Universidade Federal de Pelotas
Instituto de Física e Matemática
Departamento de Informática
Bacharelado em Ciência da Computação

Técnicas Digitais

Aula 24

Noções da linguagem VHDL: descrição e simulação de latches, flip-flops e registradores.

Profs. José Luís Güntzel & Luciano Agostini

{guntzel,agostini}@ufpel.edu.br

www.ufpel.edu.br/~guntzel/TD/TD.html

Noções de Linguagem VHDL

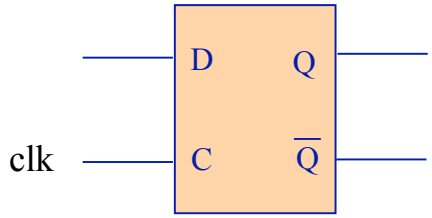
Os Operadores em VHDL

	Operator Class	Operator
Highest precedence	Miscellaneous	**, ABS, NOT
	Multiplying	*, /, MOD, REM
	Sign	+, −
	Adding	+, −, &
	Shift	SLL, SRL, SLA, SRA, ROL, ROR
	Relational	=, /=, <, <=, >, >=
Lowest precedence	Logical	AND, OR, NAND, NOR, XOR, XNOR

Noções de Linguagem VHDL

Latch D (ativado por nível lógico alto)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY latchD IS
PORT ( D, clk :  IN STD_LOGIC;
          Q :       OUT STD_LOGIC);
END latchD;

ARCHITECTURE comportamento OF latchD IS
BEGIN
   PROCESS (D, clk)
   BEGIN
       IF clk = '1' THEN
            Q <= D;
       END IF;
   END PROCESS;
END comportamento;
```

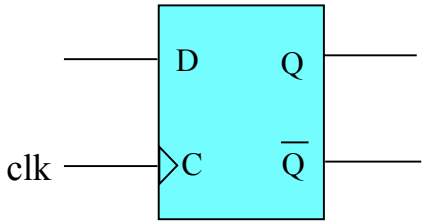


C	D	Q_{t+1}
0	X	Q_t
1	0	0
1	1	1

Noções de Linguagem VHDL

Flip-flop D (disparado pela borda ascendente)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ffD IS
PORT ( D, clk :  IN STD_LOGIC;
          Q :       OUT STD_LOGIC);
END ffD;

ARCHITECTURE comportamento OF ffD IS
BEGIN
   PROCESS (clk)
   BEGIN
      IF clk'EVENT AND clk = '1' THEN
         Q <= D;
      END IF;
   END PROCESS;
END comportamento;
```



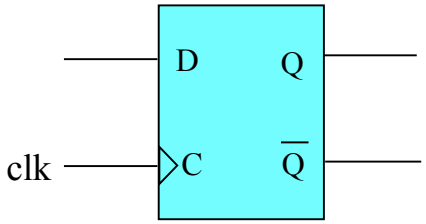
C	D	Q_{t+1}
$\neq\uparrow$	X	Q_t
$\uparrow$	0	0
$\uparrow$	1	1

Atributo: refere-se a troca de nível de clk

Noções de Linguagem VHDL

Flip-flop D (disparado pela borda ascendente) - Versão 2

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ffD IS
PORT ( D, clk :  IN STD_LOGIC;
          Q :       OUT STD_LOGIC);
END ffD;

ARCHITECTURE comportamento OF ffD IS
BEGIN
   PROCESS
   BEGIN
       WAIT UNTIL clk'EVENT AND clk = '1';
       Q <= D;
    END PROCESS;
END comportamento;
```



A lista de sensibilização é omitida

O Uso de WAIT UNTIL especifica implicitamente que somente o relógio faz parte da lista de sensibilização

Noções de Linguagem VHDL

WAIT UNTIL

- **Para efeitos de síntese de circuitos, WAIT UNTIL somente pode ser usado se ele for a primeira atribuição do processo**
- **Na verdade, 'EVENT é redundante no comando WAIT UNTIL e portanto poderíamos simplificar para**

WAIT UNTIL clk='1';

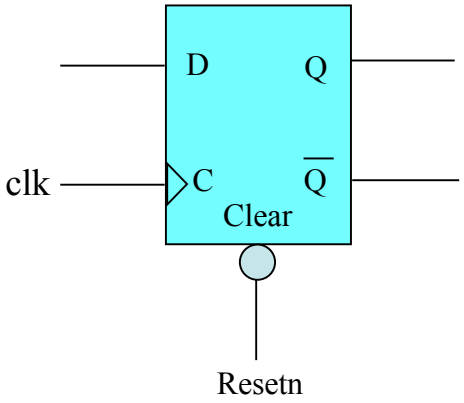
o que se refere ao sinal clk se tornar igual a "1"

- **Entretanto, algumas ferramentas de síntese de circuitos a partir de VHDL exigem a inclusão do atributo "EVENT"**

Noções de Linguagem VHDL

Flip-flop D (com reset assíncrono ativado com lógica negada)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ffD IS
PORT ( D, Resetn, clk :  IN STD_LOGIC;
          Q :                OUT STD_LOGIC);
END ffD;

ARCHITECTURE comportamento OF ffD IS
BEGIN
   PROCESS (Resetn, clk)
   BEGIN
      IF Resetn = '0' THEN
           Q <= '0' ;

      ELSEIF clk'EVENT AND clk = '1' THEN
           Q <= D;
      END IF;
   END PROCESS;
END comportamento;
```

