Estrutura do VHDL

- **Entity**: declaração de portas de entrada e saída
- **Architecture**: comportamento interno ou estrutura do módulo
- A arquitetura de uma entidade pode instanciar outras entidades

Entidade

Nome da entidade: definido pelo usuário

```
entity entity-name is
  port (signal-names : mode signal-type;
        signal-names : mode signal-type;
        ...
        signal-names : mode signal-type);
end entity-name;
```

Nome do sinal

Tipo do sinal

Direção do sinal: in, out, buffer, inout

x	y	z = x->y
0	0	1
0	1	1
1	0	0
1	1	1

```
-- Definicao da porta implica --
entity implica is
    Port ( x : in  STD_LOGIC; -- definicao da estrutura
           y : in  STD_LOGIC; -- de entradas e saidas
           z : out  STD_LOGIC);
end implica;
```

Fonte: Wakerly, J. ***Digital Design Principles and Practices***, 4th ed.

Arquitetura

Nome da arquitetura: relacionado ao nome da entidade

Definição de funções, constantes, procedimentos, sinais e tipos de sinais a serem utilizados

```
architecture architecture-name of entity-name is
   type declarations
   signal declarations
   constant declarations
   function definitions
   procedure definitions
   component declarations
begin
   concurrent-statement
   ...
   concurrent-statement
end architecture-name;
```

Deve ser o mesmo da entidade

Relações entre os sinais de entrada e saída do circuito

```
architecture Behavioral of implica is
begin
   z <=  '0' when x='1' and y='0' else
         '1';
end Behavioral;
```

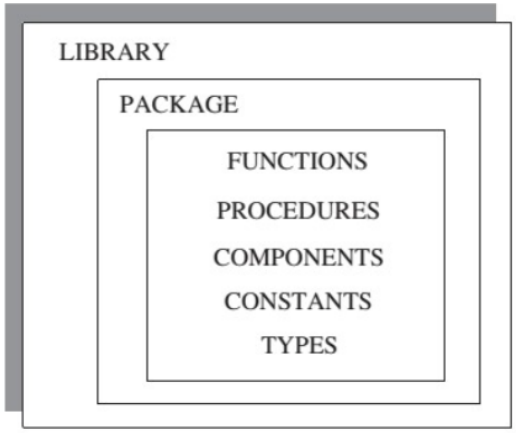
Fonte: Wakerly, J. ***Digital Design Principles and Practices***, 4th ed.

Bibliotecas

- **Library**: bibliotecas (*ieee*, *std*, *work*, etc.)
- **Library Declaration**: nome da biblioteca e a diretiva **use** para indicar quais pacotes (*packages*) serão usados

```
LIBRARY library_name;
USE library_name.package_name.package_parts;
```

Comumente, 3 *packages* são usados: ***ieee.std_logic_1164, standard, work***



Pacotes

- **std_logic_1164** especifica 8 ou 9 níveis lógicos. Possui pacotes internos. Ex:
  - **std_logic_arith**: tipos de dados **signed** e **unsigned**, funções de conversão de dados, tais como:
    - *conv_integer(p)*
    - *conv_unsigned(p,b)*
    - *conv_signed(p,b)*
    - *conv_std_logic_vector(p,b)*
  - **std_logic_signed(unsigned)**: funções que permitem a operação com dados std_logic_vector do tipo **signed(unsigned)**

