Curso Superior de Sistemas de Telecomunicações – Unidade São José

Disciplina: Síntese de Sistemas de Telecomunicações – 7º Fase

Bases tecnológicas

- Dispositivos Lógicos Programáveis.
- Introdução à Tecnologia FPGA.
- Introdução aos ambientes de software EDA (Electronic Design Automation).
- Introdução à Linguagem VHDL.
- Aritmética computacional.
- Introdução aos Kits de desenvolvimento.
- Síntese de circuitos baseada em dispositivos lógicos programáveis.

Bibliografia

- *Material de apoio fornecido pela Xilinx (Xilinx University Program – XUP).* www.xilinx.com
- *Digital Signal Processing with Field Programmable Gate Arrays; 2.ed;* ***Uwe Meyer-Baese****; Springer, 2006*
- *Digital Electronics and Design with VHDL;* ***Volnei A. Pedroni****; Elsevier Science, 2007*
- *Circuit Design with VHDL;* ***Volnei A. Pedroni****; MIT Press, 2004*
- *Projetando Controladores Digitais com FPGA;* ***César da Costa****; Novatec, 2006*

Introdução a FPGA

O material a ser apresentado foi elaborado com base nas bibliografias citadas anteriormente.

Aplicações de DSP (Digital Signal Processing)

- ***Processadores DSP****:* Nos últimos 20 anos, a maior parte das aplicações DSP foram realizadas por processadores DSP (Texas Instruments, Motorola, Analog Devices...).

- ***ASICs*** (Application Specific Integrated Circuits): São muito utilizados em aplicações específicas de DSP.

- ***FPGA*** (Field Programmable Gate Array): Tecnologia recente para aplicações de DSP de alta velocidade **(**Xilinx, ALTERA, Atmel...).

Aplicações de DSP

- A maioria dos algoritmos utilizados nas aplicações de DSP envolvem as operações de multiplicação e soma, conhecidas como operações MAC (Multiplies and accumulates).

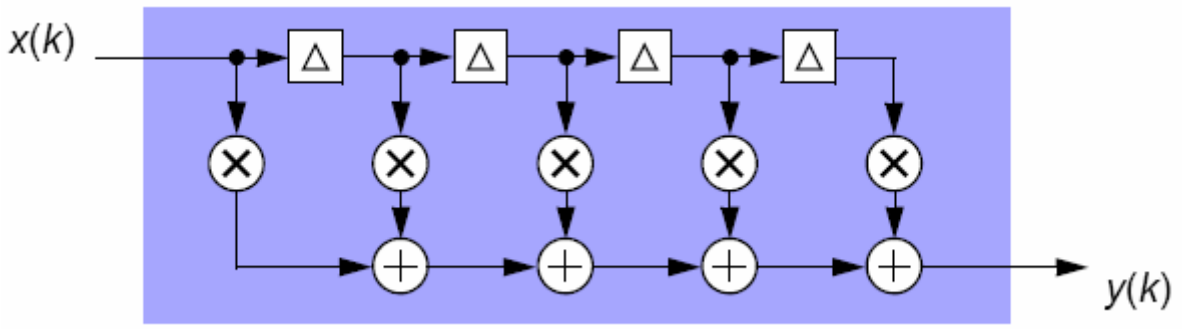
- Operações de divisão e raiz quadrada são raras em algoritmos de DSP.

- Normalmente a complexidade de um algoritmo de DSP pode ser medida em termos do número de operações MAC utilizadas.

Exemplo – Operações MAC

Filtro FIR com 5 coeficientes

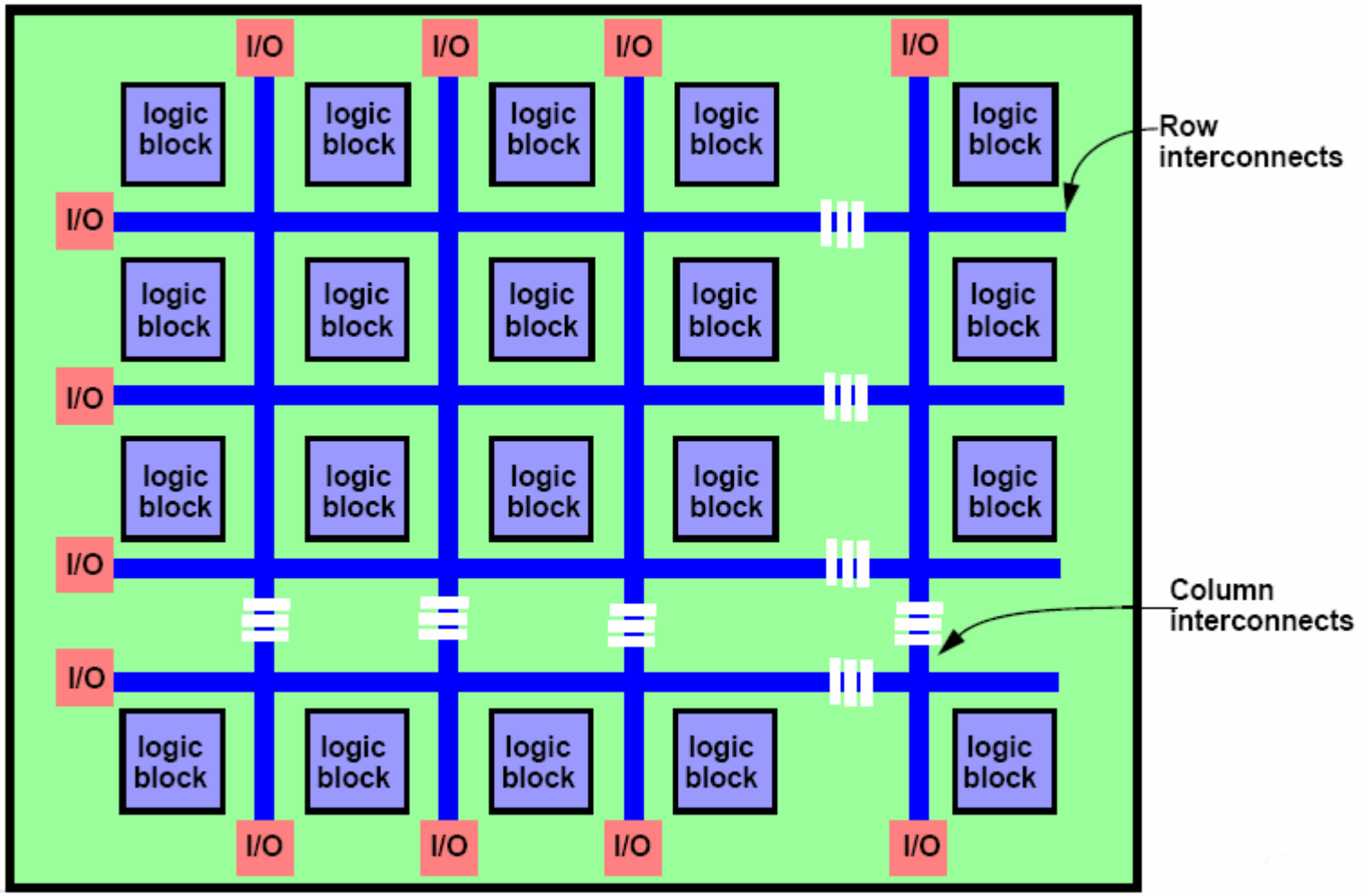
$$y(k) = \sum_{n=0}^{4} w_n x(k-n)$$



FPGA

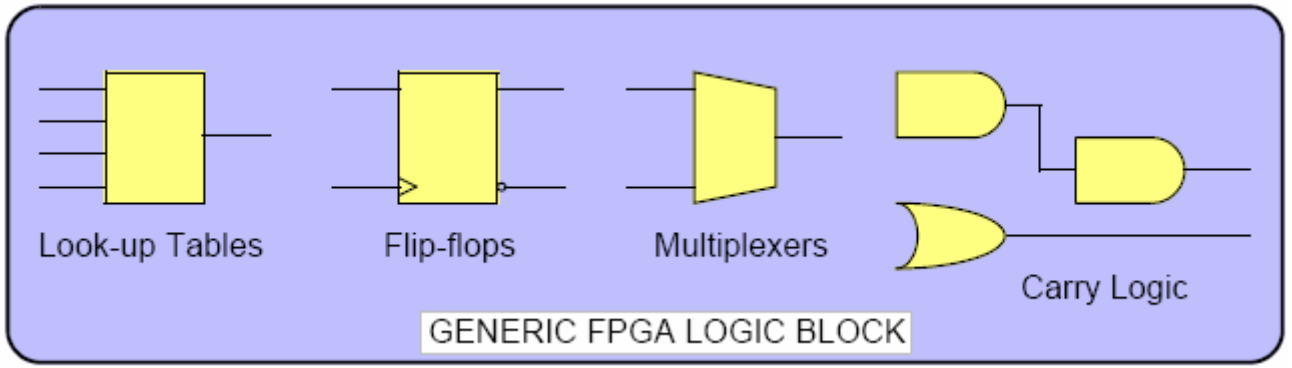
- A FPGA é um conjunto de unidades lógicas idênticas (blocos lógicos) e configuráveis contidas em um único circuito integrado.
- As unidade lógicas são conectadas por uma matriz de trilhas condutoras e por chaves de interconexão.
- As interfaces I/O do FPGA são feitas pelos blocos de I/O do componente.

