



**Universidade Federal de Santa Catarina
Centro Tecnológico – CTC
Departamento de Engenharia Elétrica**



<http://gse.ufsc.br>

“EEL7020 – Sistemas Digitais”

Prof. Eduardo Augusto Bezerra

Eduardo.Bezerra@eel.ufsc.br

Florianópolis, março de 2013.

Sistemas Digitais

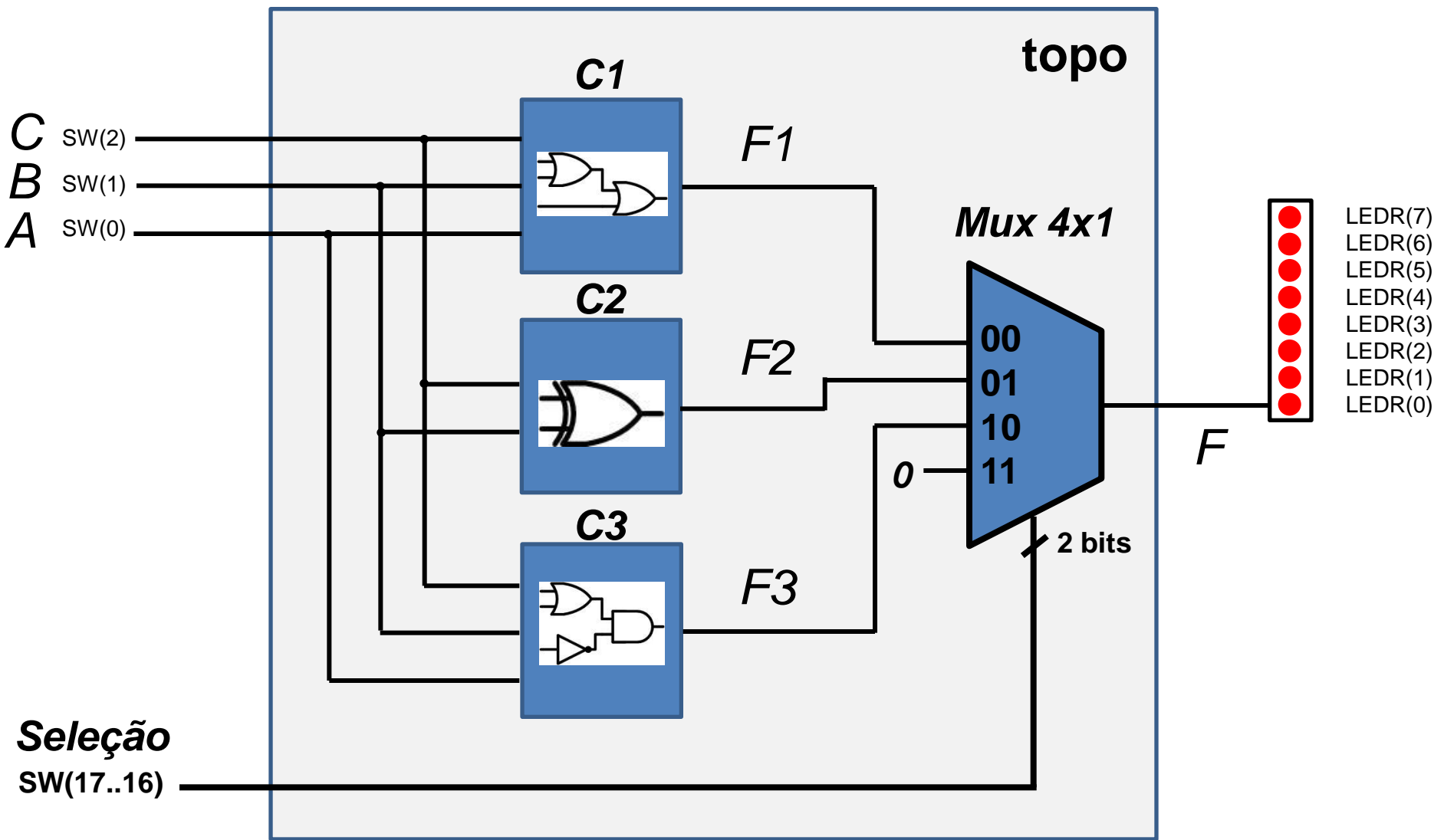
Circuitos codificadores e decodificadores

Objetivos do laboratório

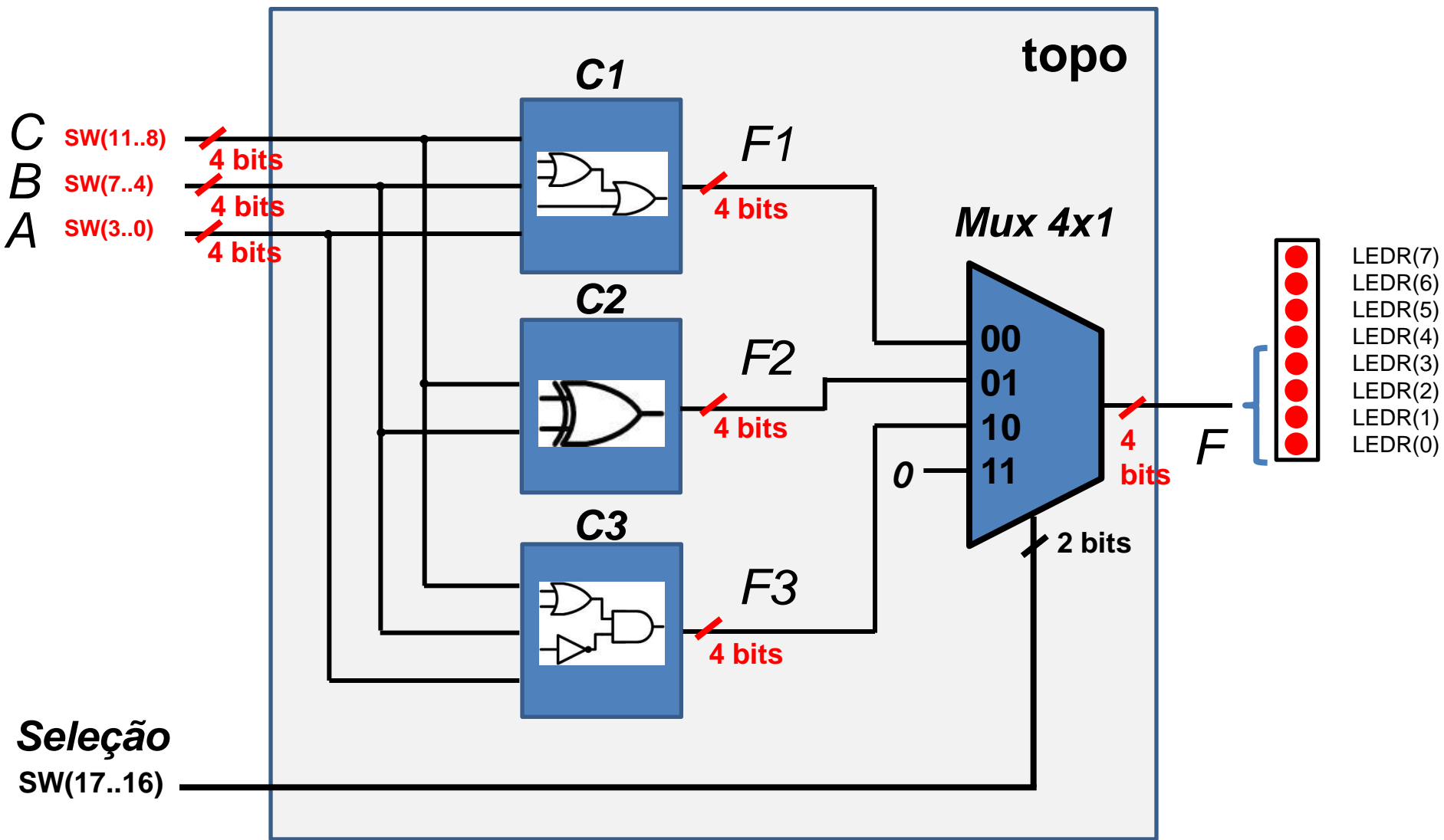
1. Entender o conceito de circuitos codificadores e decodificadores (conversores de código).
2. Implementação de codificadores e decodificadores em VHDL.
3. Uso de barramentos (vetores de sinais) em VHDL – *std_logic_vector*.
4. Estudo de caso: projeto de calculadora personalizada, com apresentação dos resultados em displays de 7-segmentos.

Barramentos em VHDL - “std_logic_vector”

No circuito do lab anterior, as entradas (A, B, C) e saídas (F) possuem o tamanho de 1 bit:



Assumir o mesmo circuito, porém com entradas e saídas de **4 bits**:



Utilizando operandos de 4 bits

- Componente C1 realiza a operação $F1 = A \text{ or } B \text{ or } C$
- Componente C2 realiza a operação $F2 = B \text{ xor } C$
- Operandos de **1 bit** - ex. $A = 0, B = 1, C = 0$
- Operandos de **4 bits** - ex. $A = 0101, B = 1000, C = 0001$

1 bit	
0	
1	0
0	1
<u> </u>	<u> </u>
$F1 = 1$	$F2 = 1$

4 bits	
0101	
1000	1000
0001	0001
<u> </u>	<u> </u>
$F1 = 1101$	$F2 = 1001$

Implementação de barramento em VHDL

std_logic → fio

std_logic_vector → vários fios (barramento)

- *std_logic_vector* é utilizado para implementar um “vetor” de sinais (vários fios, barramento).
- Por exemplo, a saída F deverá ser definida na *entity* como:

F: out **std_logic_vector**(3 downto 0);

Componente C1

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

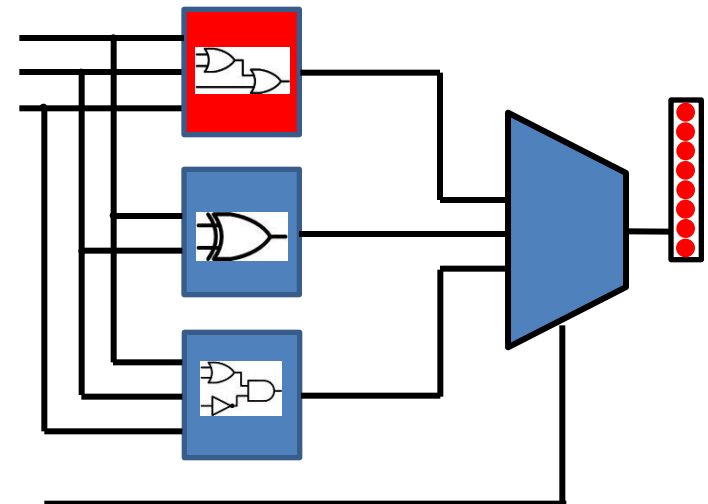
```
entity C1 is  
  port (A: in std_logic_vector(3 downto 0);  
        B: in std_logic_vector(3 downto 0);  
        C: in std_logic_vector(3 downto 0);  
        F: out std_logic_vector(3 downto 0)  
        );  
end C1;
```

```
architecture c1_estr of C1 is  
  begin  
    F <= A or B or C;  
  end c1_estr;
```

4 bits



4 bits



Componente C2

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

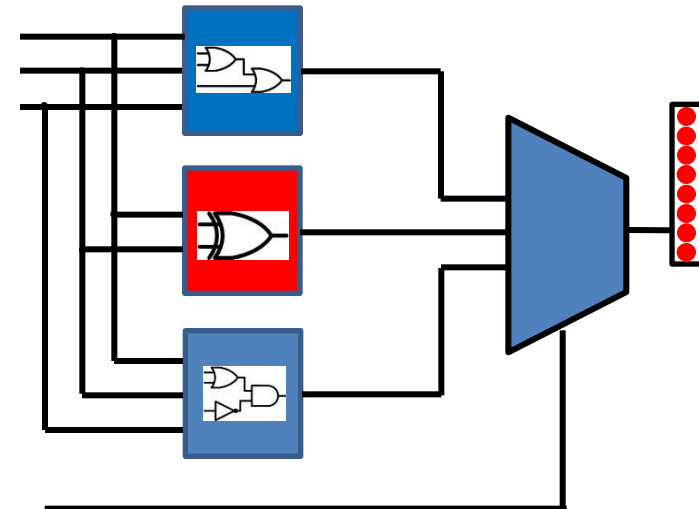
```
entity C2 is  
  port (A: in std_logic_vector(3 downto 0);  
        B: in std_logic_vector(3 downto 0);  
        F: out std_logic_vector(3 downto 0)  
        );  
end C2;
```

```
architecture c2_estr of C2 is  
  begin  
    F <= A xor B;  
  end c2_estr;
```

4 bits



4 bits



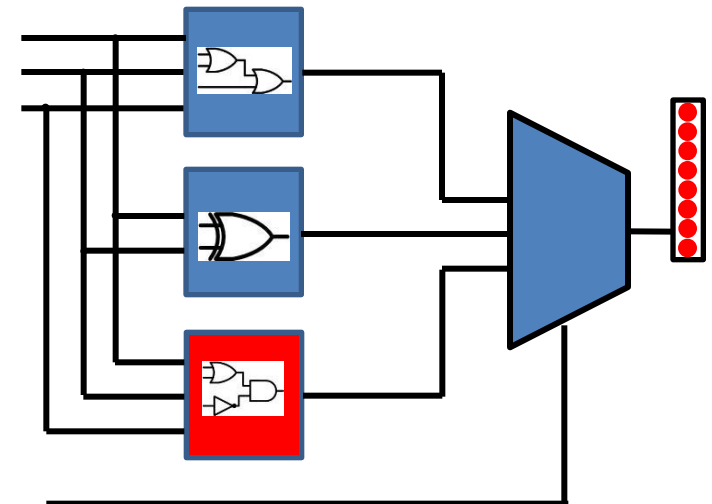
Componente C3

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

```
entity C3 is  
  port (A: in std_logic_vector(3 downto 0);  
        B: in std_logic_vector(3 downto 0);  
        C: in std_logic_vector(3 downto 0);  
        F: out std_logic_vector(3 downto 0)  
  );  
end C3;
```

```
architecture c3_estr of C3 is  
  begin  
    -- ver lab. sobre componentes  
  end c3_estr;
```

