is

signal Cnt_F : std_logic_vector (3 downto 0);

Begin

-- Processo da saída

process (Cnt_F) begin

CNT <= Cnt_F;

end process;

-- Processo intermediário

process (CLK) begin

if (CLK'event and CLK = '1') then

if (LOAD = '1') then

Cnt_F <= INDATA;

else

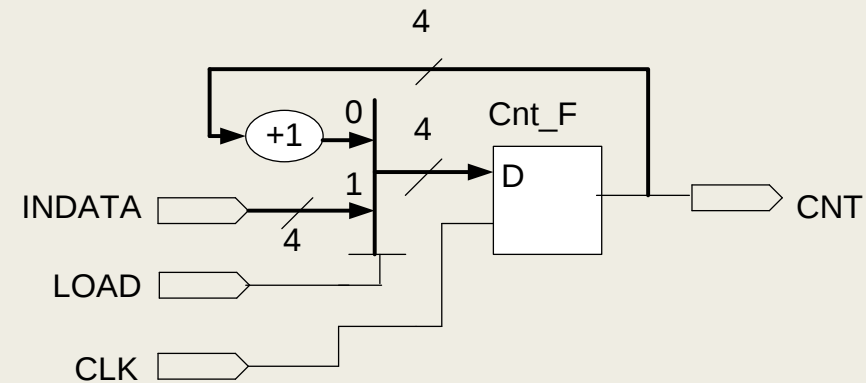
Cnt_F <= Cnt_F + 1;

end if;

end if;

end process;

end;



Exemplo 3 de programa completo

(CIRCUITO SEQUENCIAL: registrador de deslocamento simples)

-- Context Clauses

-- Library Clause

library ieee;

-- Use Clause

use ieee.std_logic_1164.all;

-- Entity Declaration

entity FORM3 is

port (

RST : in std_logic;

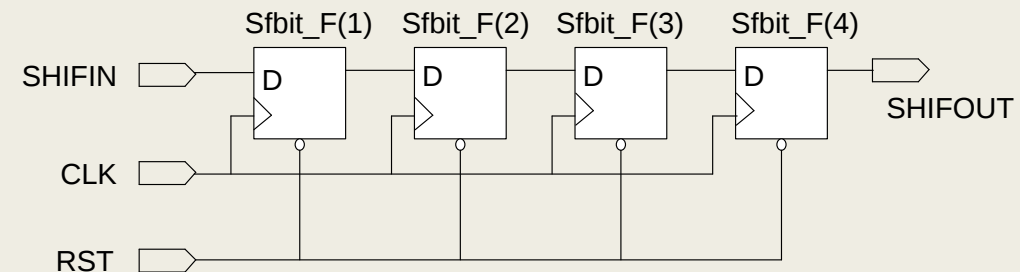
CLK : in std_logic;

SHIFIN : in std_logic;

SHIFOUT : out std_logic

);

end;



(Continuação: registrador de deslocamento simples)

– Architecture Body

architecture RTL of FORM3 is

```
    signal Sfbit_F : std_logic_vector (1 to 4);
```

```
begin
```

```
    process (Sfbit_F) begin
```

```
        SHIFOUT <= Sfbit_F (4) ;
```

```
    end process;
```

```
    process (RST, CLK) begin
```

```
        if (RST = '0' ) then
```

```
            Sfbit_F <= "0000" ;
```

```
        elsif (CLK'event and CLK = '1' ) then
```

```
            Sfbit_F(1) <= SHIFIN;
```

```
            Sfbit_F(2) <= Sfbit_F(1) ;
```

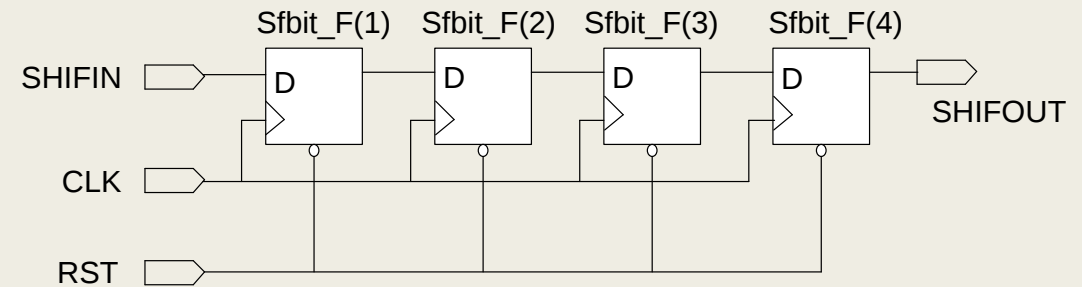
```
            Sfbit_F(3) <= Sfbit_F(2) ;
```

```
            Sfbit_F(4) <= Sfbit_F(3) ;
```

```
        end if;
```

```
    end process;
```

```
end;
```



