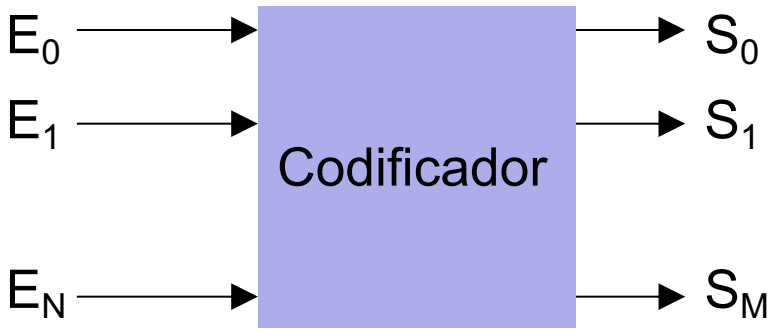
CIRCUITOS COMBINACIONAIS

Modelagem VHDL

Prof. Fernando G. Moraes

(1) (DE)CODIFICADOR

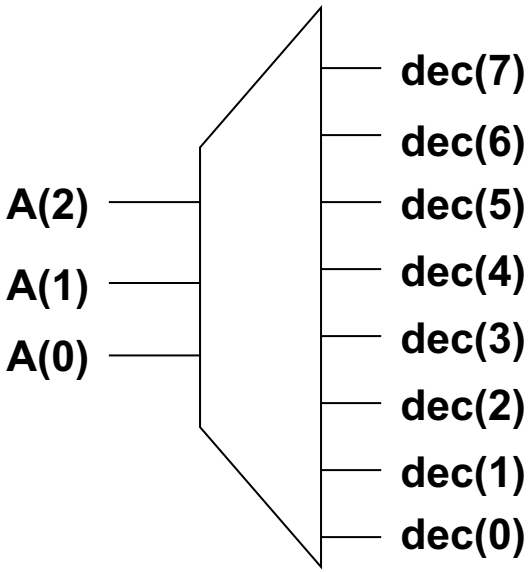
- Codificador é um circuito que mapeia um conjunto de entradas em um conjunto de saídas segundo uma função de codificação
  - Em outras palavras, é um circuito que transforma uma informação de um formato para outro
- Um codificador é normalmente implementado de forma combinacional
- Representação gráfica de codificador genérico



- **Decodificador $n \rightarrow 2^n$**
- **Codificador $2^n \rightarrow n$**

DECODIFICADOR – portas lógicas (n→2^n)

Decodificador de 3 entradas (A(2 downto 0)) e 2^3 (8) saídas (dec(7 downto 0))

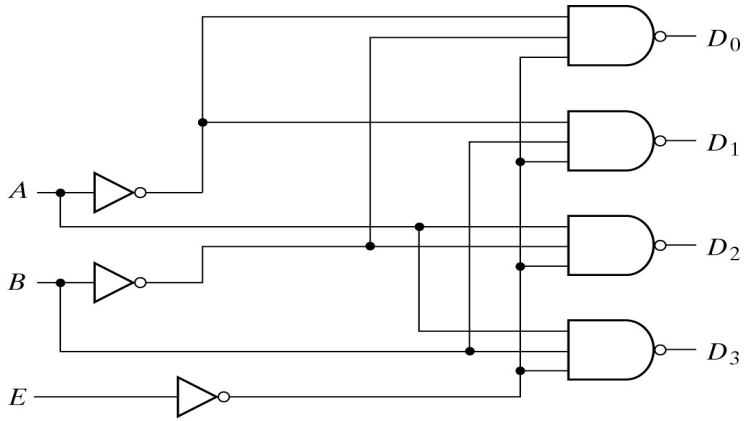



```
dec <= "00000001" when  A="000" else
       "00000010" when  A="001" else
       "00000100" when  A="010" else
       "00001000" when  A="011" else
       "00010000" when  A="100" else
       "00100000" when  A="101" else
       "01000000" when  A="110" else
       "10000000";
```

Sempre ter condição default

DECODIFICADOR – com sinal de habilitação

Conforme indicado pela tabela de verdade, apenas uma saída pode ser igual a 0 a qualquer momento, todas as outras saídas são iguais a 1.



E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

✓ Modelagem em VHDL:

```
D <= "1110" when  ( A='0' and B='0' and E='0' ) else
     "1101" when  ( A='0' and B='1' and E='0' ) else
     "1011" when  ( A='1' and B='0' and E='0' ) else
     "0111" when  ( A='1' and B='1' and E='0' ) else
     "1111";
```

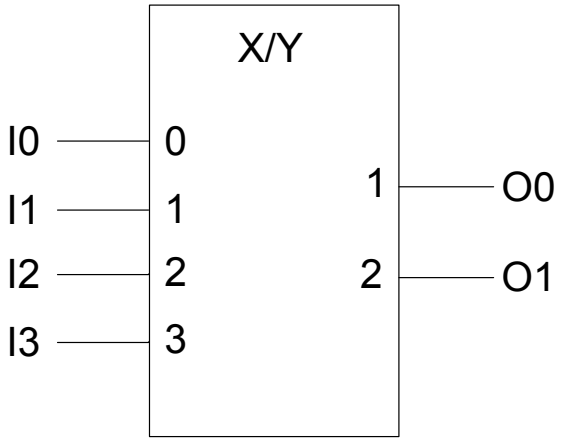
➢ Sempre ter condição *default*

CODIFICADOR

- O **codificador** binário é um circuito combinacional que indica qual das entradas possíveis está ativa (neste caso, “1”).

$2^n \rightarrow n$

I (3)	I (2)	I (1)	I (0)	O
0	0	0	1	**00**
0	0	1	0	**01**
0	1	0	0	**10**
1	0	0	0	**11**



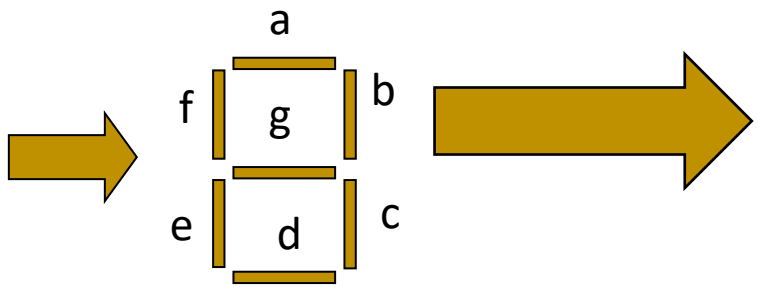
- Modelagem em VHDL:

```
O <= "00" when  I="0001" else
     "01" when  I="0010" else
     "10" when  I="0100" else
     "11" when  I="1000";
```

Display 7 segmentos – simplificando com VHDL

- Simplificando o *display de 7 segmentos* através da descrição em VHDL:

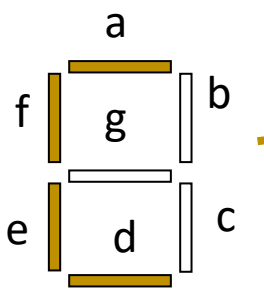
HEXA



- Modelagem em VHDL:

```
saida <= "1111110" when vetorEnt = "0000" else
         "0110000" when vetorEnt = "0001" else
         "1101101" when vetorEnt = "0010" else
         "1111001" when vetorEnt = "0011" else
         "0110011" when vetorEnt = "0100" else
         "1011011" when vetorEnt = "0101" else
         "1011111" when vetorEnt = "0101" else
         "1110000" when vetorEnt = "0111" else
         "1111111" when vetorEnt = "1000" else
         "1110011" when vetorEnt = "1001" else
         "1110111" when vetorEnt = "1010" else
         "0011111" when vetorEnt = "1011" else
         "1001110" when vetorEnt = "1100" else
         "0111101" when vetorEnt = "1101" else
         "1001111" when vetorEnt = "1110" else
         "1000111";
```

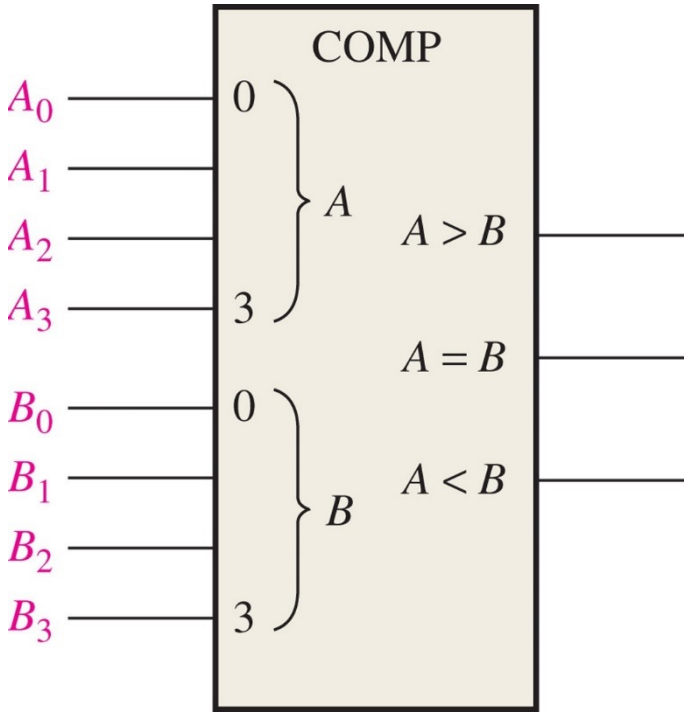
Exemplo: Entrada vetorEnt = "**C1100**" (**C**) e saída:



- Onde, **"abcdefg"** é a ordem dos bits em saida.

(2) COMPARADOR

– Dois números hexadecimal como entradas, 3 saídas



– Circuito para detectar igualdade:

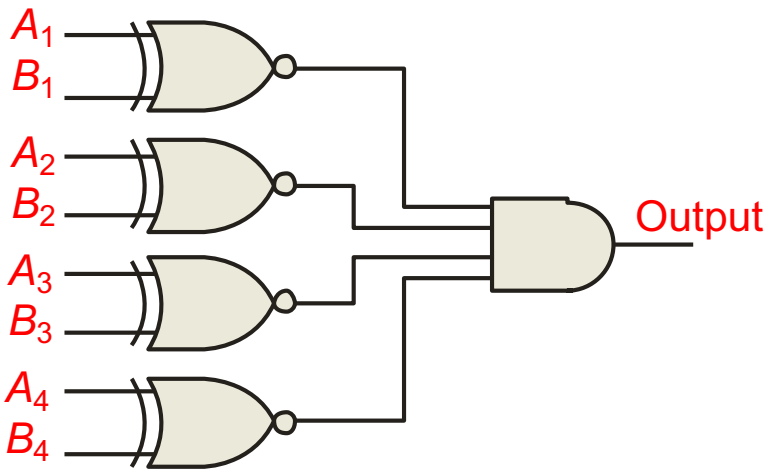
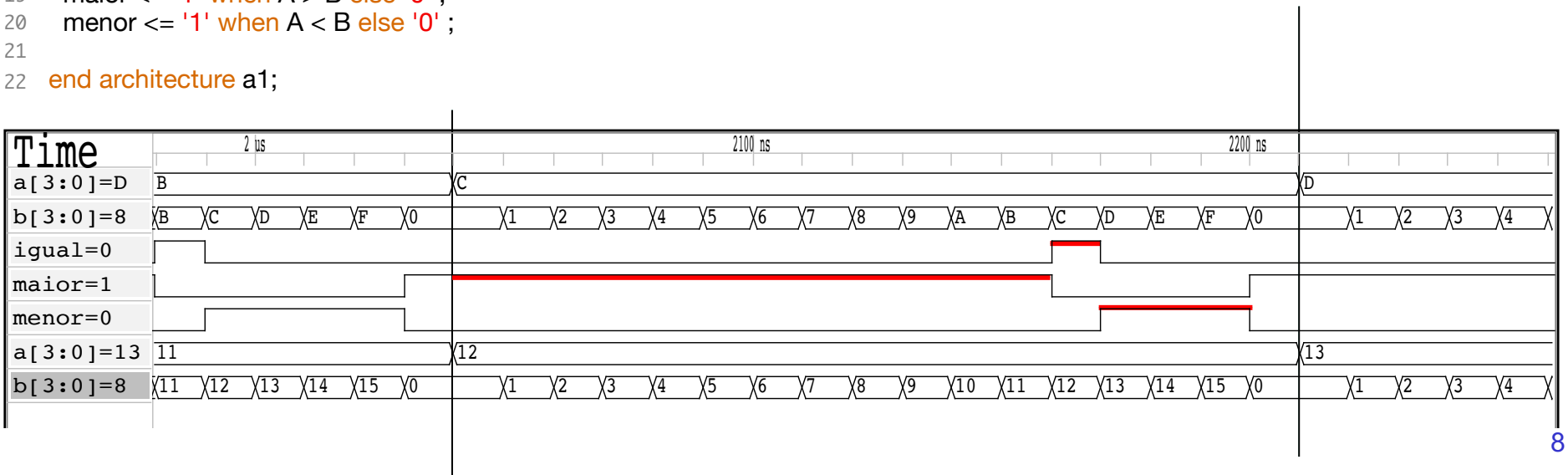


FIGURE 6-21 Logic symbol for a 4-bit comparator with inequality indication.

