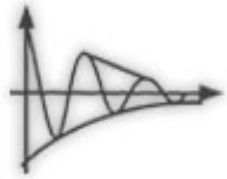
VI Escola do CBPF

Rio de Janeiro, 17 a 28 de julho de 2006

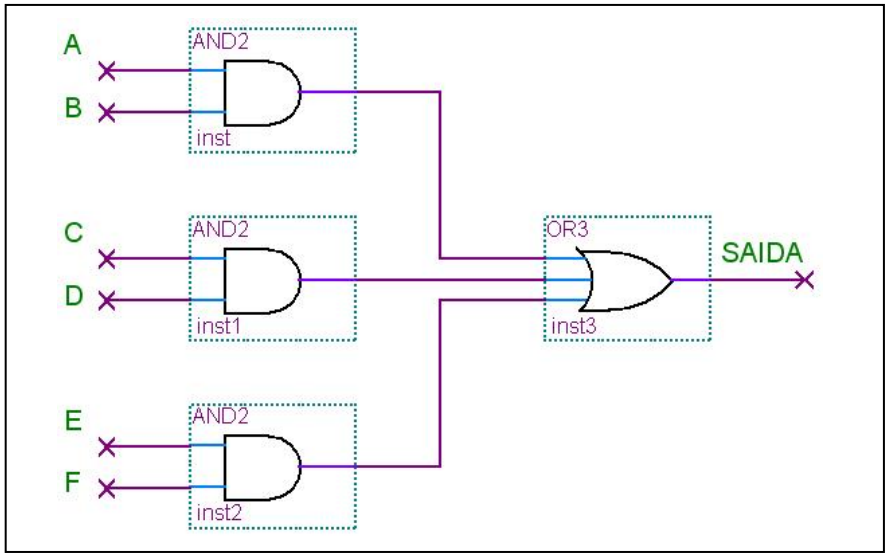


G4 Eletrônica Digital para Instrumentação

Prof. Márcio Portes de Albuquerque (mpa@cbpf.br)
Prof. Herman P. Lima Jr (hlima@cbpf.br)

Centro Brasileiro de Pesquisas Físicas
Ministério da Ciência e Tecnologia (MCT)

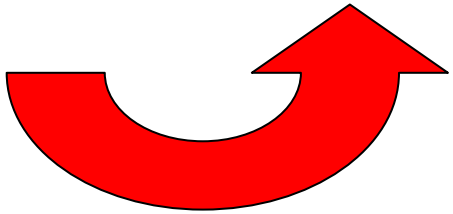
Paradigma




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

entity logica is
   port (A,B,C : in  std_logic;
         D,E,F : in  std_logic;
         SAIDA : out std_logic);
end logica;

