



FEDERAL UNIVERSITY  
OF SANTA CATARINA

# Laboratório 5:

## *Circuitos sequenciais, latches e flip-flops*

---

EEL5105 – Circuitos e Técnicas Digitais

# Introdução

---

Tarefa

Erros Comuns

# Introdução

---

## Objetivos

1. Entender o conceito de *circuitos sequenciais*, via uso de *process*;
2. Explicar o uso de descrição VHDL *comportamental*;
3. Trabalhar os conceitos de *flip-flops*, *latches* e *registradores*;
4. Implementar *flip-flops* e *registradores* em VHDL utilizando *process*;
5. Realizar estudos de caso visando fixar os conceitos estudados.

# Introdução

- **Process**

- Permite modelar **lógica sequencial** em **VHDL**.
- Envolve **atribuições** baseadas em **eventos**: processo é disparado ou iniciado **quando** há mudanças de valores dos *signals* em sua **sensitivity list**.
- **Process** nunca termina (**CÍCLICO**): é disparado novamente **SEMPRE** que ocorrer alteração em algum parâmetro da **sensitivity list**.

```
library ieee;
use ieee.std_logic_1164.all;
entity Sinais is port (
    C: in std_logic;
    D: in std_logic;
    Q: out std_logic );
end Sinais;
architecture behv of Sinais is
    signal A, B: std_logic;
begin
    A <= D;
    Q <= B;
    P1: process (C,D)
    begin
        B <= '0';
        if (C = '1') then
            B <= D;
        end if;
    end process P1;
end behv;
```

# Introdução

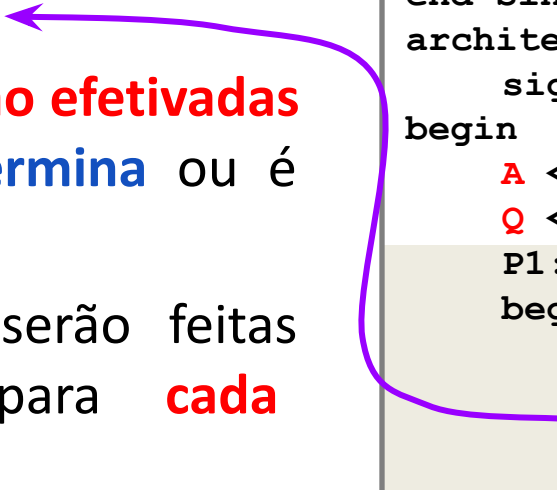
---

- **Process**

- Detalhes **MUITO** importantes:

- **Atribuições** são executadas na ordem que elas aparecem (**execução sequencial**).
- Atribuições **somente são efetivadas** quando o processo **termina** ou é **suspenso**.
- Portanto, atribuições serão feitas **apenas uma vez** para **cada execução do processo**.

```
library ieee;
use ieee.std_logic_1164.all;
entity Sinais is port (
    C: in std_logic;
    D: in std_logic;
    Q: out std_logic );
end Sinais;
architecture behv of Sinais is
    signal A, B: std_logic;
begin
    A <= D;
    Q <= B;
    P1: process (C,D)
    begin
        B <= '0';
        if (C = '1') then
            B <= D;
        end if;
    end process P1;
end behv;
```



# Introdução

---


## ● *Process*

- Ou seja, se existirem **várias atribuições** a um mesmo sinal, **APENAS a última atribuição será válida**

**Exemplo:** **B** <= **D** (no processo ao lado).

- Outros detalhes:
  - Não se pode declarar **signals** dentro de um processo.
  - Algumas estruturas só podem ser usadas dentro de processos (exemplo: **if...then...else**).

```
process (A,B,C,D)
begin
  A <= '1';
  B <= '1';
  B <= D;
  A <= not B;
  C <= A and '1';
  D <= C;
end process;
```

