



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



CTC UFSC

“EEL5105 - Circuitos e Técnicas Digitais”

Prof. Héctor Pettenghi Roldán*

Hector@eel.ufsc.br

Florianópolis, março de 2017.

***Baseados nos slides do Professor Eduardo Bezerra EEL5105 2015.2**

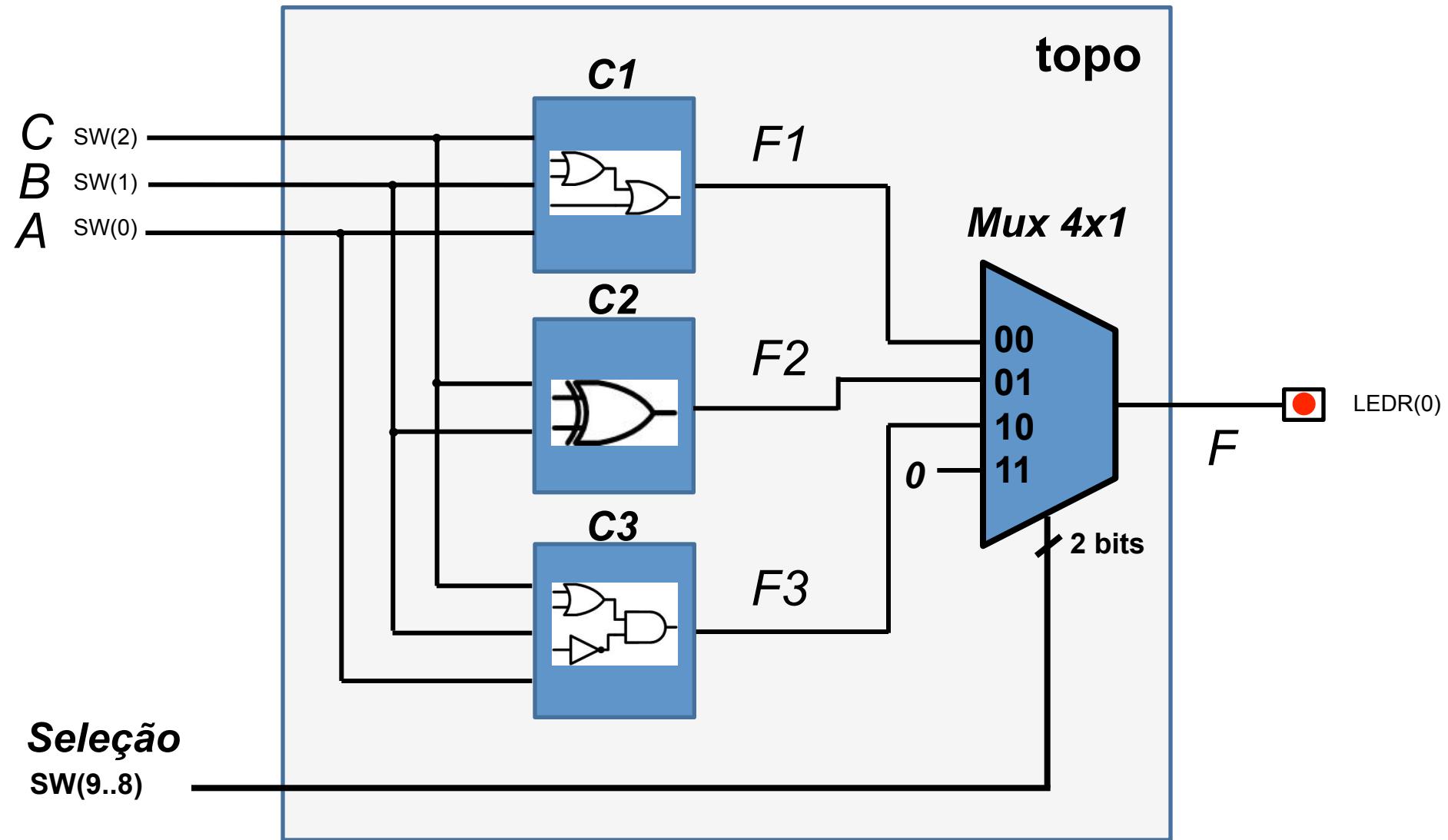
Circuitos codificadores e decodificadores

Objetivos do laboratório

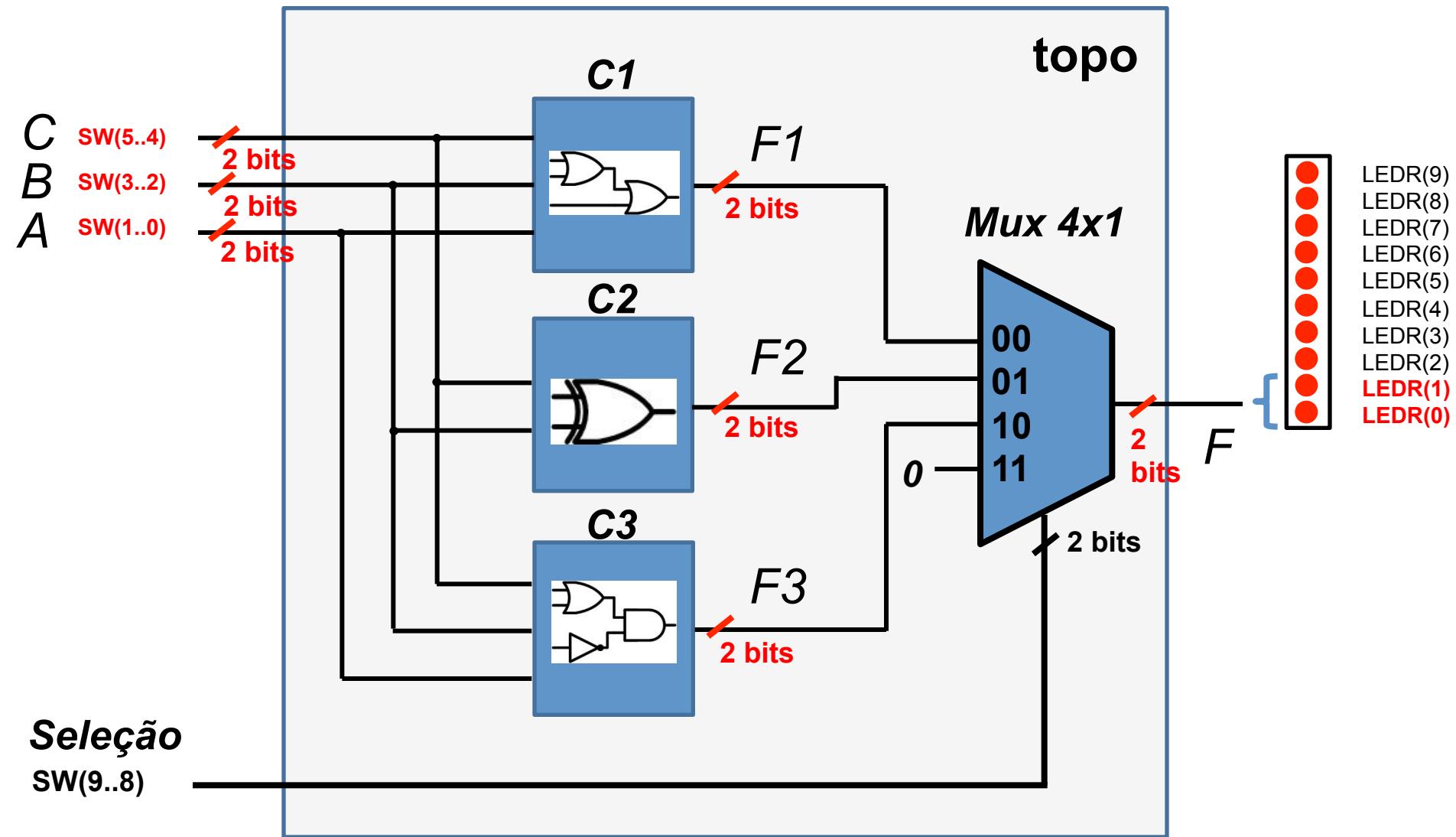
1. Uso de barramentos (vetores de sinais) em VHDL – *std_logic_vector* (Como foi visto para o controle do Multiplexador na aula anterior).
2. Entender o conceito de circuitos codificadores e decodificadores (conversores de código). Implementação de codificadores e decodificadores em VHDL.
3. Estudo de caso: projeto de calculadora personalizada, com apresentação dos resultados em displays de 7-segmentos.