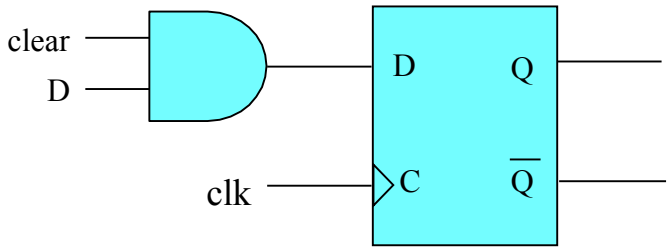


Primeiro testa se o reset assíncrono está ativado (pois é um sinal de mais alta hierarquia)

Noções de Linguagem VHDL

Flip-flop D (com reset síncrono ativado com lógica negada)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ffD IS
PORT ( D, clear, clk :  IN STD_LOGIC;
          Q :                  OUT STD_LOGIC);
END ffD;

ARCHITECTURE comportamento OF ffD IS
BEGIN
   PROCESS (clk)
   BEGIN
      IF clk'EVENT AND clk = '1' THEN
         IF clear = '0' THEN
            Q <= '0' ;
         ELSEIF
            Q <= D;
      END IF;
   END PROCESS;
END comportamento;
```

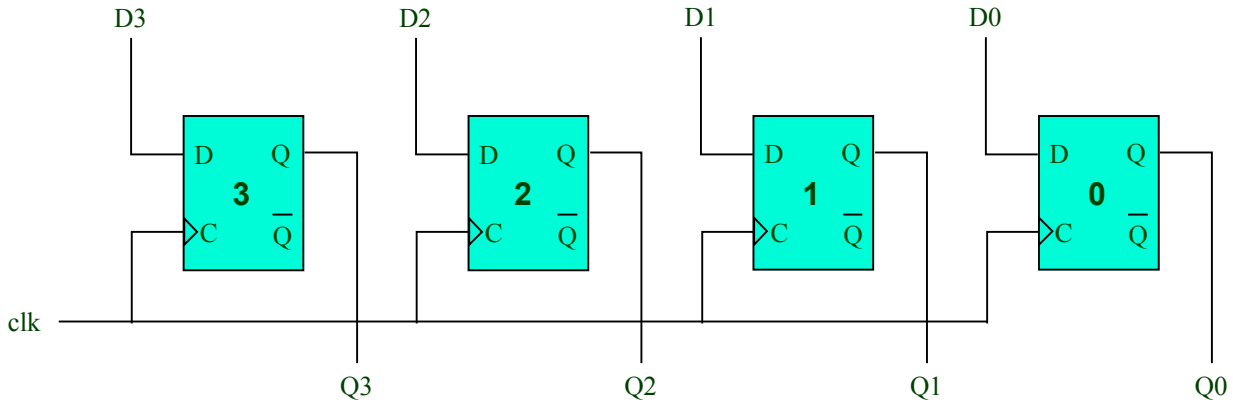


A ativação do reset está condicionada ao sinal clk estar passando pela borda ascendente (ou seja, este sinal é síncrono)

Noções de Linguagem VHDL

Registradores

Usando explicitamente flip-flops do tipo D

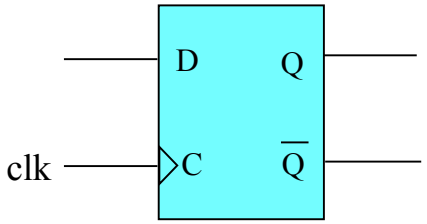


Uma abordagem possível para se descrever um registrador de vários bits é criar uma entidade que instancia flip-flops na quantidade desejada. Vejamos…

Noções de Linguagem VHDL

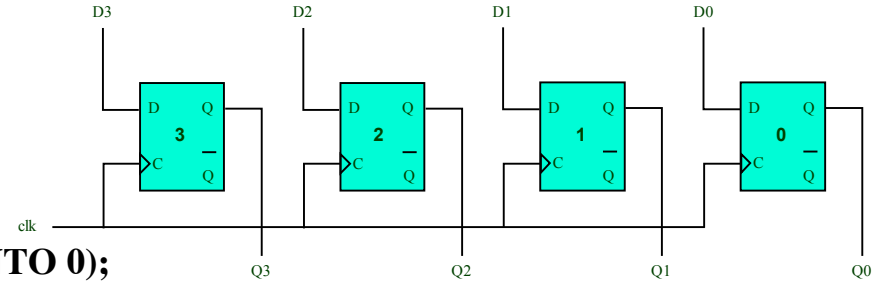
Registradores (Usando explicitamente flip-flops do tipo D)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ffD IS
PORT ( D, clk :  IN STD_LOGIC;
          Q :       OUT STD_LOGIC);
END ffD;

ARCHITECTURE comportamento OF ffD IS
BEGIN
   PROCESS (clk)
   BEGIN
       IF clk'EVENT AND clk = '1' THEN
           Q <= D;
       END IF;
   END PROCESS;
END comportamento;
```