4 bits



Componente Mux

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

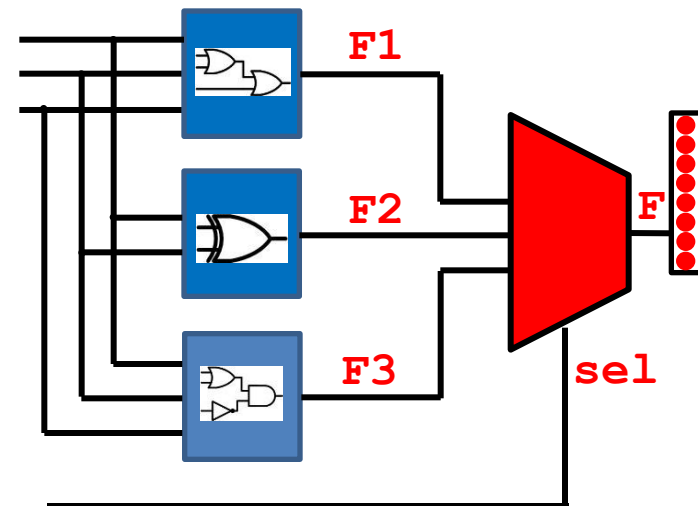
```
entity Mux is  
port (w: in std_logic_vector(3 downto 0);  
      x: in std_logic_vector(3 downto 0);  
      y: in std_logic_vector(3 downto 0);  
      z: in std_logic_vector(3 downto 0);  
      s: in std_logic_vector(1 downto 0);  
      m: out std_logic_vector(3 downto 0)  
);  
end Mux;
```

```
architecture mux_bhv of Mux is  
begin  
    -- ver lab. sobre Mux  
end mux_bhv;
```

4 bits



2 bits



Componente Topo

```

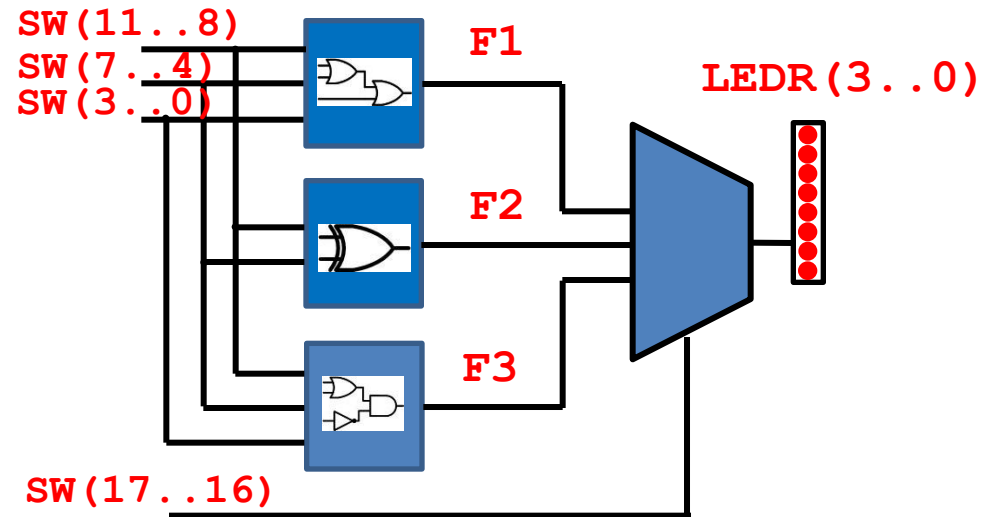
library ieee;
use ieee.std_logic_1164.all;
entity topo is
  port ( SW : IN STD_LOGIC_VECTOR(17 downto 0);
        LEDR : OUT STD_LOGIC_VECTOR(17 downto 0)
  );
end topo;
architecture topo_estru of topo is
  signal F1, F2, F3: std_logic_vector(3 downto 0);

  component C1
    port (A : in std_logic_vector(3 downto 0);
          B : in std_logic_vector(3 downto 0);
          C : in std_logic_vector(3 downto 0);
          F : out std_logic_vector(3 downto 0));
  end component;

  component C2
    port (A : in std_logic_vector(3 downto 0);
          B : in std_logic_vector(3 downto 0);
          F : out std_logic_vector(3 downto 0));
  end component;

  component C3
    port (A : in std_logic_vector(3 downto 0);
          B : in std_logic_vector(3 downto 0);
          C : in std_logic_vector(3 downto 0);
          F : out std_logic_vector(3 downto 0)
    );
  end component;
  -- INCLUIR AQUI O Mux

```



```

begin

  L0: C1 port map (SW(3 downto 0),
                  SW(7 downto 4), SW(11 downto 8), F1);

  L1: C2 port map (SW(3 downto 0),
                  SW(7 downto 4), SW(11 downto 8), F2);

  L2: C3 port map (SW(3 downto 0),
                  SW(7 downto 4), SW(11 downto 8), F3);

  L3: Mux port map (F1, F2, F3,
                  SW(17 downto 16), LEDR(3 downto 0));

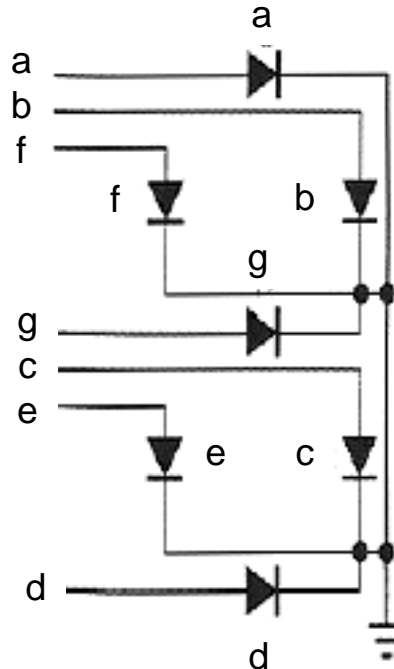
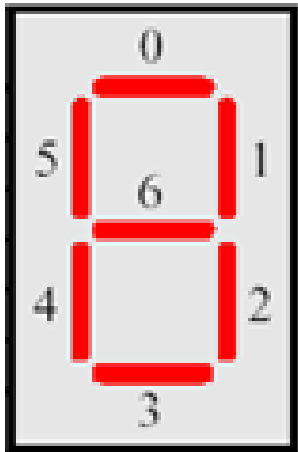
end topo_estru;  -- END da architecture

```

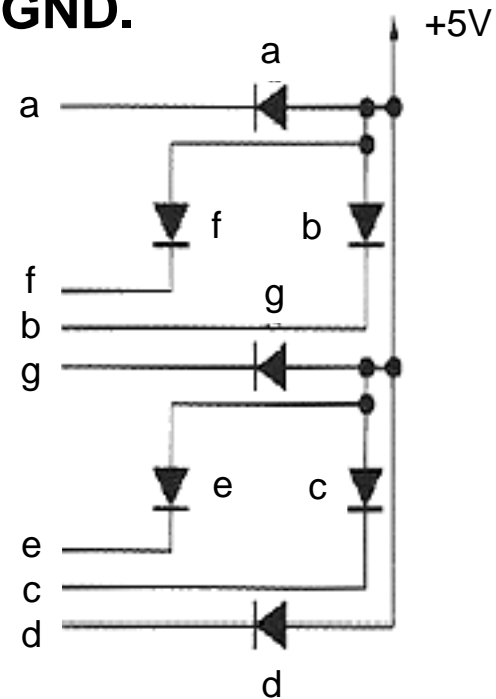
Decodificadores em VHDL

Display de 7-segmentos

- Um display de 7-segmentos é composto por sete LEDs que podem ser ligados ou desligados de forma independente.
- **Catodo comum** - terminais catodo dos LEDs estão conectados a GND, e cada LED é ligado ao conectar seu anodo em Vcc.
- **Anodo comum** - terminais anodo dos LEDs estão conectados a Vcc, e cada LED é ligado ao conectar seu catodo em GND.



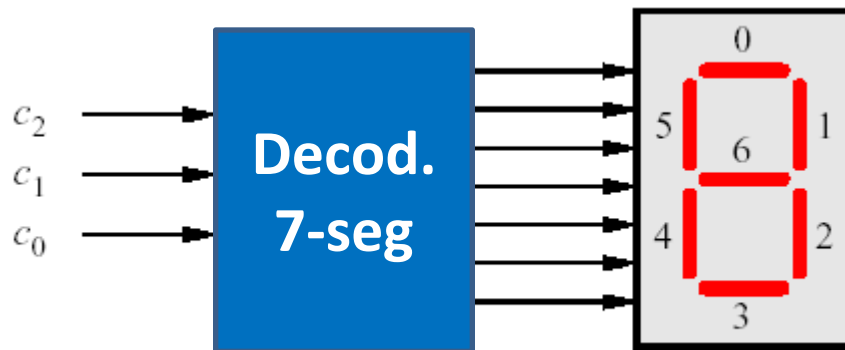
Catodo comum



Anodo Comum

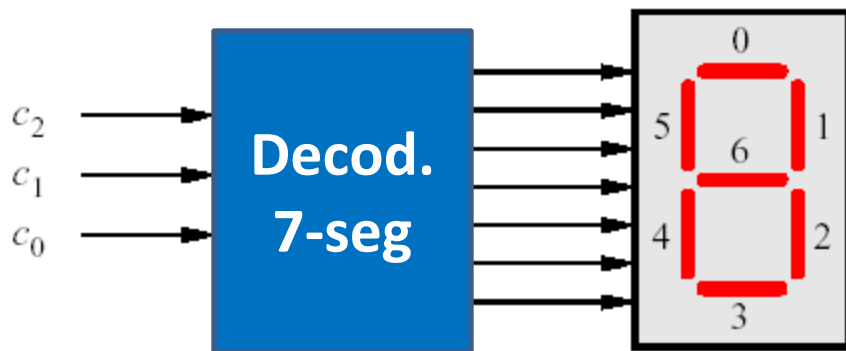
Decodificador de binário para 7-segmentos

- Placa DE2 possui 7 displays de 7-segmentos, todos do tipo anodo comum (LEDs acendem com GND, ou zero lógico).
- Para escrever um valor binário em um dos displays, é preciso realizar uma conversão do código binário para o código 7-segmentos.
- Os 3 bits de entrada do circuito “Decod. 7-seg” são *decodificados*, e a palavra de 7 bits gerada é enviada para o display de 7 segmentos.



Projeto de decodificador de binário para 7-segmentos

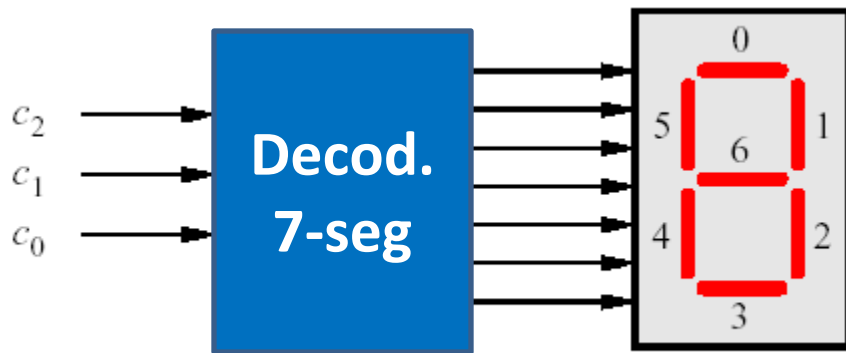
- Exemplos de valores em binários convertidos (decodificados) para 7-segmentos, visando escrita no display da placa DE2 anodo comum)



			7 bits	Letra
C2	C1	C0	6543210	
0	0	0	1000001	U
0	0	1	0001110	F
0	1	0	0010010	S
0	1	1	1000110	C
1	1	1	1111111	

- Nesse exemplo, ao receber “000” na entrada, o decodificador gera o código equivalente ao acendimento da letra “U” no display 7-seg.
- Ao receber “111”, todos os segmentos são desligados.
- Notar que por ser do tipo anodo comum, um “0” liga um segmento.