1 . 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 2 bits:



Utilizando operandos de 2 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 **2 bits** - ex. $A = 01, B = 10, C = 11$

The diagram illustrates the addition of binary numbers. On the left, under the heading "1 bit", two binary digits are added: 0 and 1. The result of the addition is 1, with a carry of 0. On the right, under the heading "2 bits", three pairs of binary digits are added: (01, 10, 10), (01, 10, 11), and (01, 11, 11). The results of the additions are 11, 01, and 11 respectively, with carries of 0, 1, and 1.

1 bit		2 bits
0		01
1		10
$\frac{0}{F1 = 1}$		10
0		11
$\frac{1}{F2 = 1}$		11
01		11
10		01
$\frac{11}{F1 = 11}$		11
11		$\frac{01}{F2 = 01}$

Implementação de barramento em VHDL

std_logic → fio

std_logic_vector → vários fios (barramento)

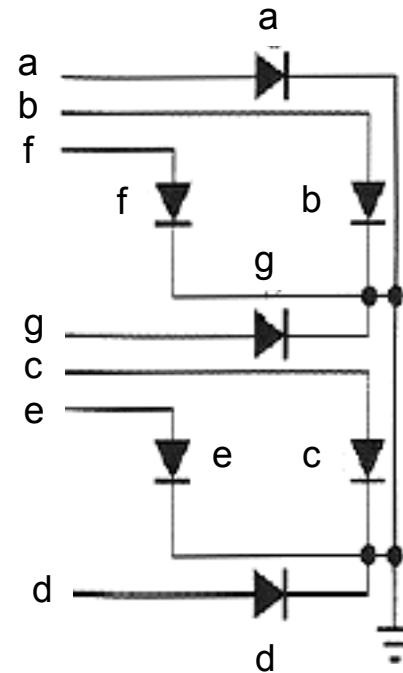
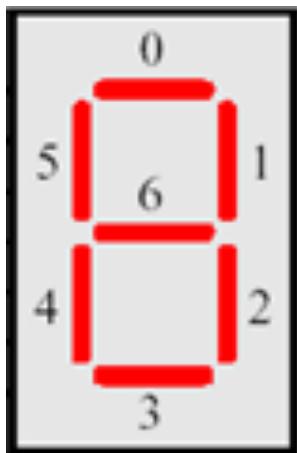
- *std_logic_vector* é utilizado para implementar um “vetor” de sinais (vários fios, barramento).
- Por exemplo, a saída F deverá ser definida na *entity* como:

F: out **std_logic_vector(1 downto 0);**

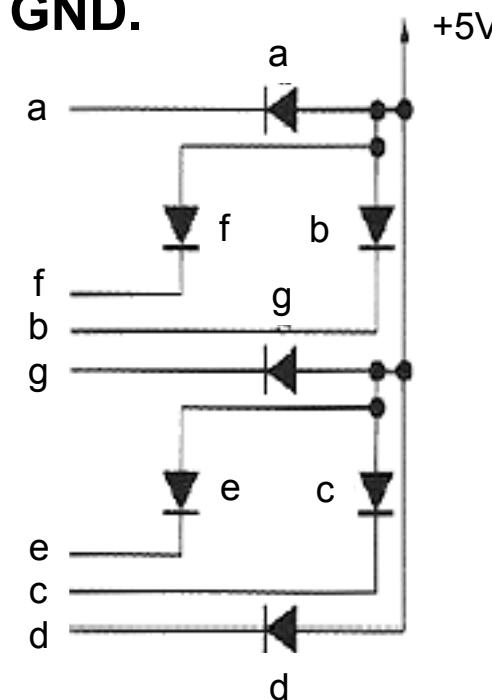
2. Decodificadores em VHDL

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

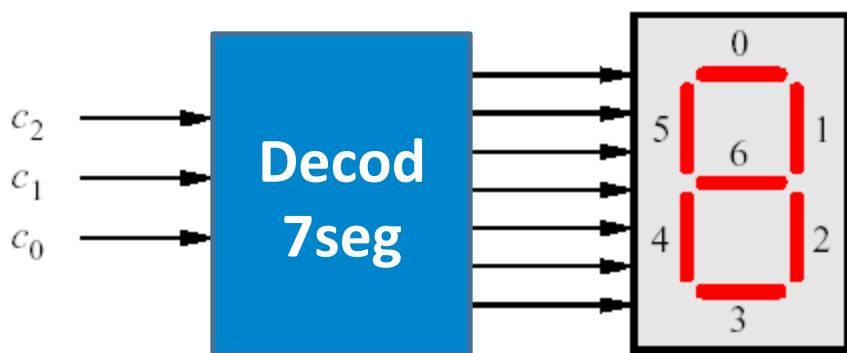
Decodificador de binário para 7-segmentos

- O kit DE1-SoC possui 6 displays de 7-segmentos, todos do tipo anodo comum (LEDs acendem com 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 “Decod7seg” 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

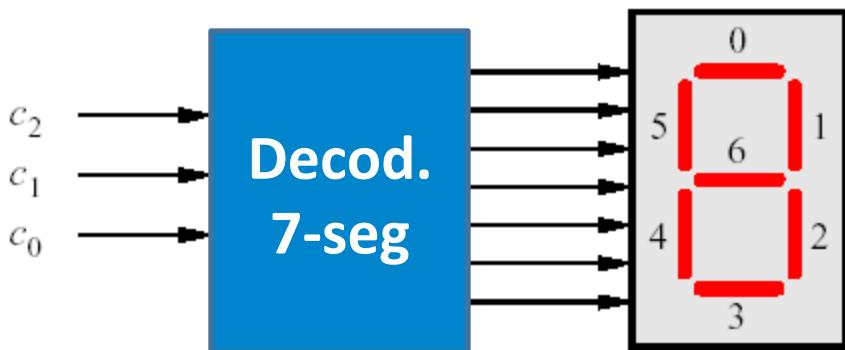
- Exemplos de valores em binários convertidos (decodificados) para 7-segmentos, visando escrita no display da placa DE2 anodo comum)



