

Respostas - Exercícios sobre a Linguagem VHDL

1)

- a) Escreva uma descrição comportamental de sua arquitetura utilizando um processo com comando **case**.

```
architecture process_case of mux8 is
```

```
begin
process (select)
begin
    case select      is
        when "00" => Z_out  <=  In_1;
        when "01" => Z_out  <=  In_2;
        when "10" => Z_out  <=  In_3;
        when "11" => Z_out  <=  In_4;
        when others   => null;
    end case;
end process;
end process_case;
```

- b) Escreva uma descrição comportamental de sua arquitetura utilizando um processo com comando **if-then-else**.

Só com o corpo do processo:

```
if(select = "00") then
    Z_out  <=  In_1;
elsif(select = "01") then
    Z_out  <=  In_2;
elsif(select = "10") then
    Z_out  <=  In_3;
elsif(select = "11") then
    Z_out  <=  In_4;
end if;
```

- c) Escreva uma descrição em fluxo de dados (*dataflow*) de sua arquitetura usando designações de sinais concorrentes.

```
architecture data_flow_1 of mux8 is
```

```
signal select_00, select_01, select_10, select_11: bit_vector (0 to 7);
```

```
begin
```

```
Z_out  <= select_00 OR select_01, OR select_10 OR select_11;
select_00 <= In_1 AND NOT (select (0) & select (0) & select (0) & select (0) &
select (0) & select (0) & select (0) & select (0)) AND NOT (select (1) & select
(1) & select (1) & select (1) & select (1) & select (1) & select (1));
select_01 <= In_1 AND NOT (select (0) & select (0) & select (0) & select (0) &
select (0) & select (0) & select (0) & select (0)) AND (select (1) & select (1) &
select (1) & select (1) & select (1) & select (1) & select (1));
```

```

select_00 <= In_1 AND (select (0) & select (0) & select (0) & select (0)
& select (0) & select (0)) AND NOT (select (1) & select (1) & select
(1) & select (1) & select (1) & select (1) & select (1));
select_00 <= In_1 AND (select (0) & select (0) & select (0) & select (0)
& select (0) & select (0)) AND (select (1) & select (1) & select (1)
& select (1) & select (1) & select (1) & select (1));
end data_flow_1;

```

- d) Escreva uma descrição em fluxo de dados (*dataflow*) de sua arquitetura usando as designações com **when-else**.

```

architecture data_flow_2 of mux8 is
    signal select_00, select_01, select_10, select_11: bit_vector (0 to 7);

begin
    Z_out <= In_1      when (select = "00") else
        In_2      when (select = "01") else
        In_2      when (select = "01") else
        In_2      when (select = "01") else
        (others => 'X');
end data_flow_2;

```

2)

```

library ieee;
use ieee.std_logic.all;
use ieee.numeric_std.all;

Entity circuito is
Generic ( WIDTH:      NATURAL :=8);
Port ( entrada : in std_logic_vector (WIDTH-1 downto 0);
       saida   : out unsigned (WIDTH-1 downto 0);
End circuito;

architecture exercicio of circuito is
    signal auxiliar: std_logic_vector (WIDTH-1 downto 0);

begin
    saida <= unsigned (signal_auxiliar);

    gen_inversao: for I in 0 to WIDTH generate
        aux (WIDTH-1-I) <= entrada (I);
    end generate;
    (
    end exercicio;

```

- 3) Caso enable=1, a saída apresenta um clock com período de 100ns será gerado.

Caso enable=0, a saída é constante 1 ou 0, dependendo do valor de lag quando da transição de enable 1 → 0.

- 4) Assunindo que updown='1' → contagem para cima e updown='0' → contagem para baixo.

```

architecture arch of step_counter is
    signal cnt_s      : UNSIGNED (3 downto 0)      := to_unsigned (0, 4);

begin
    cnt_rdy <=      '1' when (cnt_s = COUNT_MAX) else
                    '0';

    process(clk, res, updown)
    begin
        if clk'event and clk = '1' then
            if(res = '1') then
                cnt_s <= to_unsigned(0, cnt_s'length);
            elsif (updown='1') then
                if (cnt_s = COUNT_MAX) then
                    cnt_s <= to_unsigned(0, cnt_s'length);
                else
                    cnt_s <= cnt_s + 1;
                end if;
            else
                if (cnt_s =0) then
                    cnt_s <= to_unsigned(COUNT_MAX, cnt_s'length);
                else
                    cnt_s <= cnt_s - 1;
                end if;
            end if;
        end if;
    end process;

    cnt_value <= std_logic_vector (cnt_s);
end arch;

```

5)

