



**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 sequenciais, latches e flip-flops*

# Objetivos do laboratório

---

1. Entender o conceito de circuitos sequenciais.
2. Entender o conceito e diferença entre flip-flops e latches.
3. Entender o conceito de registradores em VHDL.
4. Implementação de flip-flops, latches e registradores em VHDL.
5. Estudo de caso: projeto de calculadora personalizada, com apresentação dos resultados em displays de 7-segmentos, e uso de registradores como elementos de “memória”.

## *Uso de “process” na descrição de circuitos sequenciais*

# Descrição de circuitos sequenciais em VHDL

```
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;
```

## Process

- Define uma **SEQUÊNCIA DE COMANDOS** a ser realizada pelo circuito.
- O **processo é acionado**, e sua sequência de comandos é executada, sempre que ocorrer uma **alteração** em algum elemento da sua **LISTA DE PARÂMETROS** (Ex. **C** ou **D**).
- **Um processo nunca termina - CÍCLICO.**
- Dessa forma, após a execução do último comando, o primeiro comando da sequência é executado, **SEMPRE** que ocorrer uma nova alteração em algum parâmetro.
- Obs. Comando **IF .. THEN .. ELSE** é utilizado **APENAS** dentro de um *process*.

# Descrição de circuitos sequenciais em VHDL

## Process



```
process (A, B, C, D)
begin
```

```
    A <= '1';
    B <= '1';
    B <= D;
    A <= not B;
    C <= A and '1';
    D <= C;
```

```
end process;
```

Sequencial

- A lista de parâmetros de um processo é denominada **SENSITIVITY LIST**.
- Os valores atribuídos aos sinais pelos comandos do processo, **só serão válidos após a execução do último comando**, ou seja, após “sair” do processo.
- Se existirem **várias atribuições** a um mesmo sinal, **APENAS a última atribuição será válida** (ex. sinal B no corpo do processo).
- Não é permitido declarar sinais dentro de um processo.

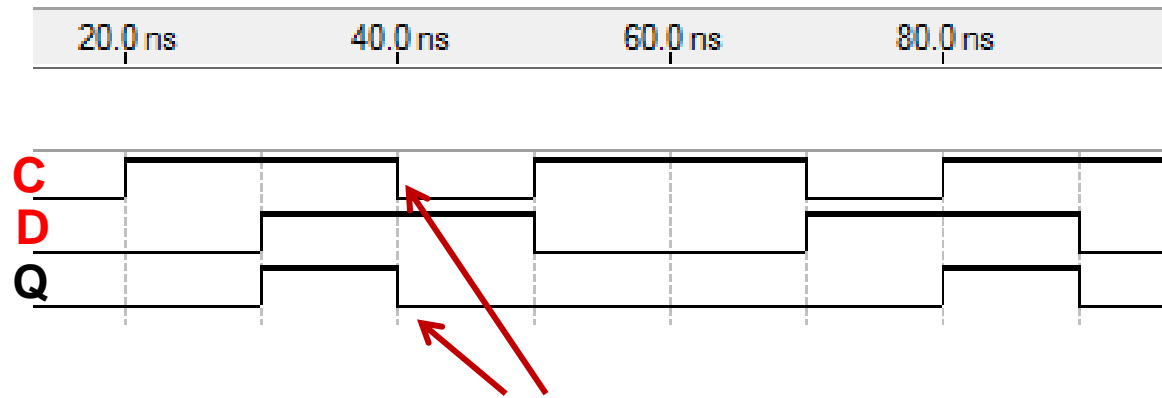
# Descrição de circuitos sequenciais em VHDL

```
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;
```

## Resultado da simulação do VHDL



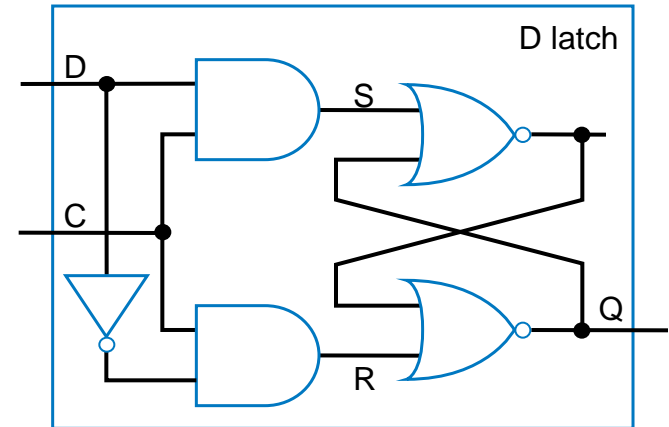
C foi alterado para '0', logo o processo é executado novamente, e B receberá '0', pois o teste do *IF* será falso. Logo, Q será zero (combinacional).