7 bits			C2	C1	C0	Letra	
6	5	4	0	0	0	1000001	U
1	6	2	0	0	1	0001110	F
3			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 7seg.
- 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			Letra
C2	C1	C0	6543210
0	0	0	1000001
0	0	1	0001110
0	1	0	0010010
0	1	1	1000110
1	1	1	1111111

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

- Soma de produtos:**

$$F(0) = C_2' C_1' C_0' + C_2 C_1 C_0$$

$$F(1) = C_2' C_1' C_0 + C_2' C_1 C_0' + C_2' C_1 C_0 + C_2 C_1 C_0$$

$$F(2) = \dots$$

- Análise comportamental:**

$F = "1000001"$ quando $C_2 C_1 C_0 = "000"$ senão

$"0001110"$ quando $C_2 C_1 C_0 = "001"$ senão

... senão

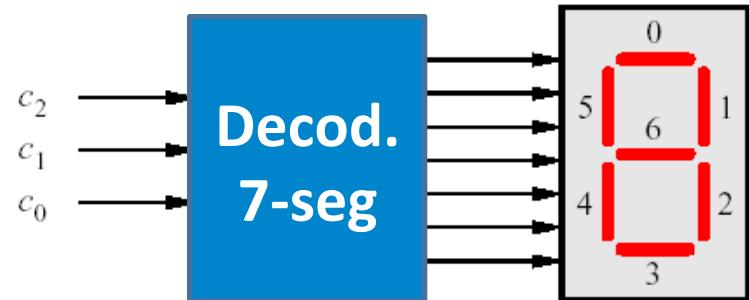
$"1111111"$

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;
```

```
architecture decod 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;
```

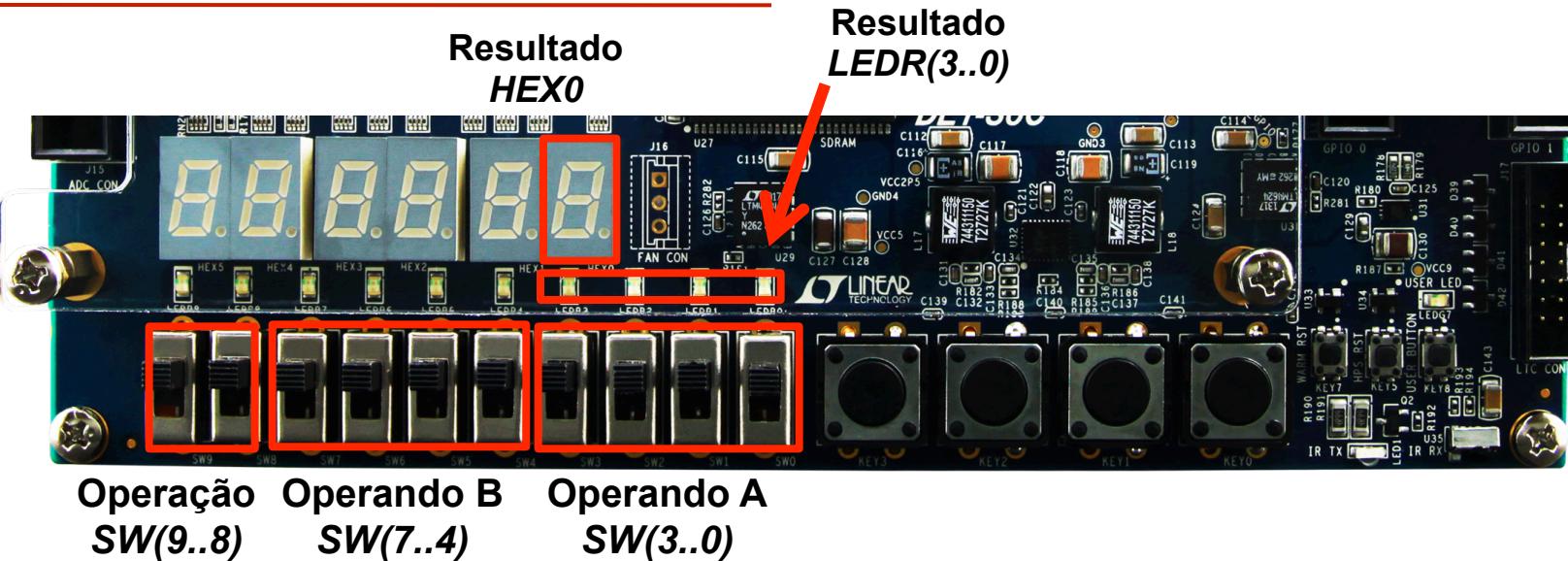