Estrutura simplificada do FPGA



Bloco Lógico de um FPGA

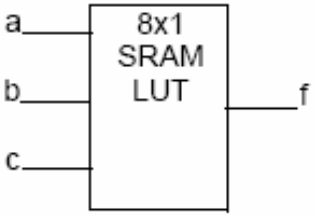
- Os blocos lógicos contém componentes lógicos que implementam funções lógicas e aritméticas. Para a fabricante Xilinx, os blocos lógicos são chamados de *Slices*.



LUT – Look-Up Table

- São blocos muito utilizados para implementar funções lógicas.
- A forma geral de uma LUT é uma SRAM, que armazena tabelas verdade para funções lógicas de n-entradas.
- Desta forma, as linhas de endereçamento da SRAM funcionam como entrada e a saída fornece o valor da função lógica.

a	b	c	f
0	0	1	1
0	0	0	0
0	1	1	1
0	1	0	0
1	0	1	1
1	0	0	0
1	1	1	1
1	1	0	0



Fluxo de Projeto em FPGA

- Especificação e entrada do projeto.
- Síntese e mapeamento da tecnologia.
- Posicionamento e roteamento.
- Verificação e teste.
- Programação do FPGA.

Especificação e Entrada do Projeto

- Diagramas lógico através de editores gráficos.
- Linguagem de descrição de hardware (Hardware Description Language) através de editores de texto.
- Modelagem de sistemas através de software específico (Ex. System Generator da Xilinx).

Síntese e mapeamento da tecnologia (netlist)

- O processo de síntese otimiza as equações boleanas (otimização lógica independente da tecnologia) geradas pelo projeto, permitindo a redução da área a ser ocupada no FPGA. Os atrasos entre os sinais internos também são reduzidos.

- No mapeamento, o projeto otimizado é adequado à tecnologia empregada no componente a ser utilizado.

Posicionamento e roteamento (Place and Route)

- O posicionamento é a atribuição de componentes disponíveis aos componentes lógicos do projeto.

- Na etapa de roteamento, as conexões entre os componentes são garantidas visando sempre maximizar a velocidade das conexões críticas.

Verificação e teste

- Nesta etapa são utilizadas algumas ferramentas para simular o funcionamento do projeto.
  - ModelSim
  - Synopsys

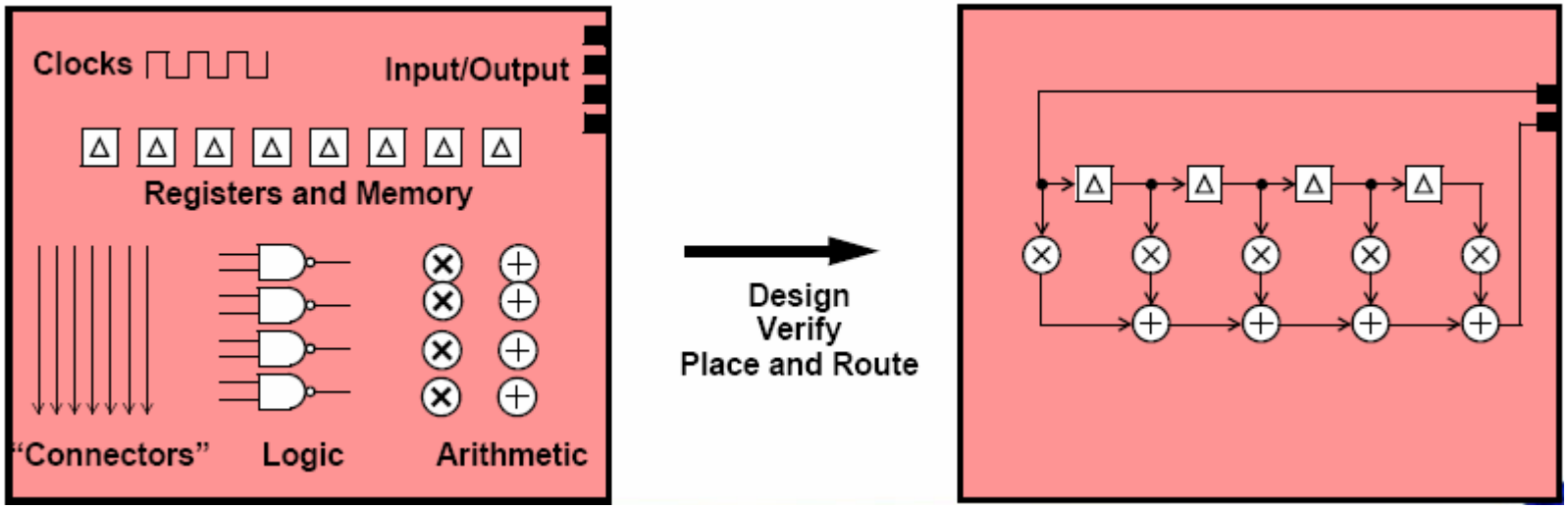
- As simulações podem ser feitas para verificar o comportamento do sistema e também para verificar restrições de temporização.

Programação do FPGA

- Nesta etapa, é gerado um arquivo binário para configuração do dispositivo.

- O download pode ser feito através de um cabo USB.

FPGA: uma caixa de blocos de DSP

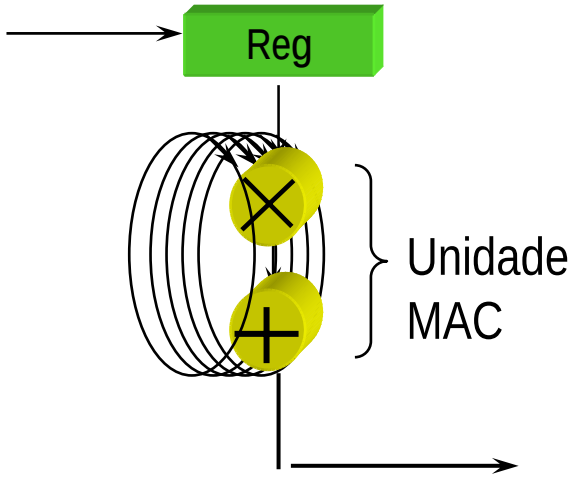


FPGA vs. Processadores DSP

- O FPGA tem a flexibilidade para variar o número de bits de acordo com a aplicação.
- No FPGA o processamento é inerentemente paralelo, para algoritmos seqüenciais um FPGA não é a melhor solução.
- Se uma determinada aplicação pode ser realiza em um processador DSP, provavelmente não é vantagem utilizar um FPGA.

