

VHDL: Processos

Engenharia Eletrônica

Prof. Renan Augusto Starke

Instituto Federal de Santa Catarina – IFSC
Campus Florianópolis
renan.starke@ifsc.edu.br

13 de março de 2023



INSTITUTO FEDERAL
SANTA CATARINA

Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
INSTITUTO FEDERAL DE SANTA CATARINA

1 Introdução

2 Processos

3 Variáveis

4 Instruções sequenciais

5 Referências

- ▶ Tópicos da aula de hoje:

- Processos

- ▶ Processos implícitos:
 - Atribuição de sinais
 - Instâncias de componentes
 - São sensíveis a todas as entradas

- ▶ Processos explícitos:
 - Utiliza-se a palavra **PROCESS**

1 Introdução

2 **Processos**

3 Variáveis

4 Instruções sequenciais

5 Referências

Exemplo

```
-- processo: declaração
label : PROCESS (sensitivity_list)
-- Declaração de constante
-- Declaração de tipos
-- Declaração de variáveis

-- corpo
BEGIN
-- Instrução sequencial #1;
...
-- Instrução sequencial #n ;
END PROCESS;
```

- ▶ Processos são sensíveis a uma lista opcional de sensibilidade **explícita**:
 - Transições de sinais desta lista acionam o processo:
 - $0 \rightarrow 1$, $X \rightarrow 1$, ...
- ▶ Secção de declaração permite declarar objetos e nomes locais
- ▶ Conteúdo do processo consiste em instruções sequencias e atribuições simples de sinais

Exemplo

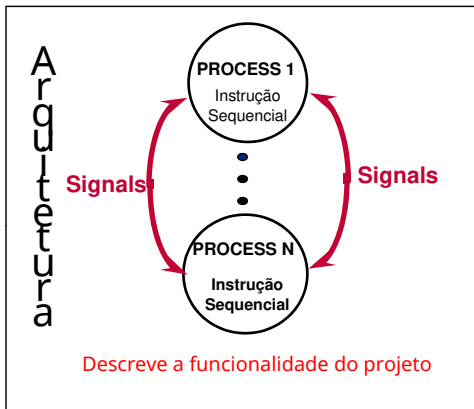
```
process (a,b)
begin
  -- instruções sequenciais
end process;
```

```
process
begin
  -- instruções sequenciais
  wait on (a,b);
end process;
```

- ▶ Processos são executados indefinidamente até uma cláusula **WAIT** ou pela lista de sensibilidade

- ▶ A lista de sensibilidade implica em uma cláusula **WAIT** no final do processo
- ▶ Processos podem ter várias cláusula **WAIT**
- ▶ Não se pode haver uma lista de sensibilidade e **WAIT**
- ▶ **Síntese impõe restrições no uso de WAIT bem como no uso da lista de sensibilidade**

Arquiteturas multi-processos



- ▶ Uma arquitetura pode ter vários processos
- ▶ Todos os processos executam concorrentemente
 - Ordem dos processos não importa
- ▶ Instruções internas do processos são executadas sequencialmente
 - Ordem das instruções **importa**

Atribuição de sinais – **delay**

- ▶ Atribuição de sinais pode ser postergada usando-se a instrução **AFTER**
- ▶ Dois tipos de atrasos:
 - Atraso inercial:
 - ▶ Escalona a alteração da saída após a passagem do atraso, a menos que a entrada mude novamente
 - ▶ A entrada deve permanecer estável até o atraso expirar para atualizar a saída

```
a <= b after 10 ns;  
a <= inertial b after 10 ns;
```

- Atraso de transporte:
 - ▶ Sempre escalona a alteração da saída depois da passagem do atraso
 - ▶ Qualquer transição na entrada é transmitida a saída

```
c <= transport d after 10 ns;
```

