Sistemas Digitais 2

Projeto lógico combinacional
Portas lógicas, multiplexadores, somadores

Prof. Daniel M. Muñoz Arboleda

FGA - UnB

Agenda

- Projeto lógico combinacional em VHDL
  - Exemplos de descrição de portas lógicas
  - Exemplo editor de esquemáticos no ISE
  - Exemplos de descrição de portas lógicas de múltiplos bits
  - Exemplos de multiplexadores
  - Exemplos de somadores

Sistemas combinacionais e sequenciais

Um *sistema digital combinacional* é qualquer sistema digital onde o comportamento de cada saída pode ser descrito como uma função que depende exclusivamente das **combinações** de valores instantâneos das entradas do sistema. Um sistema digital combinacional pode ser totalmente descrito por uma *tabela verdade*.

Um *sistema digital sequencial* é um sistema digital que, em geral, não pode ser descrito exclusivamente pela combinação das entradas. Portanto, é um sistema digital que sob as mesmas condições possui mais de um estado, isto é, depende dos valores passados das entradas (memória).

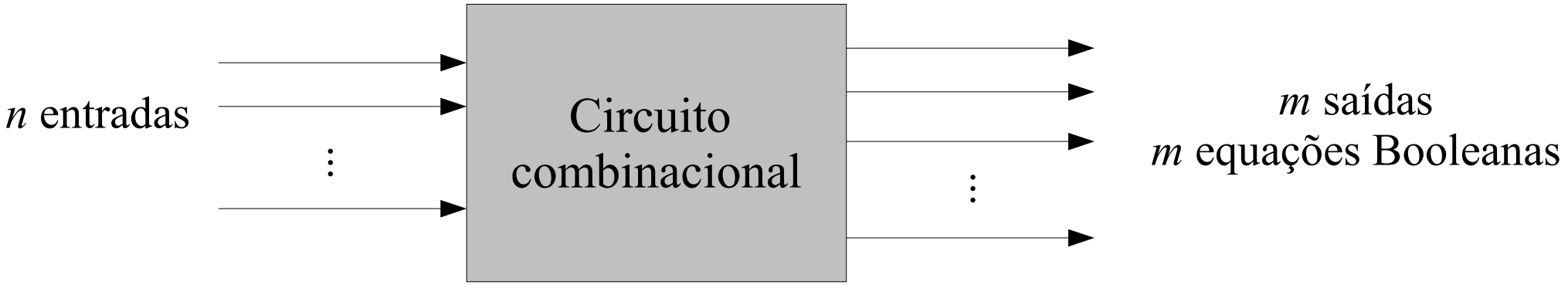
Um sistema combinacional é definido como um caso especial de um sistema sequencial.

Circuito combinacional

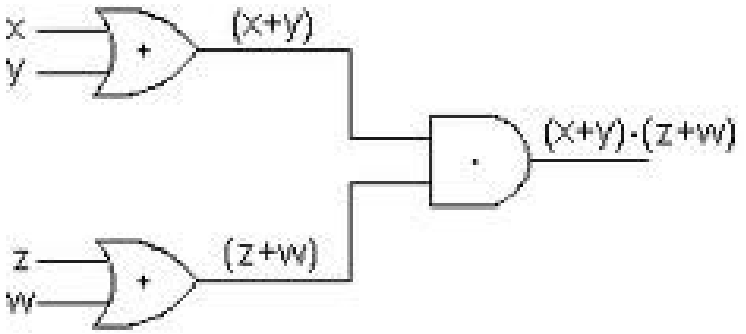
Nos circuitos lógicos combinacionais a qualquer instante o nível lógico da saída depende da combinação dos níveis lógicos presentes na entrada. Um circuito combinacional não possui características de memória e, portanto, a sua saída depende apenas do valor regular das entrada (não depende de valores passados de entradas ou saídas ou estados).

Um circuito combinacional é constituído por um conjunto de portas lógicas as quais determinam os valores das saídas diretamente a partir dos valores atuais das entradas.

Circuito combinacional



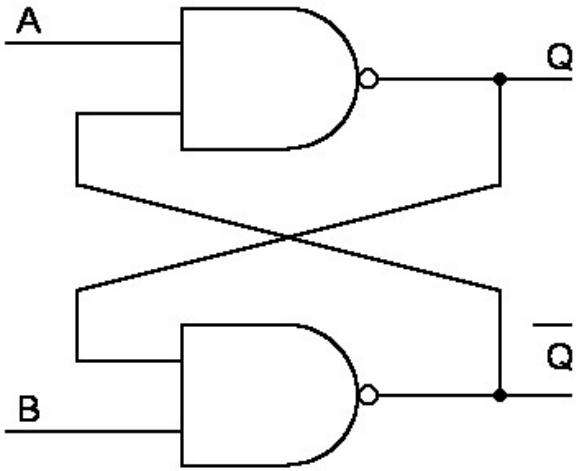
Pode-se dizer que um circuito combinacional realiza uma operação de processamento de informação a qual pode ser especificada por meio de um conjunto de equações Booleanas.



