

Laboratório de Sistemas Digitais

Aula Teórico-Prática 7

Ano Letivo 2024/25

Construção e utilização de
testbenches para simulação em VHDL

Tópicos sobre simulação vs síntese

Conteúdo

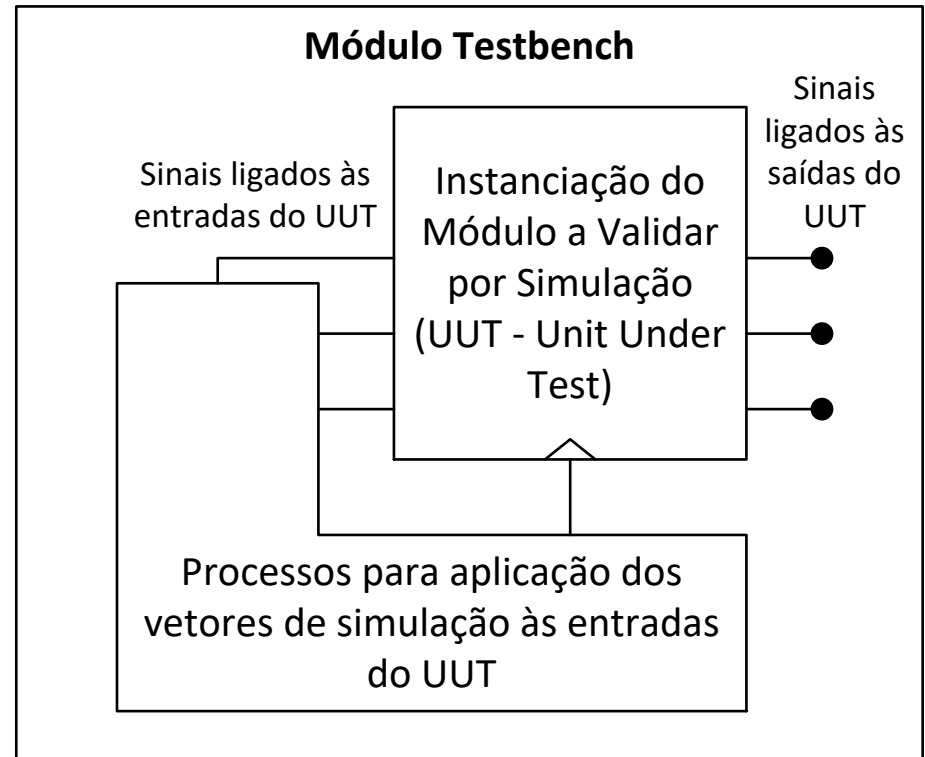
- Simulação de modelos em VHDL
 - Utilidade da simulação
 - Motivação para a utilização de *testbenches*
 - Construção de *testbenches* para simulação de componentes
 - Combinatórios
 - Sequenciais
- Tópicos fundamentais sobre simulação e síntese em VHDL
 - (Mais detalhes sobre as) construções para modelação 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