architecture v_1 of logica is
begin
   SAIDA <= (A and B) or (C and D) or (E and F);
end v_1;
```



VHDL - Introdução

- Breve histórico
- Padrões mais importantes para síntese
- Objetos em VHDL
- Descrição Estrutural
- Descrição Funcional
- Interface (*entity*)
- Implementação (*architecture*)

VHDL - Introdução

- Pacotes
- Componentes
- Ações Simultâneas e Sequenciais
- Processos
- Palavras-chave
- Simulação

VHDL - Introdução

- Modernos sistemas eletrônicos são frequentemente complexos demais para descrição através de esquemáticos.
- No início da década de 80 surge a necessidade de outro método para descrever circuitos integrados muito complexos. O resultado é a criação das linguagens descritivas de hardware (HDL's).
- Linguagens mais utilizadas: **VHDL** e **Verilog**.
- Outras linguagens de mais alto nível de abstração já estão em uso para descrever sistemas digitais complexos (ex: SystemC [1] e SystemVerilog [2]).

[1] www.systemc.org
[2] www.eda.org/sv

VHDL - Introdução

- VHDL é uma linguagem para descrever sistemas eletrônicos digitais de média ou grande complexidade.
- A sigla significa: *VHSIC Hardware Description Language*, onde VHSIC significa *Very High Speed Integrated Circuit*.
- Criada a partir de um projeto do governo norte-americano, em razão da necessidade de uma linguagem padrão para descrever a estrutura e a funcionalidade de circuitos integrados muito complexos.
- Adotada e padronizada pelo Instituto dos Engenheiros Elétricos e Eletrônicos (IEEE).
- A linguagem VHDL foi evoluindo e hoje o padrão mais utilizado é o IEEE Std.1076-1993, juntamente com o IEEE Std.1164-1993, que define um sistema de valores lógicos.

VHDL - Introdução

- Características importantes:
  - Descrição estruturada, ou seja, um projeto é composto de sub-projetos e estes últimos são interconectados.
  - Especificação de funções utilizando formas similares de linguagens de programação.
  - Simulação de um projeto antes da fabricação de um circuito integrado (ASIC), ou da configuração de um dispositivo lógico programável (FPGA).
- Vantagens imediatas:
  - Melhor legibilidade de um projeto. Possibilidade de particionar um projeto mais facilmente, desacoplando seus blocos.
  - Utilização de parâmetros que modificam capacidade e performance de um projeto, ou bloco.
  - Redução do custo de fabricação de protótipos. Redução do tempo de inserção de um novo produto no mercado.
- VHDL $\Rightarrow$ Modelagem - Simulação - **Síntese**

VHDL - Introdução

Padrões mais importantes para síntese com VHDL

- **IEEE 1076-1993**

  Define a base (núcleo) da linguagem para modelagem, simulação e síntese.

- **IEEE 1076.6-1999**

  Define o sub-conjunto destinado somente à síntese (*Register Transfer Level* - RTL).

- **IEEE 1164-1993** (STD_LOGIC)

  Define um padrão (*standard package*) de 9 valores lógicos para sinais:

  ‘U’ → *Unitialized*
  ‘X’ → *Forcing Unknown*
  ‘0’ → *Forcing 0*
  ‘1’ → *Forcing 1*
  ‘Z’ → *High Impedance*
  ‘W’ → *Weak Unknown*
  ‘L’ → *Weak 0*
  ‘H’ → *Weak 1*
  ‘-’ → *Don´t Care*

- **IEEE 1076.3** (*Numeric Standard*)

  Define, principalmente, os tipos de dados aritméticos **signed** e **unsigned**, junto com suas respectivas operações aritméticas, de deslocamento e de conversão.

VHDL - Introdução

Objetos em VHDL

- Existem 3 classes de objetos: **sinais**, **constantes** e **variáveis**.
- O nome de um objeto pode utilizar qualquer caracter alfanumérico, desde que observadas as seguintes regras: (1) não pode ser uma palavra-chave de VHDL, (2) tem que iniciar com uma letra, (3) não pode terminar com *underscore* (_), e (4) não pode ter dois caracteres *underscore* juntos.
- Para síntese, os sinais (palavra-chave *signal*) são os mais importantes, pois representam os meios de comunicação entre blocos do projeto.
- Existem 3 locais onde um sinal pode ser declarado: na entidade, na parte de declarações de uma arquitetura, e na parte de declarações de um pacote.
- Declaração de um sinal: **signal** <nome_do_sinal> : [tipo] ;
- O tipo do sinal define os valores possíveis e sua utilização.

VHDL - Introdução

Tipos comuns de objetos em VHDL

- **bit** e **bit_vector**
  - definidos nos padrões IEEE 1076 e IEEE 1164
  - o tipo **bit** pode assumir valores ‘0’ ou ‘1’
  - o tipo **bit_vector** é simplemente um *array* linear de objetos **bit**
  - ex: `signal c: bit_vector (1 to 4);` `c(1) <= ‘1’;` `c <= “1010”;`

- **std_logic** e **std_logic_vector**
  - definidos no padrão IEEE 1164
  - para utilizá-los, tem-se que incluir as seguintes linhas de código:

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

  - oferece maior flexibilidade que os tipos bit, podendo assumir valores ‘0’, ‘1’, ‘Z’, ‘-’, ‘L’, ‘H’, ‘U’, ‘X’ ou ‘W’
  - os valores ‘0’, ‘1’, ‘Z’ e ‘X’ são os mais úteis para síntese

VHDL - Introdução

Tipos comuns de objetos em VHDL (cont.)

- **signed** e **unsigned**
  - definidos no padrão IEEE 1164, no pacote *std_logic_arith*;
  - este pacote também define a implementação dos operadores aritméticos (ex: +);
  - são similares ao tipo std_logic_vector, são *arrays* de std_logic;
  - têm como objetivo permitir a indicação no código de qual representação deve ser utilizada (sinalizada – complemento a 2 ou não-sinalizada);

- **integer**
  - definido para uso com operadores aritméticos (IEEE 1076);
  - o nº de bits não fica especificado no código, como um *std_logic_vector*;
  - por definição, um inteiro utiliza 32 bits, podendo assumir valores de $-(2^{31}-1)$ a $2^{31}-1$;
  - inteiros podem utilizar menos bits, utilizando-se a palavra-chave *range*, por ex.:

```
signal x : integer range -127 to 127;
```

VHDL - Introdução

Tipos comuns de objetos em VHDL (cont.)

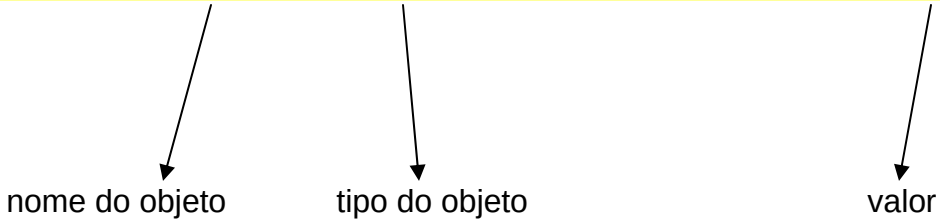
- **boolean**
  - pode assumir os valores lógicos TRUE ou FALSE, equivalentes a ‘1’ e ‘0’;
  - ex: **signal** flag : boolean ;

- tipo enumeração
  - tipo definido pelo projetista;
  - muito útil para definir estados na síntese de máquinas de estado;
  - ex:
```
type estados is (inicializa, processa);
signal y : estados ;
 ...
y <= processa;
```

Constantes em VHDL

constant

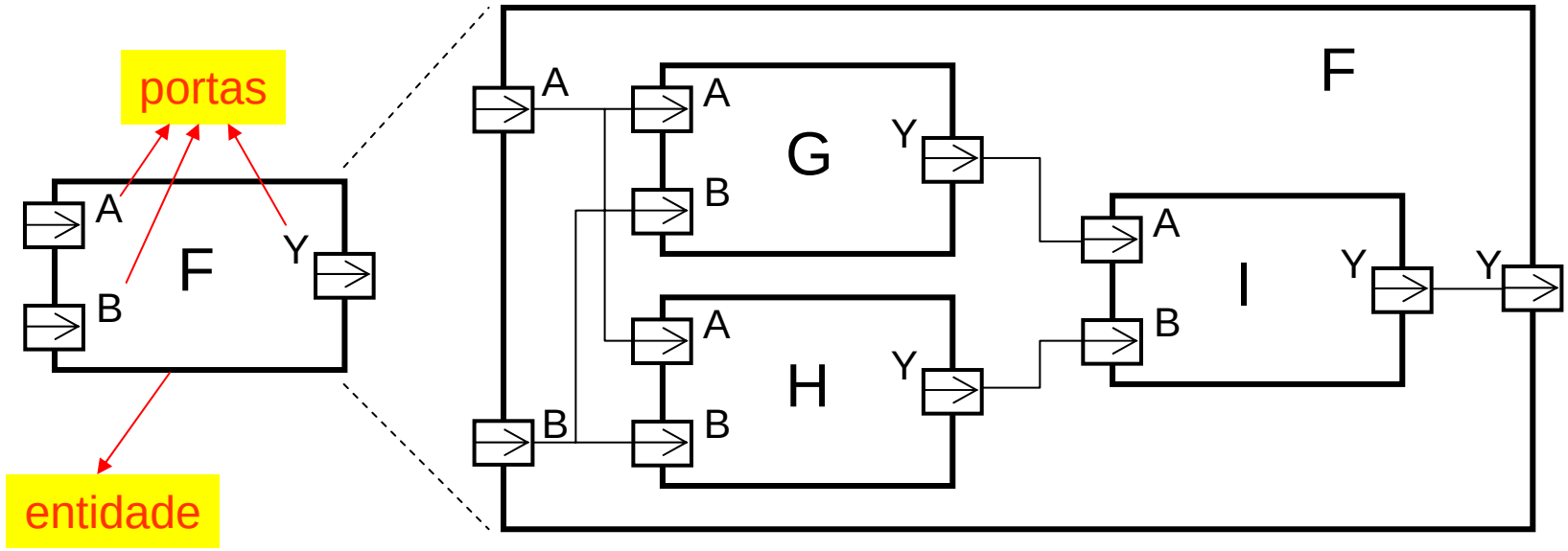
- um objeto ‘constante’ é não muda de valor em nenhuma parte do código;
- pode aparecer sem valor definido somente em pacotes (*packages*);
- muito útil para melhorar a legibilidade do código;
- ex: `constant z : std_logic_vector(2 downto 0) := “011”;`



VHDL - Introdução

Descrição ESTRUTURAL:

- Um sistema eletrônico pode ser descrito como um módulo com entradas e saídas.
- Os valores nas saídas são funções:
  - somente dos valores nas entradas em dado instante (circuito **combinacional**)
  - dos valores nas entradas e de estados internos (circuito **sequencial**)



VHDL - Introdução

Descrição FUNCIONAL:

- Um sistema eletrônico pode ser descrito simplesmente por sua função (ex: $Y = A \cdot \overline{B} + \overline{A} \cdot B$).
- Sistemas sequenciais obviamente não podem ser descritos unicamente como função de suas entradas.

Um projeto em VHDL pode ser baseado em componentes interconectados em hierarquia. Cada componente possui uma interface (entidade) e uma descrição funcional (arquitetura), seguindo a forma:

componente (projeto)

interface

