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

"Introdução a circuitos Sequenciais em VHDL "

Prof. Héctor Pettenghi Roldán*

Hector@eel.ufsc.br

Florianópolis, Agosto de 2019.

Objetivos do laboratório

1. Entender o conceito de circuitos sequenciais.
2. Explicar o uso de descrição VHDL comportamental para circuitos sequenciais.
3. Implementação de Flip-flops e registradores em VHDL.
4. Entender o conceito, e uso, de FSMs como estrutura de controle do fluxo de operações de um circuito combinacional.
5. Implementar registrador deslocador com reset assíncrono, clock e sinal de enable.

2.- 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;
```

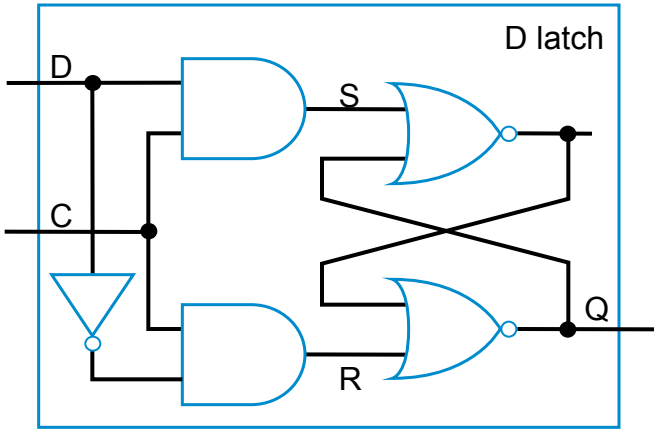
Process

- 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*.

3.- Implementação de Flip-Flop e Latches em VHDL

Latch

```
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;
```



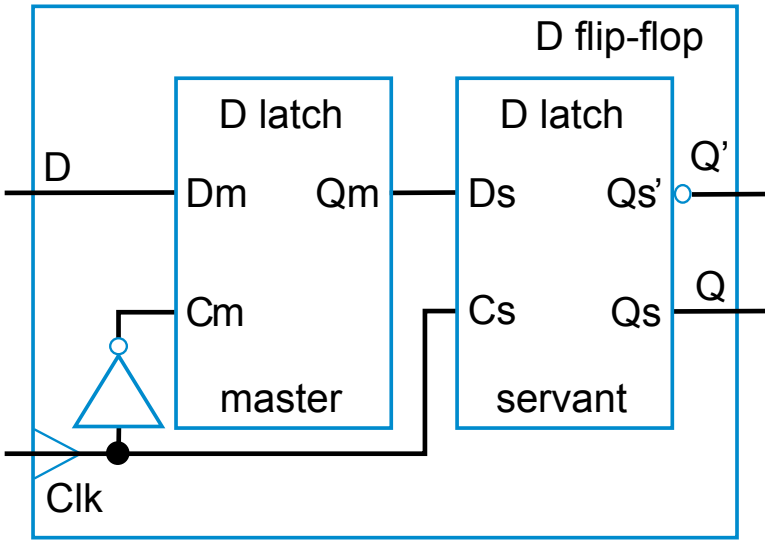
C	D	Q_{t+1}
0	X	Q_t
1	0	0
1	1	1

Mantém estado

3.- Implementação de Flip-Flop e Latches em VHDL

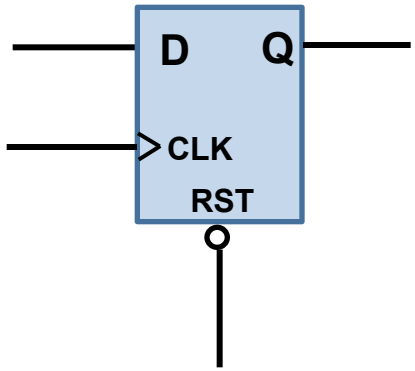
Flip-Flop D

```
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;
```



3.- Implementação de Flip-Flop e Latches em VHDL

Flip-Flop D com Reset Assíncono



- Sempre que a entrada RST for Zero, a saída Q será Zero.

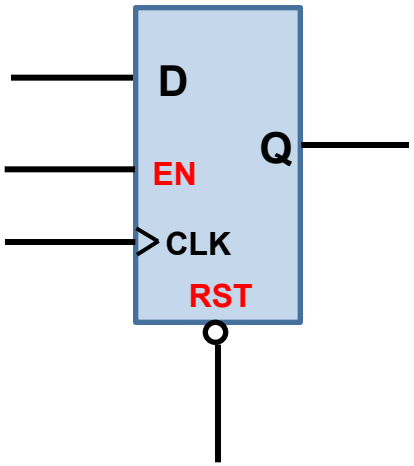
- Quando RST for diferente de Zero, o valor na saída Q vai depender da entrada CLK.

- 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;
```

