

# Laboratório de Sistemas Digitais

## Aula Teórico-Prática 6

Ano Letivo 2021/22

Construção e utilização de  
*testbenches* para simulação em VHDL

Princípios básicos de simulação

# Conteúdo

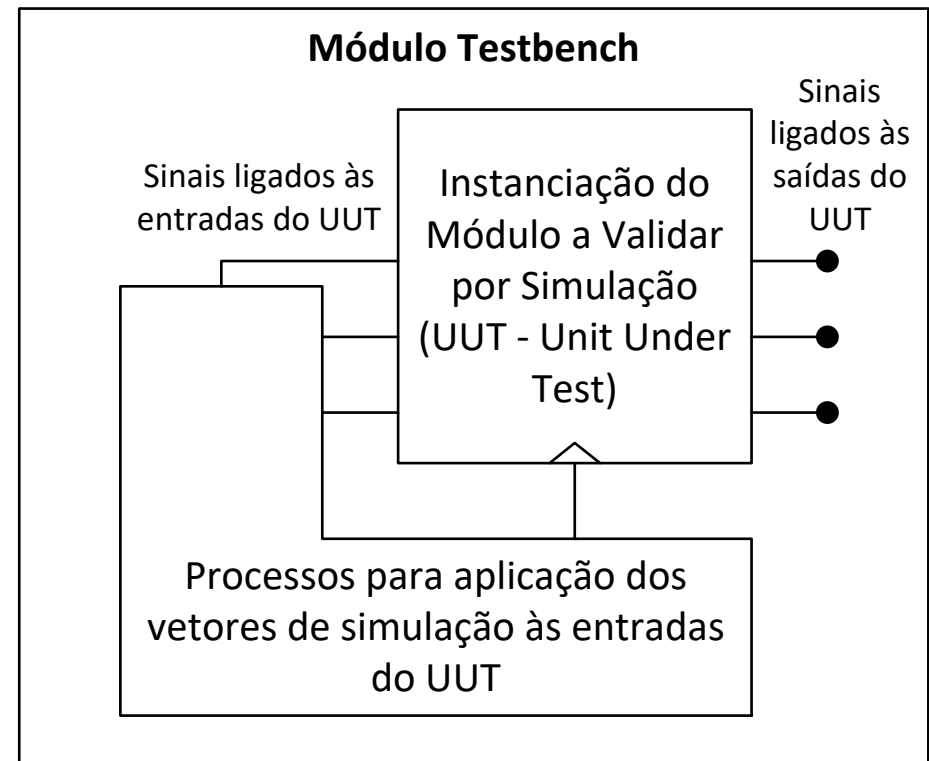
- Simulação de modelos em VHDL
  - Utilidade da simulação
  - Motivação para a utilização de *testbenches*
  - Construção de *testbenches* para simulação de componentes
    - Combinatórios
    - Sequenciais
- Tópicos fundamentais sobre simulação e síntese em VHDL
  - (Mais detalhes sobre as) construções para modelação de paralelismo
  - Conceitos sobre o funcionamento do simulador
    - Relação com a semântica dos sinais em VHDL
  - Processos e listas de sensibilidade
    - Regras fundamentais e boas práticas

# Simulação com HDLs (e.g. VHDL)

- Fundamental para validar o modelo de um sistema desde as fases iniciais de projeto até à implementação
  - Económica
  - Muito controlável
- Útil para observar qualquer ponto do sistema
  - Por vezes inacessível na implementação em hardware