Test Bench

Não há declaração de portas

Declara o UUT como componente

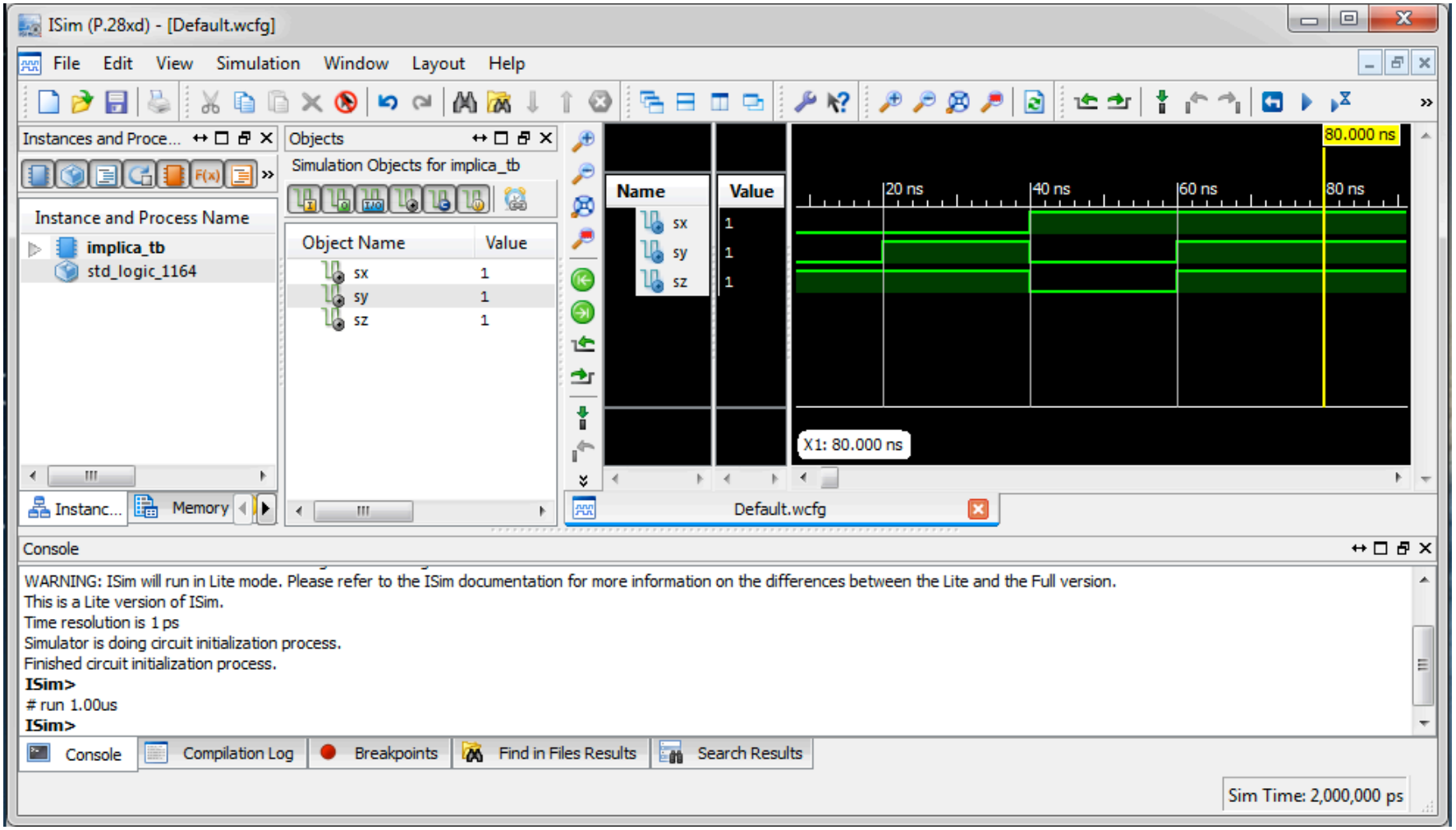
Cria sinais para entradas e saídas

Instancia o componente usando *port map*

Gera as combinações dos bits de entrada

```
ENTITY implica_tb IS
END implica_tb;

ARCHITECTURE behavior OF implica_tb IS

    COMPONENT implica
    PORT(x : IN  std_logic;
         y : IN  std_logic;
         z : OUT  std_logic);
    END COMPONENT;

   signal sx, sy, sz : std_logic;

BEGIN
   uut: implica PORT MAP (
          x => sx,
          y => sy,
          z => sz);

   stim_proc: process
   begin
      sx <= '0'; sy <= '0';
      wait for 20 ns;
      sx <= '0'; sy <= '1';
      wait for 20 ns;
      sx <= '1'; sy <= '0';
      wait for 20 ns;
      sx <= '1'; sy <= '1';
      wait;
   end process;
END;
```