3.- Implementação de Flip-Flop e Latches em VHDL

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, EN)
    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;
```

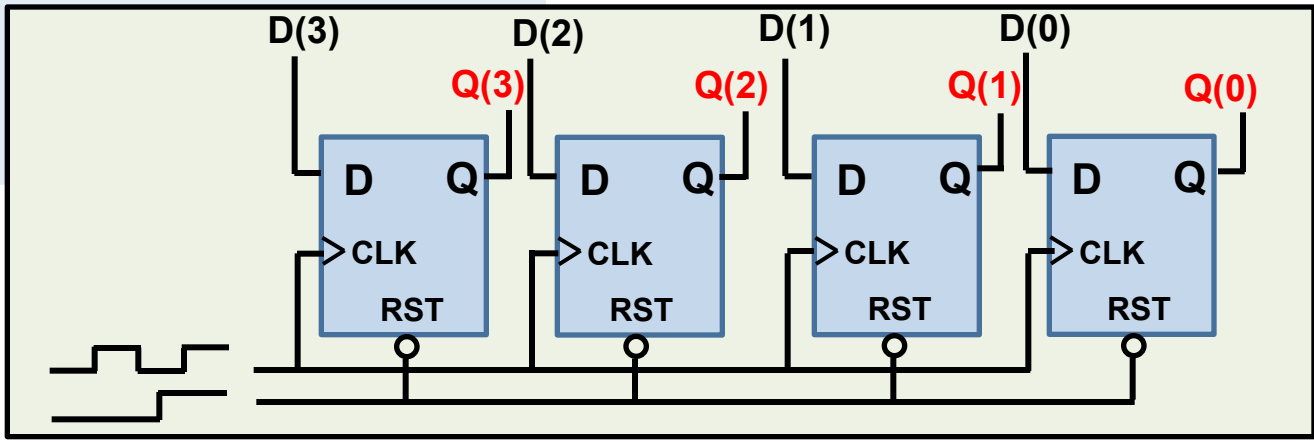
3.- Implementação de Flip-Flop e Latches em VHDL

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.

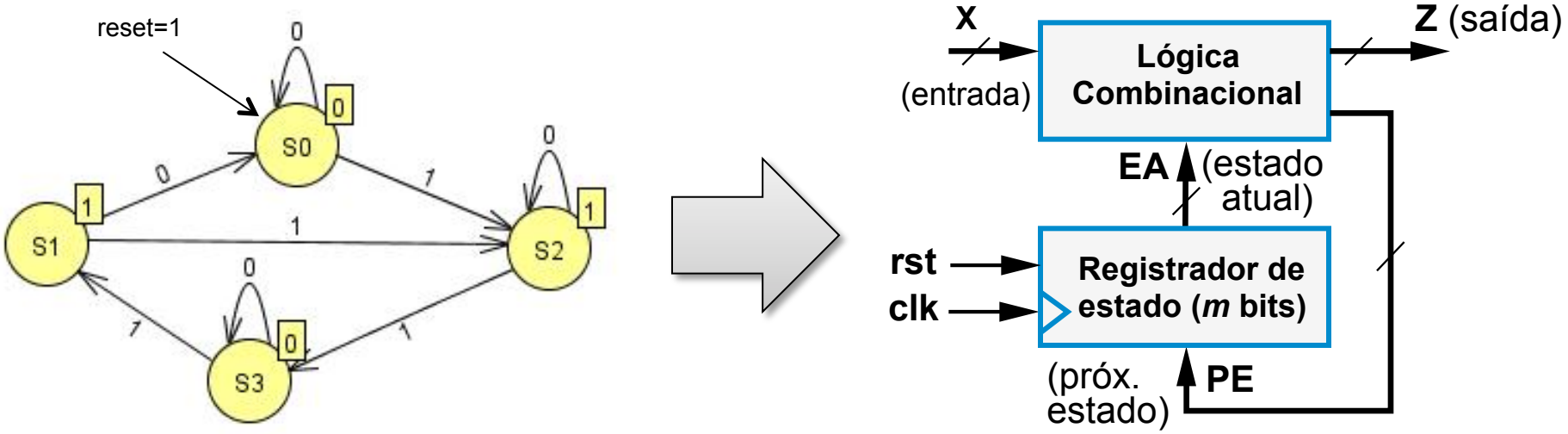


TAREFA 1:
Implementar e simular

4.- Implementação de FSM em VHDL

Estrutura da FSM

- Dois módulos:
  - Armazenamento do “estado atual”; e
  - Cálculo da “saída” e do “próximo estado”

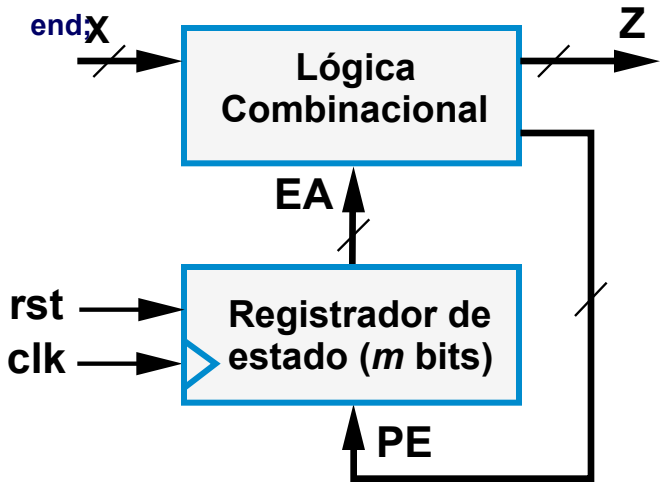


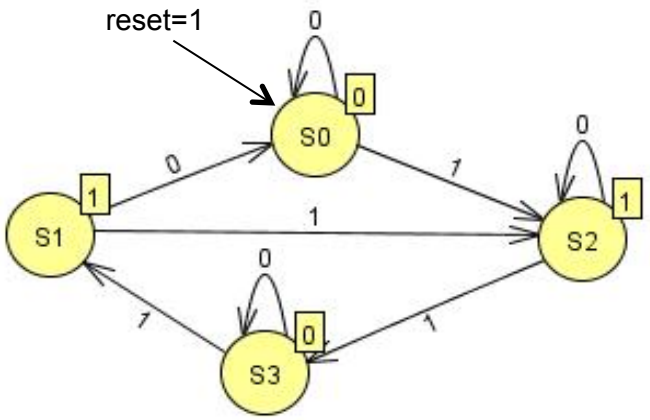
4.- Implementação de FSM em VHDL

VHDL típico

VHDL típico de uma máquina de estados – 2 processos