COMPARADOR – unsigned

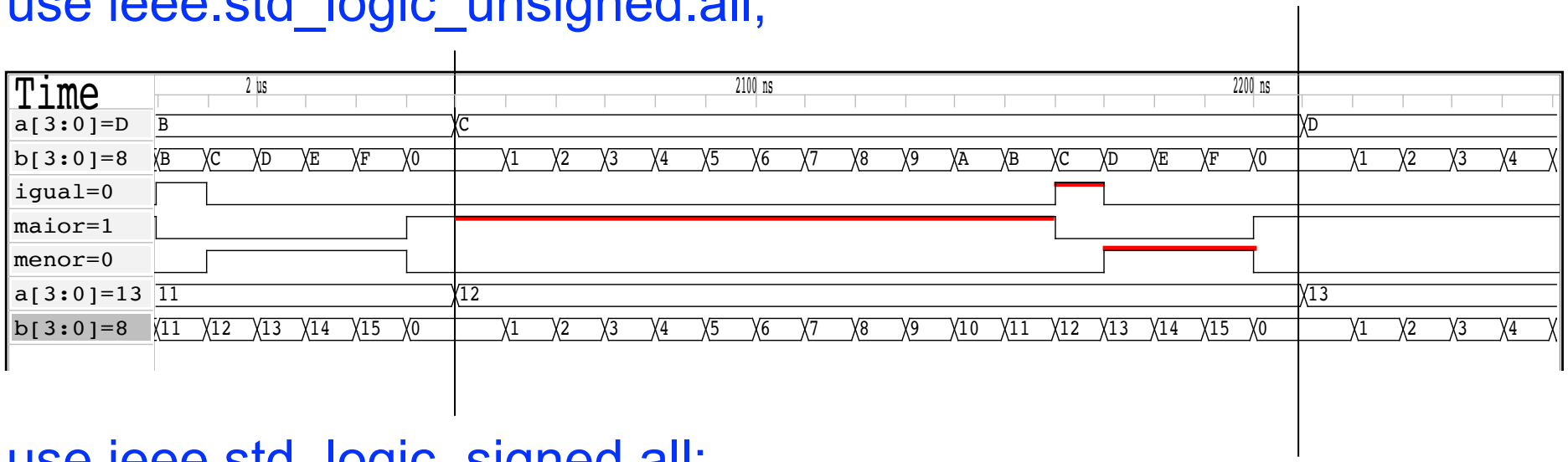
```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;   --****---
4
5  entity comparador is
6        generic(N : integer := 4);
7        port( A, B : in    std_logic_vector(N-1 downto 0);
8              igual    : out std_logic;
9              maior    : out std_logic;
10             menor    : out std_logic
11       );
12 end entity;
13
14
15 architecture  a1 of  comparador is
16 begin
17
18   igual <= '1' when A = B else '0' ;
19   maior <= '1' when A > B else '0' ;
20   menor <= '1' when A < B else '0' ;
21
22 end architecture a1;
```

```
igual <= '1' when A = B else '0' ;
maior <= '1' when A > B else '0' ;
menor <= '1' when A < B else '0' ;
```

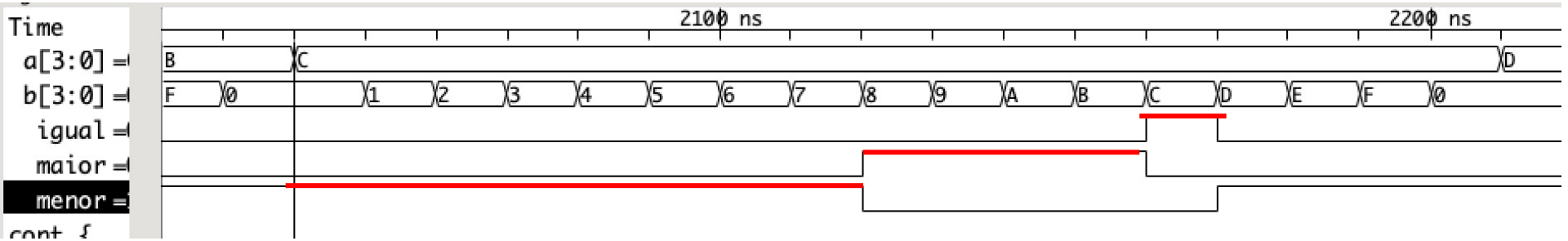


COMPARADOR – complemento de dois

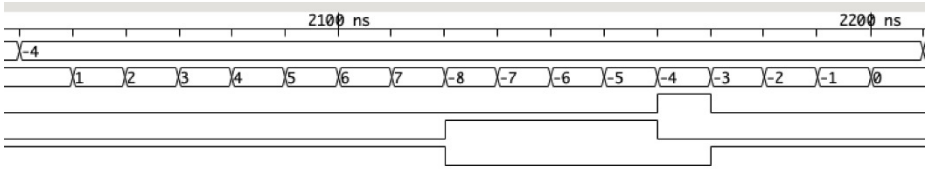
use ieee.std_logic_unsigned.all;



use ieee.std_logic_signed.all;

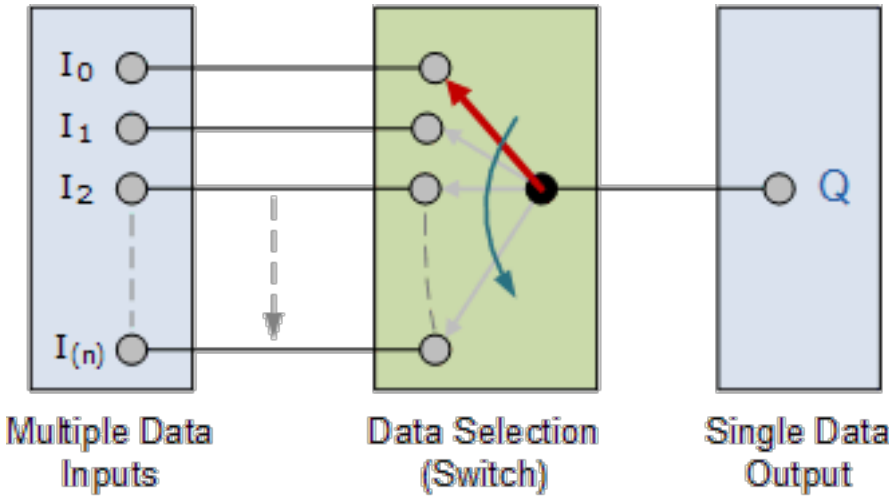


C = 1100 = -4

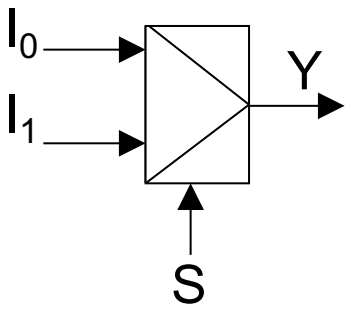


(3) MULTIPLEXADOR

- É um circuito que permite selecionar **uma** dentre várias entradas em função de uma variável de controle



2-to-1 MUX



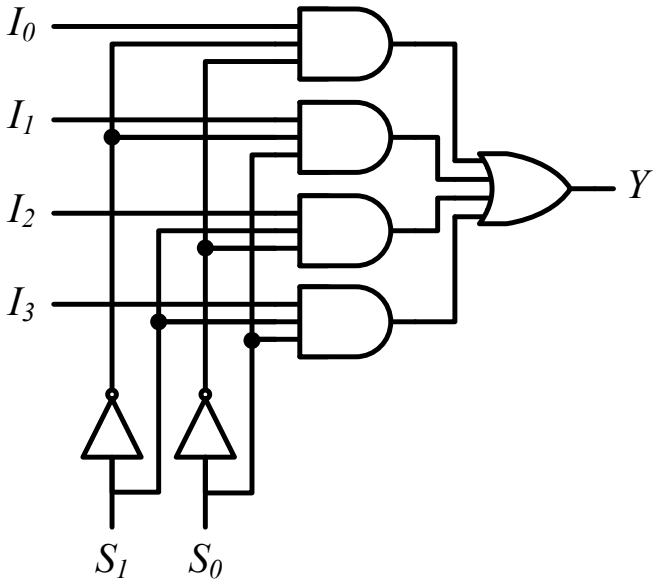
EQUIVALENTE EM SOFTWARE: ***if then else***

VHDL: `Y <= I0 when S = '0' else I1;`

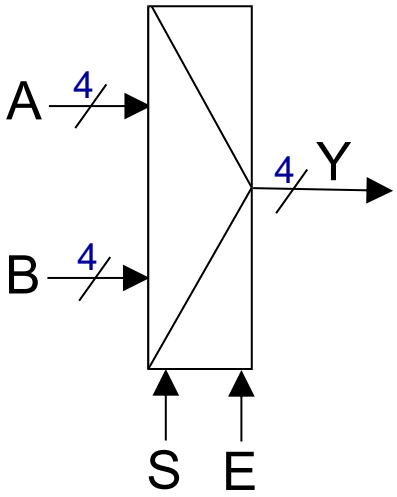
MULTIPLEXADOR

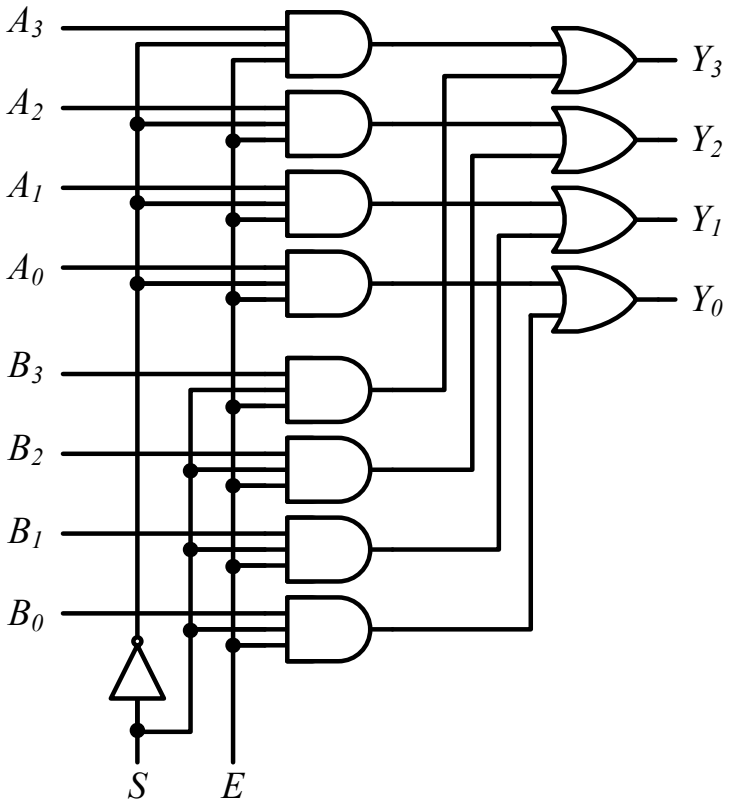
Atribuição com when ... else

```
Y <= I(0) when S = “00” else
     I(1) when S = “01” else
     I(2) when S = “10” else
     I(3) ;
```



Multiplexadores (2x1 - 4 bits com enable)

