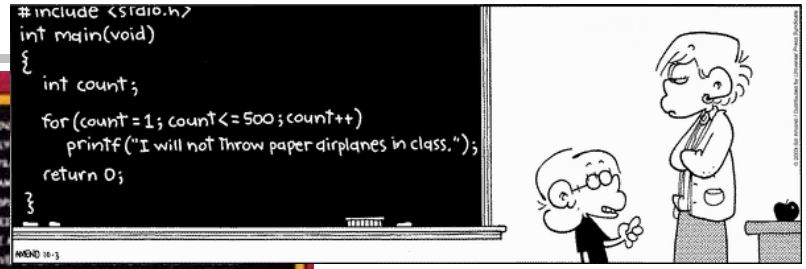
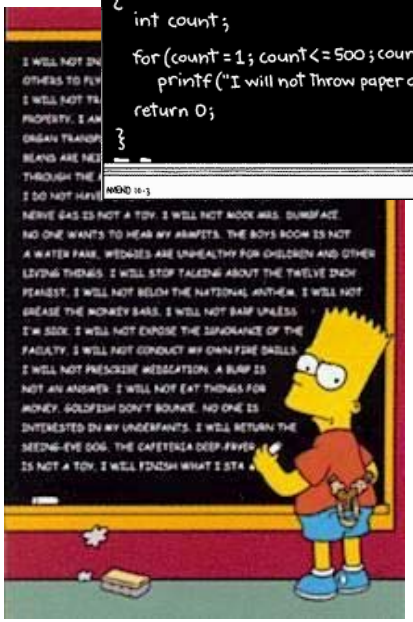
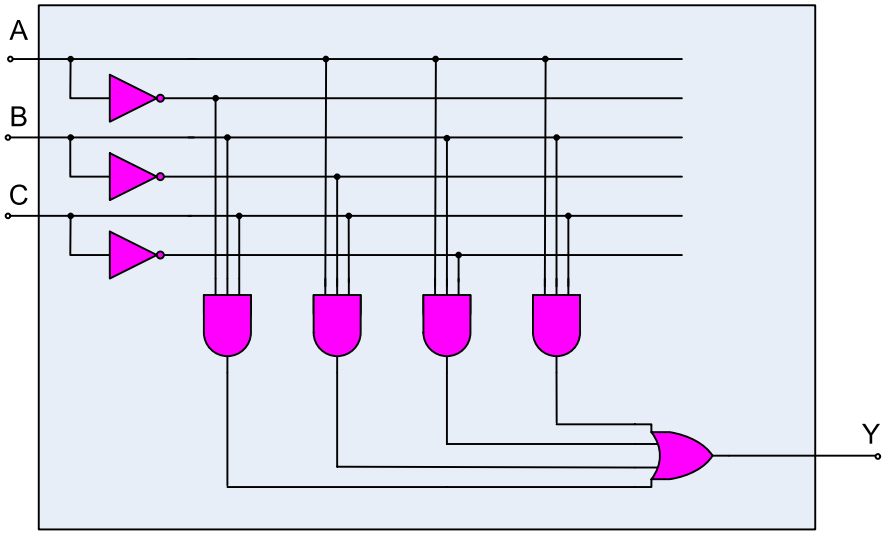


The new way!



The old way...

O “nosso” circuito votador graficamente



Introdução ao VHDL para Síntese de Sistemas Digitais

Paulo Marques

pmarques@dei.uc.pt
http://www.dei.uc.pt/~pmarques

Tecnologia dos Computadores 2004/2005

O “nosso” circuito votador em VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY votador IS
        PORT(A, B, C:  IN      STD_LOGIC;
             Y:        OUT     STD_LOGIC);
END votador;

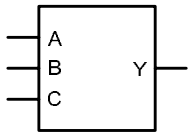
ARCHITECTURE eq OF votador IS
BEGIN
  Y <= '1' WHEN (A='1' AND B='1') ELSE
       '1' WHEN (A='1' AND C='1') ELSE
       '1' WHEN (B='1' AND C='1') ELSE
       '0';
END eq;
```

VHDL

- VHDL = VHSIC HARDWARE DESCRIPTION LANGUAGE
  (VHSIC = VERY HIGH SPEED INTEGRATED CIRCUIT)
- Trata-se de uma linguagem para...
  - Modelar circuitos integrados (sistemas digitais)
  - Simular circuitos
  - Programá-los em *hardware* (síntese)
- É um standard do IEEE e muito utilizado na indústria
  (IEEE = Institute of Electrical and Electronics Engineers)

Declaração de Circuitos (ENTITY)