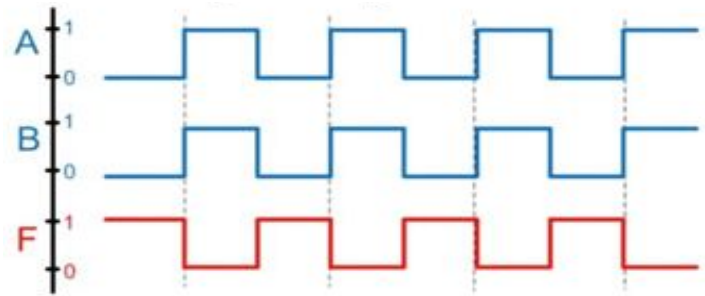


# Introdução

- Exemplo 1: Atribuições **Concorrentes** x **Sequenciais**

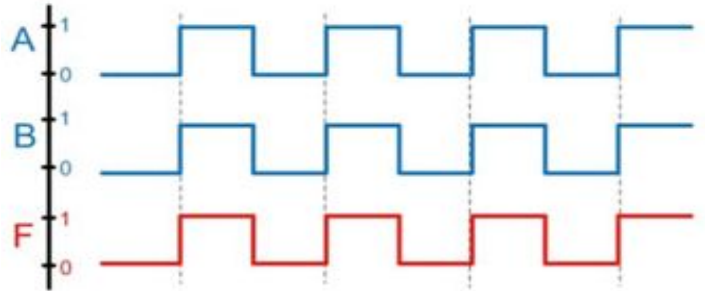
## Concorrente:

```
architecture Ex_arch of Ex is
    signal B: std_logic;
begin
    B <= A;
    F <= not B;
end Ex_arch;
```



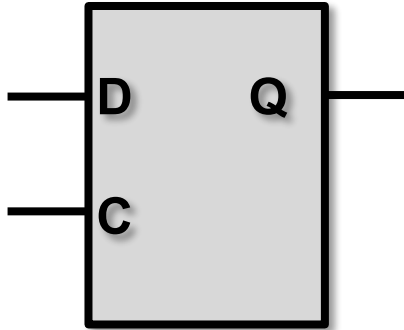
## Sequencial (process):

```
architecture Ex_arch of Ex is
    signal B: std_logic;
begin
    process(A)
    begin
        B <= A;
        F <= not B;
    end process;
end Ex_arch;
```



# Introdução

- Exemplo 2: *Latch D sensível ao nível*



C(t)	Q(t+1)
0	Q(t)
1	D(t)

(Mantém estado)

(Grava D)

```
library ieee;
use ieee.std_logic_1164.all;

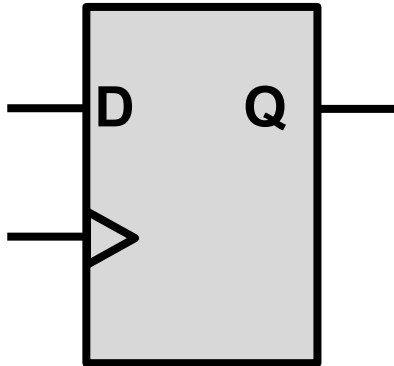
entity D_latch is port (
    C: in std_logic;
    D: in std_logic;
    Q: out std_logic );
end D_latch;

architecture behv of D_latch is
begin
    process (C,D)
    begin
        if (C = '1') then
            Q <= D;
        end if;
    end process;
end behv;
```



# Introdução

- Exemplo 3: *Flip-flop D*



CLK	D(t)	Q(t+1)
0	X	Q(t)
↑	D(t)	D(t)
1	X	Q(t)