```
library ieee;
use ieee.std_logic_1164.all;
entity MOORE is port(X, clock, reset : in std_logic;     Z: out std_logic);     end;

architecture A of MOORE is
      type STATES is (S0, S1, S2, S3);
      signal EA, PE : STATES;
begin

      process (clock, reset)
      begin
            if reset= '1' then
                  EA <= S0;
            elsif clock'event and clock='1' then
                  EA <= PE ;
            end if;
      end process;

      process(EA, X)
      begin
            case EA is
                  when S0 =>        Z <= '0';
                                    if X='0' then PE <=S0; else PE <= S2; end if;
                  when S1 =>        Z <= '1';
                                    if X='0' then PE <=S0; else PE <= S2; end if;
                  when S2 =>        Z <= '1';
                                    if X='0' then PE <=S2; else PE <= S3; end if;
                  when S3 =>        Z <= '0';
                                    if X='0' then PE <=S3; else PE <= S1; end if;
            end case;
      end process;

end A;
```





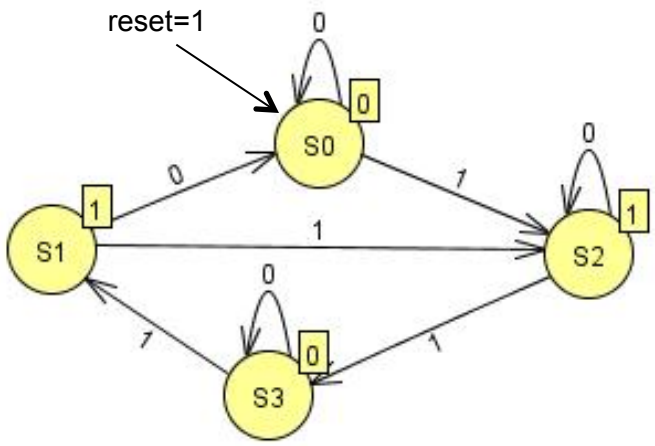
4.- Implementação de FSM em VHDL

VHDL típico

VHDL típico de uma máquina de estados – 2 processos

```
library ieee;
use ieee.std_logic_1164.all;
entity MOORE is port(X, clock, reset : in std_logic;     Z: out std_logic);    end;

architecture A of MOORE is
      type STATES is (S0, S1, S2, S3);
      signal EA, PE : STATES;
begin

      process (clock, reset)
      begin
            if reset= '1' then
                  EA <= S0;
            elsif clock'event and clock='1' then
                  EA <= PE ;
            end if;
      end process;

      process(EA, X)
      begin
            case EA is
                  when S0 =>      Z <= '0';
                                  if X='0' then PE <=S0; else PE <= S2; end if;
                  when S1 =>      Z <= '1';
                                  if X='0' then PE <=S0; else PE <= S2; end if;
                  when S2 =>      Z <= '1';
                                  if X='0' then PE <=S2; else PE <= S3; end if;
                  when S3 =>      Z <= '0';
                                  if X='0' then PE <=S3; else PE <= S1; end if;
            end case;
      end process;

end A;
```

Definição dos estados possíveis da máquina e de variáveis que podem assumir esses estados.

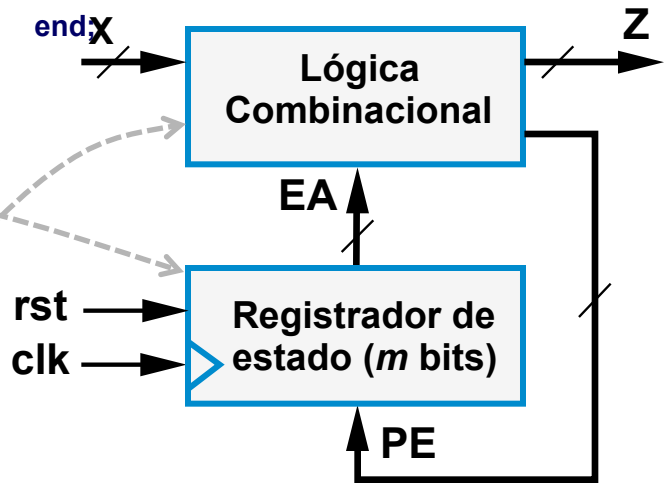


4.- Implementação de FSM em VHDL

VHDL típico

VHDL típico de uma máquina de estados – 2 processos

