Módulo 4

Introdução ao VHDL

Conceitos básicos do VHDL

Modelação, Simulação e **Síntese** de Sistemas Digitais

entity – declara o interface de um componente;

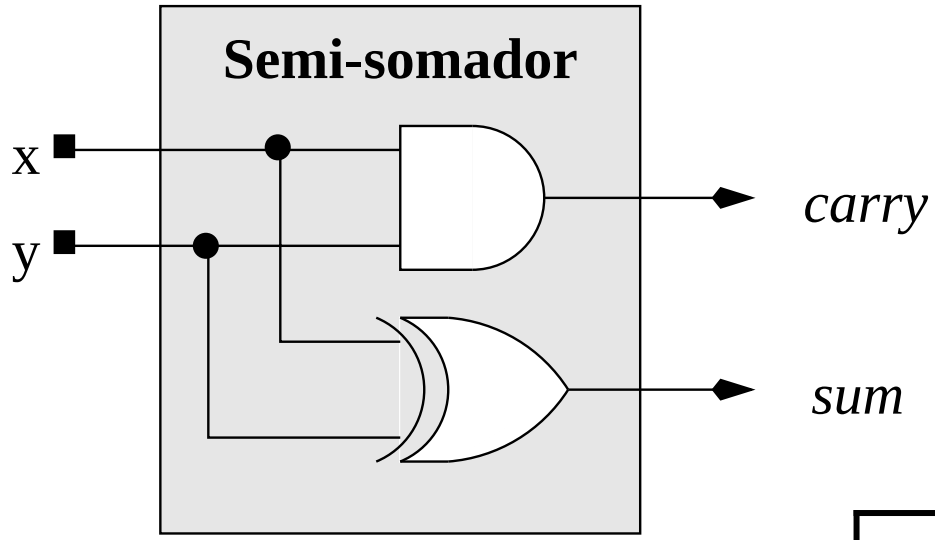
architecture – descreve a realização de um componente;

- fluxo de dados
- estrutural
- comportamental

Uma entidade pode ter várias arquitecturas.

Semi-Somador

- descrever em VHDL um semi-somador de 1 bit



x	y	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Semi-Somador

```
entity semi_somador is
  port (
        x,   y       : in  bit;
        sum, carry : out bit );
end entity semi_somador;
```

nome

modo

tipo

nome da descrição

entidade descrita

```
architecture dflow of semi_somador is
  begin
    sum   <= (x xor y) after 2 ns;
    carry <= (x and y) after 2 ns;
end architecture dflow;
```

event triggered

atribuições concorrentes aos sinais com **atraso** de **2ns**

Semi-Somador

- substituir o tipo **bit** por **std_logic**, definido no *package* **std_logic_1164** da biblioteca **IEEE**:

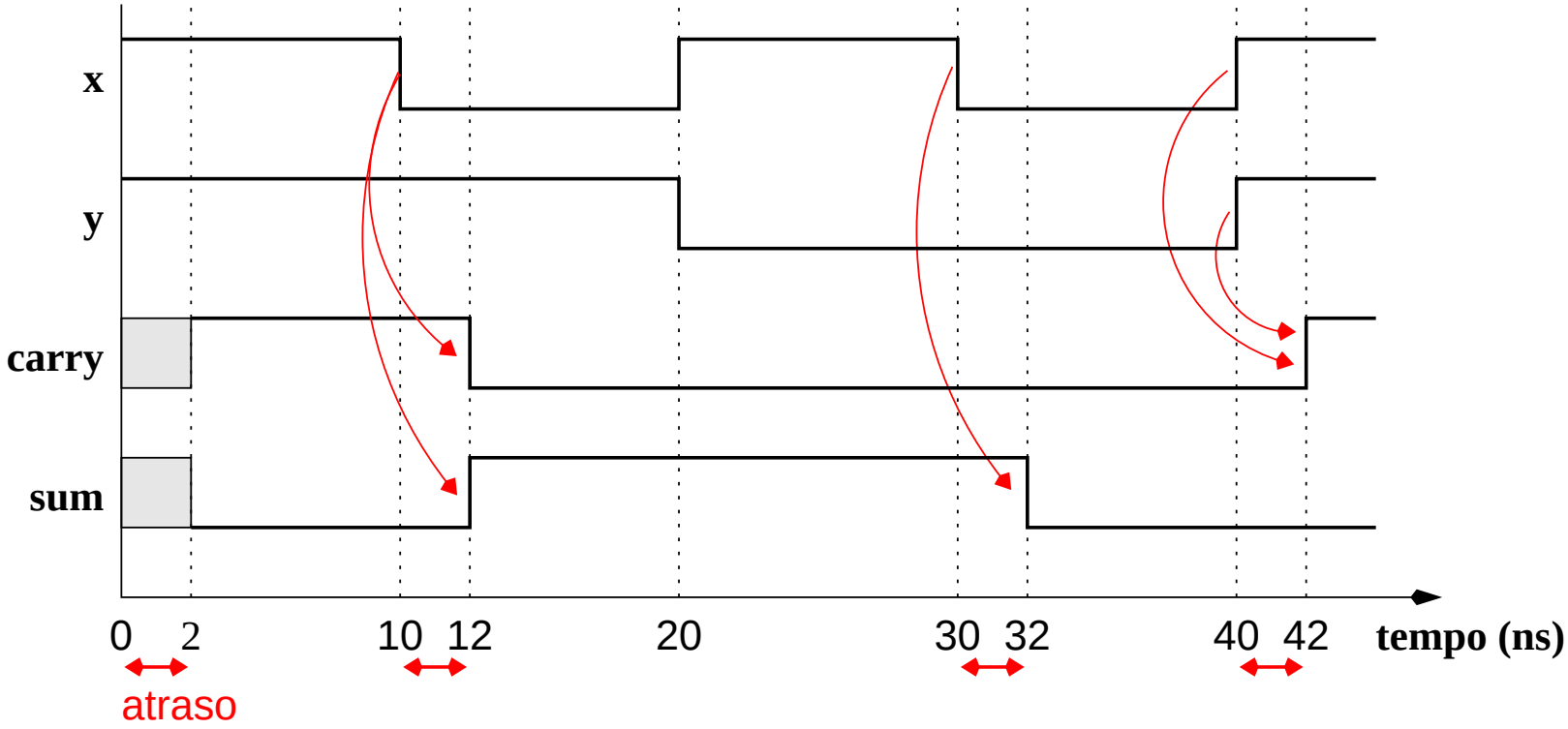
```
library IEEE;
use     IEEE.std_logic_1164.all;
```

```
entity semi_somador is
  port (
    x,   y     : in  std_logic;
    sum, carry : out std_logic );
end entity semi_somador;
```

```
architecture dflow of semi_somador is
  begin
    sum   <= (x xor y) after 2 ns;
    carry <= (x and y) after 2 ns;
end architecture dflow;
```