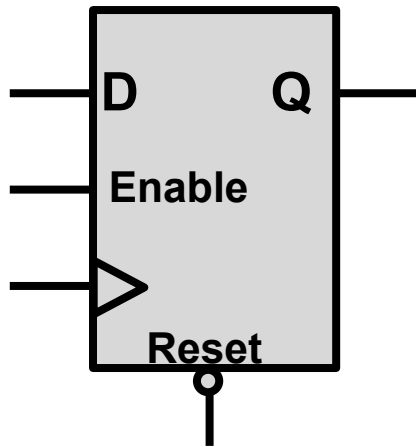
```
library ieee;
use ieee.std_logic_1164.all;

entity D_FF is port (
    CLK: in std_logic;
    D: in std_logic;
    Q: out std_logic );
end D_FF;

architecture behv of D_FF is
begin
    process (CLK)
    begin
        if (CLK'event and CLK = '1')
        then
            Q <= D;
        end if;
    end process;
end behv;
```

# Introdução

- Exemplo 4: **Flip-flop D** com **Enable** e **Reset Assíncrono**



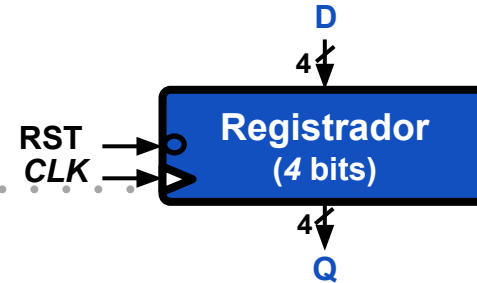
- Sempre que a entrada **RST for 0**:  $Q \leq 0$
- Quando RST dif 0: Q vai depender da entrada CLK e EN
- Se CLK for 1 E for uma borda de subida **E o sinal de Enable (EN) estiver em 1**, então  $Q \leq D$

```
library ieee;
use ieee.std_logic_1164.all;
entity D_FF is port (
    CLK, EN, RST, D: in std_logic;
    Q: out std_logic );
end D_FF;
architecture behv of D_FF is
begin
    process (CLK, RST)
    begin
        if (RST = '0') then
            Q <= '0';
        elsif (CLK'event and CLK = '1')
        then
            if (EN = '1') then
                Q <= D;
            end if;
        end if;
    end process;
end behv;
```

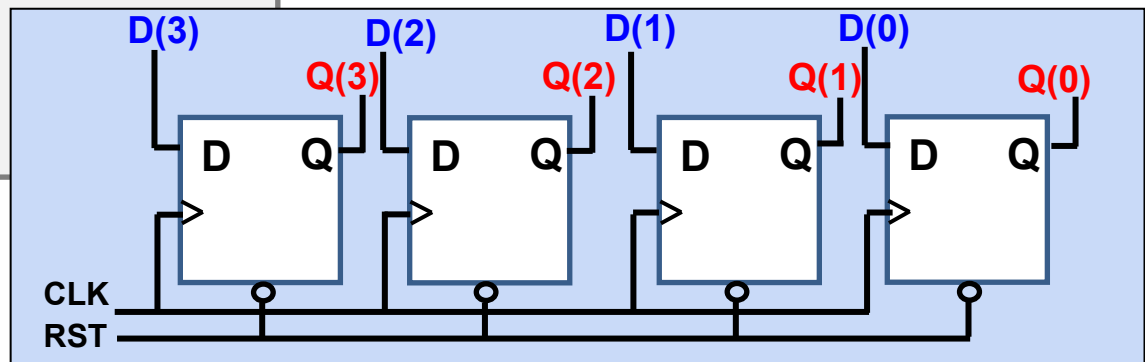
# Introdução

- Exemplo 5: Registrador de 4 bits

```
library ieee;
use ieee.std_logic_1164.all;
entity D_4FF is port (
    CLK, RST: in std_logic;
    D: in std_logic_vector(3 downto 0);
    Q: out std_logic_vector(3 downto 0)
);
end D_4FF;
architecture behv of D_4FF is
begin
    process(CLK, D, RST)
    begin
        if RST = '0' then
            Q <= "0000";
        elsif (CLK'event and CLK = '1') then
            Q <= D;
        end if;
    end process;
end behv;
```



- Enquanto não ocorrer um novo evento no sinal CLK e enquanto esse evento for diferente de '1' (borda de subida), então a saída Q do flip-flop continuará armazenando o valor atual.
- Ao ocorrer um novo evento em CLK, e se esse novo evento for uma transição de '0' para '1' (borda de subida), então a saída Q receberá o novo valor existente na entrada D.