```
library ieee;
use ieee.std_logic_1164.all;
entity MOORE is port(X, clock, reset : in std_logic;     Z: out std_logic);     end;

architecture A of MOORE is
      type STATES is (S0, S1, S2, S3);
      signal EA, PE : STATES;
begin
   process (clock, reset)
   begin
          if reset= '1' then
                EA <= S0;
        elsif clock'event and clock='1' then
                EA <= PE ;
        end if;
   end process;

   process(EA, X)
   begin
          case EA is
              when S0 =>        Z <= '0';
                                if X='0' then PE <=S0; else PE <= S2; end if;
              when S1 =>        Z <= '1';
                                if X='0' then PE <=S0; else PE <= S2; end if;
              when S2 =>        Z <= '1';
                                if X='0' then PE <=S2; else PE <= S3; end if;
              when S3 =>        Z <= '0';
                                if X='0' then PE <=S3; else PE <= S1; end if;
          end case;
   end process;

end A;
```



TAREFA 2: Implementar e simular

5.- Implementação de deslocador esquerda em VHDL

```
-- sr_in recebe palavra de N bits (vetor de N bits). A cada pulso de clock, a palavra em sr_in é deslocada 1 bit
 --para a esquerda, e copiada para sr_out, também de N bits.
library ieee;
use ieee.std_logic_1164.all;