Exemplo 4 de programa completo

(CIRCUITO COMBINATÓRIO: comparador de 4 bits)

-- Context Clauses

-- Library Clause

library ieee;

-- Use Clause

use ieee.std_logic_1164.all;

use ieee.std_logic_unsigned.all;

-- Entity Declaration

entity FORM4 is

port (

INA : in std_logic_vector (3 downto 0);

INB : in std_logic_vector (3 downto 0);

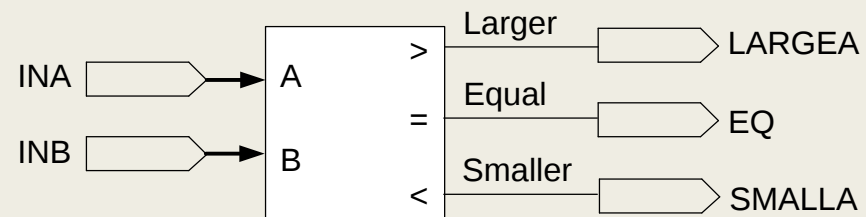
LARGEA : out std_logic;

EQ : out std_logic;

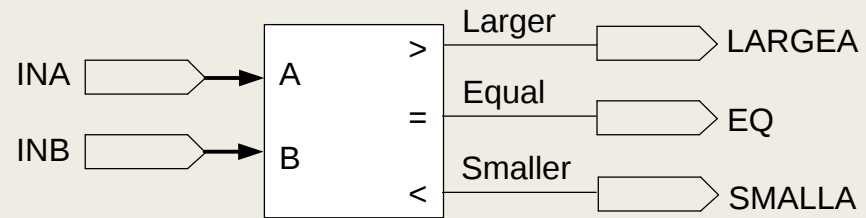
SMALLA : out std_logic

);

end;



Comparador de 4 bits



-- Architecture Body

architecture RTL of FORM4 is

signal Larger : std_logic;

signal Equal : std_logic;

signal Smaller : std_logic;

begin

process (Larger, Equal, Smaller) begin

LARGEA <= Larger;

EQ <= Equal;

SMALLA <= Smaller;

end process;

process (INA, INB) begin

if (INA > INB) then

Larger <= '1' ;

Equal <= '0' ;

Smaller <= '0' ;

elsif (INA = INB) then

Larger <= '0' ;

Equal <= '1' ;

Smaller <= '0' ;

else

Larger <= '0' ;

Equal <= '0' ;

Smaller <= '1' ;

end if;

end process;

end;

Exemplo 5 de programa completo

CIRCUITO COMBINATÓRIO: Decodificador binário de 2 bits.

-- Context Clauses

-- Library Clause

library ieee;

-- Use Clause

use ieee.std_logic_1164.all;

-- Entity Declaration

entity FORM5 is

port (

A : in std_logic_vector (1 downto 0);

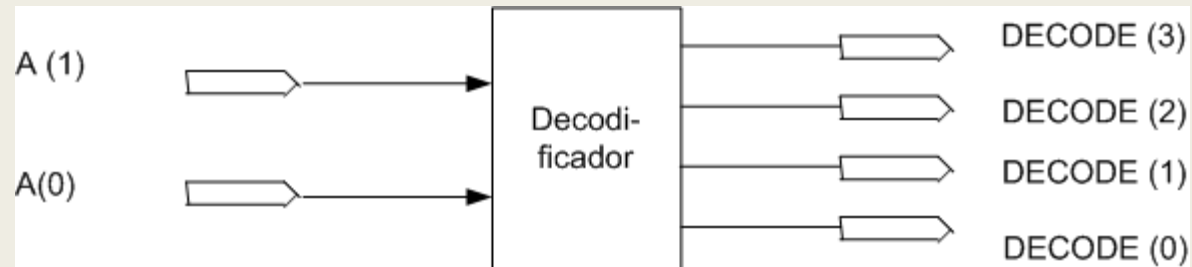
DECODE : out std_logic_vector (3 downto

0)

);

end;

A	DECODE
00	0001
01	0010
10	0100
11	1000



CIRCUITO COMBINATÓRIO: Decodificador binário de 2 bits.