Noções de Linguagem VHDL

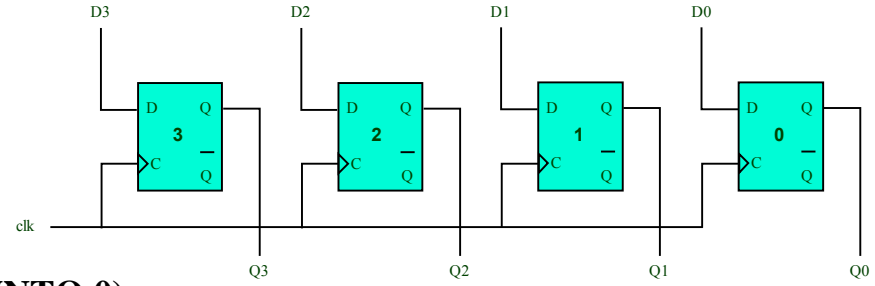
Registrador de 4 bits (com flip-flops do tipo D)




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY reg4b IS
PORT ( clk :  IN STD_LOGIC;
          D :    IN STD_LOGIC_VECTOR (3 DOWNTO 0);
          Q :    OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END reg4b;

ARCHITECTURE estrutura OF reg4b IS

COMPONENT ffD
  PORT ( D, clk : IN STD_LOGIC; Q : OUT STD_LOGIC);
END COMPONENT;

BEGIN
   ffD0: ffD PORT MAP (D(0), clk, Q(0));
   ffD1: ffD PORT MAP (D(1), clk, Q(1));
   ffD2: ffD PORT MAP (D(2), clk, Q(2));
   ffD3: ffD PORT MAP (D(3), clk, Q(3));
END estrutura;
```

Noções de Linguagem VHDL

Registrador de 4 bits (versão não hierárquica)




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY reg4b IS
PORT ( clk :  IN STD_LOGIC;
          D :    IN STD_LOGIC_VECTOR (3 DOWNTO 0);
          Q :    OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END reg4b;

ARCHITECTURE comportamento OF reg4b IS
BEGIN
   PROCESS (clk)
   BEGIN
       IF clk'EVENT AND clk = '1' THEN
           Q <= D;
       END IF;
   END PROCESS;
END comportamento;
```

Noções de Linguagem VHDL

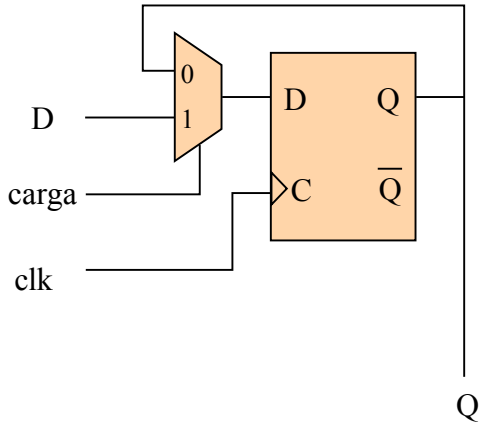
Registrador de 4 bits (versão com reset assíncrono)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY reg4b IS
PORT ( Resetn, clk :  IN STD_LOGIC;
          D :    IN STD_LOGIC_VECTOR (3 DOWNTO 0);
          Q :    OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END reg4b;