3. Implementação de mini-calculadora de 4 bits personalizada

Descrição funcional da mini-calculadora de 4 bits

- A mini-calculadora realiza uma operação aritmética e três operações lógicas, todas de **4 bits**:
 - $F_1 = A + B$ -- operação aritmética ADIÇÃO
 - $F_2 = A \text{ or } B$
 - $F_3 = A \text{ xor } B$
 - $F_4 = \text{not } A$

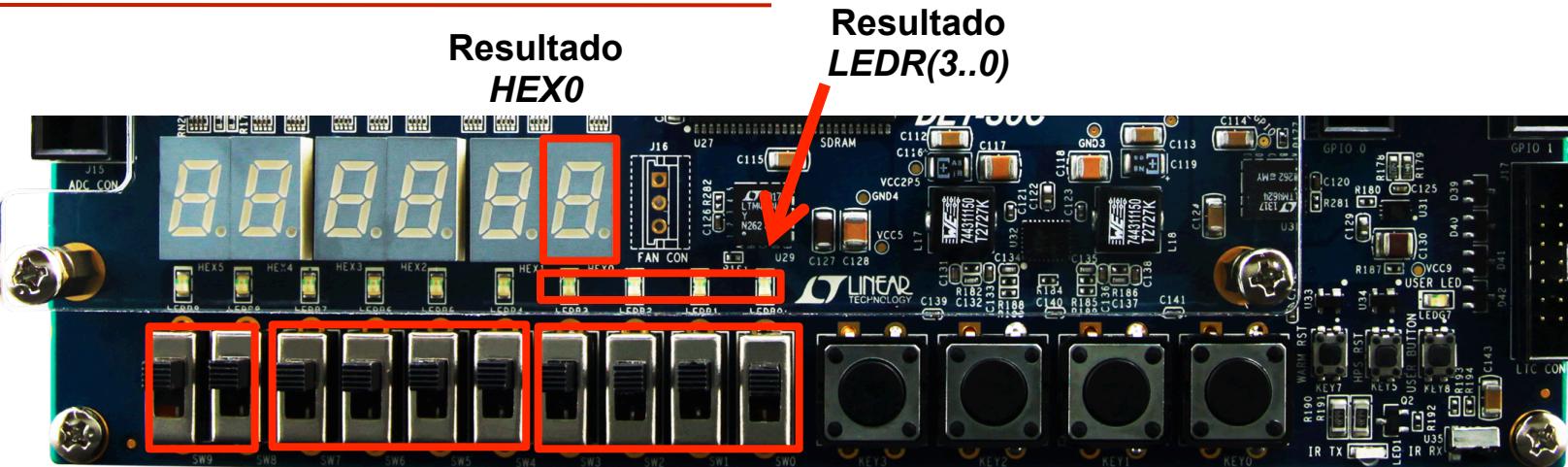
Interface com o usuário



Operação Operando B Operando A
SW(9..8) **SW(7..4)** **SW(3..0)**

- Para realizar uma das quatro operações disponíveis (**F1, F2, F3, F4**), a calculadora personalizada utiliza:
 - as chaves **SW(3..0)** para leitura do **operando A**
 - as chaves **SW(7..4)** para leitura do **operando B**
 - as chaves **SW(9..8)** para seleção da operação desejada
- Os resultados são apresentados em displays de 7-segmentos e nos LEDs vermelhos.

Interface com o usuário



Operação Operando B Operando A
