



FEDERAL UNIVERSITY  
OF SANTA CATARINA

# Laboratório 5: Processos, Flip-Flops e Registradores

---

EEL5105 – Circuitos e Técnicas Digitais

## Objetivos

---

- Entender o uso de *process* em **VHDL**.
- Trabalhar os conceitos de **flip-flop** e **registrador**.
- Implementar **flip-flops** e **registradores** em **VHDL** utilizando *processes*.
- Fazer 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 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** (e.g., somente **B <= D** para **B** 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;
```

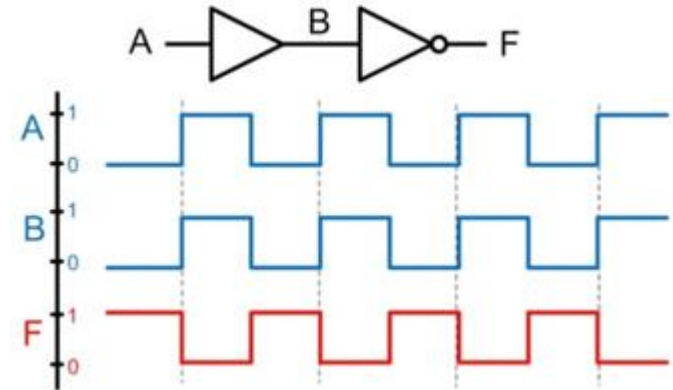
```
entity Ex is port (  
  A: in std_logic;  
  F: out std_logic );  
end Ex;
```

# Introdução

- **Exemplo 1:** Atribuições **Concorrentes** x **Sequenciais**  
*De Introduction to Logic Circuits & Logic Design (LaMeres)*

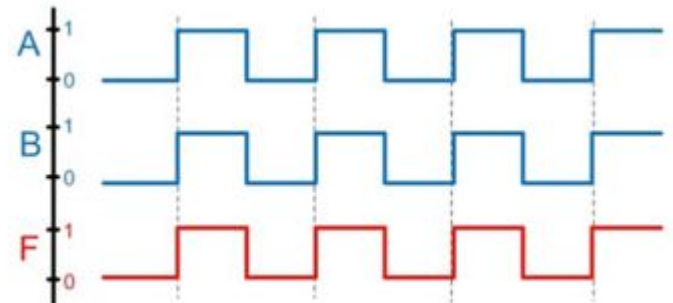
- **Concorrente:**

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



- **Sequential (*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;
```



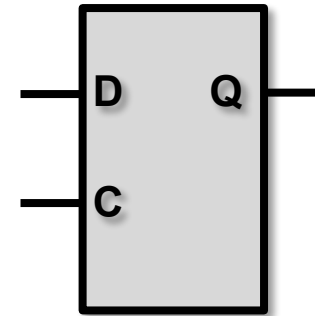
# Introdução

- Exemplo 2: **Latch D sensível ao nível**

```
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;
```



C(t)	Q(t+1)	
0	Q(t)	(Mantém estado)
1	D(t)	(Grava D)

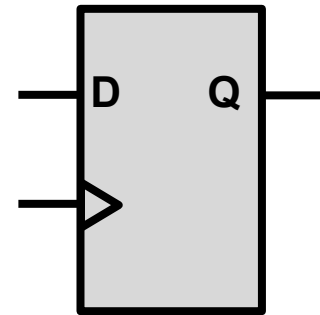
# Introdução

- Exemplo 3: *Flip-flop D*

```
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;
```

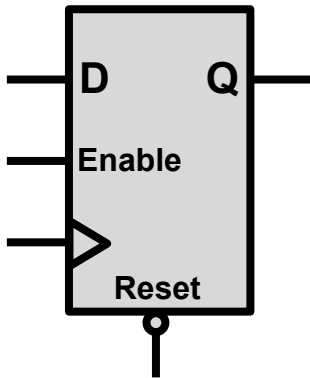


CLK	D(t)	Q(t+1)
0	X	Q(t)
↑	D(t)	D(t)
1	X	Q(t)