-- Architecture Body

architecture RTL of FORM5 is

signal Deco : std_logic_vector (3 downto 0);

begin

process (Deco) begin

DECODE <= Deco;

end process;

process (A) begin

case A is

when "00" => Deco <= "0001" ;

when "01" => Deco <= "0010" ;

when "10" => Deco <= "0100" ;

when "11" => Deco <= "1000" ;

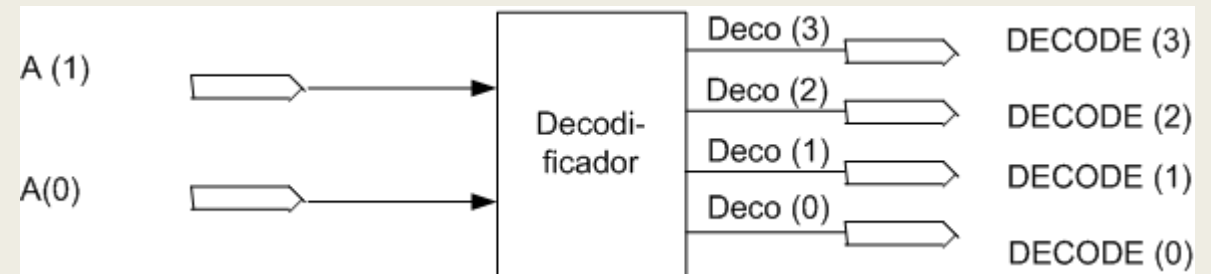
when others => Deco <= "XXXX" ;

end case;

end process;

end;

A	DECODE
00	0001
01	0010
10	0100
11	1000



Exemplo 6 de programa completo

CIRCUITO TRI-STATE

- Context Clauses
- Library Clause

library ieee;

- Use Clause

use ieee.std_logic_1164.all;

- Entity Declaration

entity FORM6 is

port (

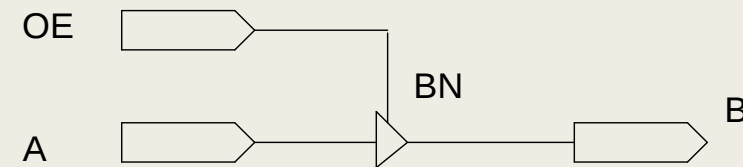
A : in std_logic;

OE : in std_logic;

B : out std_logic

);

end;



CIRCUITO TRI-STATE

-- Architecture Body

architecture RTL of FORM6 is

signal B_N: std_logic;

begin

process (B_N) begin

B <= B_N;

end process;

process (A, OE) begin

if (OE = '1') then

B_N <= A;

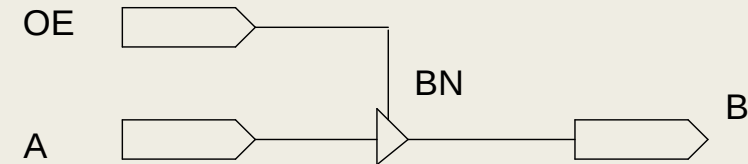
else

B_N <= 'Z' ;

end if;

end process;

end;



Exemplo 7 de programa completo

CIRCUITO TRI-STATE com saída bidirecional

– Context Clauses

– Library Clause

library ieee;

– Use Clause

use ieee.std_logic_1164.all;

– Entity Declaration

entity FORM7 is

port (

A : in std_logic;

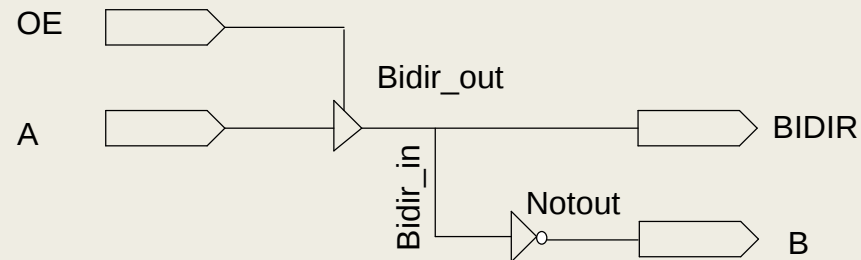
OE: in std_logic;

B : out std_logic;