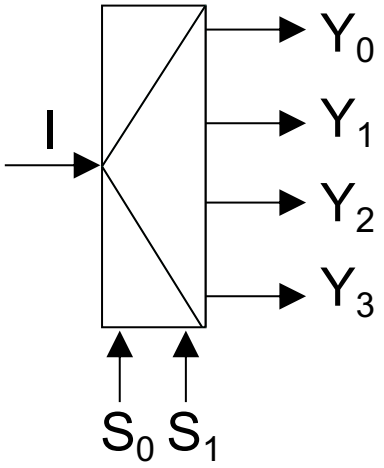


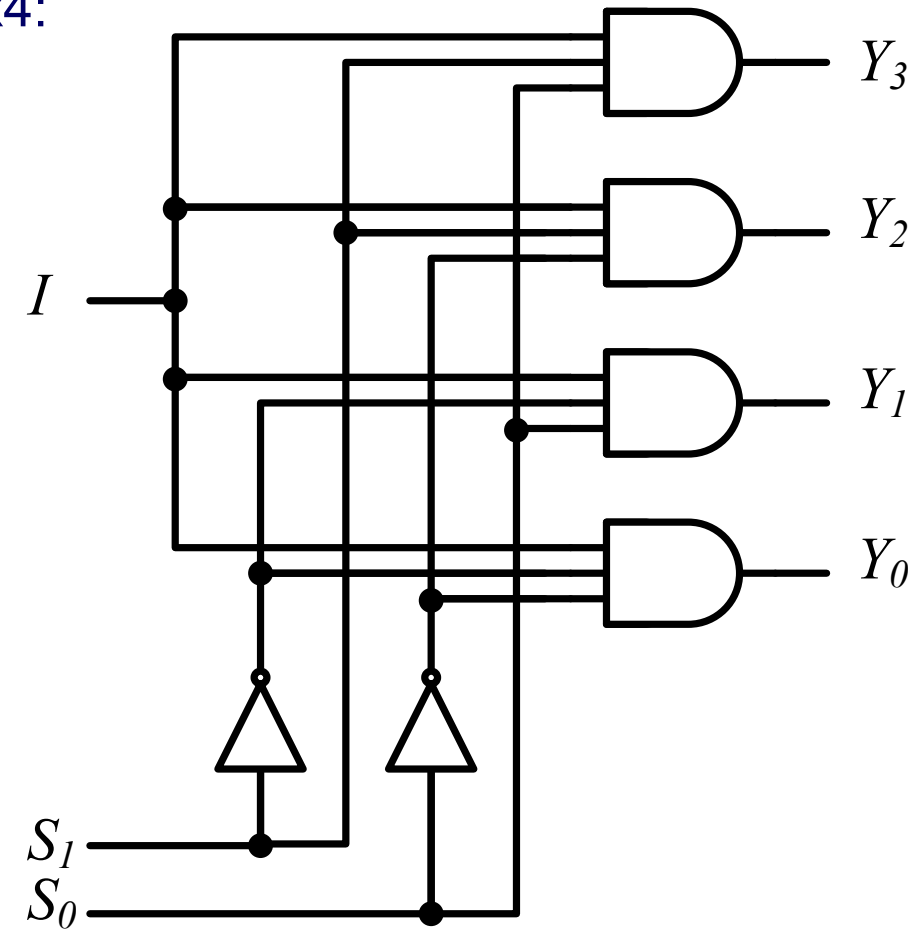



```
Y <= A  when E='1' and S='0' else
     B  when E='1' and S='1' else
     (others => '0' );
```

Demultiplexador

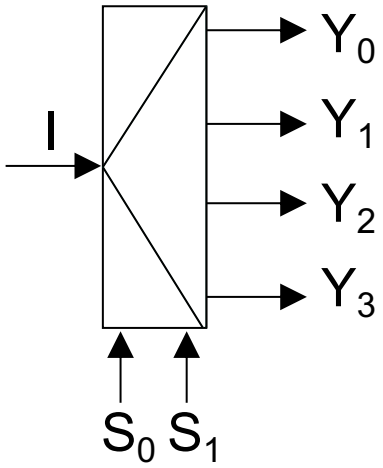
- É um circuito que opera de forma inversa ao multiplexador. Ou seja, recebe uma entrada e distribui esta em uma de várias saídas conforme um sinal de seleção
- Exemplo de um multiplexador 1x4:





Demultiplexador

- É um circuito que opera de forma inversa ao multiplexador. Ou seja, recebe uma entrada e distribui esta em uma de várias saídas conforme um sinal de seleção
- Exemplo de um multiplexador 1x4:



2 bits de seleção (S(1...0)) e 2^2 bits de saídas (Y(3...0))

```
Y <=   "000" &  I        when S = "00" else
       "00" & I & '0'    when S = "01" else
       '0' & I & "00"    when S = "10" else
       I & "000" ;
```

& ➔ operador de concatenação

OU:

```
Y(0) <= I  when S = "00" else '0';
Y(1) <= I  when S = "01" else '0';
Y(2) <= I  when S = "10" else '0';
Y(3) <= I  when S = "11" else '0';
```

EXERCÍCIOS

Exercício 1

1. Considere o circuito abaixo descrito em VHDL. Determine o comportamento do mesmo para os primeiros 80 ns de simulação, considerando o *testbench* fornecido.

Circuito Combinacional

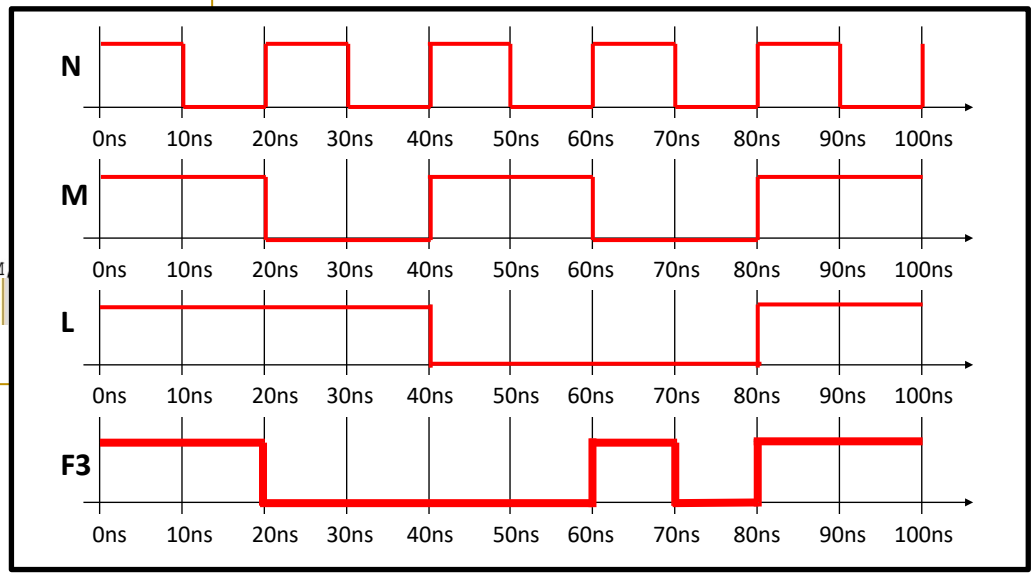
```
library IEEE;
use IEEE.std_logic_1164.all;

entity my_ckt_f3 is
port ( L,M,N  : in std_logic;
       F3     : out std_logic);
end my_ckt_f3;

architecture f3_2 of my_ckt_f3 is
begin
       F3 <= ((NOT L) AND (NOT M) AND N )  OR  (L AND M);
end f3_2;
```

Testbench

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

entity tb is
end tb;

architecture arch of tb is
       signal L, M, N, F3 : std_logic :='1';
begin
       test: entity work.my_ckt_f3 port map ( L=>L, M=>M,
                                              N=>N, F3=>F3);
       N <= not N after 10 ns;
       M <= not M after 20 ns;
       L <= not L after 40 ns;
end arch;
```

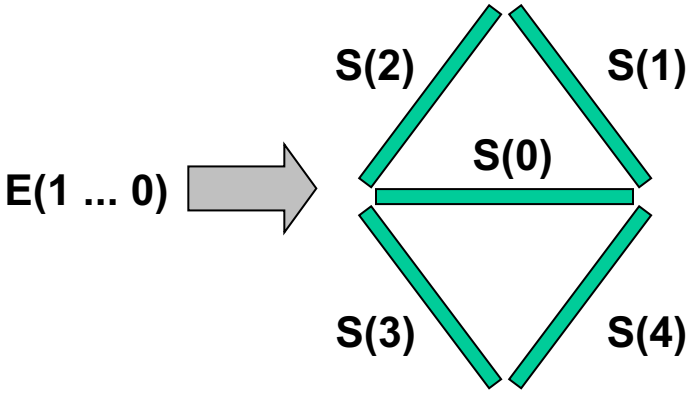
F3 = L'.M'.N + L.M

L	M	N
0	0	1
1	1	0
1	1	1



Exercício 2

2. Modele em VHDL o decodificador do display de elevador ilustrado abaixo:



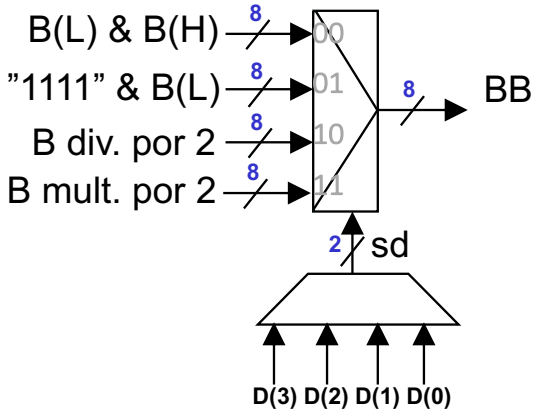
parado	00	-
subindo	01	/\
descendo	10	\/
defeito	11	todos segmentos acesos

– Este tem como entrada um vetor de 2 bits que recebe a seguinte codificação

```
S <= "00001" when E="00" else
     "00110" when E="01" else
     "11000" when E="10" else
     "11111";
```

Exercício 3

3. Considere o circuito combinacional ao lado, é composto por um multiplexador e um codificador com prioridade (4 → 2). Descreva o código VHDL para a obtenção do sinal **BB** e do sinal **sd**, isto é, duas atribuições condicionais

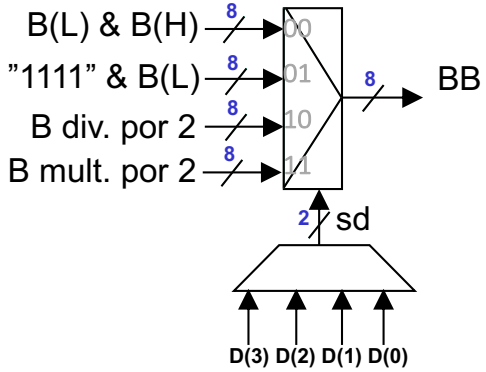


As entradas do multiplexador é o sinal **B** de 8 bits. As entradas do **mux** correspondem:

- ✓ 1ª entrada: inverte os nibbles alto e baixo de B
- ✓ 2ª entrada: fixa o nibble alto em "1111" concatenando-o com a parte baixa de B
- ✓ 3ª entrada: desloca B à direita por 1 bit (B div. por 2)
- ✓ 4ª entrada: desloca B à esquerda por 1 bit (B mult. por 2)

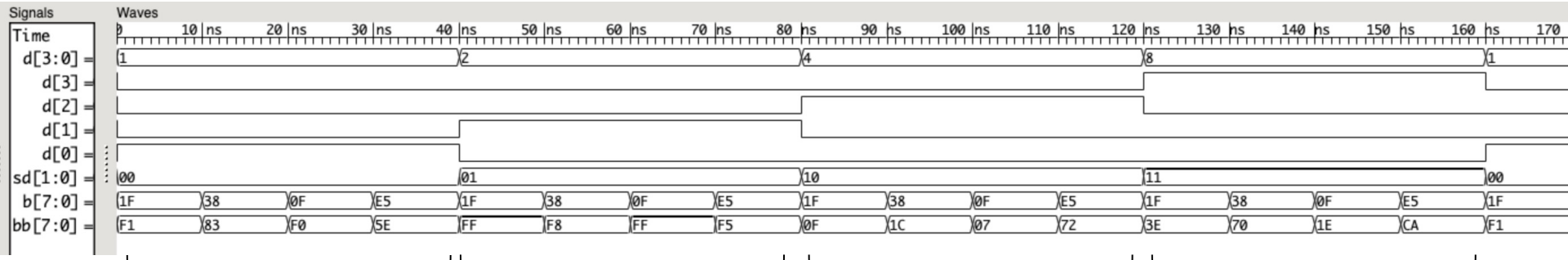
A prioridade do codificador é do bit mais alto ao mais baixo do sinal **D**

Exercício 3 – saída esperada



Entrada b:

```
constant padrao_de_teste : padroes := (
      ( b => x"1F"),
      ( b => x"38"),
      ( b => x"0F"),
      ( b => x"E5")  );
```



d="0001" →sd="00"
B(L) & B(H)

d="0010" →sd="01"
x"F" & B(L)

d="0100" →sd="10"
div(2)

0001 1111 (1F)
0000 1111 (0F)

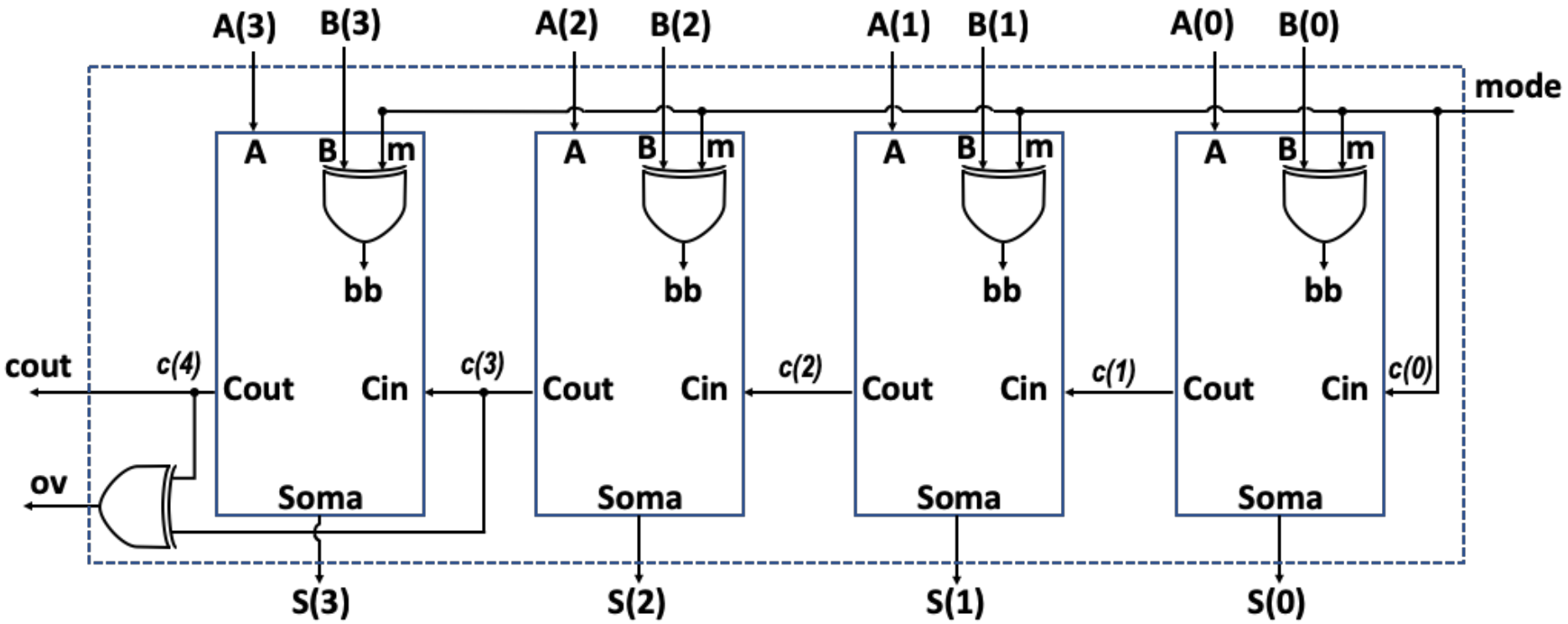
d="1000" →sd="11"
mul(2)

0001 1111 (1F)
0011 111**0** (3E)

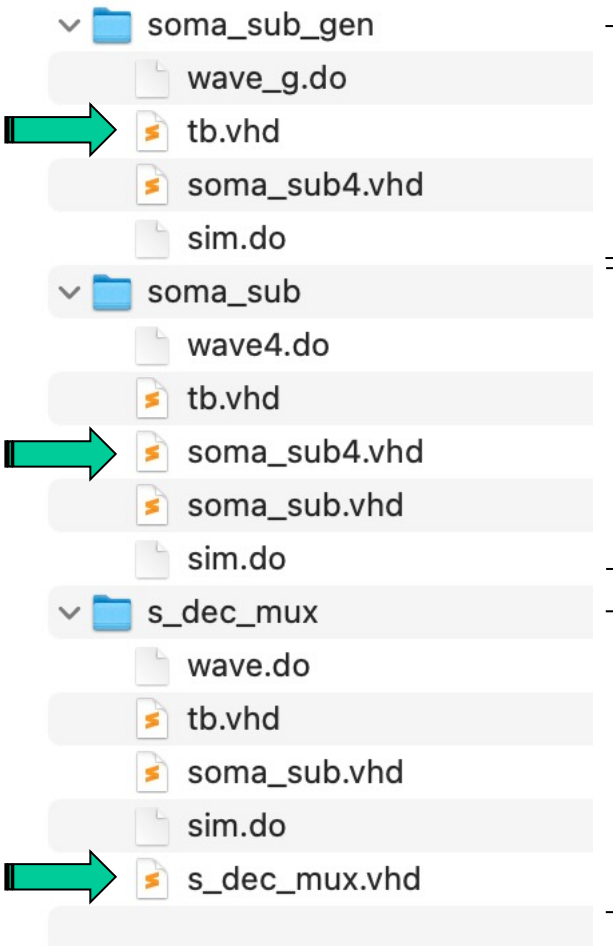
Exercício 3
Solução

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

-------------------------------------
-- Entidade
-------------------------------------
entity ex3 is
  port( B  : in  std_logic_vector(7 downto 0);
        D  : in  std_logic_vector(3 downto 0);
        BB : out std_logic_vector(7 downto 0)
      );
end entity;

-------------------------------------
-- Arquitetura
-------------------------------------
architecture ex3 of ex3 is
  signal sd : std_logic_vector(1 downto 0); -- saida do decoder
begin

  sd <= "11" when D(3)='1' else
        "10" when D(2)='1' else
        "01" when D(1)='1' else
        "00";

  BB <= B(3 downto 0) & B(7 downto 4) when sd="00" else
        "1111" & B(3 downto 0)        when sd="01" else
        '0' & B(7 downto 1)           when sd="10" else
        B(6 downto 0) & '0';

end architecture;
```

Soma e subtração



Material de apoio “soma sub”



Simulação 2 – somador genérico

Simulação 1 – somador 4 bits
Já realizado

Simulação 3 – exercício

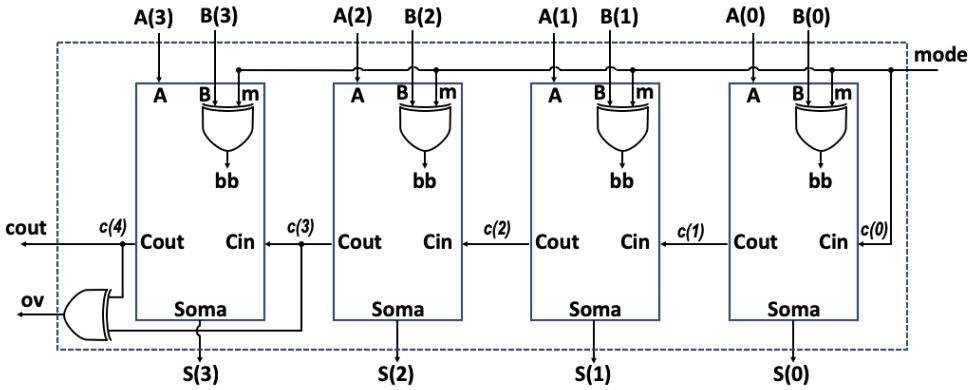
Soma e subtração – *generic*

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

biblioteca

```
entity soma_sub is
    generic(N : integer := 4);
    port( A, B :  in  std_logic_vector(N-1 downto 0);
          S :     out std_logic_vector(N-1 downto 0);
          mode : in  std_logic;        --- modo
          Cout :  out std_logic;
          OV :    out std_logic
    );
end entity;
```

Entidade com generic




```
architecture  a1 of  soma_sub is
    signal co: std_logic_vector(N downto 0);
    signal bb : std_logic_vector(N-1 downto 0);
begin
 co(0)  <= mode;       -- se o cin for 0 é soma, senão é subtração
 Cout   <= co(N);
 OV     <= co(N) xor co(N-1);

 add: for i in 0 to N-1 generate
  begin
    bb(i)  <= b(i) xor mode;
    S(i)   <= co(i) xor A(i) xor bb(i);
    co(i+1) <= (co(i) and A(i)) or (co(i) and bb(i)) or (A(i) and bb(i));
  end generate;

end architecture a1;
```

Cria *n* instâncias da lógica

Cuidado com o *generate:* curto circuitos

Testando o soma_sub de forma genérica

```
library ieee,work;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.CONV_STD_LOGIC_VECTOR;

entity testa_soma is
end;

architecture a1 of testa_soma is

  constant N  : natural := 12;        -- tamanho do somadores

  signal a,b, sum : std_logic_vector(N-1 downto 0);
  signal cout,mode, ov  : std_logic;

  type test_record is record
     a, b :  integer;
     mode  :  std_logic;
  end record;

  type padroes is array(natural range <>) of test_record;

  constant padrao_de_teste : padroes := (
      (a => 234,  b => 890,  mode => '1'),
      (a => 234,  b => 890,  mode => '0'),
      (a => 89 ,  b => 630,  mode => '1'),
      (a => 89 ,  b => 630,  mode => '0'),
      (a => 999,  b => 710,  mode => '1'),
      (a => 999,  b => 710,  mode => '0'),
      (a => 12,   b => 501,  mode => '1'),
      (a => 12,   b => 501,  mode => '0'),
      (a => 123,  b => 432,  mode => '1'),
      (a => 123,  b => 432,  mode => '0'),
      (a => 345,  b => 323,  mode => '1'),
      (a => 345,  b => 323,  mode => '0'),
      (a => 888,  b => 121,  mode => '1'),
      (a => 100,  b => 900,  mode => '0'),
      (a => 300,  b => 124,  mode => '1'),
      (a => 300,  b => 124,  mode => '0')
   );
begin
```