Projeto de decodificador “binário para 7-segmentos”



			7 bits	
C2	C1	C0	6543210	Letra
0	0	0	1000001	U
0	0	1	0001110	F
0	1	0	0010010	S
0	1	1	1000110	C
1	1	1	1111111	

• Um circuito para implementar a lógica do decodificador em questão poderia ser projetado utilizando **vários métodos**:

- **Soma de produtos:**

$$F(0) = C2'C1'C0' + C2C1C0$$

$$F(1) = C2'C1'C0 + C2'C1C0' + C2'C1C0 + C2C1C0$$

$$F(2) = \dots$$

- **Análise comportamental:**

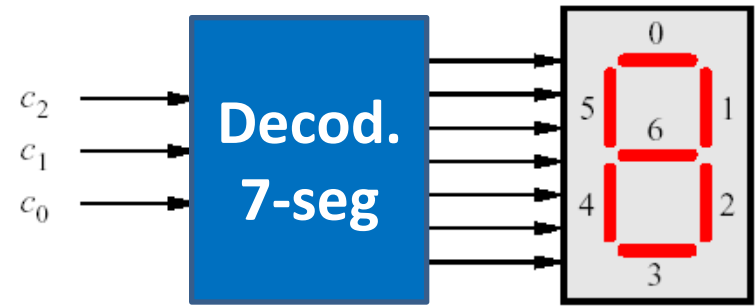
F = “1000001” quando C2C1C0 = “000” senão

“0001110” quando C2C1C0 = “001” senão

... senão

“1111111”

Componente Decod_UFSC



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

```
entity decodUFSC is  
  port (C: in std_logic_vector(2 downto 0);  
        F: out std_logic_vector(6 downto 0)  
        );  
end decodUFSC;
```

3 bits

7 bits

```
architecture decod_bhv of decodUFSC is  
begin  
  F <= "1000001" when C = "000" else -- U  
       "0001110" when C = "001" else -- F  
       "0010010" when C = "010" else -- S  
       "1000110" when C = "011" else -- C  
       "1111111";  
end decod_bhv;
```

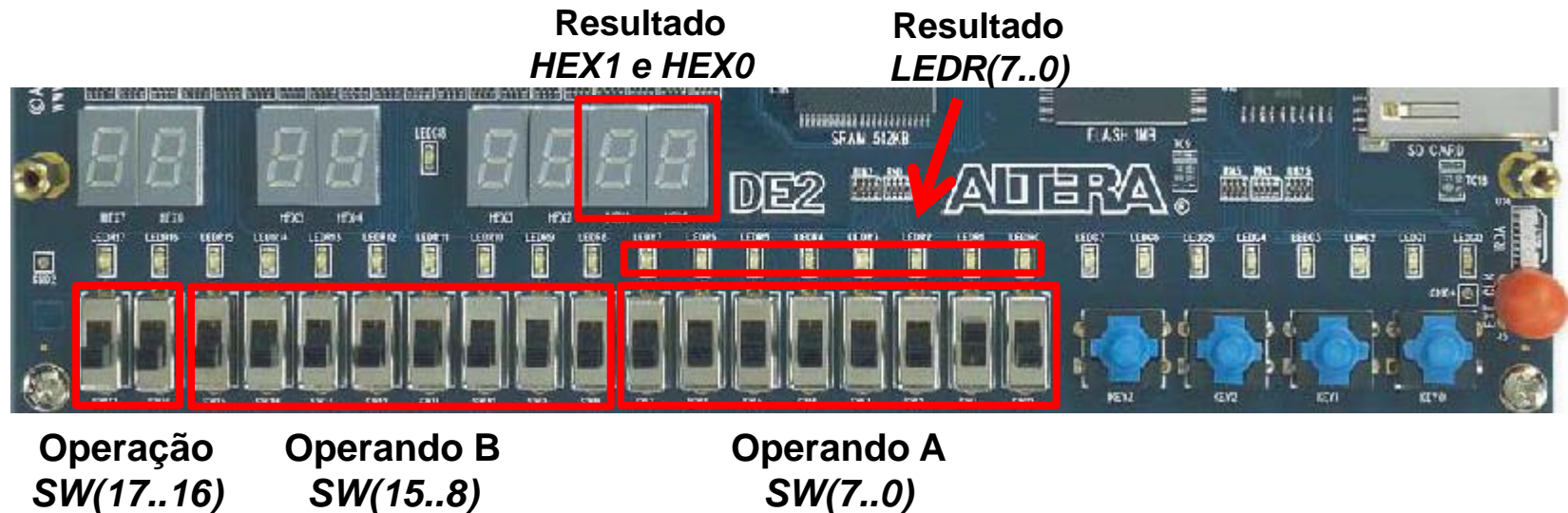
Tarefa a ser realizada na aula prática

Implementação de mini-calculadora de 8 bits personalizada

Descrição funcional da mini-calculadora de 8 bits

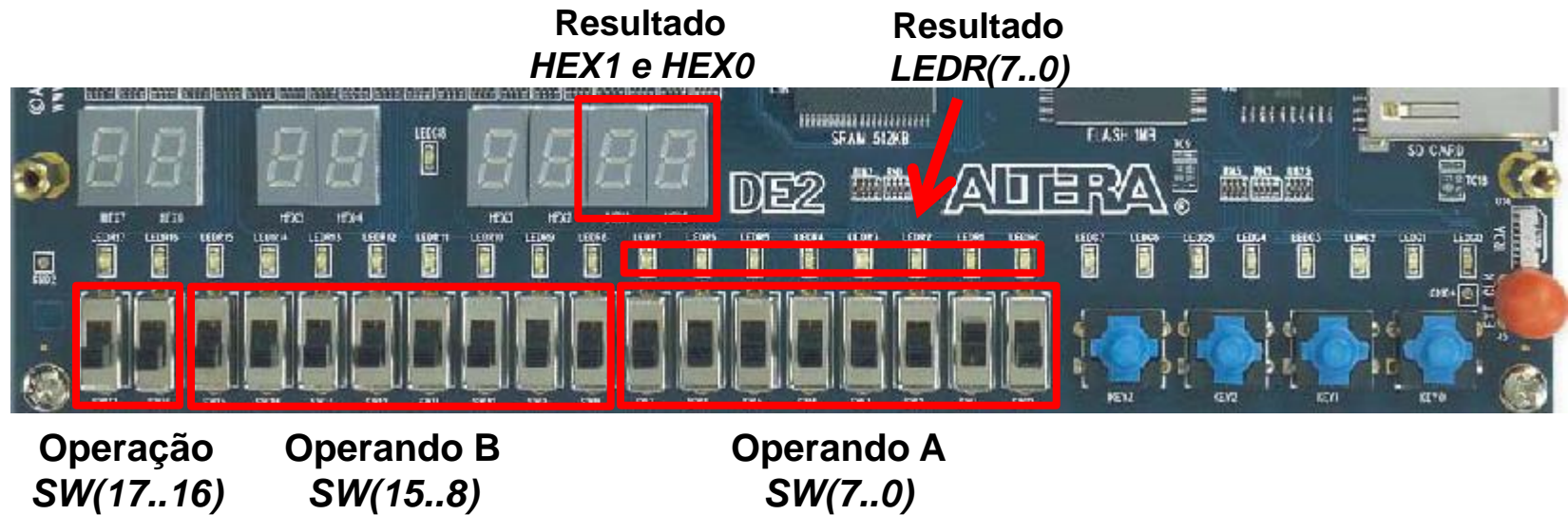
- A mini-calculadora realiza uma operação aritmética e três operações lógicas, todas de **8 bits**:
 - $F1 = A + B$ -- operação aritmética ADIÇÃO
 - $F2 = A \text{ or } B$
 - $F3 = A \text{ xor } B$
 - $F4 = \text{not } A$
- Para utilizar a mini-calculadora é necessário:
 1. Fornecer o operando A nas chaves SW(7..0).
 2. Fornecer o operando B nas chaves SW(15..8).
 3. Selecionar a operação desejada nas chaves SW(17..16).
 4. O resultado será apresentado em **HEXADECIMAL** nos displays de sete segmentos (HEX0 e HEX1), e em **BINÁRIO** nos LEDs vermelhos (LEDR).

Interface com o usuário



- Para realizar uma das quatro operações disponíveis (F1, F2, F3, F4), a calculadora personalizada utiliza:
 - as chaves SW(7..0) para leitura do **operando A**
 - as chaves SW(15..8) para leitura do **operando B**
 - as chaves SW(17..16) para seleção da operação desejada
- Os resultados são apresentados em displays de 7-segmentos e nos LEDs vermelhos.

Interface com o usuário



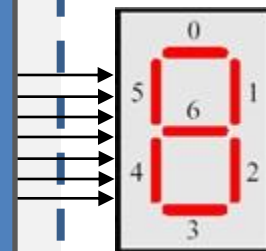
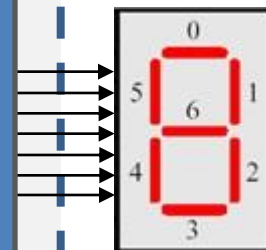
Seletor SW(17..16)	Saída (LEDR e displays 7-seg)
00	A + B
01	A or B
10	A xor B
11	not A

Operando B
SW(15 downto 8)

Operando A
SW(7 downto 0)

FPGA

**Componente
TOP_CALC**
(esse componente possui 7
componentes internamente)



LEDR(7 downto 0)

Seleção da operação

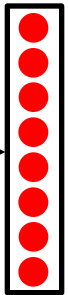
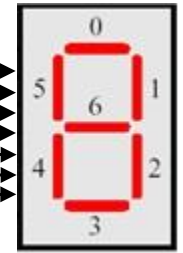
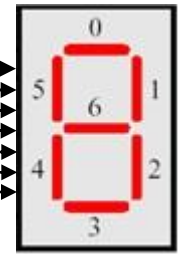
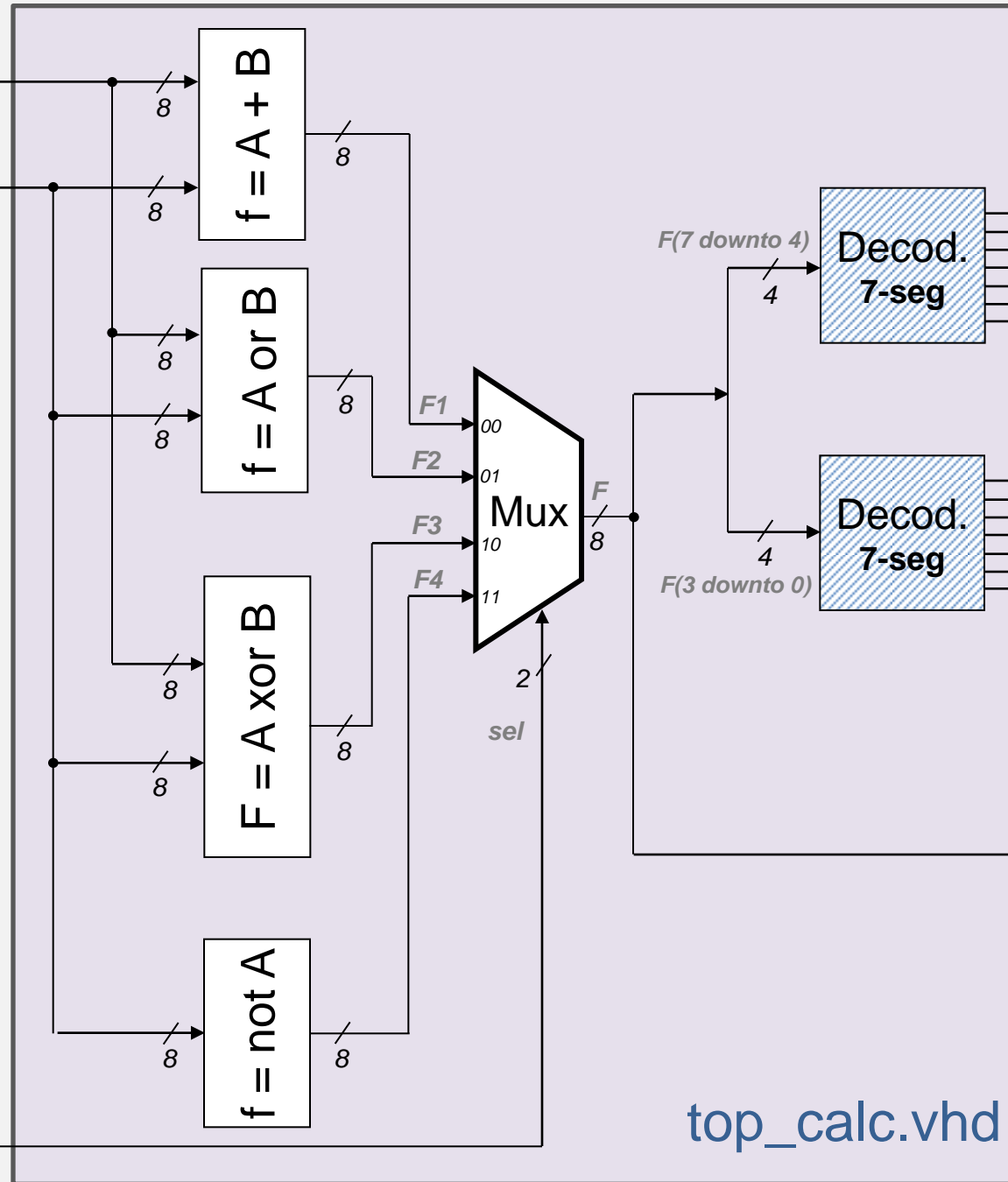
SW(17 downto 16)

Operando B
SW(15 downto 8)

Operando A
SW(7 downto 0)