Atribuição de sinais – delay

► Exemplo dos delays

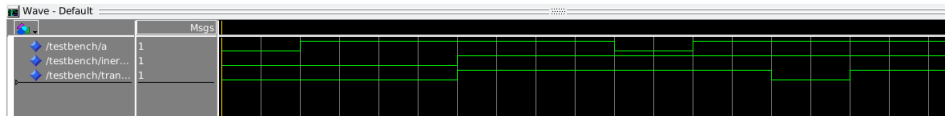
```
-- geração dos sinais          -- atraso inercial          -- atraso transporte
process
begin
  a <= '0';
  wait for 10 ns;
  a <= '1';
  wait for 40 ns;
  a <= '0';
  wait for 10 ns;
  a <= '1';
  wait;
end process;

process(a)
begin
  inercial <= a after 20 ns;
end process;

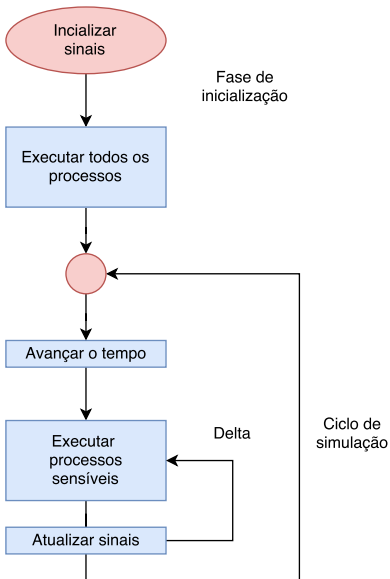
-- Similiar ao atraso de uma
-- porta analógica
--

process(a)
begin
  transporte <= transport
    a after 20 ns;
end process;

-- Usado para modelar
-- atrasos de fios ou
-- conexões na PCB
```



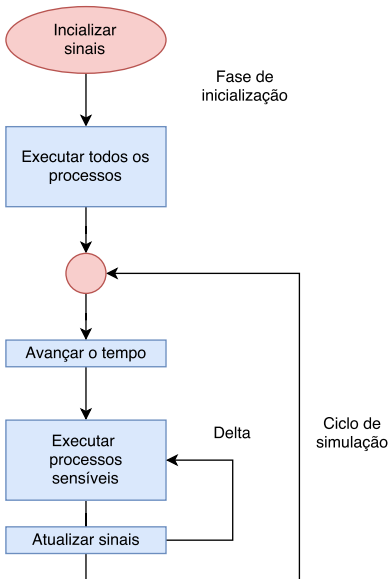
Estimando o comportamento do modelo



► Ciclo de simulação

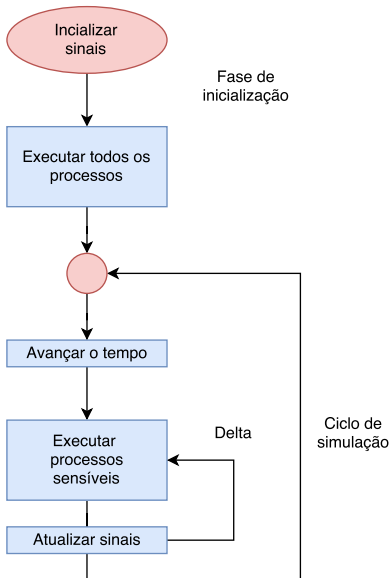
- **Wall clock time:** tempo total de execução
- Delta:
 - Fase de execução do processo
 - Fase de atualização dos sinais

Estimando o comportamento do modelo



- ▶ Quando um ciclo delta termina:
 - Quando todos os processos executam
 - ▶ Final do processo (com lista de sensibilidade)
 - ▶ Quando um processo encontra um **WAIT**
 - Depois disto, todos os sinais escritos neste ciclo são atualizados

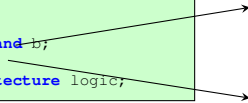
Estimando o comportamento do modelo



- ▶ Quando um ciclo de simulação termina:
 - Quando a atualização de todos os sinais de um ciclo delta não causa um novo ciclo (convergência)
 - Processos não são acionados com mudanças de sinais
- ▶ Sinais são atualizados **SOMENTE** no final do ciclo delta

Processos: equivalência

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity simp is  
port (  
    a, b : in std_logic;  
    y : out std_logic );  
end entity simp;  
  
architecture logic of simp is  
    signal c : std_logic;  
  
begin  
    c <= a and b;  
    y <= c;  
end architecture logic;
```



```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity simp is  
port (  
    a, b : in std_logic;  
    y : out std_logic );  
end entity simp;  
  
architecture logic of simp is  
    signal c : std_logic;  
begin  
  
    process1: process (a,b)  
begin  
    c <= a and b;  
end process process1;  
  
    process2: process (c)  
begin  
    y <= c;  
end process process2;  
  
end architecture logic;
```

Equivalentes

Processos: equivalência

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

entity simp is
port (
  a, b : in std_logic;
  y : out std_logic );
end entity simp;

architecture logic of simp is
signal c : std_logic;

begin
  c <= a and b;
  y <= c;
end architecture logic;
```