```
ENTITY votador IS
        PORT(A, B, C: IN   STD_LOGIC;
             Y:       OUT  STD_LOGIC);
END votador;
```

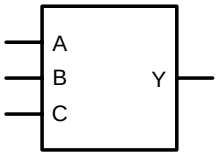


- Define o circuito em termos de entradas e saídas
- IN → Linhas de entrada de dados
- OUT → Linhas de saída de dados
- STD_LOGIC → Representa um *bit*
  - *Na verdade, representa mais informação*
  - *('0', '1', 'Z', 'X', etc.)*

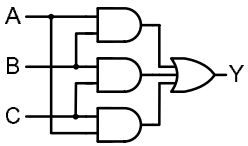
Estrutura de um ficheiro VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

```
ENTITY votador IS
        PORT(A, B, C: IN   STD_LOGIC;
             Y:       OUT  STD_LOGIC);
END votador;
```

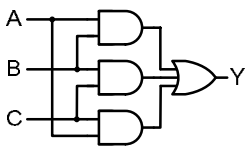



```
ARCHITECTURE eq OF votador IS
BEGIN
  Y <= (A AND B) OR (A AND C) OR (B AND C);
END eq;
```



Arquitectura de um circuito (ARCHITECTURE)

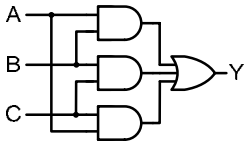
```
ARCHITECTURE arch OF votador IS
BEGIN
  Y <= (A AND B) OR (A AND C) OR (B AND C);
END arch;
```



- Diz como é que o circuito funciona por dentro
- O operador <= significa "*toma o valor de*"
- Podem-se utilizar diversos operadores
  - AND, OR, NOT, XOR, +, -
- Atribuições lógicas a um *bit* são feitas entre plicas. Por exemplo: Y <= '1';

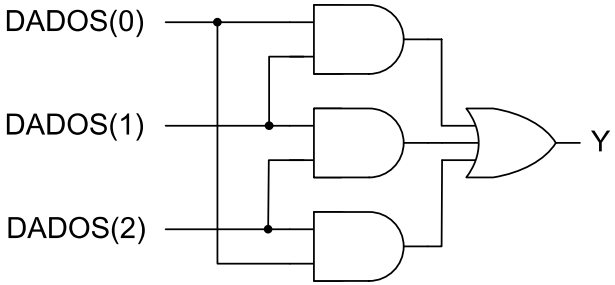
Declaração de sinais

```
ARCHITECTURE arch OF votador IS
  SIGNAL COND1, COND2, COND3 : STD_LOGIC;
BEGIN
  COND1 <= A AND B;
  COND2 <= A AND C;
  COND3 <= B AND C;
  Y     <= COND1 OR COND2 OR COND3;
END arch;
```

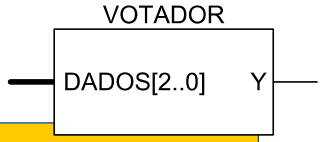


- Na arquitectura de um circuito podem-se declarar sinais internos, auxiliares
- É de notar que as atribuições NÃO SÃO FEITAS por ordem. Tudo ocorre simultaneamente.
  - Estamos a descrever um circuito digital, não a implementar um programa!