# Introdução

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



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

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

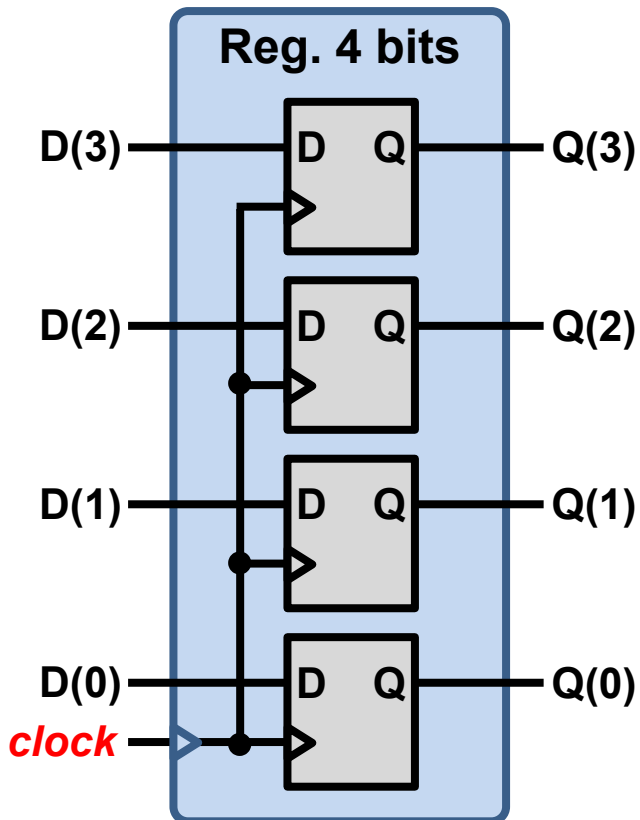
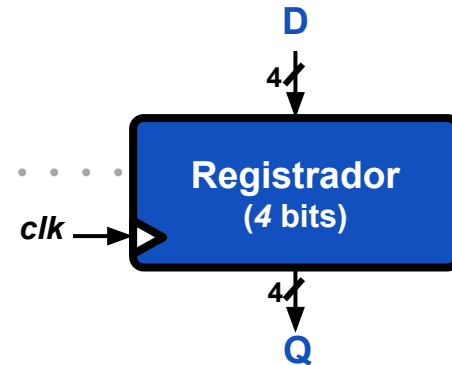
architecture behv of D_FF is
begin

    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 Reg4 is port (
    CLK: in std_logic;
    D: in std_logic_vector(3 downto 0);
    Q: out std_logic_vector(3 downto 0));
end Reg4;