SW(9..8) SW(7..4) SW(3..0)

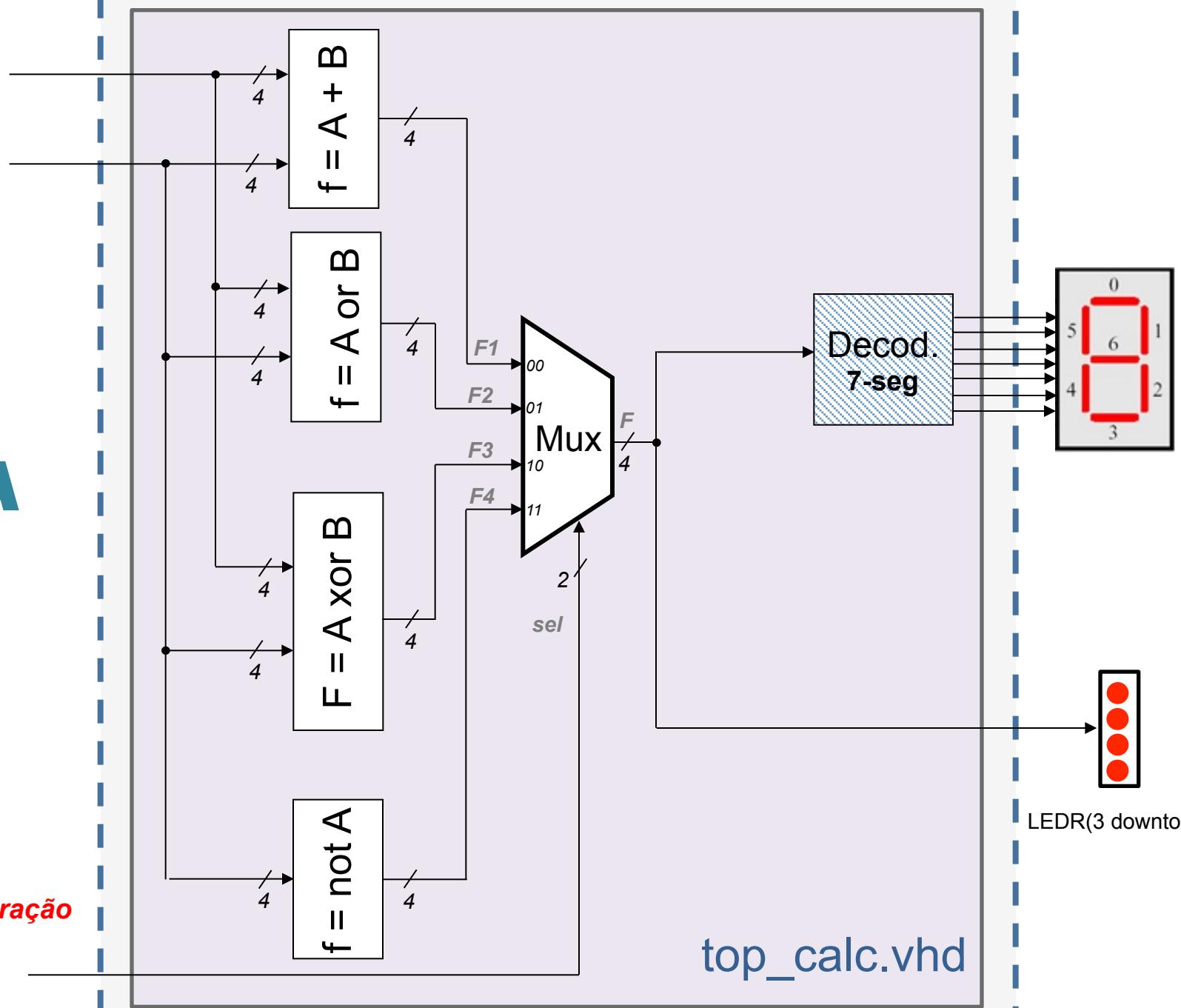
Seletor SW(9..8)	Saída (LEDR e display 7-seg)
00	A + B
01	A or B
10	A xor B
11	not A

FPGA

Seleção da operação
(+, or, xor, not)
SW(9 downto 8)

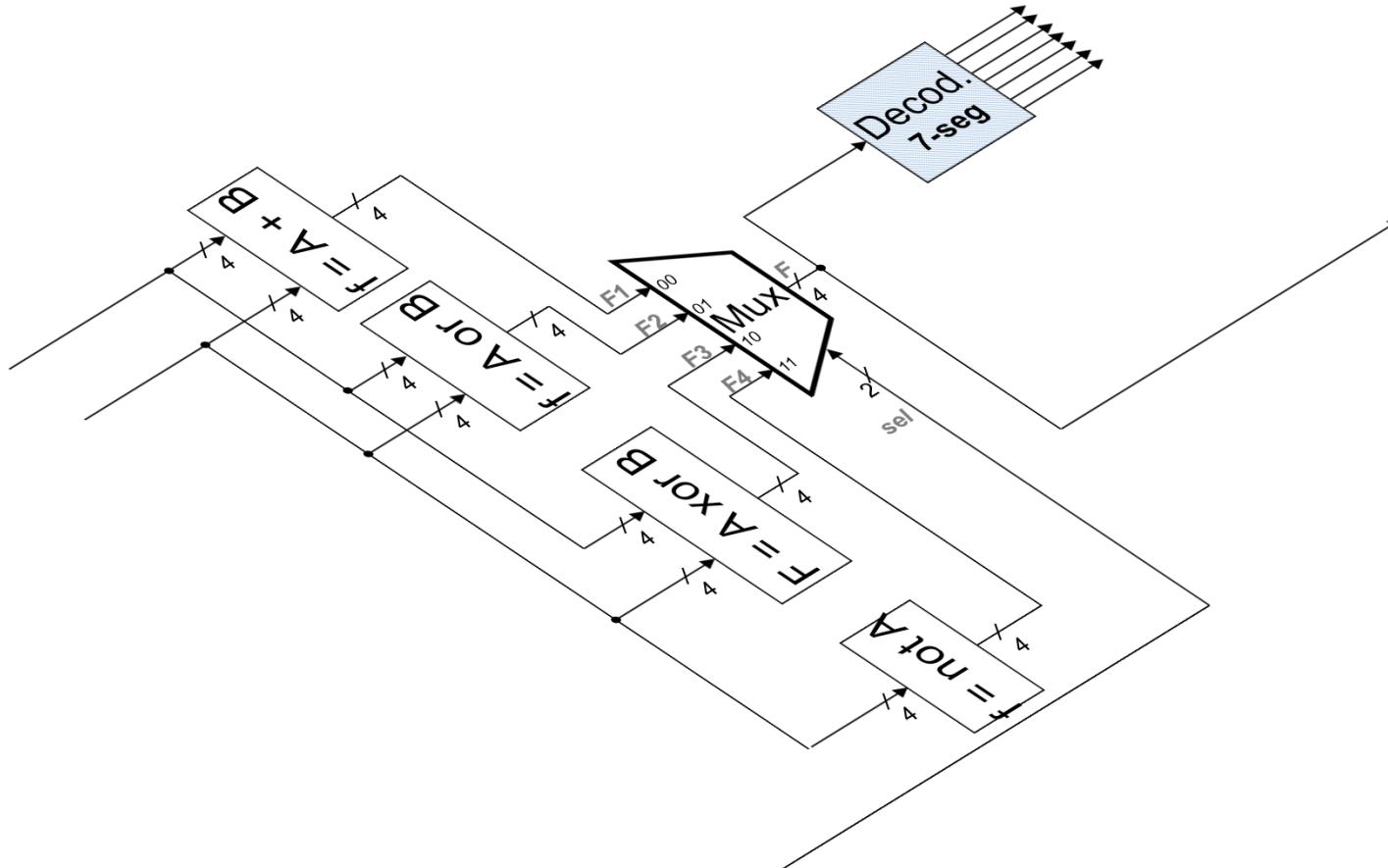
Operando B
SW(7 downto 4)

Operando A
SW(3 downto 0)



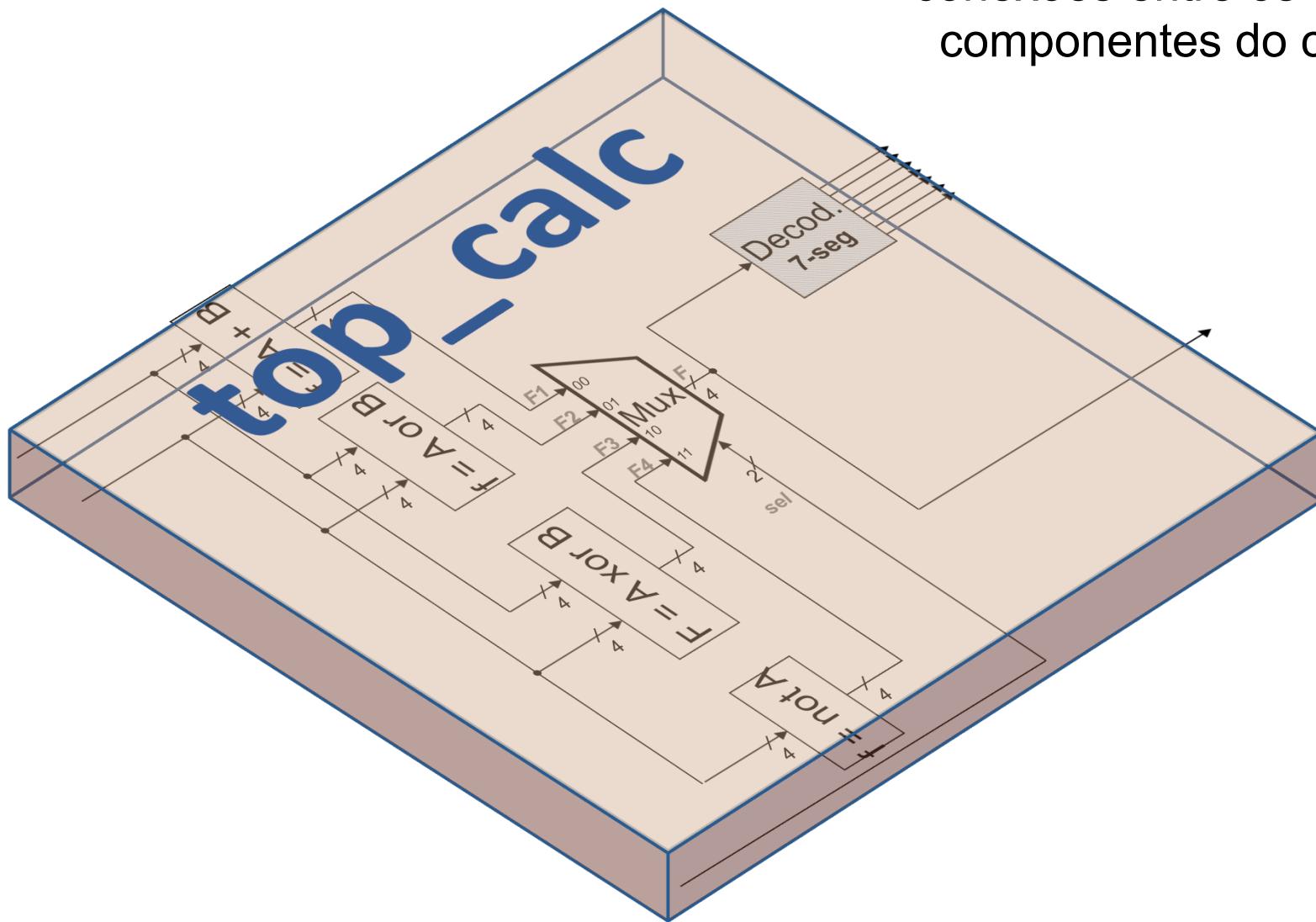
Projeto hierárquico

Calculadora é composta por 7 componentes: quatro “funções lógicas e aritméticas”, “mux”, “decodificador”, e “top_calc”



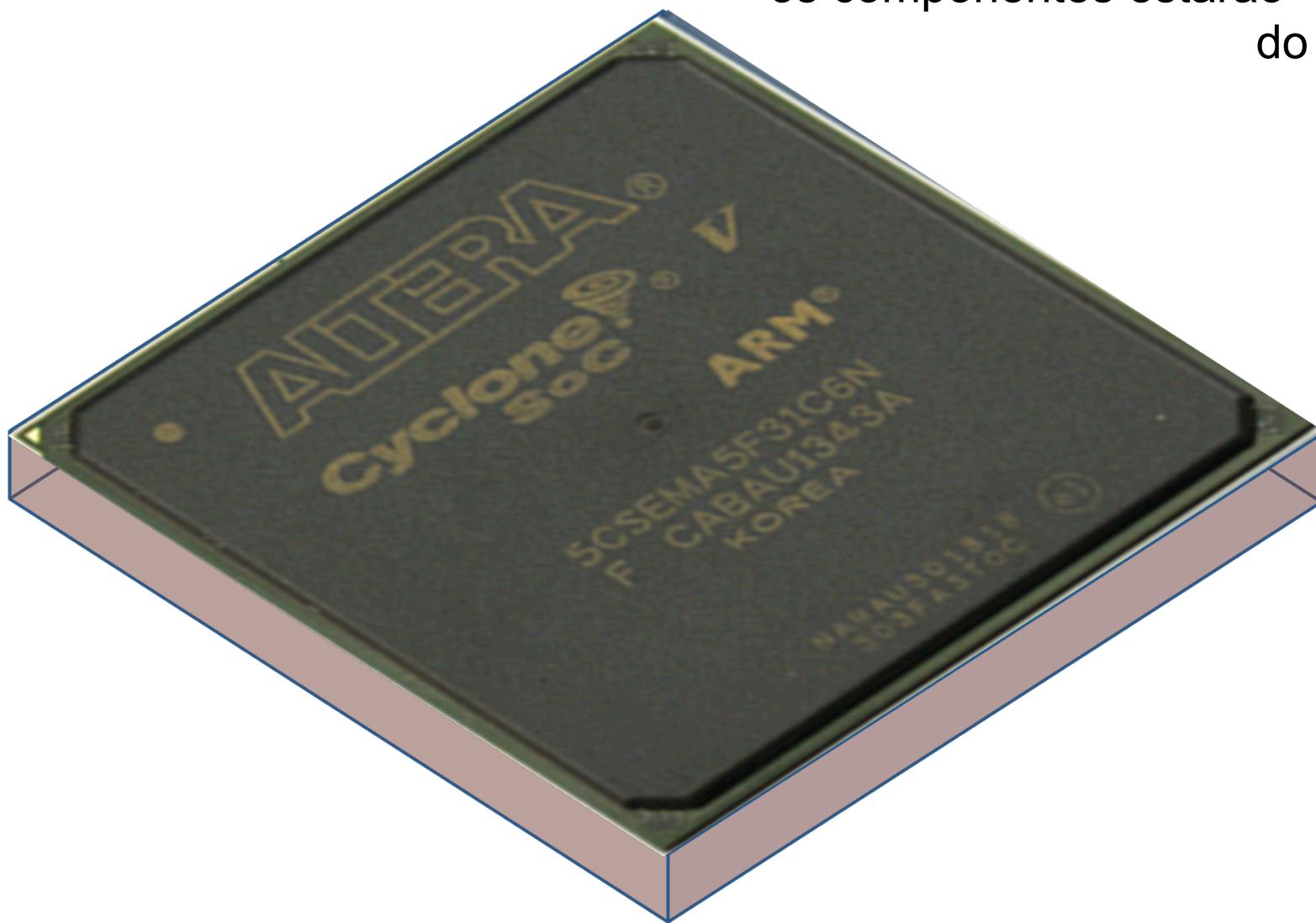
Projeto hierárquico

O componente “**top_calc**” é responsável por realizar as conexões entre os demais componentes do circuito.



Projeto hierárquico

Após o download para a placa do circuito gerado na síntese, todos os componentes estarão “dentro” do FPGA.

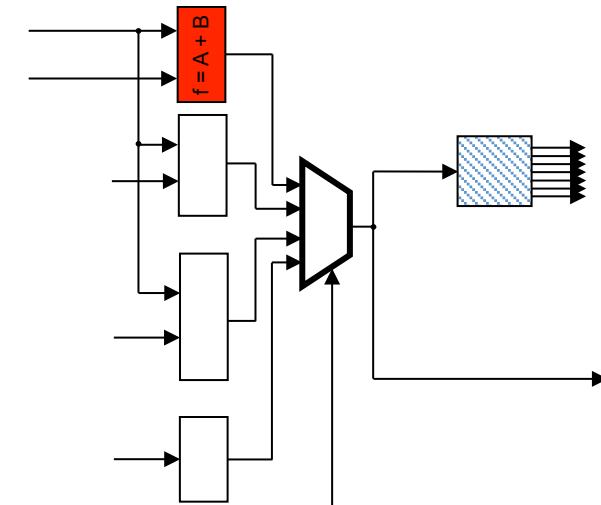


Componente C1 (arquivo *c1.vhd*)