```
ARCHITECTURE arch OF votador IS
BEGIN
  Y <= (  DADOS(0) AND DADOS(1)  )   OR
       (  DADOS(0) AND DADOS(2)  )   OR
       (  DADOS(1) AND DADOS(2)  );
END arch;
```



BUS




```
ENTITY votador IS
        PORT(DADOS: IN   STD_LOGIC_VECTOR(2 DOWNTO 0);
             Y:     OUT  STD_LOGIC);
END votador;
```

- Também é possível declarar um agrupamento de fios (BUS): STD_LOGIC_VECTOR
- Os elementos são acedidos utilizando o nome do *bus* e o índice correspondente:
  - DADOS(0), DADOS(1) e DADOS(2)
  - Atribuições lógicas a um agrupamento são feitas entre aspas.
    Por exemplo: DADOS <= “101”;

Circuitos combinacionais simples

- Utilização de equações directas
  - ```
    Y  <= ( D(0) AND D(1) )  OR
          ( D(0) AND D(2) )  OR
          ( D(1) AND D(2) );
    ```
- *Conditional Signal Assigment*
  - ```
    Y  <= ‘1’ WHEN (D(0)=‘1’ AND D(1)=‘1’) ELSE
          ‘1’ WHEN (D(0)=‘1’ AND D(2)=‘1’) ELSE
          ‘1’ WHEN (D(1)=‘1’ AND D(2)=‘1’) ELSE
          ‘0’;
    ```
- *Selected Signal Assigment*
  - ```
    WITH D SELECT
          Y  <= ‘1’ WHEN “110”,
                ‘1’ WHEN “101”,
                ‘1’ WHEN “011”,
                ‘1’ WHEN “111”,
                ‘0’ WHEN OTHERS;
    ```

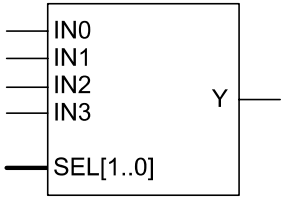
(Nota: Para poupar espaço abreviou-se “DADOS” para “D”)
    ```

Alguns exemplos...

- Multiplexador de dados
- Descodificador BCD para 7-segmentos
- Somador de 4 bits

Multiplexador 4x1 - Arquitectura

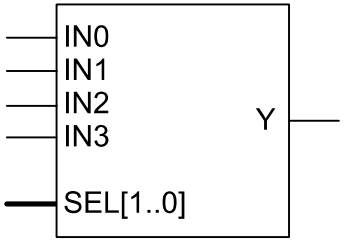
```
ARCHITECTURE arch OF multiplexer4x1 IS
BEGIN
    WITH SEL SELECT
      Y <= IN0      WHEN "00",
           IN1      WHEN "01",
           IN2      WHEN "10",
           IN3      WHEN "11",
           '0'      WHEN OTHERS;
END arch;
```



SEL[1]	SEL[0]	Y
0	0	IN0
0	1	IN1
1	0	IN2
1	1	IN3

Multiplexador 4x1

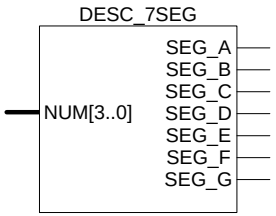
```
ENTITY multiplexer4x1 IS
  PORT(
    IN0, IN1, IN2, IN3:  IN  STD_LOGIC;
    SEL:                 IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
    Y:                   OUT STD_LOGIC);
END multiplexer4x1;
```

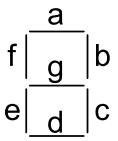


SEL[1]	SEL[0]	Y
0	0	IN0
0	1	IN1
1	0	IN2
1	1	IN3

Descodificador BCD para 7-segmentos

```
ENTITY desc_7seg IS
        PORT(NUM:    IN   STD_LOGIC_VECTOR(3 DOWNTO 0);
             SEG_A: OUT  STD_LOGIC;
             SEG_B: OUT  STD_LOGIC;
             SEG_C: OUT  STD_LOGIC;
             SEG_D: OUT  STD_LOGIC;
             SEG_E: OUT  STD_LOGIC;
             SEG_F: OUT  STD_LOGIC;
             SEG_G: OUT  STD_LOGIC);
END desc_7seg;
```

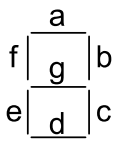




Arquitectura...