- a) Escreva a declaração da entidade.

```

Entity bloco_3en_2sa is
    Port ( A, B, C : in Bit;
              X,Y: out Bit );
End bloco_3en_2sa;

```

- b) Escreva a arquitetura da entidade usando o estilo estrutural. Use somente a seguinte entidade como componente:

```

architecture arch of bloco_3en_2sa is
    signal aux_1, aux_2, aux_3, aux_4, aux_5, aux_6, aux_7 : Bit;

    component NAND2 is
        Port ( A, B : in Bit;
                  Z: out Bit);
    End component;

```

```

begin
    inv_a: port map (A, A, aux_1);
    inv_b: port map (B, B, aux_2);
    inv_c: port map (C, C, aux_3);
    nand_1: port map (aux_1, C, aux_4);
    nand_2: port map (aux_2, aux_4, X);
    nand_3: port map (A, aux_3, aux_5);
    nand_4: port map (B, C, aux_6);
    inv_1: port map (aux_6, aux_6, aux_7);
    nand_5: port map (aux_1, aux_7, aux_8);
    nand_6: port map (aux_5, aux_8, Y);
end arch;

```

- c) Escreva a arquitetura da entidade usando o estilo fluxo de dados (*dataflow*).

```

architecture dataflow of bloco_3en_2sa is
    signal aux_1, aux_2, aux_3, aux_4, aux_5, aux_6, aux_7 : Bit;

begin
    Aux_1 <= NOT (A) AND C;
    X <= B OR Aux_1;
    Aux_2 <= A AND NOT(C);
    Aux_3 <= NOT(A) AND B AND C;
    Y <= Aux_2 OR Aux_3;

```

End dataflow

6)

a)

```

Library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity dff is
    Generic ( t_FF:  t      := 12ns);
    port
        (d,clk, reset      :      in STD_LOGIC;
         q                  :      out STD_LOGIC
        );
end dff;

architecture behavior of dff is

begin
    process ( clk , reset)
    begin
        if clk'event and clk = '1' then
            if reset = '1' then
                q <= '0' AFTER t_FF;
            else
                q <= d AFTER t_FF;
            end if;
        end if;
    end process;
end behavior;

```

b)

```
entity xor is
Generic ( t_XOR:      t      := 8ns);
port
(A, B      :      in STD_LOGIC;
Z         :      out STD_LOGIC
);
end dff;
```

architecture behavior of dff is

```
begin
      Z <= A EXOR B AFTER t_XOR;
end behavior;
```

c)

```
Entity celular_automata is
Generic ( WIDTH:      NATURAL :=8);
Port (  clk, reset: in std_logic;
        z   : out std_logic_vector (WIDTH-1 downto 0);
End celular_automata;

architecture strucural of celular_automata is
signal q, aux : std_logic_vector (WIDTH-1 downto 0);

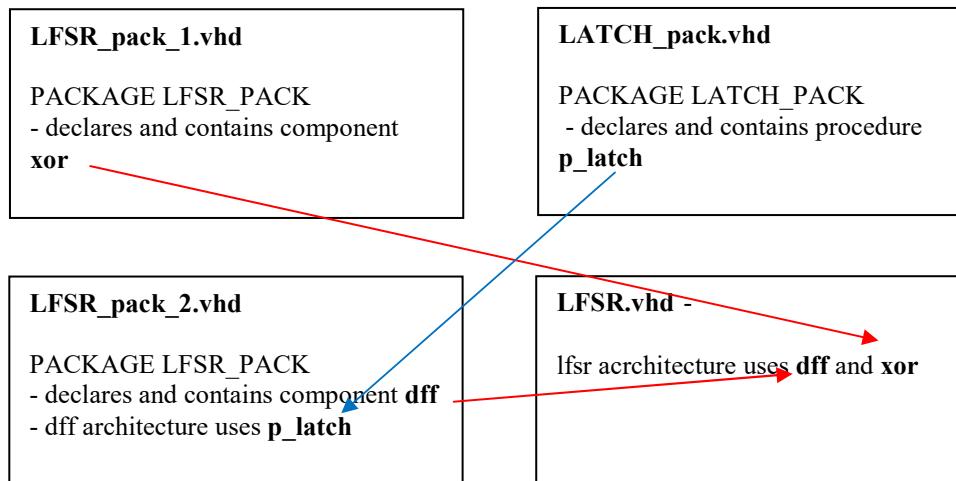
component EXOR is (...etc..)
...
End component;

component dff is (...etc..)
...
End component;

begin
DFF_0:      dff generic map (t_FF=15ns) port map (q(1), clk, reset, q(0));
DFF_N-1:    dff generic map (t_FF=15ns) port map (q(WIDTH-2), clk, reset,
q(WIDTH-1));
generate_FFs: for I in 1 to WIDTH-2 generate
            dff_array:      dff generic map (t_FF=15ns)
                           port map (aux(I), clk, reset, q(I));
            exor_array:    exor generic map (t_FF=10 ns)
                           port map (q(I-1), q(I+1), aux(I));
end generate;
z <= q;
end structural;
```