## *Flip-flops e latches em VHDL*



# Latch 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;
```



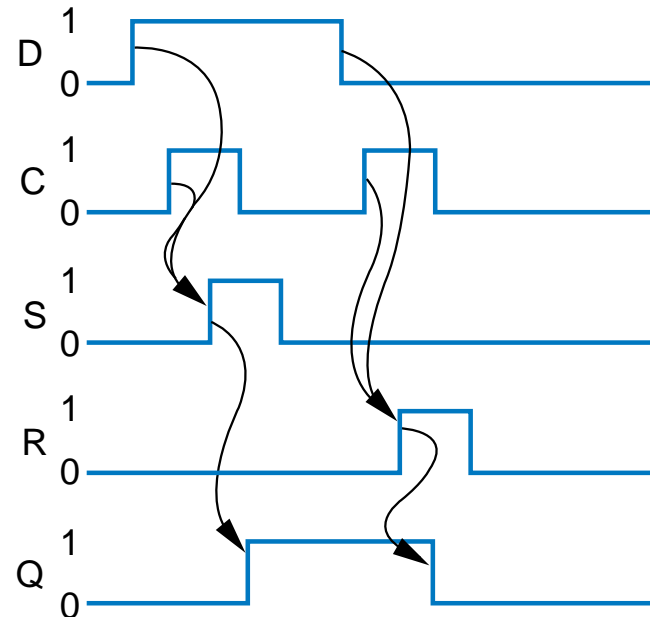
C	D	$Q_{t+1}$
0	X	$Q_t$
1	0	0
1	1	1

**Mantém estado**

# Latch D

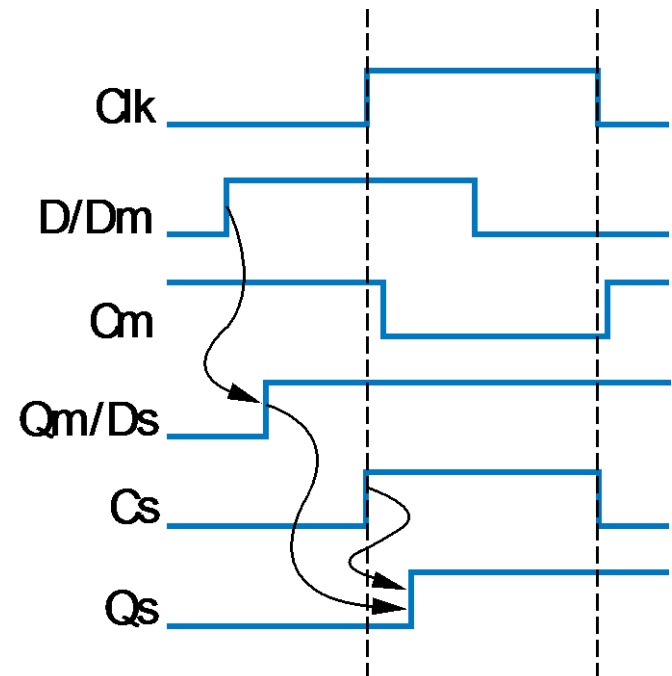
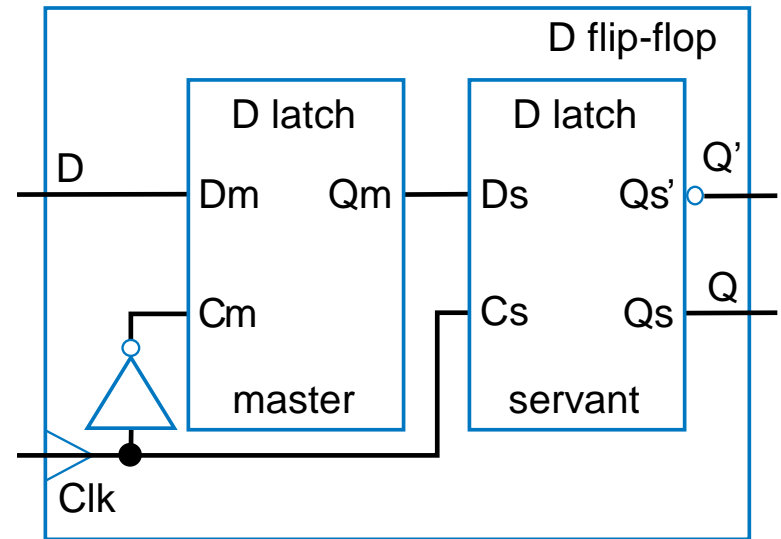
```
library ieee;
use ieee.std_logic_1164.all;
entity D_latch is port (
    C: in std_logic;
    D: in std_logic;
    Q: out std_logic
);
end D_latch;
architecture behv of D_latch is
begin
    process(C, D)
    begin
        if (C = '1') then
            Q <= D;
        end if;
    end process;
end behv;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