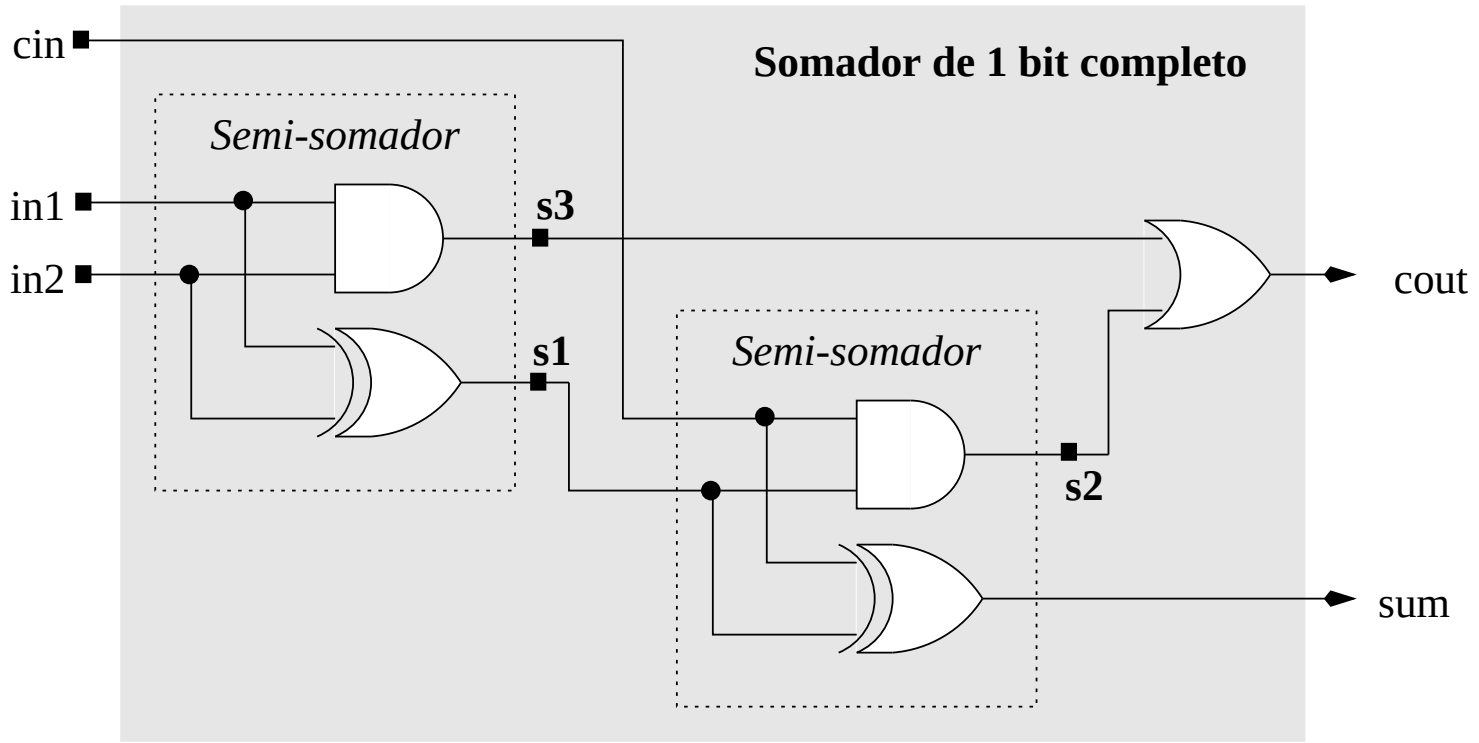
Semi-Somador

- ilustração do funcionamento do semi-somador



Somador Completo

- descrever em VHDL um somador de 1 bit



Somador Completo

```
library IEEE;
use     IEEE.std_logic_1164.all;

entity somador is
  port (
    in1, in2, cin : in  std_logic;
    sum, cout     : out std_logic );
end entity somador;
```

Somador Completo

```
library IEEE;
use     IEEE.std_logic_1164.all;


entity somador is
  port (
    in1, in2, cin  : in  std_logic;
   sum, cout       : out std_logic );
end entity somador;



architecture estrutural of somador is

  component semi_somador is
    port (
      x, y       : in  std_logic;
      sum, carry : out std_logic );
  end component semi_somador;
```

```
signal s1, s2, s3 : std_logic;
constant atraso   : time := 2 ns;

begin

 SUM1: semi_somador
      port map (
        x  => in1  ,
        y  => in2  ,
        sum  => s1   ,
        carry => s3 );

 SUM2: semi_somador
      port map (
        x => cin     ,
        y => s1      ,
        sum => sum     ,
        carry   => s2 );

 FIN: cout <= s3 or s2 after atraso;

end architecture estrutural ;
```

Somador Completo : alternativa *data flow*

```
architecture dflow of somador is

    signal s1, s2, s3 : std_logic;
    constant atraso   : time := 2 ns;

  begin

  X1: s1   <= (in1 xor in2) after atraso;
  A1: s2   <= (cin and  s1) after atraso;
  A2: s3   <= (in1 and in2) after atraso;
  X2: sum  <= (s1  xor cin) after atraso;
  O1: cout <= (s2  or   s3) after atraso;

end architecture dflow ;
```

Descrição comportamental

Descrição comportamental ⇔ algorítmica

A sequência de instruções aparece dentro de um *process*

Um processo tem uma lista de sensibilidade, indicando quais os sinais que activam o processo quando se registar um evento

Um processo pode ter variáveis.