entity desloc_1_bit_esq is
  generic  ( N : natural := 64 );
  port       (  clk, enable, reset : in std_logic;
                sr_in              : in std_logic_vector((N - 1) downto 0);
                sr_out    : out std_logic_vector((N - 1) downto 0)
               );
end entity;
architecture rtl of desloc_1_bit_esq is
  signal sr: std_logic_vector ((N - 1) downto 0); -- Registrador de N bits
begin
  process (clk, reset)
  begin
    if (reset = '0') then                -- Reset assíncrono do registrador
      sr <= (others => '0');
    elsif (rising_edge(clk)) then   -- Sinal de clock do registrador (subida)
      if (enable = '1') then           -- Sinal de enable do registrador
        -- Desloca 1 bit para a esquerda. Bit mais significativo é perdido.
        sr((N - 1) downto 1) <= sr_in((N - 2) downto 0);
        sr(0) <= '0';
      end if;
    end if;
  end process;
  sr_out <= sr;
end rtl;
```

TAREFA 3:
Implementar e simular

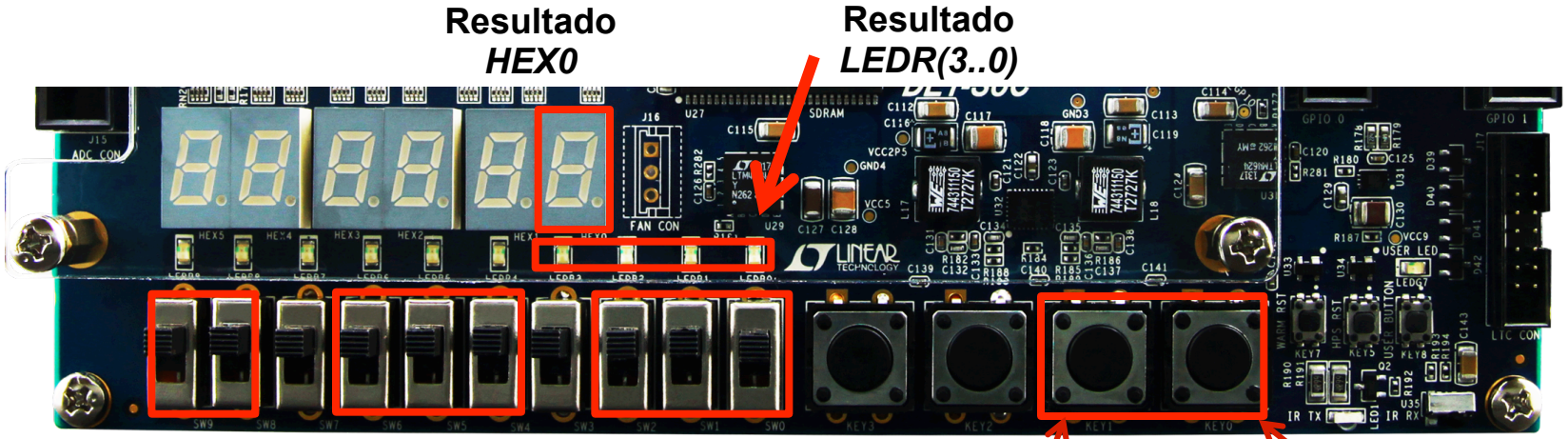
5.- Implementação de deslocador esquerda em VHDL

Tarefa a ser realizada:

Circuito aritmético registrado com armazenamento dos resultados das operações.

Tarefa

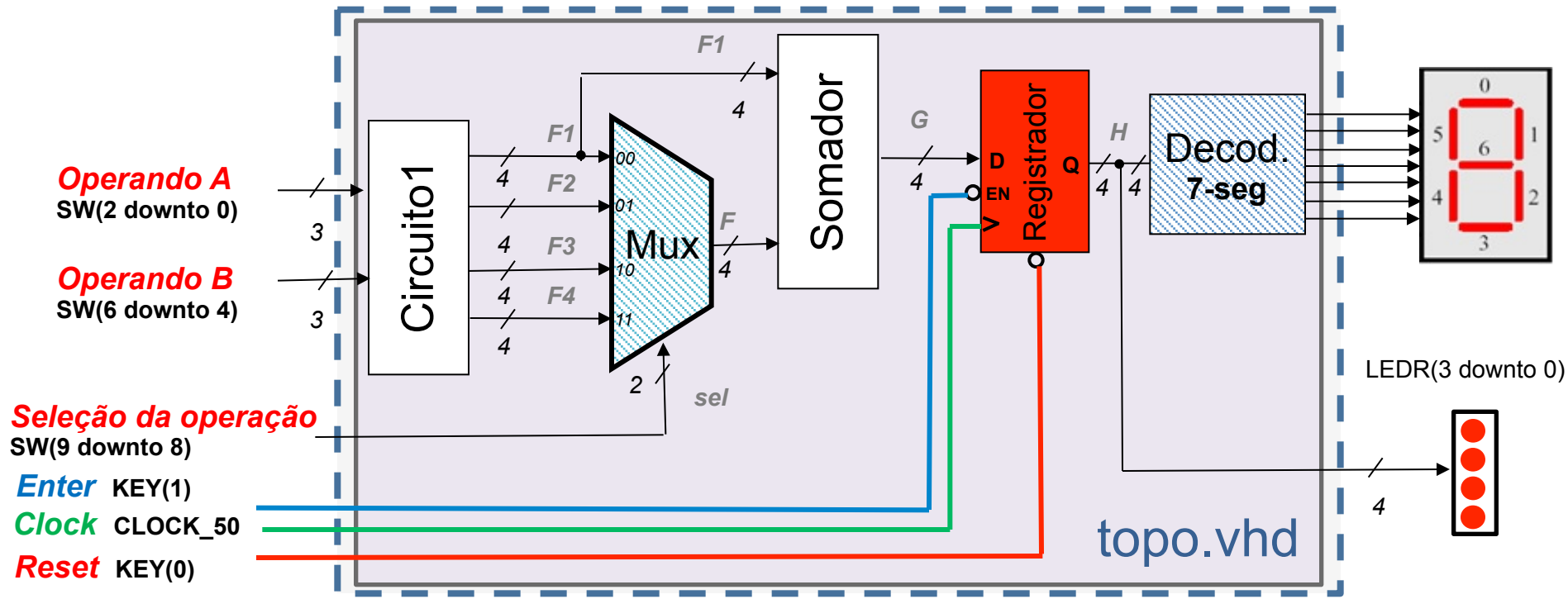
- O circuito consiste no circuito aritmético com um registrador para armazenar os resultados das operações.



Seletor SW(9..8)	Saída (LEDR e display 7-seg)
00	2A
01	3A
10	A+B
11	A+2B

Tarefa

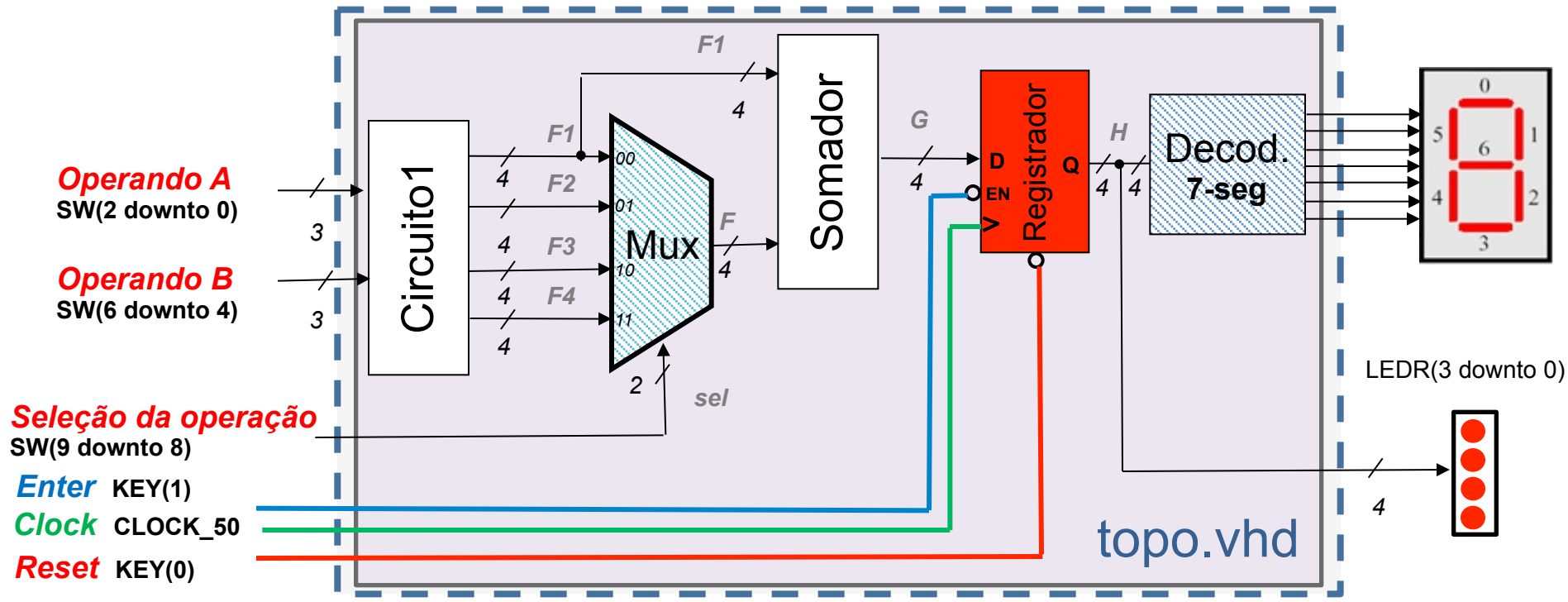
- O circuito vai ser implementado usando um **multiplexador**, um **somador** e um **decoder** como mostrado na figura. Todos estes circuitos são dados no **Moodle**.



Tarefa

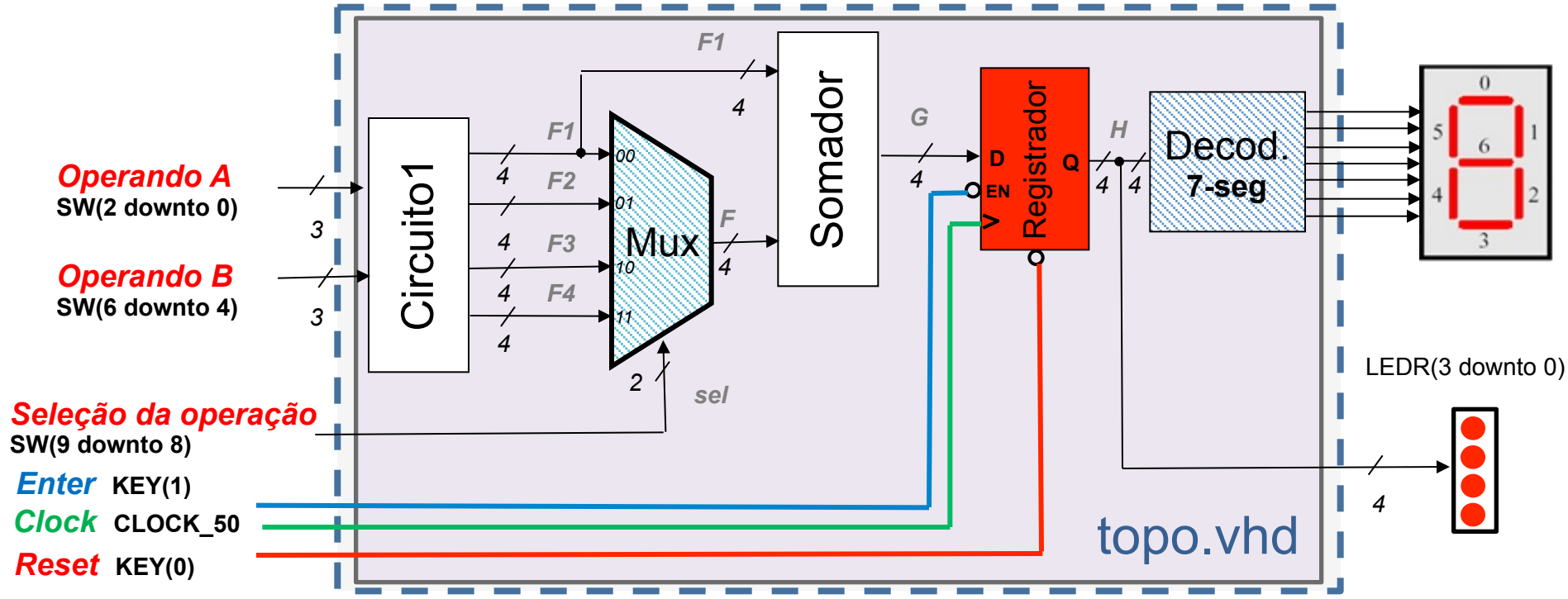
- Implementar o bloco denominado **Circuito1** de acordo as seguintes linhas de código no **topo.vhd**:

```
F1 <= '0' & A;
F2 <= A & '0';
F3 <= '0' & B;
F4 <= B & '0';
```



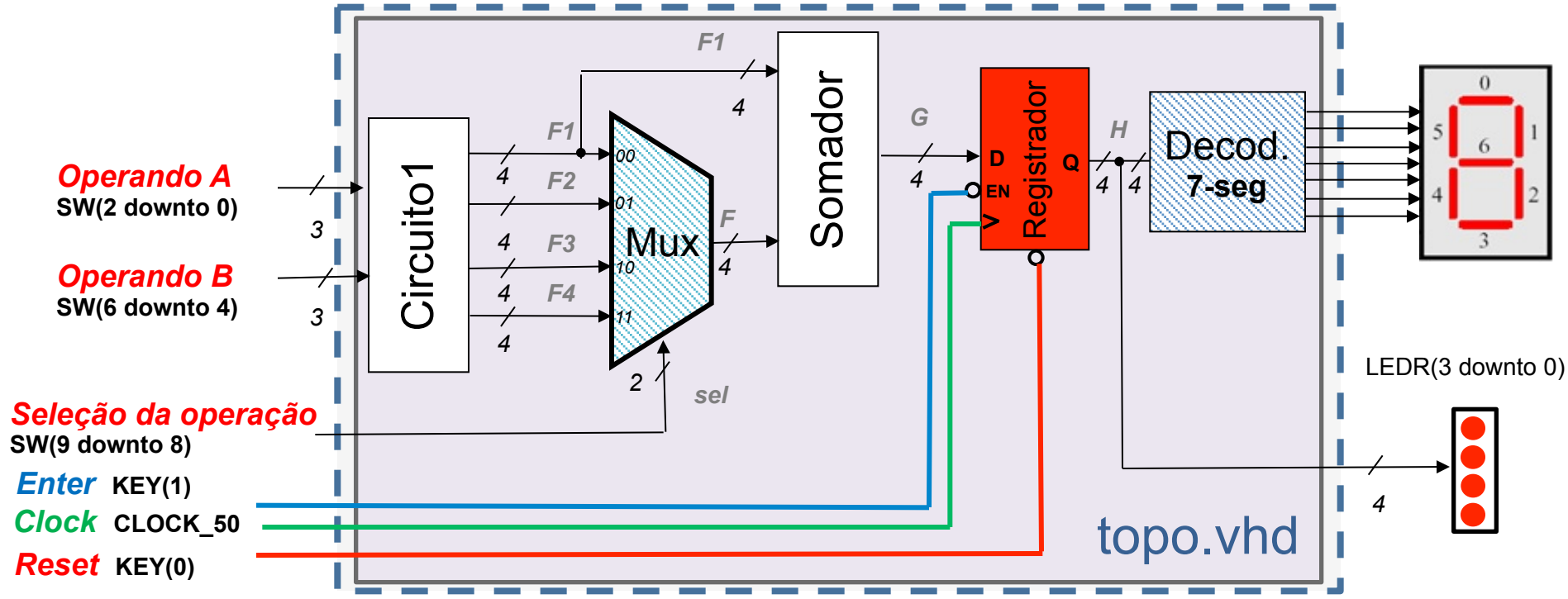
Tarefa

- Um **registrador de 4 bits** para armazenar o resultado a ser apresentado em **HEX0** e nos LEDRs.
- No slide 9 é apresentado um registrador de 4 bits, implementado com 4 ***flip-flops***, porém sem **Enable**.
- Utilizar os circuitos dos **slides 7 e 8** como base para para escrever o VHDL do registrador de 4 bits solicitados no exercício, com **Reset** e **Enable**.



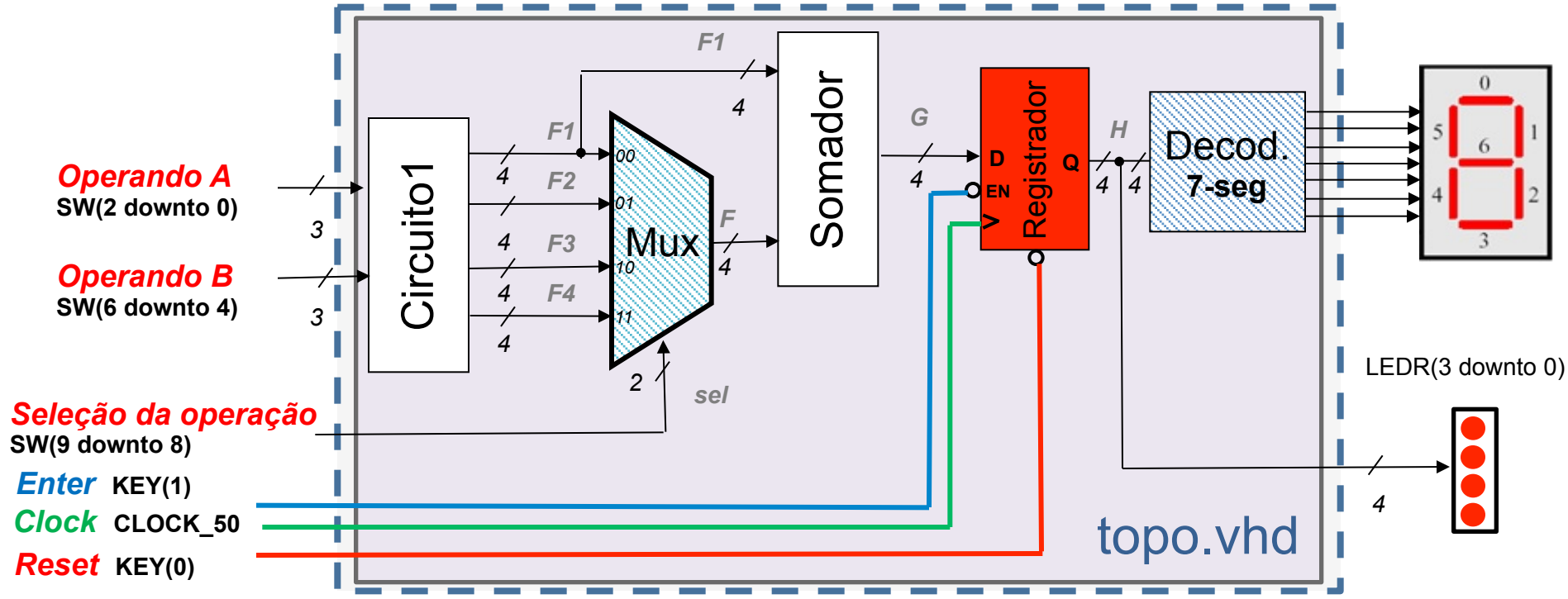
Tarefa

- Ao se pressionar o **botão KEY(0)** – "**Reset** ", os flip-flops deverão ser "limpos", apagando os LEDs, e apresentando o valor 0 (zero) no display 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").



Tarefa

- O sinal de relógio (Clock ou CLK) do circuito deve ser obtido diretamente da placa, utilizando o sinal **CLOCK_50** (PIN_AF14).
- Os push buttons (KEY), **quando pressionados**, fornecem '0'.
- Na **entity do novo topo**, devem ser adicionados os sinais **key** (2 bits) e **clock_50** (1 bit).

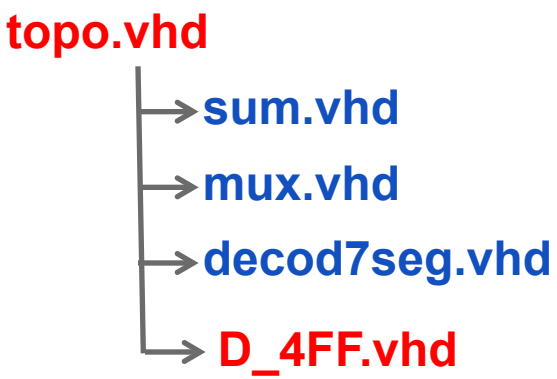


Tarefa

- **Passo 1**: Criar projeto no **Quartus II (versão 15.0)**
  - Acessar **File -> New Project Wizard** e criar um projeto com as seguintes características:
    - **Pasta**: criar uma pasta em ***Desktop***
    - **Sugestão de nome do Projeto**: **Topo**
    - **Tipo**: *Empty Project*
    - Em ***Add Files***, não adicionar nenhum arquivo
    - Na escolha do dispositivo:
      - Família: **Cyclone V**
      - Dispositivo: **5CSEMA5F31C6**
    - Demais opções deixar como padrão

Tarefa

- **Passo 2**: Criar arquivo do tipo **VHDL** dentro do projeto