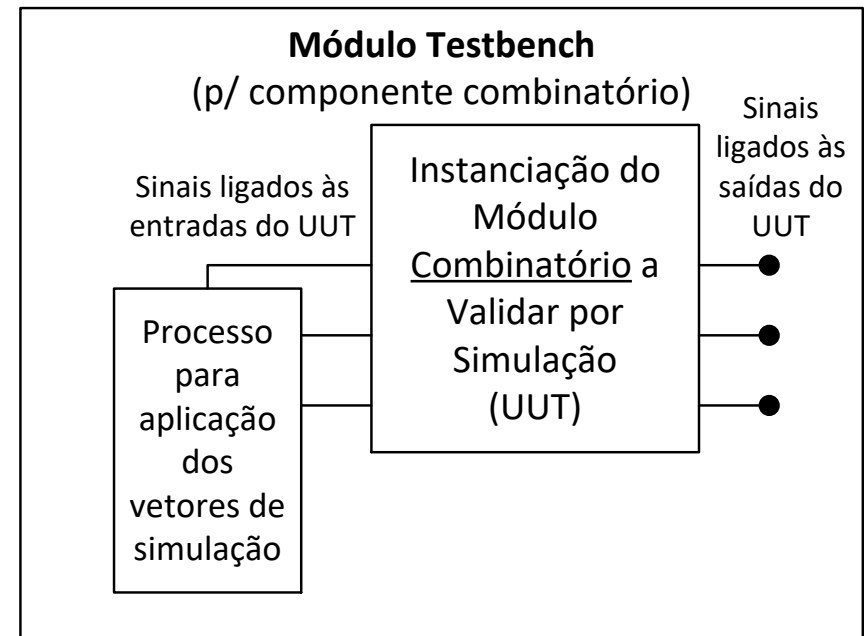
Simulação em VHDL

- 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

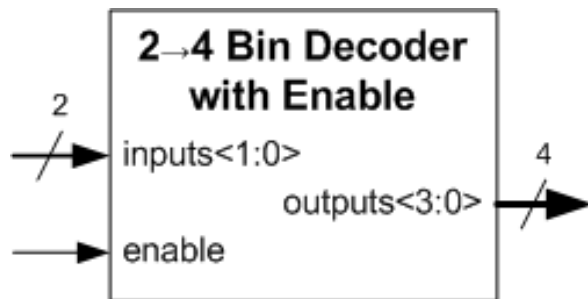


Exemplo de um Componente Combinacional

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;
```



(inputs = "11") e não só!!

```
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 *Testbench* para um Componente Combinacional

```
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

```
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;
```

ModelSim ALTERA STARTER EDITION 10.1d

Wave - Default

Kind	Msgs
Signal	1
Signal	00
Signal	0001
Signal	0
Signal	0
Signal	0
Signal	0
Signal	1

Now: 000 ns
Cursor 1: 0.00 ns

2 → 4 Bin Decoder with Enable

inputs<1:0>

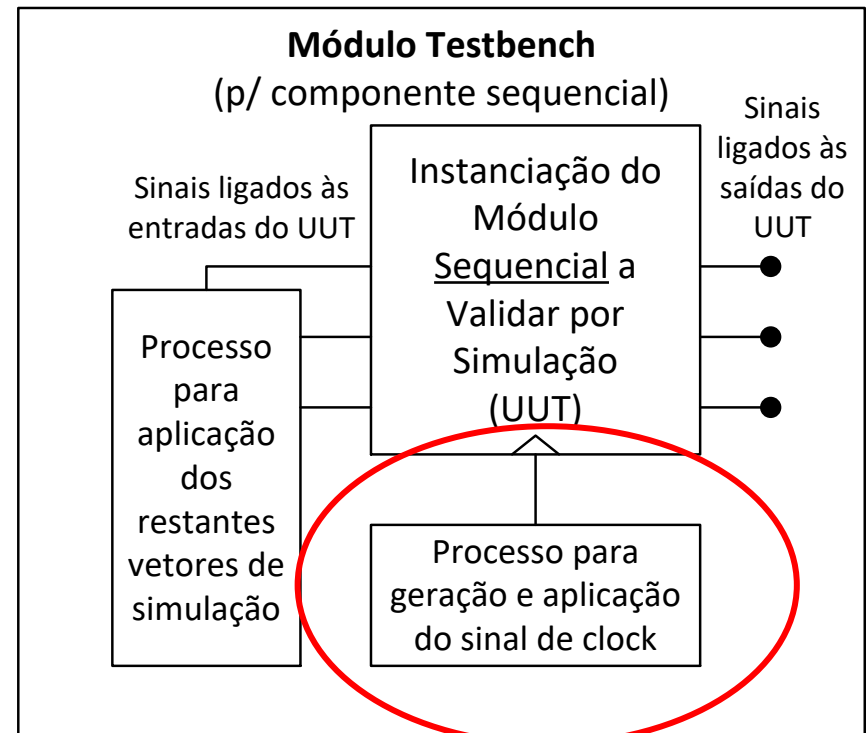
enable

outputs<3:0>

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

Estrutura típica de *Testbench* para componente Sequencial

- Entidade
 - Sem portas (interface vazia)
- Arquitectura
 - Instanciação da UUT
 - Declaração dos sinais a ligar aos portos da UUT
 - **Processo para gerar *clock***
 - Processo para aplicar os vectores de simulação ao longo do tempo
 - Múltiplos processos em sistemas mais complexos



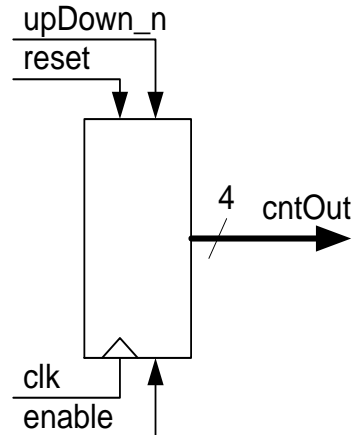
Exemplo de um Componente Sequencial

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;
```

reset
síncrono!



```
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;
    end process;

    cntOut <= std_logic_vector(s_cntValue);

end Behavioral;
```