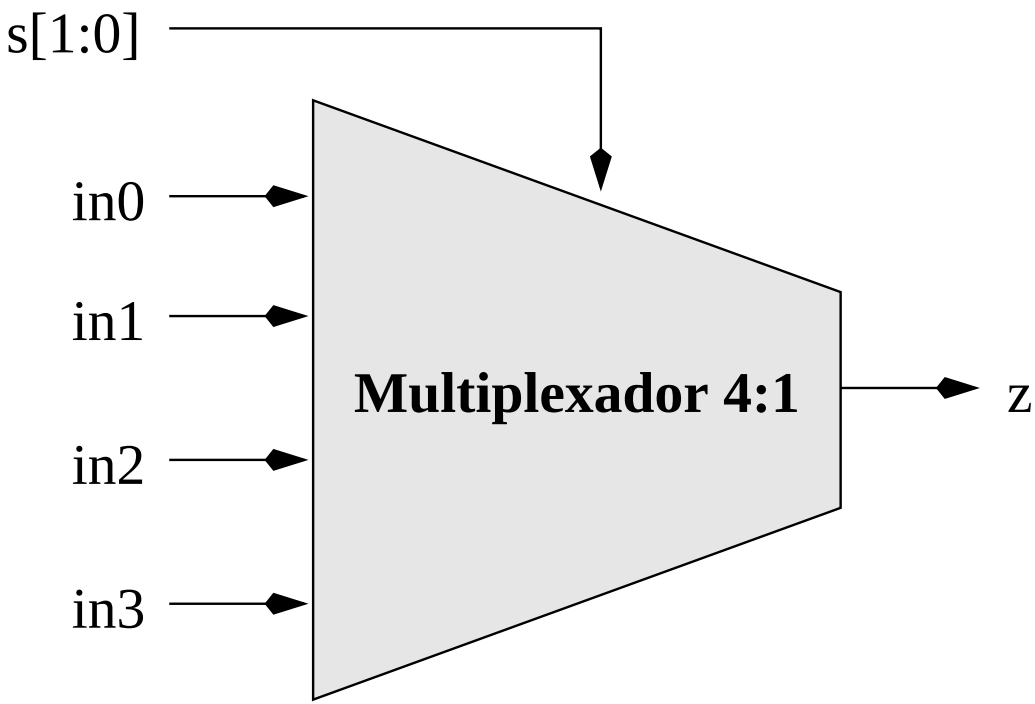
```
process (A,B)
  variable T1, T2: BIT;
begin
  .......    -- sequência de instruções
end
```

Somador Completo : descrição comportamental

```
architecture processos of somador is
  signal s1, s2, s3 : std_logic;
  constant atraso   : time := 2 ns;
begin

SS1: process (in1,in2) is
     begin
       s1  <= (in1 xor in2) after atraso;
       s3  <= (in1 and in2) after atraso;
     end process SS1;

SS2: process (s1, cin) is
     begin
       s2  <= (cin and  s1) after atraso;
       sum <= (s1  xor cin) after atraso;
     end process SS2;
O1:  cout  <= (s2  or   s3) after atraso;

end architecture processos ;
```

Multiplexador 4:1

- descrever em VHDL um **multiplexador 4:1**



Multiplexador 4:1

```
library IEEE;
use     IEEE.std_logic_1164.all;

entity mux4to1 is
  port (
    in0, in1, in2, in3 : in  std_logic;
    s : in std_logic_vector (1 downto 0);
    z                  : out std_logic );
end entity mux4to1 ;
```

Multiplexador 4:1 – descrição *data flow*

```
architecture dflow of mux4to1 is

   constant atraso : time := 2 ns;

 begin

   z <= in0 after atraso when s="00" else
        in1 after atraso when s="01" else
        in2 after atraso when s="10" else
        in3 after atraso when s="11" else
        '0' after atraso ;

end architecture dflow;
```

Multiplexador 4:1 – descrição comportamental [1]

```
architecture comportamental of mux4to1 is

    constant atraso : time := 2 ns;

  begin
    process (s, in3, in2, in1, in0) is
    begin
      if    s="00" then
        z <= in0 after atraso ;
      elsif s="01" then
        z <= in1 after atraso ;
      elsif s="10" then
        z <= in2 after atraso ;
      elsif s="11" then
        z <= in3 after atraso ;
      else
        z <= '0' after atraso ;
      end if;
    end process;

end architecture comportamental ;
```

Multiplexador 4:1 – descrição comportamental [2]

```
architecture comportamental2 of mux4to1 is
    constant atraso : time := 2 ns;
  begin
    process (s, in3, in2, in1, in0)
    begin
      case s is -- diferente s(0 to 1)
        when "00" =>
          z <= in0 after atraso ;
        when "01" =>
          z <= in1 after atraso ;
        when "10" =>
          z <= in2 after atraso ;
        when "11" =>
          z <= in3 after atraso ;
        when others =>
          z <= '0' after atraso ;
      end case;
    end process;

end architecture comportamental2 ;
```