  - No **Quartus II**, acessar
    File -> New -> Design Files -> VHDL File
    e o arquivo "***Vhdl1.vhd***" será criado:
  - Em seguida, clique em **File -> Save As** e salve seu arquivo com algum nome desejado (sugestão: "***topo.vhd***").
- **Passo 3**: **Crie** um projeto **VHDL** que implemente o circuito apresentado no slide 17 com a seguinte hierarquia:



- **Passo 4**: Realizar a **simulação** do circuito por intermédio de diagramas de formas de onda, verificando o funcionamento correcto para as 4 operações possiveis.

Tarefa

- **Passo 5:** Associar as entradas e saídas ao **FPGA** de tal forma que as entrada dos dados **A** e **B** a partir das chaves **SW(2 downto 0)** e **SW(6 downto 4)**, **Control** no **SW(9 downto 8),** e a apresentação dos resultados **S** na saída **LEDR(3 downto 0)**, botão enter em **Key(1)**, botão reset em **Key(0)** e display **HEX0.**

- **Passo 6**: Fazer associação de **entradas/saídas** com **chaves/leds** do DE1-SOC usando arquivo externo com ***assignments***

  - Baixe o arquivo **Pinos.qsf** disponível no Moodle
    - Esse arquivo contém a descrição de relações entre os nomes que você usou no seu projeto reformulado [**SW, LEDR, etc**] e os **pinos** do **FPGA**.
  - No **Quartus II**, acesse **Assignments -> Import Assignments...** e faça a importação de **Pinos.qsf**.

- **Passo 7: Testar** o circuito no kit **DE1-SoC**, usando as chaves **SW(6 downto 4)** e **SW(2 downto 0)** para entrar com os operandos e sinal de seleção **SW(9 downto 8)**, botões **Key(1)** e **Key(0)** como enter e reset e observar os resultados nos **LEDs**, e Display **HEX0**.