ARCHITECTURE comportamento OF reg4b IS
BEGIN
   PROCESS (clk)
   BEGIN
       IF Resetn = '0' THEN
          Q <= "0000";
       ELSEIF clk'EVENT AND clk = '1' THEN
          Q <= D;
       END IF;
   END PROCESS;
END comportamento;
```

Noções de Linguagem VHDL

Registrador de 4 bits (versão não hierárquica com carga)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY reg4b IS
PORT ( clk, carga : IN STD_LOGIC;
          D :           IN STD_LOGIC_VECTOR (3 DOWNTO 0);
          Q :           OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END reg4b;

ARCHITECTURE comportamento OF reg4b IS
BEGIN
   PROCESS (clk)
   BEGIN
       IF clk'EVENT AND clk = '1' THEN
           IF carga='1' THEN
               Q <= D;
           END IF;
       END IF;
   END PROCESS;
END comportamento;
```



Noções de Linguagem VHDL

Registrador de 32 bits (versão não hierárquica, c/ carga)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY reg32b IS
PORT ( clk, carga : IN STD_LOGIC;
          D :          IN STD_LOGIC_VECTOR (31 DOWNTO 0);
          Q :          OUT STD_LOGIC_VECTOR (31 DOWNTO 0));
END reg32b;

ARCHITECTURE comportamento OF reg32b IS
BEGIN
   PROCESS (clk)
   BEGIN
       IF clk'EVENT AND clk = '1' THEN
           IF carga='1' THEN
               Q  <= D;
           END IF;
       END IF;
   END PROCESS;
END comportamento;
```

Noções de Linguagem VHDL

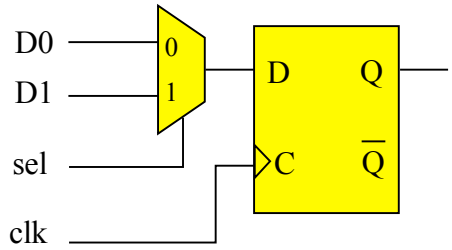
Generalizando para N bits

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY regn IS
GENERIC (N : INTEGER := 16 );
PORT ( Resetn, clk, carga : IN     STD_LOGIC;
          D :               IN     STD_LOGIC_VECTOR (N-1 DOWNTO 0);
          Q :               OUT    STD_LOGIC_VECTOR (N-1 DOWNTO 0));
END regn;

ARCHITECTURE comportamento OF regn IS
BEGIN
   PROCESS (clk)
   BEGIN
       IF Resetn = '0' THEN
          Q <=  (OTHERS => '0' );
       ELSEIF clk'EVENT AND clk = '1' THEN
          IF carga='1' THEN
             Q  <= D;
          END IF;
       END IF;
   END PROCESS;
END comportamento;
```

Noções de Linguagem VHDL

Registrador-Deslocador

- **Uma alternativa para se descrever um registrador-deslocador é utilizar a hierarquia**
- **Suponha que se deseje usar o circuito abaixo como componente básico**

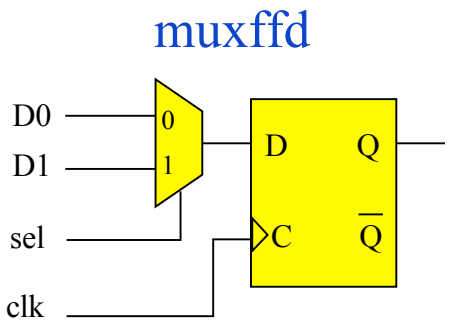


Então, descrevendo-o em VHDL…

Noções de Linguagem VHDL

Registrador-Deslocador (componente básico)

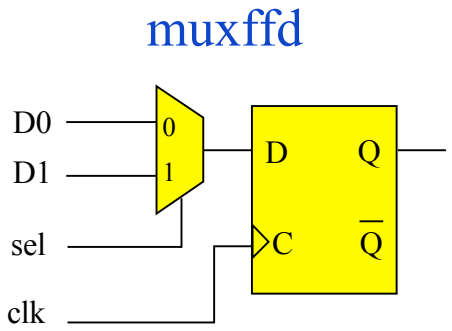
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY muxffd  IS
PORT ( D0, D1, sel, clk: IN STD_LOGIC;
          Q :                OUT STD_LOGIC );
END muxffd;