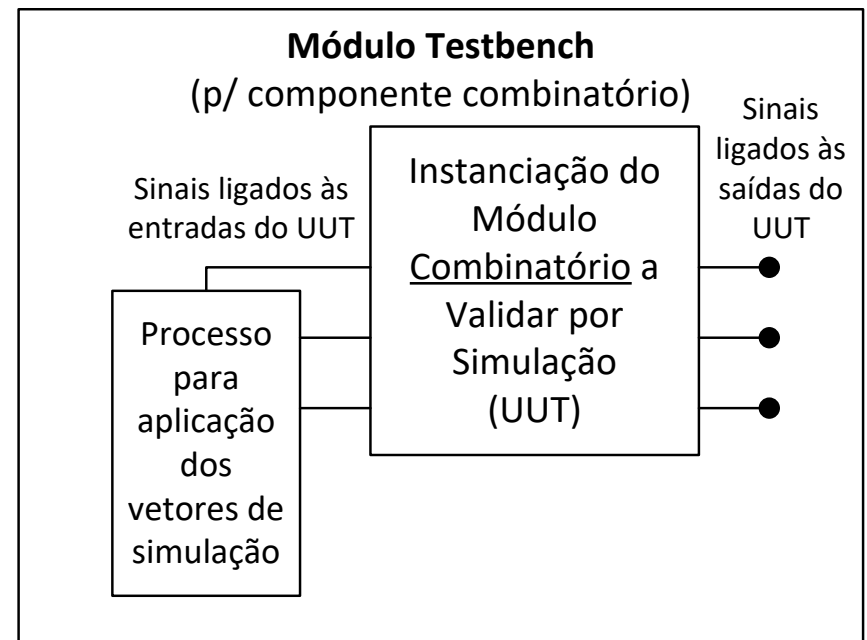
# Simulação em VHDL

- Baseada em *testbenches*
  - Módulo onde o modelo VHDL a simular (Unit Under Test) é instanciado e onde são aplicados estímulos (vetores de simulação) para validar o comportamento
- Uma *testbench*
  - Atua como *top level* no simulador
  - Pode ser construída de forma
    - Gráfica (e.g. através do ficheiro VWF e aplicação com GUI) – “amarradas” a uma ferramenta específica
    - Textual (como um ficheiro VHDL – com uma estrutura específica) – portáveis / independentes da ferramenta



# Estrutura Típica de uma Testbench para um Componente Combinatório

- Entidade sem portos
- Arquitetura
  - Instanciação da UUT no corpo da arquitetura
  - Declaração dos sinais a ligar aos portos da UUT na parte declarativa da arquitetura
  - Definição de um processo para aplicar os vetores de simulação ao longo do tempo
    - Em sistemas mais complexos pode ser usado mais do que um processo para este efeito

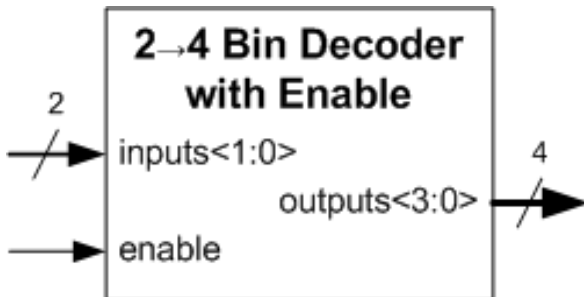


# Exemplo de um Componente Combinatório

## Módulo a simular: decodificador 2->4

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

entity Dec2_4En is
    port(enable : in std_logic;
          inputs : in std_logic_vector(1 downto 0);
          outputs : out std_logic_vector(3 downto 0));
end Dec2_4En;
```



```
architecture Behavioral of Dec2_4En is
begin
    process(enable, inputs)
    begin
        if (enable = '0') then
            outputs <= "0000";
        else
            if (inputs = "00") then
                outputs <= "0001";
            elsif (inputs = "01") then
                outputs <= "0010";
            elsif (inputs = "10") then
                outputs <= "0100";
            else
                outputs <= "1000";
            end if;
        end if;
    end process;
end Behavioral;
```

# Exemplo de uma *Testbench* para um Componente Combinatório

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

-- Entidade sem portos
entity Dec2_4EnTb is
end Dec2_4EnTb;

architecture Stimulus of Dec2_4EnTb is
    -- Sinais para ligar às entradas da uut
    signal s_enable : std_logic;
    signal s_inputs : std_logic_vector(1 downto 0);
    -- Sinal para ligar às saídas da uut
    signal s_outputs : std_logic_vector(3 downto 0);
begin
    -- Instanciação da Unit Under Test (UUT)
    uut: entity work.Dec2_4En(Behavioral)
        port map(enable => s_enable,
                  inputs => s_inputs,
                  outputs => s_outputs);
```

```
--Process stim
stim_proc : process
begin
    wait for 100 ns;

    s_enable <= '0';
    wait for 100 ns;

    s_enable <= '1';
    wait for 100 ns;

    s_inputs <= "00";
    wait for 100 ns;

    s_inputs <= "10";
    wait for 100 ns;

    s_inputs <= "01";
    wait for 100 ns;

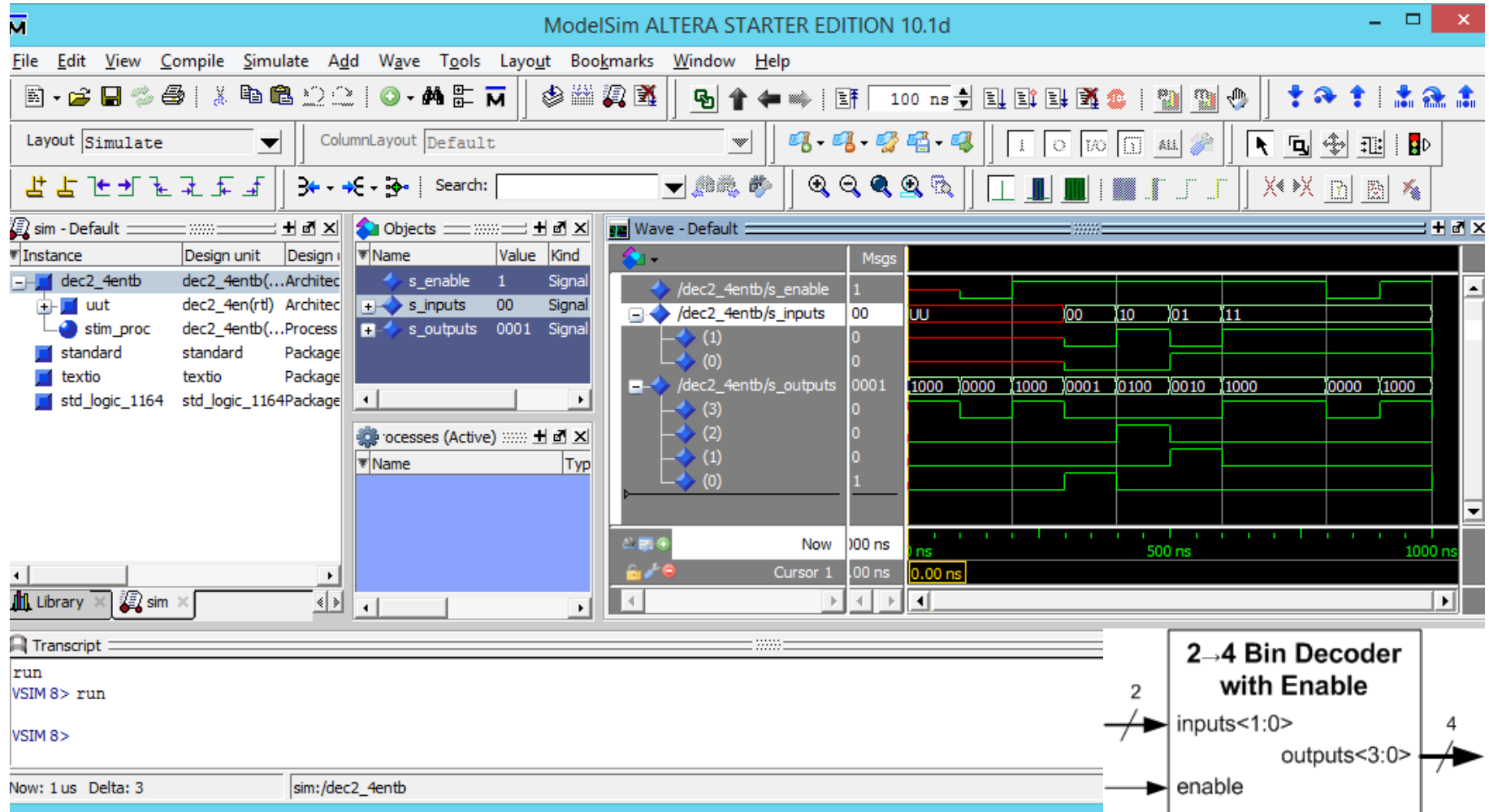
    s_inputs <= "11";
    wait for 100 ns;

end process;
end Stimulus;
```

