

**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...).

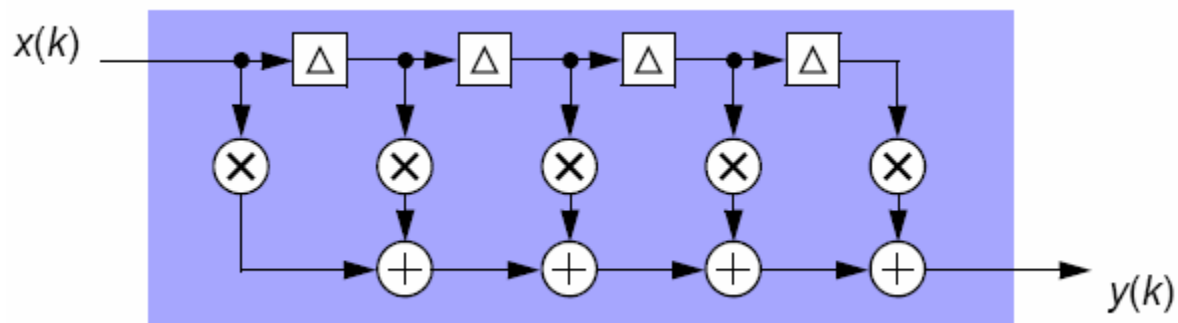
Aplicações de DSP

- 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

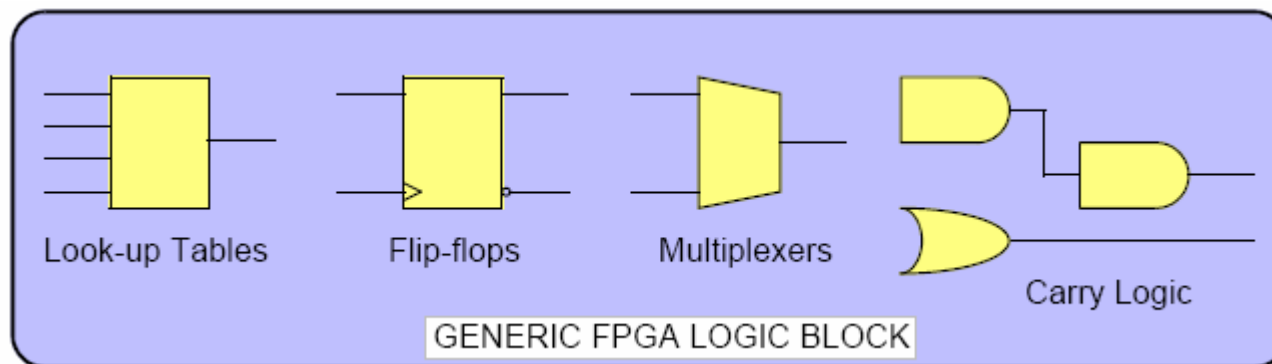
$$y(k) = \sum_{n=0}^4 w_n x(k-n)$$



- 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.

Bloco Lógico de um FPGA

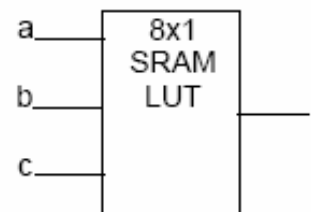
- Os blocos lógicos contêm componentes lógicos que implementam funções lógicas e aritméticas. Para a fabricante Xilinx, os blocos lógicos são chamados de *Slices*.



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 booleanas (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.

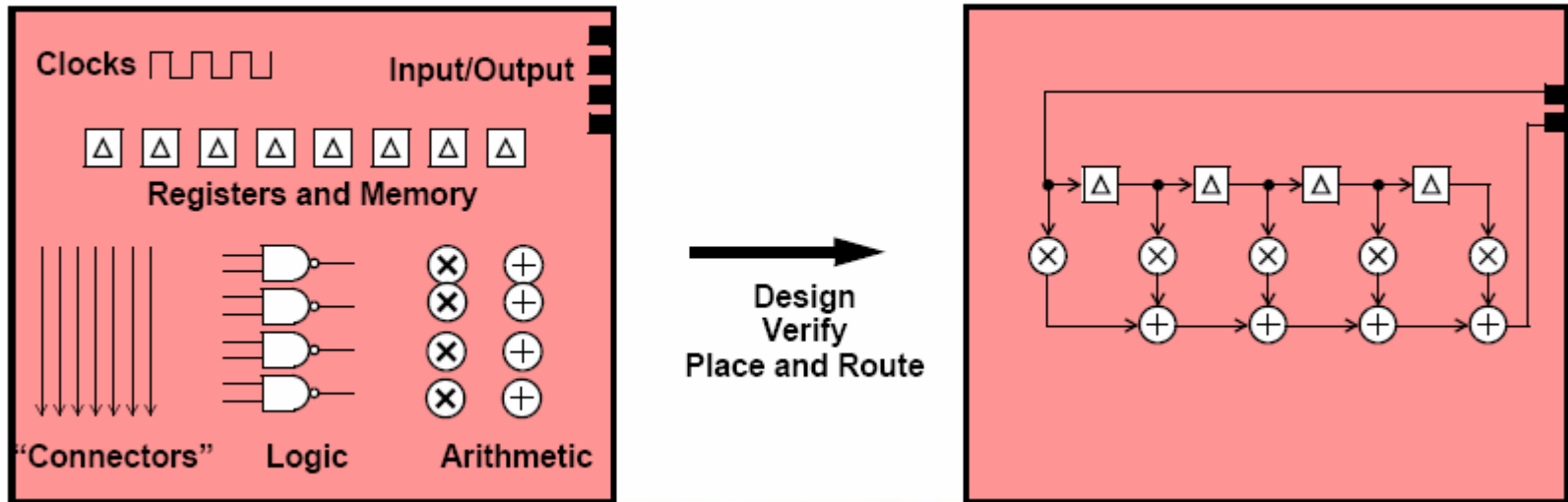


Programação do FPGA

- 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

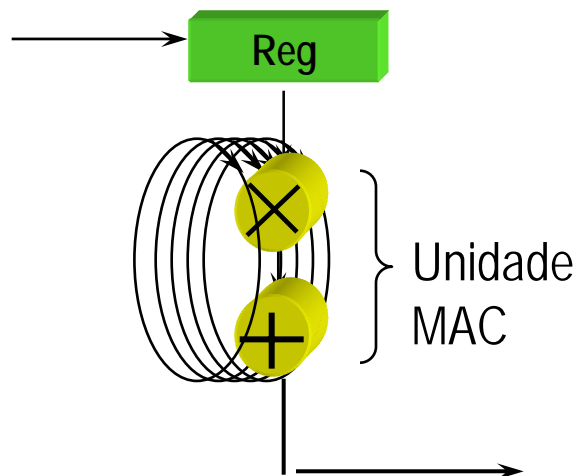


FPGA vs. Processadores DSP

- 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.