bibliotecas

Entity sem pinos

Número de bits do somador

Sinais internos

Estrutura de dados para os estímulos

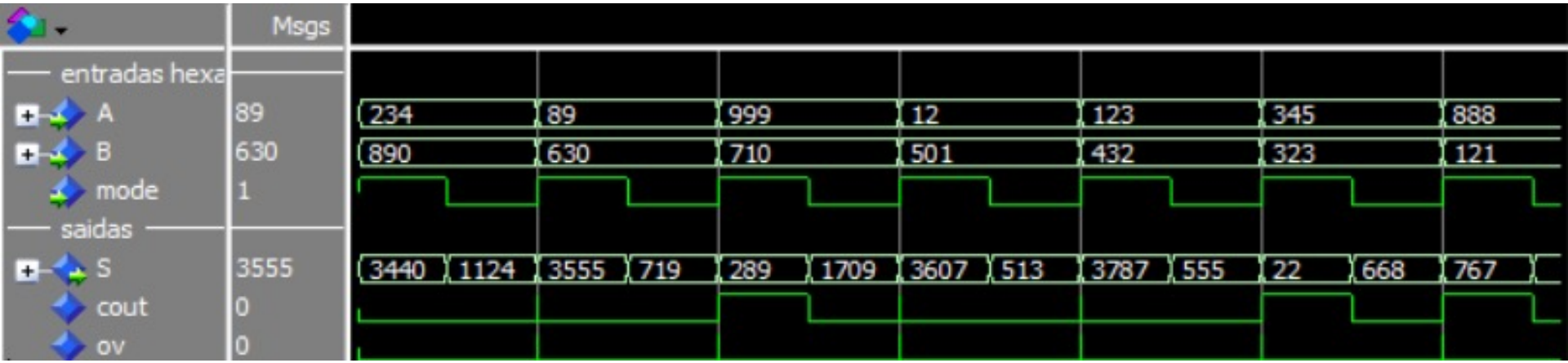
Array sem dimensão

Vetores de teste

Testando o soma_sub genérico

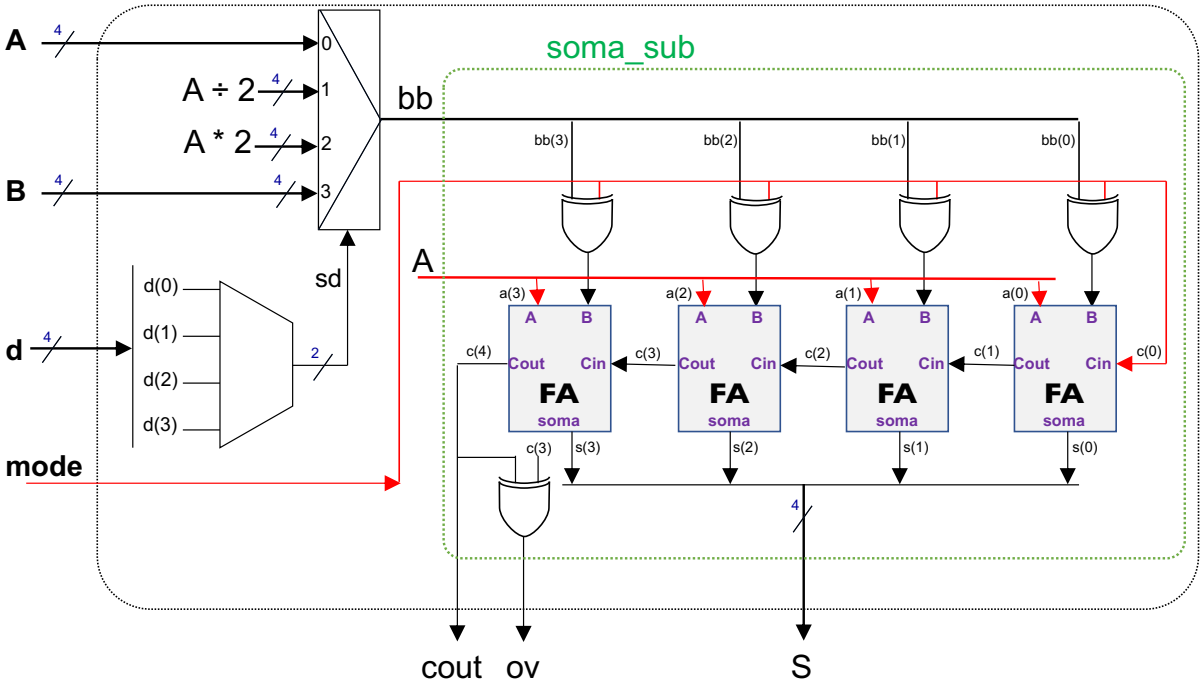
```
begin

    sb: entity work.soma_sub        instância
        generic map(N)
        port map( a=> a, b=> b, mode=>mode, S=> sum, cout=> cout, ov=>ov);

    test: process      Loop de varredura
      begin
            for i in 0 to padrao_de_teste'high loop
               a <= CONV_STD_LOGIC_VECTOR(padrao_de_teste(i).a, a'length);
               b <= CONV_STD_LOGIC_VECTOR(padrao_de_teste(i).b, b'length);
               mode <= padrao_de_teste(i).mode;
               wait for 10 ns;
            end loop;
    end process;

end architecture;
```

Resultado


Msgs
entradas hexa
A
89
B
630
mode
1
saidas
S
3555
cout
0
ov
0
234
89
999
12
123
345
888
890
630
710
501
432
323
121
3440
1124
3555
719
289
1709
3607
513
3787
555
22
668
767

Exercício no simulador

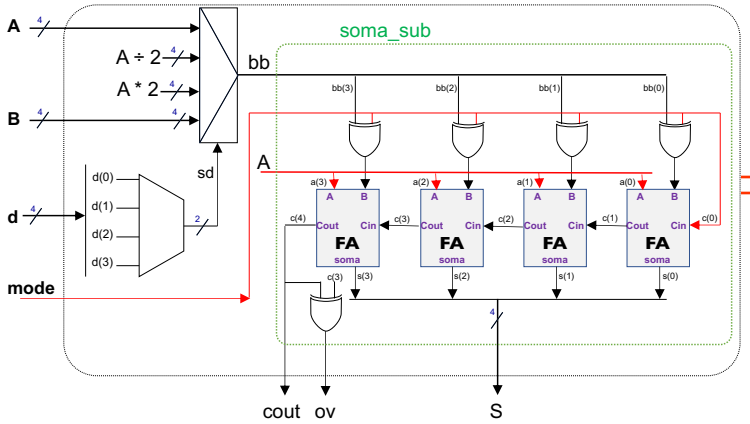
```
entity exercicio is
      port( A, B  :  in  std_logic_vector(7 downto 0);
            S     :  out std_logic_vector(7 downto 0);
            mode  : in  std_logic;
            d     :  in  std_logic_vector(3 downto 0);
            cout  :out std_logic;
            ov    :  out std_logic    );
end entity;
```



Versão 4 bits – **exercício: 8 bits**

- Unir o soma_sub com o exercício dec_mux
- O circuito realiza 8 operações:

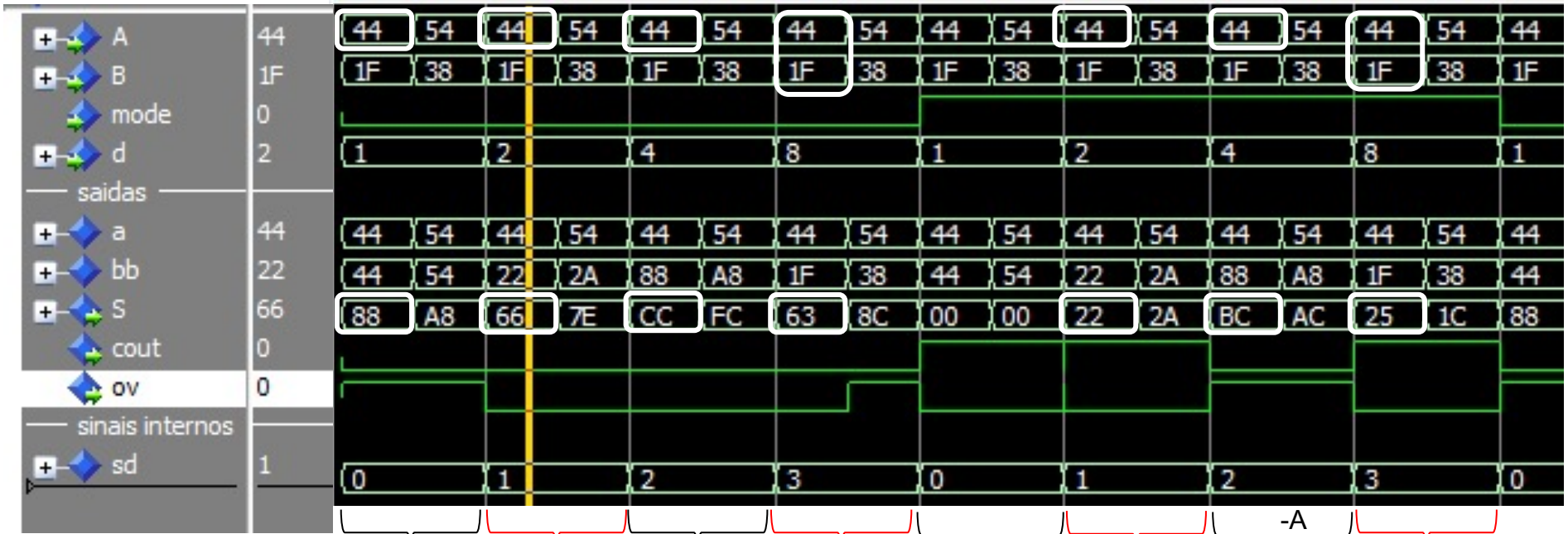
mode	d	operação
0	d(0)=1	2.A
0	d(1)=1	1,5.A
0	d(2)=1	3.A
0	d(3)=1	A+B
1	d(0)=1	0
1	d(1)=1	0,5.A
1	d(2)=1	-A
1	d(3)=1	A-B



s_dec_mux.vhd

mode	D	oper
0	d(0)=1	2.A
0	d(1)=1	1,5.A
0	d(2)=1	3.A
0	d(3)=1	A+B
1	d(0)=1	0
1	d(1)=1	0,5.A
1	d(2)=1	-A
1	d(3)=1	A-B

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

entity s_dec_mux is
      port( completar );
end entity;

architecture  a1 of  s_dec_mux is
  <completar>
begin

   sd <= <completar>

   bb <= <completar>

   sb: entity work.soma_sub
         generic map(8)
         port map( <completar> )

end architecture a1;
```

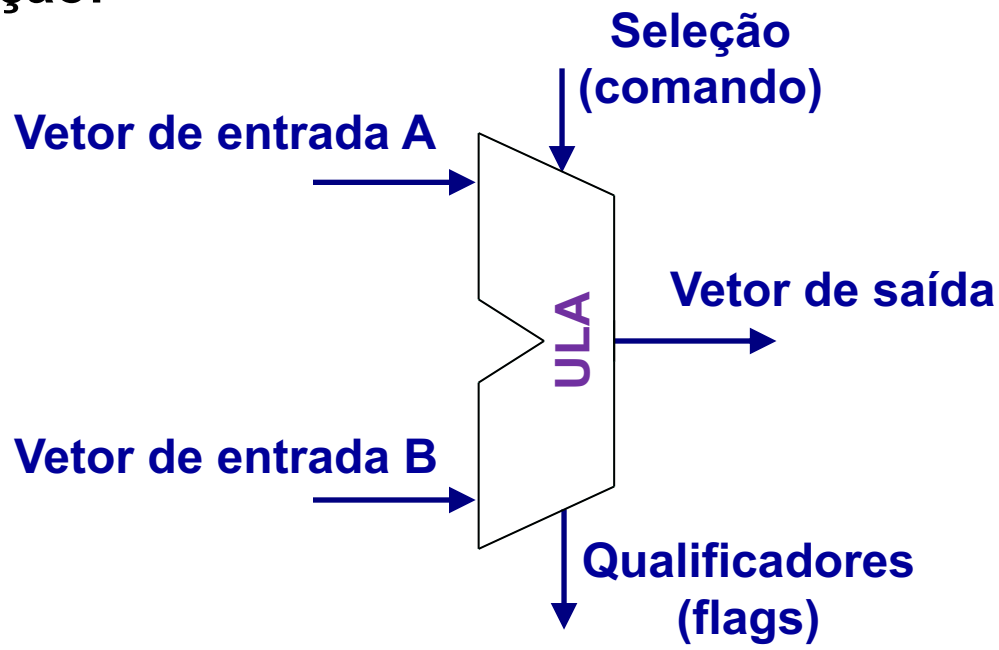


CIRCUITOS COMBINACIONAIS

ULA

ULA - UNIDADE LÓGICO E ARITMÉTICA

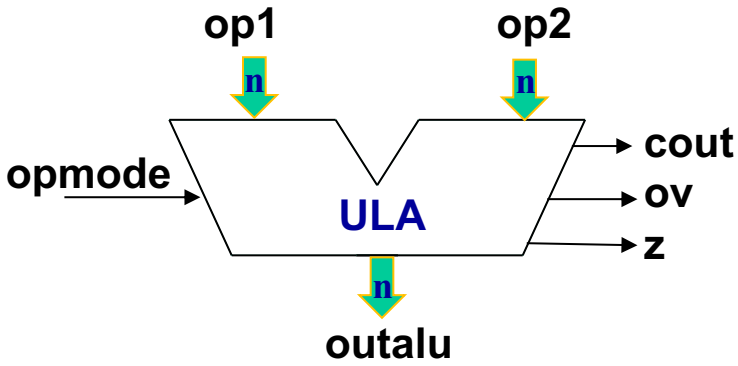
- Unidade Lógica e Aritmética (ULA) é um circuito que realiza funções lógicas e aritméticas
- É um dos componentes de transformação de dados principais de um processador
- Normalmente implementado de forma combinacional
- Representação:



ULA - comandos

Package:

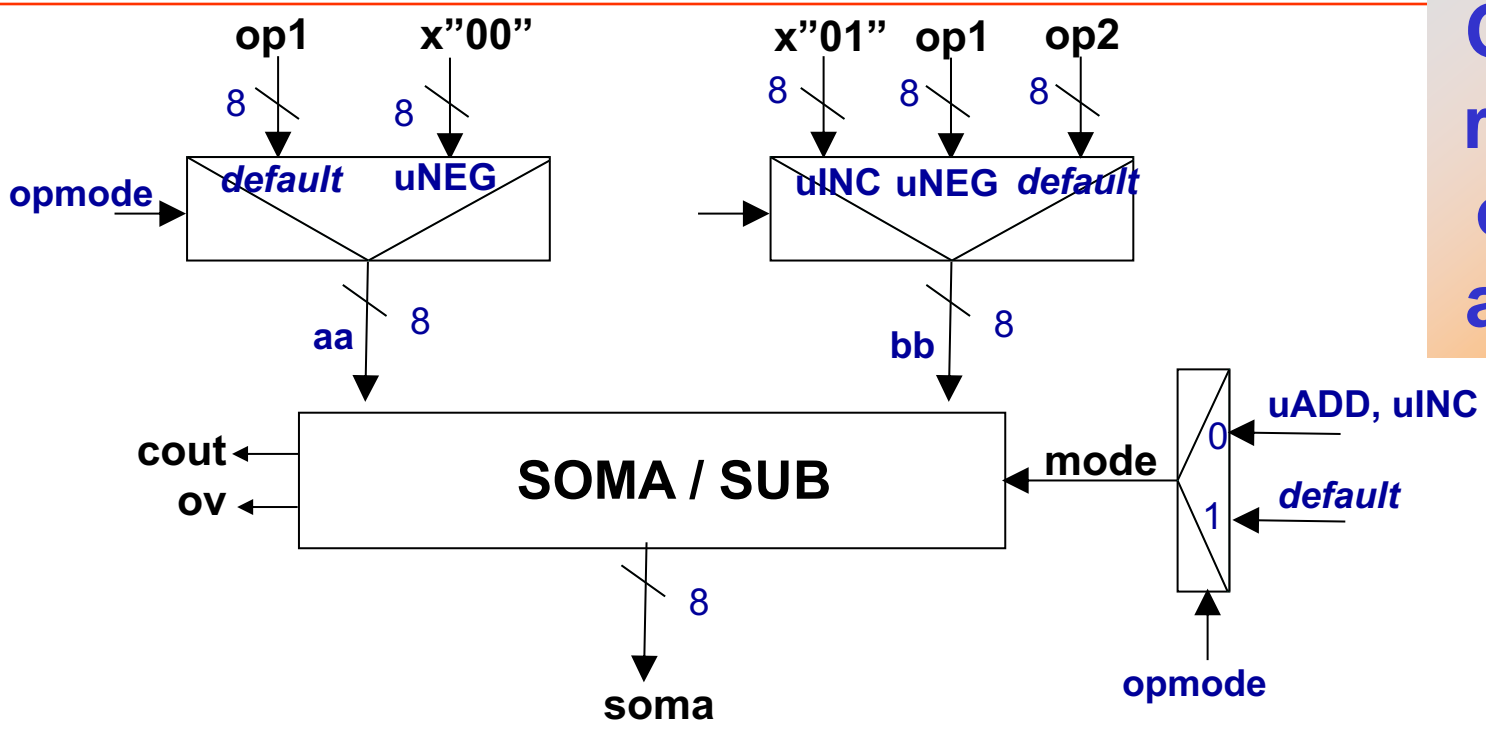
- Declara constantes, tipos de dados, subprogramas.
- Objetivo: reutilização de código



Exemplo de package com 9 operações para a ula (arquivo ula_pack.vhd):