C	D	$Q_{t+1}$
0	X	$Q_t$ <b>Mantém estado</b>
1	0	0
1	1	1

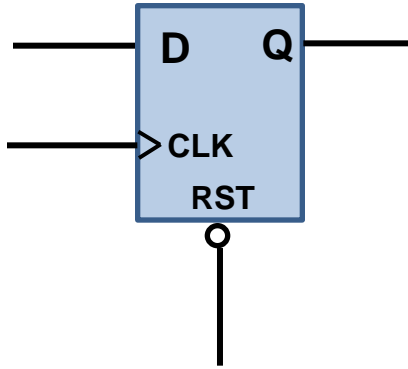


# Flip-Flop D

```
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, D)
    begin
        if (CLK'event and CLK = '1') then
            Q <= D;
        end if;
    end process;
end behv;
```



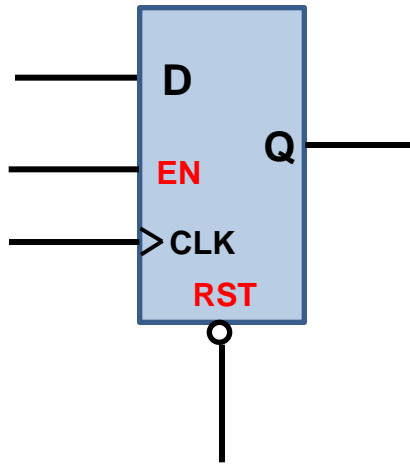
# Flip-Flop D com RESET assíncrono



- Sempre que a entrada RST for Zero, a saída Q será Zero.
- Quando RST for diferente de Zero, o valor na saída Q vai depender da entrada CLK.
- Se CLK for '1' E for uma borda de subida, então Q receberá a entrada D.

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

# Flip-Flop D com *Reset* e *Enable*



- Sempre que a entrada RST for Zero, a saída Q será Zero.
- Se CLK for '1' E for uma borda de subida **E o sinal de Enable (En) estiver em '1'**, então Q receberá a entrada D.

