



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



# **“EEL5105 - Circuitos e Técnicas Digitais”**

**Prof. Héctor Pettenghi Roldán\***

[Hector@eel.ufsc.br](mailto:Hector@eel.ufsc.br)

**Florianópolis, março de 2017.**

**\*Baseados nos slides do Professor Eduardo Bezerra e Eduardo Batista EEL5105 2015.2**

*Uso de FSMs no controle do fluxo  
de execução de sistemas digitais.*

*Estudo de caso: Projeto de Calculadora.*

# Objetivos do laboratório

---

1. Entender o uso de FSMs como estrutura de controle do fluxo de operações de um circuito combinacional.
2. Implementar uma FSM em VHDL para controlar as operações da calculadora desenvolvida nas aulas anteriores.

## **Tarefa a ser realizada na aula prática**

# Definição do problema: calculadora com entrada de dados reduzida

---

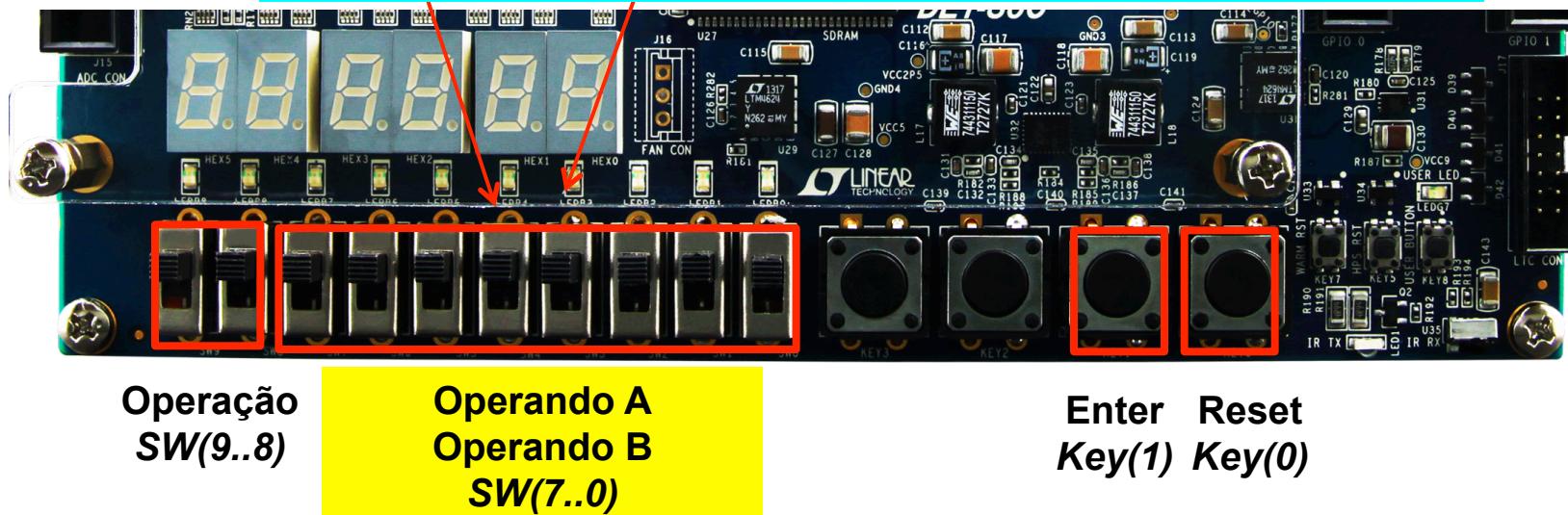
- Para realizar as operações, alguns componentes da calculadora desenvolvida nas aulas anteriores utilizam:
  - as chaves **SW(3..0)** para leitura do **operando A**, e
  - as chaves **SW(7..4)** para leitura do **operando B**.
- Com isso foi possível implementar uma calculadora que realiza operações com operandos com tamanho de 4 bits.
- Visando implementar uma calculadora que realiza operações com operandos de 8 bits, e considerando que a placa disponível no laboratório possui 10 chaves, solicita-se utilizar **apenas um conjunto** com 8 chaves, **SW(7..0)**, para **leitura dos dois operandos**.
- A seleção da operação desejada continua sendo realizada pelas chaves **SW(9..8)**.