Introdução

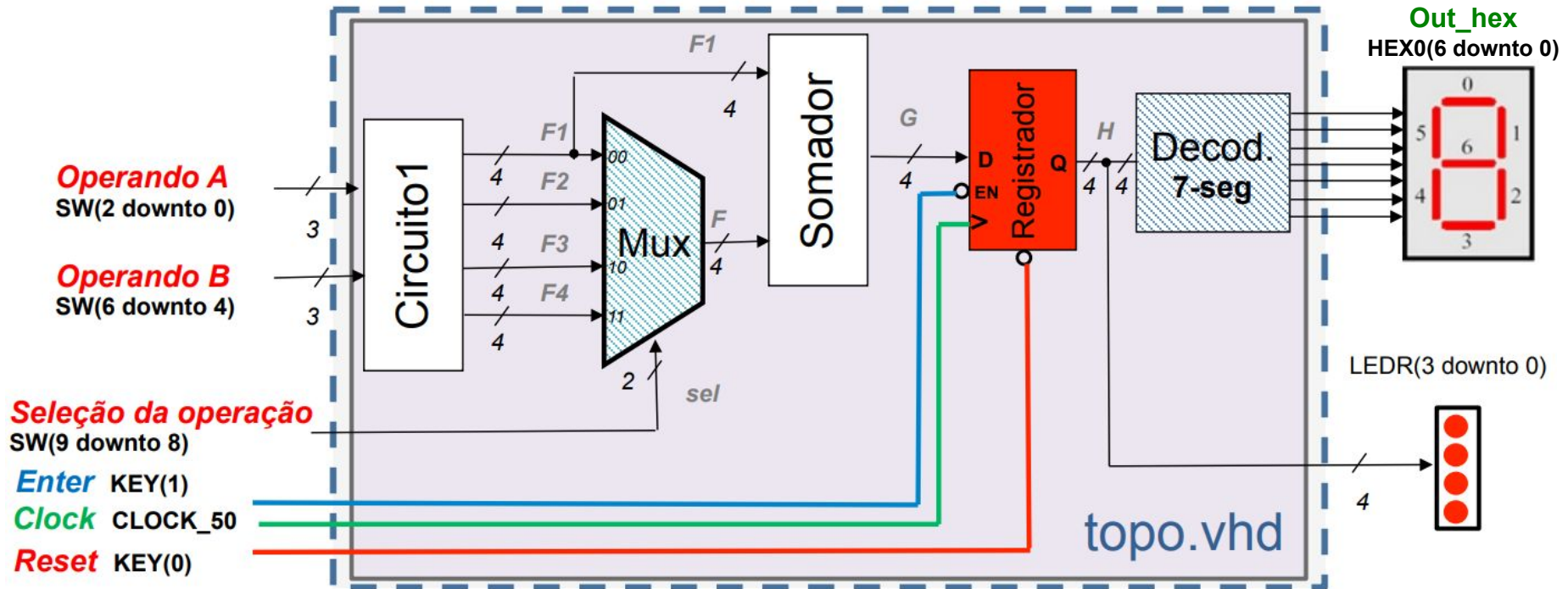
## Tarefa

---

Erros comuns

# Tarefa

- Implementar o circuito aritmético do **lab. 4**, porém com **incluir** um **registrador** para **armazenar** os resultados das operações.