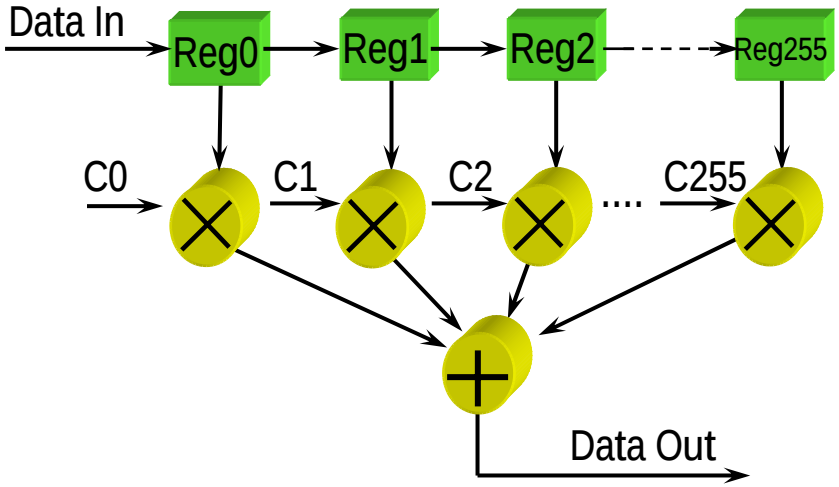
Filtro FIR com 256 coeficientes

DSP Convencional



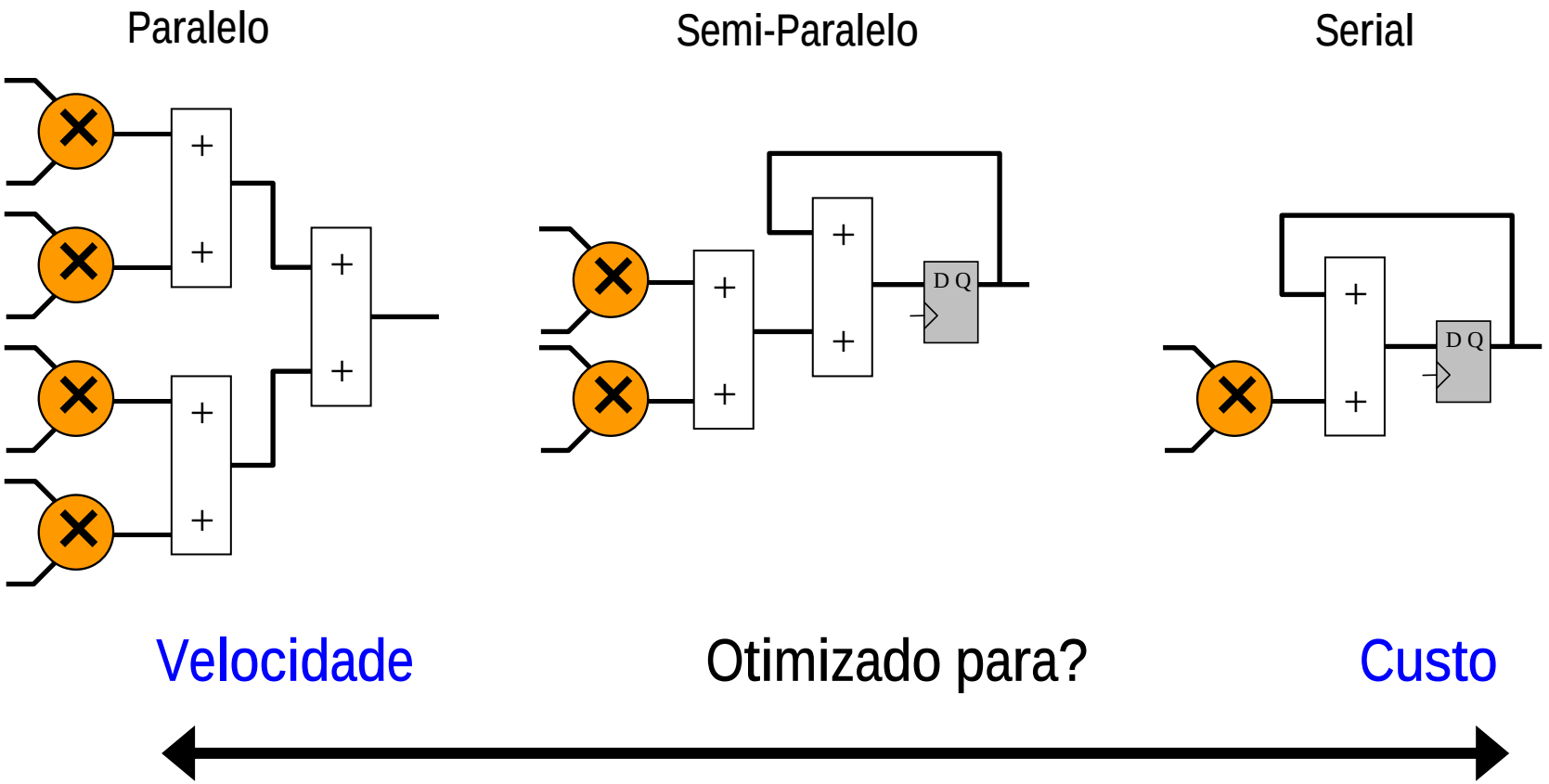
256 Loops necessários para processar amostras

FPGA



Todas as 256 operações MAC são realizadas em 1 ciclo de relógio (clock)

Flexibilidade do FPGA: Custo vs. Velocidade



Ferramentas utilizadas na disciplina

- ISE (Xilinx).
- Quartus (ALTERA).
- Matlab/System Generator (Xilinx).

VHDL

- VHDL (**V**ery Hight Specific Integrated Circuit **H**ardware **D**escription **L**anguage) é uma linguagem de descrição de hardware.

- A linguagem VHDL é um padrão especificado pelo IEEE 1076 e IEEE 1164.

- Por ser um padrão o código desenvolvido independe do fabricante.

VHDL

- As duas aplicações principais são em dispositivos lógicos programáveis (CPLD, FPGA) e em ASICs.
- A linguagem (código) VHDL é inerentemente paralela.
- Apenas comandos colocados dentro de *processos*, *funções ou procedimentos* são executados seqüencialmente.

Unidades básicas do VHDL

- ***Declaração de bibliotecas:*** Lista de todas as bibliotecas que podem ser usadas no projeto.

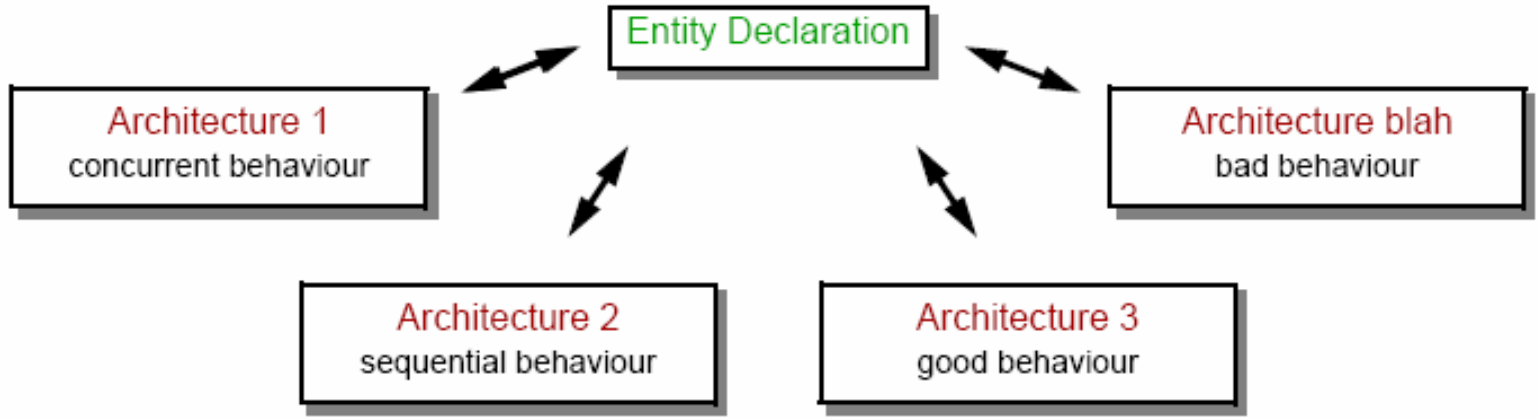
- ***Entidade:*** Especifica os pinos de I/O do circuito.

- ***Arquitetura:*** Contém o código VHDL dizendo como o circuito deve funcionar.

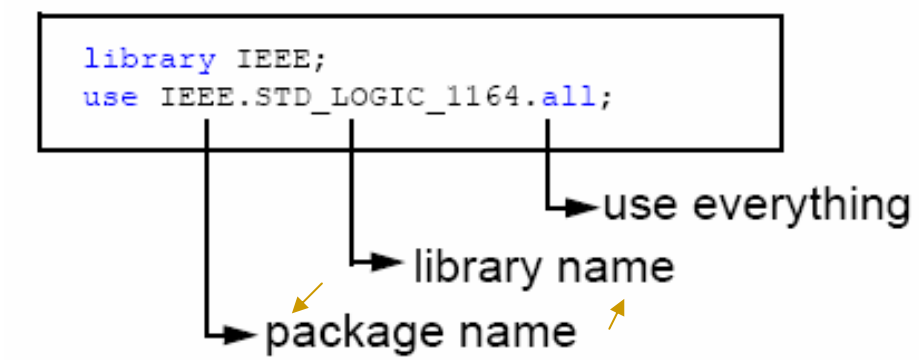
VHDL

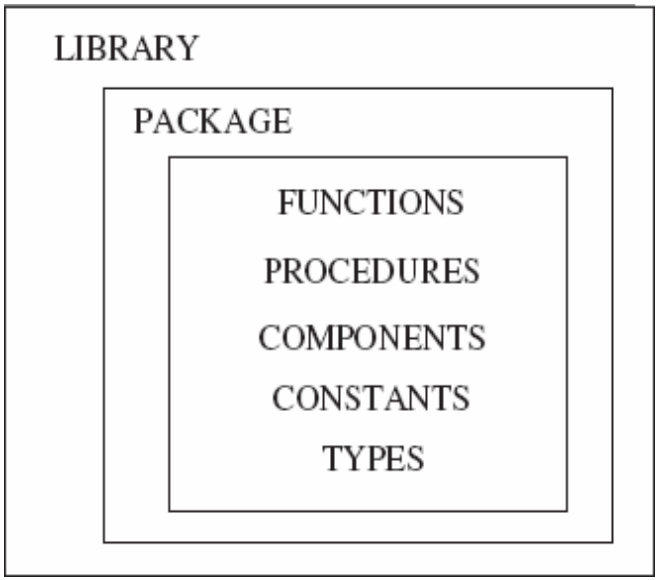
- Todo o projeto em VHDL deve conter pelo menos um par entidade-arquitetura.