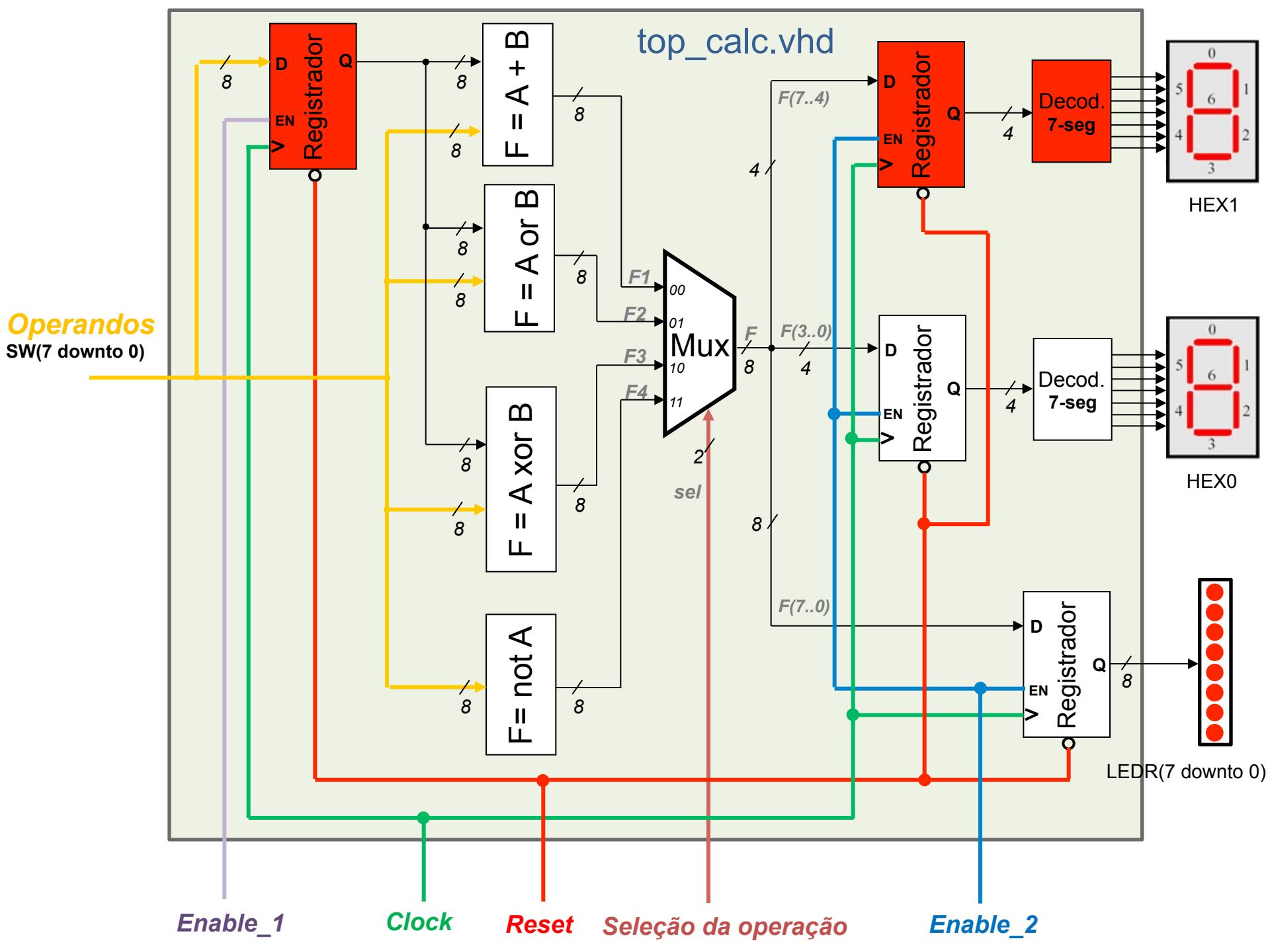
# Entrada de dados para realizar uma soma

Entrada de dados na calculadora original (**aulas anteriores**):



Entrada de dados na **nova calculadora modificada**:





# FSM para controle do fluxo de operações da calculadora

- No slide a seguir é apresentado o diagrama de blocos do circuito, incluindo o controlador (FSM) para gerenciar o fluxo de operações.
- Sugestão de interface para o novo componente (FSM):

```
component FSMctrl
  port (
      Clk, Rst, Enter : IN STD_LOGIC;
      Operacao : IN STD_LOGIC_VECTOR(1 downto 0);
      Selecao : OUT STD_LOGIC_VECTOR(1 downto 0);
      Enable_1, Enable_2 : OUT STD_LOGIC
  );
end component;
```

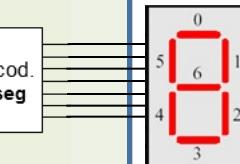
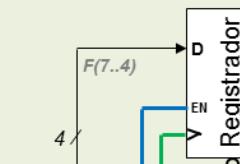
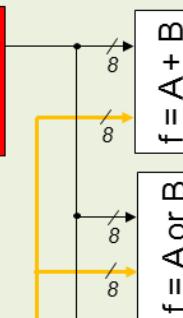
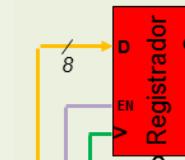
- O top\_calc.vhd passará a ter 12 componentes instanciados! (9 “Components” no total).

