IC-UNICAMP

MC 613

IC/Unicamp
2012s1
Prof Guido Araújo
Prof Mario Côrtes

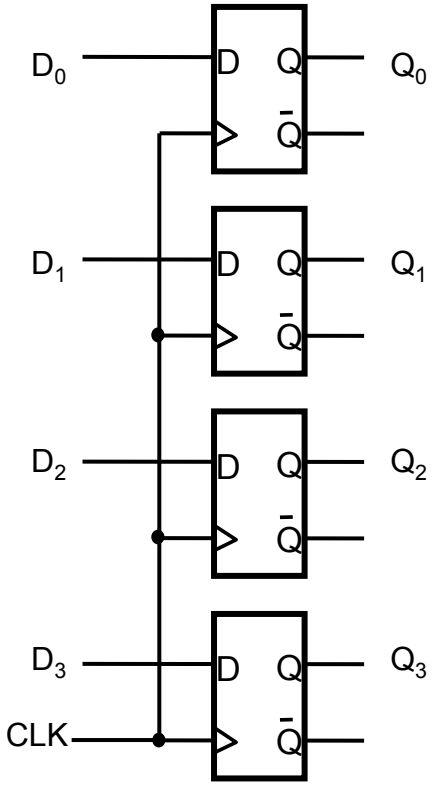
Registradores e Contadores

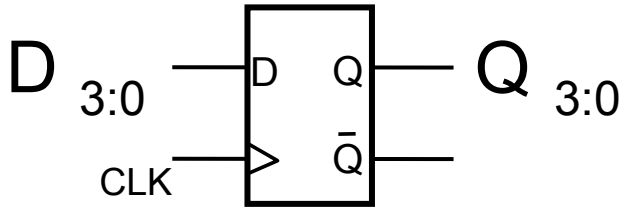
Tópicos de Registradores

- Construção usando flip-flops
- Clear assíncrono e Enable
- Registradores deslocamento
- Carga paralela
- Registrador deslocamento universal
- Exemplo de uso em barramento

Registradores

- Conjunto de elementos de memória (flip-flops) utilizados para armazenar n bits.
- Utilizam em comum os sinais de clock e controle





IC-UNICAMP

8-bit register with asynchronous clear

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

ENTITY reg8 IS
    PORT ( D   : IN   STD_LOGIC_VECTOR(7 DOWNTO 0) ;
           Resetn, Clock: IN STD_LOGIC ;
           Q   : OUT  STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END reg8 ;

ARCHITECTURE Behavior OF reg8 IS
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Q <= "00000000" ;
        ELSIF Clock'EVENT AND Clock = '1' THEN Q <= D ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

IC-UNICAMP

n-bit register with enable

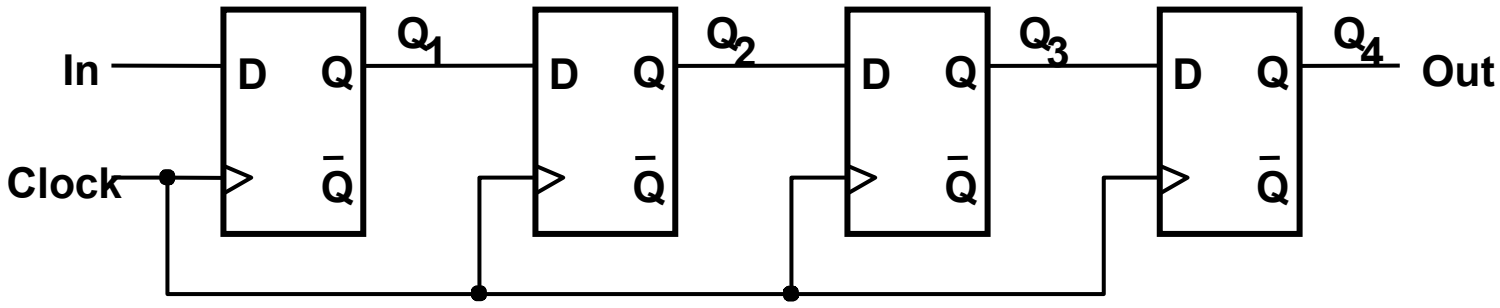
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY regn IS
    GENERIC ( N : INTEGER := 8 ) ;
    PORT (R : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
           Rin, Clock: IN STD_LOGIC ;
           Q   : OUT  STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END regn ;

ARCHITECTURE Behavior OF regn IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF Rin = '1' THEN Q <= R ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Shift Register

	In	Q_1	Q_2	Q_3	Q_4 = Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1



Alternative Shift Register

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY shift4 IS
    PORT ( R       : IN     STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           Clock   : IN     STD_LOGIC ;
           L, w    : IN     STD_LOGIC ;
           Q       : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift4 ;

ARCHITECTURE Behavior OF shift4 IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF L = '1' THEN  Q <= R ;
        ELSE
            Q(0) <= Q(1) ;
            Q(1) <= Q(2);
            Q(2) <= Q(3) ;
            Q(3) <= w ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