```
entity <nome_ent> is
    port (a,b: in bit;
          y: out bit);
end <nome_ent>;
```

implementação

```
architecture <nome_arq> of <nome_ent> is
begin
    ---
    ---
end <nome_arq>;
```

VHDL - Introdução

Interface (entity):

- Define as portas de acesso ao componente.
- Forma genérica de uma entidade:

```
entity <nome_entidade> is
    port ( <nome_sinal>: [modo] [tipo];
           <nome_sinal>: [modo] [tipo] );
end <nome_entidade>;
```

- Possíveis modos de um sinal:
  - **in** ⇒ o sinal é uma entrada para a entidade.
  - **out** ⇒ o sinal é uma saída para a entidade. O valor do sinal não pode ser usado dentro da entidade. Sua posição é sempre à esquerda do operador de atribuição <=.
  - **inout** ⇒ o sinal pode ser entrada e/ou saída para a entidade.
  - **buffer** ⇒ o sinal é uma saída para a entidade, mas seu valor pode ser lido dentro da entidade. Ele pode estar à esquerda ou à direita do operador de atribuição <=.

VHDL - Introdução

Implementação (architecture):

- Define a implementação de uma entidade.
- Forma genérica de uma arquitetura:

```
architecture <nome_arquitetura> of <nome_entidade> is
    [declaração de sinais]
    [declaração de constantes]
    [declaração de tipos]
    [declaração de componentes]
    [especificação de atributos]
begin
    [instância de componente]
    [ação simultânea]
    [processo]
    [geração]
end <nome_arquitetura>;
```

VHDL - Introdução

Exemplo: comparador de 8 bits

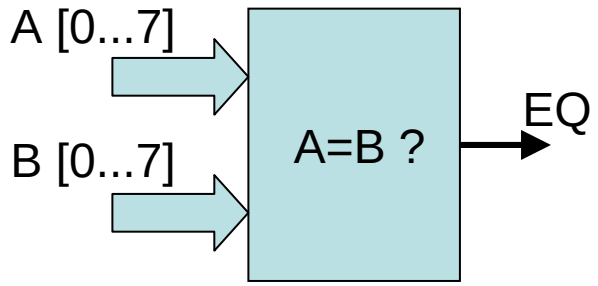
```
library ieee;

use ieee.std_logic_1164.all;

entity compare is
    port (A,B:   in std_logic_vector(0 to 7);
          EQ:   out std_logic);
end compare;

architecture one of compare is
begin
    EQ <= '1' when (A=B) else '0';
end one;
```

Representação esquemática:



- O circuito é **combinacional**.
- As palavras em destaque são palavras-chave em VHDL.
- VHDL não é *case-sensitive* (não diferencia letras maiúsculas de minúsculas).

VHDL - Introdução

PACOTES (package)

- **Entidades** e **Arquiteturas** são consideradas unidades básicas de um projeto.
- Unidades de projeto são segmentos de código que podem ser compilados separadamente e armazenados em uma biblioteca.
- Os 3 tipos de unidades mais úteis para síntese em VHDL são: entidades, arquiteturas e pacotes.
- Um pacote é designado pela palavra-chave *package*, e é utilizado para agrupar declarações de uso comum para diferentes unidades.
- Exemplo de um pacote:

```
package <nome_pacote> is

    constant NUM : integer := 16;

    type       MEM is array(0 to 31) of std_logic_vector(7 downto 0);

end <nome_pacote>;
```

PACOTES - exemplo

- Pacote:

```
package cpu_types is
    constant word_size : positive := 16;
    constant address_size : positive := 24;
    subtype addr is bit_vector(address_size-1 downto 0);
end cpu_types;
```

- Uso do pacote:

```
use work.cpu_types.all;
entity address_decoder is
    port ( addr : in work.cpu_types.addr);
end address_decoder;
architecture functional of address_decoder is
    constant mem_low : work.cpu_types.addr := x"000000";
begin
    ---
end functional.
```

VHDL - Introdução

COMPONENTES (component)

- Componentes são considerados sub-circuitos de um projeto.
- Somente através de componentes é possível projetar um sistema em hierarquia.
- Para um bloco ser utilizado como componente de outro projeto, o componente tem que ser declarado e instanciado.
- Declaração de um componente:

```
component <nome_comp> is
port (a,b: in bit;
          y: out bit);
end component;
```

- Instância de um componente:

```
chip1: <nome_comp>
port map (a => data(1), b => data(0); y => out);
```

VHDL - Introdução

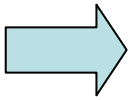
COMPONENTES – exemplo

Comparador de 8 bits

```
library ieee;
use ieee.std_logic_1164.all;
entity compare is
      port (A,B:   in std_logic_vector(0 to 7);
                EQ: out std_logic);
end compare;
architecture one of compare is
begin
      EQ <= '1' when (A=B) else '0';
end one;
```

```
component <nome_comp> is
port (a,b: in bit;
           y: out bit);
end component;
```

```
chip1: <nome_comp>
port map (a => data(1), b => data(0); y => out);
```



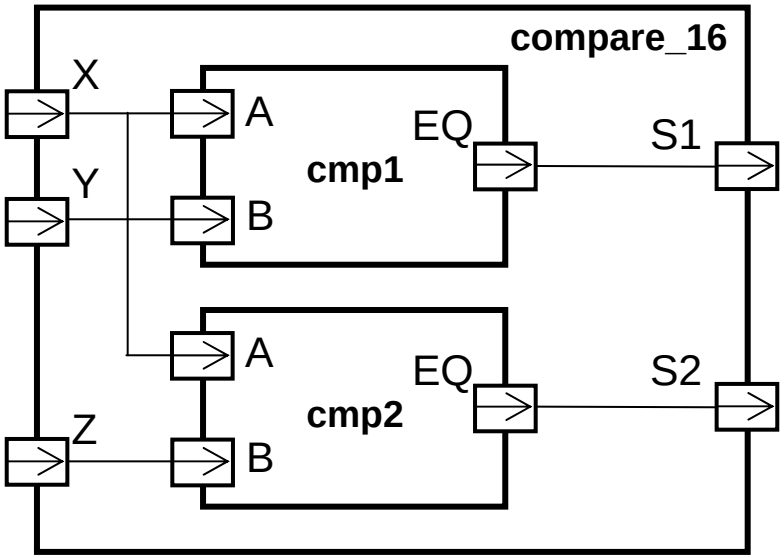
Comparador de 8 bits duplo

```
library ieee;
use ieee.std_logic_1164.all;
entity compare_16 is
      port (X,Y,Z:  in std_logic_vector(0 to 7);
               S1,S2: out std_logic);
end compare_16;
architecture arq of compare_16 is
   component compare is
   port (A,B: in std_logic_vector(0 to 7);
             EQ: out std_logic);
   end component;
begin
   cmp1: compare
   port map(A=>X, B=>Y, EQ=>S1);
   cmp2: compare
   port map(A=>X, B=>Z, EQ=>S2);
end arq;
```

COMPONENTES – exemplo (cont.)

Comparador de 8 bits duplo

```
library ieee;
use ieee.std_logic_1164.all;
entity compare_16 is
      port (X,Y,Z:  in std_logic_vector(0 to 7);
            S1,S2: out std_logic);
end compare_16;
architecture arq of compare_16 is
  component compare is
  port (A,B: in std_logic_vector(0 to 7);
        EQ: out std_logic);
  end component;
begin
  cmp1: compare
  port map(A=>X, B=>Y, EQ=>S1);
  cmp2: compare
  port map(A=>X, B=>Z, EQ=>S2);
end arq;
```



- O circuito é **combinacional**.
- O circuito é **assíncrono**.
- As saídas S1 e S2 mudam de estado devido a mudanças nas entradas, no instante em que estas ocorrem (na prática existem atrasos de propagação dos sinais).

VHDL - Introdução

AÇÕES SIMULTÂNEAS E SEQUENCIAIS

- Um projeto em VHDL pode conter ações simultâneas e sequenciais (*concurrent assignments* e *sequential assignments*).
- Ações simultâneas são implementadas dentro de uma arquitetura.
- Ações sequenciais são implementadas dentro de processos.
- Existem 4 tipos de ações simultâneas em VHDL:
  - atribuição simples de sinal, atribuição selecionada de sinal, atribuição condicional de sinal e declaração geradora.
- **Atribuição simples de sinal:**

```
<nome_sinal>  <=  <expressão>;
```

  - O símbolo **<=** é chamado operador de atribuição em VHDL.
  - Exemplos:

```
f <= (x1 AND x2) OR x2;
f <= 'Z';
```

VHDL - Introdução

AÇÕES SIMULTÂNEAS E SEQUENCIAIS (continuação)

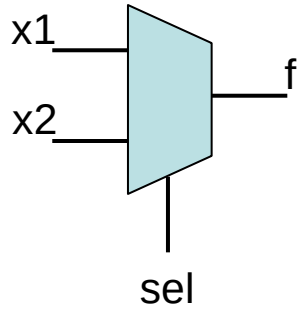
- **Atribuição selecionada de sinal:**