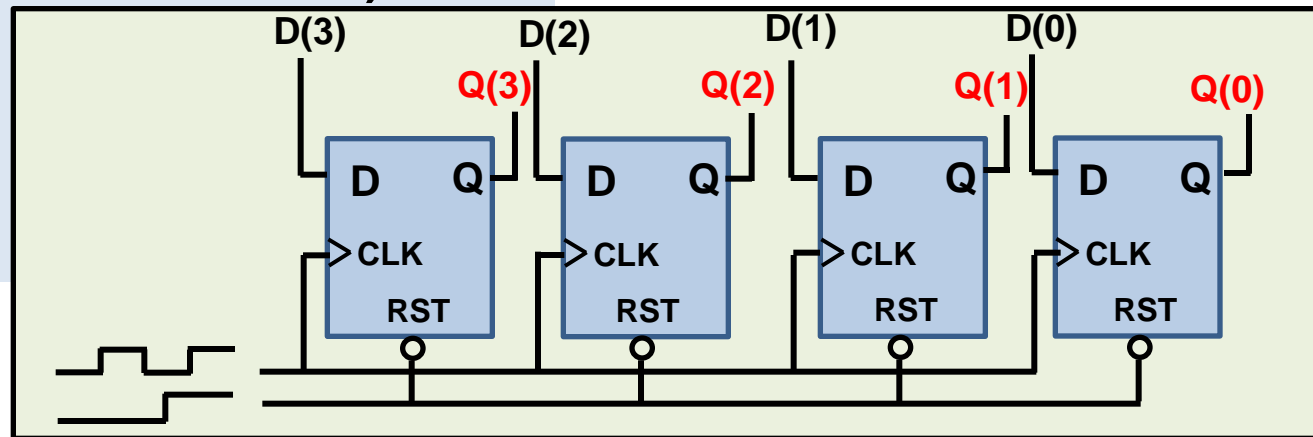
```
library ieee;
use ieee.std_logic_1164.all;
entity D_FF is port (
    CLK: in std_logic;
    RST: in std_logic;
    EN: in std_logic;
    D: in std_logic;
    Q: out std_logic
);
end D_FF;
architecture behv of D_FF is
begin
    process (CLK, RST, D)
    begin
        if RST = '0' then
           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

# 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)
    begin
        if RST = '0' then
            Q <= "0000";
        elsif (CLK'event and CLK = '1') then
            Q <= D;
        end if;
    end process;
end behv;
```

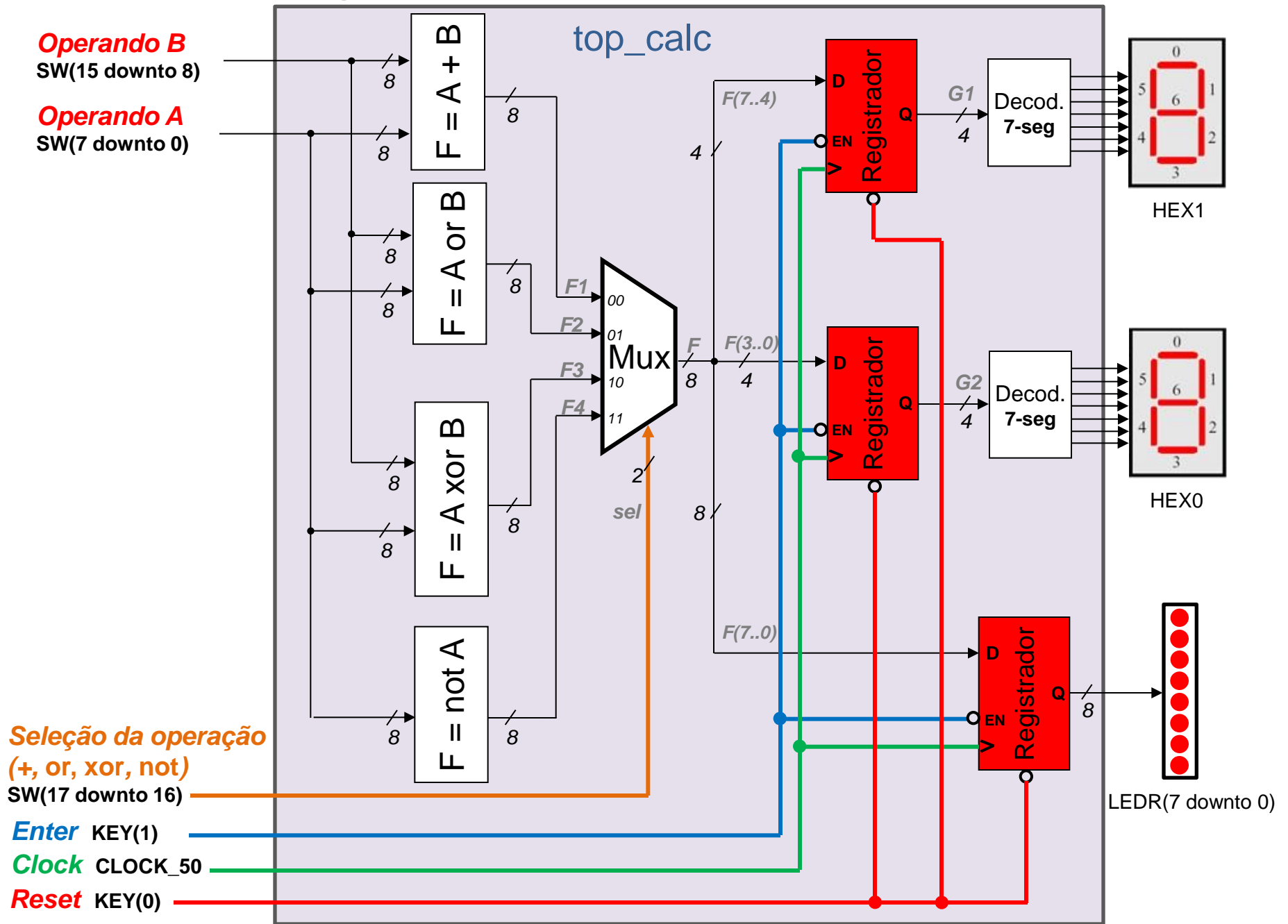
Enquanto não ocorrer um novo evento no sinal CLK e enquanto esse evento for diferente de '1' (borda de subida), então a saída Q do flip-flop continuará armazenando o valor atual.