```
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(3 downto 0);
      B: in std_logic_vector(3 downto 0);
      F: out std_logic_vector(3 downto 0)
    );
end C1;
```

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

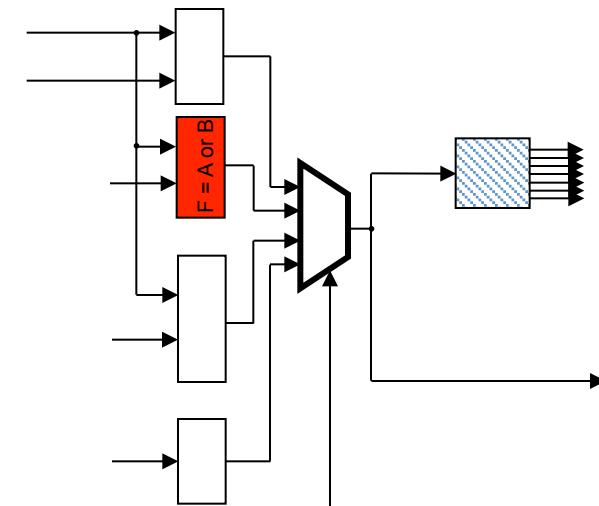


Componente C2 (arquivo c2.vhd)

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

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

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

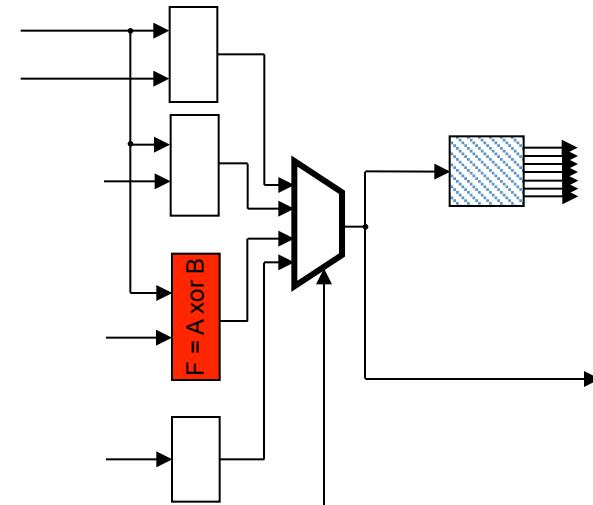


Componente C3 (arquivo c3.vhd)

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

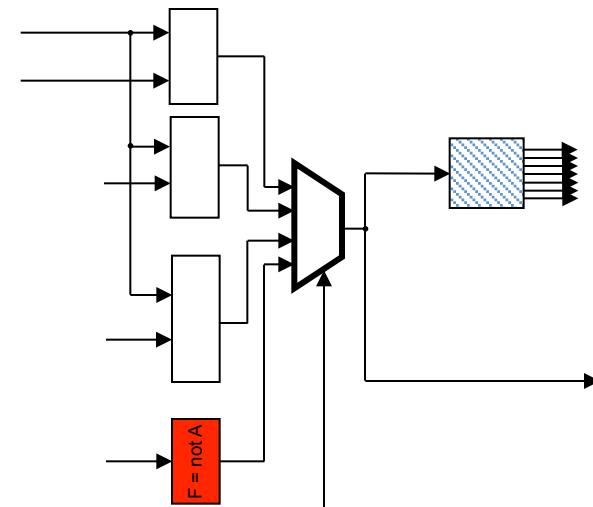
entity C3 is
port (A: in std_logic_vector(3 downto 0);
      B: in std_logic_vector(3 downto 0);
      F: out std_logic_vector(3 downto 0)
    );
end C3;

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(3 downto 0);  
      F: out std_logic_vector(3 downto 0)  
      );  
end C4;  
  
architecture circuito of C4 is  
begin  
  F <= not A;  
end circuito;
```