```
with <expressão> select
        <nome_sinal> <= <expressão> when <valor constante>{,
                     <= <expressão> when <valor constante>};
```

  - Exemplo:

```
signal x1,x2,sel,f : std_logic;
--
with sel select
        f <= x1 when '0',
             x2 when others;
```

→ multiplexador 2-1 →



- **Atribuição condicional de sinal:**

```
<nome_sinal> <= <expressão> when <valor constante> else
                {<expressão> when <valor constante>}
                 <expressão>;
```

  - Ao contrário da atribuição selecionada, na atribuição condicional as condições não precisam ser mutuamente exclusivas, pois são testadas em ordem de prioridade.

VHDL - Introdução

AÇÕES SIMULTÂNEAS E SEQUENCIAIS (continuação)

- **Declaração geradora:** permite a repetição de uma função lógica ou de instância de componente. Pode ser usada da forma FOR ou IF.

```
for <variável índice> in <faixa> generate
        ação;
        {ação;}
end generate;
```

```
if <expressão> generate
        ação;
        {ação;}
end generate;
```

  – Exemplo:

```
for i in 0 to 3 generate
        bit: fulladd port map ( C(i), X(i), Y(i), S(i), C(i+1) );
end generate;
```

O código acima gera 4 instâncias de um componente chamado *fulladd*.

VHDL - Introdução

PROCESSOS

- Processos são utilizados para a implementação de ações sequenciais.
- Um processo tem a forma genérica:

```
{<nome_processo> :}
process (<nome_sinal>{,<nome_sinal>})
    [declaração de variáveis]
begin
    [estruturas WAIT]
    [atribuição de sinais]
    [atribuição de variáveis]
    [estruturas IF]
    [estruturas CASE]
    [estruturas LOOP]
end process [<nome_processo>];
```

lista de suscetibilidade:
⇒ as ações dentro do processo são executadas SOMENTE se um (ou mais) dos sinais da lista muda de estado.

• As ações são executadas na sequência em que ocorrem no código.
• Os sinais e variáveis somente mudarão de estado no final da execução do processo.

VHDL - Introdução

PROCESSOS (continuação)

Estrutura condicional IF:

```
if <expressão> then
    ação;
    {ação;}
elsif <expressão> then
    ação;
    {ação;}
else
    ação;
    {ação;}
end if;
```

Exemplo: multiplexador 2-1

```
if sel = '0' then
    f <= x1;
else
    f <= x2;
end if;
```

Estrutura CASE:

```
case <expressão> is
    when <valor constante> =>
    ação;
    {ação;}
    when <valor constante> =>
    ação;
    {ação;}
    when others =>
    ação;
    {ação;}
end case;
```

Exemplo: multiplexador 2-1

```
case sel is
    when '0' =>
        f <= x1;
    when '1' =>
        f <= x2;
end case;
```

VHDL - Introdução

PROCESSOS (continuação)

Forma comum de um processo gerando ação síncrona:

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

entity top is
   port (clk, rst: in std_logic;
         data: in std_logic_vector(7 downto 0);
         q: out std_logic_vector(7 downto 0));
end top;

architecture top_arch of top is
begin
   reg:                                          <-- opcional
   process (rst,clk)
      variable qreg: std_logic_vector(7 downto 0);
   begin
      if rst = '1' then                          -- reset assíncrono
         qreg := "00000000";
      elsif (clk = '1' and clk'event) then
         qreg := data;                           -- ação síncrona
      end if;
      q <= qreg;
   end process;
end top_arch;
```

Representação esquemática:

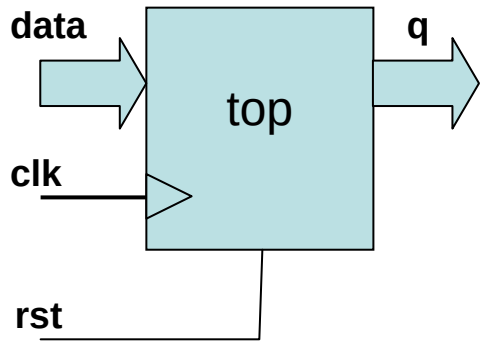
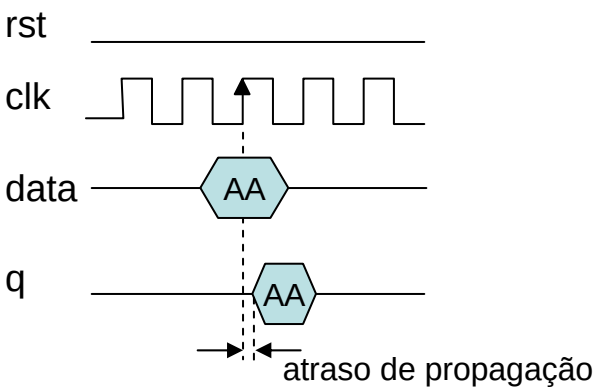