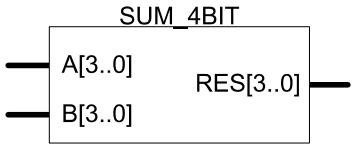
```
ARCHITECTURE arch OF desc_7seg IS
    SIGNAL OUTPUT: STD_LOGIC_VECTOR(6 DOWNTO 0);
BEGIN
    WITH NUM SELECT
       OUTPUT <= "1111110" WHEN "0000",        -- 0
                 "0110000" WHEN "0001",        -- 1
                 "1101101" WHEN "0010",        -- 2
                 "1111001" WHEN "0011",        -- 3
                 "0110011" WHEN "0100",        -- 4
                 "1011011" WHEN "0101",        -- 5
                 "1011111" WHEN "0110",        -- 6
                 "1110000" WHEN "0111",        -- 7
                 "1111111" WHEN "1000",        -- 8
                 "1111011" WHEN "1001",        -- 9
                 "0000000" WHEN OTHERS;        --

    SEG_A <= NOT OUTPUT(6);
    SEG_B <= NOT OUTPUT(5);
    SEG_C <= NOT OUTPUT(4);
    SEG_D <= NOT OUTPUT(3);
    SEG_E <= NOT OUTPUT(2);
    SEG_F <= NOT OUTPUT(1);
    SEG_G <= NOT OUTPUT(0);
END arch;
```



Somador de 4 bits - Arquitectura

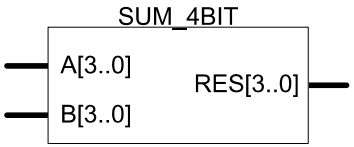
```
ARCHITECTURE arch OF sum_4bit IS
BEGIN
  RES <= A + B;
END arch;
```



Somador de 4 bits

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY sum_4bit IS
    PORT(A:      IN   STD_LOGIC_VECTOR(3 DOWNTO 0);
         B:      IN   STD_LOGIC_VECTOR(3 DOWNTO 0);
         RES:    OUT  STD_LOGIC_VECTOR(3 DOWNTO 0));
END sum_4bit;
```

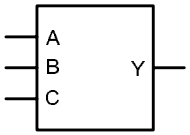


O circuito votador revisitado...

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY votador IS
        PORT(A, B, C:  IN      STD_LOGIC;
             Y:        OUT     STD_LOGIC);
END votador;

ARCHITECTURE eq OF votador IS
BEGIN
  Y <= '1' WHEN (A='1' AND B='1') ELSE
       '1' WHEN (A='1' AND C='1') ELSE
       '1' WHEN (B='1' AND C='1') ELSE
       '0';
END eq;
```



Exercícios

- Escreva em VHDL um circuito votador com:
  - 4 entradas
  - 3 saídas (MAIORIA_1, EMPATE, MAIORIA_0)
- Escreva em VHDL um circuito com 4 entradas e uma saída em *bus*. Na saída 7-segmentos é mostrado o número de entradas que se encontram a 1.
- Escreva em VHDL um circuito que dados dois números de quatro bits à entrada (A e B), possua três saídas que indicam se A<B, A=B ou A>B.

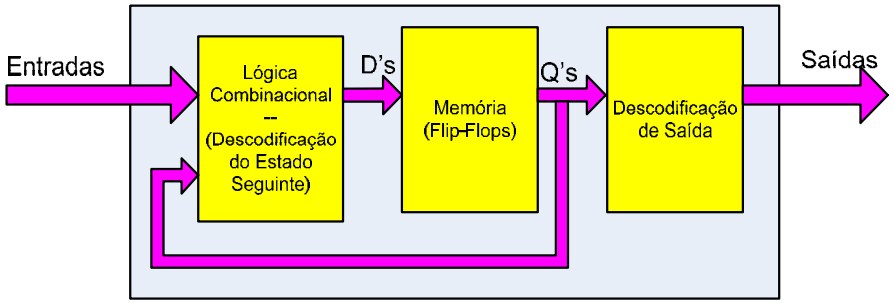
VHDL para Síntese de Circuitos Sequenciais

Circuitos Sequenciais

- Circuitos que necessitam de memória (i.e. Flip-flops)
- Em cada vertente de relógio, o circuito evolui para uma nova situação. A evolução depende de:
  - Qual o estado em que se encontra a memória
  - Qual o valor das entradas nesse momento
- O valor das saídas depende tipicamente do estado presente, embora também possa depender das entradas
- Os circuitos sequenciais são descritos através de uma <u>máquina de estados</u>

Arquitectura de Máquinas de Estados

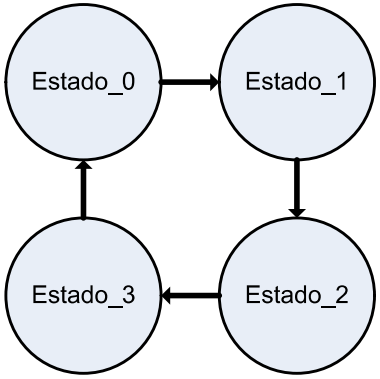
- Só utilizaremos um tipo de arquitectura...
  - Máquinas de Moore!