7)

a)



- b) Mudanças anotadas em vermelho - cada vhdl corresponde a uma pasta/library com o mesmo nome do arquivo)

====ARQUIVO LFSR_pack_1.vhd=====

PACKAGE LFSR_PACK IS

```
COMPONENT xor2
  GENERIC(t_xor : time := 4 ns);
  PORT (x, y : IN BIT;
        z : OUT BIT);
END COMPONENT;

END;
```

-- descrição do xor

ENTITY xor2 IS

```
  GENERIC(t_xor : time := 4 ns);
  PORT( x, y: IN BIT;
        z: OUT BIT);
```

END xor2;

ARCHITECTURE dataflow OF xor2 IS

BEGIN

```
  z <= x XOR y AFTER t_xor;
```

END dataflow;

==== Fim do ARQUIVO LFSR_pack_1.vhd=====

====ARQUIVO LFSR_pack_2.vhd=====

PACKAGE LFSR_PACK IS

COMPONENT dff

GENERIC(*t_q* : time := 12 ns; *t_qbar* : time := 14 ns);

PORT(*D*, *clock*: IN Bit;

q: OUT Bit;

qbar: OUT Bit);

END COMPONENT;

END;

-- descricao do dff

library LATCH_pack;

use LATCH_pack.LATCH_PACK.all;

ENTITY dff IS

GENERIC(*t_q* : time := 12 ns; *t_qbar* : time := 14 ns);

PORT(*D*, *clock*: IN Bit;

q: OUT Bit:='1';

qbar: OUT Bit);

END dff;

ARCHITECTURE with_package OF dff IS

BEGIN

PROCESS (*clock*)

BEGIN

IF (*clock*'EVENT AND *clock* ='0') THEN

p_latch(*d*, *t_qbar*, *q*, *qbar*);

END IF;

END PROCESS;

END with_package;

==== Fim do ARQUIVO LFSR_pack_2.vhd=====

====ARQUIVO LATCH_pack.vhd=====

```
PACKAGE LATCH_PACK IS
    PROCEDURE p_latch (SIGNAL d : IN BIT;
                        CONSTANT tempo :IN time;
                        SIGNAL q, qbar: OUT BIT);
END;
```

```
PACKAGE BODY LATCH_PACK IS
    PROCEDURE p_latch ( SIGNAL d : IN BIT;
                        CONSTANT tempo :IN time;
                        SIGNAL q, qbar: OUT BIT) IS
        BEGIN
            q <= d AFTER tempo;
            qbar <= NOT d AFTER tempo;
        END ;
    END ;
```

==== Fim do ARQUIVO LATCH_pack.vhd =====

== ARQUIVO LFSR.vhd =====

```
library LFSR_pack_1.vhd;
use LFSR_pack_1.LFSR_pack.all;
library LFSR_pack_2.vhd;
use LFSR_pack_2.LFSR_pack.all;
```

```
ENTITY LFSR IS
    PORT( clock: IN BIT;
          o: OUT  BIT_VECTOR (2 DOWNTO 0));
END LFSR;
```

ARCHITECTURE structure OF LFSR IS

```
COMPONENT dff
    GENERIC(t_q : time := 12 ns; t_qbar : time := 14 ns);
    PORT( D, clock: IN Bit;
          q: OUT Bit;
          qbar: OUT Bit);
END COMPONENT;
```

```
COMPONENT xor2
    GENERIC(t_xor : time := 4 ns);
    PORT (x, y : IN BIT;
```

```
    z : OUT BIT);  
END COMPONENT;
```

```
signal xor_out, q0, q1, q2, qbar :BIT;
```

(arquivo continua na próxima página)

```
BEGIN
```

```
  FF0 : dff GENERIC MAP (11 ns, 13 ns) PORT MAP (q1, clock, q0, qbar);
```

```
  FF1 : dff GENERIC MAP (11 ns, 13 ns) PORT MAP (q=>q1, d=>xor_out,  
clock=>clock);
```

```
  FF2 : dff GENERIC MAP (11 ns, 13 ns) PORT MAP (clock=>clock, q=>q2,  
d=>q0,);
```

```
  X0 : xor2 PORT MAP (q2, q0, xor_out);
```

```
  o <= (q2,q1,q0);
```

```
END structure;
```

== Fim do ARQUIVO LFSR.vhd =====