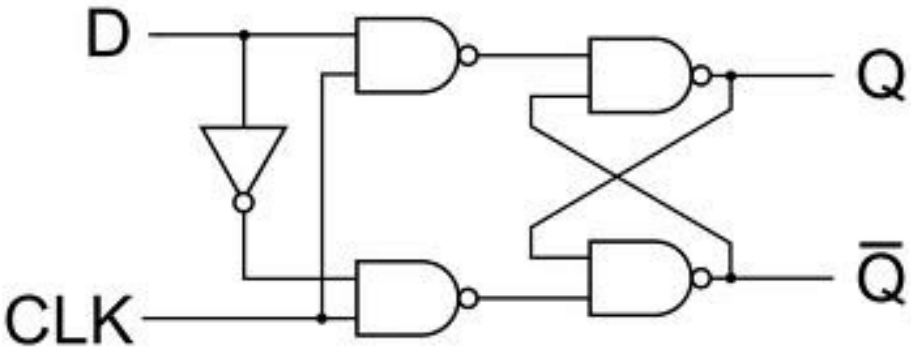
Circuito sequencial

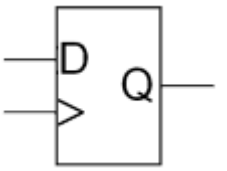
Um circuito sequencial, por sua vez, emprega elementos de armazenamento denominados latches e flip-flops, além de portas lógicas. Os valores das saídas do circuito dependem dos valores das entradas e dos estados dos latches ou flip-flops utilizados.

Latch SR



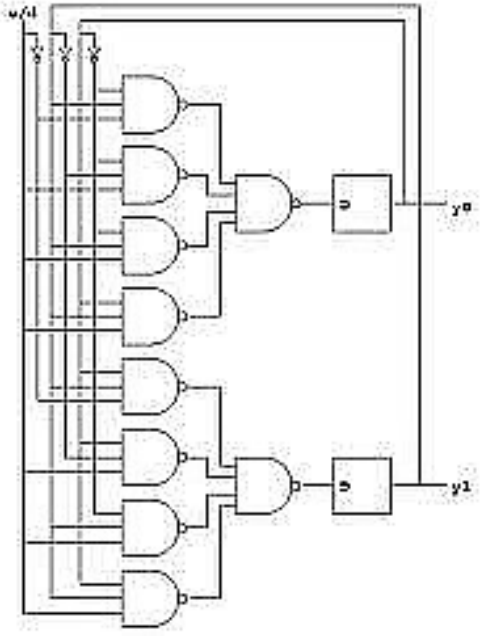
Flip-flop D

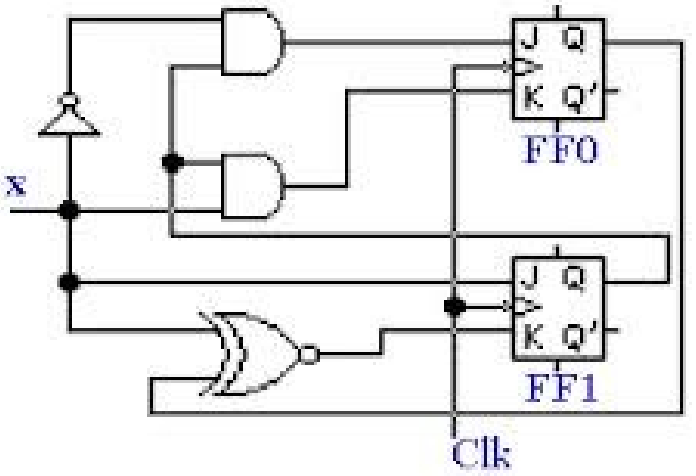




Circuito sequencial

Como o estado dos latches e flip-flops é função dos valores anteriores das entradas, diz-se que as saídas de um circuito sequencial dependem dos valores das entradas e do histórico do próprio circuito. Logo, o comportamento de um circuito sequencial é especificado pela sequência temporal das entradas e de seus estados internos.



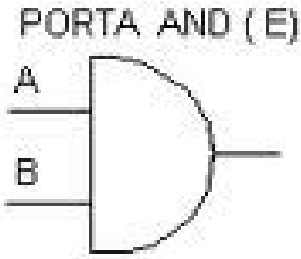


Agenda

- Projeto lógico combinacional em VHDL
  - **Exemplos de descrição de portas lógicas**
  - Exemplo editor de esquemáticos no ISE
  - Exemplos de descrição de portas lógicas de múltiplos bits
  - Exemplos de multiplexadores
  - Exemplos de somadores

Descrição VHDL porta AND2

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   entity and2 is
5   port(
6         a, b: in std_logic;
7        s    : out std_logic);
8   end and2;
9
10  architecture comportamental of and2 is
11  begin
12        s <= a and b;
13
14  end comportamental;
15
16
```

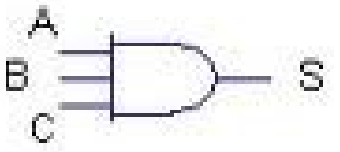


X = A . B

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Descrição VHDL porta AND3

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   entity and3 is port(
5       a, b, c: in std_logic;
6       s: out std_logic);
7   end and3;
8
9   architecture comportamental of and3 is
10  begin
11        s <= (a and b)and c;
12
13  end comportamental;
14
```



A	B	C	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Descrição VHDL porta NOT

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   entity inv is port(
5       a: in std_logic;
6       s: out std_logic);
7   end inv;
8
9   architecture comportamental of inv is
10  begin
11        s <= not a;
12
13  end comportamental;
14
```

Descrição VHDL porta NAND2

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   entity nand2 is port(
5       a, b: in std_logic;
6       s: out std_logic);
7   end nand2;
8
9   architecture comportamental of nand2 is
10  begin
11        s <= a nand a;
12
13  end comportamental;
14
```