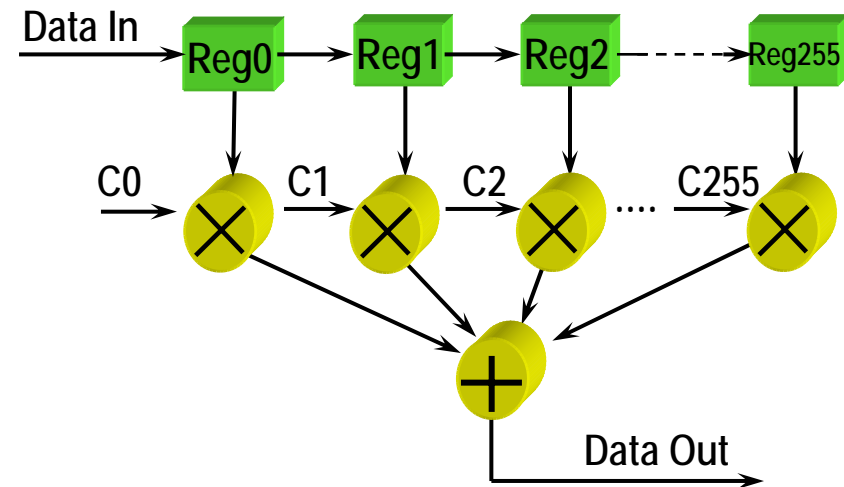
Filtro FIR com 256 coeficientes

DSP Convencional



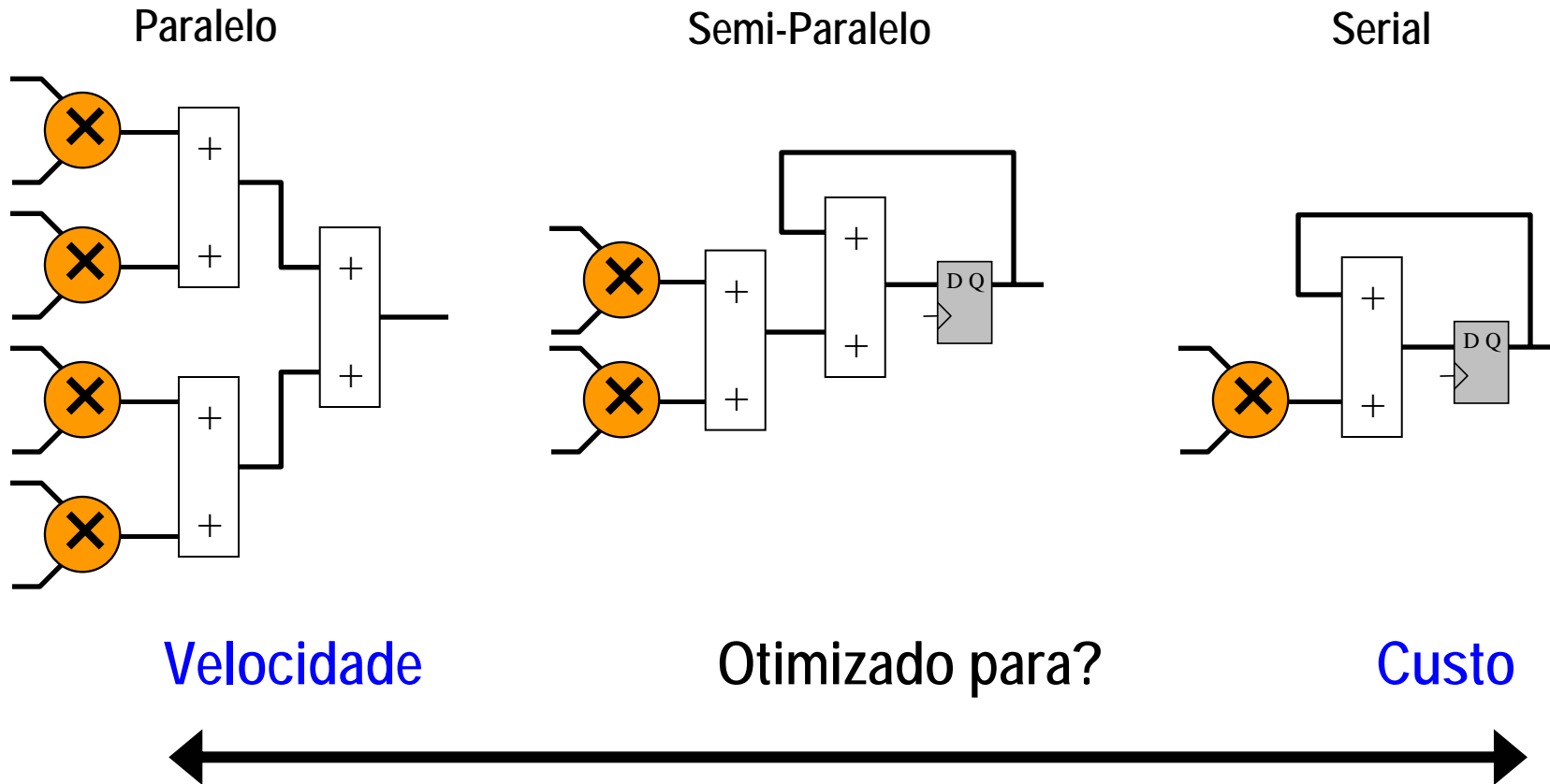
256 Loops necessários para processar amostras

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

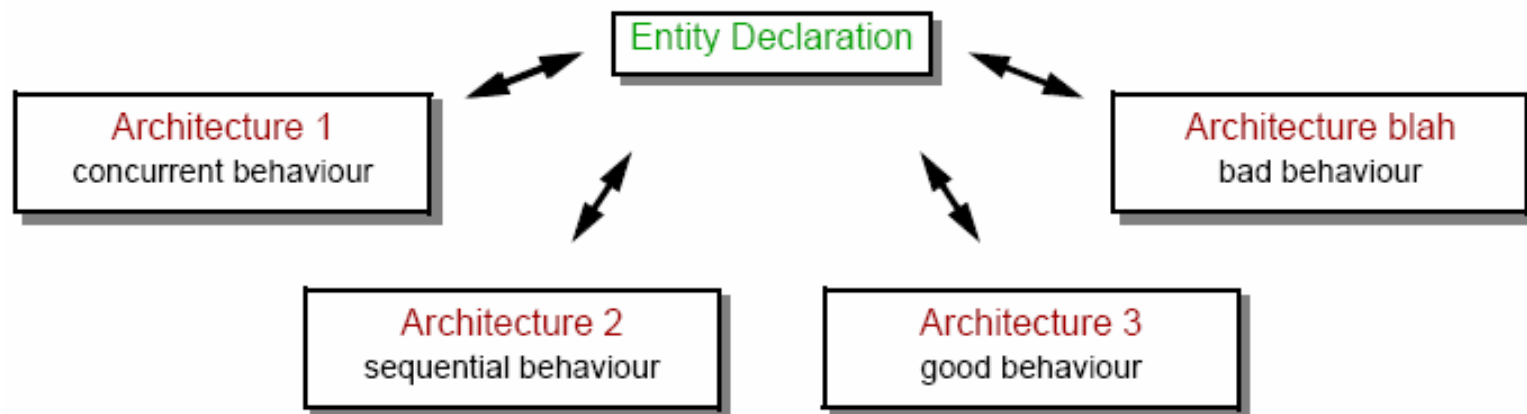
- VHDL (Very High Specific Integrated Circuit **H**ardware **D**escription **L**anguage) é uma linguagem de descrição de hardware.
- A linguagem VHDL é um padrão especificado pelo IEEE 1076 e IEEE 1164.
- Por ser um padrão o código desenvolvido independe do fabricante.

- 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.

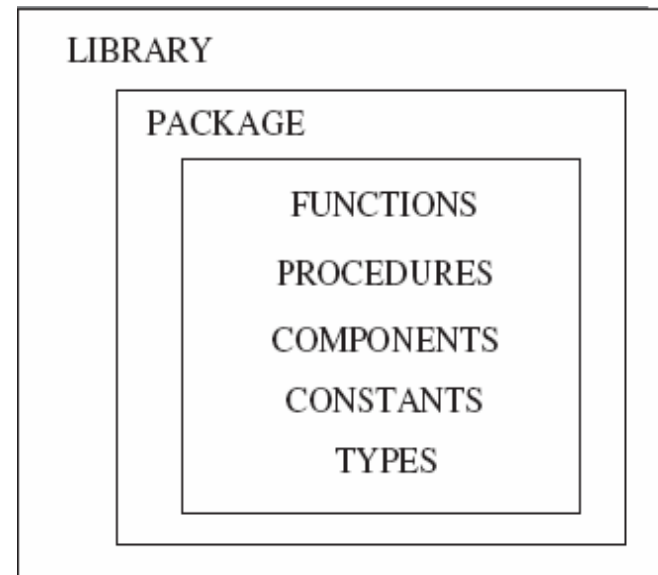
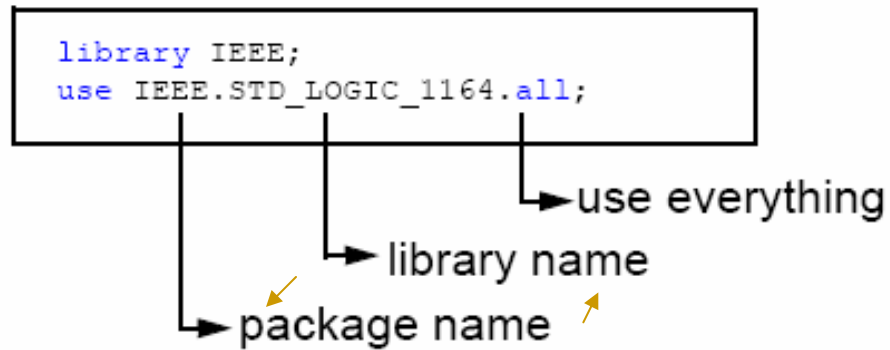
Unidades básicas do VHDL

- ***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.

- 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



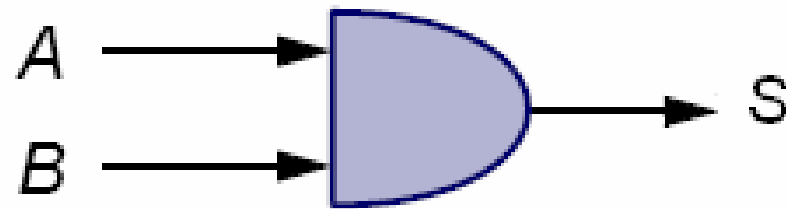
As bibliotecas 'work' e 'std' são incluídas por definição, não necessitando serem citadas no código.