Exemplo para Componente 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;

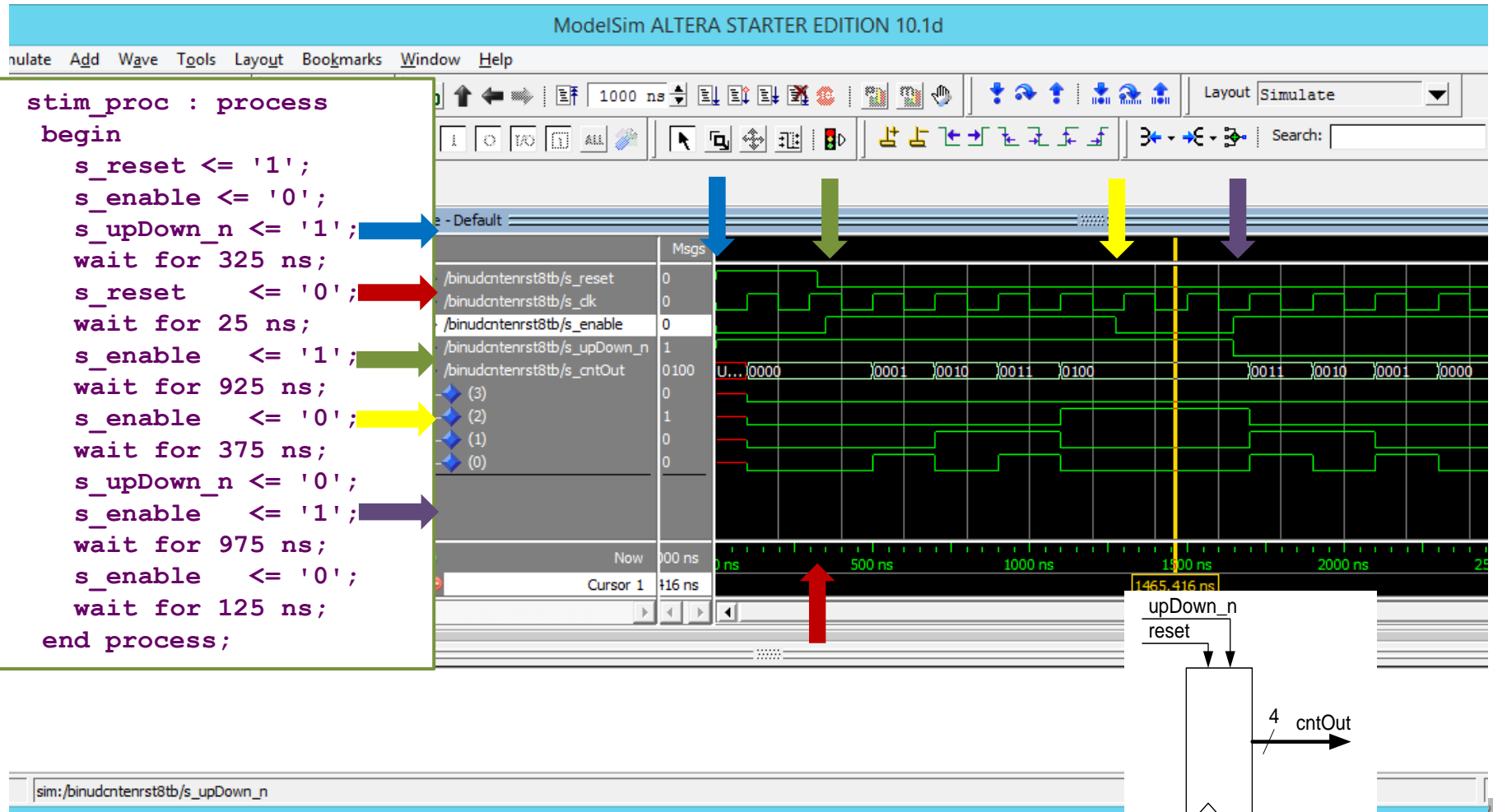
    s_enable   <= '0';
    wait for 125 ns;