Descrição VHDL porta XNOR2

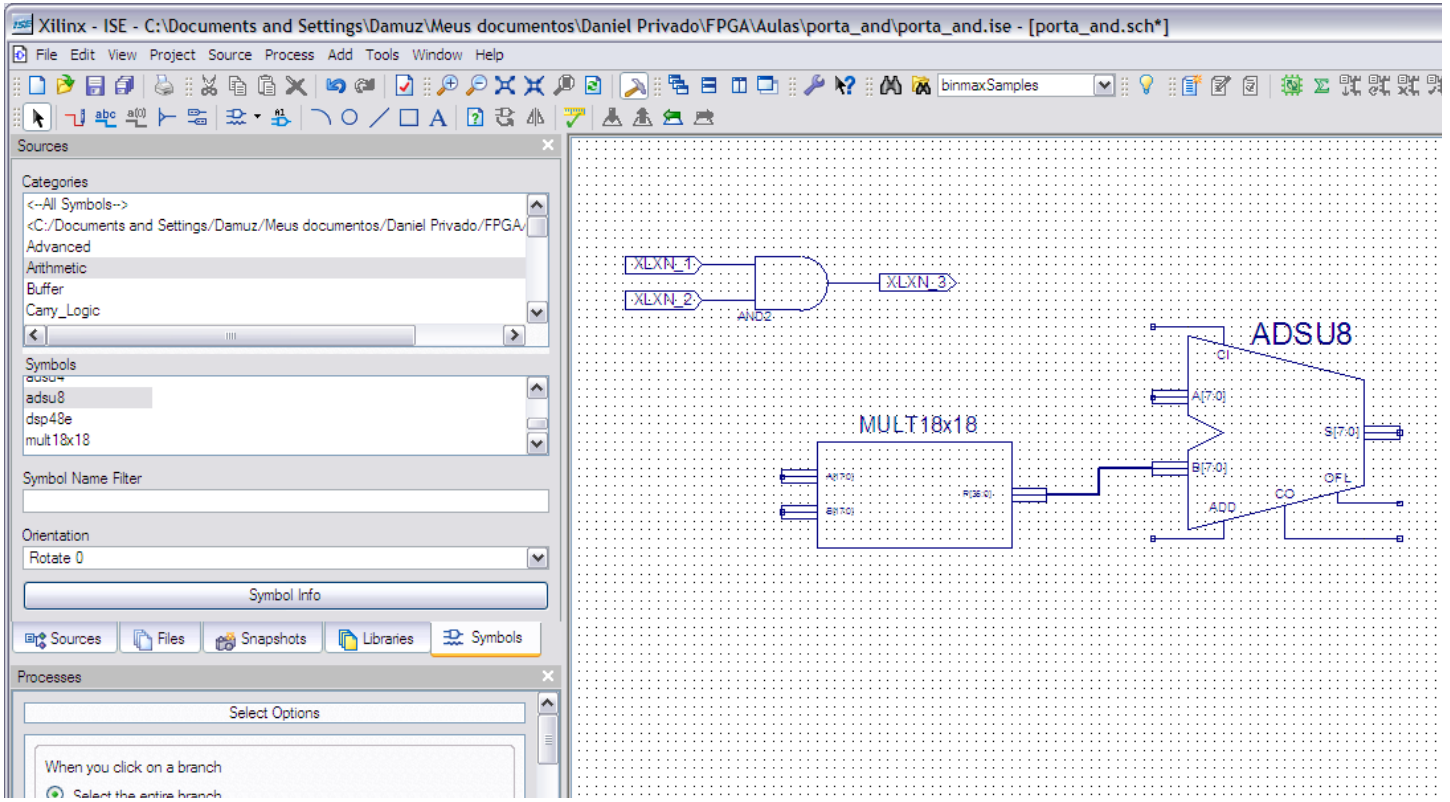
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  entity xnor2 is port(
5      a, b: in std_logic;
6      s: out std_logic);
7  end xnor2;
8
9  architecture comportamental of xnor2 is
10 begin
11      s <= a xnor b;
12
13 end comportamental;
14
```

Agenda

- Projeto lógico combinacional em VHDL
  - Exemplos de descrição de portas lógicas
  - **Exemplo editor de esquemáticos no ISE**
  - Exemplos de descrição de portas lógicas de múltiplos bits
  - Exemplos de multiplexadores
  - Exemplos de somadores

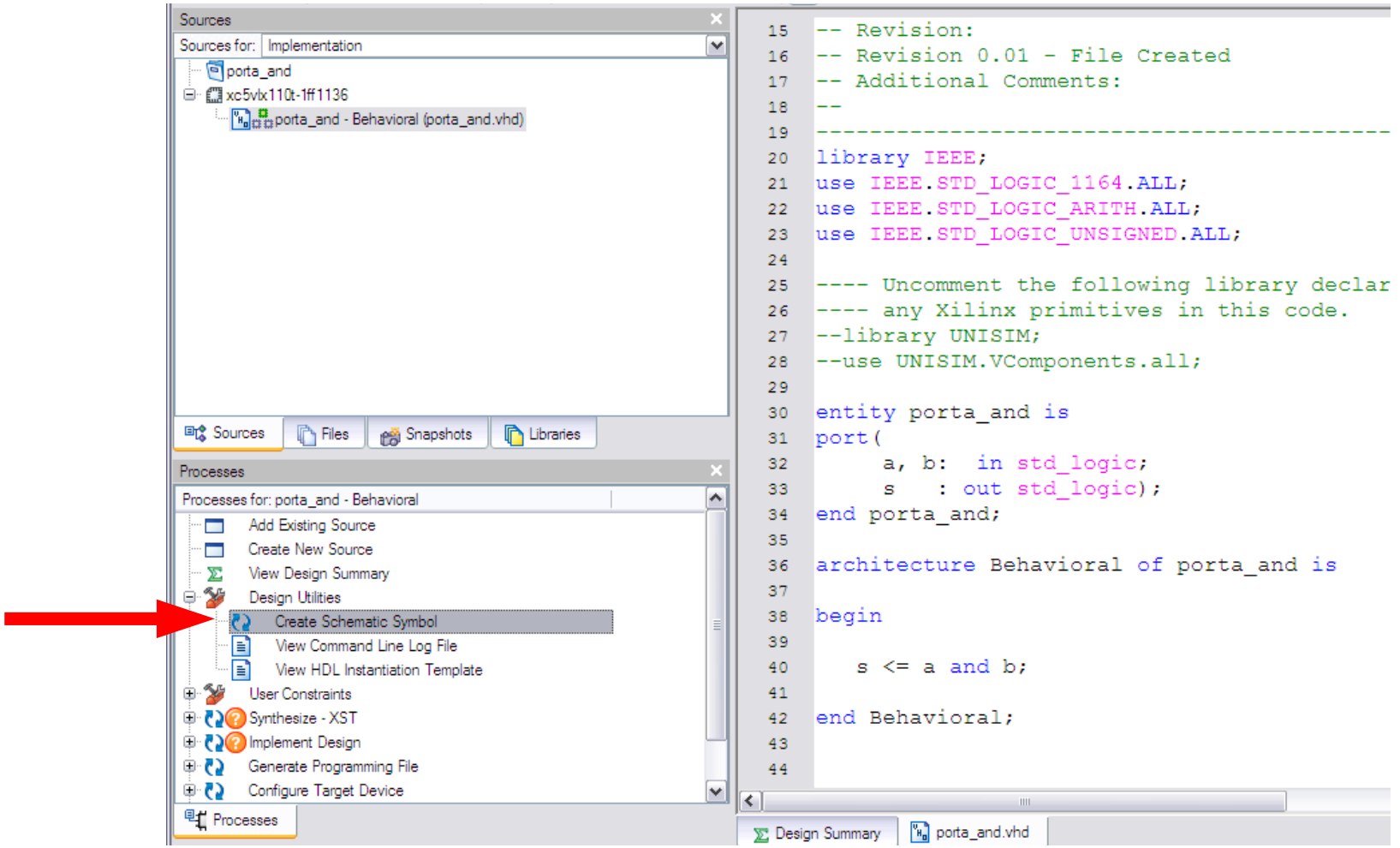
Desenho lógico combinacional em VHDL

Projeto 3. Editor de esquemáticos do ISE



Desenho lógico combinacional em VHDL

Projeto 3. Criação de símbolos esquemáticos



Agenda

- Projeto lógico combinacional em VHDL
  - Exemplos de descrição de portas lógicas
  - Exemplo editor de esquemáticos no ISE