- Um projeto pode ter mais de uma arquitetura, cada uma descrevendo um comportamento para o sistema



Declaração de bibliotecas





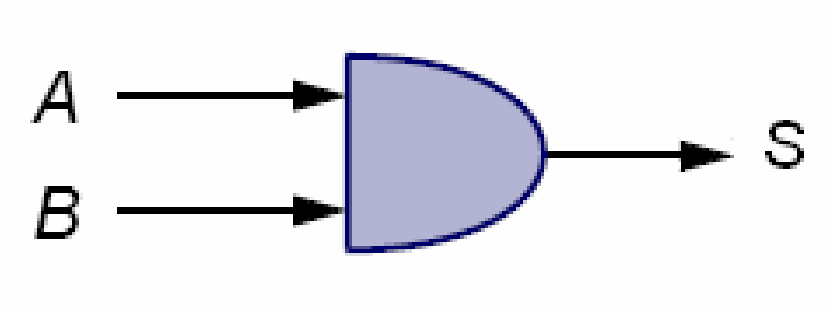
As bibliotecas 'work' e 'std' são incluídas por definição, não necessitando serem citadas no código.

Declaração de Entidade

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

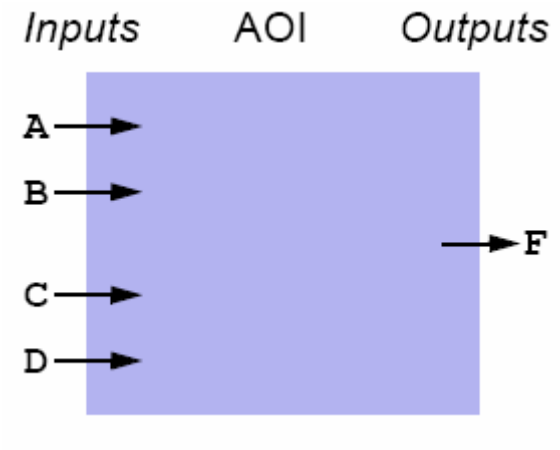
- O modo do sinal pode ser *in* (entrada), *out* (saída), *inout* (bidirecional) *e buffer* (saída e realimentação externa).
- O tipo pode ser *bit*, *std_logic*, *integer*, *etc*.
- O nome da entidade deve ser o mesmo do arquivo *.vhd*.

Exemplo 1




```
ENTITY PORTA_E IS
    PORT
    (
        A, B: IN BIT;
        S: OUT BIT
    );
END ENTITY PORTA_E;
```

Exemplo 2



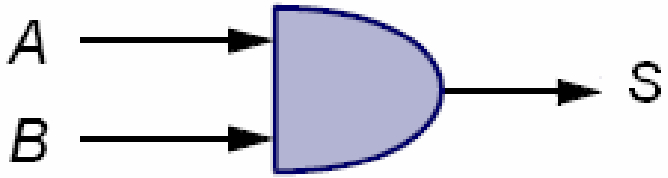

```
entity AOI is

   port(A, B, C, D : in  std_logic;
        F          : out std_logic);

end AOI;
```

Declaração da arquitetura

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

- A declaração de sinais internos e constantes é feita entre a arquitetura e o início do código.
- O nome da entidade por ser qualquer nome, exceto as palavras reservadas do VHDL.

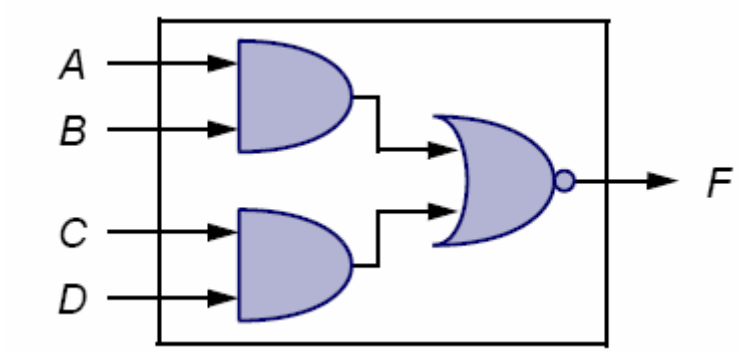
Exemplo 1




```
ARCHITECTURE MINHA_PORTA OF PORTA_E2 IS

BEGIN

    S <= A AND B;

END ARCHITECTURE;
```

Para atribuir valores aos sinais utiliza-se '<=' e para variáveis utiliza-se ':=', neste exemplo o valor de 'A.B' está sendo atribuído ao sinal de saída 'S'.

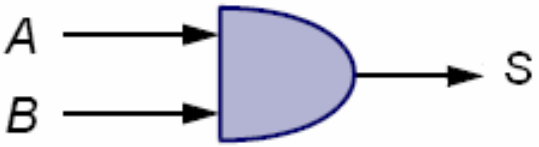
Exemplo 2



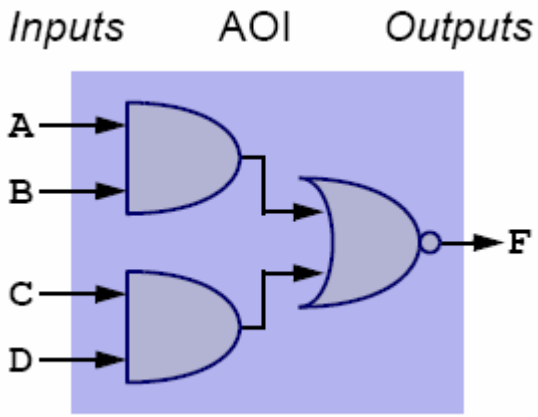

```
architecture V1 of AOI is
begin

   F <= not((A and B) or (C and D));

end V1;
```

Exemplo: Porta E

```
ENTITY PORTA_E IS
    PORT
    (
        A, B: IN BIT;
        S: OUT BIT
    );
END ENTITY PORTA_E;

ARCHITECTURE MINHA_PORTA OF PORTA_E  IS

BEGIN

    S <= A AND B;

END ARCHITECTURE;
```



Exemplo: Porta E_OU




```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

-- entity declaration
entity AOI is

    port(A, B, C, D : in  std_logic;
         F          : out std_logic);

end AOI;

-- architecture body
architecture V1 of AOI is
begin

    F <= not((A and B) or (C and D));

end V1;
```

Palavras Reservadas do VHDL

ABS
ACCESS
AFTER
ALIAS
ALL
AND
ARCHITECTURE
ARRAY
ASSERT
ATTRIBUTE

BEGIN
BLOCK
BODY
BUFFER
BUS

CASE
COMPONENT
CONFIGURATION
CONSTANT

DISCONNECT
DOWNTO

ELSE
ELSIF
END
ENTITY
EXIT

FILE
FOR
FUNCTION

GENERATE
GENERIC
GROUP
GUARDED

IF
IMPURE
IN
INERTIAL
INOUT
IS

LABEL
LIBRARY
LINKAGE
LITERAL
LOOP

MAP
MOD

NAND
NEW
NEXT
NOR
NOT
NULL

OF
ON
OPEN
OR
OTHERS
OUT

PACKAGE
PORT
POSTPONED
PROCEDURE
PROCESS
PURE

RANGE
RECORD
REGISTER
REJECT
REM
REPORT
RETURN
ROL
ROR

SELECT
SEVERITY
SIGNAL
SHARED
SLA
SLL
SRA
SRL
SUBTYPE

THEN
TO
TRANSPORT
TYPE

UNAFFECTED
UNITS
UNTIL
USE

VARIABLE

WAIT
WHEN
WHILE
WITH

XNOR
XOR

Introdução ao Quartus II

- www.altera.com

Código Concorrente (paralelos) e Seqüencial

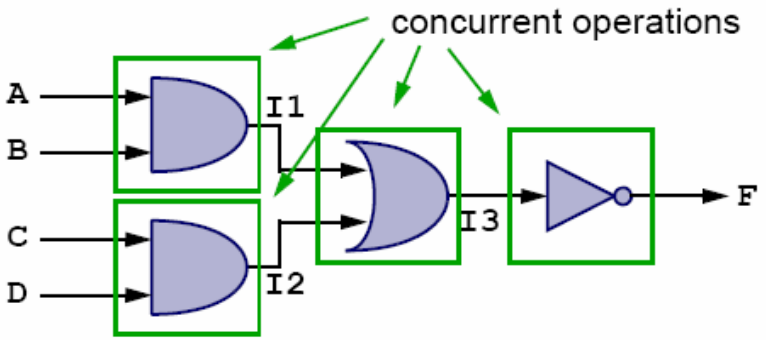
- Os códigos paralelos em geral implementam circuitos lógicos combinacionais, onde a saída do circuito depende apenas da entrada atual