Declaração de Entidade

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

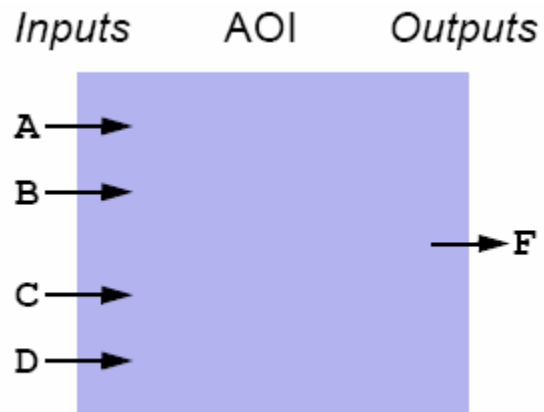
- O modo do sinal pode ser *in* (entrada), *out* (saída), *inout* (bidirecional) e *buffer* (saída e realimentação externa).
- O tipo pode ser *bit*, *std_logic*, *integer*, etc.
- O nome da entidade deve ser o mesmo do arquivo *.vhd*.

Exemplo 1



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

Exemplo 2



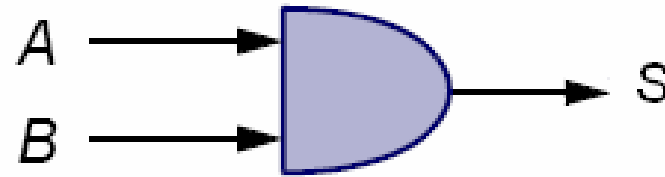
```
entity AOI is  
    port (A, B, C, D : in  std_logic;  
          F          : out std_logic);  
end AOI;
```

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.

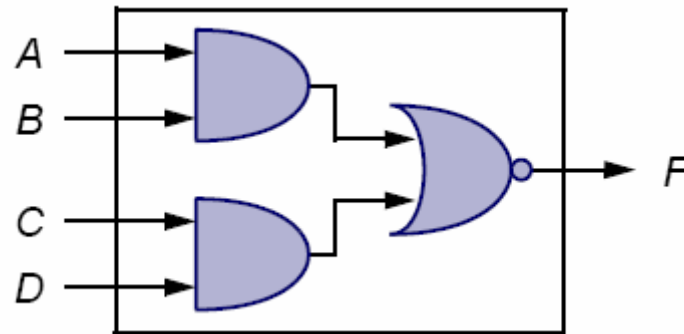
Exemplo 1



```
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  $\overline{\text{AOI}}$  is
begin
    F <= not ((A and B) or (C and D));
end V1;
```

Exemplo: Porta E

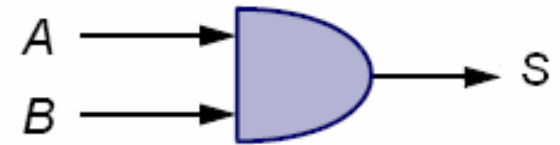
```
ENTITY PORTA_E IS
  PORT
  (
    A, B: IN BIT;
    S: OUT BIT
  );
END ENTITY PORTA_E;

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

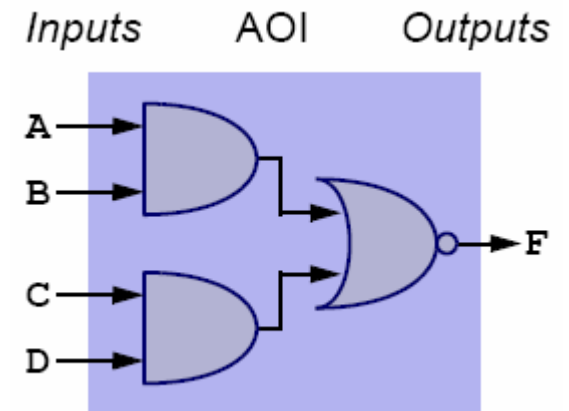
    port (A, B, C, D : in  std_logic;
          F           : out std_logic);

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	FILE	OF	THEN
ACCESS	FOR	ON	TO
AFTER	FUNCTION	OPEN	TRANSPORT
ALIAS		OR	TYPE
ALL	GENERATE	OTHERS	
AND	GENERIC	OUT	UNAFFECTED
ARCHITECTURE	GROUP		UNITS
ARRAY	GUARDED	PACKAGE	UNTIL
ASSERT		PORT	USE
ATTRIBUTE	IF	POSTPONED	
	IMPURE	PROCEDURE	VARIABLE
BEGIN	IN	PROCESS	
BLOCK	INERTIAL	PURE	WAIT
BODY	INOUT		WHEN
BUFFER	IS	RANGE	WHILE
BUS		RECORD	WITH
	LABEL	REGISTER	
CASE	LIBRARY	REJECT	XNOR
COMPONENT	LINKAGE	REM	XOR
CONFIGURATION	LITERAL	REPORT	
CONSTANT	LOOP	RETURN	
		ROL	
DISCONNECT	MAP	ROR	
DOWNTO	MOD		
		SELECT	
ELSE	NAND	SEVERITY	
ELSIF	NEW	SIGNAL	
END	NEXT	SHARED	
ENTITY	NOR	SLA	
EXIT	NOT	SLL	
	NULL	SRA	
		SRL	
		SUBTYPE	



Introdução ao Quartus II

➤ www.altera.com



Código Concorrente (paralelos) e Sequencial

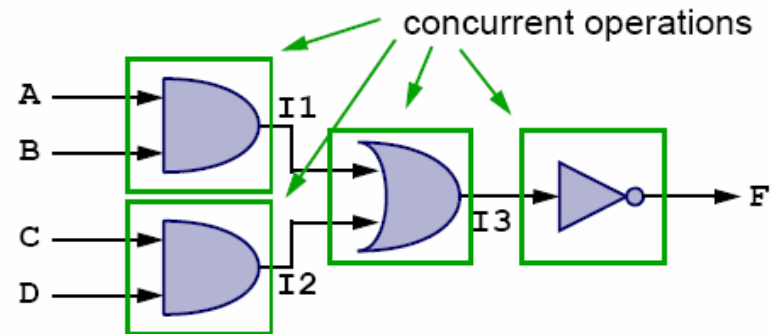
- Os códigos paralelos em geral implementam circuitos lógicos combinacionais, onde a saída do circuito depende apenas da entrada atual
- Os códigos sequenciais implementam circuitos lógicos sequenciais, onde a saída do circuito depende de entradas anteriores

Código Concorrente

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

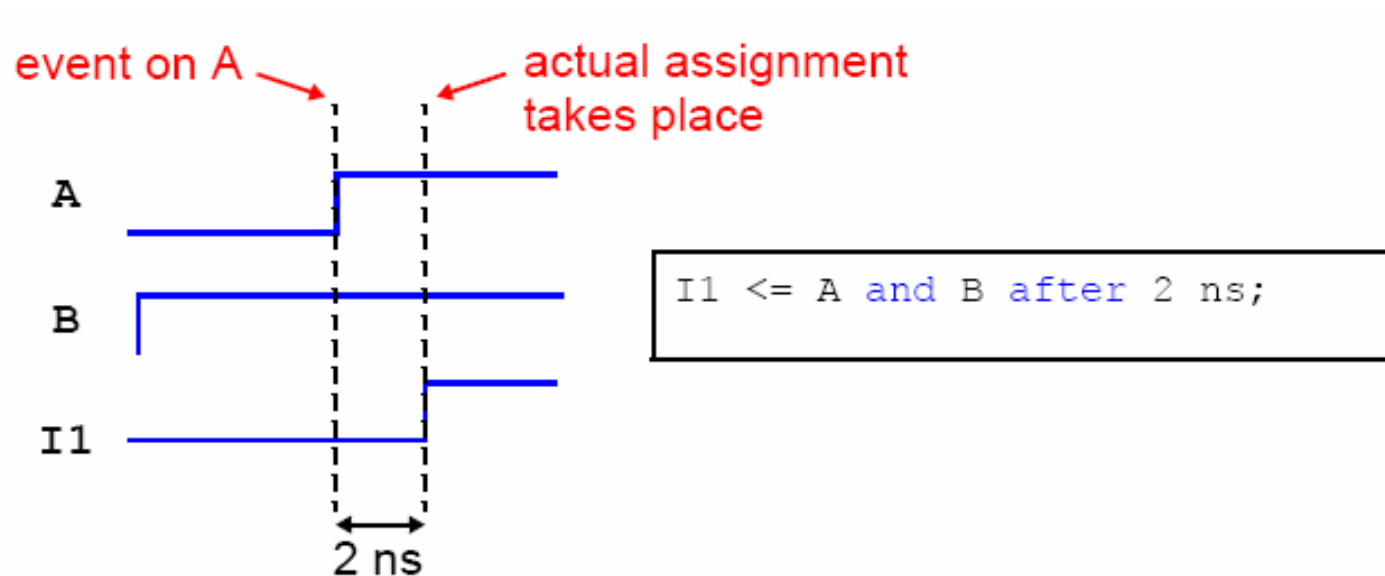
```
-- entity declaration
entity AOI is
  port(A, B, C, D : in  std_logic;
        F          : out std_logic);
end AOI;