  - **Exemplos de descrição de portas lógicas de múltiplos bits**
  - Exemplos de multiplexadores
  - Exemplos de somadores

Descrição VHDL porta AND2 de 8 bits

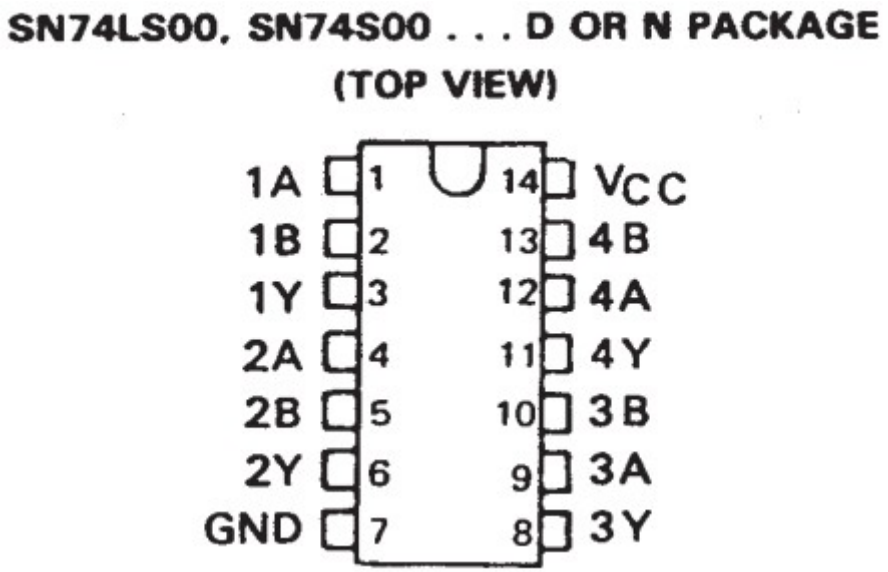
```
1  entity porta_and2_vetor is port(
2      a, b: in std_logic_vector(7 downto 0);
3      x: out std_logic_vector(7 downto 0));
4  end porta_and2_vetor;
5
6
7  architecture behavioral of porta_and2_vetor is
8  begin
9      x <= a and b;
10 end behavioral;
11
```

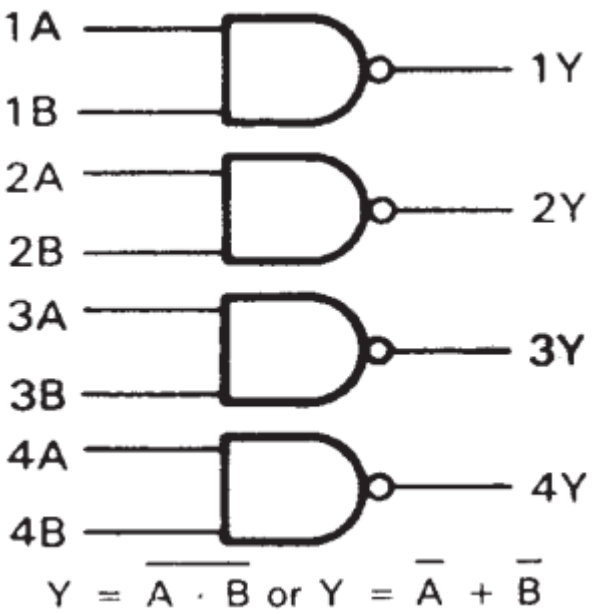
Descrição VHDL porta AND2 de 8 bits

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  entity and_vector is port(
5      a, b: in bit_vector (7 downto 0);
6      s: out bit_vector (7 downto 0));
7  end and_vector;
8
9  architecture comportamental of and_vector is
10 begin
11
12     process (a,b)
13     begin
14       for i in 7 downto 0 LOOP
15         s(i) <= a(i) and b(i);
16       end LOOP;
17     end process;
18
19 end comportamental;
20
```