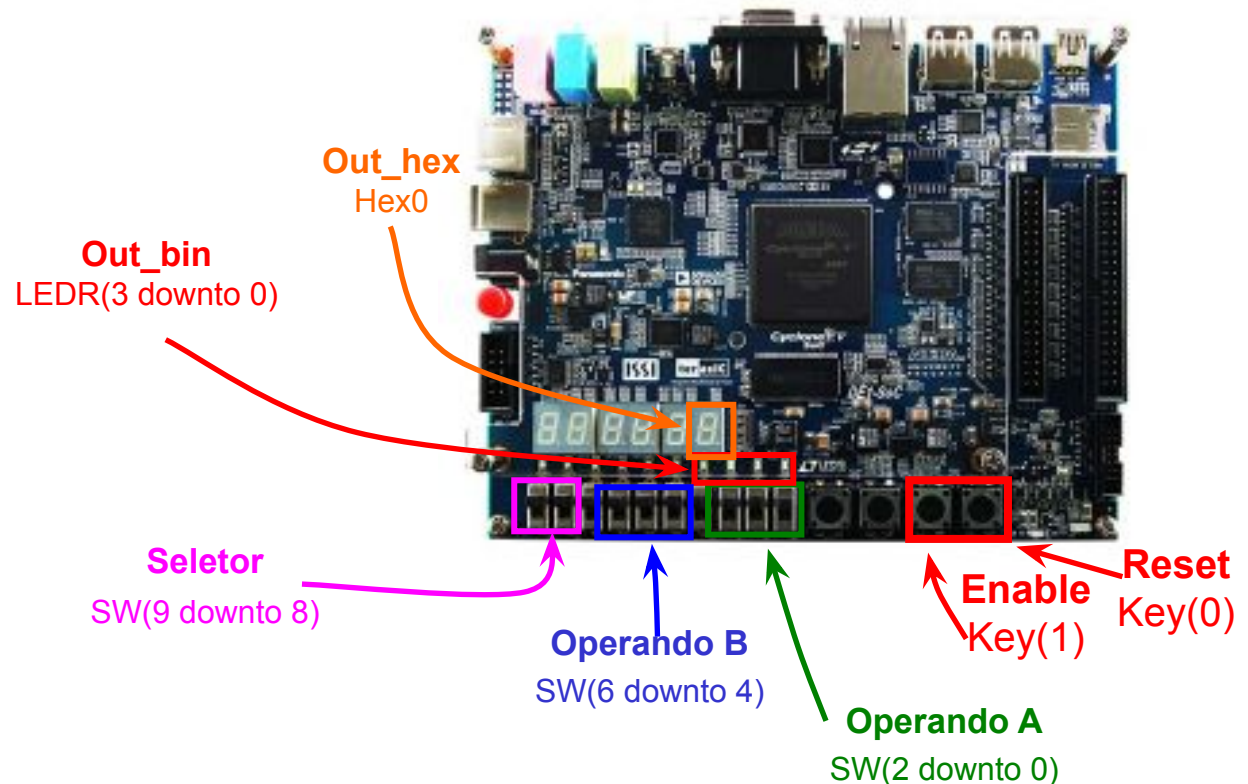


# Tarefa

- Implementar e testar o circuito aritmético do **lab. 4**, porém com **incluir** um **registrador** para **armazenar** os resultados das operações.