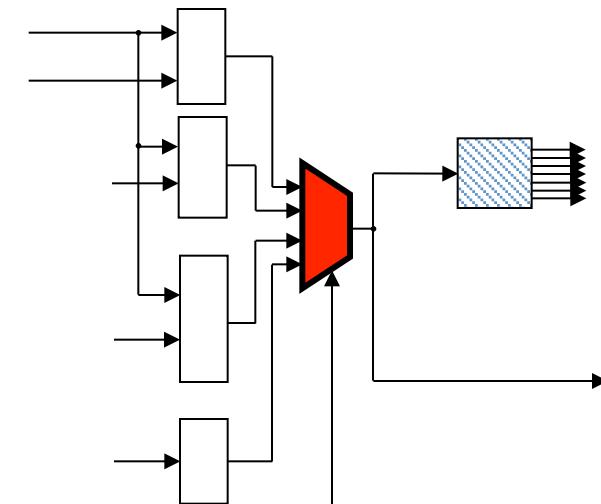


Componente Mux (arquivo *mux4x1.vhd*)

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

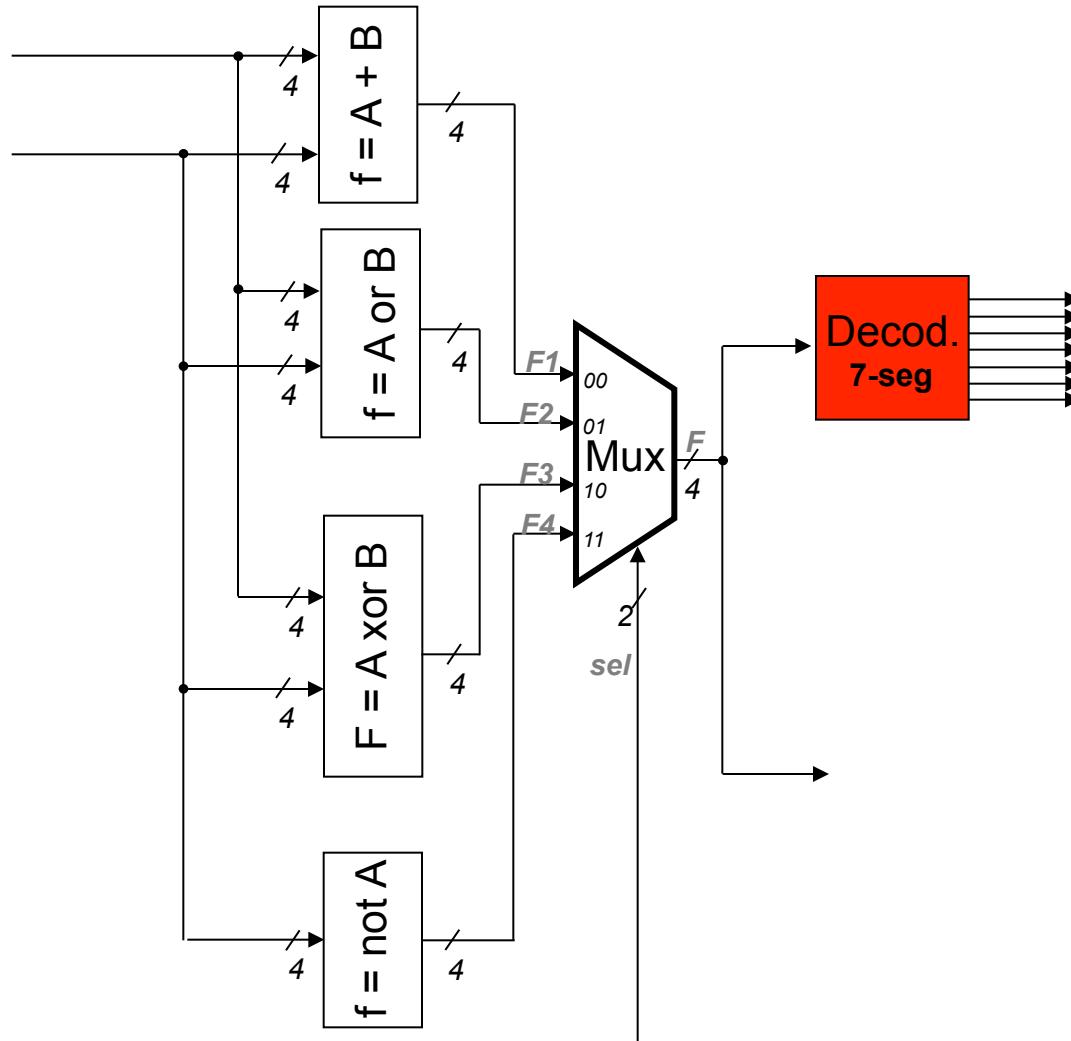
entity mux4x1 is
port (w, x, y, z: in std_logic_vector(3 downto 0);
      s: in std_logic_vector(1 downto 0);
      m: out std_logic_vector(3 downto 0)
);
end mux4x1;

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 laboratório.