architecture behv of Reg4 is
begin
    process (CLK)
    begin
        if (CLK'event and CLK = '1') then
            Q <= D;
        end if;
    end process;
end behv;
```

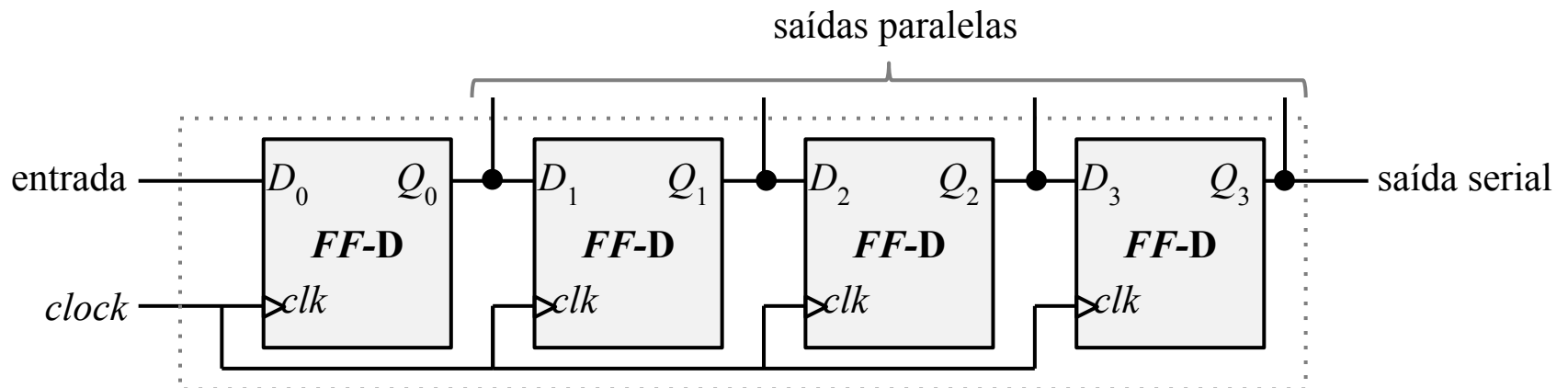
# Tarefas

---

# Tarefa 1

---

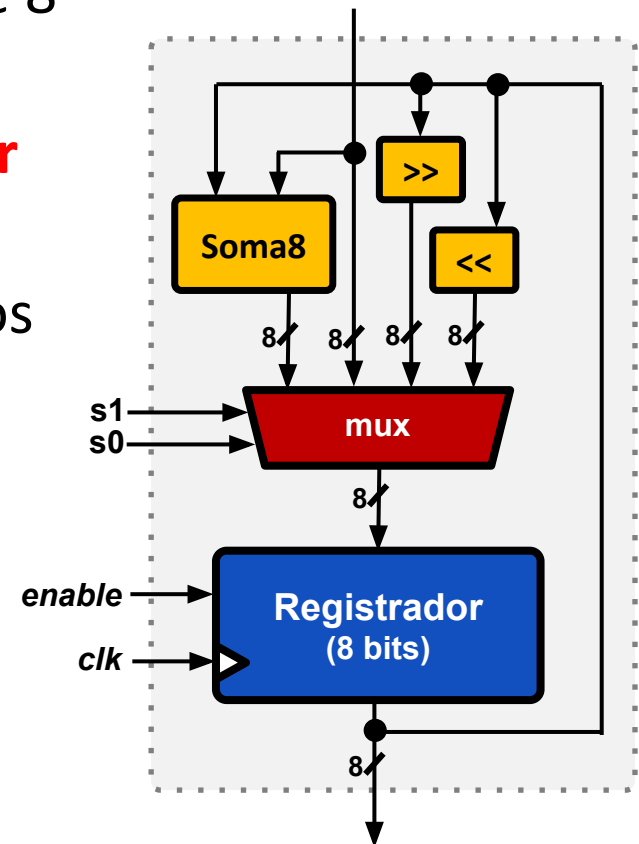
- Montar um **registrador de deslocamento** de 4 bits.
  - Saídas paralelas em **LEDRs**;
  - Entrada em **SW(0)**;
  - Inicialmente, coloque **clock** em **SW(1)** para ver o funcionamento em “câmera lenta” e depois use **CLK\_1Hz** (**port** em usertop.vhd) como sinal de **clock**.
    - **CLK\_1Hz** = **clock** (impreciso) de 1 Hz no Emulador.



## Tarefa 2

- Montar um **registrador multimodo** de 8 bits.
  - **Passo 1)** Projete um registrador de 8 bits com **enable** e **clock**.
  - **Passo 2)** Projete um **multiplexador** com 4 entradas de 8 bits.
  - **Passo 3)** Faça a integração de todos os elementos do projeto, usando **soma8.vhd** para fazer a soma e fazendo os deslocamentos como descrito no próximo slide.

s1	s0	Operação
0	0	Soma
0	1	Carga paralela
1	0	Desloca à direita
1	1	Desloca à esquerda

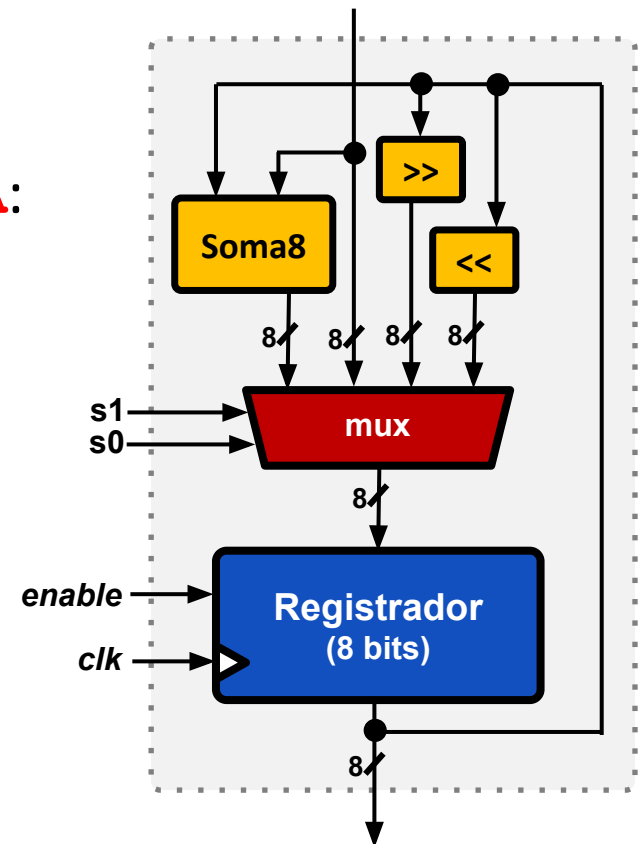


## Tarefa 2

- Montar um **registrador multimodo** de 8 bits.

- Deslocamento à direita **>>** de **A**:  