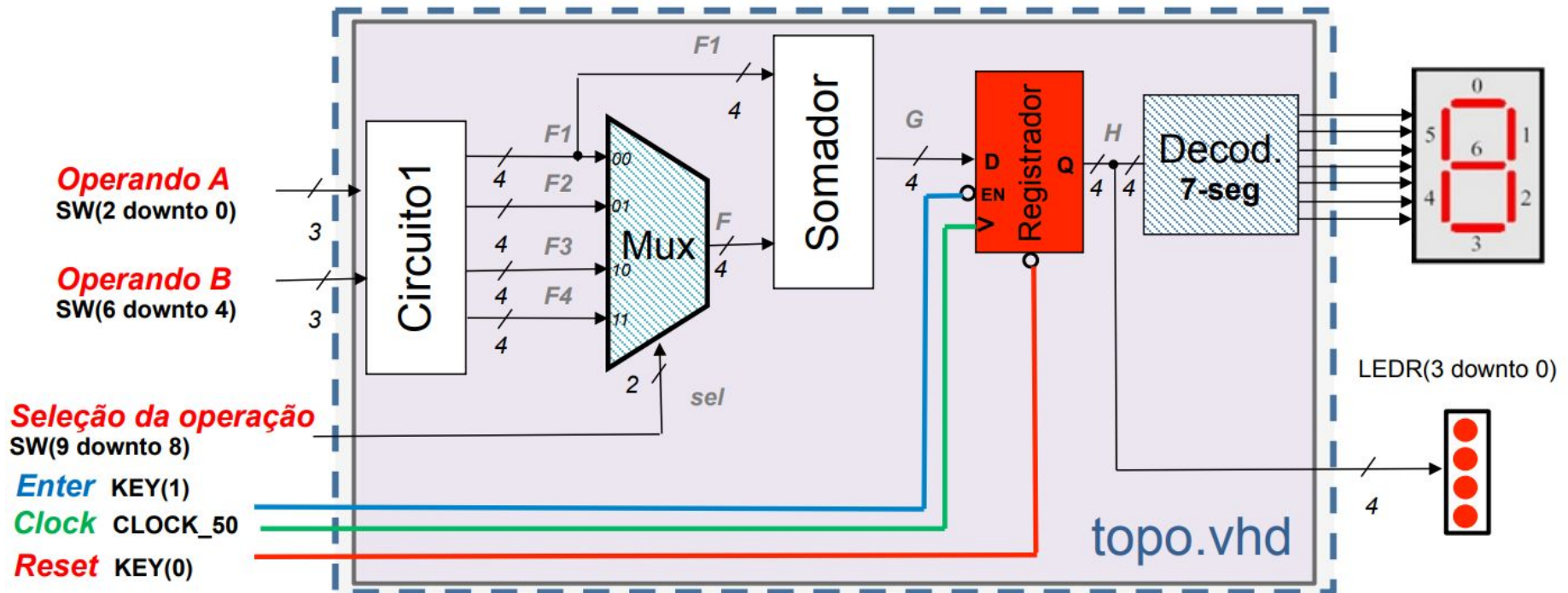
*Tabela de operações*

Seletor	Saída final
00	2A
01	3A
10	A+B
11	A+2B



# Tarefa

- Um **registrador de 4 bits** para armazenar o resultado a ser apresentado nas saídas.
- No slide 11 é apresentado um registrador de 4 bits, implementado com 4 *flip-flops*, porém sem **Enable**. Utilizar os circuitos dos **slides 10 e 11** como base para para escrever o VHDL do registrador de 4 bits solicitados no exercício, com **Reset** e **Enable**.



.....