Construção "wait for..." suportada apenas para simulação!



# Simulação c/ a Testbench Dec2\_4EnTb

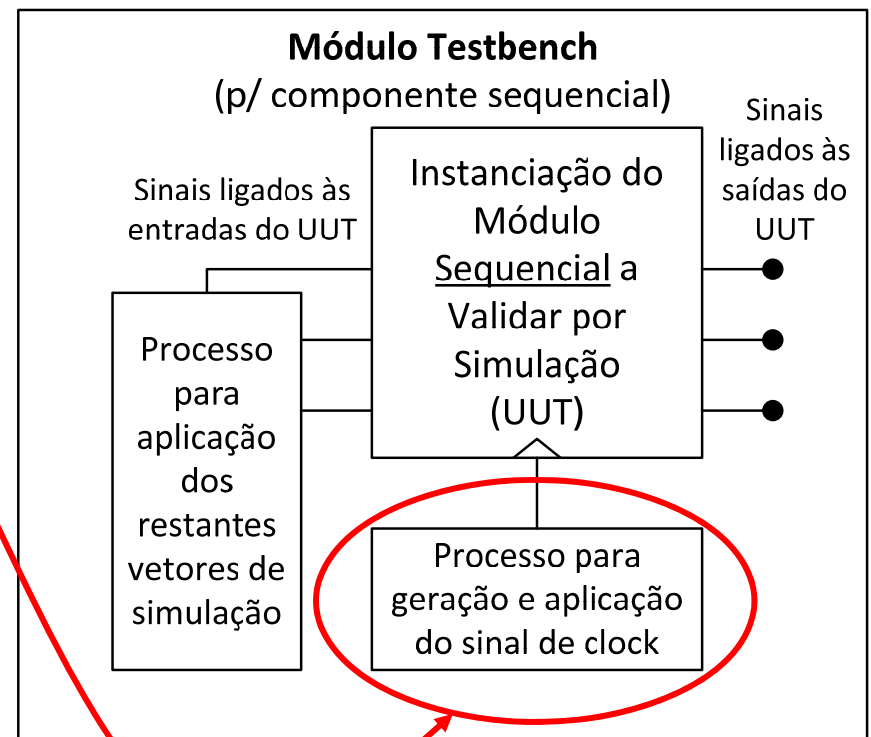


Nota: os passos de compilação e simulação serão abordados no guião prático 7.