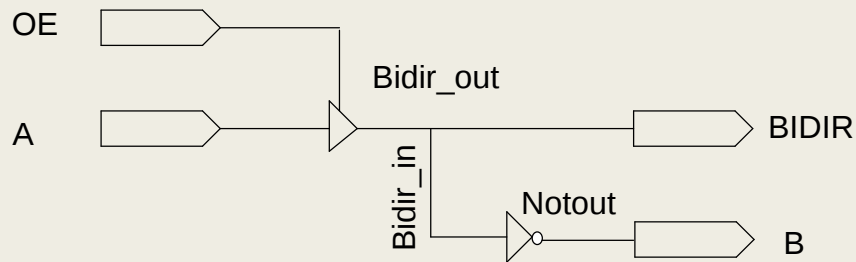
BIDIR : inout std_logic

);

end;



CIRCUITO TRI-STATE com saída bidirecional



```

-- Architecture Body
architecture RTL of FORM7 is
  signal Bidir_in : std_logic;
  signal Bidir_out : std_logic;
  signal Notout : std_logic;
begin
  process (Notout) begin
    B <= Notout;
  end process;
  process (Bidir_out) begin
    BIDIR <= Bidir_out;
  end process;
  process (BIDIR) begin
    Bidir_in <= BIDIR;
  end process;

  process (A, OE) begin
    if (OE = '1' ) then
      Bidir_out <= A;
    else
      Bidir_out <= 'Z' ;
    end if;
  end process;
  process (Bidir_in) begin
    Notout <= not Bidir_in;
  end process;
end;

```

Exemplo 8 de programa completo

(máquina de estado)

-- Context Clauses

-- Library Clause

library ieee;

-- Use Clause

use ieee.std_logic_1164.all;

use ieee.std_logic_unsigned.all;

-- Entity Declaration

entity FORM8 is

port (

CLK : in std_logic;

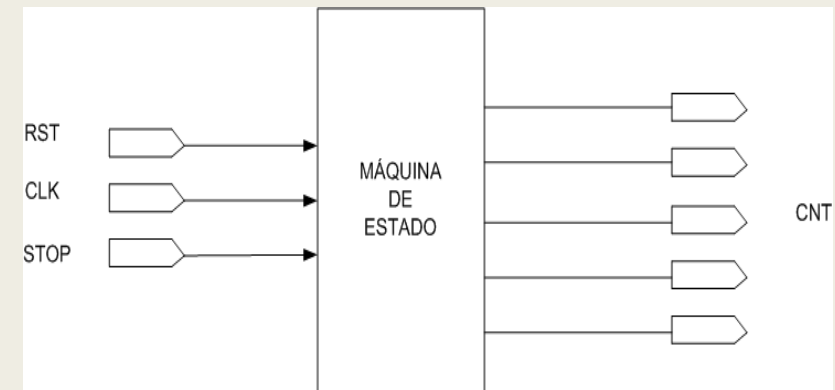
RST : in std_logic;

STOP : in std_logic;

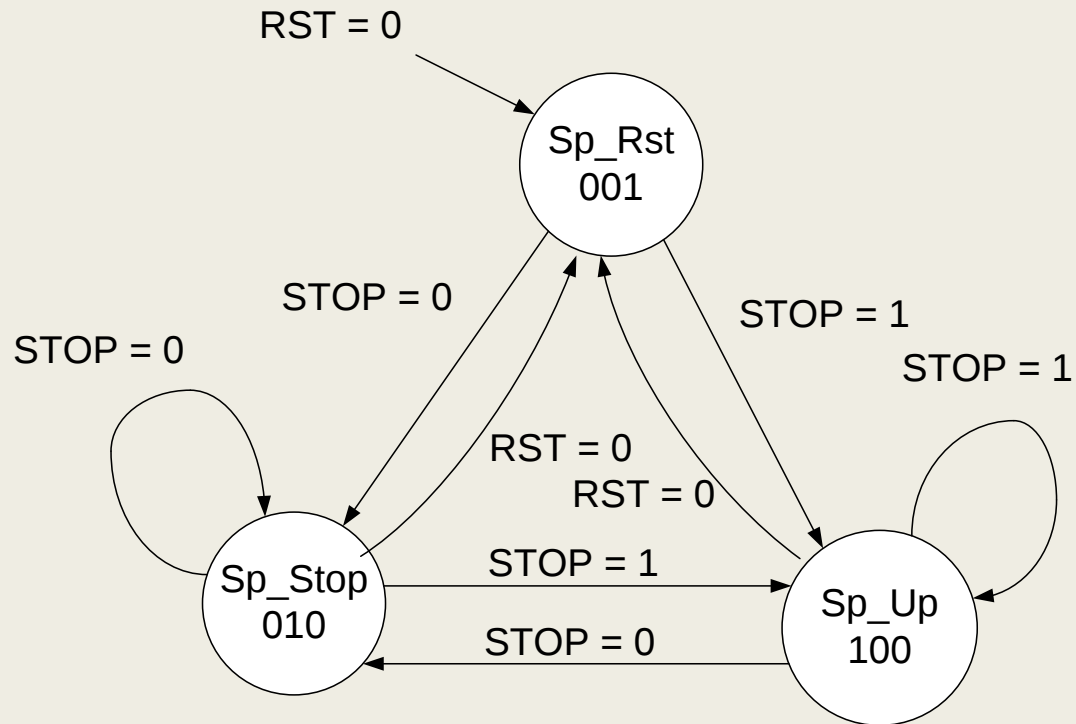
CNT : out std_logic_vector (5 downto 0)