-- 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

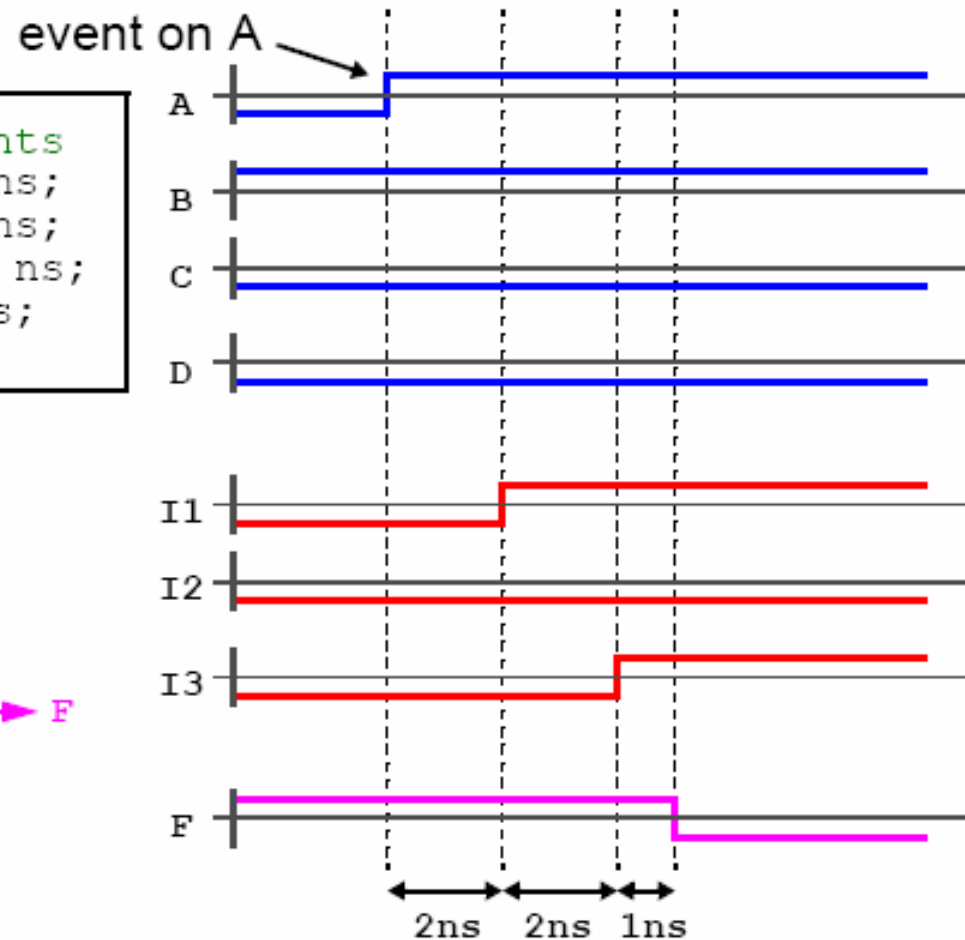
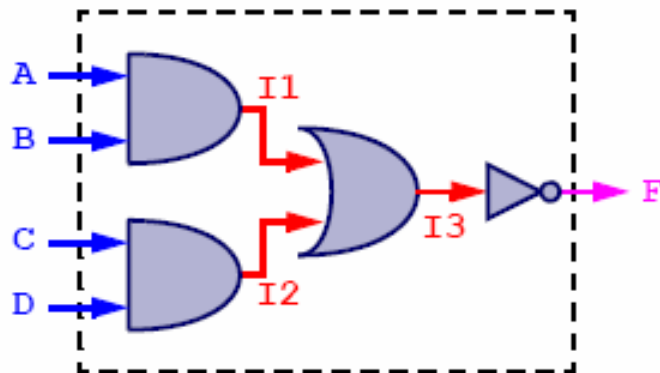


Código Concorrente

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

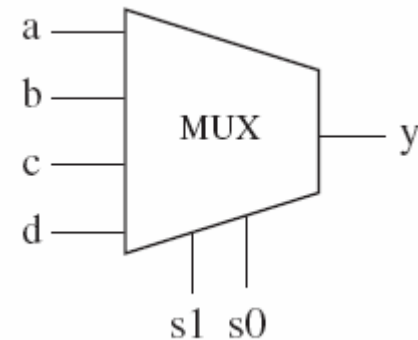
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-----

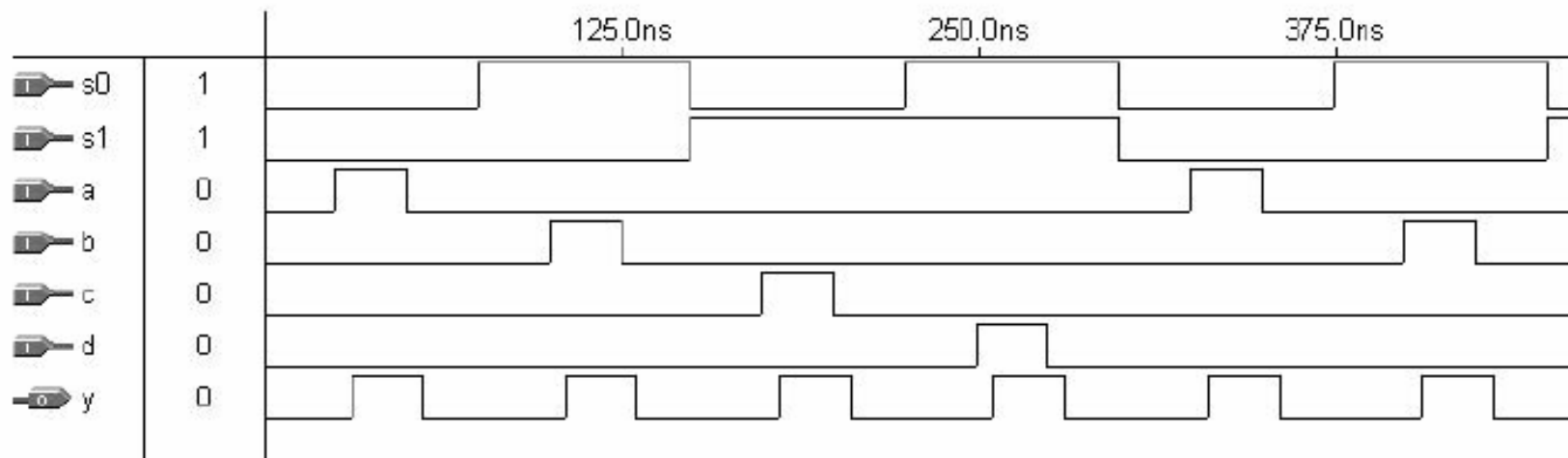
ENTITY mux IS
    PORT ( a, b, c, d, s0, s1: IN STD_LOGIC;
          y: OUT STD_LOGIC);
END mux;

-----

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



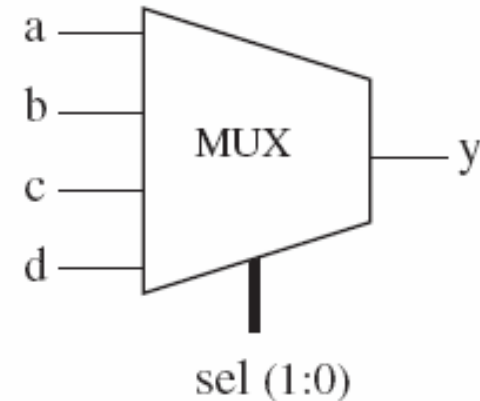
Exemplo 2: Multiplexador (when/else)

```
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;

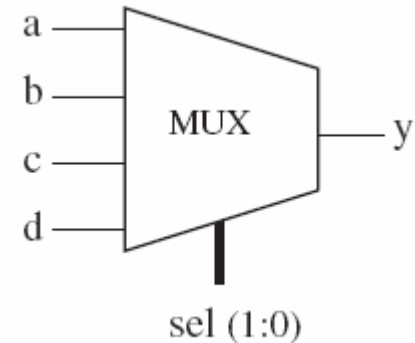
-----

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;
```