FPGA

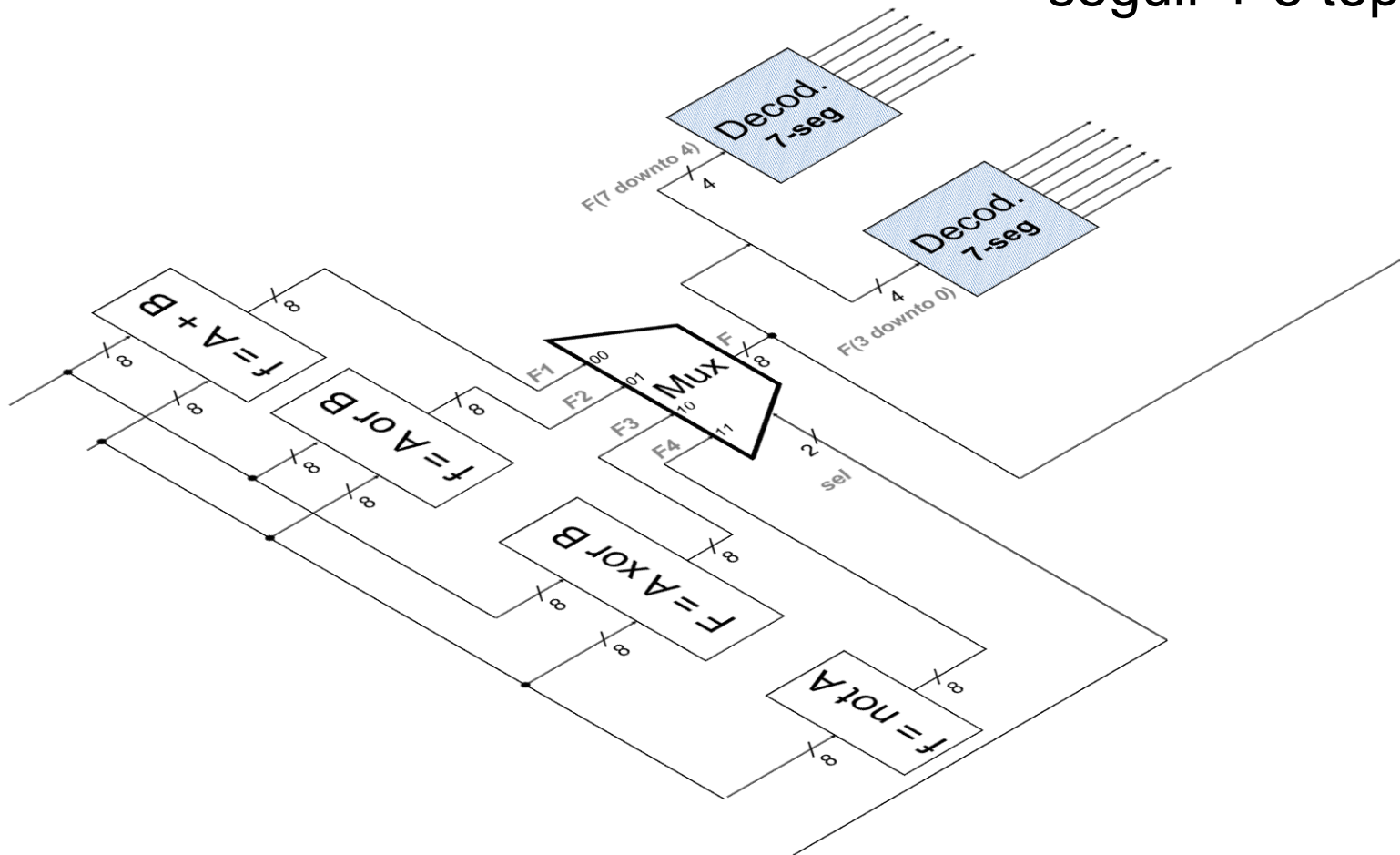
Seleção da operação
(+, or, xor, not)
SW(17 downto 16)



`top_calc.vhd`

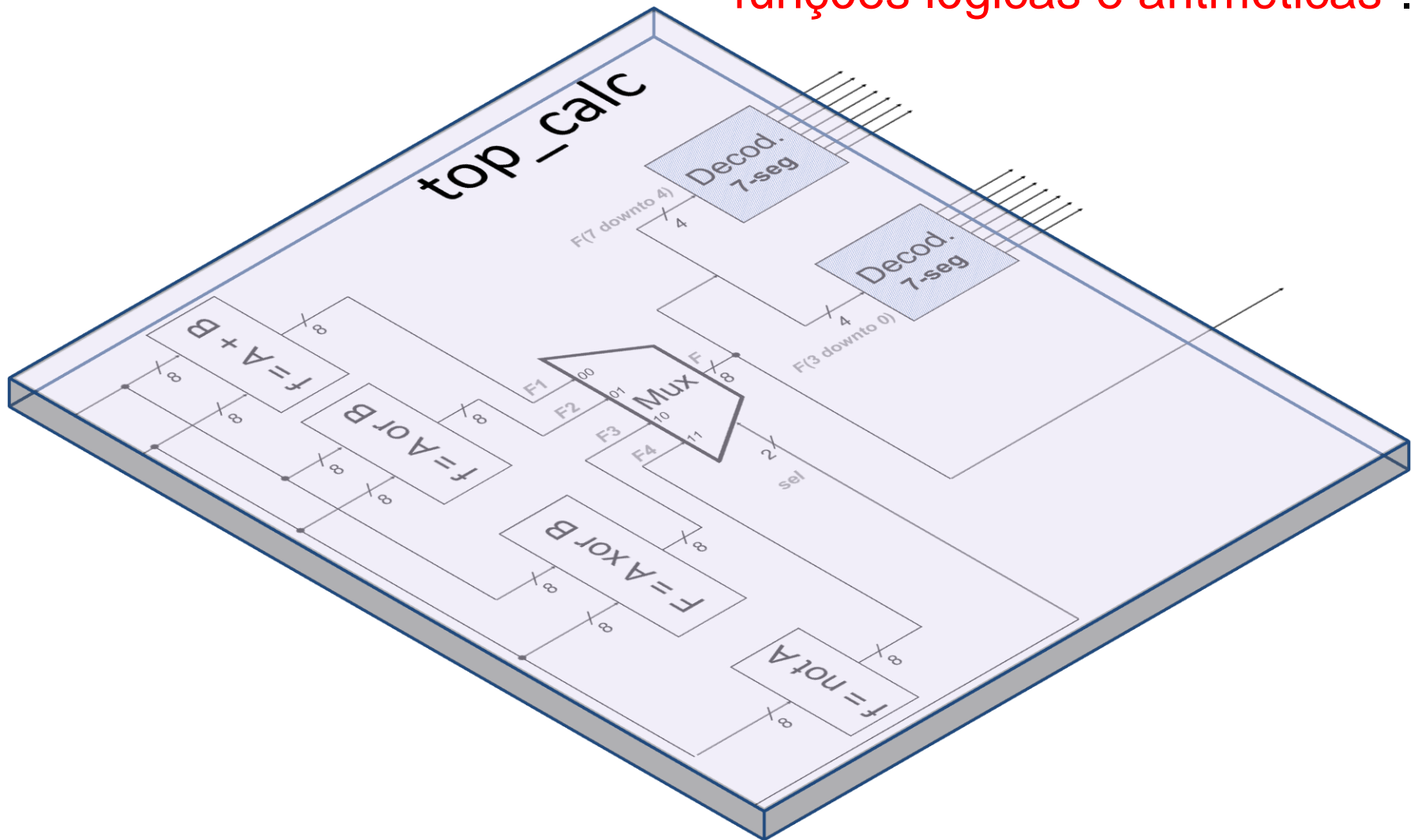
Projeto hierárquico

Calculadora é composta por 8 componentes, sendo 7 apresentados a seguir + o top_calc.

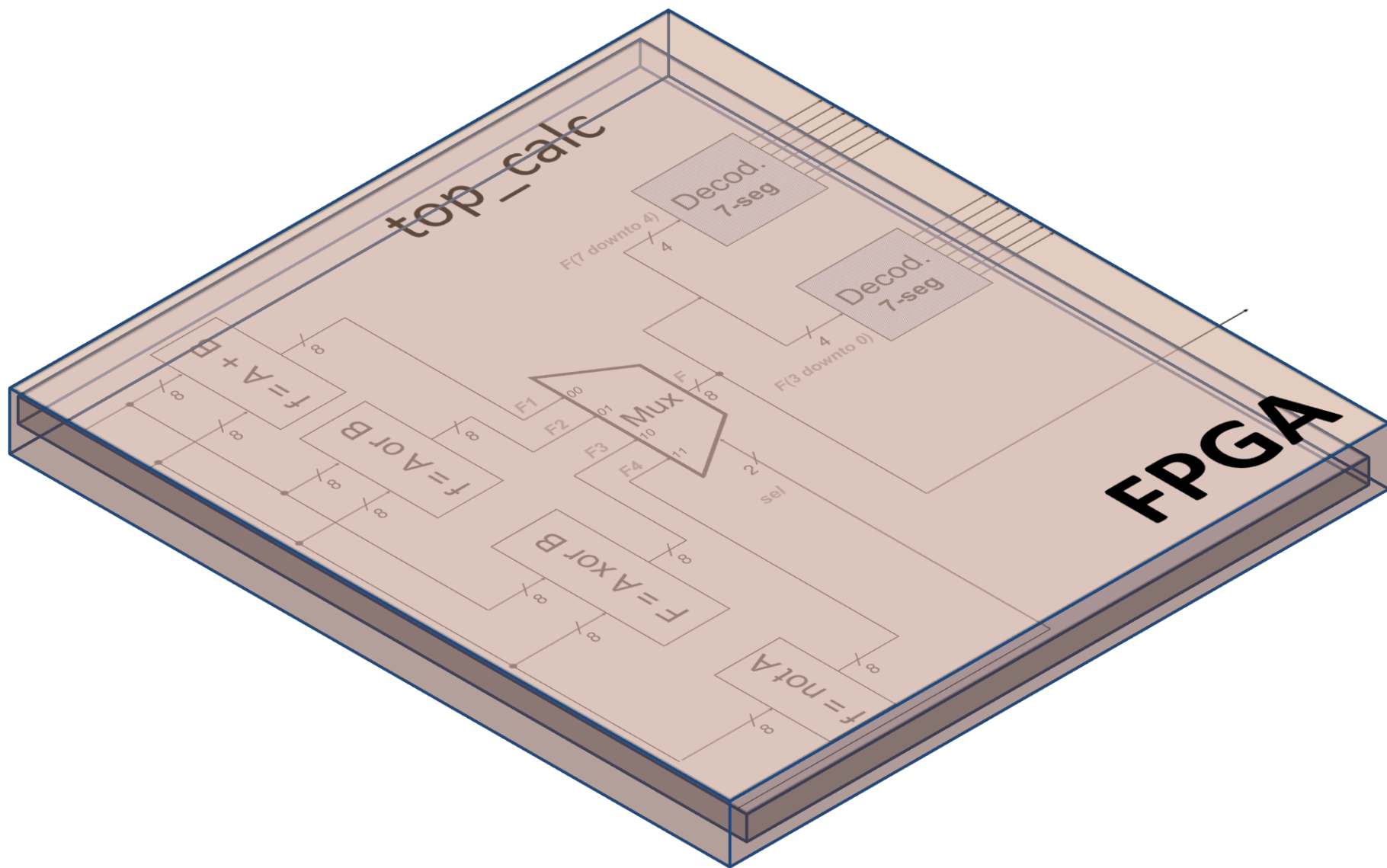


Projeto hierárquico

“top_calc”, “mux”, dois “decodificadores”, e quatro “funções lógicas e aritméticas”.



Projeto hierárquico

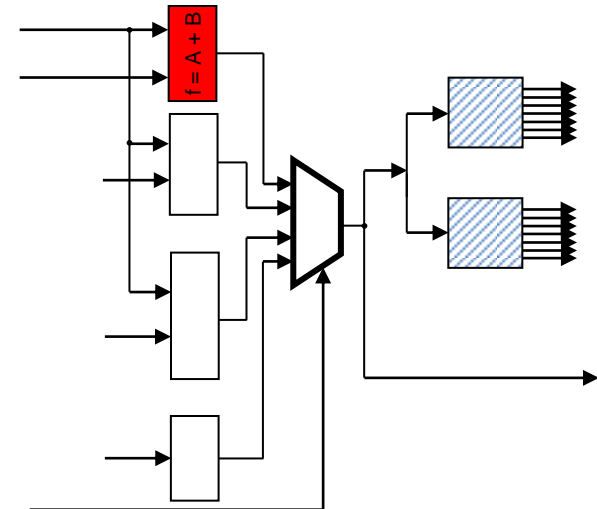


Componente C1 (arquivo *c1.vhd*)

```
library IEEE;  
use IEEE.Std_Logic_1164.all;  
use IEEE.std_logic_unsigned.all; -- necessário para o +
```

```
entity C1 is  
  port (A: in std_logic_vector(7 downto 0);  
        B: in std_logic_vector(7 downto 0);  
        F: out std_logic_vector(7 downto 0)  
        );  
end C1;
```

```
architecture circuito of C1 is  
  begin  
    F <= A + B;  
  end circuito;
```

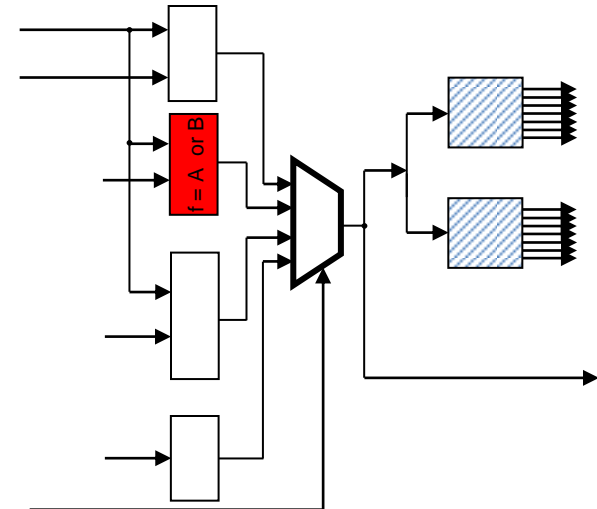


Componente C2 (arquivo *c2.vhd*)

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

entity C2 is
port (A: in std_logic_vector(7 downto 0);
      B: in std_logic_vector(7 downto 0);
      F: out std_logic_vector(7 downto 0)
    );
end C2;

architecture circuito of C2 is
begin
    F <= A or B;
end circuito;
```

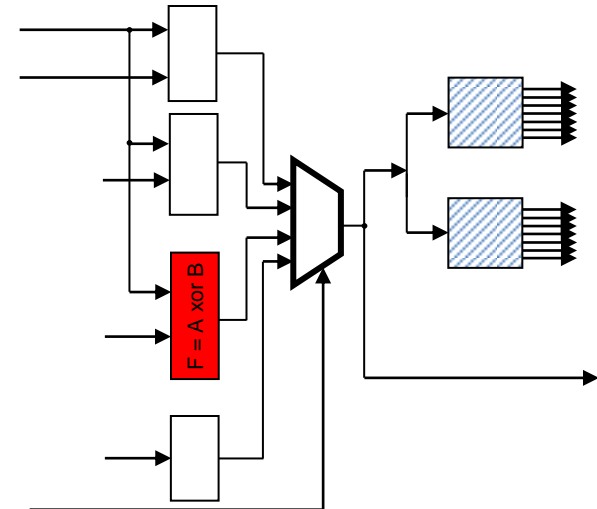


Componente C3 (arquivo *c3.vhd*)

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