# TOPO: SUGESTÃO I

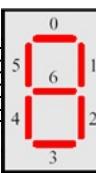
top\_calc.vhd

**Operandos**  
SW(7 downto 0)

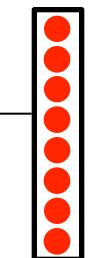
8



HEX1



HEX0



LEDR  
(7 downto 0)

Enable\_1

Seleção

Enable\_2

**FSM**

*Máquina de estados com a função de “controlador” do fluxo de operações realizadas pela calculadora.*

Reset KEY(0)



Clock CLOCK\_50



Enter KEY(1)



Operação SW(9..8)



Rst

Clk

Enter

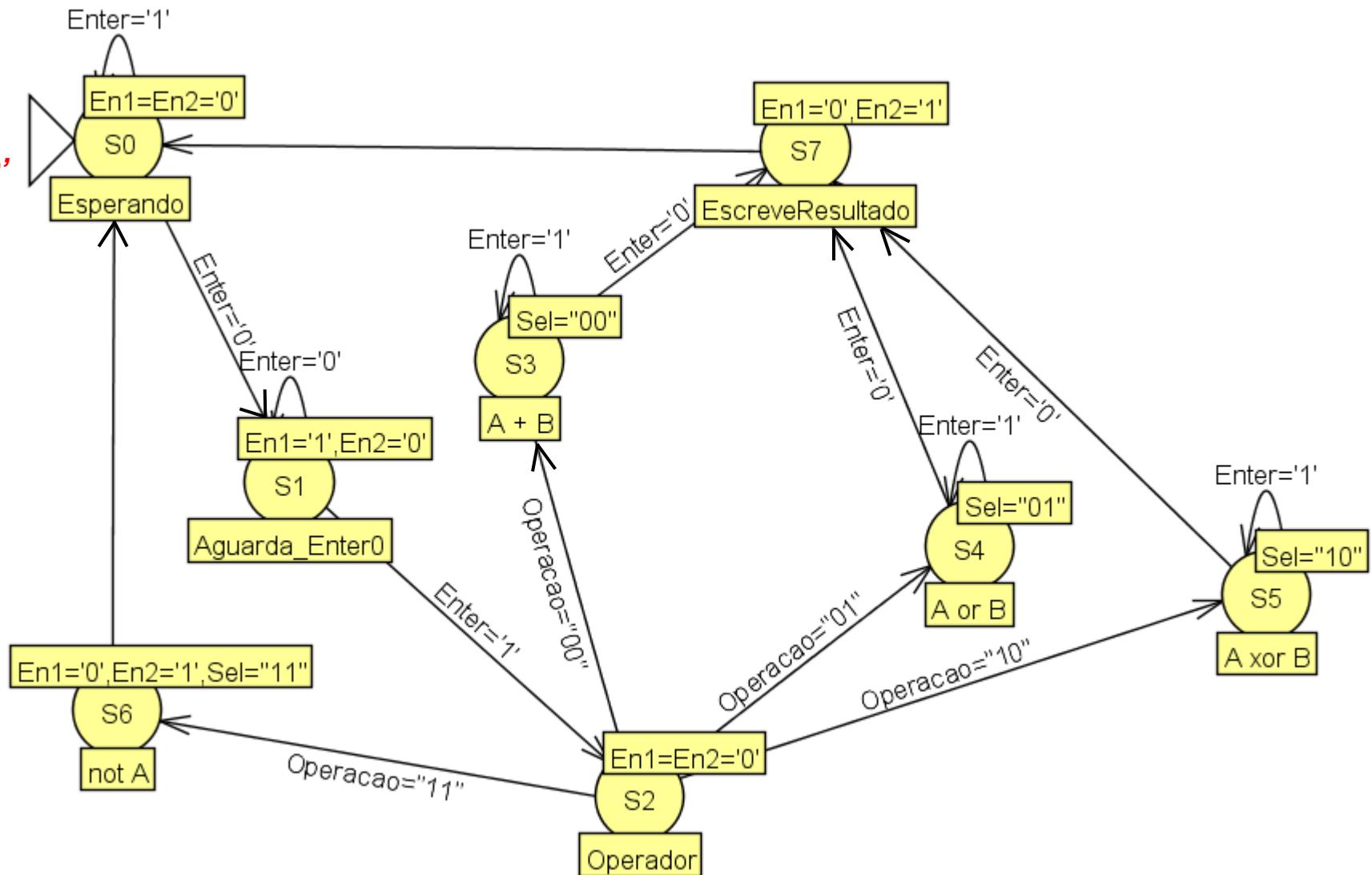
Operacao

# Passo 1: Descrição Funcional

- No estado inicial, “Esperando”, as saídas da FSM estão desabilitadas ( $Enable\_1='0'$ ,  $Enable\_2='0'$ ,  $Selecao="XX"$ ).
- Isso garante que nenhuma atividade ocorrerá na calculadora, enquanto o usuário não fornecer os operandos e a operação.
- Quando  $Enter$  ( $Key(1)$ ) for ‘0’, a FSM avança para o próximo estado, para aguardar até o  $Enter$  voltar para ‘1’ (botão “não pressionado”).
- No estado “Operação” é realizada uma transição para o próximo estado, de acordo com a operação selecionada.
- Se a operação for “not A” (apenas um operando), a operação é realizada, o resultado é armazenado no registrador de saída ( $Enable\_2='1'$ ) e a FSM retorna para o estado inicial “Esperando”.
- Para as demais operações, a leitura do segundo operando é realizada em um estado adicional -  $Enable\_1$  e  $Enable\_2$  precisam ser devidamente controlados em cada estado.
- Para essas operações, quando o  $Enter$  for pressionado pela segunda vez ( $Key(1) = '0'$ ), o resultado é escrito nos registradores de saída ( $Enable\_1='1'$ ), e a FSM retorna para o estado inicial “Esperando”.

# Passo 2: Diagrama de Estados

**Estado Inicial  
 $Rst = '0'$**



### Passo 3: Tabela de transição de estados

Entradas				Saídas			
Reset	Enter	Operação	EA	PE	Enable_1	Seleção	Enable_2
0	X	XX	S0	S0	0	00	0
1	1	XX	S0	S0	0	00	0
1	0	XX	S0	S1	0	00	0
1	0	XX	S1	S1	1	00	0
1	1	XX	S1	S2	1	00	0
1	X	00	S2	S3	0	00	0
1	X	01	S2	S4	0	00	0