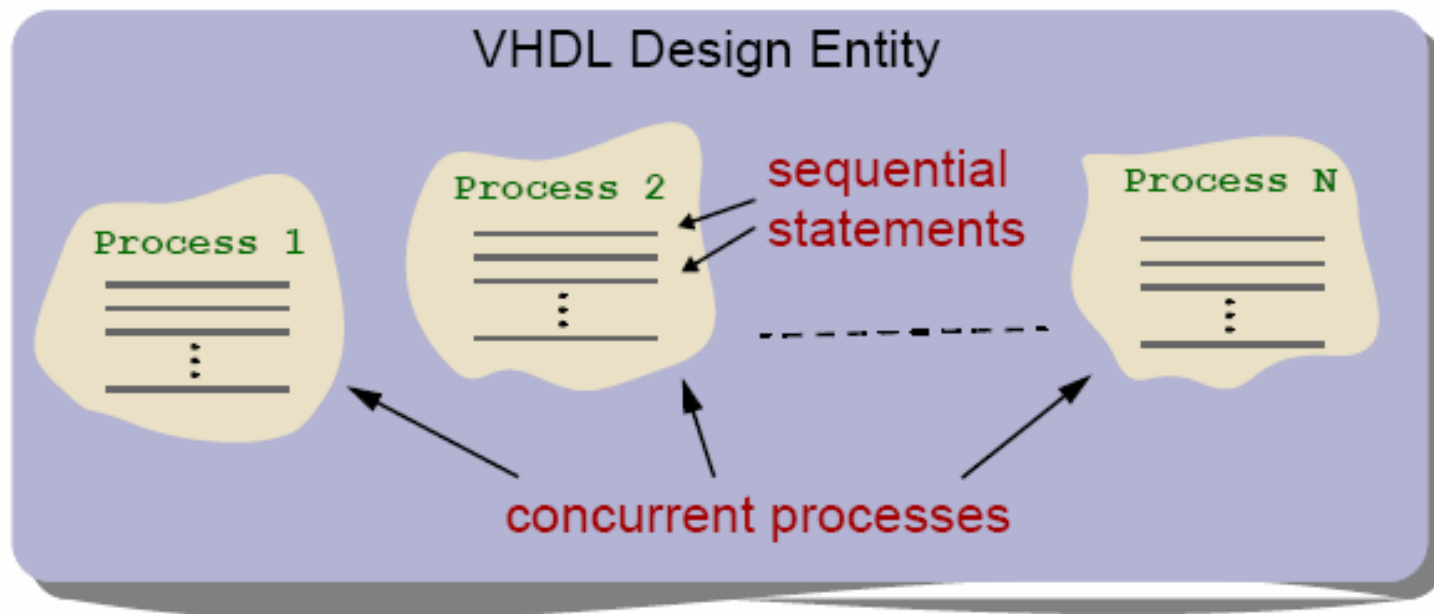
Exemplo 3: Multiplexador (with/select/when)

```
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

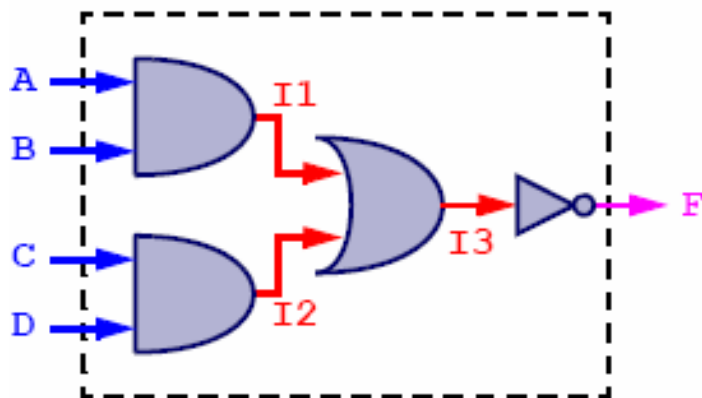
- A lista de sensibilidade é opcional.
- Processos com lista de sensibilidade apenas são executados quando um evento ocorre.
- O evento é uma mudança do sinal contido na lista de sensibilidade.

```
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;

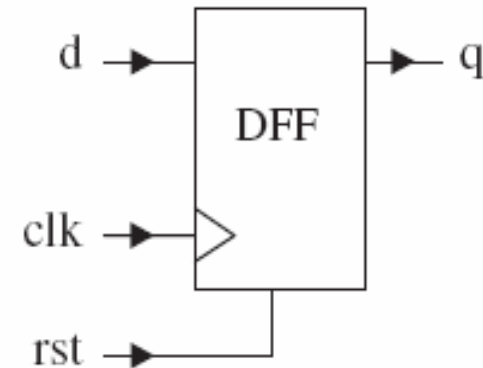
    -- process to model the inverter
    NOT_GATE : process (I3)
    begin
        F <= not I3 after 1 ns;
    end process;

end V3;
```

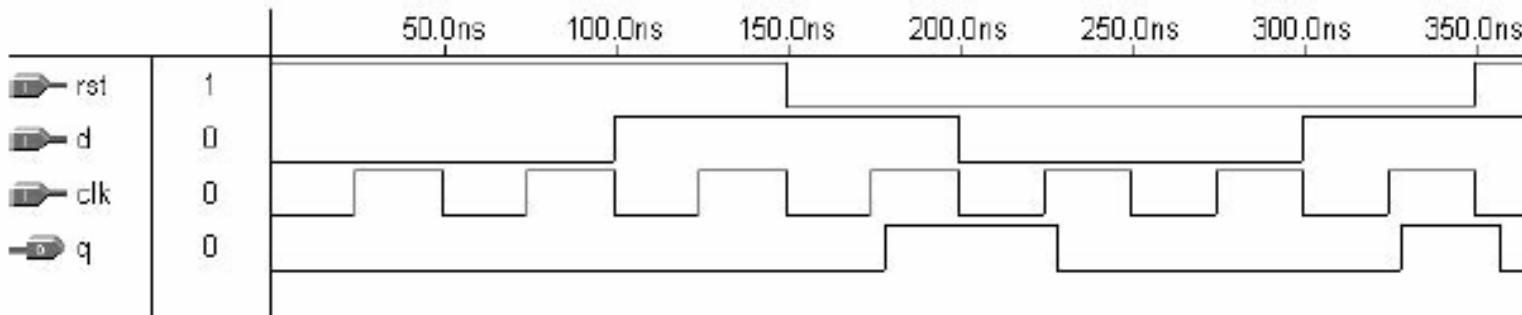



Exemplo 1: Flip-flop tipo D com reset assíncrono.

```
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;
```

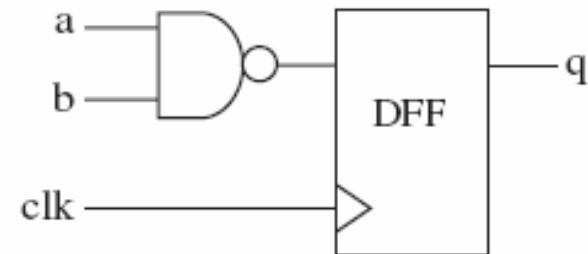


Resultado da simulação

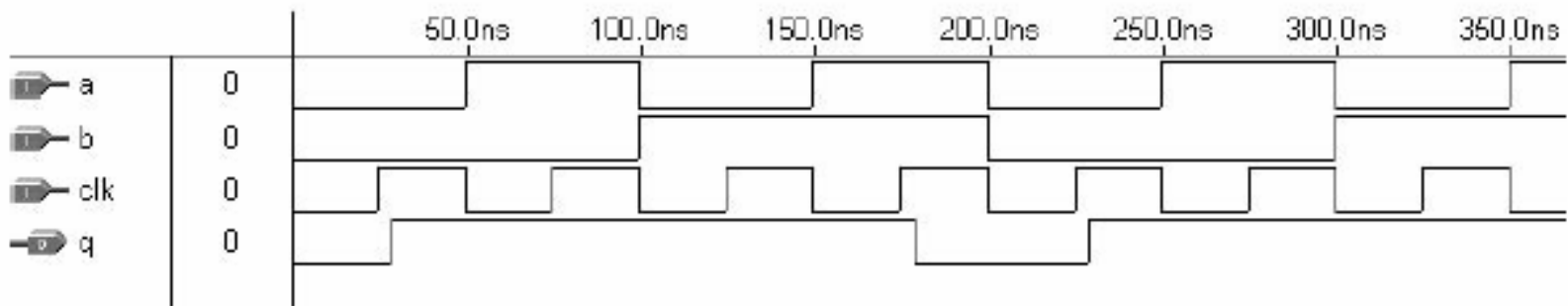


Exemplo 2: Flip-flop D com porta nand