Simulação


ISim (P.28xd) - [Default.wcfg]
File Edit View Simulation Window Layout Help
Instances and Proce...
Instance and Process Name
implica_tb
std_logic_1164
Objects
Simulation Objects for implica_tb
Object Name
Value
sx 1
sy 1
sz 1
Name
Value
80.000 ns
20 ns
40 ns
60 ns
80 ns
X1: 80.000 ns
Default.wcfg
Console
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
run 1.00us
ISim>
Console
Compilation Log
Breakpoints
Find in Files Results
Search Results
Sim Time: 2,000,000 ps

Codificação VHDL: Estrutural, *Dataflow*, Comportamental

VHDL Estrutural

- Define a estrutura de interconexão entre os módulos (como um esquemático ou ***netlist***)
- Declaração e instanciação de componentes
- ***port map***: associa portas da entidade instanciada aos sinais da arquitetura em uso

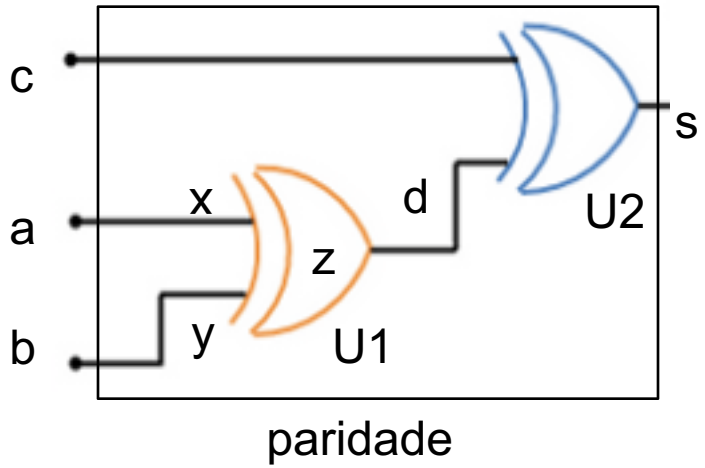
VHDL Estrutural

- Exemplo: circuito de **detecção de paridade de 3 bits** (saída é 1 se houver um número ímpar de entradas iguais a 1) composto de portas **ou-exclusivo**
- Porta ou-exclusivo de 2 entradas:

```
entity xor2 is
    Port ( x : in  STD_LOGIC;
           y : in  STD_LOGIC;
           z : out  STD_LOGIC);
end xor2;

architecture Behavioral of xor2 is

begin
   z <= (not(x) and y) or (x and not(y));
end Behavioral;
```

VHDL Estrutural

- Detector de paridade:




```
entity paridade is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : in  STD_LOGIC;
           s : out  STD_LOGIC);
end paridade;

architecture arch_paridade of paridade is
   component xor2 is
      port (x,y: in STD_LOGIC;
            z: out STD_LOGIC;
   end component;
   signal d: STD_LOGIC;
begin
   U1: xor2 port map(a,b,d);
   U2: xor2 port map(c,d,s);
end arch_paridade;
```

VHDL Dataflow

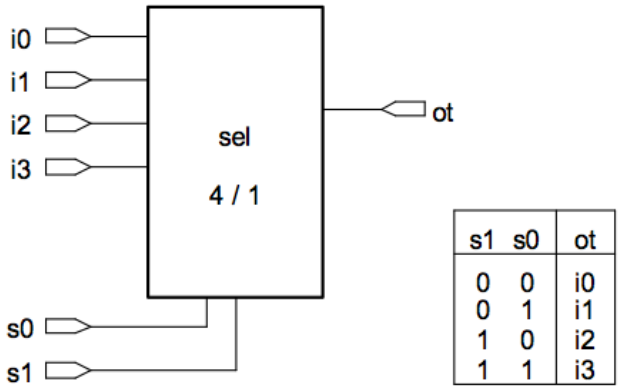
- Várias instruções concorrentes descrevendo o circuito em termos de fluxo de dados e operações
- Atribuição de sinais
- Formas condicionais:
  - **when-else**
  - **with-select**