```
library IEEE;
use IEEE.Std_Logic_1164.all;

package p_ula is
        type op_alu is
          ( uAND, uOR, uXOR, uSLL, uSRL, uADD, uSUB, uINC, uNEG);
end p_ula;
```

Módulo Aritmético (soma/sub/inc/neg)



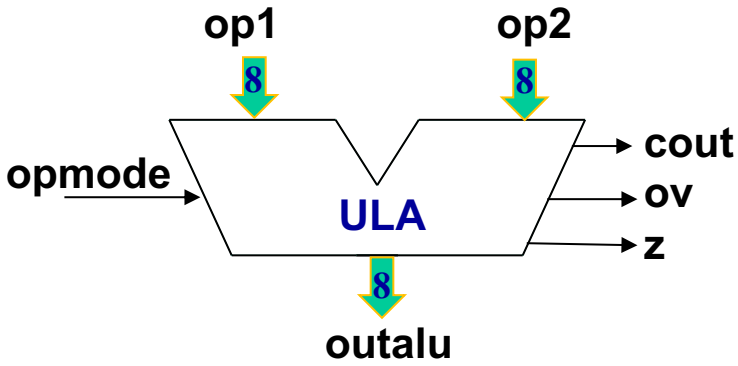
O somador realiza as 4 operações aritméticas

opmode	Ação
uINC	op1 + 1 + 0
uADD	op1 + op2 + 0
uSUB	op1+ not(op2) + 1
uNEG (2's comp)	0 + not(op1) + 1

```
type op_alu is
        ( uAND, uOR, uXOR, uSLL, uSRL,
          uADD, uSUB, uINC, uNEG);
```

Obs: o not do segundo operando deve-se ao mode=1

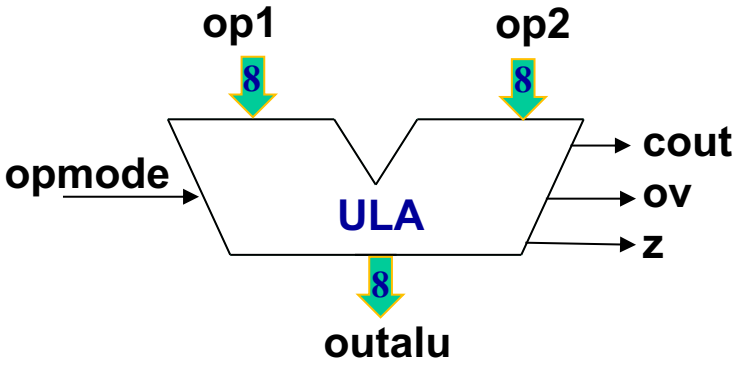
VHDL

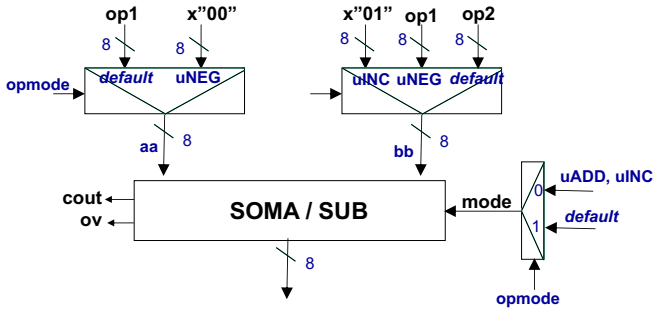



```
-------------------------------------
-- Biblioteca
-------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use work.p_ula.all;

-------------------------------------
-- Entidade
-------------------------------------
entity alu is
  port(op1, op2 : in std_logic_vector(7 downto 0);
       opmode : in op_alu;
       z, cout, ov: out std_logic;
       outalu : out std_logic_vector(7 downto 0));
end entity;
```

Arquitetura - visão geral

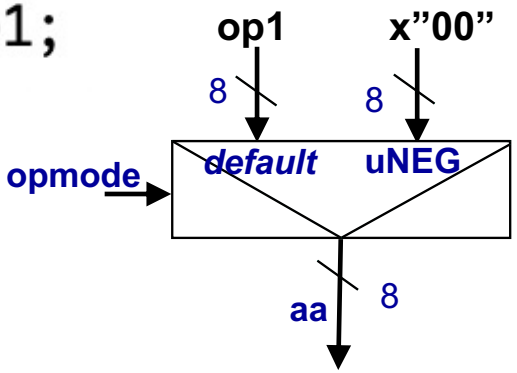
```
architecture alu of alu is
  signal aa, bb, soma: std_logic_vector(7 downto 0);
  signal mode: std_logic;
begin

  ---------------------------------
  -- Instancia o SomaSub
  ---------------------------------
  z <= '1' when soma = x"00" else '0';

  mode <= '0' when opmode=uADD or opmode=uINC or opmode=uDEC else '1';

  aa <= x"00" when opmode=uNEG else op1;

  bb <= x"01" when opmode=uINC else
        x"FF" when opmode=uDEC else
        op1   when opmode=uNEG or opmode=uZero else
        op2;

  sb: entity work.soma_sub
    generic map(8)
    port map (a => aa, b => bb, mode => mode,
              S => soma, cout => cout, ov => ov);
 ---------------------------------
  -- ULA
  ---------------------------------
  outalu <= op1 and op2 when opmode=uAND else
            op1 or op2  when opmode=uOR else
            op1 xor op2 when opmode=uXOR else
            op1(6 downto 0) & '0' when opmode=uSLL else
            '0' & op1(7 downto 1) when opmode=uSRL else
            soma; --- uADD, uSUB, uINC, uNEG, DEC

end architecture;
```



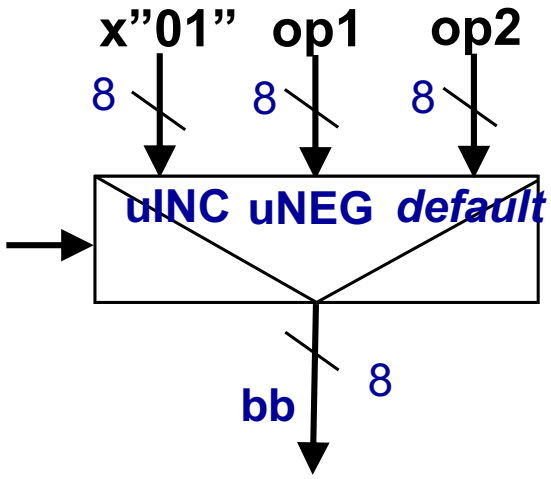


VHDL - multiplexadores

```
aa <= x"00" when opmode=uNEG else op1;
```



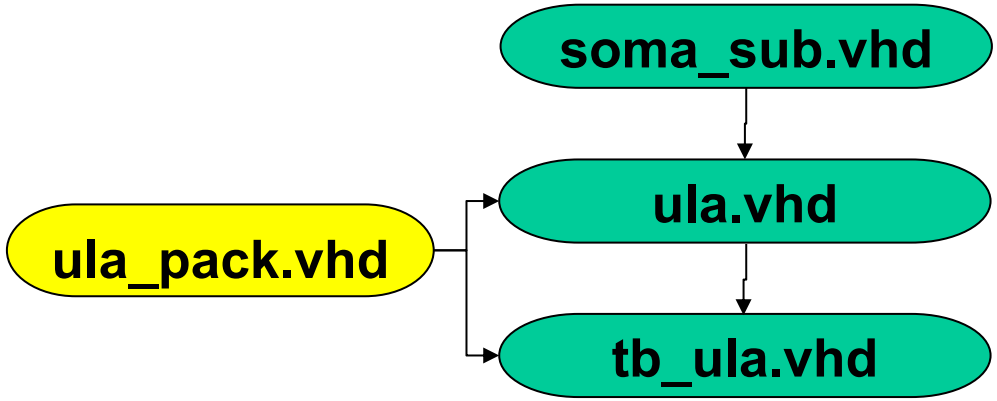

```
bb <= x"01" when opmode=uINC else
      op1   when opmode=uNEG else
      op2;
```



VHDL - ULA

```
sb: entity work.soma_sub
  generic map(8)
  port map (a => aa, b => bb, mode => mode,
            S => soma, cout => cout, ov => ov);
--------------------------------
-- ULA
--------------------------------
outalu <= op1 and op2 when opmode=uAND else
          op1 or op2  when opmode=uOR else
          op1 xor op2 when opmode=uXOR else
          op1(6 downto 0) & '0' when opmode=uSLL else
          '0' & op1(7 downto 1) when opmode=uSRL else
          soma; --- uADD, uSUB, uINC, uNEG, DEC
```

Estrutura de Arquivos

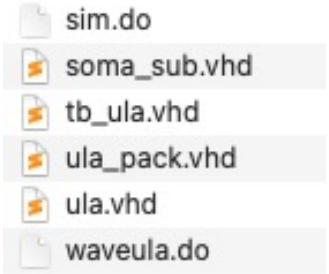


→ **soma_sub**: somador/subtrator parametrizável

→ Parte do script para a ula conforme este esquema de arquivos:

```
vcom -work work ula_pack.vhd
vcom -work work soma_sub.vhd
vcom -work work ula.vhd
vcom -work work tb_ula.vhd

vsim -voptargs=+acc=lprn -t ns work.tb
```

Arquivos de apoio



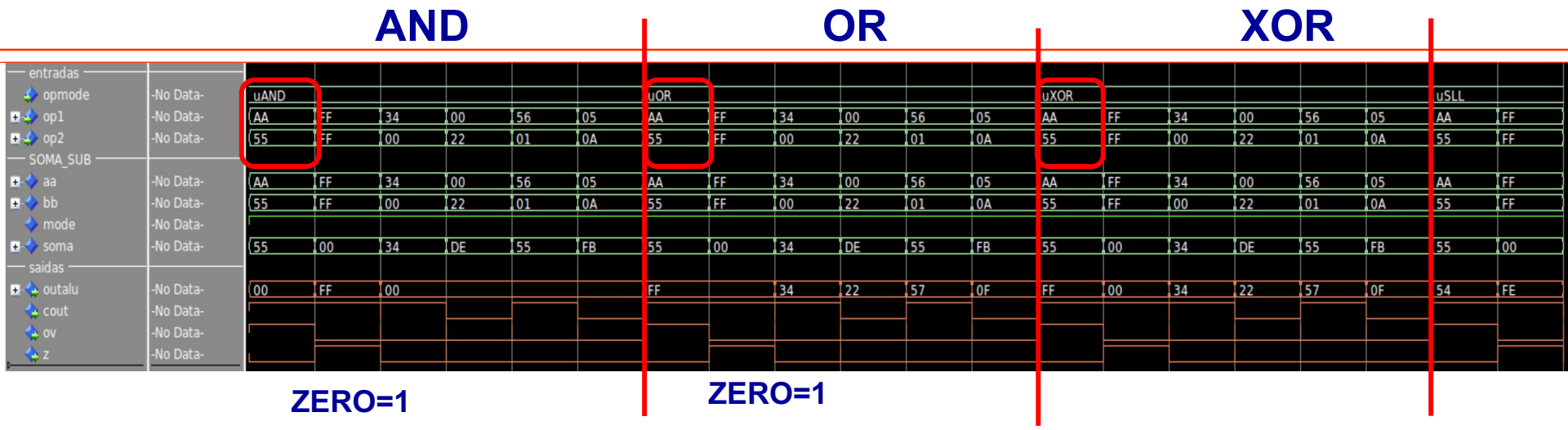
Testando a ULA