end process;
end Stimulus;
```



Simulação c/ a Testbench

BinUDCntEnRst8Tb

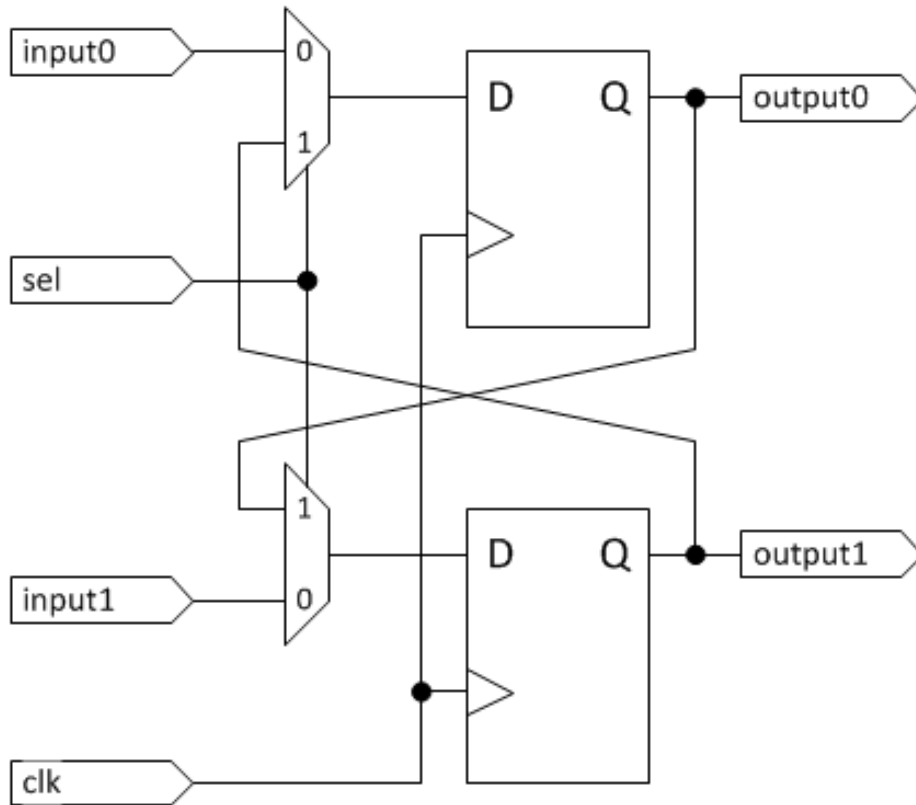


Nota: os passos de compilação e simulação serão abordados no guião prático 7.

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



Flanco ascendente do `clk`:

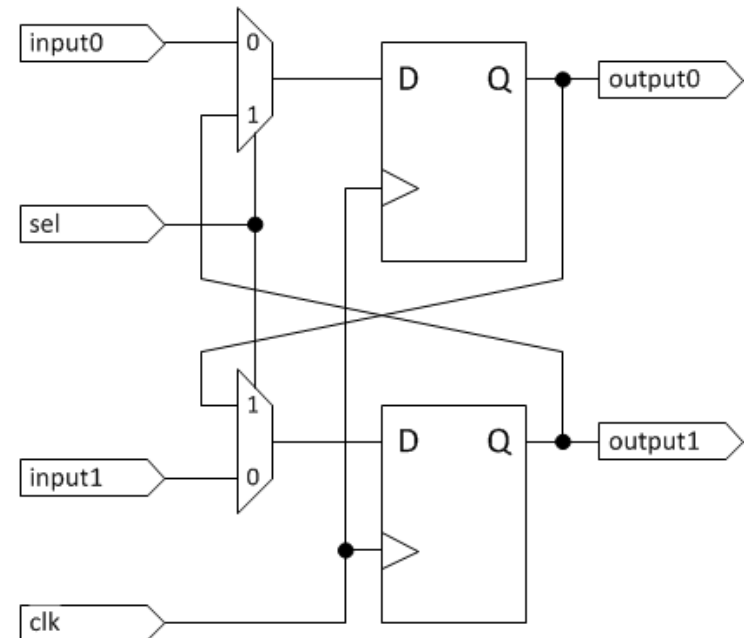
- Quando `sel = '0'`
 - `output0 <= input0`
 - `output1 <= input1`
- Quando `sel = '1'`
 - `output0 <= output1`
 - `output1 <= output0`

- `clk`, `sel`, `input0`, `input1` – portos ou sinais
- `output0`, `output1` – sinais

Primeira Abordagem de Modelação

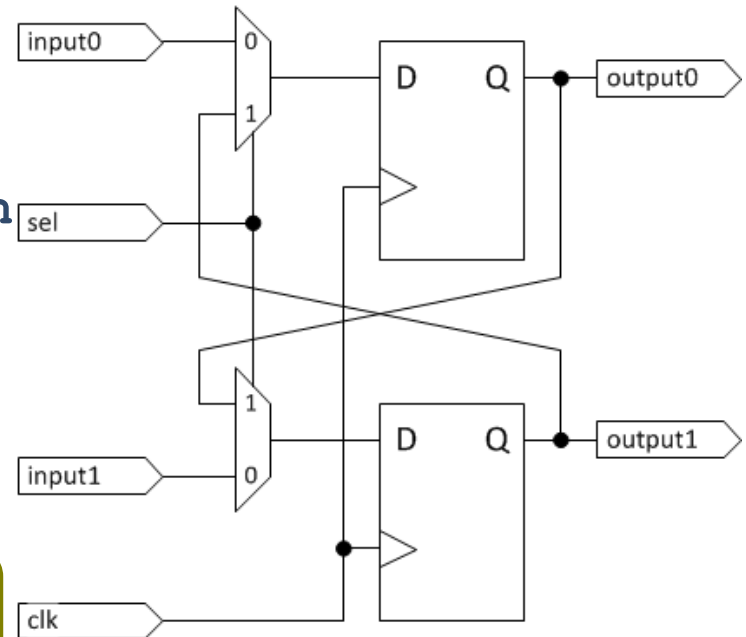
```
p_01 : process (clk)
begin
    if (rising_edge(clk)) then
        if (sel = '0') then
            output0 <= input0;
            output1 <= input1;
        else
            output0 <= output1;
            output1 <= output0;
        end if;
    end if;
end process;
```

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;
```