# Estrutura de uma *Testbench* para um Componente Sequencial (com *clock*)

- Entidade sem portos
- Arquitetura
  - Instanciação da UUT no corpo da arquitetura
  - Declaração dos sinais a ligar aos portos da UUT na parte declarativa da arquitetura
  - Definição de um processo para geração do sinal de *clock*
  - Definição de um processo para aplicar os vetores de simulação ao longo do tempo
    - Em sistemas mais complexos pode ser usado mais do que um processo para este efeito

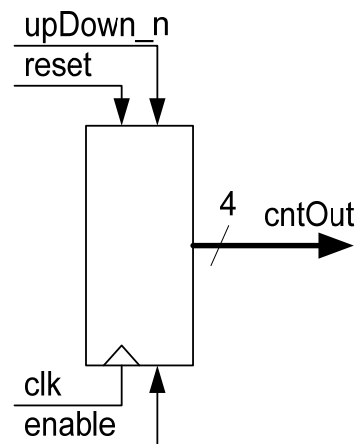


# Exemplo de um Componente Sequencial

**Módulo a simular: contador up/down de 4 bits**

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

entity BinUDCntEnRst4 is
port(reset : in  std_logic;
     clk   : in  std_logic;
     enable : in  std_logic;
     upDown_n : in  std_logic;
     cntOut  : out std_logic_vector(3 downto 0));
end BinUDCntEnRst4;
```



```
architecture Behavioral of BinUDCntEnRst4 is
    signal s_cntValue : unsigned(3 downto 0);
begin
    process(clk)
    begin
        if (rising_edge(clk)) then
            if (reset = '1') then
                s_cntValue <= (others => '0');
            elsif (enable = '1') then
                if (upDown_n = '0') then
                    s_cntValue <= s_cntValue - 1;
                else
                    s_cntValue <= s_cntValue + 1;
                end if;
            end if;
        end if;

        cntOut <= std_logic_vector(s_cntValue);
    end process;
end Behavioral;
```

# Ex. de uma *Testbench* para um Comp.

## Sequencial

```
-- Entidade sem portos
entity BinUDCntEnRst8Tb is
end BinUDCntEnRst8Tb;

architecture Stimulus of BinUDCntEnRst8Tb is
    -- Sinais para ligar às entradas da uut
    signal s_reset, s_clk          : std_logic;
    signal s_enable, s_upDown_n : std_logic;
    -- Sinal para ligar às saídas da uut
    signal s_cntOut : std_logic_vector(3 downto 0);
begin
    -- Instanciação da Unit Under Test (UUT)
    uut : entity work.BinUDCntEnRst4(Behavioral)
        port map(reset    => s_reset,
                  clk      => s_clk,
                  enable   => s_enable,
                  upDown_n => s_upDown_n,
                  cntOut   => s_cntOut);

    -- Process clock
    clock_proc : process
    begin
        s_clk <= '0'; wait for 100 ns;
        s_clk <= '1'; wait for 100 ns;
    end process;
```

```
--Process stim
stim_proc : process
begin
    s_reset    <= '1';
    s_enable    <= '0';
    s_upDown_n <= '1';
    wait for 325 ns;

    s_reset    <= '0';
    wait for 25 ns;

    s_enable    <= '1';
    wait for 925 ns;
    s_enable    <= '0';
    wait for 375 ns;