ARCHITECTURE comportamento OF muxffd IS
BEGIN
   PROCESS
   BEGIN
       WAIT UNTIL  clk'EVENT AND clk = '1' ;
          IF sel = '0' THEN
              Q  <= D0;
          ELSE
              Q  <= D1;
          END IF;
       END PROCESS;
END comportamento;
```



Noções de Linguagem VHDL

Registrador-Deslocador (de 4 bits, hierárquico)

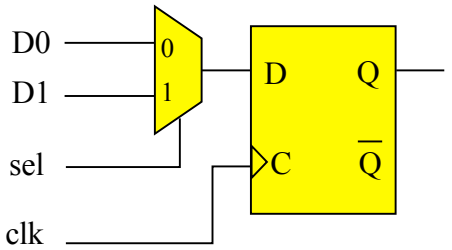
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shift4  IS
    PORT ( R                    : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
           carga, serial, clk   : IN STD_LOGIC;
           Q                    : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) );
END shift4;

ARCHITECTURE estrutura OF shift4 IS
    COMPONENT muxffd
        PORT ( D0, D1, sel, clk    : IN STD_LOGIC;
               Q                   : OUT STD_LOGIC );
     END COMPONENT;
BEGIN
    estagio3: muxffd  PORT  MAP ( serial, R(3), carga, clk, Q(3) );
    estagio2: muxffd  PORT  MAP (  Q(3), R(2), carga, clk, Q(2) );
    estagio1: muxffd  PORT  MAP (  Q(2), R(1), carga, clk, Q(1) );
    estagio0: muxffd  PORT  MAP (  Q(1), R(0), carga, clk, Q(0) );
END estrutura;
```



Noções de Linguagem VHDL

Registrador-Deslocador (de 4 bits, nao hierárquico)

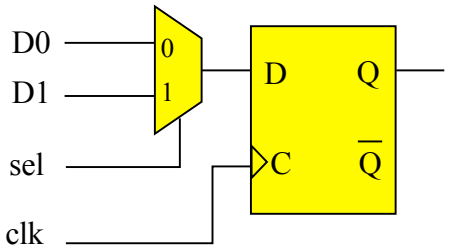
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shift4  IS
    PORT ( R                    : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
           carga, serial, clk   : IN STD_LOGIC;
           Q                    : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) );
END shift4;

ARCHITECTURE comportamento OF shift4 IS
BEGIN
  PROCESS
  BEGIN
      WAIT UNTIL  clk'EVENT AND clk = '1' ;
         IF carga = '1' THEN
            Q  <=  R;
         ELSE
            Q(0)  <=  Q(1);
            Q(1)  <=  Q(2);
            Q(2)  <=  Q(3);
            Q(3)  <=  serial;
         END IF;
      END PROCESS;
END comportamento;
```



Noções de Linguagem VHDL

Registrador-Deslocador (com n bits, não hierárquico)

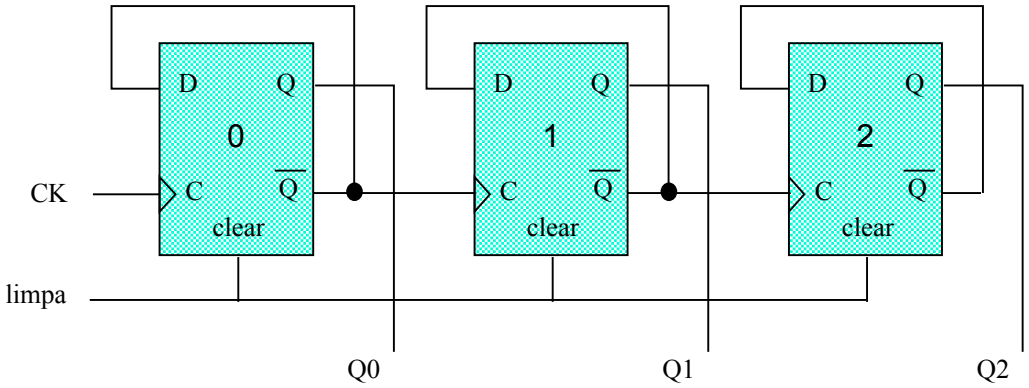
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shiftn  IS
    GENERIC ( N : INTEGER := 8 );
    PORT ( R                    : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
           carga, serial, clk   : IN STD_LOGIC;
           Q                    : BUFFER STD_LOGIC_VECTOR(N-1 DOWNTO 0) );
END shiftn;

ARCHITECTURE comportamento OF shiftn IS
BEGIN
  PROCESS
  BEGIN
      WAIT UNTIL  clk'EVENT AND clk = '1' ;
         IF carga = '1' THEN
            Q  <=  R;
         ELSE
            Genbits: FOR i IN 0 TO N-2 LOOP
                 Q(i)  <=  Q(i+1);
            END LOOP;
            Q(N-1)  <=  w;
         END IF;
      END PROCESS;
END comportamento;
```