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 (9 downto 0);
           HEX0: out std_logic_vector(6 downto 0);
           LEDR : out std_logic_vector (9 downto 0)
        );
end top_calc;
architecture topo_stru of top_calc is
    signal F, F1, F2, F3, F4: 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);
              F : out std_logic_vector(3 downto 0));
    end component;
    -- componentes C2 e C3, idem C1
    component C4
        port (A : in std_logic_vector(3 downto 0);
              F : out std_logic_vector(3 downto 0)
        );
    end component;
    component mux4x1
        port (w, x, y, 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 component;
    -- Incluir aqui o componente Decod7seg
```

```
begin
    L1: C1 port map (SW(3 downto 0),
                      SW(7 downto 4), F1);
    L2: C2 port map (SW(3 downto 0),
                      SW(7 downto 4), F2);
    L3: C3 port map (SW(3 downto 0),
                      SW(7 downto 4), F3);
    L4: C4 port map (SW(3 downto 0), F4);
    L5: mux4x1 port map (F1, F2, F3, F4,
                          SW(9 downto 8), F);
    L6: Decod7seg port map (F, HEX0);
    LEDR <= "000000" & F;
end topo_stru; -- END da architecture
```



Tarefa a ser realizada na aula prática

Resumo da tarefa a ser realizada

Tutorial com o professor:

1. Criar um novo projeto denominado **top_calc** (esse é o nome da *entity* do arquivo “topo” – slide 29), e utilizar os componentes VHDL dos slides 23 a 27 para implementação da calculadora.

Tarefa a ser realizada pelo aluno:

1. Implementar o decodificador “binário para 7-segmentos” usando como base o VHDL apresentado no slide 14. 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 33. 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 33.
2. Editar o componente **top_calc** do slide 29, e incluir a declaração do novo componente decodificador (usando **component**).
3. Verificar o funcionamento da calculadora **no simulador e preencher a tabela do slide 35.**
4. Verificar o funcionamento **no kit FPGA uma vez feita a simulação.**

Dicas úteis para a realização da tarefa

1. O projeto é composto por 7 arquivos (7 componentes):
 - *c1.vhd* – fornecido nos slides anteriores (slide 23)
 - *c2.vhd* – fornecido nos slides anteriores (slide 24)
 - *c3.vhd* – fornecido nos slides anteriores (slide 25)
 - *c4.vhd* – fornecido nos slides anteriores (slide 26)
 - *mux4x1.vhd* – fornecido nos slides anteriores (slide 27)
 - *decod7seg.vhd* – a ser desenvolvido (ver exemplo no slide 14)
 - *top_calc.vhd* – parcialmente fornecido (slide 29)
2. Para implementar o decodificador de binário para 7-segmentos, utilizar como base o exemplo do slide 14. Completar a tabela do slide 33, de forma a obter todos os códigos em hexadecimal necessários.
3. Para utilizar a mini-calculadora é necessário:
 - Fornecer o operando A nas chaves SW(3..0).
 - Fornecer o operando B nas chaves SW(7..4).
 - Selecionar a operação desejada nas chaves SW(9..8).
 - O resultado será apresentado em **HEXADECIMAL** nos displays de sete segmentos (HEX0 e HEX1), e em **BINÁRIO** nos LEDs vermelhos (LEDR).

Tabela de decodificação binário para 7-segmentos

Entrada	Saída 6543210	Display
0000	1000000	
0001	1111001	
0010	0100100	
0011	0110000	
0100	0011001	
0101	0010010	
0110	0000010	
0111	1111000	
1000	0000000	
...	...	9, A, b, C, d → Fazer
1110	0000110	
1111	0001110	

Simulação

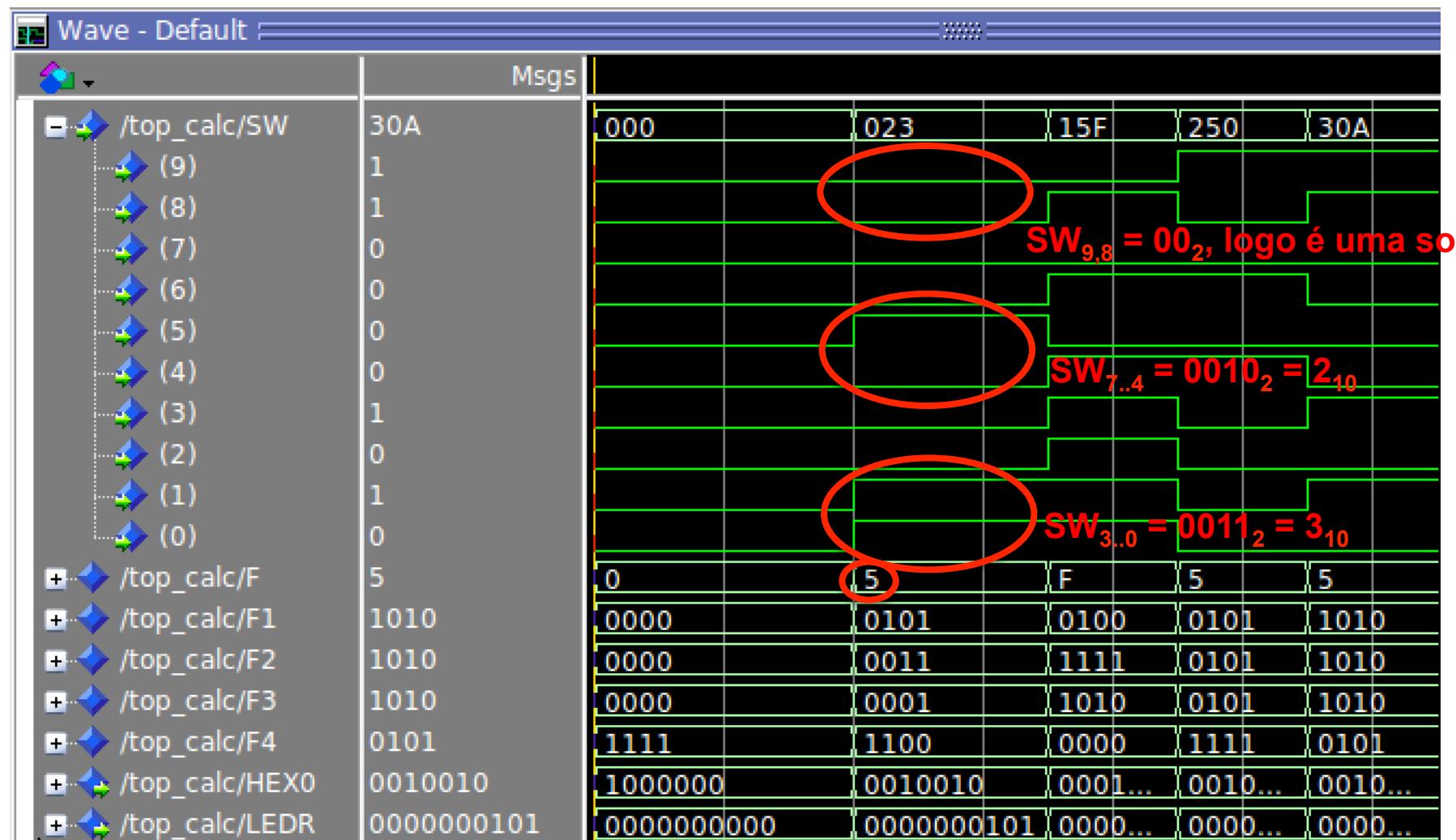


Tabela verdade

Entradas		Saídas					
SW _{7..0}	SW _{9..8}	$F1 = A + B$	$F2 = A \text{ or } B$	Simulação HEX0	Simulação LEDR _{6..0}	FPGA HEX0	FPGA LEDR _{3..0}
B	A	Seleção	$F3 = A \text{ xor } B$	$F4 = \text{not } A$			
0	0	00	F1 =				
2	3	00	F1 =				
5	A	01	F2 =				
5	F	01	F2 =				
5	0	10	F3 =				
5	A	10	F3 =				
0	A	11	F4 =				
F	5	11	F4 =				