`S <= "0" & A(7 downto 1);`
- Deslocamento à esquerda **<<** de **A**:  
`S <= A(6 downto 0) & "0";`

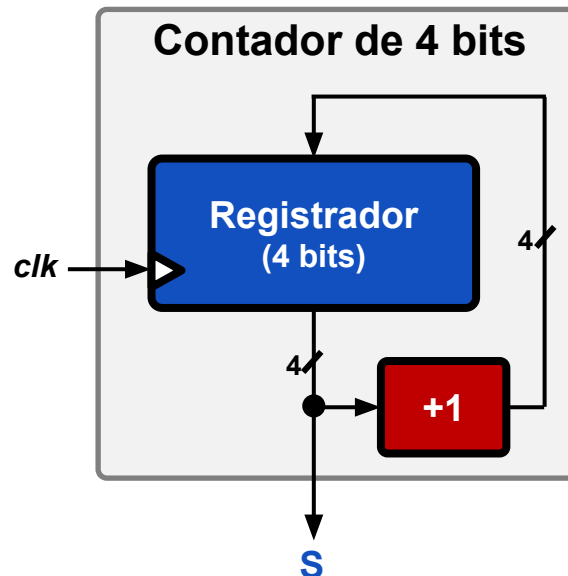
s1	s0	Operação
0	0	Soma
0	1	Carga paralela
1	0	Desloca à direita
1	1	Desloca à esquerda



## Tarefa 3

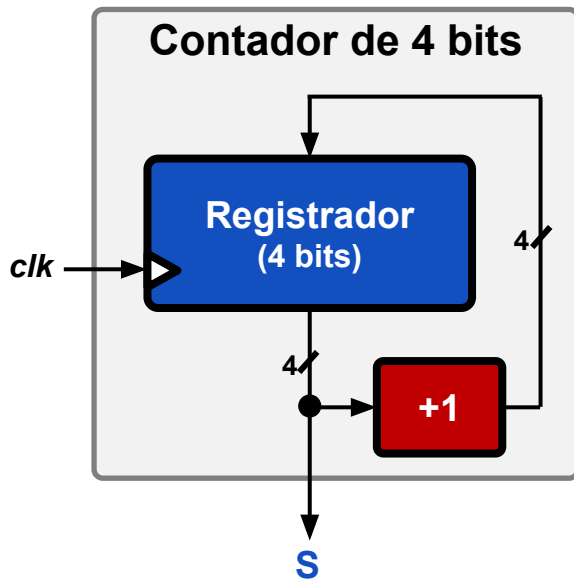
---

- Modifique o contador de 4 bits cujo código é mostrado no próximo slide, de forma a obter uma contagem de **0** a **9**. Inclua ainda:
  - **Enable**: pausa contagem
  - **Clear**: zera a contagem de forma assíncrona
  - **Max**: nível alto quando contagem atinge máximo



## Tarefa 3

- Ideia agora é trabalhar diretamente no *process*, sem usar **componentes** externos.



```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Conta4 is port (
    CLK: in std_logic;
    S: out std_logic_vector(3 downto 0) );
end Conta4;

architecture behv of Conta4 is
    signal cnt: std_logic_vector(3 downto 0) := "0000";
begin
    process(CLK)
    begin
        if (CLK'event and CLK = '1') then
            cnt <= cnt + "0001";
        end if;
    end process;
    S <= cnt;
end behv;
```

Definição de um valor inicial para um **signal**.

Essa soma requer o uso das bibliotecas de aritmética: **std\_logic\_arith/std\_logic\_unsigned**