Elementos necessários

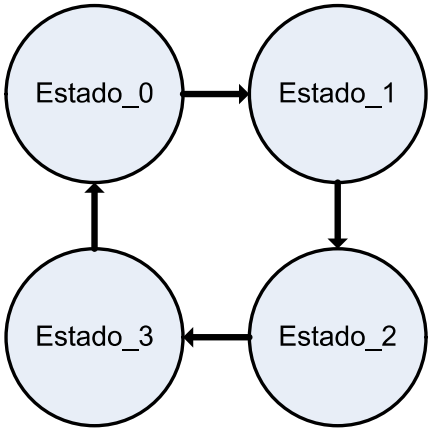
- Diagrama de transição de estados
- Tabela de saídas dado o estado corrente

Estado	OUT[1]	OUT[0]
ESTADO_0	0	0
ESTADO_1	0	1
ESTADO_2	1	0
ESTADO_3	1	1



Um circuito sequêncial simples...

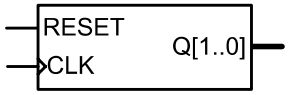
- Contador binário de 0-3
  - 4 números → 4 estados → 2 bits → 2 flip-flops



Declaração do Circuito

```
LIBRARY ieee;
USE     ieee.std_logic_1164.ALL;

ENTITY count2 IS
  PORT(CLK:         IN     STD_LOGIC;
       RESET:       IN     STD_LOGIC;
       Q:           OUT    STD_LOGIC_VECTOR(1 DOWNTO 0));
END count2;
```



Arquitectura do Circuito

```
ARCHITECTURE arch OF count2 IS
  TYPE    ESTADO IS (estado_0, estado_1, estado_2, estado_3);
  SIGNAL  ESTADO_CORRENTE: ESTADO;
BEGIN
  PROCESS(RESET, CLK)
  BEGIN
    IF (RESET='1') THEN
      ESTADO_CORRENTE <= estado_0;
    ELSIF (CLK'EVENT AND CLK='1') THEN
      CASE ESTADO_CORRENTE IS
        WHEN estado_0 =>    ESTADO_CORRENTE <= estado_1;
        WHEN estado_1 =>    ESTADO_CORRENTE <= estado_2;
        WHEN estado_2 =>    ESTADO_CORRENTE <= estado_3;
        WHEN estado_3 =>    ESTADO_CORRENTE <= estado_0;
      END CASE;
    END IF;
  END PROCESS;

  WITH ESTADO_CORRENTE SELECT
    Q <= "00" WHEN estado_0,
         "01" WHEN estado_1,
         "10" WHEN estado_2,
         "11" WHEN estado_3;
END arch;
```


Declaração de Processos

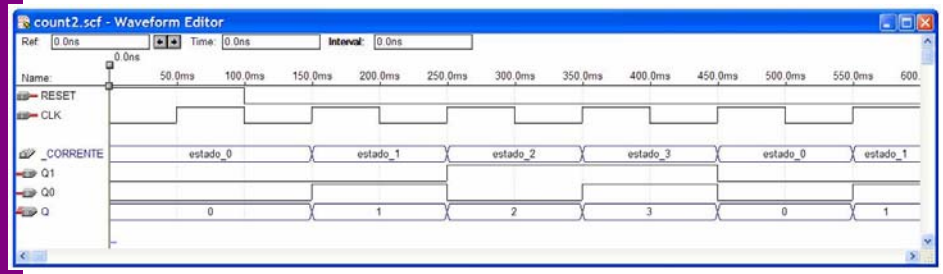
- Um processo representa uma parte do circuito potencialmente sequencial
  - Podem ser gerados flip-flops para guardar memória
  - A lista de sensibilidade indica quais os sinais que levam à "execução" do processo

```
PROCESS(RESET, CLK)
BEGIN
  ...
END PROCESS;
```