```
entity C3 is  
port (A: in std_logic_vector(7 downto 0);  
      B: in std_logic_vector(7 downto 0);  
      F: out std_logic_vector(7 downto 0)  
      );  
end C3;
```

```
architecture circuito of C3 is  
begin  
    F <= A xor B;  
end circuito;
```

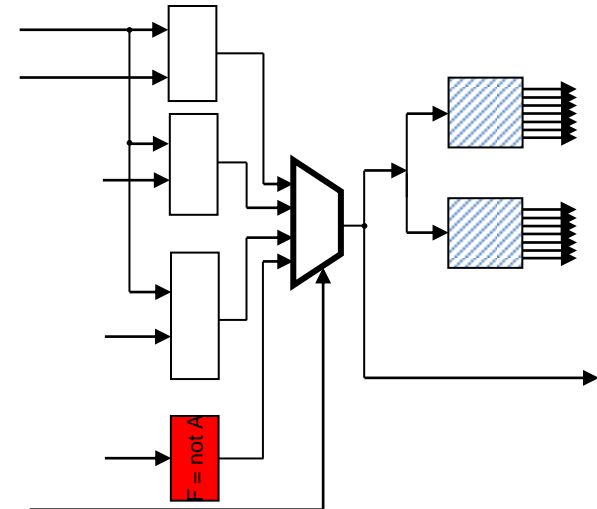


Componente C4 (arquivo *c4.vhd*)

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

entity C4 is
port (A: in std_logic_vector(7 downto 0);
      F: out std_logic_vector(7 downto 0)
    );
end C4;

architecture circuito of C4 is
begin
    F <= not A;
end circuito;
```

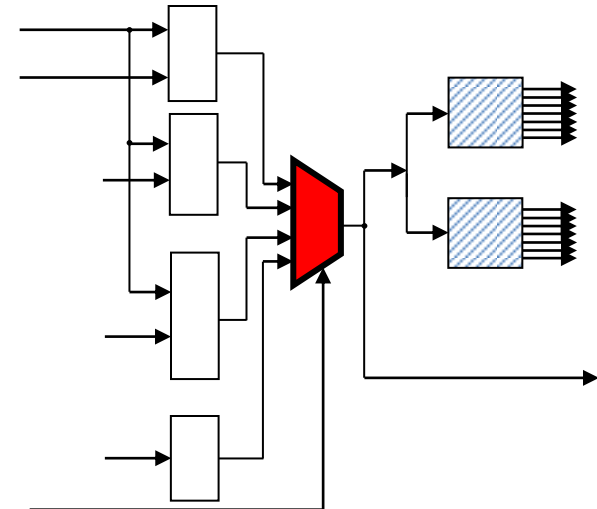


Componente Mux (arquivo *mux4x1.vhd*)

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

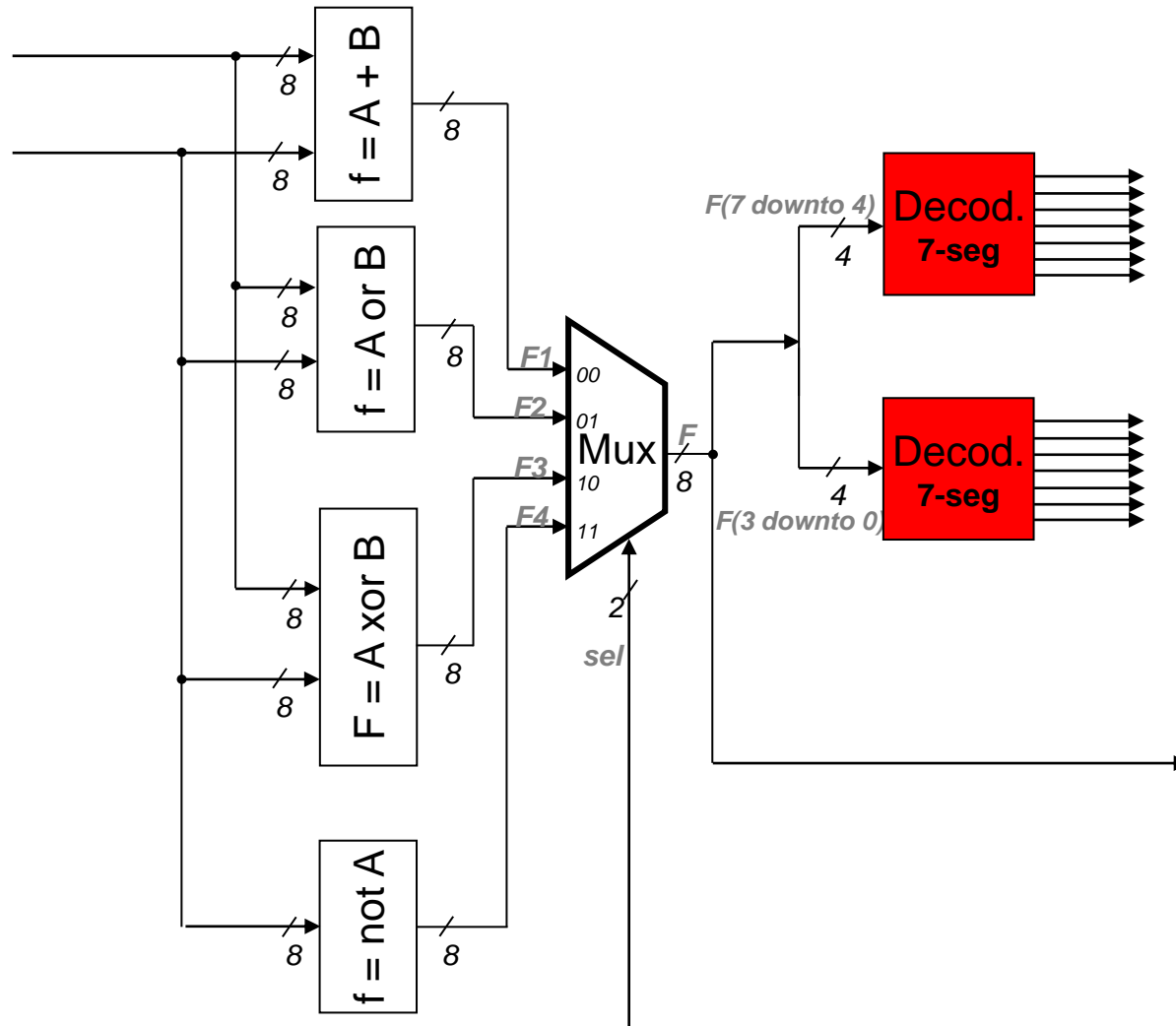
```
entity mux4x1 is  
  port (w, x, y, z: in std_logic_vector(7 downto 0);  
        s: in std_logic_vector(1 downto 0);  
        m: out std_logic_vector(7 downto 0)  
  );  
end mux4x1;
```

```
architecture circuito of mux4x1 is  
begin  
  m <= w when s = "00" else  
    x when s = "01" else  
    y when s = "10" else  
    z;  
end circuito;
```



Componente Decod7seg

Componente a ser implementado no presente lab.



Componente top_calc (arquivo *top_calc.vhd*)

```
library ieee;
use ieee.std_logic_1164.all;
entity top_calc is
  port ( SW : in std_logic_vector (17 downto 0);
        HEX0, HEX1: out std_logic_vector(6 downto 0);
        LEDR : out std_logic_vector (17 downto 0)
  );
end top_calc;
architecture topo_stru of top_calc is
  signal F, F1, F2, F3, F4: std_logic_vector (7 downto 0);
  component C1
    port (A : in std_logic_vector(7 downto 0);
          B : in std_logic_vector(7 downto 0);
          F : out std_logic_vector(7 downto 0));
  end component;
  -- componentes C2 e C3, idem C1
  component C4
    port (A : in std_logic_vector(7 downto 0);
          F : out std_logic_vector(7 downto 0)
    );
  end component;
  component mux4x1
    port (w, x, y, z: in
          std_logic_vector(7 downto 0) ;
          s: in std_logic_vector(1 downto 0) ;
          m: out std_logic_vector(7 downto 0)
    );
  end component;
  -- Incluir aqui o componente Decod7seg
  -- (apenas uma declaração do Decod7seg)
```

begin

L1: C1 port map (SW(7 downto 0),
SW(15 downto 8), F1);

L2: C2 port map (SW(7 downto 0),
SW(15 downto 8), F2);

L3: C3 port map (SW(7 downto 0),
SW(15 downto 8), F3);

L4: C4 port map (SW(7 downto 0), F4);

L5: mux4x1 port map (F1, F2, F3, F4,
SW(17 downto 16), F);

→ L6: Decod7seg port map (F(3 downto 0),
HEX0);

→ L7: Decod7seg port map (F(7 downto 4),
HEX1);

LEDR(7 downto 0) <= F;

end topo_stru; -- END da architecture

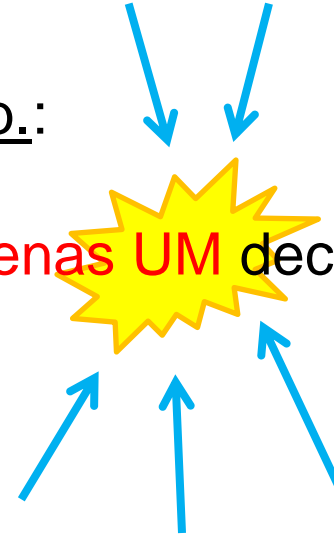
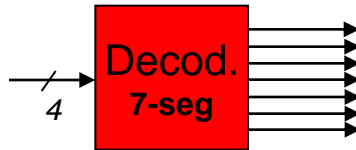
Dicas úteis

1. O projeto é composto por 7 arquivos (8 componentes):
 - *c1.vhd* – fornecido nos slides anteriores (slide 29)
 - *c2.vhd* – fornecido nos slides anteriores (slide 30)
 - *c3.vhd* – fornecido nos slides anteriores (slide 31)
 - *c4.vhd* – fornecido nos slides anteriores (slide 32)
 - *mux4x1.vhd* – fornecido nos slides anteriores (slide 33)
 - *decod7seg.vhd* – a ser desenvolvido (ver exemplo no slide 19)
 - *top_calc.vhd* – parcialmente fornecido nos slides anteriores (slide 35)
2. Cuidar a utilização correta do componente decodificador no circuito (topo). É necessário **definir apenas um *component decod7seg***, e a seguir criar **duas cópias desse componente com o port map** (ver *top_calc* no slide 35).
3. Para implementar o decodificador de binário para 7-segmentos, utilizar como base o exemplo do slide 19. Completar a tabela do slide 38, de forma a obter todos os códigos em hexadecimal necessários.

Componente Decod7seg

Componente a ser desenvolvido no presente lab.:

- Está sendo solicitada a implementação de **apenas UM** decodificador



- A *entity* desse decodificador deverá possuir apenas **uma porta de entrada de 4 bits**, e apenas **uma porta de saída de 7 bits**.
- A replicação do decodificador é realizada no “topo” da arquitetura, ao se utilizar o comando **port map** do VHDL – VER **L6:** e **L7:** no slide 35.

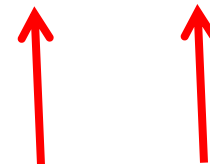


Tabela de decodificação binário para 7-segmentos

Entrada	Saída <i>6543210</i>	Display
0000	1000000	0
0001	1111001	1
0010	0100100	2
0011	0110000	3
0100	0011001	4
0101	0010010	5
0110	0000010	6
0111	1111000	7
1000	0000000	8
...	...	9, A, b, C, d
1110	0000110	E
1111	0001110	F

Resumo da tarefa a ser realizada

1. Implementar o decodificador “binário para 7-segmentos” (slides 19 e 38).
2. Esse novo componente será um **decodificador genérico**, ou seja, poderá ser utilizado em qualquer projeto que precise de um decodificador com a tabela verdade do slide 38. Assim, não deverá ter nenhum “ajuste especial” para funcionamento com os demais componentes da calculadora. Deverá possuir a entrada de 4 bits e a saída de 7 bits definida no slide 38.
3. Criar um novo projeto denominado **top_calc** (esse é o nome da *entity* do arquivo “topo” – slide 35), e utilizar os componentes VHDL dos slides 29 a 33 para implementação da calculadora.
4. Editar o componente **top_calc** do slide 35, e incluir a declaração do novo componente decodificador (usando **component**).
5. Verificar o funcionamento da calculadora no simulador e no kit FPGA.