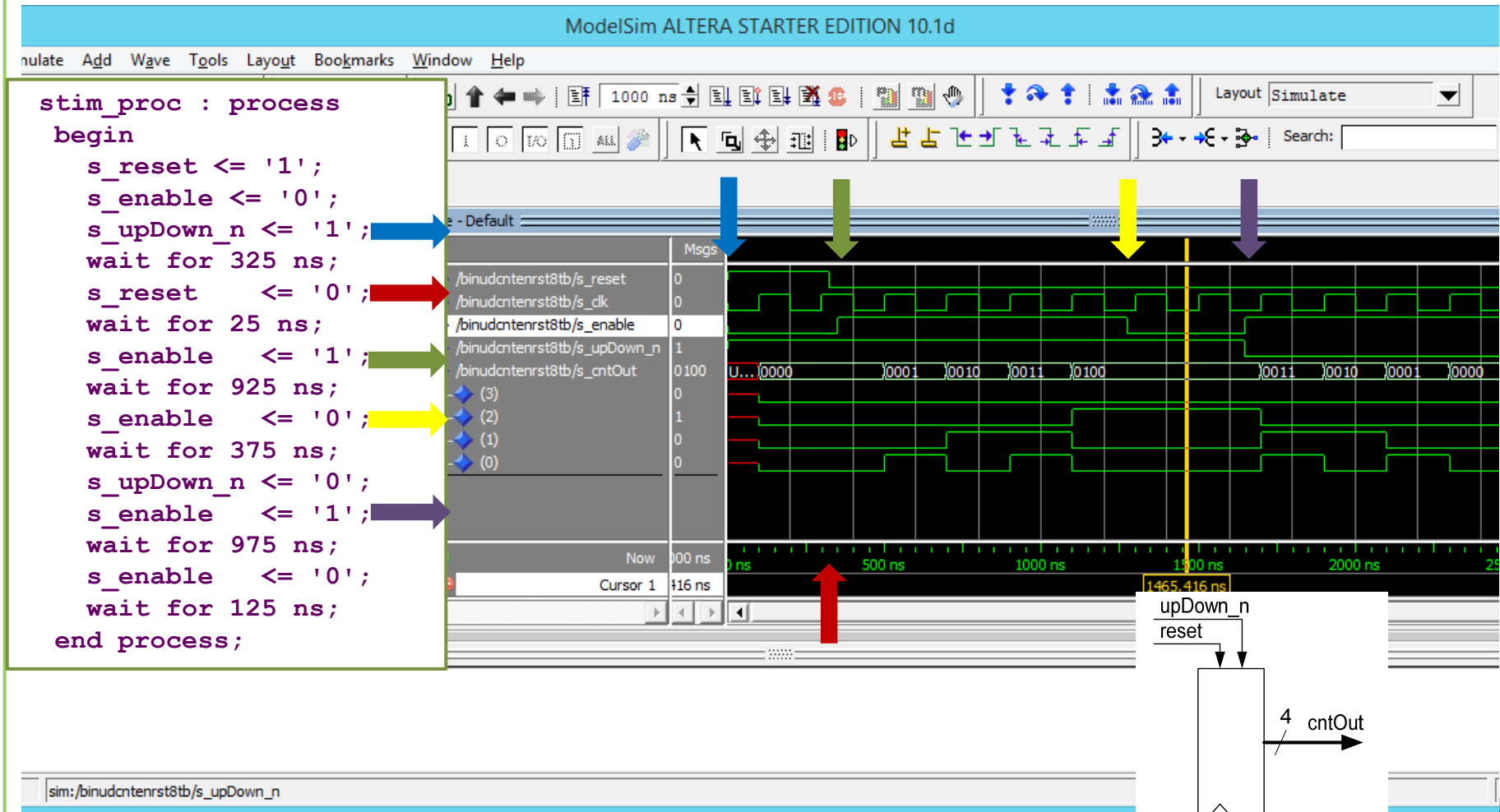
    s_upDown_n <= '0';
    s_enable    <= '1';
    wait for 975 ns;