  - Ao colocar um IF que espera por um evento de clock está-se implicitamente a gerar flip-flops

```
IF (CLK'EVENT AND CLK='1')
  ...
END IF;
```


Simulação



Descodificação do Estado Seguinte

- Os estados são declarados através da opção TYPE:

```
ARCHITECTURE arch OF count2 IS
  TYPE ESTADO IS (estado_0, estado_1, estado_2, estado_3);
  SIGNAL  ESTADO_CORRENTE: ESTADO;
BEGIN
  ...
END arch;
```

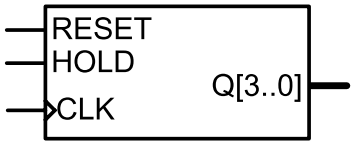
- A construção CASE permite dizer o que acontece no estado seguinte:

```
CASE ESTADO_CORRENTE IS
    WHEN estado_0 =>    ESTADO_CORRENTE <= estado_1;
    WHEN estado_1 =>    ESTADO_CORRENTE <= estado_2;
    ...
END CASE;
```

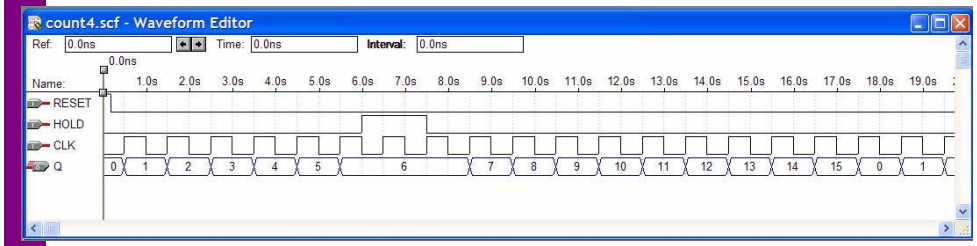
Exemplo: Contador de 4 bits c/ HOLD

```
LIBRARY ieee;
USE     ieee.std_logic_1164.ALL;
USE     ieee.std_logic_unsigned.ALL;
USE     ieee.std_logic_arith.ALL;

ENTITY count4 IS
  PORT(CLK:         IN     STD_LOGIC;
       RESET:       IN     STD_LOGIC;
       HOLD:        IN     STD_LOGIC;
       Q:           OUT    STD_LOGIC_VECTOR(3 DOWNTO 0));
END count4;
```



Simulação



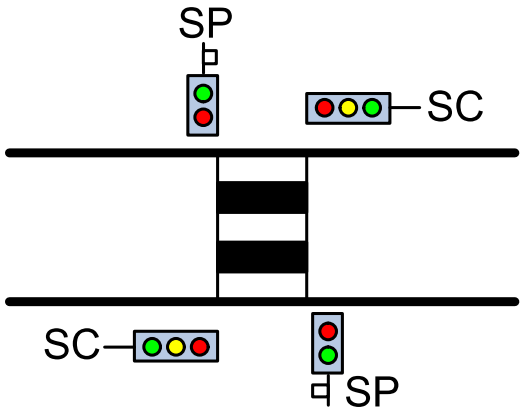
Arquitectura

```
ARCHITECTURE arch OF count4 IS
  SIGNAL COUNT:   STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN

  PROCESS(RESET, CLK)
  BEGIN
    IF (RESET='1') THEN
            COUNT <= "0000";
    ELSIF (CLK'EVENT AND CLK='1') THEN
            IF (HOLD = '0') THEN
                  COUNT <= COUNT + 1;
            ELSE
                  COUNT <= COUNT;
            END IF;
    END IF;
  END PROCESS;

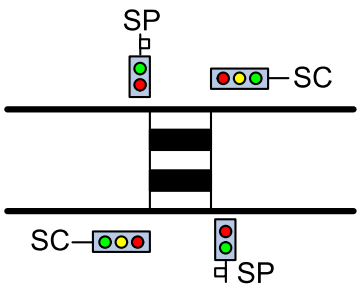
  Q <= COUNT;
END arch;
```