Animação do **PORT MAP**

begin

end topo_stru; -- END da architecture

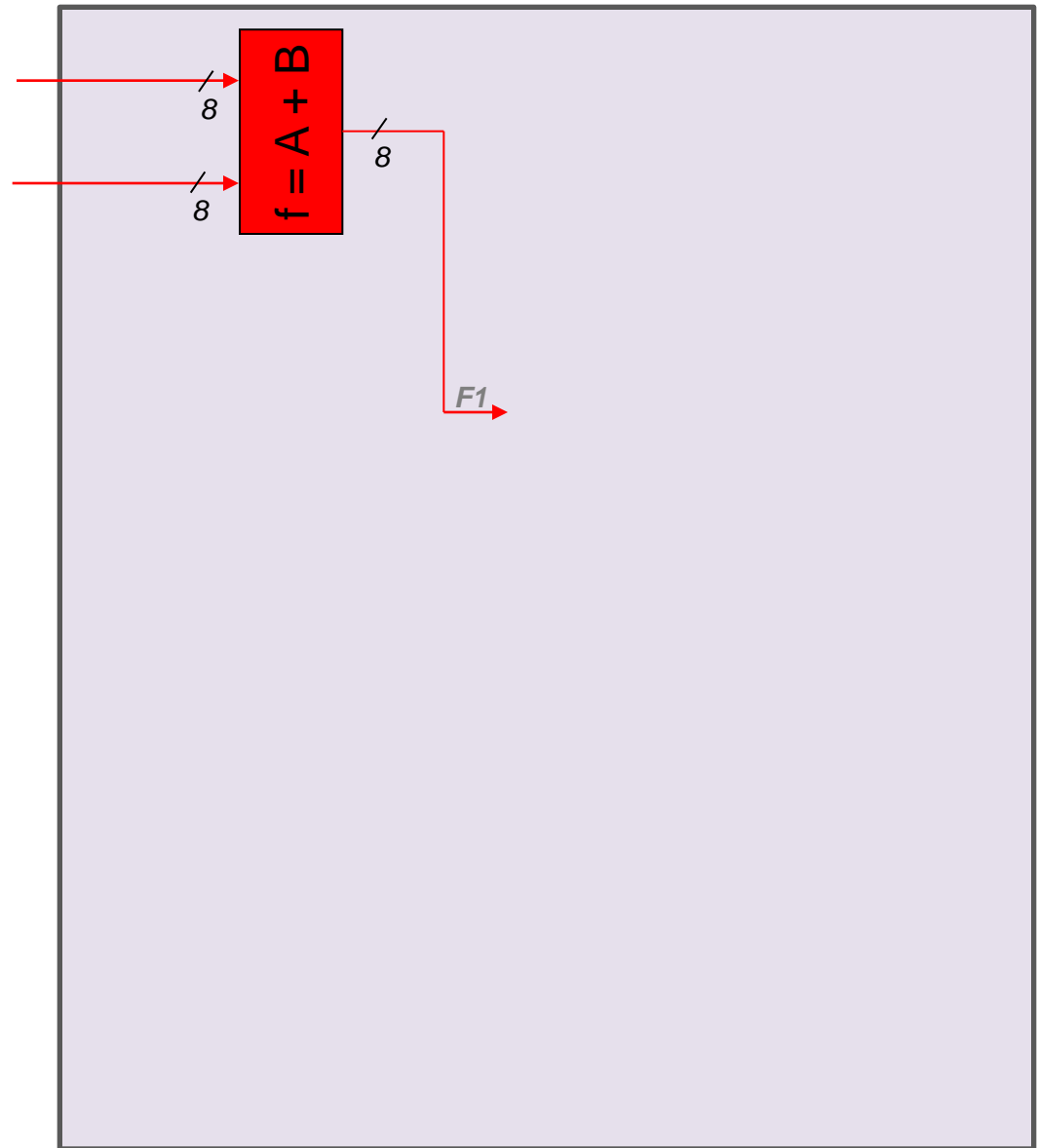
top_calc

Animação do **PORT MAP** – C1

begin

L1: C1 port map (**SW**(7 downto 0),
SW(15 downto 8), **F1**);

end topo_stru; -- **END da architecture**



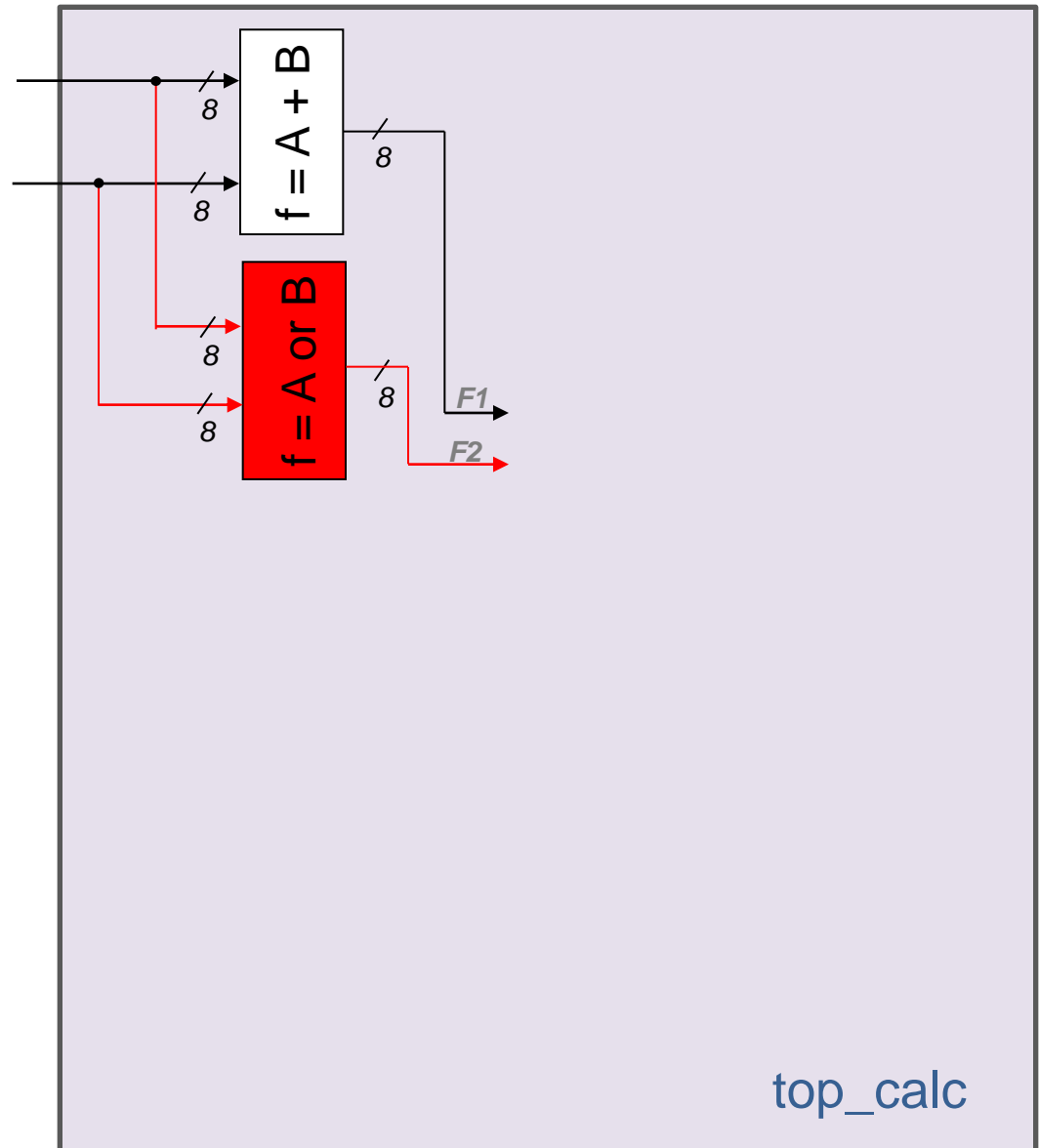
Animação do **PORT MAP** – C2

begin

L1: C1 port map (**SW**(7 downto 0),
SW(15 downto 8), **F1**);

L2: C2 port map (**SW**(7 downto 0),
SW(15 downto 8), **F2**);

end topo_stru; -- **END da architecture**



Animação do **PORT MAP** – C3

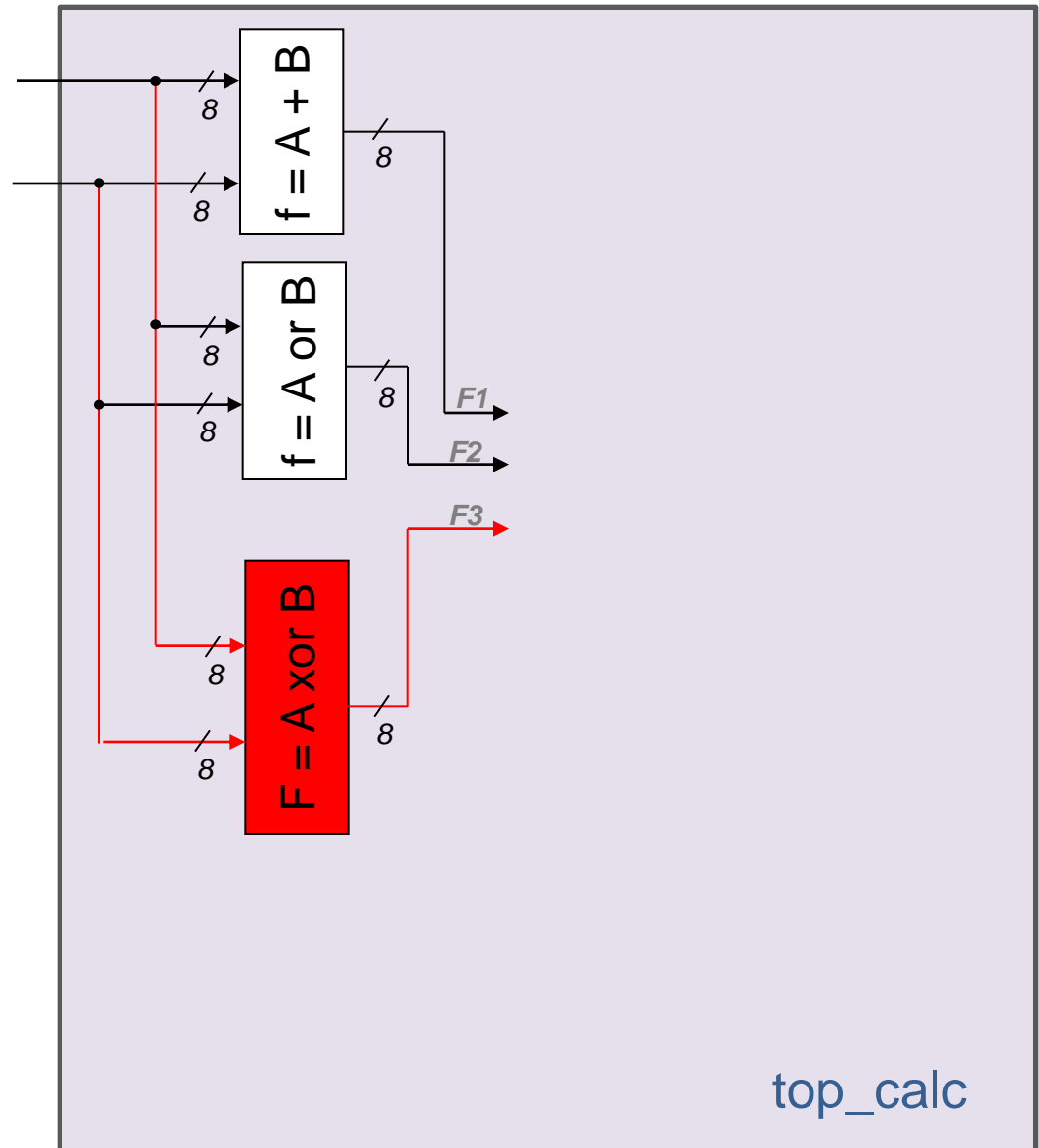
begin

L1: C1 port map (**SW**(7 downto 0),
SW(15 downto 8), **F1**);

L2: C2 port map (**SW**(7 downto 0),
SW(15 downto 8), **F2**);

L3: C3 port map (**SW**(7 downto 0),
SW(15 downto 8), **F3**);

end topo_stru; -- **END da architecture**



Animação do **PORT MAP** – C4

begin

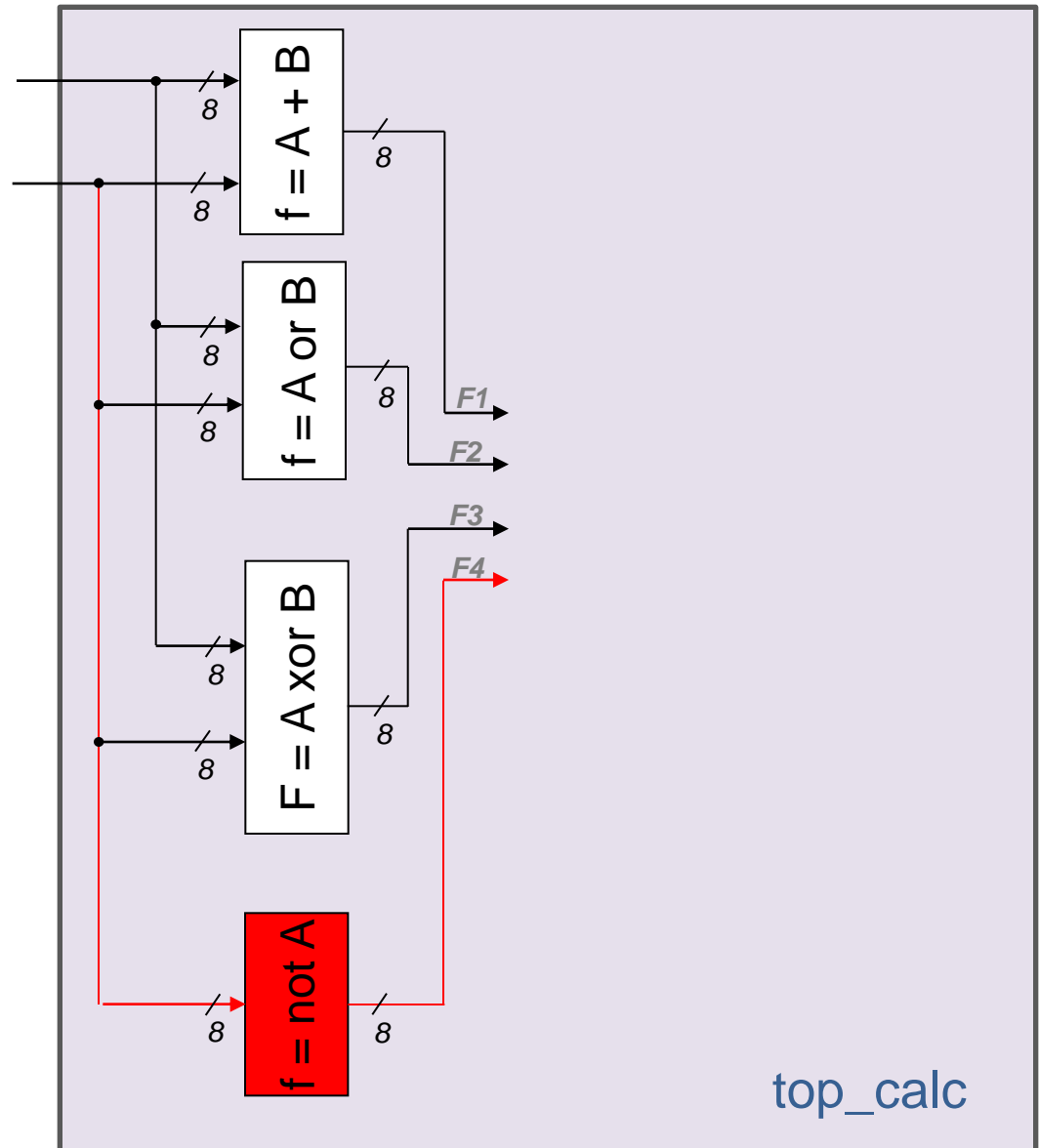
L1: C1 port map (**SW**(7 downto 0),
SW(15 downto 8), **F1**);