- \_\_\_\_\_

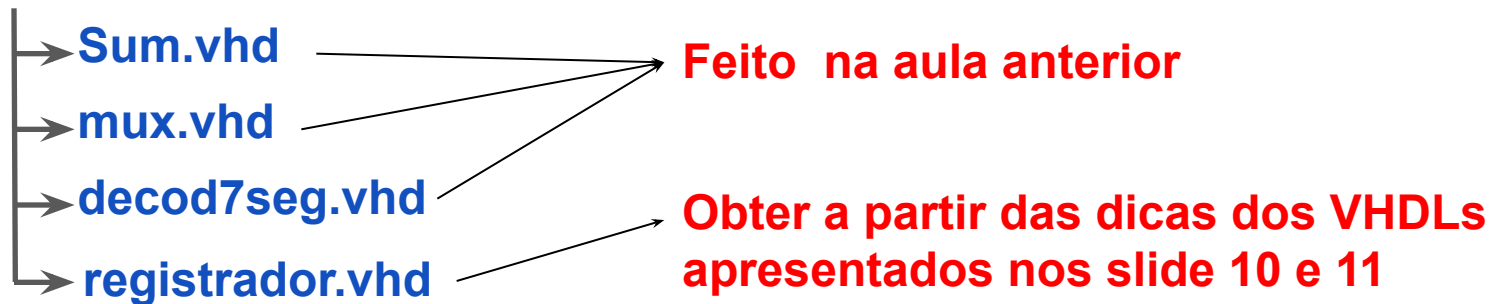


# Tarefa

---

- Sugestão de nome do Projeto: **Aritmetico\_reg**

**Aritmetico\_reg.vhd** → **Incluir a componente registrador**

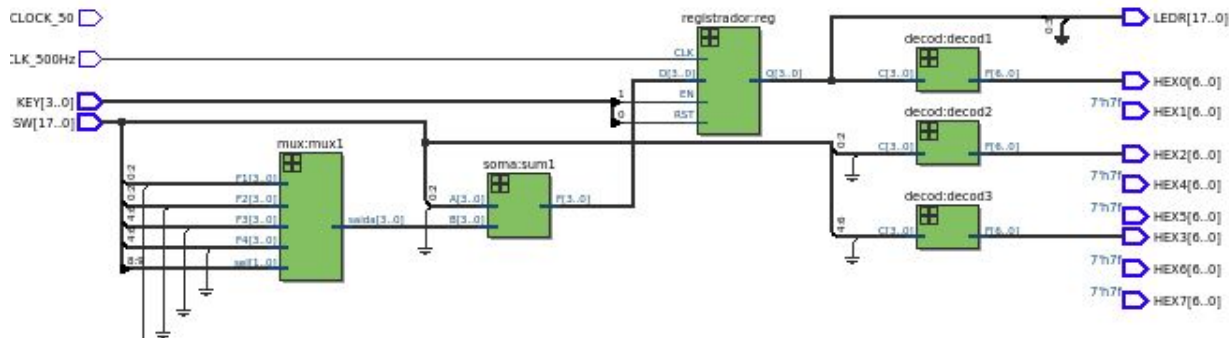


Na entidade do  
topo podem usar:

```
port(CLOCK_50:in std_logic;  
      CLK_500Hz:in std_logic;  
      KEY:in std_logic_vector(3 downto 0);  
      SW:in std_logic_vector(17 downto 0);  
      LEDR:out std_logic_vector(17 downto 0);  
      HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,HEX6,HEX7: out  
      std_logic_vector(6 downto 0));  
end topo;
```

## Tarefa

- Uma vez compilado, e verifique se há erros de sintaxe;
- Visualize o diagrama de blocos gerado:  
**Tools-> Netlist Viewers-> RTL Viewer.**



- Teste o circuito no kit de desenvolvimento **DE1-SoC** como nas aulas anteriores.

## Tarefa Avançada

---

- Faça modificações apropriadas para que as entradas **A** e **B** sejam mostradas nos displays **HEX2** e **HEX3**, respectivamente.  
**Dica:** Não é preciso criar três decodificadores
- Visualize o **diagrama de blocos** gerado:  
**Tools-> Netlist Viewers-> RTL Viewer.**
- Teste o circuito no kit de desenvolvimento **DE1-SoC** como nas aulas anteriores.

.....

**Atenção:** Na próxima aula  
teremos o exercício 3!

## Após finalizar o laboratório:

- 1) Salve os arquivos desejados no moodle;
- 2) Responda o **miniteste** da aula;
- 3) Faça logout e desligue o computador;
- 4) Organize sua bancada e cadeiras;
- 5) Organize o kit de desenvolvimento (não troque as fontes e cabos com outros kits);
- 6) Devolva o kit à professora.