1	X	10	S2	S5	0	00	0
1	X	11	S2	S6	0	00	0
1	0	XX	S3	S7	0	00	0
1	1	XX	S3	S3	0	00	0
1	0	XX	S4	S7	0	01	0
1	1	XX	S4	S4	0	01	0
1	0	XX	S5	S7	0	10	0
1	1	XX	S5	S5	0	10	0
1	X	XX	S6	S0	0	11	1
1	X	XX	S7	S0	0	00	1

## Passo 4: Descrição VHDL

```
library ieee; use ieee.std_logic_1164.all;
entity FSMctrl is
port ( Clk, Rst, Enter : in std_logic;
       Operacao: in std_logic_vector(1 downto 0);
       Selecao: out std_logic_vector(1 downto 0);
       Enable_1, Enable_2: out std_logic
);
end FSMctr;
architecture FSM_beh of FSMctrl is
type states is (S0, S1, S2, S3, S4, S5,
S6);
signal EA, PE: states;
signal clock: std_logic;
signal reset: std_logic;
begin
  clock <= Clk;
P1: process (clock, reset)
begin
  if reset = '0' then
    EA <= S0;
  elsif clock'event and clock = '1' then
    EA <= PE;
  end if;
end process;
```

```
P2: process (EA, Enter, Operacao)
begin
  case EA is
    when S0 =>
      if Enter = '1' then
        PE <= S0;
      else
        PE <= S1;
      end if;
      Enable_1 <= '0';
      Enable_2 <= '0';
    when S1 => ... (a fazer)
    when S2 => -- Operador
      Enable_1 <= '0';
      Enable_2 <= '0';
      if Operacao = "00" then
        PE <= S3; -- Fazer soma
      elsif Operacao = "01" then
        PE <= S4; -- Fazer OR
      elsif ... (a fazer)
      end case;
    end process;
end FSM_beh; -- fim da architecture
```

↓ *Essa architecture continua na coluna da direita ...*

# Dicas para teste no simulador *Modelsim* e validação na placa DE1

---

1. Selecionar a operação desejada nas chaves SW(9..8).
2. Fornecer um valor nas chaves SW(7..0) - operando A.
3. Pressionar *Enter* - o botão KEY(1) será ‘0’ quando pressionado.
4. Se for a operação “*not A*”, o resultado será apresentado nos displays de 7-segmentos e LEDs.
5. Se for operação de *soma*, *xor* ou *or*, fornecer o segundo operando nas chaves SW(7..0), e pressionar *Enter*.
6. Após apresentado o resultado, essa sequência é reiniciada, voltando ao passo 1.