- Os códigos seqüenciais implementam circuitos lógicos seqüenciais, onde a saída do circuito depende de entradas anteriores

Código Concorrente

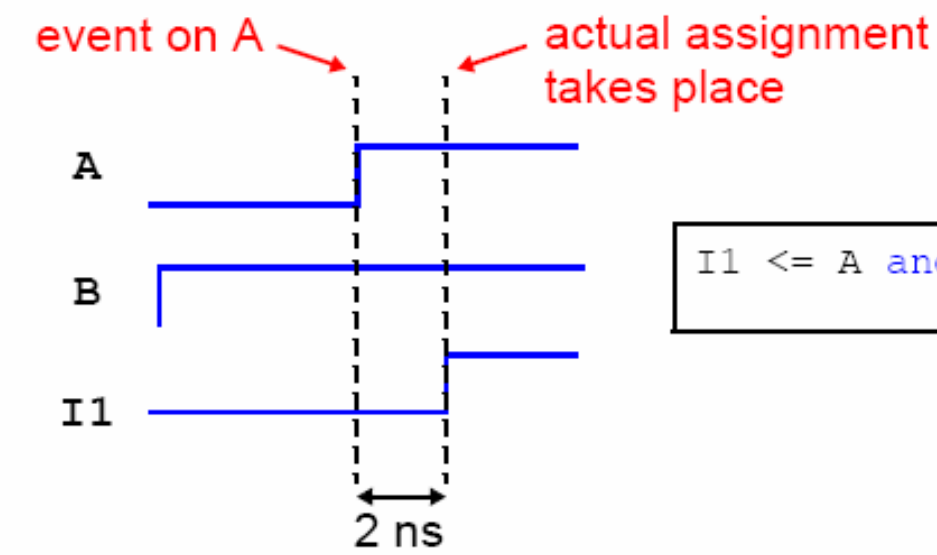
Não importa a ordem em que os sinais são escritos, eles podem ser executados ao mesmo tempo

```
-- entity declaration
entity AOI is
    port(A, B, C, D : in  std_logic;
         F          : out std_logic);
end AOI;

-- architecture body
architecture V2 of AOI is
    signal I1, I2, I3 : std_logic;
begin
    -- concurrent assignments
    I1 <= A and B;
    I2 <= C and D;
    I3 <= I1 or I2;
    F  <= not I3;
end V2;
```



Código Concorrente

- A atribuição de um valor a um sinal concorrente é disparada por um **evento** em um determinado sinal.
- O evento neste caso é a **mudança de valor** de um sinal.
- No caso abaixo, um atraso de 2ns na atribuição do valor para I1 é considerado

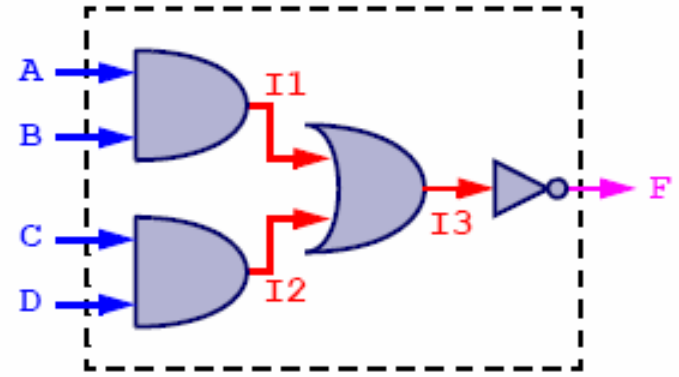


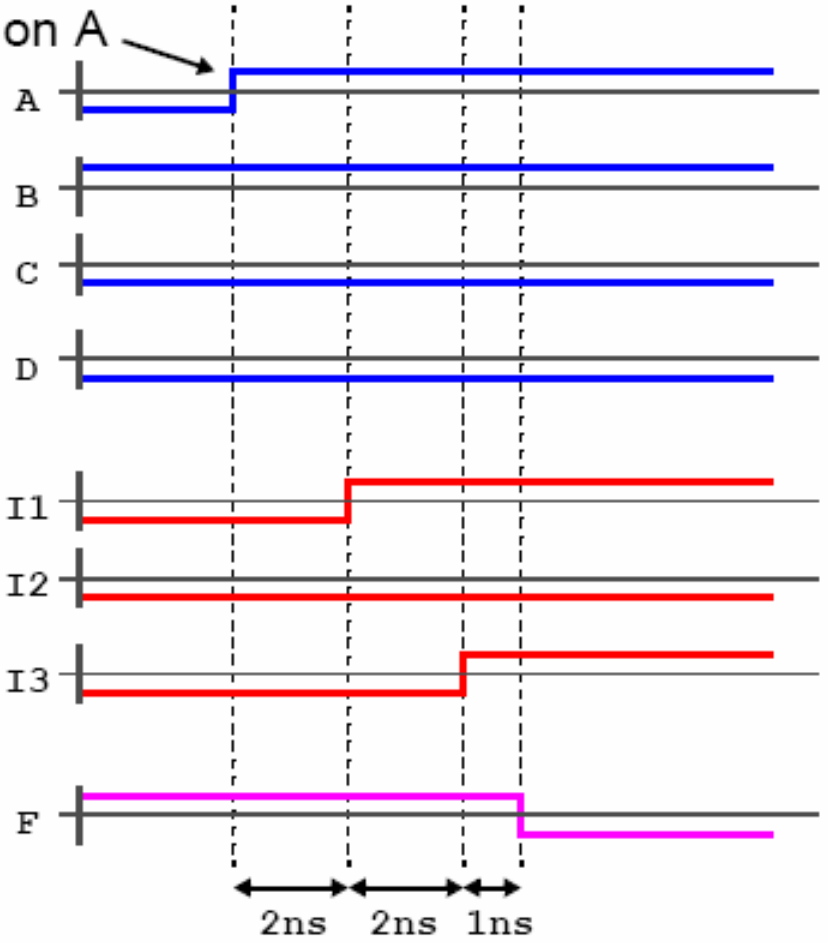

```
I1 <= A and B after 2 ns;
```

Código Concorrente

O diagrama abaixo mostra a resposta de um circuito ao evento A

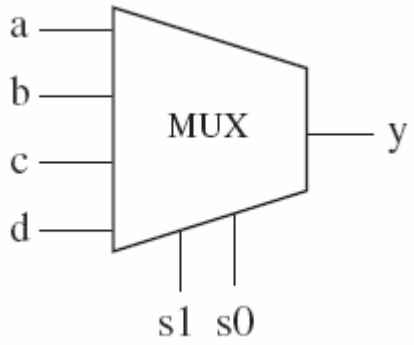
```
-- concurrent assignments
I1 <= A and B after 2 ns;
I2 <= C and D after 2 ns;
I3 <= I1 or I2 after 2 ns;
F  <= not I3 after 1 ns;
```