```
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;
```

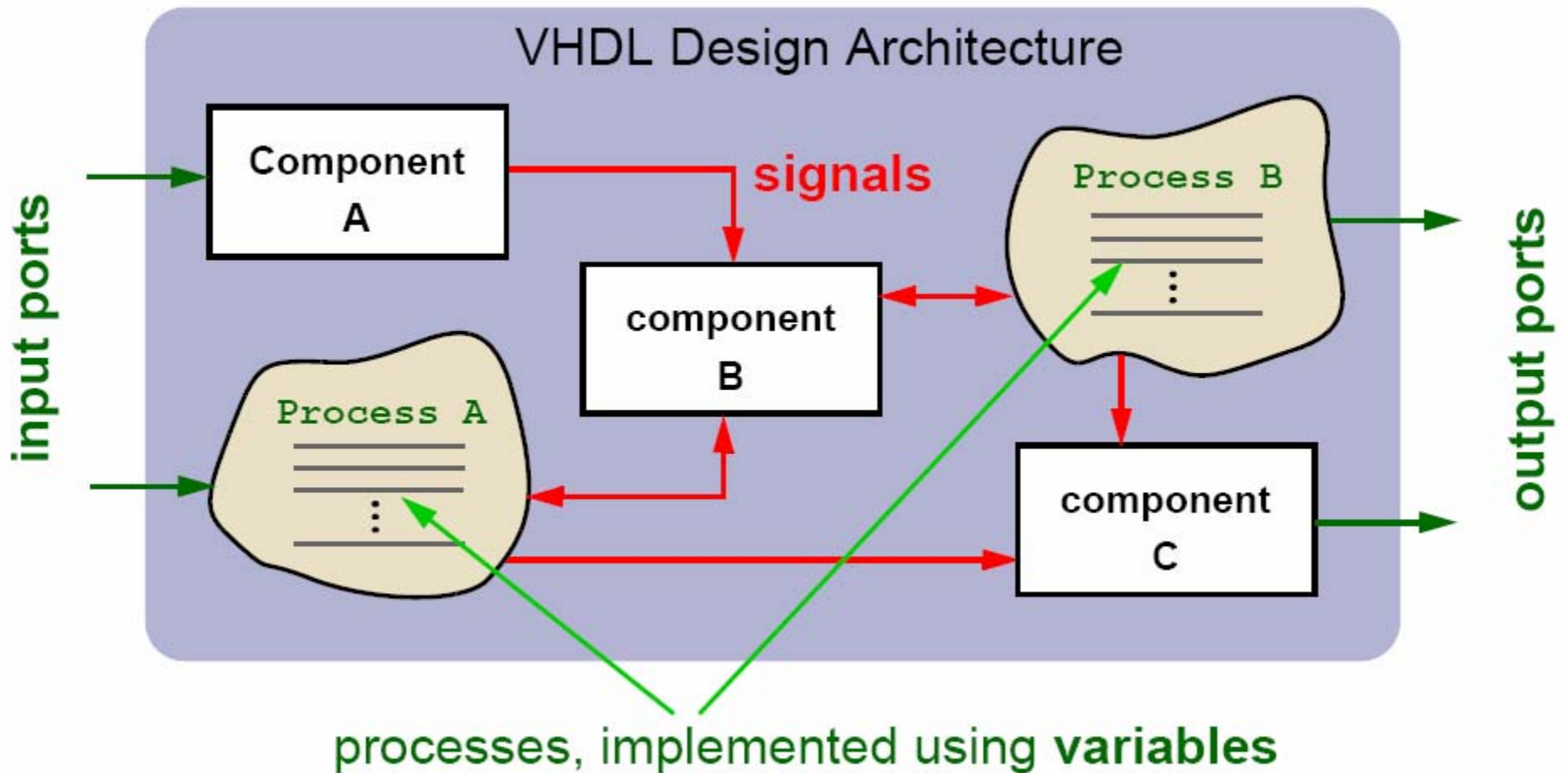
```
signal carry_out: std_logic := '0';
```

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 DOWNT0 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').
```



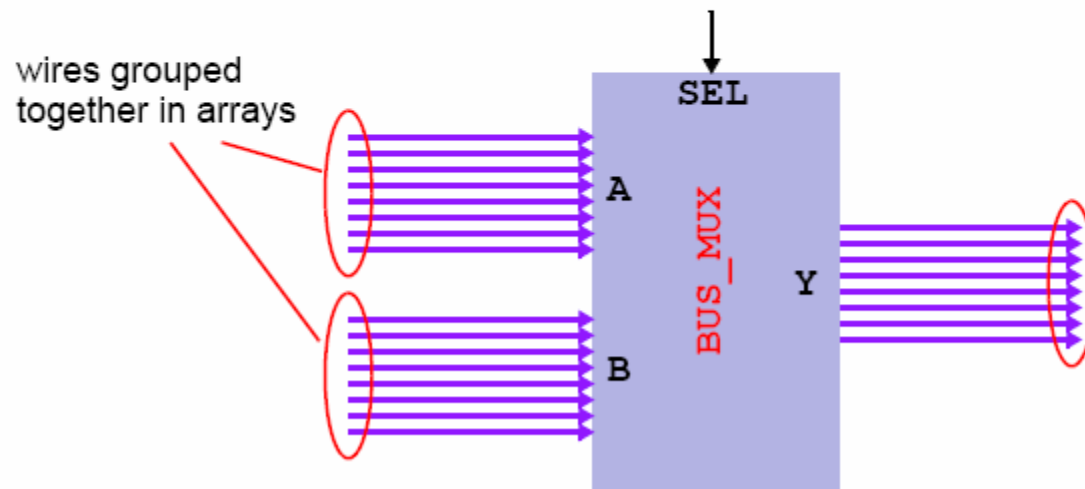
Exemplos de Tipos de Sinais: STD_LOGIC

- 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.



Exercício 1: Solução 1

- **Solução (Processo implícito): Decodificador**
- LIBRARY IEEE;
- USE IEEE.STD_LOGIC_1164.ALL;
- ENTITY exe_cod_2_4 IS
- PORT
- (
• info: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
• saida_cod: OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
-);
- END ENTITY exe_cod_2_4;
- ARCHITECTURE regra OF exe_cod_2_4 IS
- BEGIN
- saida_cod <= "0001" WHEN info = "00" ELSE
- "0010" WHEN info = "01" ELSE
- "0100" WHEN info = "10" ELSE
- "1000" WHEN info = "11" ELSE
- "0000";
- END ARCHITECTURE;



Exercício 1: Solução 2

- **Solução (Processo Explícito): Decodificador**
- LIBRARY IEEE;
- USE IEEE.STD_LOGIC_1164.ALL;

- ENTITY exe_cod_2_4 IS
- PORT
- (
- info: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
- saida_cod: OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
-);
- END ENTITY exe_cod_2_4;

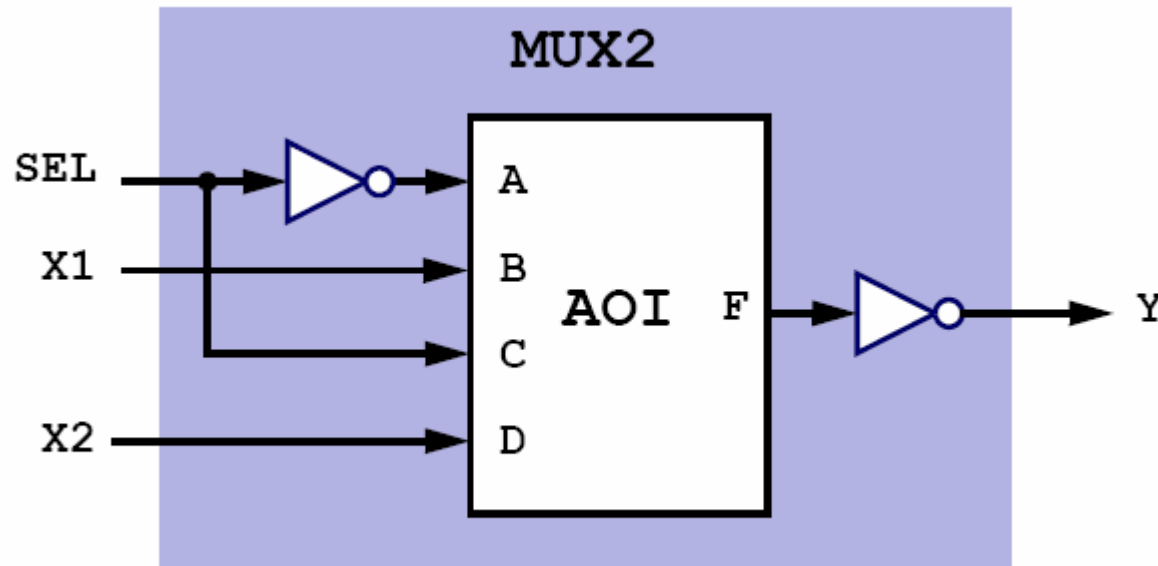
- ARCHITECTURE regra OF exe_cod_2_4 IS
- BEGIN
- PROCESS (info)
- BEGIN
- IF info = "00" THEN
- saida_cod <= "0001";
- ELSIF info = "01" THEN
- saida_cod <= "0010";
- ...
- ELSE
- saida_cod <= "0000";
- END IF
- ...