Diagrama temporal:



VHDL - Introdução

PALAVRAS-CHAVE (IEEE Std.1076-1993)

abs	disconnect	label	package	sla
access	downto	library	port	sll
after	else	linkage	postponed	sra
alias	elsif	literal	procedure	srl
all	end	loop	process	subtype
and	entity	map	protected	then
architecture	exit	mod	pure	to
array	file	nand	range	transport
assert	for	new	record	type
attribute	function	next	register	unaffected
begin	generate	nor	reject	units
block	generic	not	rem	until
body	group	null	report	use
buffer	guarded	of	return	variable
bus	if	on	rol	wait
case	impure	open	ror	when
component	in	or	select	while
configuration	inertial	others	severity	with
constant	inout	out	shared	xnor
	is		signal	xor

VHDL - Introdução

Simulação

- Modos mais comuns:
  - Test Bench
  - Simuladores visuais

- Test Bench:
  - É um projeto a parte utilizado somente para simular um projeto.
  - É formado por uma entidade e uma arquitetura.
  - A entidade de um Test Bench não possui portas.
  - Geralmente contém 2 processos: um processo gerador de *clock* e um processo que gera os outros estímulos.

VHDL - Introdução

Simulação

Test Bench para o registrador de 8 bits.

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

entity test is
end test;

architecture one of test is
   component top is
     port (clk, rst: in std_logic;
           data: in std_logic_vector(7 downto 0);
           q: out std_logic_vector(7 downto 0));
   end component;
   signal clk, rst: std_logic;
   signal data, q: std_logic_vector(7 downto 0);
begin
   DUT: <projeto> port map (clk, rst, data, q);
```

```
   clock: process
      variable clktmp: std_logic := '1';
   begin
      clktmp := not(clktmp);
      clk <= clktmp;
      wait for 25 ns;
   end process;
   stimulus: process
   begin
      rst <= '0';
      data <= "10101010";
      wait for 100 ns;
      data <= "01010101";
      wait for 200 ns;
   end process;
end one;
```

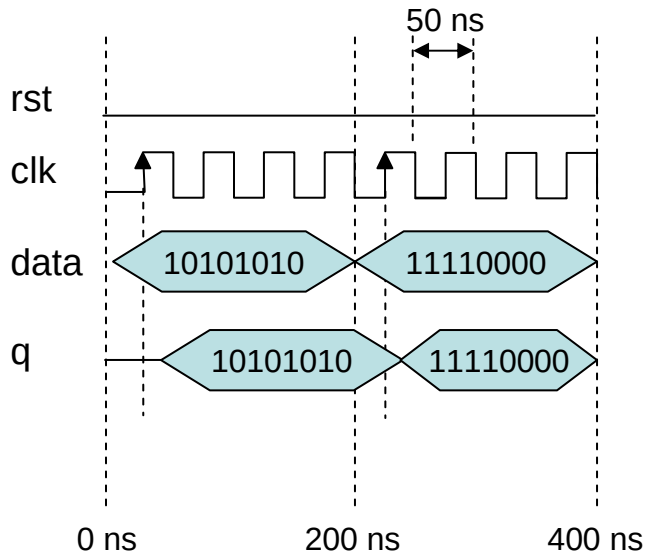
VHDL - Introdução

Simulação

Exemplo de Test Bench: testando o registrador de 8 bits.