Noções de Linguagem VHDL

Contador Assíncrono (*ripple*) de 3 bits

Usando flip-flops D – versão 2



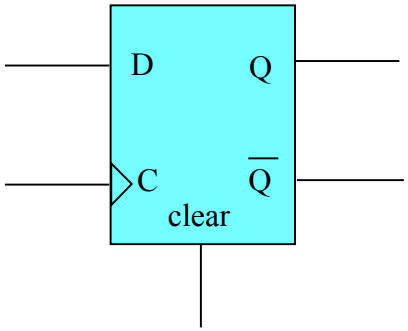
- **Para descrever este tipo de contador em VHDL usando hierarquia, iremos instanciar o flip-flop D**
- **Porém, a descrição do flip-flop D em VHDL deve conter ambas saídas, Q e NQ**

Noções de Linguagem VHDL

Flip-flop D (versão 2)

Com saídas Q e NQ

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ffDv2 IS
PORT ( D, clk, limpa : IN STD_LOGIC;
          Q, NQ : OUT STD_LOGIC);
END ffDv2;

ARCHITECTURE comportamento OF ffDv2 IS
BEGIN
   PROCESS (clk, limpa)
   BEGIN
       IF limpa='1' THEN
           Q <= '0';     NQ <= '1';
       ELSIF clk'EVENT AND clk = '1' THEN
           Q <= D;     NQ <= not D;
       END IF;
   END PROCESS;
END comportamento;
```

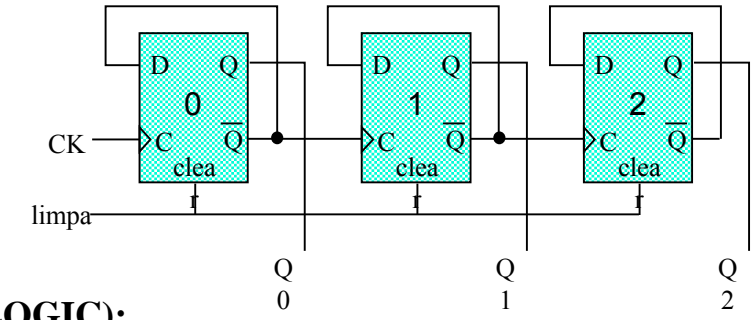


Noções de Linguagem VHDL

Contador Assíncrono (*ripple*) de 3 bits

(Usando o flip-flop versão 2)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY cont3b IS
PORT ( clk, limpa : IN STD_LOGIC;
          Aout : OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
END cont3b;

ARCHITECTURE estrutura OF cont3b IS

     SIGNAL NQ0, NQ1, NQ2 : STD_LOGIC;

COMPONENT ffDv2
  PORT ( D, clk, limpa: IN STD_LOGIC; Q, NQ : OUT STD_LOGIC);
END COMPONENT;

BEGIN
   ffD0: ffDv2 PORT MAP (NQ0, clk, limpa, Aout(0), NQ0);
   ffD1: ffDv2 PORT MAP (NQ1, NQ0, limpa, Aout(1), NQ1);
   ffD2: ffDv2 PORT MAP (NQ2, NQ1, limpa, Aout(2), NQ2);
END estrutura;
```



Noções de Linguagem VHDL

Contador Assíncrono (*ripple*) de 4 bits

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY contador IS
PORT ( clk, limpa , carga   : IN       STD_LOGIC;
           Q                : OUT    STD_LOGIC_VECTOR( 3 DOWNTO 0) );
END contador ;

ARCHITECTURE comportamento OF contador IS
   SIGNAL conta : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
BEGIN
   PROCESS (clk, limpa)
   BEGIN
      IF limpa = '0' THEN
            conta <= "0000";
      ELSEIF ( clk'EVENT AND clk = '1' ) THEN
          IF carga = '1' THEN
              conta  <=  conta + 1;
          ELSE
              conta <=  conta;
          END IF;
       END IF;
    END PROCESS;
    Q <= conta;
END comportamento;
```

Noções de Linguagem VHDL

Contador Assíncrono (*ripple*) de n bits

(Com carga paralela e usando sinais do tipo "integer")

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY contnb IS
PORT ( R                    : IN        INTEGER RANGE 0 TO 15;
       clk, limpa , carga   : IN        STD_LOGIC;
       Q                    : BUFFER INTEGER RANGE 0 TO 15 );
END contnb ;

ARCHITECTURE comportamento OF contnb IS
BEGIN
   PROCESS (clk, limpa)
   BEGIN
      IF limpa = '0' THEN
            Q <= 0;
      ELSEIF ( clk'EVENT AND clk = '1' ) THEN
          IF carga = '1' THEN
              Q  <=  R;
          ELSE
              Q  <=  Q + 1;
          END IF;
              END IF;
       END PROCESS;
END comportamento;
```