L2: C2 port map (**SW**(7 downto 0),
SW(15 downto 8), **F2**);

L3: C3 port map (**SW**(7 downto 0),
SW(15 downto 8), **F3**);

L4: C4 port map (**SW**(7 downto 0), **F4**);

end topo_stru; -- END da architecture



Animação do **PORT MAP** – mux4x1

begin

L1: C1 port map (**SW**(7 downto 0),
SW(15 downto 8), **F1**);

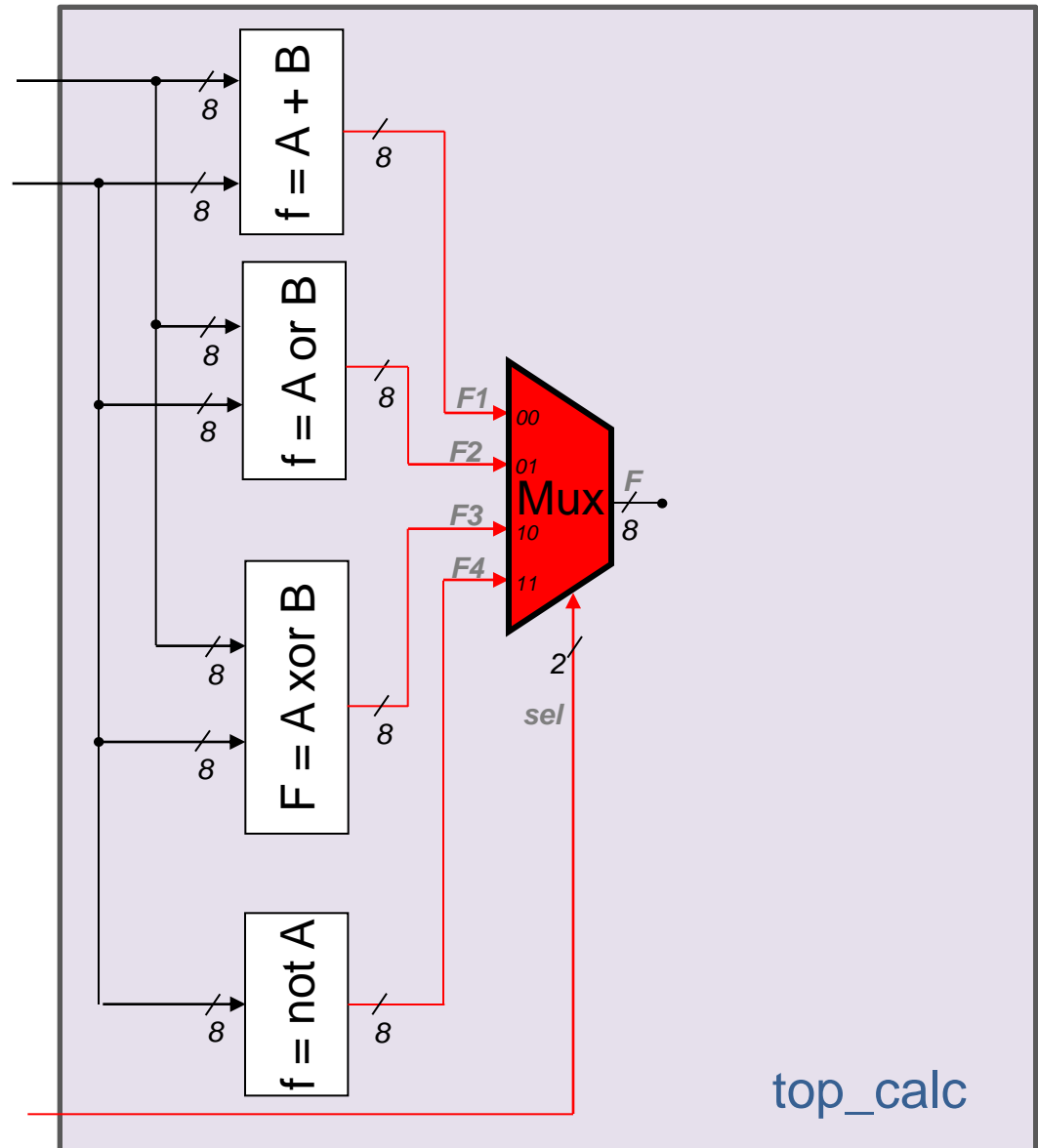
L2: C2 port map (**SW**(7 downto 0),
SW(15 downto 8), **F2**);

L3: C3 port map (**SW**(7 downto 0),
SW(15 downto 8), **F3**);

L4: C4 port map (**SW**(7 downto 0), **F4**);

L5: mux4x1 port map (**F1**, **F2**, **F3**, **F4**,
SW(17 downto 16), **F**);

end topo_stru; -- END da architecture



Animação do **PORT MAP** – Decod7seg

begin

L1: C1 port map (**SW**(7 downto 0),
SW(15 downto 8), **F1**);

L2: C2 port map (**SW**(7 downto 0),
SW(15 downto 8), **F2**);

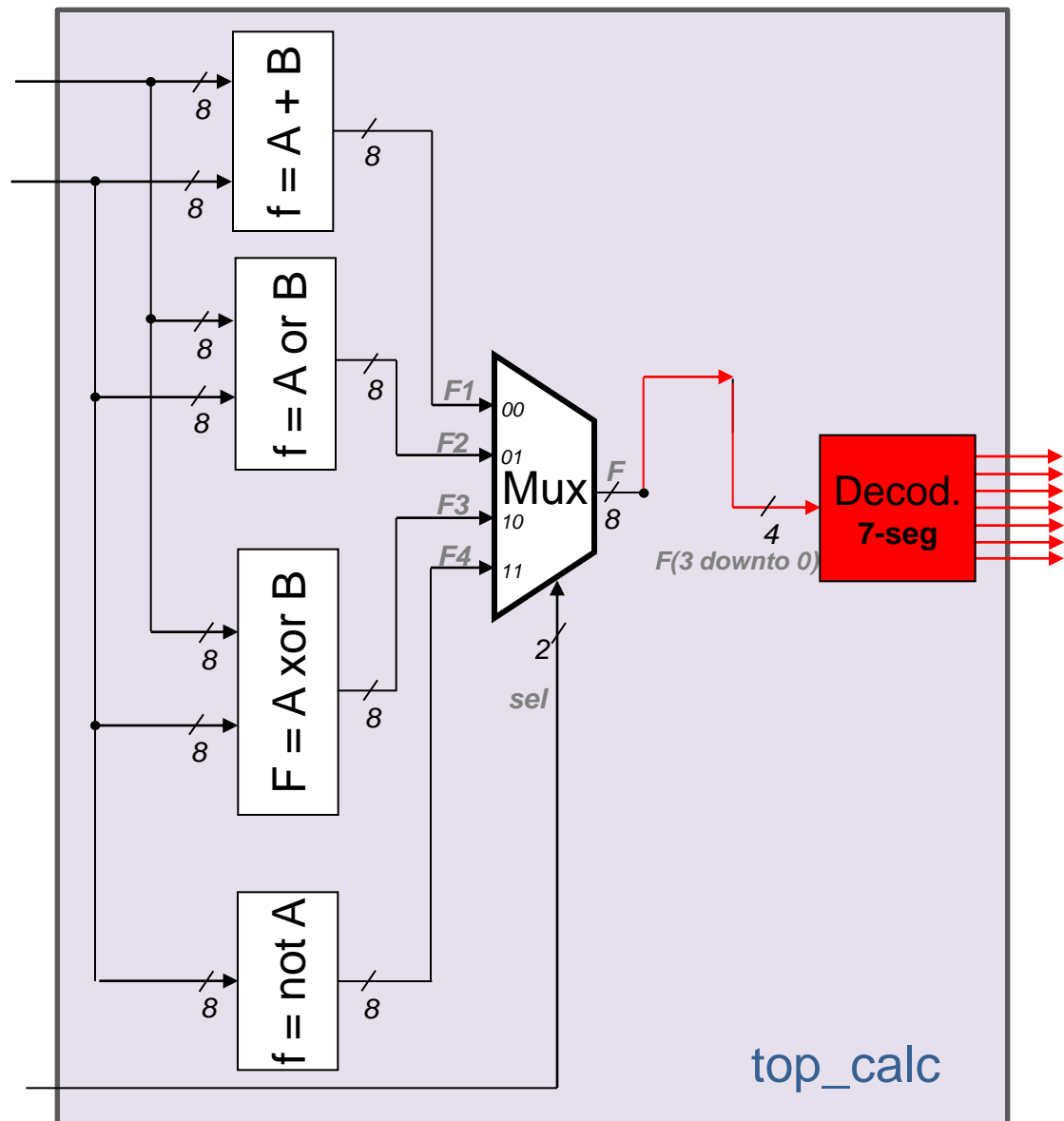
L3: C3 port map (**SW**(7 downto 0),
SW(15 downto 8), **F3**);

L4: C4 port map (**SW**(7 downto 0), **F4**);

L5: mux4x1 port map (**F1**, **F2**, **F3**, **F4**,
SW(17 downto 16), **F**);

L6: Decod7seg port map (**F**(3 downto 0),
HEX0);

end topo_stru; -- END da architecture



Animação do **PORT MAP** – Decod7seg

begin

L1: C1 port map (**SW**(7 downto 0),
SW(15 downto 8), **F1**);

L2: C2 port map (**SW**(7 downto 0),
SW(15 downto 8), **F2**);

L3: C3 port map (**SW**(7 downto 0),
SW(15 downto 8), **F3**);

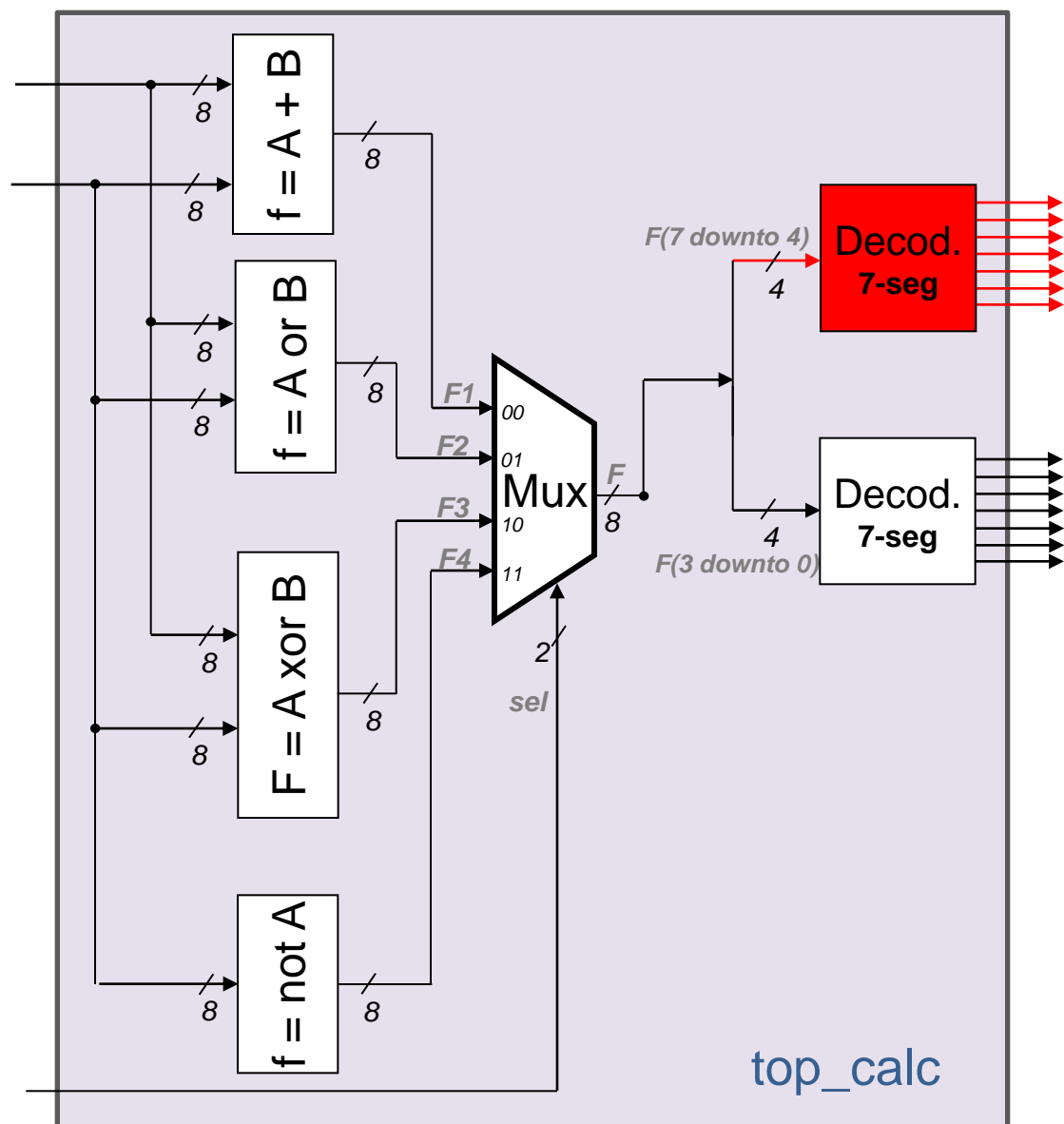
L4: C4 port map (**SW**(7 downto 0), **F4**);