Nota: o conceito de processo será apresentado no projeto lógico sequencial.
Processo é um trecho de código sequencial que é executado quando um elemento da **lista sensível** muda de valor.

Exemplo VHDL chip TTL SN7400 – Quadruple 2-input NAND gates





Serão apresentadas duas soluções:
1) Entradas e saídas de um bit *std_logic*;
2) Entradas e saídas como vetor de 4 bits *std_logic_vector (3 downto 0)*;
Finalmente, será realizado o testbench e a simulação do circuito.

Chip TTL SN7400 – solução 1: entradas e saídas de um bit

```
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23
24  entity SN7400_bit is
25      Port ( a1 : in  STD_LOGIC;
26             b1 : in  STD_LOGIC;
27             y1 : out  STD_LOGIC;
28             a2 : in  STD_LOGIC;
29             b2 : in  STD_LOGIC;
30             y2 : out  STD_LOGIC;
31             a3 : in  STD_LOGIC;
32             b3 : in  STD_LOGIC;
33             y3 : out  STD_LOGIC;
34             a4 : in  STD_LOGIC;
35             b4 : in  STD_LOGIC;
36             y4 : out  STD_LOGIC);
37  end SN7400_bit;
38
39  architecture Behavioral of SN7400_bit is
40
41  begin
42
43  y1 <= a1 nand b1;
44  y2 <= a2 nand b2;
45  y3 <= a3 nand b3;
46  y4 <= a4 nand b4;
47
48  end Behavioral;
```