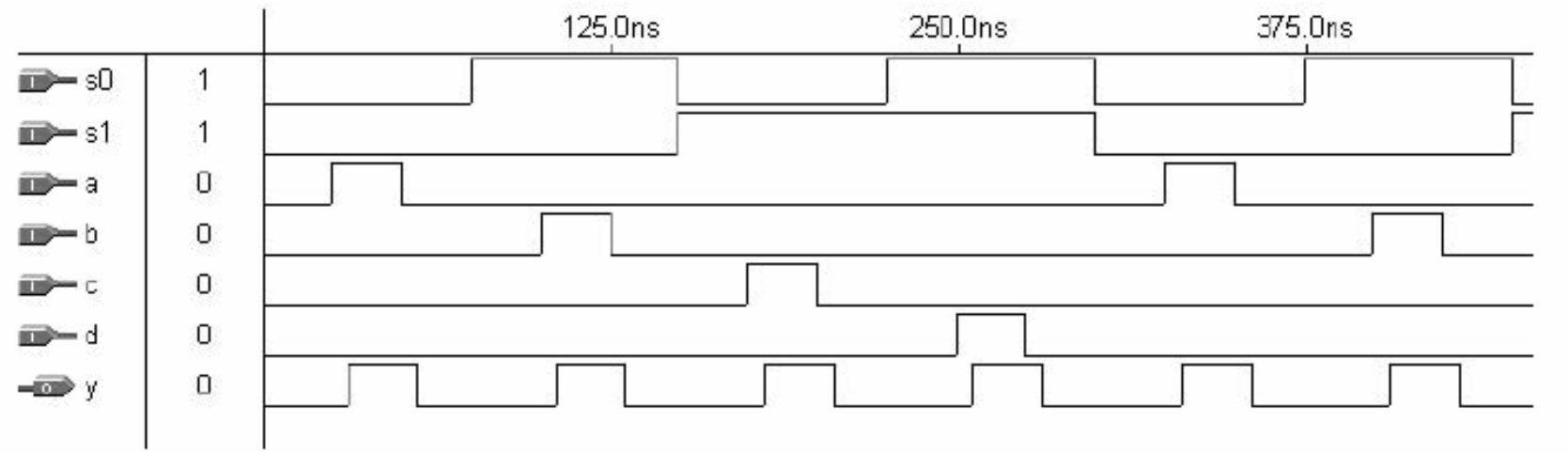


Exemplo 1: Multiplexador

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d, s0, s1: IN STD_LOGIC;
          y: OUT STD_LOGIC);
END mux;
---------------------------------------
ARCHITECTURE pure_logic OF mux IS
BEGIN
   y <=  (a AND NOT s1 AND NOT s0) OR
         (b AND NOT s1 AND s0) OR
         (c AND s1 AND NOT s0) OR
         (d AND s1 AND s0);
END pure_logic;
```

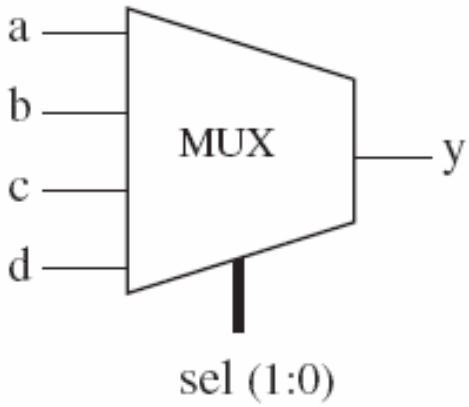


Resultado da simulação



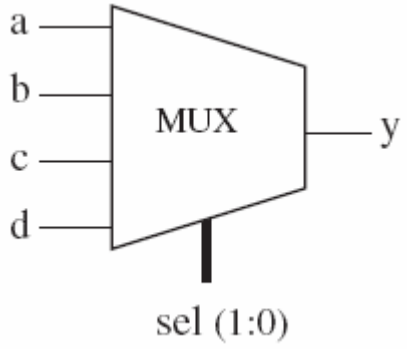
Exemplo 2: Multiplexador (when/else)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
          sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
          y: OUT STD_LOGIC);
END mux;
-----------------------------------------
ARCHITECTURE mux1 OF mux IS
BEGIN
   y <=  a WHEN sel="00" ELSE
         b WHEN sel="01" ELSE
         c WHEN sel="10" ELSE
         d;
END mux1;
```



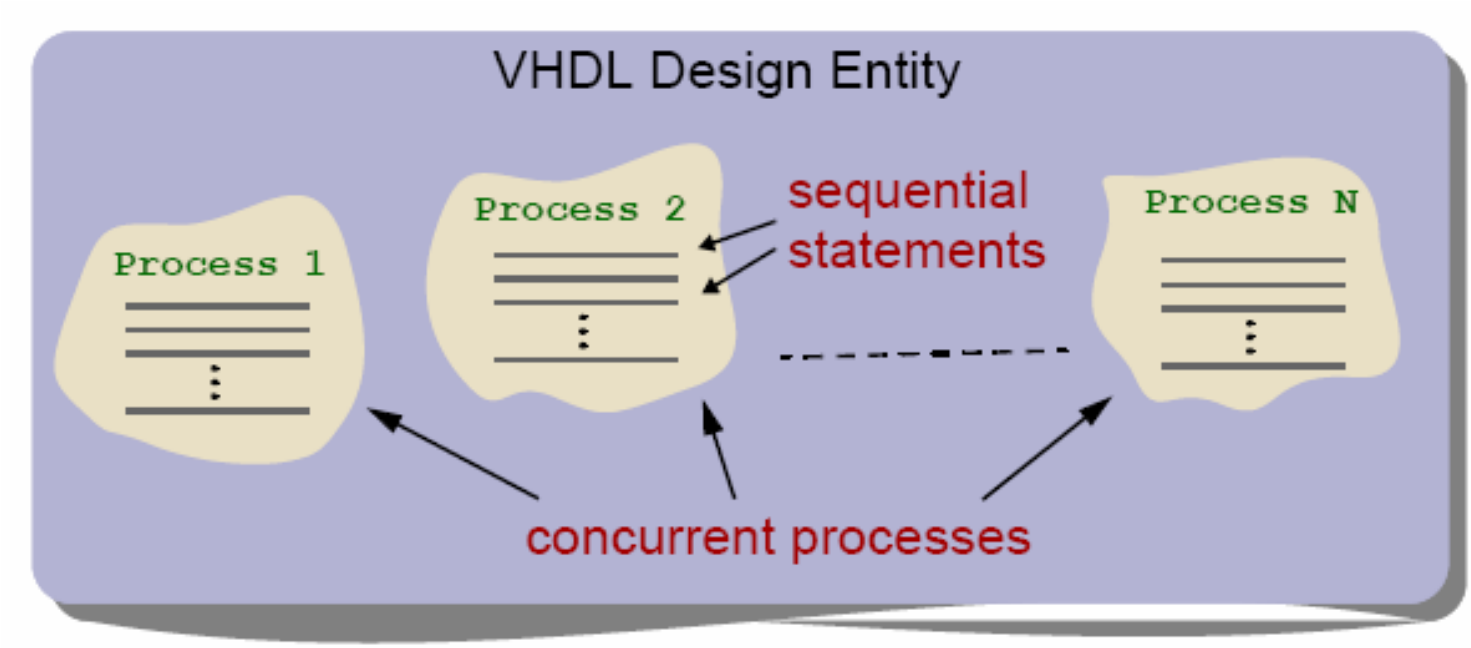
Exemplo 3: Multiplexador (with/select/when)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
          sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
          y: OUT STD_LOGIC);
END mux;
-----------------------------------------
ARCHITECTURE mux2 OF mux IS
BEGIN
   WITH sel SELECT
      y <=  a WHEN "00",      -- notice "," instead of ";"
            b WHEN "01",
            c WHEN "10",
            d WHEN OTHERS;    -- cannot be "d WHEN "11" "
END mux2;
```



Código Seqüencial - Processos

- Um processo é uma seção seqüencial de um código VHDL.
- As instruções dentro de um processo são executadas em ordem.
- Processos dentro de uma arquitetura são executados em paralelo.

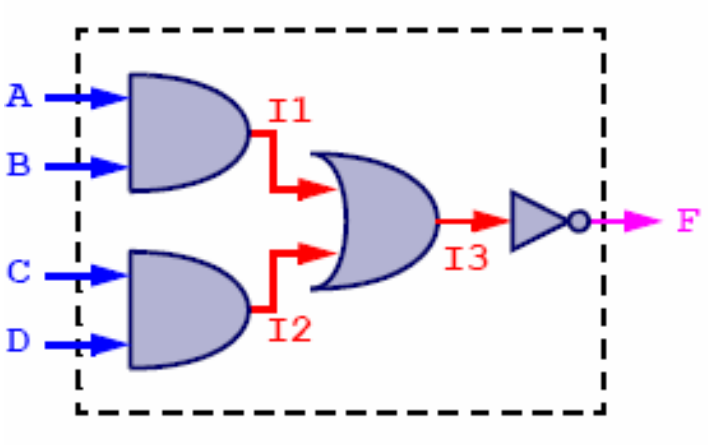


Processos com Lista de Sensibilidade

- A lista de sensibilidade é opcional.
- Processos com lista de sensibilidade apenas são executados quando um evento ocorre.
- O evento é uma mudança do sinal contido na lista de sensibilidade.