Componentes

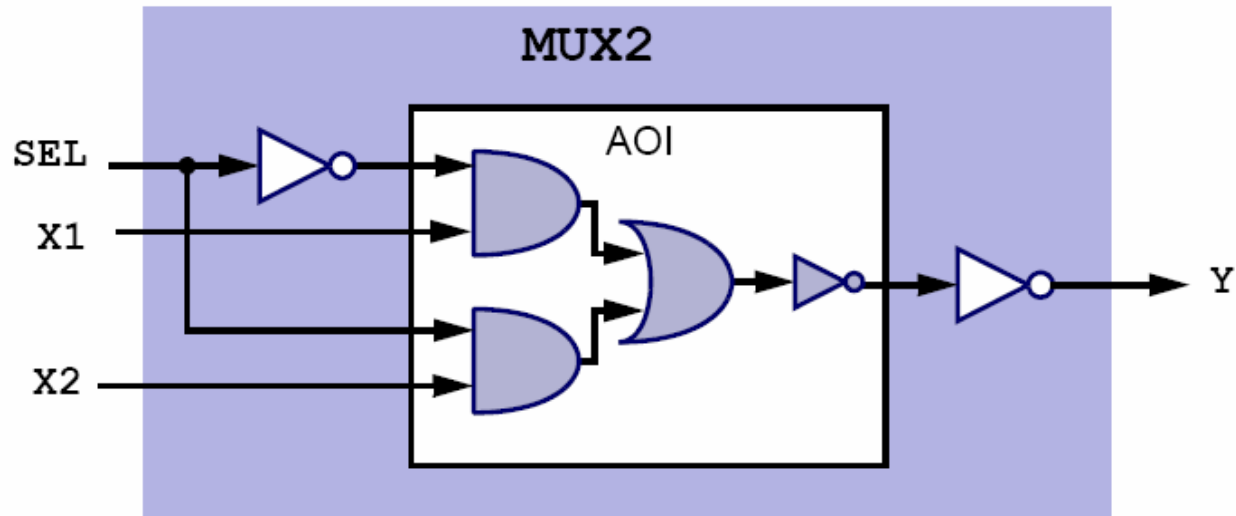
- Uma entidade já projetada que será utilizada para o projeto de uma entidade de nível maior é chamada de *componente*.
- Quando uma componente é utilizada, devemos especificar suas conectividades com outros componente, sinais e portas dentro da entidade.

Exemplo: Multiplexador

- Considerando o projeto MUX2, devemos utilizar a entidade de projeto AOI como uma componente de MUX2.



Entidade do MUX2



```
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
  
entity MUX2 is  
    port (SEL, X1, X2 : in  std_logic;  
          Y           : out std_logic);  
end entity MUX2;
```

Arquitetura do MUX2

- A palavra-chave *port map* especifica as conectividades do componente.

```
architecture ARCH1 of MUX2 is

    component AOI -- declare component
        port(A, B, C, D : in std_logic;
             F           : out std_logic);
    end component;

    signal SELB, T : std_logic;

begin

    SELB <= not SEL;

    AOI_inst : AOI -- instantiate component
    port map (A => SELB, B => X1, C => SEL, D => X2, F => T);

    Y <= not T;

end ARCH1;
```



Declaração de Vários Componentes

```
architecture GEN_ARCH of GEN_ENT is

    component C1
        -- declare C1 interface here
    end component;

    component C2
        -- declare C2 interface here
    end component;

    signal S1, S2, S3 .....

begin

    -- declare C1 instances
    C1_inst1 : C1 port map ( ..... );
    C1_inst2 : C1 port map ( ..... );
    :       :       :       :
    C1_instN : C1 port map ( ..... );

    -- declare C2 instances
    C2_inst1 : C2 port map ( ..... );
    C2_inst2 : C2 port map ( ..... );
    :       :       :       :
    C2_instM : C2 port map ( ..... );

end GEN_ARCH;
```

Mapeamento das portas

- Sintaxe por associação de nome.

```
[instance_label] : [design entity]  
    port map ( [lp1] => [gp1], [lp2] => [gp2], ..... );
```

- Cada associação é da forma: *porta local* => *porta global*
 - *Porta local*: Portas dos componentes.
 - *Porta global*: Portas/sinais que estão acessíveis dentro da arquitetura do projeto.



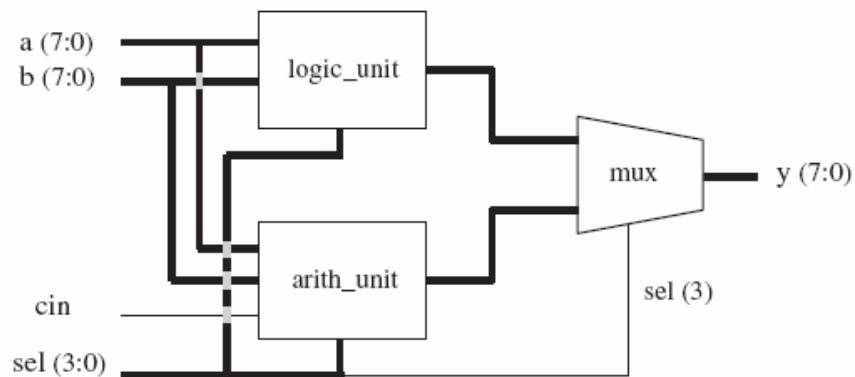
Mapeamento das Portas: Associação por Posição

- Neste tipo de associação a ordem do mapeamento que define as conexões.

```
AOI_inst : AOI                -- positional association  
  port map (SELB, X1, SEL, X2, T);
```

Exercício

- Exercício 1: Faça um projeto de uma unidade lógica aritmética (ULA) conforme a figura abaixo. Utilize o conceito de componentes. Gere um arquivo de formas de onda para simular e verificar o funcionamento do projeto.



sel	Operation	Function	Unit
0000	$y \leq a$	Transfer a	Arithmetic
0001	$y \leq a+1$	Increment a	
0010	$y \leq a-1$	Decrement a	
0011	$y \leq b$	Transfer b	
0100	$y \leq b+1$	Increment b	
0101	$y \leq b-1$	Decrement b	
0110	$y \leq a+b$	Add a and b	
0111	$y \leq a+b+cin$	Add a and b with carry	
1000	$y \leq \text{NOT } a$	Complement a	Logic
1001	$y \leq \text{NOT } b$	Complement b	
1010	$y \leq a \text{ AND } b$	AND	
1011	$y \leq a \text{ OR } b$	OR	
1100	$y \leq a \text{ NAND } b$	NAND	
1101	$y \leq a \text{ NOR } b$	NOR	
1110	$y \leq a \text{ XOR } b$	XOR	
1111	$y \leq a \text{ XNOR } b$	XNOR	



Solução: Unidade Aritmética

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

-----

ENTITY uni_arit IS
PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      sel: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
      cin: IN STD_LOGIC;
      x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END uni_arit;

-----

ARCHITECTURE unidade OF uni_arit IS
SIGNAL arith, logic: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
  WITH sel SELECT
    x <= a  WHEN "000",
        a+1 WHEN "001",
        a-1 WHEN "010",
        b   WHEN "011",
        b+1 WHEN "100",
        b-1 WHEN "101",
        a+b WHEN "110",
        a+b+cin WHEN OTHERS;
END unidade;
```



Solução: Unidade Lógica

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

-----

ENTITY uni_logica IS
PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      sel: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
      x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END uni_logica;

-----

ARCHITECTURE unidade OF uni_logica IS
BEGIN
    WITH sel SELECT
        x <= NOT a    WHEN "000",
           NOT b     WHEN "001",
           a AND b   WHEN "010",
           a OR b    WHEN "011",
           a NAND b  WHEN "100",
           a NOR b   WHEN "101",
           a XOR b   WHEN "110",
           NOT (a XOR b) WHEN OTHERS;
END unidade;
```