    s_enable    <= '0';
    wait for 125 ns;
end process;
end Stimulus;
```



# Simulação c/ a Testbench

## BinUDCntEnRst8Tb



Nota: os passos de compilação e simulação serão abordados no guião prático 7.

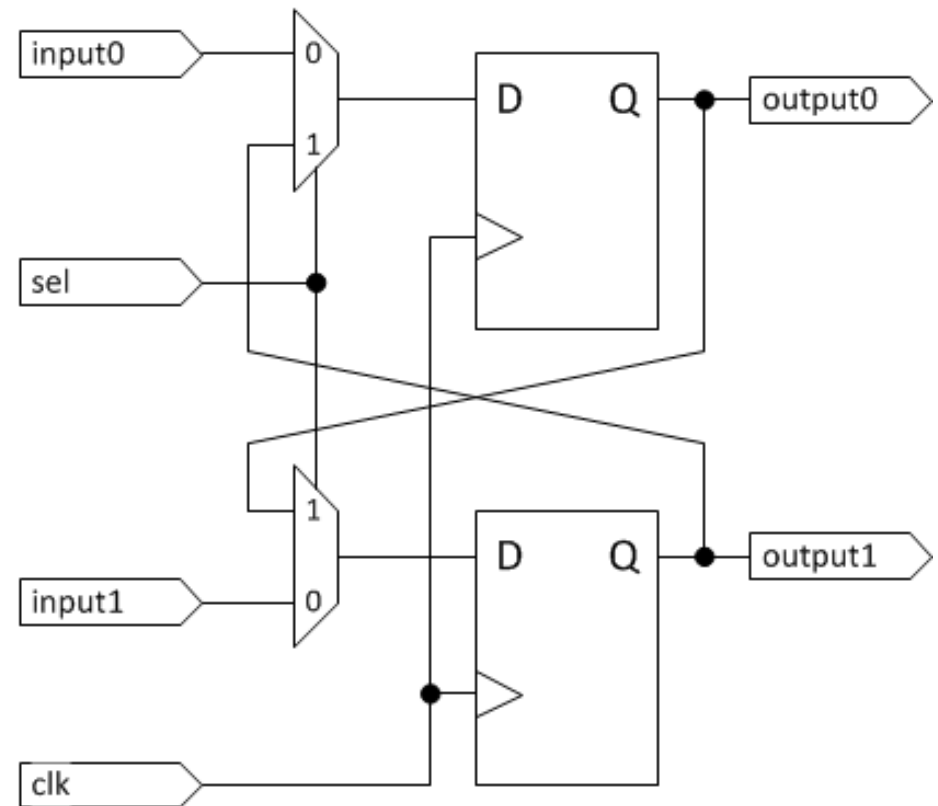
# VHDL (e outras HDLs)

- Linguagem de descrição de hardware
  - Suporta o conceito de concorrência para modelar o paralelismo do hardware
    - Atribuições concorrentes
    - Processos e listas de sensibilidade
    - Sinais (para comunicação entre processos e módulos)
    - Portos (para interligação de módulos)
- Um engenheiro de sistemas digitais deve dominar:
  - Os fundamentos da simulação, as suas vantagens e limitações
  - O subconjunto sintetizável de VHDL e aplicar estilos de codificação corretos
    - ... para assegurar resultados concordantes entre a simulação e a implementação!

# Modelação do Paralelismo do Hardware

Um exemplo simples:

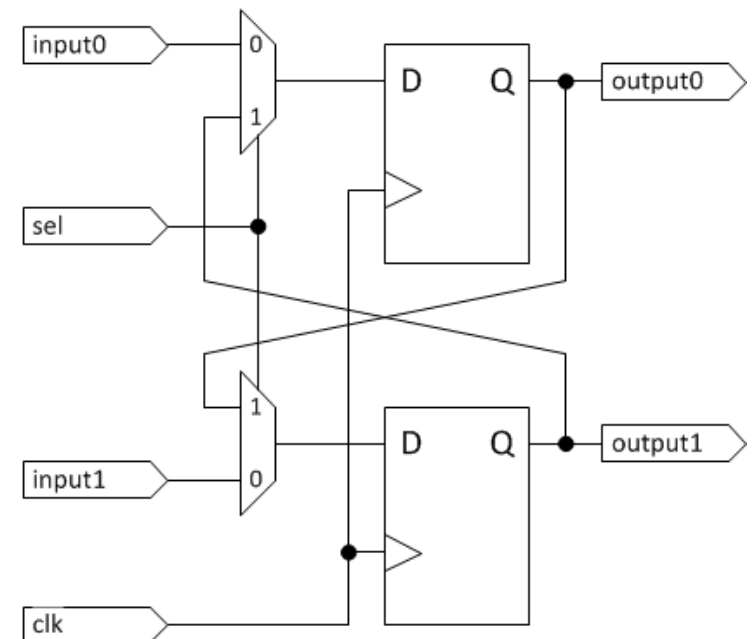
- No flanco ascendente do **clk**
  - Quando **sel** = '0'
    - **output0** <= **input0**
    - **output1** <= **input1**
  - Quando **sel** = '1'
    - **output0** <= **output1**
    - **output1** <= **output0**
- **clk**, **sel**, **input0**, **input1** – portos ou sinais
- **output0**, **output1** - sinais



# Primeira Abordagem de Modelação

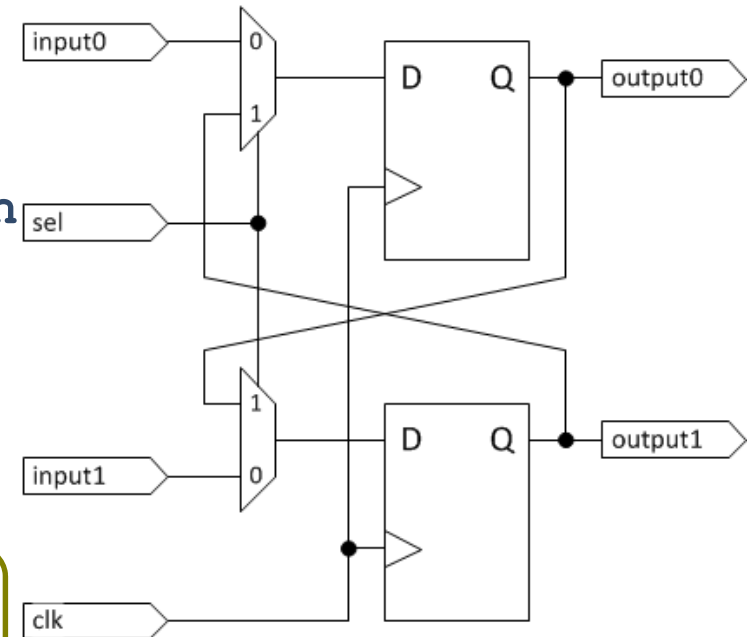
```
p_01 : process (clk)
begin
  if (rising_edge(clk)) then
    if (sel = '0') then
      output0 <= input0;
      output1 <= input1;
    else
      output0 <= output1;
      output1 <= output0;
    end if;
  end if;
end process;
```

**Existe algo de errado neste processo?**



# Primeira Abordagem de Modelação