```
library ieee,work;
use ieee.std_logic_1164.all;
use work.p_ula.all;

entity tb is
end tb;

architecture a1 of tb is

  signal op1, op2, outalu: std_logic_vector(7 downto 0);
  signal z, cout, ov  : std_logic;
  signal opmode : op_alu;

  type test_record is record
     a, b :  std_logic_vector(7 downto 0);
  end record;

  type padroes is array(natural range <>) of test_record;

  constant padrao_de_teste : padroes := (
      (a => x"AA",  b => x"55"),
      (a => x"FF",  b => x"FF"),
      (a => x"34",  b => x"00"),
      (a => x"00",  b => x"22"),
      (a => x"56",  b => x"01"),
      (a => x"05",  b => x"0A")
   );

 begin
```

Vetores de teste

```
begin

  sb: entity work.alu
        port map( op1=> op1, op2=> op2, outalu=>outalu,
                  z=>z, cout=>cout, ov=>ov, opmode=>opmode);

  test: process
   begin                    Varre o tipo enumerado
                                Modo de operação
    for op in op_alu loop    --  percorre o tipo enumerado

        opmode <= op;

        for i in 0 to padrao_de_teste'high loop
           op1 <= padrao_de_teste(i).a;
           op2 <= padrao_de_teste(i).b;
           wait for 10 ns;
        end loop;

  end loop;

  end process;

end architecture;
```

Neste exemplo para cada vetor de teste são testadas todas as operações



AND: AA.55= 0, FF.FF=FF, 34.00=0, 00.22=0, 56.1=0, 5.A=0

OR: AA+55= FF, FF+FF=FF, 34+00=34, 00+22=22, 56+1=57, 5+A=F

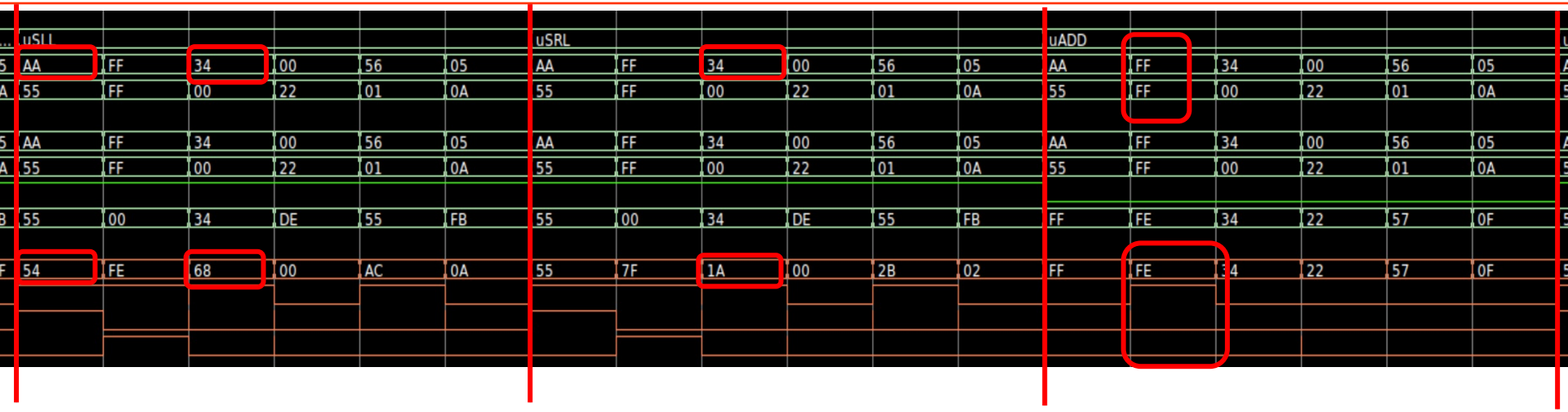
XOR: AA⊕55= FF, FF⊕FF=00,34⊕00=34, 00⊕22=22,56⊕1=57, 5⊕A=F

```
constant padrao_de_teste : padroes := (
    (a => x"AA",  b => x"55"),
    (a => x"FF",  b => x"FF"),
    (a => x"34",  b => x"00"),
    (a => x"00",  b => x"22"),
    (a => x"56",  b => x"01"),
    (a => x"05",  b => x"0A")
);
```

SSL (vezes 2)

SSR (div. por 2)

ADD



SSL: AA=54, FF=FE, 34=68, 00=00, 56=AC, 5=A

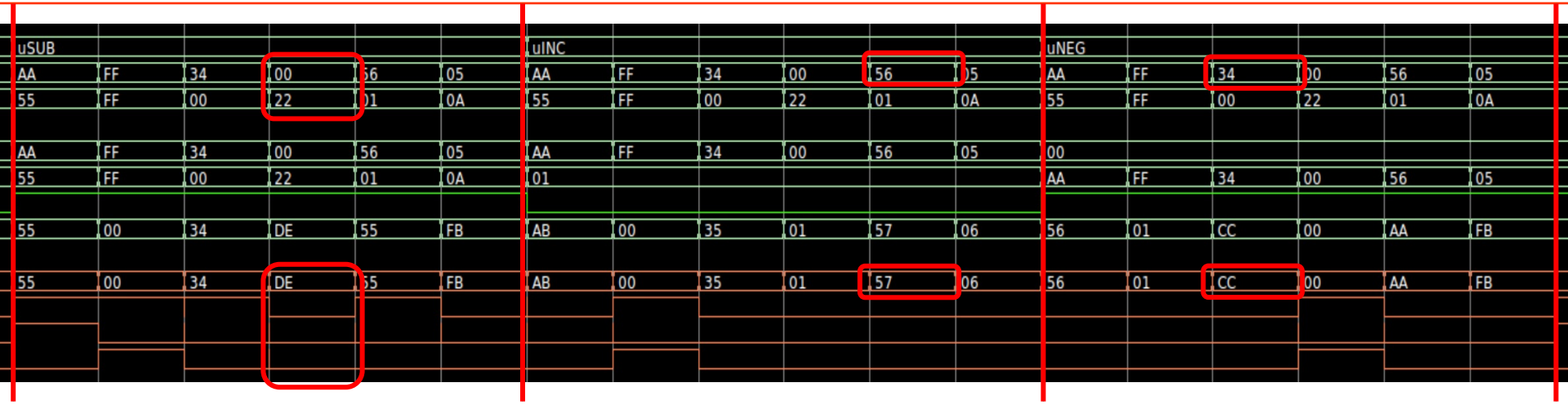
SSR: AA=55, FF=7F, 34=1A, 00=00, 56=2B, 5=2

ADD: AA+55= FF, FF+FF=(1)FE, 34+00=34, 00+22=22, 56+1=57, 5+A=F

SSL (esquerda): AA (1010 1010) → 0101 0100 = 54

SSR (direita): 34 (0011 0100) → 0001 1010 = 1A

SUB INC 2’S COMP



SUB: AA-55= 55, FF-FF=00, 34-00=34, 00-22=DE, 56-1=55, 5-A=FB

00000000 - 0010 0010 = 00000000 + 1101 1101 +1 = 1101 1110

INC: AA-AB, FF= 00, 34=35, 00=01, 56=57, 5=6

2’s: AA=56, FF=01, 34=CC, 00=00, 56=AA, 5=FB

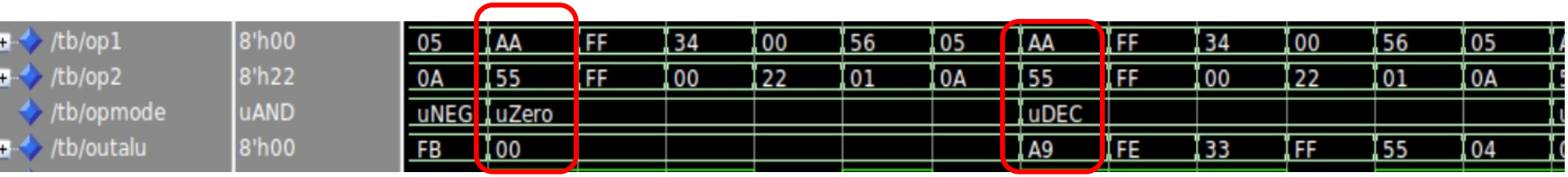
2’s de 0011 0100 = 1100 1011 + 1 = 1100 1100 = CC

Exercício

1. **Simular e entender o funcionamento da ULA**

2. **Modificar a ULA, acrescentando duas novas instruções:**
   uZero (outalu ← 0) **e uDEC** (outalu ← op1 - 1)

```
library IEEE;
use IEEE.Std_Logic_1164.all;

package p_ula is
        type op_alu is
          ( uAND, uOR, uXOR, uSLL, uSRL, uADD, uSUB, uINC, uNEG , uZero, uDec);
end p_ula;
```



Sempre zero

Exercício - Solução

Modificar a ULA, acrescentando duas novas instruções: uZero (outalu ← 0) **e uDEC** (outalu ← op1 - 1)

```
architecture alu of alu is
  signal aa, bb, soma: std_logic_vector(7 downto 0);
  signal mode: std_logic;
begin

  z <= '1' when soma = x"00" else '0';

  mode <= '0' when opmode=uADD or opmode=uINC or opmode=uDEC else '1';

  aa <= x"00" when opmode=uNEG else op1;

  bb <= x"01" when opmode=uINC else
        x"FF" when opmode=uDEC else
        op1   when opmode=uNEG or opmode=uZero else
        op2;

  sb: entity work.soma_sub
    generic map(8)
    port map (a => aa, b => bb, mode => mode,
              S => soma, cout => cout, ov => ov);

  outalu <= op1 and op2 when opmode=uAND else
            op1 or op2  when opmode=uOR else
            op1 xor op2 when opmode=uXOR else
            op1(6 downto 0) & '0' when opmode=uSLL else
            '0' & op1(7 downto 1) when opmode=uSRL else
            soma; --- uADD, uSUB, uINC, uNEG, DEC