VHDL Dataflow

- **when-else**

```
sinal_destino <= expressao_a  WHEN condicao_1 ELSE    -- condicao_1 = verdadeira
                 expressao_b  WHEN condicao_2 ELSE    -- condicao_2 = verdadeira
                 expressao_c;                         -- nenhuma condicao verdadeira
```

- **Exemplo**: multiplexador de 4 entradas



s1	s0	ot
0	0	i0
0	1	i1
1	0	i2
1	1	i3

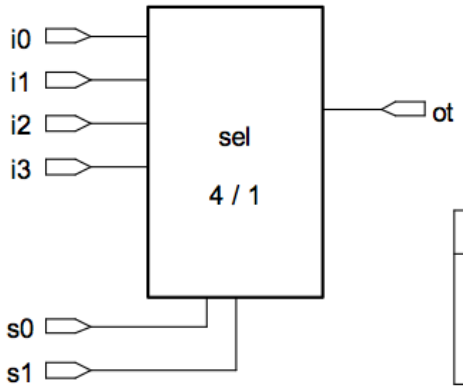
```
architecture arch_mux4 of mux4 is
begin
   ot <= i0 when s1= '0' and s0='0' else
         i1 when s1= '0' and s0='1' else
         i2 when s1= '1' and s0='0' else
         i3;
end arch_mux4;
```

VHDL Dataflow

- **with-select**

```
WITH expressao_escolha SELECT                -- expressao_escolha =
 sinal_destino <= expressao_a WHEN condicao_1,                -- condicao_1
                  expressao_b WHEN condicao_2,                -- condicao_2
                  expressao_c WHEN condicao_3 | condicao_4,    -- condicao_3 ou condicao_4
                  expressao_d WHEN condicao_5 TO condicao_7,   -- condicao_5 ate condicao_7
                  expressao_e WHEN OTHERS;                    -- condicoes restantes
```

- **Exemplo**: multiplexador de 4 entradas



s1	s0	ot
0	0	i0
0	1	i1
1	0	i2
1	1	i3

```
architecture arch_mux4 of mux4 is
   signal sel: STD_LOGIC_VECTOR(1 downto 0);
begin
   sel <= s1 & s0;
   with sel select
   ot <= i0 when "00",
         i1 when "01",
         i2 when "10",
         i3 when "11";
end arch_mux4;
```

VHDL Comportamental

- Baseado em **processos**
- Um processo é um conjunto de intruções sequenciais
- Os processos são executados em paralelo
- Palavra-chave: ***process***
- Lista de sensibilidade:
  - Define quais sinais disparam o processo
  - Processo é executado quando um sinal é alterado

VHDL Comportamental

- **if-else**

```
IF condicao_1 THEN
  comando_sequencial;
  comando_sequencial;
ELSIF condicao_2 THEN      -- clausula ELSIF opcional
  comando_sequencial;
  comando_sequencial;
ELSIF condicao_3 THEN
  comando_sequencial;
ELSE                       -- clausula ELSE opcional
    comando_sequencial;
END IF;
```

VHDL Comportamental

- **Exemplo**: multiplexador de 4 entradas

```
architecture arch_mux4 of mux4 is
   signal sel: STD_LOGIC_VECTOR(1 downto 0);
begin
   sel <= s1 & s0;
   mux:process (i0,i1,i2,13,sel)
   begin
      if sel = "00" then ot <= i0;
      elsif sel = "01" then ot <= i1;
      elsif sel = "10" then ot <= i2;
      else ot <= i0;
   end process;
end arch_mux4;
```

VHDL Comportamental

- **case-when**

```
CASE expressao_escolha IS                          -- expressao_escolha =
  WHEN condicao_1                 => comando_a;                  -- condicao_1
  WHEN condicao_2                 => comando_b; comando_c;       -- condicao_2
  WHEN condicao_3 |  condicao_4 => comando_d;                    -- condicao_3 ou condicao_4
  WHEN condicao_5 TO condicao_9 => comando_d;                    -- condicao_5 ate condicao_9
  WHEN OTHERS                     => comando_e; comando_f;       -- condicoes restantes
END CASE;
```