```
p_01 : process (clk)
begin
  if (rising_edge(clk)) then
    if (sel = '0') then
      output0 <= input0;
      output1 <= input1;
    else
      output1 <= output0;
      output0 <= output1;
    end if;
  end if;
end process;
```



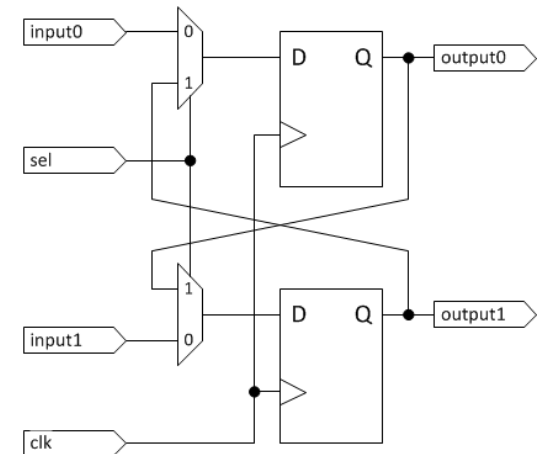
Também podemos trocar estas duas atribuições. O **comportamento simulado** e o **circuito sintetizado** será o mesmo e igualmente correto!



# Segunda Abordagem de Modelação

```
p_0 : process(clk)
begin
  if (rising_edge(clk)) then
    if (sel = '0') then
      output0 <= input0;
    else
      output0 <= output1;
    end if;
  end if;
end process;
```

Também podemos **dividir em dois processos**. A **ordem dos processos** no ficheiro VHDL é **irrelevante!** Mais uma vez, o **comportamento simulado** e o **circuito sintetizado** será o mesmo!