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.



***Tarefa a ser realizada:  
Mini-calculadora com registradores para  
armazenamento (memória) de resultados.***

# Uso de registradores na mini-calculadora VHDL





# Descrição do circuito a ser implementado

---

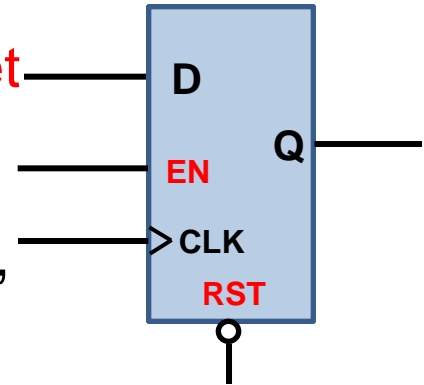
- O circuito consiste na **mini-calculadora do lab. anterior**, porém com a **inclusão de registradores** para armazenar os resultados das operações.
  - Um **registrador de 4 bits** para armazenar a parte baixa do resultado a ser apresentado em **HEX0**.
  - Um **registrador de 4 bits** para armazenar a parte alta do resultado a ser apresentado em **HEX1**.
  - Um **registrador de 8 bits** para armazenar o resultado a ser apresentado, em binário, nos **LEDs vermelhos**.
- Ao se pressionar o **botão KEY(0) – “Reset”**, os flip-flops deverão ser “limpos”, apagando os LEDs, e apresentando o valor 0 (zero) nos displays de sete segmentos.
- Ao se pressionar o **botão KEY(1) – “Enter”**, os flip-flops são habilitados para escrita, armazenando os valores presentes nas suas entradas (“memória”).
- As entradas **CLK (clock)** dos flip-flops recebem um sinal de relógio gerado por um cristal presente na placa DE2 (**CLOCK\_50**).

## Dicas úteis

# Dicas úteis

- No slide 13 é apresentado um flip-flop D com **Reset** e **Enable** (ver símbolo ao lado).
- No slide 14 é apresentado um registrador de 4 bits, implementado com 4 flip-flops, porém sem **Enable**.
- Utilizar os circuitos dos **slides 13 e 14** como base para para escrever o VHDL dos dois registradores de 4 bits e do registrador de 8 bits solicitado no exercício, com **Reset** e **Enable**.
- O sinal de relógio (Clock ou CLK) do circuito deve ser obtido diretamente da placa, utilizando o sinal **CLOCK\_50**.
- Os push buttons (KEY), **quando pressionados**, fornecem '0'.
- Na **entity do novo topo**, devem ser adicionados os sinais:

**key**: in std\_logic\_vector(1 downto 0);    — — **KEY(0)** e **KEY(1)**  
**clock\_50**: in std\_logic;                    — — clock 50 MHz



# Interface com o usuário (entrada/saída)