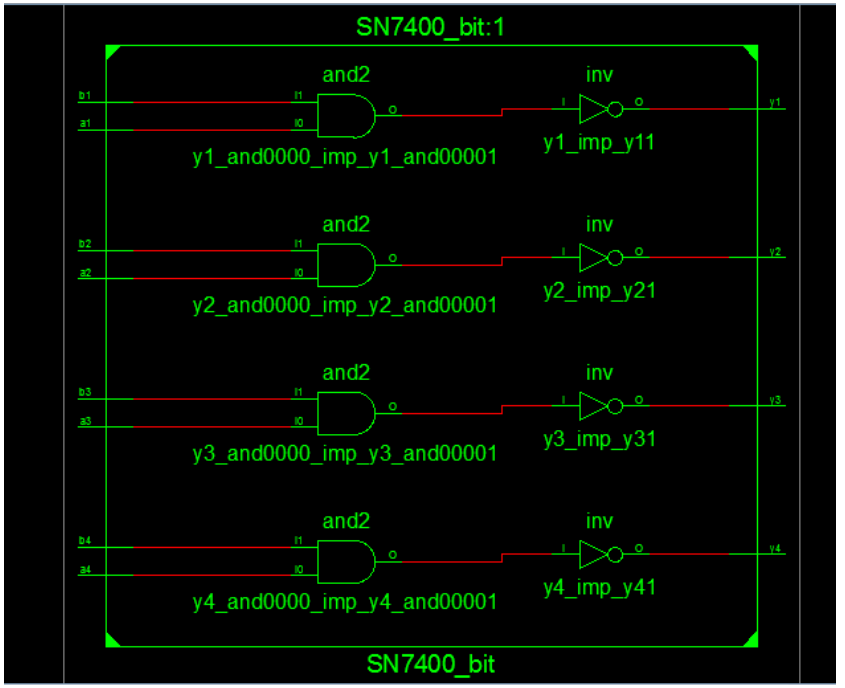


Diagrama RTL Schematic
Obtido após a síntese lógica

Chip TTL SN7400 – solução 2: entradas e saídas como vetor de 4 bits

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity SN7400 is
5      Port ( a : in  STD_LOGIC_VECTOR(3 downto 0);
6             b : in  STD_LOGIC_VECTOR(3 downto 0);
7             y : out  STD_LOGIC_VECTOR(3 downto 0));
8  end SN7400;
9
10 architecture Behavioral of SN7400 is
11
12 begin
13
14 y <= a nand b;
15
16 end Behavioral;
```

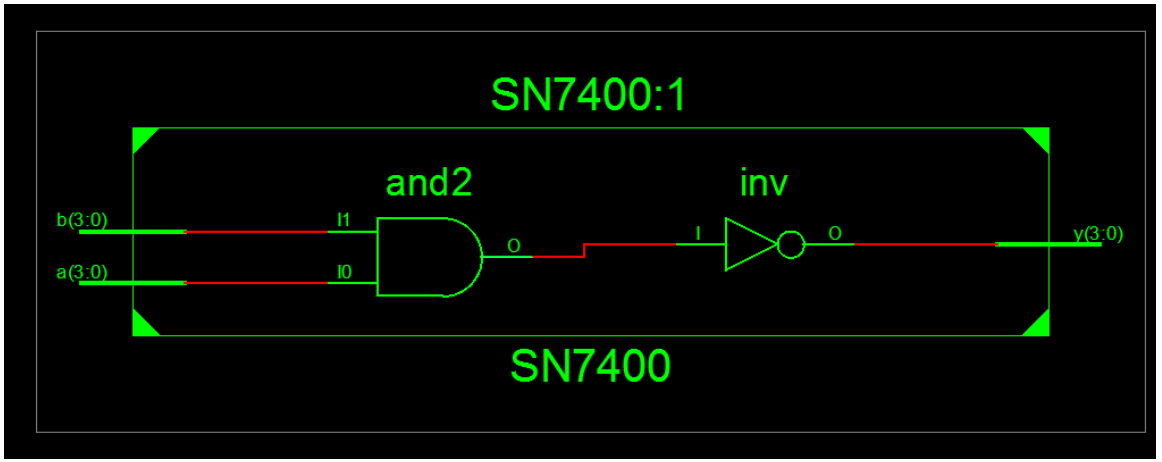


Diagrama RTL Schematic

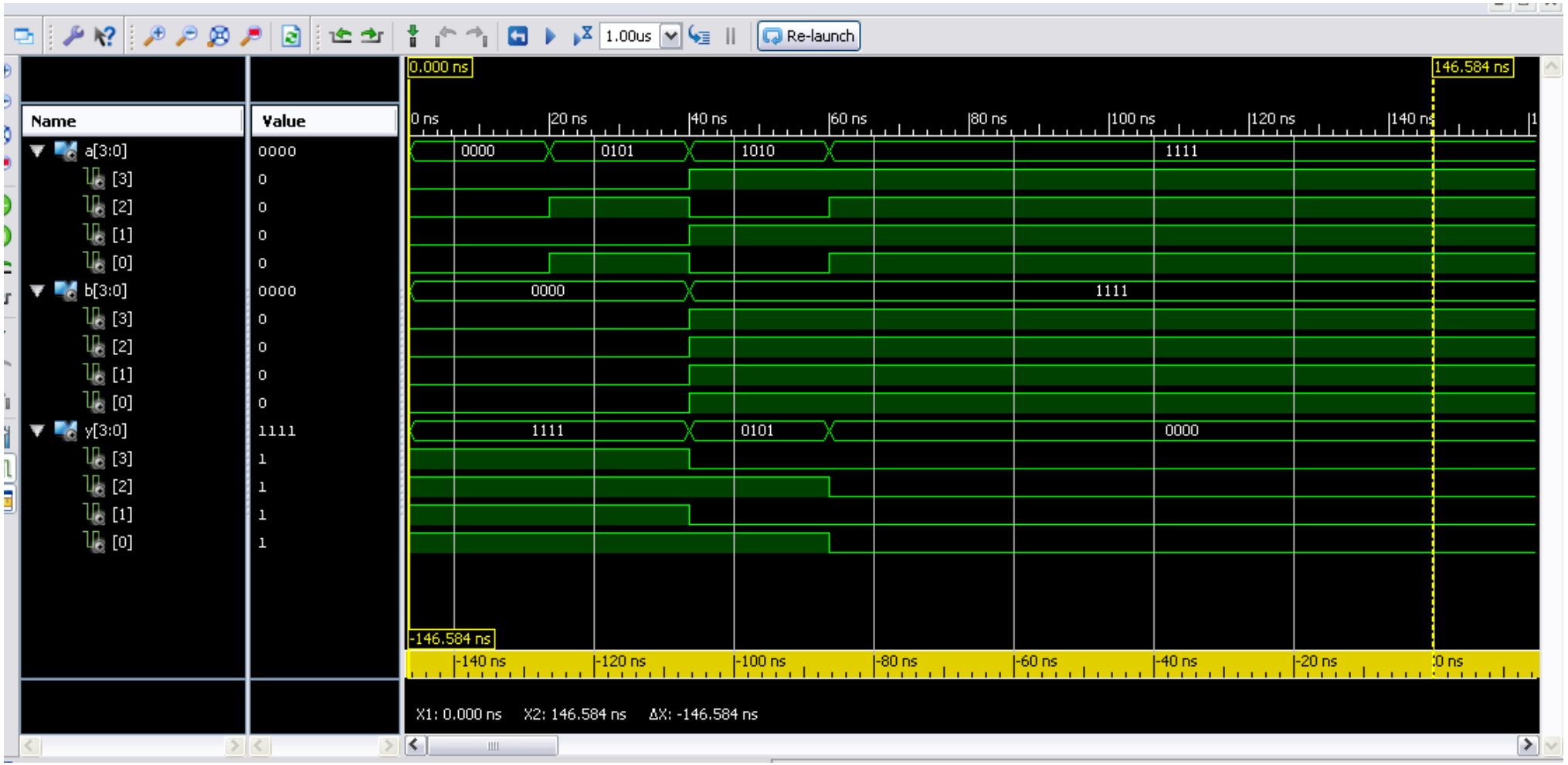
Observe que a ferramenta de síntese entende as entradas e saídas como vetores.

A ferramenta inferiu uma porta and2 e uma porta not.

Chip TTL SN7400 – testbench e simulação (solução 2)

```
51  ENTITY tb_sn7400 IS
52  END tb_sn7400;
53
54  ARCHITECTURE behavior OF tb_sn7400 IS
55
56      COMPONENT SN7400 -- declaracao do componente
57      PORT(
58           a : IN  std_logic_vector(3 downto 0);
59           b : IN  std_logic_vector(3 downto 0);
60           y : OUT  std_logic_vector(3 downto 0)
61          );
62      END COMPONENT;
63
64     -- declaracao dos sinais de entrada e saida
65     signal a : std_logic_vector(3 downto 0) := (others => '0');
66     signal b : std_logic_vector(3 downto 0) := (others => '0');
67     signal y : std_logic_vector(3 downto 0);
68
69  BEGIN
70     uut: SN7400 PORT MAP ( -- instanciacao do componente
71           a => a,
72           b => b,
73           y => y
74          );
75    -- estímulos de entrada nas portas a e b
76    a <= "0000", "0101" after 20 ns, "1010" after 40 ns, "1111" after 60 ns;
77    b <= "0000", "1111" after 40 ns;
78
79  END;
80
```

Chip TTL SN7400 – testbench e simulação (solução 2)



Agenda

- Projeto lógico combinacional em VHDL
  - Exemplos de descrição de portas lógicas
  - Exemplo editor de esquemáticos no ISE
  - Exemplos de descrição de portas lógicas de múltiplos bits
  - **Exemplos de multiplexadores**
  - Exemplos de somadores

Descrição VHDL de multiplexadores (exemplo 1)

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   entity mux1 is port(
5       a, b, sel: in std_logic;
6       s: out std_logic);
7   end mux1;
8
9   architecture comportamental of mux1 is
10  begin
11      with sel select
12          s <= a when '0',
13               b when others;
14
15  end comportamental;
16
```