```
process (A, B)      -- process with a sensitivity list
begin

    F <= A and B;   -- these statements are executed one ...
    G <= A or B;    -- ... after the other

end process;
```

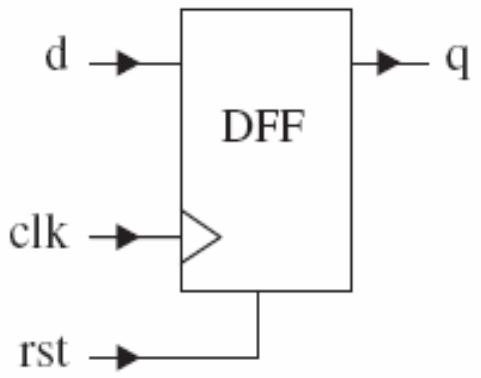
Vários Processos em uma Arquitetura



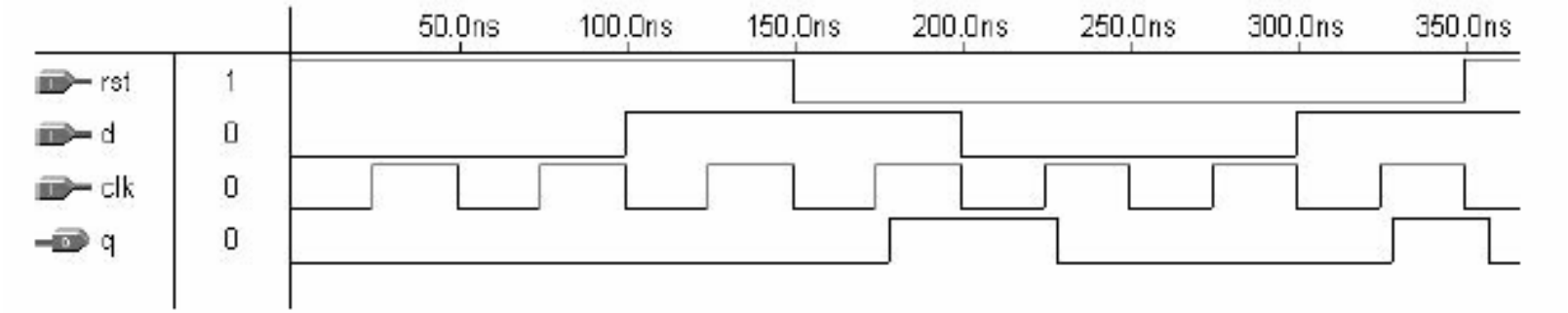

```
-- architecture body
architecture V3 of AOI is
    signal I1, I2, I3 : std_logic;
begin

    -- process to model the AND gates
    AND_GATES : process (A, B, C, D)
    begin
        I1 <= A and B after 2 ns;
        I2 <= C and D after 2 ns;
    end process;

    -- process to model the OR gate
    OR_GATE : process (I1, I2)
    begin
        I3 <= I1 or I2 after 2 ns;
    end process;

    -- process to model the inverter
    NOT_GATE : process (I3)
    begin
        F  <= not I3 after 1 ns;
    end process;

end V3;
```

Exemplo 1: Flip-flop tipo D com reset assíncrono.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY dff IS
   PORT ( d, clk, rst: IN STD_LOGIC;
          q: OUT STD_LOGIC);
END dff;
---------------------------------------
ARCHITECTURE behavior OF dff IS
BEGIN
   PROCESS (rst, clk)
   BEGIN
      IF (rst='1') THEN
         q <= '0';
      ELSIF (clk'EVENT AND clk='1') THEN
         q <= d;
      END IF;
   END PROCESS;
END behavior;
```

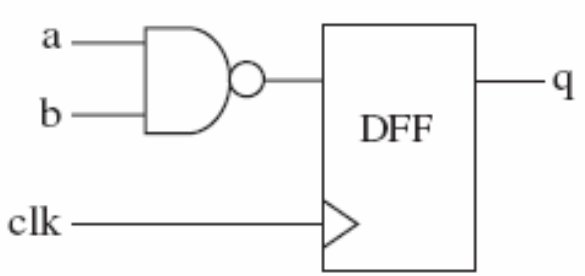


Resultado da simulação

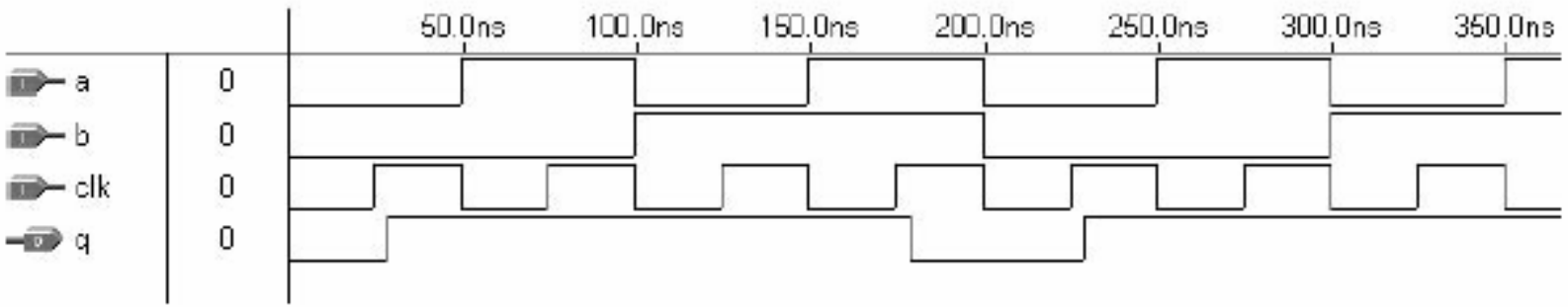


Exemplo 2: Flip-flop D com porta nand

```
ENTITY example IS
   PORT ( a, b, clk: IN BIT;
          q: OUT BIT);
END example;
---------------------------------------
ARCHITECTURE example OF example IS
   SIGNAL temp : BIT;
BEGIN
   temp <= a NAND b;
   PROCESS (clk)
   BEGIN
      IF (clk'EVENT AND clk='1') THEN q<=temp;
      END IF;
   END PROCESS;
END example;
```



Resultado da simulação



Sinais, Variáveis e Constantes em VHDL

- Sinais: São utilizados para representar fios e barramentos em um circuito
- Variáveis: Similares às variáveis utilizadas em programas de computadores
- Constantes: Valores que não se alteram e que são rotulados para facilitar a programação.

Sinais, Variáveis e Constantes em VHDL

- No caso da variável e do sinal, o valor inicial é opcional.

```
constant NUMBER_OF_BITS: integer := 8;
```

```
variable current_bit: integer := 0;
```

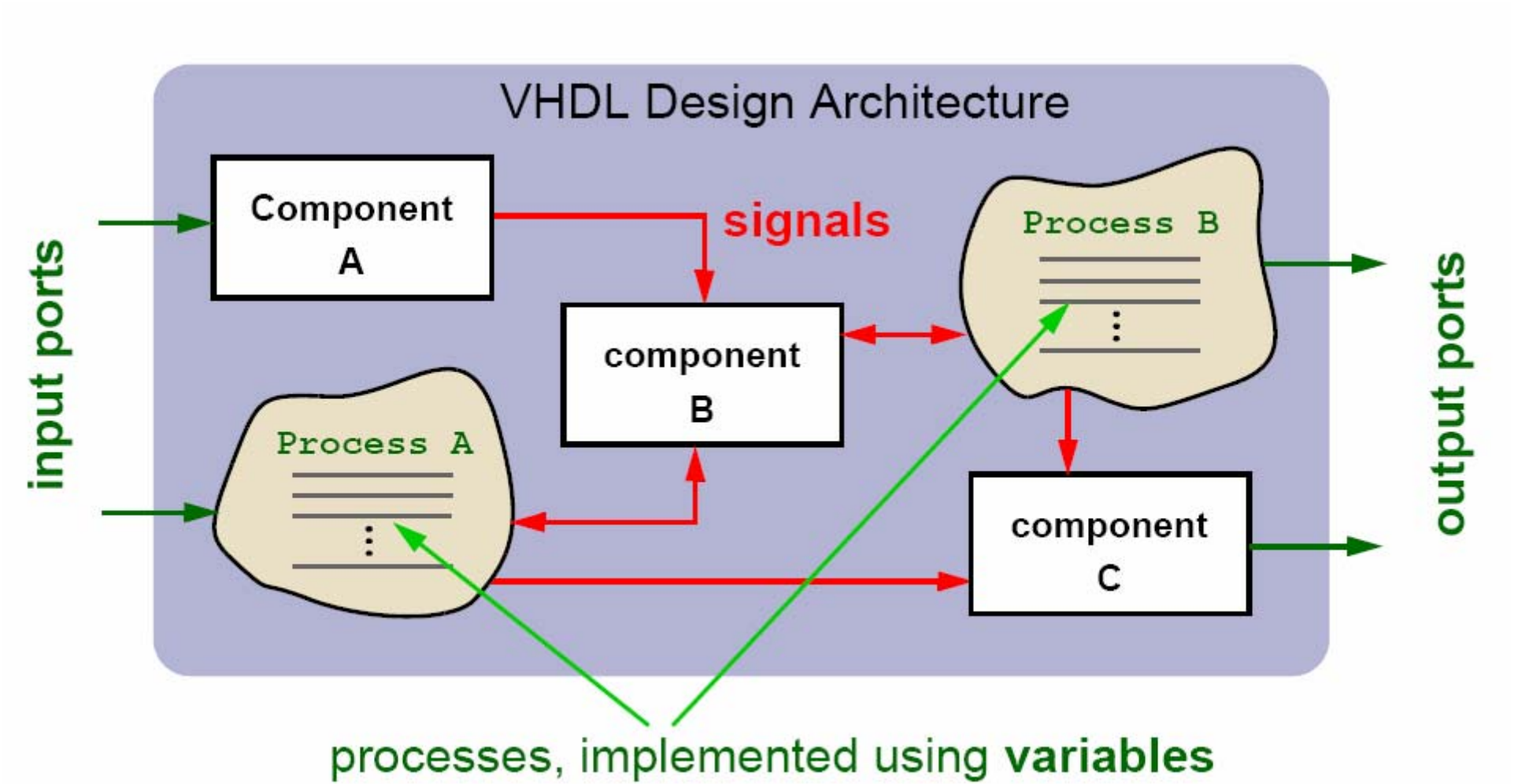
```
signal carry_out: std_logic := '0';
```