Multiplexador 4:1 – descrição comportamental [3]

```
architecture comportamental3 of mux4to1 is
   constant atraso : time := 2 ns;
   begin
   process(s,in0,in1,in2,in3)
   variable value : integer := 0;
       begin
     if s(0)='1' then value := value + 1; end if;
     if s(1)='1' then value := value + 2; end if;
     case value is
       when 0 => z <= in0 after atraso;
       when 1 => z <= in1 after atraso;
       when 2 => z <= in2 after atraso;
       when 3 => z <= in3 after atraso;
       when others => z <= '0';
    end case;
 end process;
end architecture comportamental3 ;
```

Exercício

Pretende-se modelar um componente que indique a paridade de 3 bits, isto é, coloque a sua saída a 1 se o número de 1's nas 3 entradas for par.

1. Faça a tabela de verdade do circuito e obtenha a expressão booleana simplificada.
2. Faça a modelação *data flow* em VHDL
3. Faça a modelação comportamental
4. Descreva estruturalmente um circuito que detecte a paridade de 6 bits usando o componente desenvolvido nas alíneas anteriores.
5. Simule o detector de paridade 3 para ambas as arquitecturas.

Exercício: alínea 2

```
library IEEE;
use     IEEE.std_logic_1164.all;

entity par3 is
  port (
    e0, e1, e2 : in std_logic;
    z : out std_logic );
end entity par3 ;
```

Exercício: alínea 2

```
architecture dflow of par3 is

signal s1, s2, s3, s4 : std_logic;
signal ne0, ne1, ne2: std_logic;
constant atraso   : time := 2 ns;

begin
        n1: ne0 <= not e0 after atraso;
        n2: ne1 <= not e1 after atraso;
        n3: ne2 <= not e2 after atraso;

        m1: s1 <= (ne0 and ne1 and ne2) after atraso;
        m2: s2 <= (ne0 and e1 and e2) after atraso;
        m3: s3 <= (e0 and ne1 and e2) after atraso;
        m4: s4 <= (e0 and e1 and ne2) after atraso;

        final: z <= (s1 or s2 or s3 or s4) after atraso;

end architecture dflow ;
```

Exercício: alínea 3

```
architecture beha of par3 is
begin
process (e0,e1,e2)
variable contador,resto:integer;
begin
        contador := 0;
        if e0 = '1‘ then contador := contador+1;
        end if;
        if e1 = '1‘ then contador := contador+1;
        end if;
        if e2 = '1’ then contador := contador+1;
        end if;
        resto := contador rem 2;
        if resto=1 then z <= '0';
        else z <= '1';
        end if;
        end process;
end architecture beha ;
```