Descrição VHDL de multiplexadores (exemplo 2)

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   entity mux2 is port(
5       a, b: in bit_vector (7 downto 0);
6       sel: in std_logic;
7       s: out bit_vector (7 downto 0));
8   end mux2;
9
10  architecture comportamental of mux2 is
11  BEGIN
12      with sel select
13          s <= a when '0',
14               b when '1';
15
16  END comportamental;
17
```

Descrição VHDL de multiplexadores (exemplo 3)

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   entity mux3 is port(
5       a, b, c, d: in bit_vector (7 downto 0);
6       sel: in bit_vector (1 downto 0);
7       s: out bit_vector (7 downto 0));
8   end mux3;
9
10  architecture comportamental of mux3 is
11  BEGIN
12      with sel select
13          s <= a when "00",
14               b when "01",
15               c when "10",
16               d when others;
17
18  END comportamental;
19
```

Descrição VHDL de multiplexadores (exemplo 4)

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   entity mux4 is port(
5       a, b, c, d: in bit_vector (7 downto 0);
6       sel: in bit_vector (1 downto 0);
7       s: out bit_vector (7 downto 0));
8   end mux4;
9
10  architecture comportamental of mux4 is
11  BEGIN
12          s <= a when sel = "00" else
13               b when sel = "01" else
14               c when sel = "10" else
15               d;
16
17  END comportamental;
18
```

Descrição VHDL de multiplexadores (exemplo 5)

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  entity mux5 is port(
5      a, b, c, d: in bit_vector (7 downto 0);
6      sel: in bit_vector (1 downto 0);
7      s: out bit_vector (7 downto 0);
8      s2 : out bit_vector(7 downto 0));
9  end mux5;
10
11 architecture comportamental of mux5 is
12
13 BEGIN
14     process (a,b,c,d,sel)
15     begin
16       if sel = "00"  then
17             s <= a;
18       elsif sel = "01" then
19             s <= b;
20       elsif sel = "10" then
21             s <= c;
22       else
23             s <= d;
24       end if;
25     end process;
26
```

```
27 process (sel)
28   s2<= c-d;
29 end process;
30 END comportamental;
31
```

Aqui temos dois processos!

Embora a execução de cada bloco (processo) é sequencial, o bloco como um todo é **concorrente** com outros blocos sequenciais ou códigos combinacionais externos (*with, when, select*).

Códigos concorrentes são executados de forma **simultânea**. Também é chamado de ***data flow*** (fluxo de dados).

Descrição VHDL de um decodificador binário a 7 segmentos

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE work.data_types.all;
4
5  entity mux_7_seg is port(
6      a, b   : in std_logic_vector (MAX downto 0);
7      sel    : in std_logic;
8      SEG_A, SEG_B, SEG_C, SEG_D, SEG_E, SEG_F, SEG_G, DP: out std_logic);
9  end mux_7_seg;
10
11 architecture comportamental of mux_7_seg is
12 -------signal definitions-----------------------------
13 signal mux_output: std_logic_vector (MAX downto 0);
14 signal RES : std_logic_vector(Segment_Number downto 0);
15 ------------------------------------------------------
16 BEGIN
17     process (a,b,sel)
18     begin
19       if sel = '0'  then
20           mux_output <= a;
21       elsif sel = '1' then
22           mux_output <= b;
23       end if;
24     end process;
25
```

Processo1 implementa um multiplexador.

Descrição VHDL de um decodificador binário a 7 segmentos

```
26  process(mux_output)
27  begin
28      case mux_output is
29          when "0000" => RES <= "11000000";
30          when "0001" => RES <= "11111001";
31          when "0010" => RES <= "10100100";
32          when "0011" => RES <= "10110000";
33          when "0100" => RES <= "10011001";
34          when "0101" => RES <= "10010010";
35          when "0110" => RES <= "10000010";
36          when "0111" => RES <= "11111000";
37          when "1000" => RES <= "10000000";
38          when "1001" => RES <= "10010000";
39          when "1010" => RES <= "10001000";
40          when "1011" => RES <= "10000011";
41          when "1100" => RES <= "11000110";
42          when "1101" => RES <= "10100001";
43          when "1110" => RES <= "10000110";
44          when "1111" => RES <= "10001110";
45          when others => RES <= "11111111";
46      end case;
47  end process;
48
49  SEG_A <= RES(0);
50  SEG_B <= RES(1);
51  SEG_C <= RES(2);
52  SEG_D <= RES(3);
53  SEG_E <= RES(4);
54  SEG_F <= RES(5);
55  SEG_G <= RES(6);
56  DP <= RES(7);
57  END comportamental;
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  Package data_types is
5      constant Max : integer := 3;
6      constant Segment_Number: integer := 7;
7  end data_types;
8  ------------------------------------------------
```

Processo2 implementa o decodificador em função do sinal intermediário *mux_output* que representa a saída do mulpilexador.

O processos1 (mulpilexador), o processo2 (decodificador) e as atribuições das portas de saída são executados de forma concorrente (simultânea).

Decodificador binário a 7 segmentos por componentes