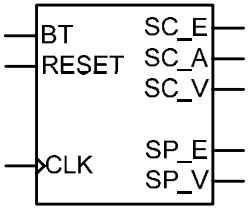

Exemplo: Semáforos numa passadeira



SP = Semáforo dos Peões
SC = Semáforo dos Carros

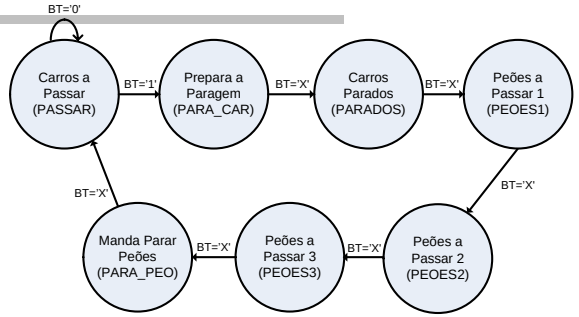
Exemplo: Semáforos numa passadeira





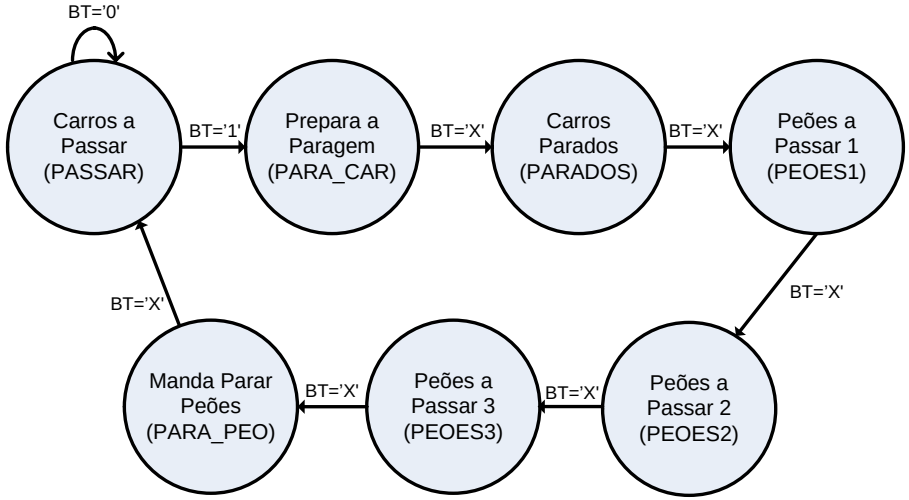
SP = Semáforo dos Peões
SC = Semáforo dos Carros
SC_E / SP_E = Encarnado
SC_A = Amarelo
SC_V / SP_V = Verde
BT = Botão para passar na passadeira

Especificação Completa



Estado	SC_E	SC_A	SC_V	SP_E	SP_V
PASSAR	0	0	1	1	0
A_PARAR	0	1	0	1	0
PARADOS	1	0	0	1	0
PEOES1	1	0	0	0	1
PEOES2	1	0	0	0	1
PEOES3	1	0	0	0	1
PARA_PEO	1	0	0	1	0

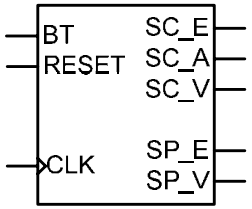
Máquina de Estados



Declaração do Circuito

```
LIBRARY ieee;
USE     ieee.std_logic_1164.ALL;

ENTITY semaf IS
  PORT(CLK:                 IN     STD_LOGIC;
       RESET:               IN     STD_LOGIC;
       BT:                  IN     STD_LOGIC;
       SC_E, SC_A, SC_V:    OUT    STD_LOGIC;
       SP_E, SP_V:          OUT    STD_LOGIC);
END semaf;
```