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

entity simp is
port (
  a, b : in std_logic;
  y : out std_logic );
end entity simp;

architecture logic of simp is
signal c : std_logic;
begin

process1: process (a,b,c)
begin
  c <= a and b;
  y <= c;
end process process1;

end architecture logic;
```

NÃO Equivalentes

1 Introdução

2 Processos

3 Variáveis

4 Instruções sequenciais

5 Referências

Declaração de variáveis

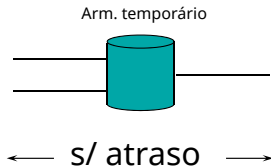
- ▶ Variáveis são declaradas dentro de um **processo**

- ▶ Declaração:

```
variable <nome> : <TIPO> := <VALOR>
```

```
variable temp : std_logic_vector(7 downto 0);
```

- ▶ Variáveis são atribuídas imediatamente
 - Não há atrasos
 - Representam um armazenamento temporário



Declaração de variáveis

```
variable temp : std_logic_vector(7 downto 0);
```

- ▶ Usa-se := na atribuição de variáveis

- ▶ Exemplos:

- Todos os bits:

```
temp := "10101010";  
temp := x"aa"; -- 'o' para octal e 'b' para binário
```

- Parcial:

```
temp(7 downto 4) := "1010";
```

- Bit:

```
temp(7) := '1';
```

- ▶ Use aspas normais (" ") para atribuição multi-bit e aspas simples (' ') para bit único

Atribuição de variáveis

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity var is  
port (  
    a, b : in std_logic;  
    y : out std_logic  
);  
end entity var;  
  
architecture logic of var is  
begin  
    process (a, b)  
        variable c : std_logic;  
    begin  
        c := a and b;  
        y <= c;  
    end process;  
end architecture logic;
```

Variável c é atualizada imediatamente e um novo valor é atribuído ao sinal y

Declaração de variáveis

Atribuição

Variável é atribuída a um sinal para sintetizar “algum” hardware

Sinais e variáveis

	Sinais (\leq)	Variáveis ($:=$)
Utilidade	Representa interconexão	Representa armazenamento local
Escopo	Dentro Arquitetura (comm. entre processos)	Local (Dentro do processo)
Comportamento	Atualizado no final de um ciclo (novo valor não disponível imediatamente)	Atualizado imediatamente

- 1 Introdução
- 2 Processos
- 3 Variáveis
- 4 Instruções sequenciais**
- 5 Referências

Instruções sequenciais

- ▶ Indicam comportamento e expressam ordem
- ▶ **Apenas** disponíveis dentro de um processo **explícito**
- ▶ Instruções sequenciais:
 - IF-THEN
 - CASE
 - LOOP
 - WAIT

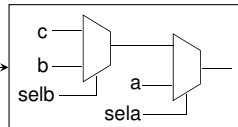
Instrução IF-THEN

► Formato:

```
if <condicao_11> then
{sequencia de comandos}
elsif <condition2> then
{sequencia de comandos}
...
else
{sequencia de comandos}
end if;
```

► Exemplo:

```
process (sela, selb, a, b, c)
begin
  if sela = '1' then
    q <= a;
  elsif selb = '1' then
    q <= b;
  else
    q <= c;
  end if;
end process;
```



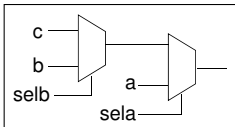
- ▶ Condições são avaliadas de cima para baixo.
 - Priorização.
- ▶ A primeira condição que for verdadeira causará a execução das instruções abaixo.
- ▶ Se todas são falsas, cláusula **ELSE** executará.