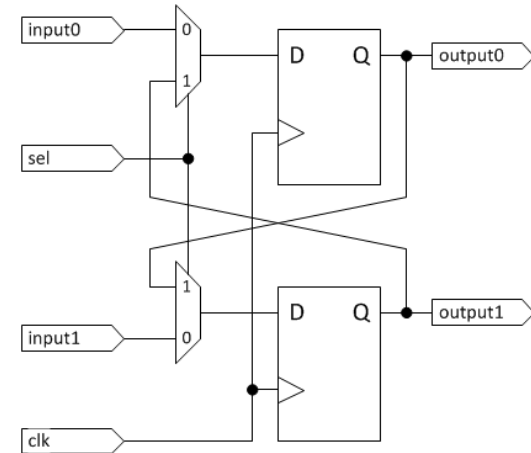


Nada errado!
Podemos trocar a ordem
destas duas atribuições.
São concorrentes!

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;
```

Podemos trocar a ordem
destes dois processos.
São concorrentes!



```
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;
```

Paralelismo do hardware em Simulação

Ferramentas de simulação

- Aplicáveis em vários níveis / fases do projecto:
 - Comportamental (inicial, ideal - sem atrasos)
 - Funcional (pós-síntese, sem atrasos)
 - Temporal (pós-implementação, considerando os atrasos do circuito)
- Construídas em software (PC de uso geral)...
- ...mas desenhadas para modelar paralelismo
 - Algoritmos para simulação de eventos discretos
 - Ciclos de simulação muito inferiores aos períodos dos sinais do sistema (i.e. com resolução muito mais fina)

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 afectam 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

Excertos de código incorrectos

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

Como corrigir?

Comentários Finais

- No final desta aula e do trabalho prático 7, deverá:
 - Conhecer os fundamentos da simulação em VHDL
 - Ser capaz de escrever *testbenches* para simulação de componentes combinacionais e sequenciais
 - Saber seleccionar 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!)
 - Compreender (ainda melhor) a modelação do paralelismo do hardware nas construções VHDL e em simulação
 - Reconhecer a utilidade da simulação em diversas etapas do fluxo de projeto