```
  clock: process
    variable clktmp: std_logic := '1';
  begin
    clktmp := not(clktmp);
    clk <= clktmp;
    wait for 25 ns;
  end process;
  stimulus: process
  begin
    rst <= '0';
    data <= "10101010";
    wait for 200 ns;
    data <= "11110000";
    wait for 200 ns;
  end process;
end behavior;
```

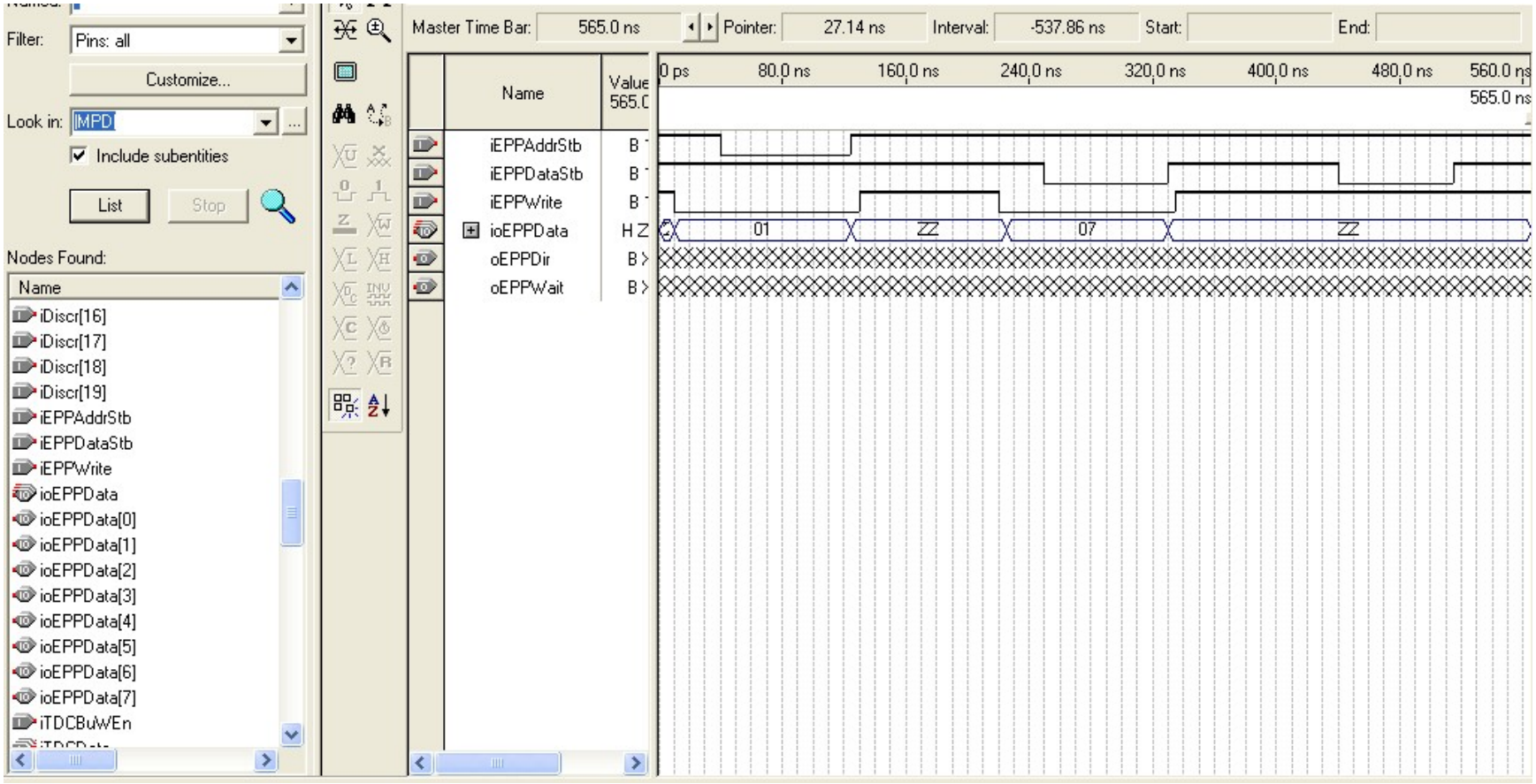
Resultado da simulação:



VHDL - Introdução

Simulação

Exemplo de Simulador Visual (Quartus II):



Referências

- **Fundamentals of Digital Logic with VHDL Design**, *Stephen Brown, Zvonko Vranesic,* McGraw-Hill, 2000.
- **The Designer's Guide to VHDL**, *Peter Ashenden*, 2nd Edition, Morgan Kaufmann, 2002.
- **VHDL Coding Styles and Methodologies**, *Ben Cohen*, 2nd Edition, Kluwer Academic Publishers, 1999.
- **Digital Systems Design with VHDL and Synthesis: An Integrated Approach**, *K. C. Chang*, Wiley-IEEE Computer Society Press, 1999.
- **Application-Specific Integrated Circuits**, *Michael Smith*, Addison-Wesley, 1997.

- **www.altera.com** (*datasheets, application notes, reference designs*)
- **www.xilinx.com** (*datasheets, application notes, reference designs*)
- **www.doulos.com/knowhow/vhdl_designers_guide** (*The Designer's Guide to VHDL*)
- **www.acc-eda.com/vhdlref/index.html** (*VHDL Language Guide*)
- **www.vhdl.org**