Sinais e Variáveis

- Sinais:
  - Podem ser utilizados para a comunicação entre processos.
  - Podem aparecer em listas de sensibilidade de um processo
  - Podem ter atrasos.

- Variáveis
  - Podem ser utilizadas apenas dentro do processo que foram declaradas
  - Não podem aparecer em listas de sensibilidade
  - Não podem ter atrasos

Sinais e Variáveis



Exemplos de Tipos de Sinais: BIT

- BIT (and BIT_VECTOR): 2-level logic ('0', '1').

Examples:

```
SIGNAL x: BIT;
-- x is declared as a one-digit signal of type BIT.

SIGNAL y: BIT_VECTOR (3 DOWNTO 0);
-- y is a 4-bit vector, with the leftmost bit being the MSB.

SIGNAL w: BIT_VECTOR (0 TO 7);
-- w is an 8-bit vector, with the rightmost bit being the MSB.


x <= '1';
-- x is a single-bit signal (as specified above), whose value is
-- '1'. Notice that single quotes (' ') are used for a single bit.

y <= "0111";
-- y is a 4-bit signal (as specified above), whose value is "0111"
-- (MSB='0'). Notice that double quotes (" ") are used for
-- vectors.

w <= "01110001";
-- w is an 8-bit signal, whose value is "01110001" (MSB='1').
```

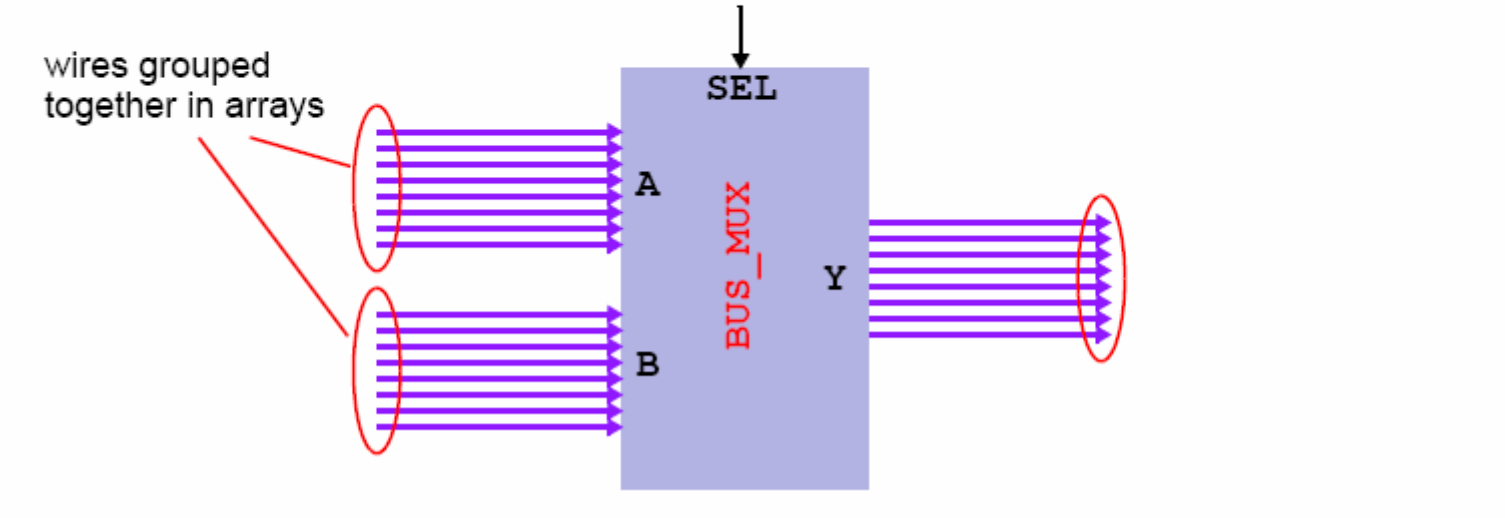
Exemplos de Tipos de Sinais: STD_LOGIC

- STD_LOGIC (and STD_LOGIC_VECTOR): 8-valued logic system introduced in the IEEE 1164 standard.

'X'	Forcing Unknown
'0'	Forcing Low
'1'	Forcing High
'Z'	High impedance
'W'	Weak unknown
'L'	Weak low
'H'	Weak high
'–'	Don't care

Exemplo

```
entity BUS_MUX is

    port(A, B : in  std_logic_vector(7 downto 0);
         SEL  : in  std_logic;
         Y    : out std_logic_vector(7 downto 0));

end entity BUS_MUX;
```



Outros Tipos de Dados

- Boolean: Verdadeiro e falso.
- Integer: inteiro de 32 bits [-2.147.483.647 até +2.147.483.647].
- Natural: Inteiros não negativos
- Signed e unsigned: Definidos no pacote *std_logic_arith*. São semelhantes ao STD_LOGIC_VECTOR mas aceitam operações aritméticas.

Exemplos

```
SIGNAL a: BIT;
SIGNAL b: BIT_VECTOR(7 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;
...
a <= b(5);     -- legal (same scalar type: BIT)
b(0) <= a;     -- legal (same scalar type: BIT)
c <= d(5);     -- legal (same scalar type: STD_LOGIC)
d(0) <= c;     -- legal (same scalar type: STD_LOGIC)
a <= c;        -- illegal (type mismatch: BIT x STD_LOGIC)
b <= d;        -- illegal (type mismatch: BIT_VECTOR x
               -- STD_LOGIC_VECTOR)
e <= b;        -- illegal (type mismatch: INTEGER x BIT_VECTOR)
e <= d;        -- illegal (type mismatch: INTEGER x
               -- STD_LOGIC_VECTOR)
```

Consultar bibliografia (tipos de dado, operadores, atributos)

- *Digital Electronics and Design with VHDL*; ***Volnei A. Pedroni***; *Elsevier Science, 2007*

- *Circuit Design with VHDL*; ***Volnei A. Pedroni***; *MIT Press, 2004*

Exemplo: Contador de um dígito

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------
ENTITY counter IS
   PORT (clk : IN STD_LOGIC;
         digit : OUT INTEGER RANGE 0 TO 9);
END counter;
-----------------------------------------
ARCHITECTURE counter OF counter IS
BEGIN
   PROCESS          -- no sensitivity list
      VARIABLE temp : INTEGER RANGE 0 TO 10;
   BEGIN
      WAIT UNTIL (clk'EVENT AND clk='1');
      temp := temp + 1;
      IF (temp=10) THEN temp := 0;
      END IF;
      digit <= temp;
   END PROCESS;
END counter;
```

Exercícios

- Exercício 1: Descreva um código que para cada dois bits de entrada coloque quatro bits na saída.
- Exercício 2: Faça um contador com dois dígitos que conte de 0 a 59.