Noções de Linguagem VHDL

Máquinas de Estados

- VHDL não define padrão para a descrição de máquinas de estados finitos
- Existe mais de uma maneira de se descrever uma dada FSM
- Considere a FSM com o seguinte diagrama de estados

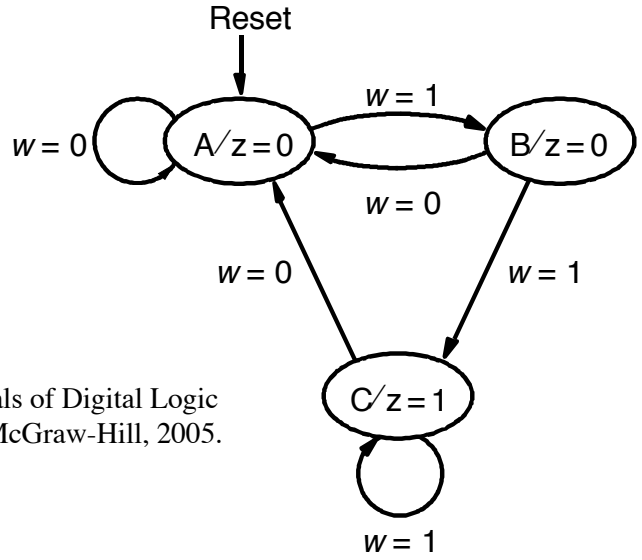


Figura © Brown, S & Vranesic, Z. Fundamentals of Digital Logic with VHDL Design. 2nd Edition, New York; McGraw-Hill, 2005.

- Esta FSM segue o **modelo de Moore**

Noções de Linguagem VHDL

Máquinas de Estados

FSM descrita segundo o Modelo de Moore, Versão 1 (somente 1 processo)

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;

3   ENTITY simple IS
4         PORT (   Clock, Resetn, w      : IN     STD_LOGIC ;
                   z                     : OUT    STD_LOGIC ) ;
5   END simple ;

7   ARCHITECTURE Behavior OF simple IS
8         TYPE State_type IS (A, B, C) ;
9         SIGNAL y : State_type ;
10  BEGIN
11        PROCESS ( Resetn, Clock )
12        BEGIN
13              IF Resetn = '0' THEN
14                    y <= A ;
15              ELSIF (Clock'EVENT AND Clock = '1') THEN
16                    CASE y IS
17                          WHEN A =>
18                                IF w = '0' THEN
19                                      y <= A ;
20                                ELSE
21                                      y <= B ;
22                                END IF ;
```

Noções de Linguagem VHDL

Máquinas de Estados

```
15              ELSIF (Clock'EVENT AND Clock = '1') THEN
16                  CASE y IS
17                      WHEN A =>
18                          IF w = '0' THEN
19                              y <= A ;
20                          ELSE
21                              y <= B ;
22                          END IF ;
23                      WHEN B =>
24                          IF w = '0' THEN
25                              y <= A ;
26                          ELSE
27                              y <= C ;
28                          END IF ;
29                      WHEN C =>
30                          IF w = '0' THEN
31                              y <= A ;
32                          ELSE
33                              y <= C ;
34                          END IF ;
35                  END CASE ;
36              END IF ;
37          END PROCESS ;
38          z <= '1' WHEN y = C ELSE '0' ;
39  END Behavior ;
```

FSM descrita segundo o Modelo de Moore, Versão 1 (continuação)

Noções de Linguagem VHDL

Máquinas de Estados

```
ARCHITECTURE Behavior OF simple IS
        TYPE State_type IS (A, B, C) ;
        SIGNAL y_present, y_next : State_type ;
BEGIN
        PROCESS ( w, y_present )
        BEGIN
                CASE y_present IS
                        WHEN A =>
                                IF w = '0' THEN
                                        y_next <= A ;
                                ELSE
                                        y_next <= B ;
                                END IF ;
                        WHEN B =>
                                IF w = '0' THEN
                                        y_next <= A ;
                                ELSE
                                        y_next <= C ;
                                END IF ;
                        WHEN C =>
                                IF w = '0' THEN
                                        y_next <= A ;
                                ELSE
                                        y_next <= C ;
                        END IF ;
                END CASE ;
        END PROCESS ;
```

FSM descrita segundo o Modelo de Moore, Versão 2 (2 processos)

Um processo para o bloco de próxima estado …

Noções de Linguagem VHDL

Máquinas de Estados

FSM descrita segundo o Modelo de Moore, Versão 2 (2 processos)