Solução: Mux

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

-----

ENTITY mux1 IS
PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      sel: IN STD_LOGIC;
      x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END mux1;

-----

ARCHITECTURE funcionamento OF mux1 IS
BEGIN
    WITH sel SELECT
        x <= a WHEN '0',
        b WHEN OTHERS;
END funcionamento;
```



Solução: Código Principal - ULA

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

-----

ENTITY ula IS
PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      cin: IN STD_LOGIC;
      sel: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      y: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END ula;

-----

ARCHITECTURE funcionamento OF ula IS

-----

COMPONENT uni_arit IS
PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      cin: IN STD_LOGIC;
      sel: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
      x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;

-----

COMPONENT uni_logica IS
PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      sel: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
      x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;

-----

COMPONENT mux1 IS
PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      sel: IN STD_LOGIC;
      x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;

-----

SIGNAL x1, x2: STD_LOGIC_VECTOR(7 DOWNTO 0);

-----

BEGIN
    U1: uni_arit PORT MAP (a, b, cin, sel(2 DOWNTO 0), x1);
    U2: uni_logica PORT MAP (a, b, sel(2 DOWNTO 0), x2);
    U3: mux1 PORT MAP (x1, x2, sel(3), y);
END funcionamento;
```

Funções e Procedimentos

- São pedaços de códigos VHDL sequenciais que podem ser utilizados ou compartilhados por diferentes projetos.
- Podem ser declarados nos pacotes ou no programa principal.

➤ Corpo da função:

```
FUNCTION function_name [<parameter list>] RETURN data_type IS  
[declarations]  
BEGIN  
(sequential statements)  
END function_name;
```

➤ Exemplo:

```
FUNCTION f1 (a, b: INTEGER; SIGNAL c: STD_LOGIC_VECTOR)  
RETURN BOOLEAN IS  
BEGIN  
(sequential statements)  
END f1;
```


- Função localizada no código principal

```
ARCHITECTURE my_arch OF my_entity IS
-----
    FUNCTION positive_edge(SIGNAL s: STD_LOGIC)
        RETURN BOOLEAN IS
    BEGIN
        RETURN s'EVENT AND s='1';
    END positive_edge;
-----
BEGIN
    PROCESS (clk, rst)
    BEGIN
        IF (rst='1') THEN q <= '0';
        ELSIF positive_edge(clk) THEN q <= d;
        END IF;
    END PROCESS;
END my_arch;
```

➤ Função localizada em um pacote

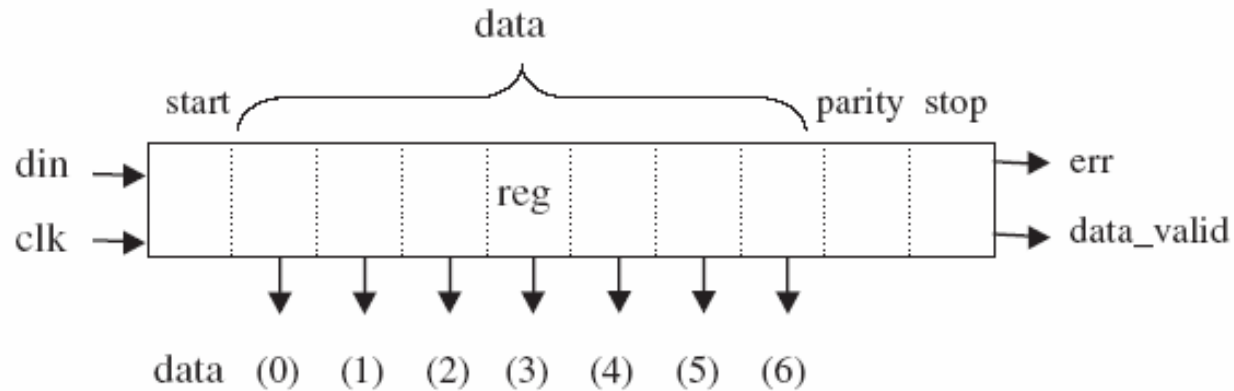
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----
PACKAGE my_package IS
    FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN;
END my_package;
-----
PACKAGE BODY my_package IS
    FUNCTION positive_edge(SIGNAL s: STD_LOGIC)
    RETURN BOOLEAN IS
    BEGIN
        RETURN s'EVENT AND s='1';
    END positive_edge;
END my_package;
-----
----- Main code: -----
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
```

Procedimentos

- Os procedimentos são muito parecidos com as funções com a diferença que eles podem retornar mais de um valor.
- Exemplo:

```
PROCEDURE my_procedure ( a: IN BIT; SIGNAL b, c: IN BIT;  
                        SIGNAL x: OUT BIT_VECTOR(7 DOWNTO 0);  
                        SIGNAL y: INOUT INTEGER RANGE 0 TO 99) IS  
  
BEGIN  
  ...  
END my_procedure;
```

Exercício: Receptor de Dados Serial



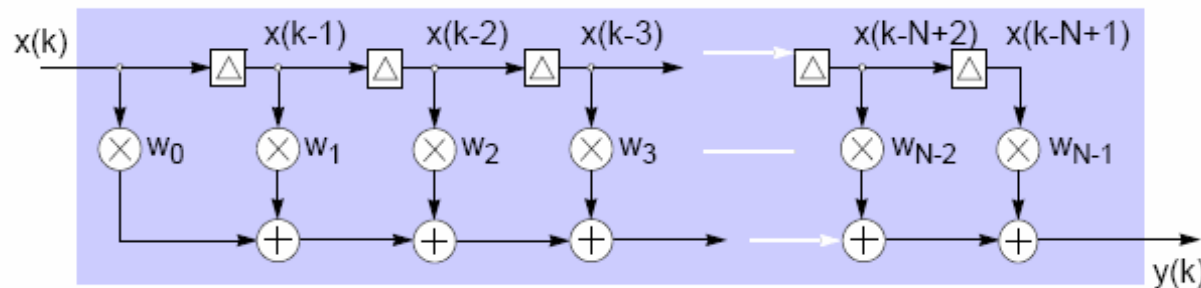
➤ Entidade

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
-----  
ENTITY receiver IS  
    PORT ( din, clk, rst: IN BIT;  
          data: OUT BIT_VECTOR (6 DOWNTO 0);  
          err, data_valid: OUT BIT);  
END receiver;  
-----
```



```
ARCHITECTURE rtl OF receiver IS
BEGIN
    PROCESS (rst, clk)
        VARIABLE count: INTEGER RANGE 0 TO 10;
        VARIABLE reg: BIT_VECTOR (10 DOWNT0 0);
        VARIABLE temp : BIT;
    BEGIN
        IF (rst='1') THEN
            count:=0;
            reg := (reg'RANGE => '0');
            temp := '0';
            err <= '0';
            data_valid <= '0';
        ELSIF (clk'EVENT AND clk='1') THEN
            IF (reg(0)='0' AND din='1') THEN
                reg(0) := '1';
            ELSIF (reg(0)='1') THEN
                count := count + 1;
                IF (count < 10) THEN
                    reg(count) := din;
                ELSIF (count = 10) THEN
                    temp := (reg(1) XOR reg(2) XOR reg(3) XOR
                        reg(4) XOR reg(5) XOR reg(6) XOR
                        reg(7) XOR reg(8)) OR NOT reg(9);
                    err <= temp;
                    count := 0;
                    reg(0) := din;
                    IF (temp = '0') THEN
                        data_valid <= '1';
                        data <= reg(7 DOWNT0 1);
                    END IF;
                END IF;
            END IF;
        END IF;
    END PROCESS;
END rtl;
```

Exercícios: Filtro FIR 8 taps



$$y(k) = \sum_{n=0}^{N-1} w_n x(k-n)$$

- Coeficientes:

$$[w_0=2^{-3}, w_1=2^{-2}, w_2=2^{-1}, w_3=2^{-0}, w_4=2^{-0}, w_5=2^{-1}, w_6=2^{-2}, w_7=2^{-3}]$$

- Para realizar a simulação, considere que o valor do sinal de entrada seja igual a 100.



Filtro FIR

```
library ieee;
use ieee.std_logic_1164.all;

entity FIR1 is
    port (clk : in std_logic;
          x   : in integer range 0 to 255;
          y   : out integer range 0 to 511);
end entity FIR1;

architecture arch1 of FIR1 is
begin
    process (clk)
        type RegType is array (7 downto 0) of integer;
        variable Reg: RegType := (others => 0);
```