Atribuição condicional

► Similar a atribuição condicional:

Processo implícito

```
q <= a when sela = '1' else  
b when selb = '1' else  
c;
```



Processo explícito

```
process (sela, selb, a, b, c)  
begin  
    if sela = '1' then  
        q <= a;  
    elsif selb = '1' then  
        q <= b;  
    else  
        q <= c;  
    end if;  
end process;
```

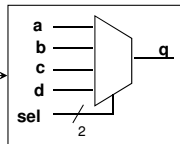
Instrução CASE

► Formato:

```
CASE {expressao} IS
WHEN <condicao_1> =>
{instrucoes sequenciais}
WHEN <condicao_1> =>
{instrucoes sequenciais}
...
WHEN OTHERS => -- (opcional)
{instrucoes sequenciais}
END CASE;
```

► Exemplo:

```
process (sel, a, b, c, d)
begin
  case sel is
    when "00" =>
      q <= a;
    when "01" =>
      q <= b;
    when "10" =>
      q <= c;
    when others =>
      q <= d;
  end case;
end process;
```



- ▶ Condições são avaliadas apenas uma vez
 - Não há priorização
- ▶ Todas as possíveis condições **devem** ser consideradas
- ▶ **WHEN OTHERS** considera todas as condições que não foram especificadas anteriormente

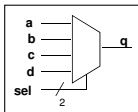
- Similar a atribuição selecionável:

Processo implícito

```
with sel select
  q <= a when "00",
  b when "01",
  c when "10",
  d when others;
```

Processo explícito

```
process (sel, a, b, c, d)
begin
  case sel is
    when "00" => q <= a;
    when "01" => q <= b;
    when "10" => q <= c;
    when others => q <= d;
  end case;
end process;
```



Laços sequenciais

- ▶ Laços infinitos
 - Iteram eternamente
- ▶ Laços **WHILE**
 - Iteram enquanto uma condição for verdadeira
- ▶ Laços **FOR**
 - Iteram uma quantidade de iterações
 - Nota: identificador de iteração não precisa ser previamente declarado
- ▶ Instruções adicionais:
 - **NEXT**: reinicia laço
 - **EXIT**: terminar laço

▶ Laço infinito:

```
[label]: LOOP
    -- instrução sequencial
EXIT loop_label;
END LOOP;
```

▶ WHILE

```
WHILE <condicao> LOOP
    -- Instruções sequenciais
END LOOP;
```

▶ FOR

```
FOR <identificador> IN <limiterange> LOOP
    -- Instruções sequenciais
END LOOP;
```

Contador de 1 usando for

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

entity ones_count is
port (
  invec : in std_logic_vector (31 downto 0);
  outvec : out std_logic_vector (7 downto 0)
);
end entity ones_count;

architecture rtl of ones_count is
begin

process (invec)
  variable count: std_logic_vector (7 downto 0);
begin
  count:= (others => '0');

  for i in invec'range loop
    if (invec(i) /= '0') then
      out:=count+1;
    end if;
  end loop;

  outvec <= count;

end process;
end architecture rtl;
```

Instrução **WAIT**

- ▶ Interrompe a execução do processo até a cláusula **WAIT** estar satisfeita

- ▶ Tipos:

- **WAIT ON <sinal>**

- ▶ Espera um evento acontecer no sinal

```
wait on a,b;
```

- **WAIT UNTIL <expressão>:**

- ▶ Espera até a expressão ser verdadeira

```
wait until (int < 100);
```

- **WAIT FOR <tempo>:**

- ▶ Espera até o tempo expirar

```
wait for 20 ns;
```

- **WAIT** combinado:

```
wait until (a = '1') for 5 us;
```

Exemplo

```
stim: process
  variable error : boolean;
begin

  wait until clk = '0';
  a <= (others => '0');
  b <= (others => '0');

  wait for 40 ns;

  if (sum /= 0) then
    error := true;
  end if;

  wait until clk = '0';

  a <= "0010";
  b <= "0011";

  wait for 40 ns;

  if (sum /= 5) then
    error := true;
  end if;

  ...

  wait;
end process stim;
```


- ▶ Implemente e teste o componente exemplo de contador de 1's;
- ▶ Usando laços, implemente um componente para geração de paridade.
 - Entradas: sinal com 8-bits, enable.
 - Saída: um bit (paridade par ou ímpar).
 - A contagem só deve ser executada quando enable = '1'.
 - Use uma xor.

- 1 Introdução
- 2 Processos
- 3 Variáveis
- 4 Instruções sequenciais
- 5 Referências**

- ▶ *Introduction to Altera Devices and Design Software*
- ▶ D'AMORE, Roberto. VHDL: descrição e síntese de circuitos digitais. 2. ed. Rio de Janeiro: Livros Técnicos e Científicos, 2012. 292 p., il. ISBN 9788521620549.
- ▶ PEDRONI, Volnei A. Eletrônica digital moderna e VHDL. Rio de Janeiro: Elsevier, 2010. 619 p., ISBN 9788535234657.