- **Exemplo**: multiplexador de 4 entradas

```
architecture arch_mux4 of mux4 is
   signal sel: STD_LOGIC_VECTOR(1 downto 0);
begin
   sel <= s1 & s0;
   mux:process (i0,i1,i2,13,sel)
   begin
      case sel is
         when "00" => ot <= i0;
         when "01" => ot <= i1;
         when "10" => ot <= i2;
         when others => ot <= i3;
      end case;
   end process mux;
end arch_mux4;
```

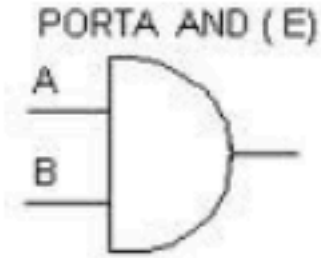
Projeto Lógico Combinacional em VHDL

Circuito Combinacional

- Nível lógico da saída depende da combinação dos níveis lógicos presentes nas entradas
- Não possui características de memória, ou seja, não depende de valores passados de entradas ou saídas ou estados
- É constituído por **portas lógicas**
- Pode ser especificado por equações Booleanas

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
```

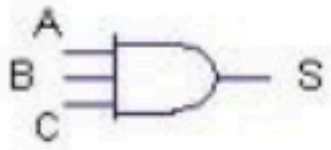


X = A . B

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

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

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
```

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
```

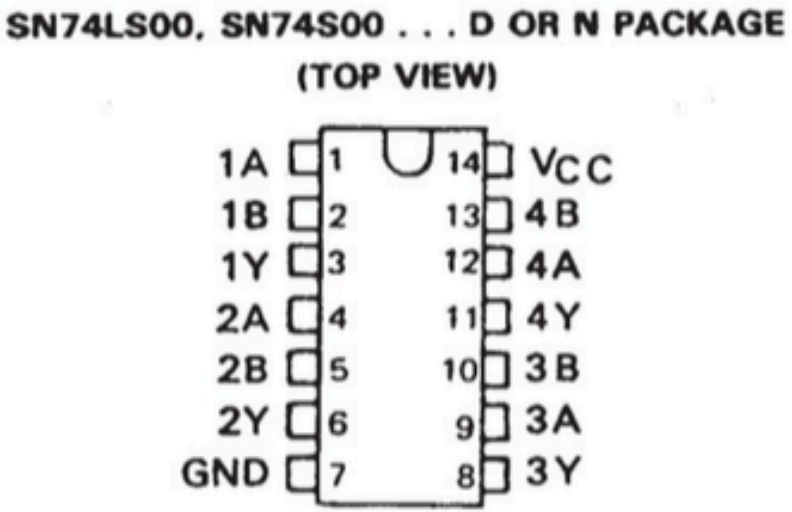
XNOR2

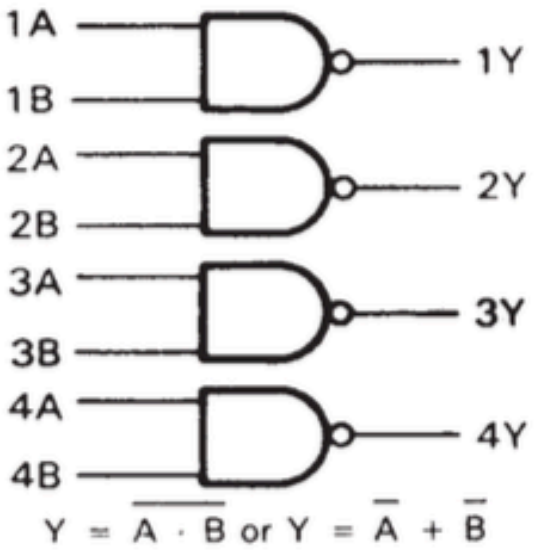
```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   entity xnor2 is port(
5       a, b: in std_logic;
6       s: out std_logic);
7   end xnor2;
8
9   architecture comportamental of xnor2 is
10  begin
11      s <= a xnor b;
12
13  end comportamental;
14
```

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
```

Exemplo: 7400





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.