```
3   library IEEE;
4   use ieee.std_logic_1164.all;
5   use ieee.std_logic_unsigned.all;
6   USE work.data_types.all;
7
8   entity COMPONENTES is
9   port    (
10      data_1, data_2  : in std_logic_vector (MAX downto 0);
11      control   : in std_logic;
12      D1A, D1B, D1C, D1D, D1E, D1F, D1G, DP1 : out std_logic);
13
14  end COMPONENTES;
15
16  architecture ESTRUTURA of COMPONENTES is
17  ---------------signals declaration--------------
18  signal mux_output : std_logic_vector(3 downto 0);
19
20  ------------Components declaration---------------
21
22  component Mux5
23  port (
24      a, b  : in std_logic_vector (MAX downto 0);
25      sel   : in std_logic;
26      s: out std_logic_vector (MAX downto 0)
27  );
28  end component;
29
30  component BIN_7SEG
31  port    (
32      BIN : in std_logic_vector(3 downto 0);
33      SEG_A, SEG_B, SEG_C, SEG_D, SEG_E, SEG_F, SEG_G, DP : out std_logic
34      );
35  end component;
```

Decodificador binário a 7 segmentos por componentes

```
37  --------------Maping the instances---------------
38  begin
39  Mux : Mux5 port map (a => data_1,
40                       b => data_2,
41                       sel => control,
42                       s => mux_output);
43
44  conv1 : BIN_7SEG port map (BIN => mux_output,
45                             SEG_A => D1A,
46                             SEG_B => D1B,
47                             SEG_C => D1C,
48                             SEG_D => D1D,
49                             SEG_E => D1E,
50                             SEG_F => D1F,
51                             DP    =>DP1);
52  end ESTRUTURA;
53
```

Componentes instanciam blocos de código VHDL que podem ser implementados com lógica sequencial e/ou combinacional.

A saída de um componentes **pode** ser a entrada de outro componente (veja sinal *mux_output*)

Os componentes são executados de forma concorrente.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  Package data_types is
5     constant Max : integer := 3;
6     constant Segment_Number: integer := 7;
7  end data_types;
8  ------------------------------------------------
```

Descrição VHDL de um comparador

```
25  entity comparador is
26  port (
27      A   : in std_logic_vector(3 downto 0);
28      B   : in std_logic_vector(3 downto 0);
29      AlB : out std_logic;
30      AeB : out std_logic;
31      AhB : out std_logic
32  );
33  end comparador;
34
35  architecture behavioral of comparador is
36  begin -- architecture body
37
38      process(A,B)
39      begin
40          if A=B then
41              AlB <= '0';
42              AeB <= '1';
43              AhB <= '0';
44          elsif A<B then
45              AlB <= '1';
46              AeB <= '0';
47              AhB <= '0';
48          else
49              AlB <= '0';
50              AeB <= '0';
51              AhB <= '1';
52          end if;
53      end process;
54
55  end behavioral;
```

Agenda

- Projeto lógico combinacional em VHDL
  - Exemplos de descrição de portas lógicas
  - Exemplo editor de esquemáticos no ISE
  - Exemplos de descrição de portas lógicas de múltiplos bits
  - Exemplos de multiplexadores
  - **Exemplos de somadores**

Descrição VHDL somador 8 bits (exemplo 1)

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4
5   entity somador is port(
6       a, b: in unsigned (7 downto 0);
7       s: out std_logic_vector(7 downto 0));
8   end somador;
9
10  architecture comportamental of somador is
11  signal result : integer;
12  begin
13
14      result <= CONV_INTEGER(a) + CONV_INTEGER(b);
15      s <= CONV_STD_LOGIC_VECTOR(result, 8);
16  end comportamental;
17
```

Descrição VHDL somador 8 bits (exemplo 2)

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  USE ieee.std_logic_signed.all;
5  USE work.math.all;
6
7  entity somadorB is port(
8      a, b: in  std_logic_vector(15 downto 0);
9      s: out std_logic_vector(15 downto 0));
10 end somadorB;
11
12 architecture comportamental of somadorB is
13 signal result1: integer;  -- signed
14 signal result2: integer;  -- signed
15 signal result3: integer;  -- signed
16 begin
17
18    result1 <= vect_to_int(a);
19    result2 <= vect_to_int(b);
20    result3 <= result1 + result2;
21    s <= int_to_st16 (result3);
22 end comportamental;
23
```

Descrição VHDL somador 8 bits (exemplo 3)

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  USE ieee.std_logic_signed.all;
5  USE work.math.all;
6
7  entity somador is port(
8      a, b: in std_logic_vector(15 downto 0);
9      s: out  std_logic_vector(15 downto 0));
10 end somador;
11
12 architecture comportamental of somador is
13 signal result1: signed(15 downto 0);  -- signed
14 signal result2: signed(15 downto 0);  -- signed
15 signal result3: signed(15 downto 0);  -- signed
16 begin
17
18    result1 <= signed(a);
19    result2 <= signed(b);
20    result3 <= result1 + result2;
21    s <= std_logic_vector( result3);
22 end comportamental;
23
```

Como evitar o *overflow* do somador?

Sugestão video aulas

• Implementação em VHDL de um meio somador subtrator usando lógica combinacional (2 vídeos, 17 minutos em total):

https://www.youtube.com/watch?v=v-CXIrh1S78&list=PLKIWpQ56tY7KeqdSf36lrdsvTm2TGvdFq&index=1
https://www.youtube.com/watch?v=L5T2Dx7JOII&list=PLKIWpQ56tY7KeqdSf36lrdsvTm2TGvdFq&index=2

• Implementação em VHDL de um somador subtrator completo usando lógica combinacional (2 vídeos, 13 minutos em total):

https://www.youtube.com/watch?v=gbPNWr6yyDo&list=PLKIWpQ56tY7KeqdSf36lrdsvTm2TGvdFq&index=3
https://www.youtube.com/watch?v=2UmYzbXPPlM&list=PLKIWpQ56tY7KeqdSf36lrdsvTm2TGvdFq&index=4