Exercício: alínea 4

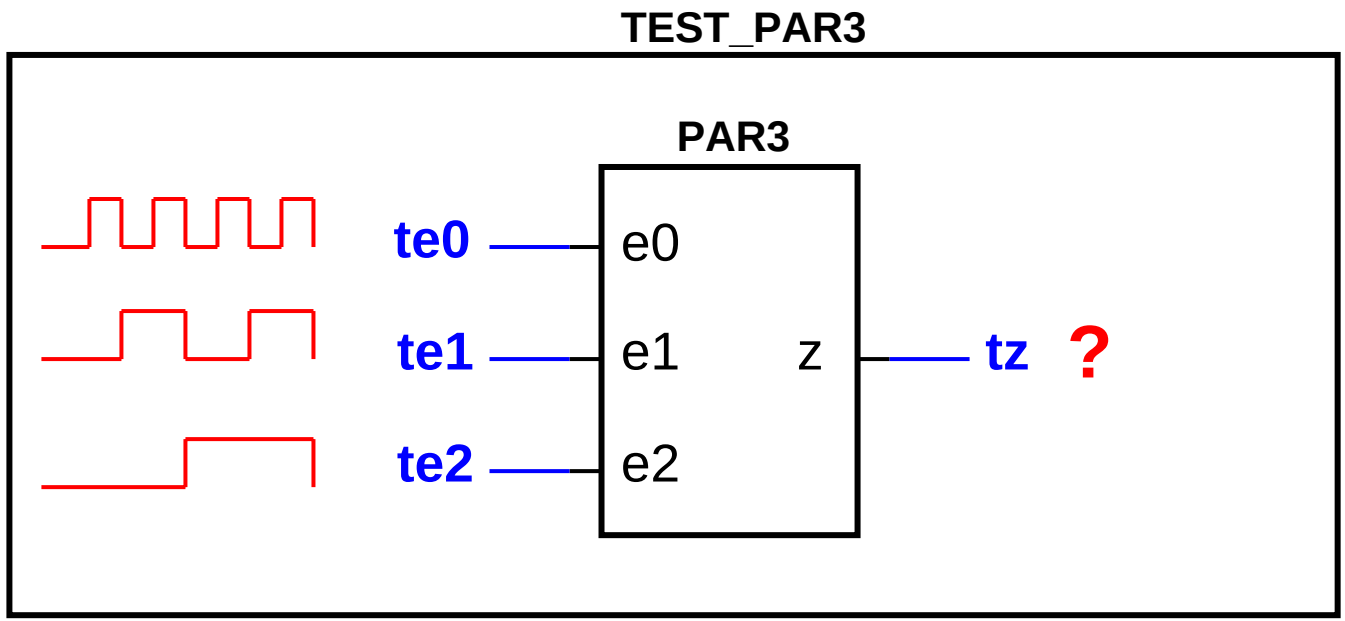
```
library IEEE;
use     IEEE.std_logic_1164.all;

entity par6 is
  port (
    e : in  std_logic_vector(0 to 5);
    par     : out std_logic );
end entity par6;

architecture estrutural of par6 is

  component par3 is
  port (
    e0, e1, e2 : in std_logic;
    z : out std_logic );
   end component par3 ;

  signal s1, s2, s3 : std_logic;
  constant atraso   : time := 2 ns;
```

```
 begin
  PAR3_1: par3
  port map (
  e0 => e(0),
  e1 => e(1),
  e2 => e(2),
  z => s1);

  PAR3_2: par3
  port map (
  e0 => e(3),
  e1 => e(4),
  e2 => e(5),
  z => s2);

  s3 <= (s1 xor s2) after atraso;
  par <= not s3 after atraso;

end architecture estrutural;
```

Exercício: alínea 5

Para fazer uma simulação é necessário desenvolver uma bancada de teste (*test bench*).

Este componente não tem entradas nem saídas, mas os seus sinais internos correspondem aos sinais de entrada/saída do componente a testar. São estes sinais que podemos modificar e observar.



Exercício: alínea 5

```
library IEEE;
use     IEEE.std_logic_1164.all;

entity test_par3 is
end entity test_par3;

architecture tstbench of test_par3 is

component par3 is
  port (
    e0, e1, e2  : in std_logic;
    z   : out std_logic );
end component par3;

  signal te0  : std_logic := '0';
  signal te1  : std_logic := '0';
  signal te2  : std_logic := '0';
  signal tz   : std_logic ;

begin
-- comportamento dos sinais de entrada
```