7400: Solução 1

```
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;
```

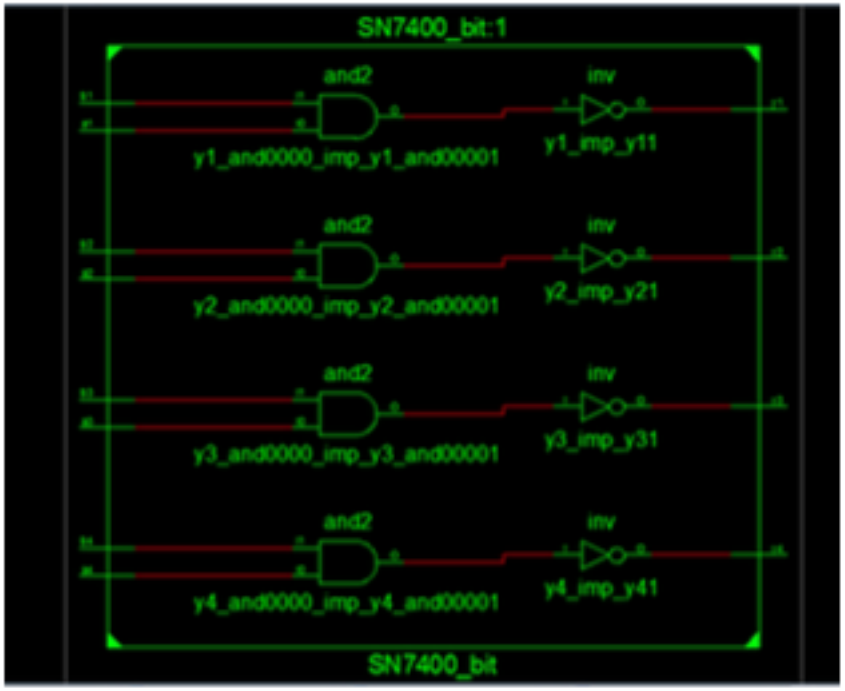


Diagrama RTL Schematic
Obtido após a síntese lógica

7400: Solução 2

```
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;
```

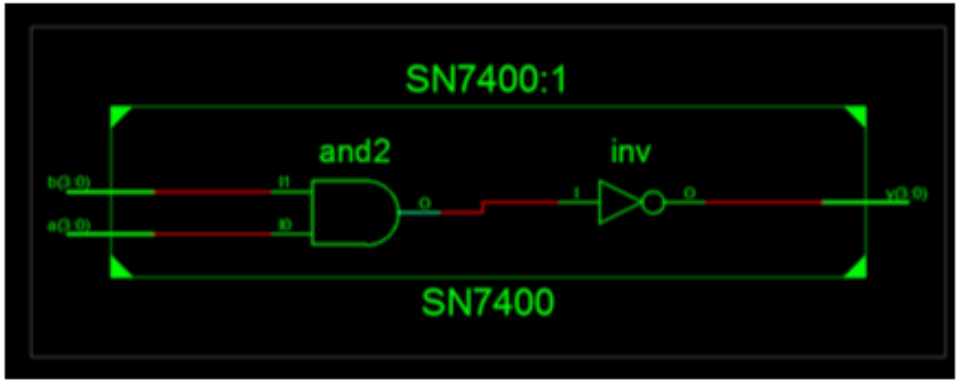
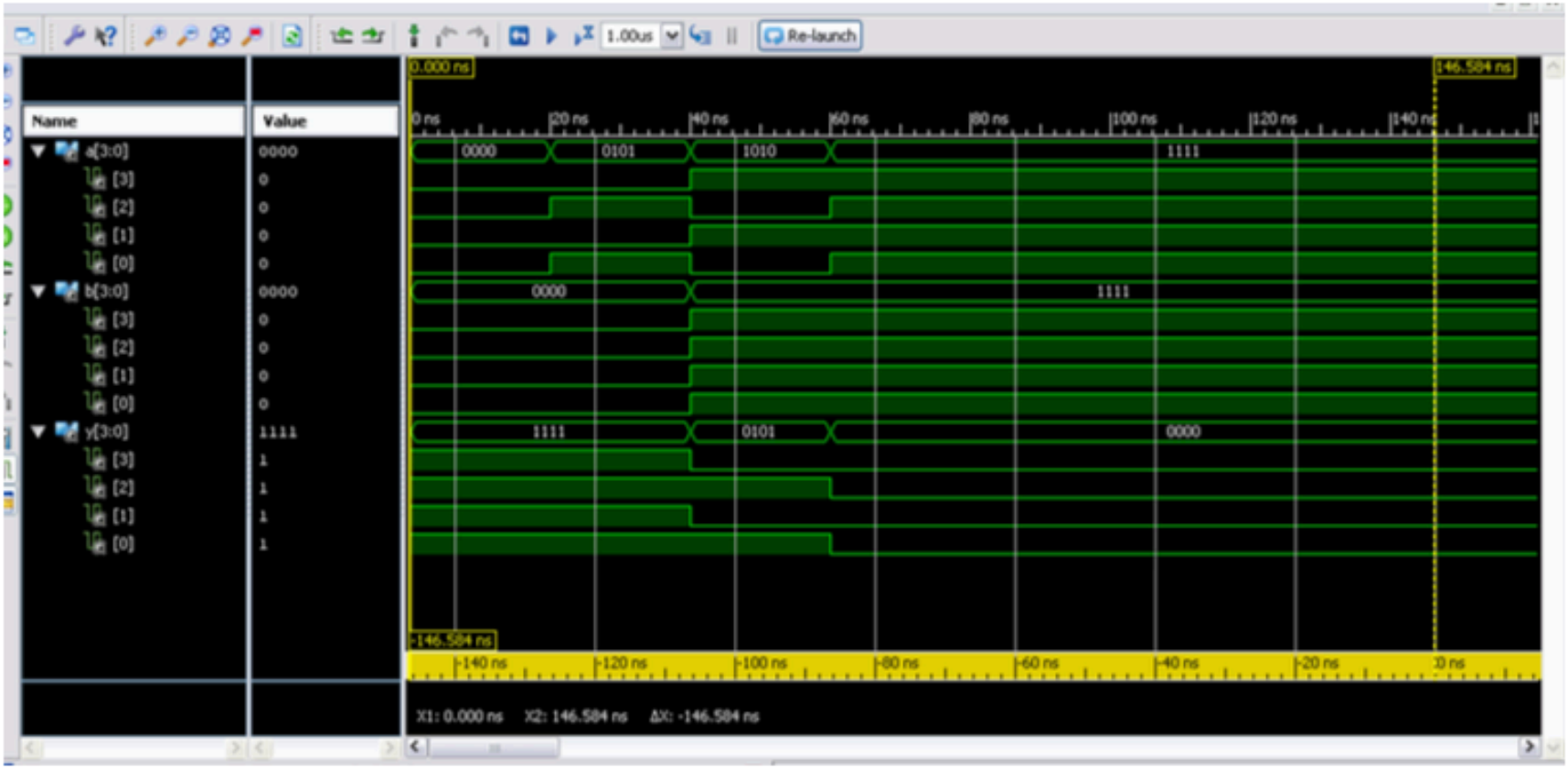


Diagrama RTL Schematic
Observe que a ferramenta de síntese entende as entradas e saídas como vetores.
A ferramenta inferiu uma porta and2 e uma porta not.

7400: 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    -- estimulos 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;
```

7400: Solução 2



Exemplo: decodificador 7seg (processos)

```
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.

Exemplo: decodificador 7seg (processos)

```
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).

Exemplo: decodificador 7seg (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;
```

Exemplo: decodificador 7seg (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
```

```
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  ------------------------------------------------
```

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.

Exemplo: 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;
```

Exemplos: somadores

• 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