```
p_1 : process(clk)
begin
  if (rising_edge(clk)) then
    if (sel = '0') then
      output1 <= input1;
    else
      output1 <= output0;
    end if;
  end if;
end process;
```

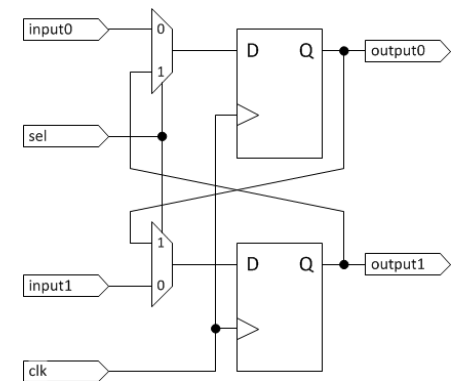
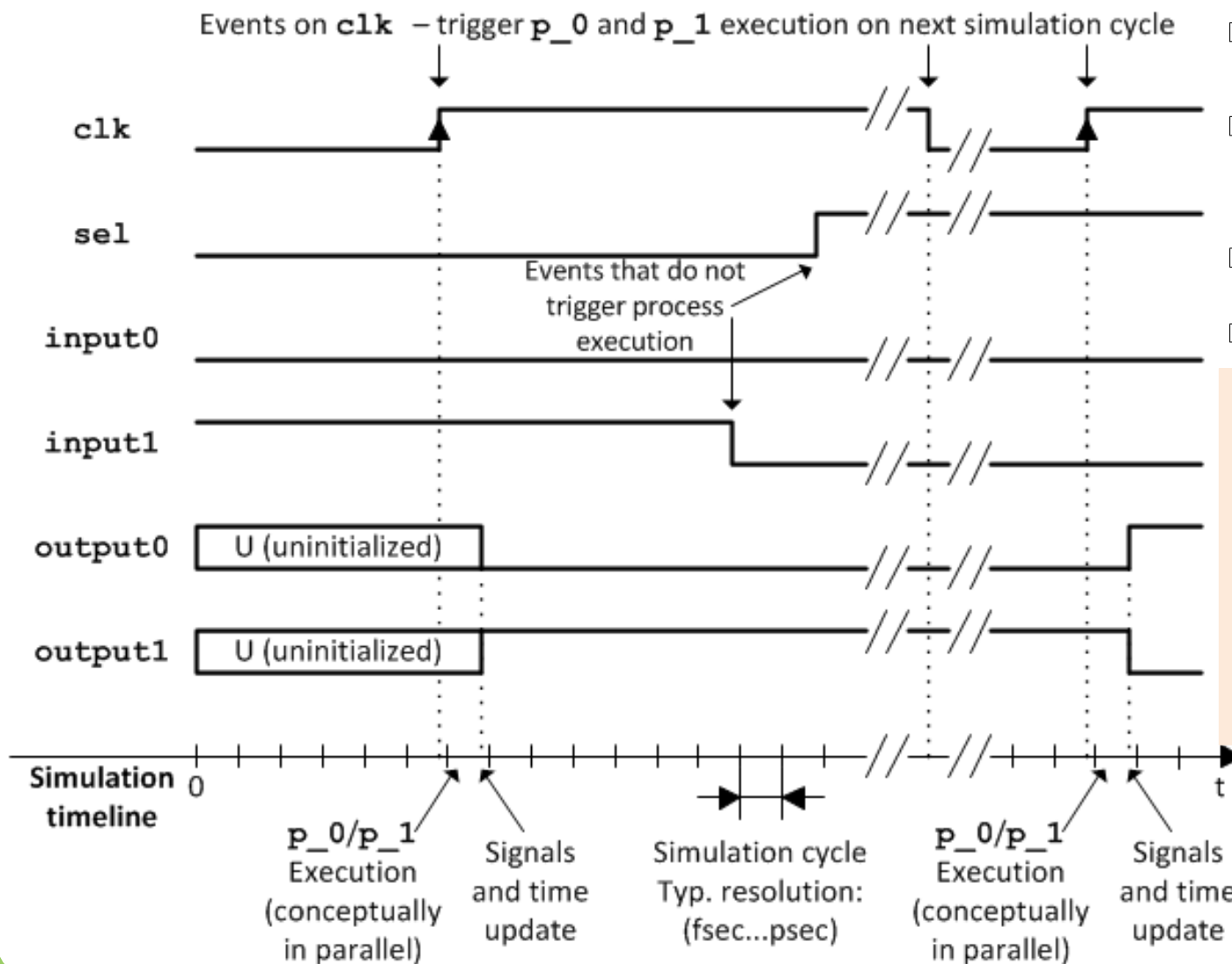
# Hardware *versus* Simulação do Modelo VHDL

- Hardware real
  - Todos os módulos operam em paralelo
    - Atualizam as saídas de acordo com
      - o seu estado interno (se aplicável)
      - entradas de inicialização, sincronização, controlo e dados
  - Os atrasos são impostos pela tecnologia e projeto do sistema

# Hardware *versus* Simulação do Modelo VHDL

- Ambiente de simulação
  - Baseada em ferramentas de simulação de eventos discretos a executar sobre processadores de uso geral
  - Pode ser realizada a vários níveis / fases do projeto:
    - Comportamental (inicial, ideal - sem atrasos)
    - Funcional (pós-síntese, sem atrasos)
    - Temporal (pós-implementação, considerando os atrasos do circuito)
  - Ciclos de simulação muito inferiores (resolução muito mais fina) que os períodos dos sinais do sistema
  - Construções para modelar o paralelismo
    - Atribuições concorrentes
    - Processos
      - Execução concorrente (conceptualmente em paralelo e em tempo nulo)
      - Ativação controlada por eventos em sinais presentes nas listas de sensibilidade

# Aspetos Básicos da Simulação de Eventos Discretos



Na simulação os sinais são atualizados conceitualmente em paralelo no final do ciclo de simulação

# Processos e Listas de Sensibilidade

- Para evitar discrepâncias entre o comportamento em simulação e em hardware (FPGA) a funcionalidade de um processo deve ser completamente descrita no seu corpo
  - As listas de sensibilidade são apenas uma forma de otimizar o desempenho da simulação (i.e. para evitar execuções desnecessárias de processos no simulador)
  - O comportamento de um processo deve ser o mesmo com ou sem lista de sensibilidade
    - As ferramentas de síntese são capazes de detetar sinais em falta na listas de sensibilidade
    - Por outro lado, algumas ferramentas de simulação executam os modelos “as is”

# Listas de Sensibilidade em Processos

- Não têm qualquer influência na síntese do sistema
  - Comportamento é idêntico com ou sem lista de sensibilidade
  - Funcionalidade de um processo tem que ser completamente descrita no seu corpo
- Apenas afetam a simulação
  - Permitem otimizar o desempenho (evitando execução desnecessária de processos) - um processo só é desencadeado quando há alteração em pelo menos 1 dos sinais da lista de sensibilidade
  - => **(IMPORTANTE)** pode haver discrepância entre o comportamento observado em simulação e o real (FPGA) devido a lista de sensibilidade incompleta

# Alguns Excertos de Código Incorretos

Módulo	Descrição Incorreta	Comentário
Flip-flop tipo D	<pre>process (clk) begin     if (clk = '1') then         dataOut &lt;= dataIn;     end if; end process;</pre>	Simula corretamente, <b><u>mas</u></b> sintetiza e funciona incorretamente em hardware!!!
Flip-flop tipo D com reset assíncrono	<pre>process (clk) begin     if (reset = '1') then         dataOut &lt;= '0';     elsif (rising_edge(clk)) then         dataOut &lt;= dataIn;     end if; end process;</pre>	Não simula corretamente, <b><u>apesar</u></b> de sintetizar e funcionar corretamente em hardware!!!

Como corrigir?

# Comentários Finais

- No final desta aula e do trabalho prático 7 de LSD, deverá ser capaz de:
  - Escrever *testbenches* para simulação de componentes combinatórios e sequenciais
  - Compreender (ainda melhor) as construções VHDL usadas para modelar o paralelismo do hardware
    - Usar corretamente o paradigma de modelação na descrição de sistemas digitais
  - Conhecer os fundamentos da simulação em VHDL
  - Selecionar os sinais a incluir na lista de sensibilidade de um processo
    - Todas as entradas no caso de processos combinatórios
    - *Clock* e sinais assíncronos no caso de componentes sequenciais  
(embora se deva optar sempre por sinais de inicialização síncronos!)
  - Realizar simulações em diversas etapas do fluxo de projeto