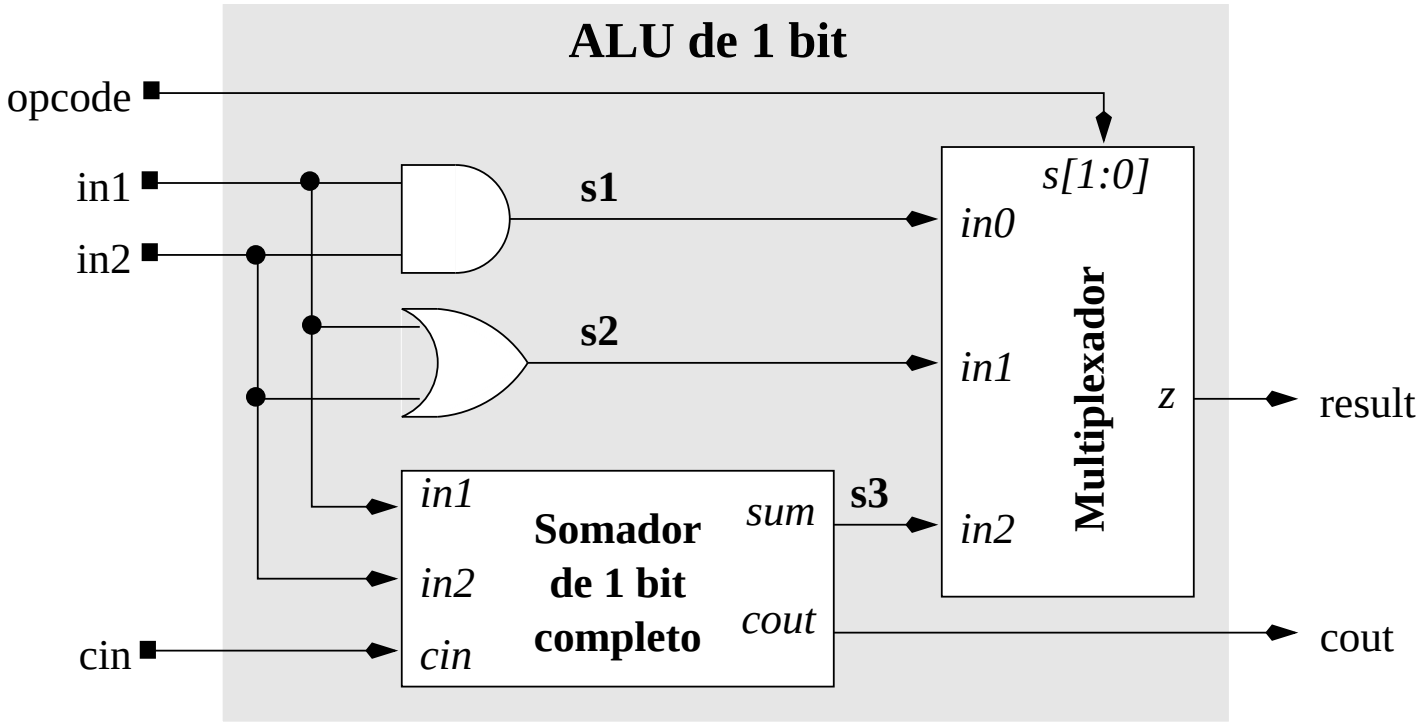
```
process (te0) is          -- te0
 begin
  if te0='1' then
     te0 <= '0' after 10ns;
  elsif te0='0' then
     te0 <= '1' after 10ns;
  end if;
end process;

process (te1) is          -- te1
 begin
  if te1='1' then
   te1 <= '0' after 20ns;
  elsif te1='0' then
   te1 <= '1' after 20ns;
  end if;
end process;

process (te2) is          -- cin
 begin
  if te2='1' then
   te2 <= '0' after 40ns;
  elsif te2='0' then
   te2 <= '1' after 40ns;
  end if;
end process;
```

Exercício 4 : *testbench*

```
-- instanciar o sistema a testar
    PROC_PAR3: entity par3(dflow)
      port map (
        e0    => te0  ,
        e1    => te1  ,
        e2    => te2  ,
         z   => tz  );

end architecture tstbench ;
```

Exercício 4 alínea 6

- descrever e simular em VHDL uma ALU de 1 bit



Exercício 4 alínea 6 : somador de 1 bit

```
library IEEE;
use     IEEE.std_logic_1164.all;

entity somador is
  port (
    in1, in2, cin : in  std_logic;
    sum, cout     : out std_logic );
end entity somador;

architecture dflow of somador is

    signal s1, s2, s3 : std_logic;
    constant atraso   : time := 2 ns;

  begin

    X1: s1   <= (in1 xor in2) after atraso;
    A1: s2   <= (cin and  s1) after atraso;
    A2: s3   <= (in1 and in2) after atraso;
    X2: sum  <= (s1  xor cin) after atraso;
    O1: cout <= (s2  or   s3) after atraso;

end architecture dflow ;
```

Exercício 4 alínea 6: multiplexador

```
library IEEE;
use     IEEE.std_logic_1164.all;

entity mux3to1 is
  port (
    in0, in1, in2 : in std_logic;
    s : in std_logic_vector (1 downto 0);
    z : out std_logic );
end entity mux3to1 ;
```

```
architecture funcional_case of mux3to1 is
   constant atraso : time := 2 ns;
  begin
    process (s, in2, in1, in0)
    begin
      case s(1 downto 0) is
        when "00" =>
          z <= in0 after atraso ;
        when "01" =>
          z <= in1 after atraso ;
        when "10" =>
          z <= in2 after atraso ;
        when others =>
          z <= '0' after atraso ;
      end case;
    end process;
end architecture funcional_case;
```