end architecture;
```

uDec:
op1 + FF

uZero:
op1 – op1
op1 –not(op1) +1

CIRCUITOS COMBINACIONAIS

Multiplicador

MULTIPLICADOR

- **Dois números de *n* bits quando multiplicados, geram *2n* bits**

- **Exemplo:**

multiplicand 1101 (13)

multiplier * 1011 (11)

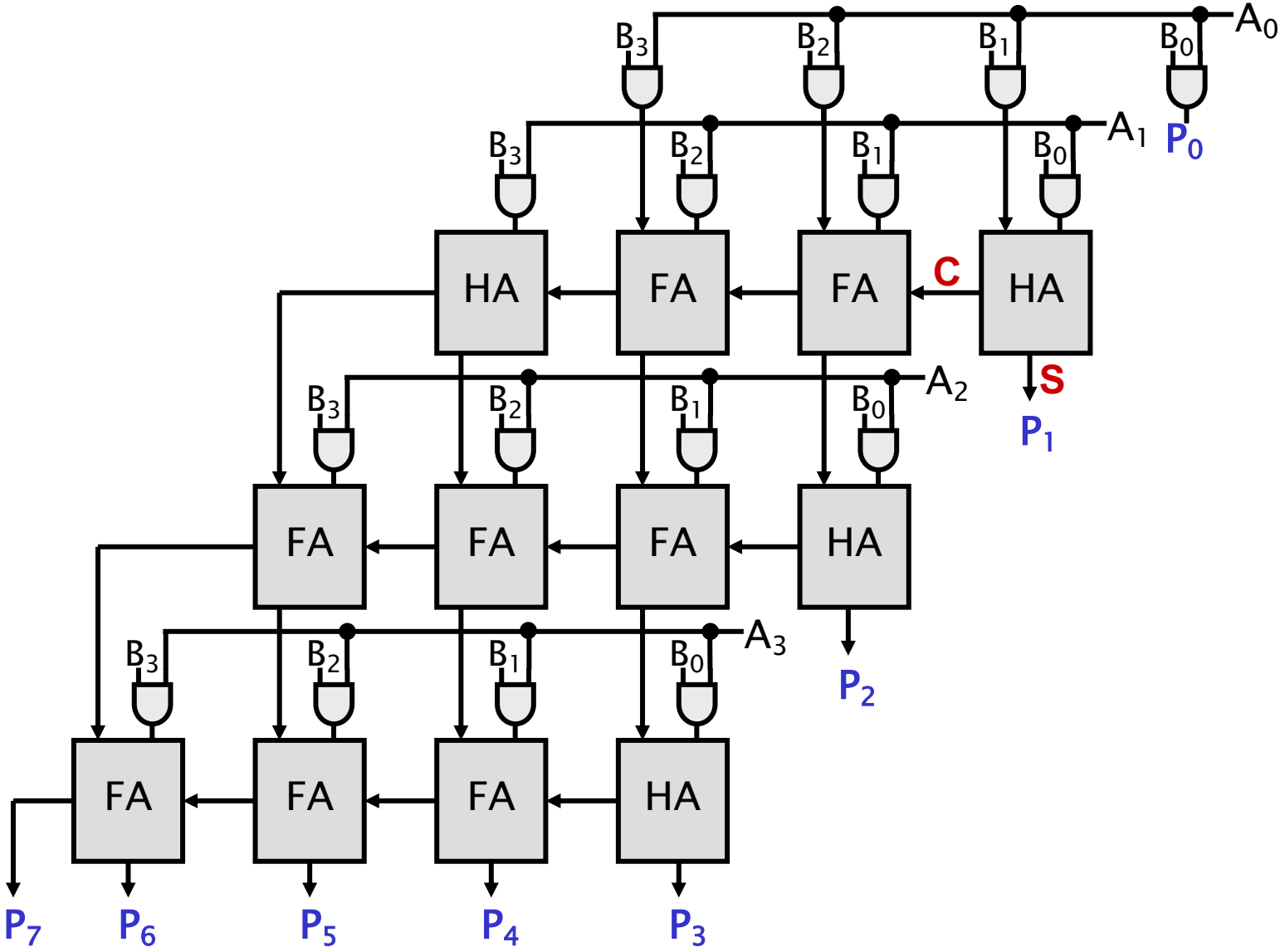
Partial products
1101
1101
0000
1101

10001111 (143)

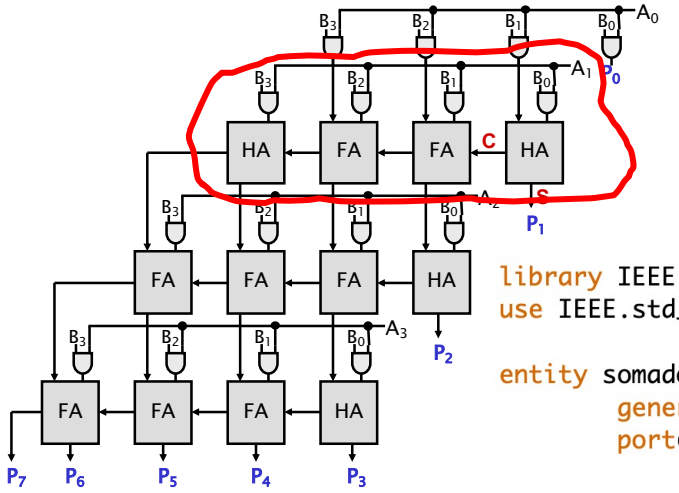
MULTIPLICAÇÃO

				A3	A2	A1	A0
				B3	B2	B1	B0
				A3 B0	A2 B0	A1 B0	A0 B0
			A3 B1	A2 B1	A1 B1	A0 B1	
		A3 B2	A2 B2	A1 B2	A0 B2		
	A3 B3	A2 B3	A1 B3	A0 B3			
S7	S6	S5	S4	S3	S2	S1	S0

MULTIPLICAÇÃO – arquitetura ”array”



Modelagem do Multiplicador (1/2)



Bloco básico - somador com FAs em série:

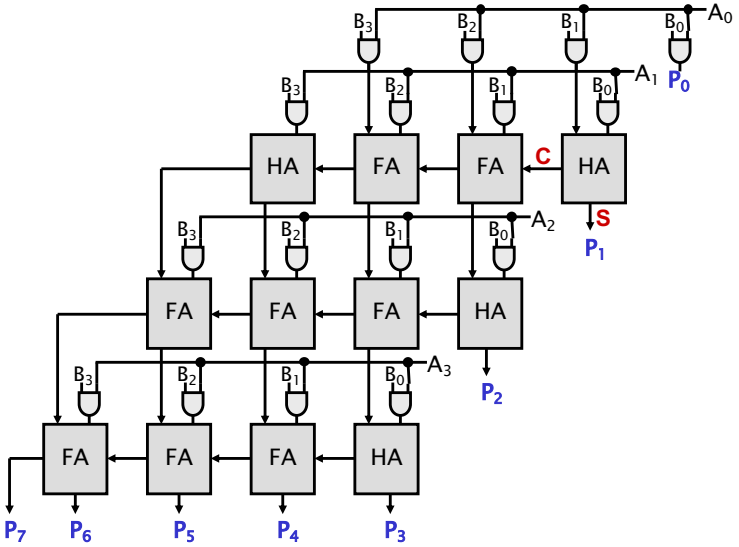
- vai-um igual a '0'
- vetor 'b' é 'mascarado' pelo bit A corrente
- o vetor soma de saída tem um bit a mais (Cout)

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

entity somador_serie is
        generic(N : integer := 8);
        port(   prev_sum, b  : in  std_logic_vector(N-1 downto 0);
                a            : in  std_logic;
                sum          : out std_logic_vector(N downto 0)
             );
end entity;

architecture a1 of  somador_serie is
   signal co: std_logic_vector(N downto 0);
   signal bb : std_logic_vector(N-1 downto 0);
begin

   co(0)   <= '0';
   sum(N)  <= co(N);

   add:  for i in 0  to N-1  generate
        bb(i)   <= b(i) and a;
        sum(i)  <= co(i) xor prev_sum(i) xor bb(i);
        co(i+1) <= (co(i) and prev_sum(i)) or (co(i) and bb(i)) or (prev_sum(i) and bb(i));
   end generate add;

end a1;
```

Soma_sub para comparação

```
co(0)  <= mode;
Cout   <= co(N);
OV     <= co(N) xor co(N-1);

 add: for i in 0 to N-1 generate
  begin
     bb(i)  <= b(i) xor mode;
     S(i)   <= co(i) xor A(i) xor bb(i);
     co(i+1) <= (co(i) and A(i)) or (co(i)
and bb(i)) or (A(i) and bb(i));
   end generate;

end architecture a1;
```

Modelagem do Multiplicador (2/2)




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

entity multiplicador is
        generic(N : integer := 4);
        port(   A : in  std_logic_vector(N-1 downto 0);
                B : in  std_logic_vector(N-1 downto 0);
                P : out std_logic_vector(N*2-1 downto 0)
            );
end entity;

architecture multiplicador of  multiplicador is
        type   matriz is array(0 to N-1) of std_logic_vector(N downto 0);
        signal sum : matriz;
begin

   -- primeira camada da multiplicação - só ands
   soma0:  for i in 0  to N-1  generate
        sum(0)(i) <= a(0) and b(i);
   end generate soma0;
   sum(0)(N) <= '0';

   linhas:  for i in 1 to N-1  generate

      s: entity work.somador_serie port map( prev_sum=> sum(i-1)(N downto 1),  b=>b, a=>a(i),
                                             sum => sum(i) );
   end generate linhas;

   pp:  for i in 0  to N-1  generate   -- resultado
        P(i) <= sum(i)(0);
        P(i+N) <= sum(N-1)(i+1);
   end generate pp;

end multiplicador;
```

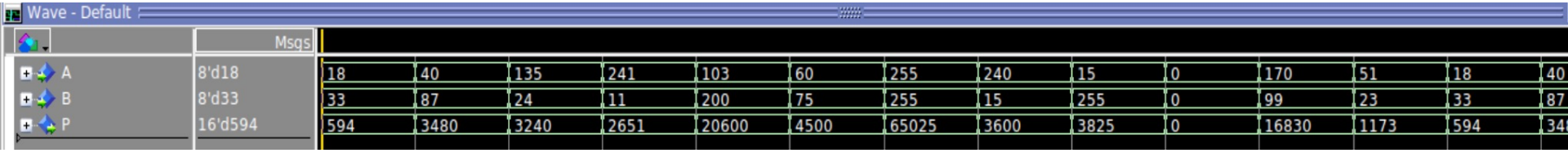
Multiplicador

- primeira linha: só portas lógicas and
- demais linhas: laço de geração
- terceiro laço: produto

Simulando

```
architecture tb of tb is

   constant N  : natural := 8;

   signal a    : std_logic_vector(N-1 downto 0);
   signal b    : std_logic_vector(N-1 downto 0);
   signal prod : std_logic_vector(N*2-1 downto 0);

   type test_record is record
       a, b :  integer;
   end record;

   type padroes is array(natural range <>) of test_record;

   constant padrao_de_teste : padroes := (
        (a =>  18,  b =>   33),  -- 594
        (a =>  40,  b =>   87),  -- 3480
        (a => 135,  b =>   24),  -- 3240
        (a => 241,  b =>   11),  -- 2651
        (a => 103,  b =>  200),  -- 20600
        (a =>  60,  b =>   75),  -- 4500
        (a => 255,  b =>  255),  -- 65025
        (a => 240,  b =>   15),  -- 3600
        (a =>  15,  b =>  255),  -- 3825
        (a =>   0,  b =>    0),  -- 0
        (a => 170,  b =>   99),  -- 16830
        (a =>  51,  b =>   23)   -- 1173
    );

begin
```

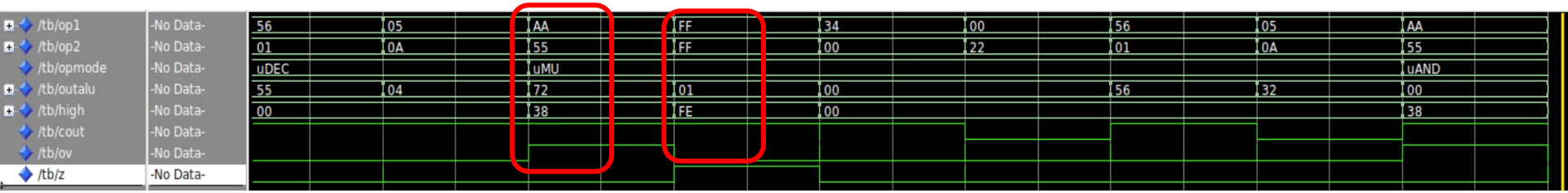
```
begin

    multiplicador:  entity work.multiplicador
              generic map(N)
              port map( A=>a, B=>b, P=>prod);

    test: process
     begin
        for i in 0 to padrao_de_teste'high loop
                a <= CONV_STD_LOGIC_VECTOR(padrao_de_teste(i).a, a'length);
                b <= CONV_STD_LOGIC_VECTOR(padrao_de_teste(i).b, b'length);
                wait for 10 ns;
        end loop;

    end process;

end architecture tb;
```



Exercício

1. **Simular e compreender o funcionamento do multiplicador** (fornecido no material de apoio)

2. **Acrescentar a operação uMU na ULA, sendo a saída *outula* a parte baixa da multiplicação, e uma nova saída de 8 bits, *high*, a parte alta da multiplicação.**



AA*55=3872 FF*FF=FE01

Guia para o exercício

1. Copie o *multip.vhd* para a pasta da ULA de 8 bits (já modificada com uZero e uDEC)

2. Modifique o *sim.do*, acrescentando a compilação do *multip.vhd :*

```
vcom -work work ula_pack.vhd
vcom -work work soma_sub.vhd
vcom -work work multip.vhd
vcom -work work ula.vhd
vcom -work work tb_ula.vhd
```

3. Altere o ula_pack.vhd para conter a nova instrução:

```
package p_ula is
        type op_alu is
            ( uAND, uOR, uXOR, uSLL, uSRL, uADD, uSUB, uINC, uNEG, uZero, uDEC, uMU);
end p_ula;
```

4. Altere o *ula.vhd*:
   (a) inserir o sinal *high* (8 bits) na *entity*;
   (b) inserir o sinal produto (16 bits) entre *architecture* e o *begin*
   (c) realizar o *port map* do multiplicador
   (d) na atribuição da ULA incluir o caso da multiplicação (*uMU*) atribuindo produto(7 downto 0) à outula
   (e) atribuir produto(15 downto 8) a high

5. Altere o *tb_ula.vhd*:
   (a) inserir o sinal high: *signal high: std_logic_vector(7 downto 0);*
   (b) inserir o sinal high no *port map* da ula