L5: mux4x1 port map (**F1**, **F2**, **F3**, **F4**,
SW(17 downto 16), **F**);

L6: Decod7seg port map (**F**(3 downto 0),
HEX0);

L7: Decod7seg port map (**F**(7 downto 4),
HEX1);

end topo_stru; -- END da architecture



Animação do **PORT MAP**

begin

L1: C1 port map (**SW**(7 downto 0),
SW(15 downto 8), **F1**);

L2: C2 port map (**SW**(7 downto 0),
SW(15 downto 8), **F2**);

L3: C3 port map (**SW**(7 downto 0),
SW(15 downto 8), **F3**);

L4: C4 port map (**SW**(7 downto 0), **F4**);

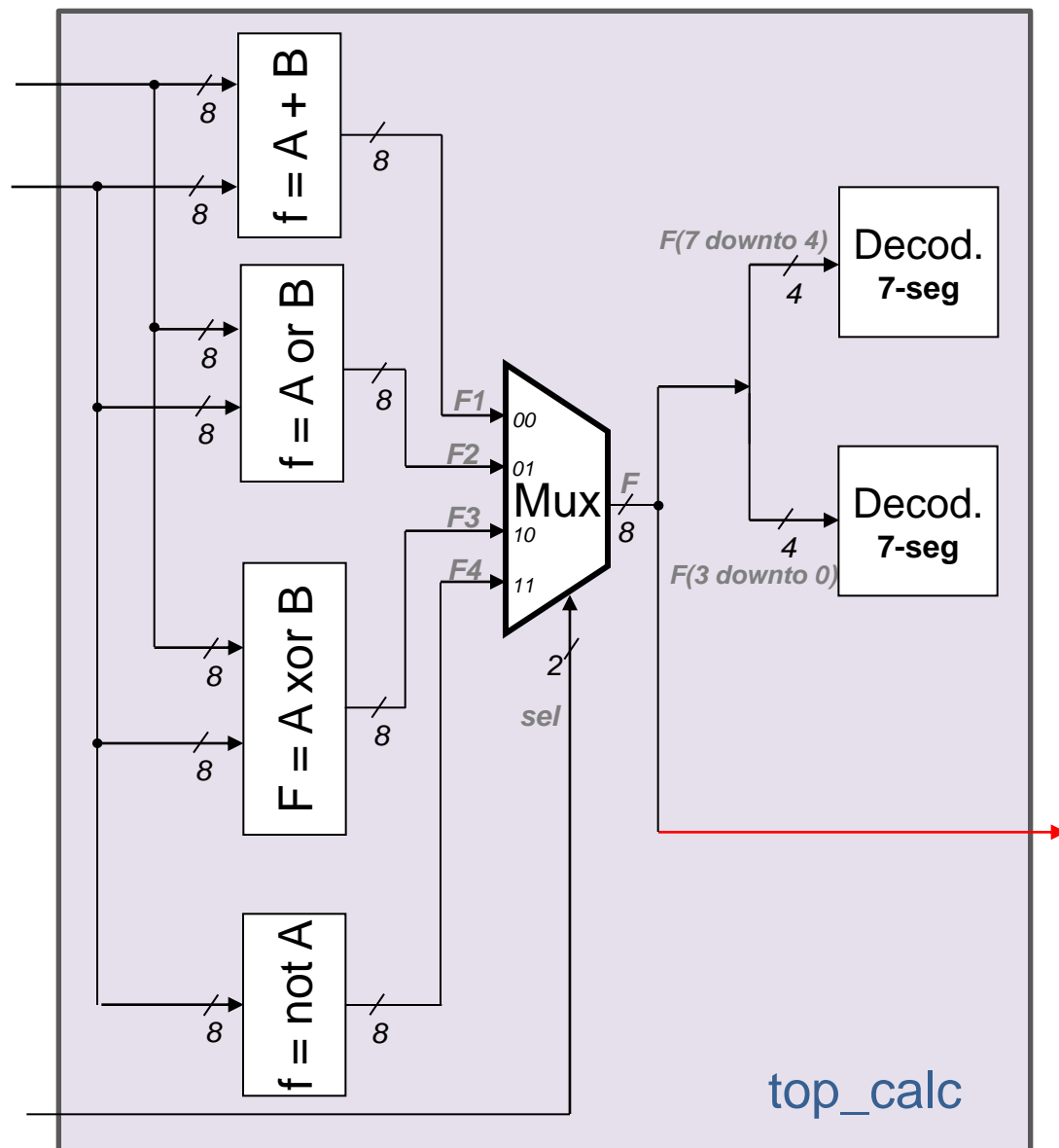
L5: mux4x1 port map (**F1**, **F2**, **F3**, **F4**,
SW(17 downto 16), **F**);

L6: Decod7seg port map (**F**(3 downto 0),
HEX0);

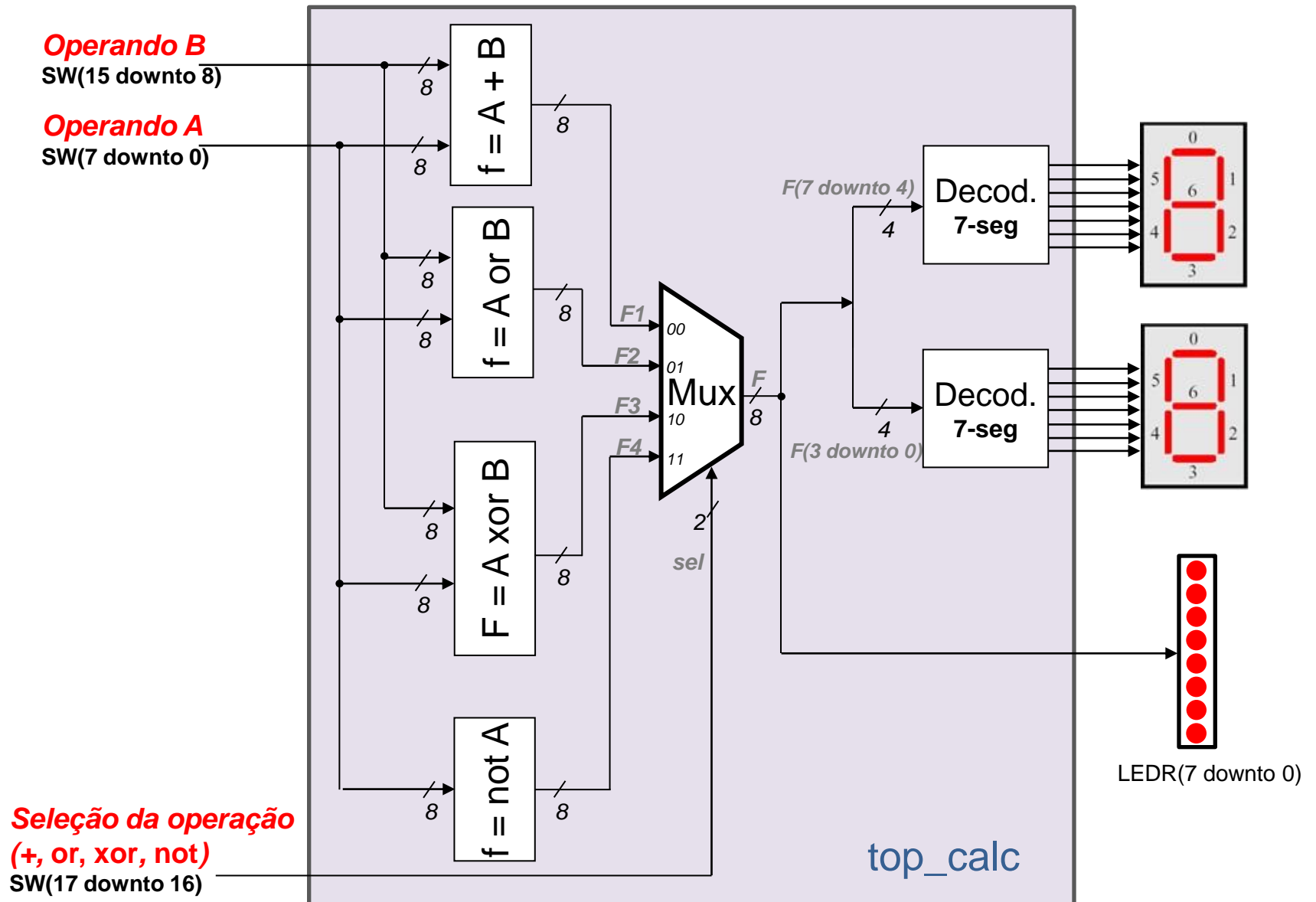
L7: Decod7seg port map (**F**(7 downto 4),
HEX1);

LEDR(7 downto 0) <= **F**;

end topo_stru; -- **END da architecture**



Mini-calculadora de 8 bits



Simulação

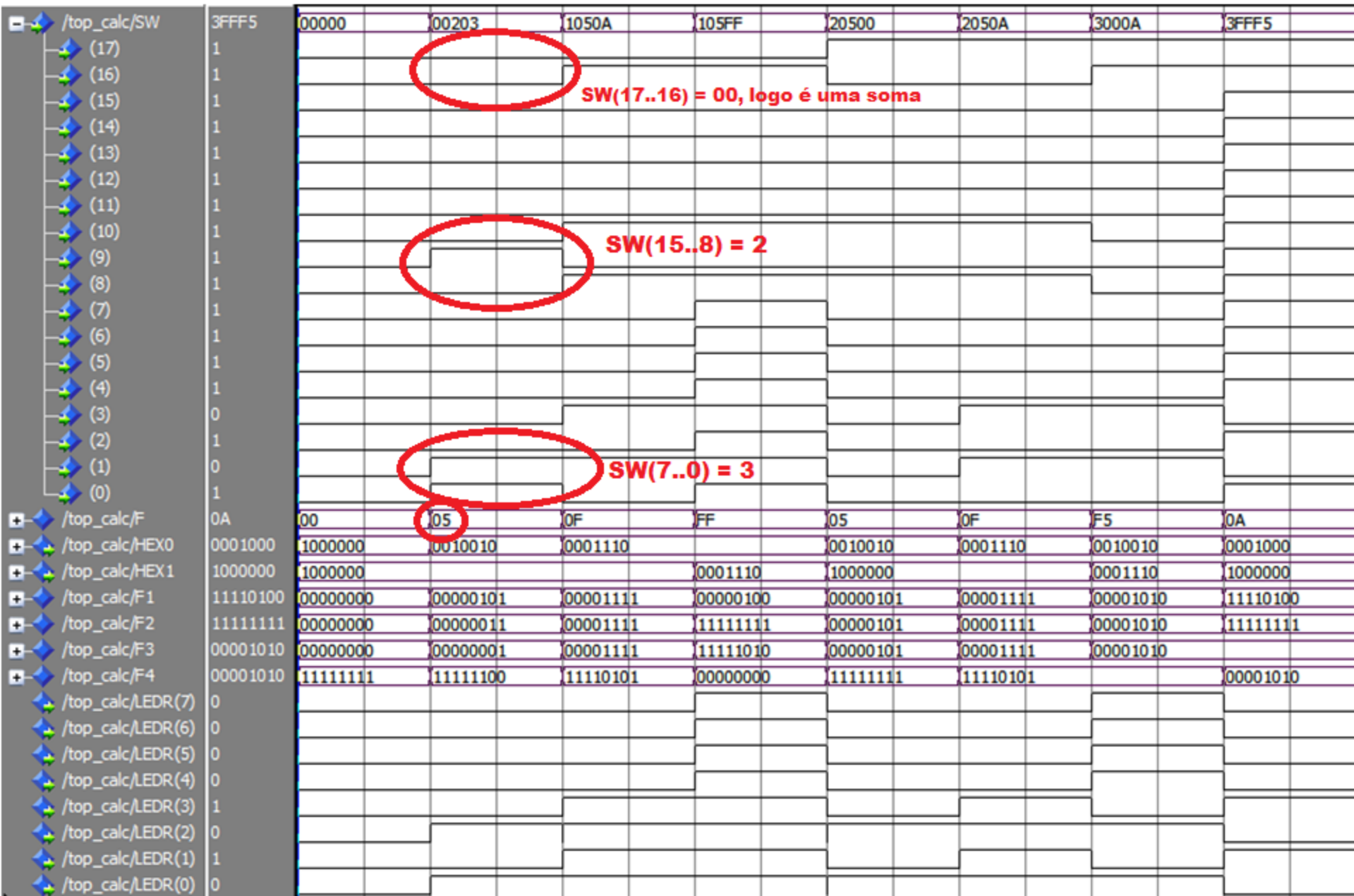


Tabela verdade

Entradas			Saídas			
SW _{15..0} B A		SW _{17..16} Seleção	<i>F1 = A + B</i> <i>F2 = A or B</i> <i>F3 = A xor B</i> <i>F4 = not A</i>	Simulação HEX1 HEX0	Simulação LEDR _{6..0}	FPGA HEX1 HEX0 LEDR _{6..0}
00	00	00	F1 =			
02	03	00	F1 =			
05	0A	01	F2 =			
05	FF	01	F2 =			
05	00	10	F3 =			
05	0A	10	F3 =			
00	0A	11	F4 =			
FF	F5	11	F4 =			