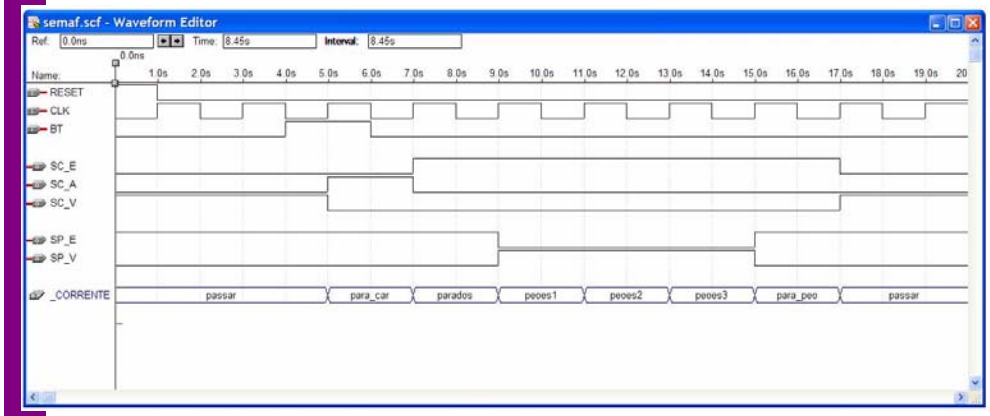
Arquitectura

```
ARCHITECTURE arch OF semaf IS
  TYPE   ESTADO IS
         (passar, para_car, parados, peoes1, peoes2, peoes3, para_peo);
  SIGNAL ESTADO_CORRENTE: ESTADO;
  SIGNAL SAIDAS: STD_LOGIC_VECTOR(4 DOWNTO 0);
BEGIN
  PROCESS(RESET, CLK)
  BEGIN
    IF (RESET='1') THEN
      ESTADO_CORRENTE <= passar;
    ELSIF (CLK'EVENT AND CLK='1') THEN

      CASE ESTADO_CORRENTE IS
        WHEN passar   => IF (BT='1') THEN  ESTADO_CORRENTE <= para_car;
                                   ELSE ESTADO_CORRENTE  <= passar;
                         END IF;
        WHEN para_car => ESTADO_CORRENTE <= parados;
        WHEN parados  => ESTADO_CORRENTE <= peoes1;
        WHEN peoes1   => ESTADO_CORRENTE <= peoes2;
        WHEN peoes2   => ESTADO_CORRENTE <= peoes3;
        WHEN peoes3   => ESTADO_CORRENTE <= para_peo;
        WHEN para_peo => ESTADO_CORRENTE <= passar;
        WHEN OTHERS   => ESTADO_CORRENTE <= passar;
      END CASE;

    END IF;
  END PROCESS;
  -- CONTINUA NO PRÓXIMO SLIDE!
```


Simulação



Arquitectura (2)

```
-- Parte do slide anterior...

  WITH ESTADO_CORRENTE SELECT
    SAIDAS <= "00110" WHEN passar,
              "01010" WHEN para_car,
              "10010" WHEN parados,
              "10001" WHEN peoes1,
              "10001" WHEN peoes2,
              "10001" WHEN peoes3,
              "10010" WHEN para_peo,
              "10010" WHEN OTHERS;

    SC_E <= SAIDAS(4);
    SC_A <= SAIDAS(3);
    SC_V <= SAIDAS(2);
    SP_E <= SAIDAS(1);
    SP_V <= SAIDAS(0);

END arch;
```


Exercícios

- Escreva e simule, em VHDL, os seguintes circuitos:
  - Um contador de 8 bits que conte para cima e para baixo.
  - Um circuito com uma entrada que representa um botão. Quando o botão é carregado, é gerado um valor 1 durante exactamente um pulso de relógio. Independentemente de quanto tempo o botão se mantiver carregado, só volta a ser gerado um novo pulso quando ele for largado e premido novamente.
  - Um circuito que receba um relógio de 1Hz e que tenha como saídas “minutos” e “segundos” em BCD

Para saber mais...

- Digital Design with CPLD Applications and VHDL
  - Capítulo 4 (4.6)
  - Capítulo 5 (5.1, a parte do “VHDL Binary Decoder” e “Seven-Segment Decoders”
  - Capítulo 9 (9.4)
  - Capítulo 10 (10.1, 10.2, 10.3)

- Notas:
  - 1: Ver as anotações do professor, nem tudo o que é mencionado importa saber em profundidade
  - 2: A documentação acima não é especialmente adequada ao nível deste curso. Certifique-se que percebe bem estes apontamentos. Não será exigido mais do que o que aqui é coberto.

THE FIRST MOUSE, USA 1968
(Invented at PALO ALTO, XEROX, by Dr. Douglas Engelbart and his team)