);

end;



Máquina de estado - diagrama



Estados: Sp_Rst = 001 - restart
Sp_Stop = 010 - parada
Sp_Up = 100 - contagem

CNT = contador módulo 60

Entradas:

RST - 0 - restart

STOP - 0 - para

1 - conta

CLK - pulso

Máquina de estado (cont.)

-- Architecture Body

architecture RTL of FORM8 is

```
constant Sp_Rst : std_logic_vector (2 downto 0) := "001" ;  
constant Sp_Stop : std_logic_vector (2 downto 0) := "010" ;  
constant Sp_Up   : std_logic_vector (2 downto 0) := "100" ;  
signal Sp_State_C : std_logic_vector (2 downto 0);  
signal Sp_State_N : std_logic_vector (2 downto 0);  
signal Cnt_F      : std_logic_vector (5 downto 0);
```

begin

```
process (Cnt_F) begin
```

```
    CNT <= Cnt_F;
```

```
end process;
```

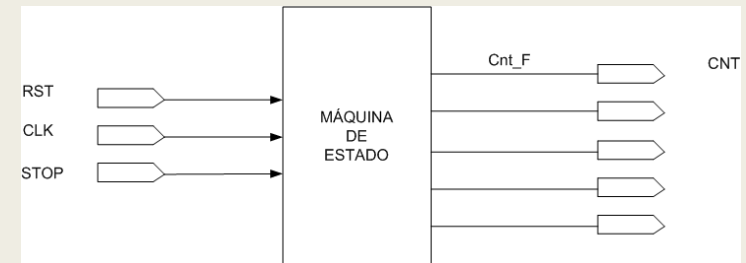
```
process (CLK) begin
```

```
    if (CLK'event and CLK = '1' ) then
```

```
        Sp_State_C <= Sp_State_N;
```

```
    end if;
```

```
end process;
```



Máquina de estado (cont.)

```
process (Sp_State_C, RST, STOP) begin
    if (RST = '0' ) then
        Sp_State_N <= Sp_Rst;
    else
        case Sp_State_C is
            when Sp_Rst =>
                if (STOP = '0' ) then Sp_State_N <= Sp_Stop;
                else Sp_State_N <= Sp_Up;
                end if;
            when Sp_Up =>
                if (STOP = '0' ) then Sp_State_N <= Sp_Stop;
                else Sp_State_N <= Sp_State_C;
                end if;
            when Sp_Stop =>
                if (STOP = '1' ) then Sp_State_N <= Sp_Up;
                else Sp_State_N <= Sp_State_C;
                end if;
            when others =>
                Sp_State_N <= Sp_Rst;
        end case;
    end if;
end process;
```

Máquina de estado (cont.)

```
process (CLK, Sp_State_C, Cnt_F) begin
    if (CLK'event and CLK = '1' ) then
        case Sp_State_C is
            when Sp_Rst =>
                Cnt_F <= "000000" ;
            when Sp_Up =>
                if (Cnt_F >= 59) then
                    Cnt_F <= "000000" ;
                else
                    Cnt_F <= Cnt_F + 1;
                end if;
            when Sp_Stop =>
                Cnt_F <= Cnt_F;
            when others =>
                Cnt_F <= "000000" ;
        end case;
    end if;
end process;

end;
```