```
        PROCESS (Clock, Resetn)
        BEGIN
                IF Resetn = '0' THEN
                        y_present <= A ;
                ELSIF (Clock'EVENT AND Clock = '1') THEN
                        y_present <= y_next ;
                END IF ;
        END PROCESS ;

        z <= '1' WHEN y_present = C ELSE '0' ;
END Behavior ;
```

… e outro processo para o registrador de estado

Noções de Linguagem VHDL

Máquinas de Estados

- **Considere a FSM descrita pelo diagrama de estados que segue, a qual segue o modelo de Mealy**

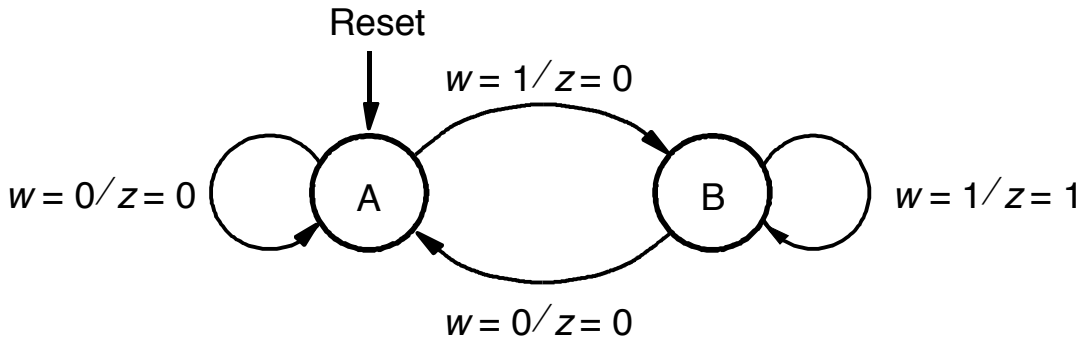


Figura © Brown, S & Vranesic, Z. Fundamentals of Digital Logic with VHDL Design. 2nd Edition, New York; McGraw-Hill, 2005.

Noções de Linguagem VHDL

Máquinas de Estados

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY mealy IS
      PORT ( Clock, Resetn, w      : IN     STD_LOGIC ;
                   z               : OUT    STD_LOGIC ) ;
END mealy ;

ARCHITECTURE Behavior OF mealy IS
      TYPE State_type IS (A, B) ;
      SIGNAL y : State_type ;
BEGIN
      PROCESS ( Resetn, Clock )
      BEGIN
            IF Resetn = '0' THEN
                  y <= A ;
            ELSIF (Clock'EVENT AND Clock = '1') THEN
                  CASE y IS
                        WHEN A =>
                              IF w = '0' THEN y <= A ;
                              ELSE y <= B ;
                              END IF ;
                         WHEN B =>
                              IF w = '0' THEN y <= A ;
                              ELSE y <= B ;
                              END IF ;
                  END CASE ;
            END IF ;
      END PROCESS ;
```

FSM descrita segundo o Modelo de Mealy (2 processos)

Noções de Linguagem VHDL

Máquinas de Estados

FSM descrita segundo o Modelo de Mealy (2 processos)
Continuação

```
    PROCESS ( y, w )
    BEGIN
        CASE y IS
            WHEN A =>
                z <= '0' ;
            WHEN B =>
                z <= w ;
        END CASE ;
    END PROCESS ;
END Behavior ;
```

Noções de Linguagem VHDL

Máquinas de Estados

Outra FSM

```
Use ieee.std_logic_1164.all;

ENTITY Cont IS
  PORT(clk, reset, flag1 :  IN STD_LOGIC;
           S1, S2, S3 : OUT STD_LOGIC);
END Cont;

ARCHITECTURE comportamento OF Cont IS
SIGNAL state: STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
PROCESS (clk, reset)
BEGIN
  IF (reset='0') THEN
    state <= "00";
```

Definição das variáveis de estado

Noções de Linguagem VHDL

Máquinas de Estados (continuação)

Outra FSM

```
ELSIF (clk'EVENT AND clk = '1') THEN
   CASE state IS
     WHEN "00" =>  state <= "01";   S1 <= '0';   S2 <= '1';   S3 <= '1';
     WHEN "01" =>  IF flag1 = '1' THEN   state <= "10";
                   ELSE state <= "11";  END IF;
                   S1 <= '1';   S2 <= '1';   S3 <= '0';
     WHEN "10" =>  state <= "11";  S1 <= '1';   S2 <= '1';   S3 <= '1';
     WHEN "11" =>  state <= "00";  S1 <= '1';   S2 <= '0';   S3 <= '1';
     WHEN OTHERS => state <= "00";
   END CASE;
 END IF;
END PROCESS;
END comportamento;
```