IC-UNICAMP

n-bit left-to-right shift register

```
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) ;
           Clock   : IN      STD_LOGIC ;
           L, w: IN          STD_LOGIC ;
           Q : BUFFER STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END shiftn ;
ARCHITECTURE Behavior OF shiftn IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF L = '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 Behavior ;
```

IC-UNICAMP

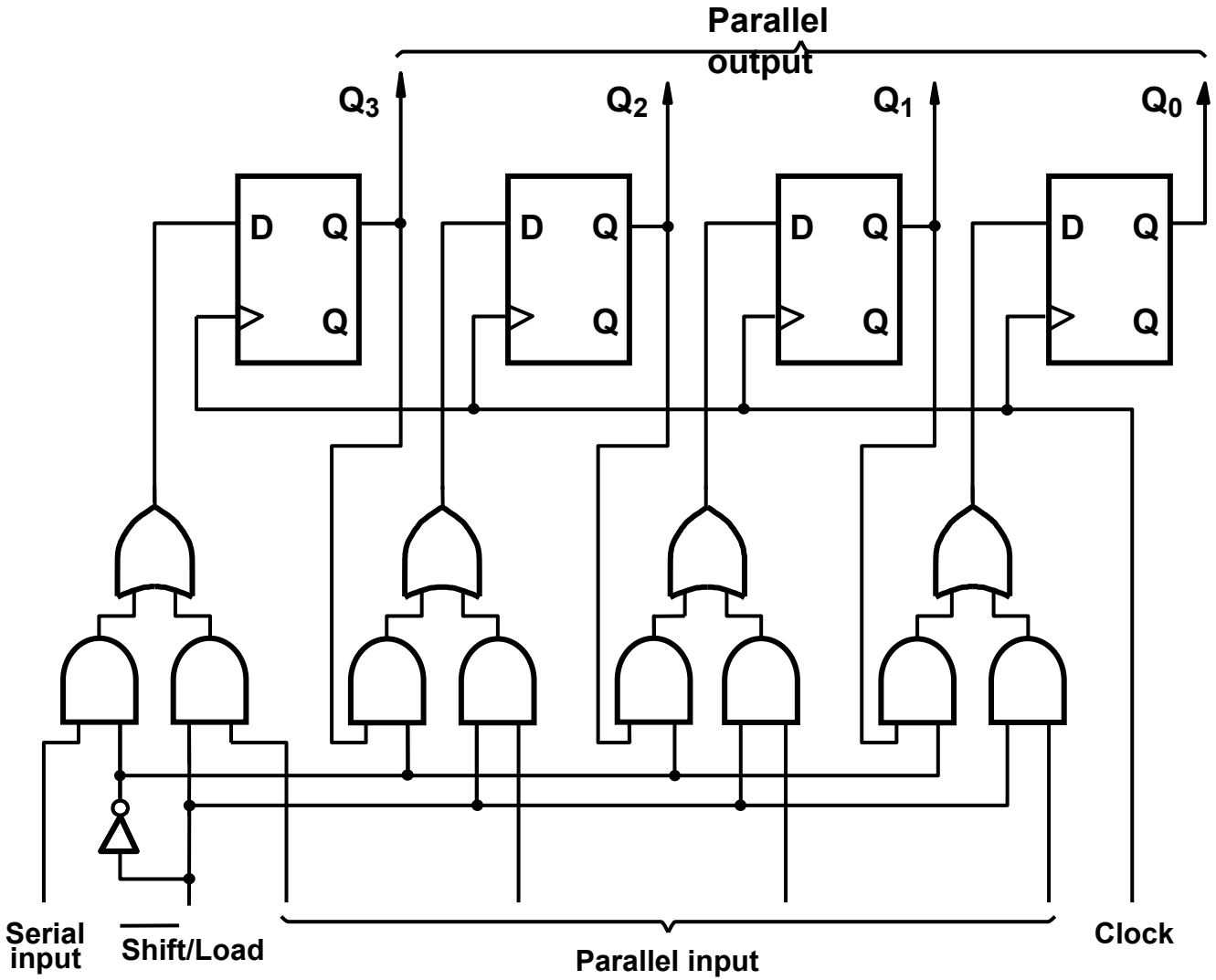
Hierarchical code for a four-bit shift register

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

ENTITY shift4 IS
    PORT ( R : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           L, w, Clock   : IN      STD_LOGIC ;
           Q   : BUFFER  STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift4 ;

ARCHITECTURE Structure OF shift4 IS
    COMPONENT muxdff
        PORT ( D0, D1, Sel, Clock  : IN STD_LOGIC ;
               Q       : OUT  STD_LOGIC ) ;
    END COMPONENT ;
BEGIN
    Stage3: muxdff PORT MAP ( w, R(3), L, Clock, Q(3) ) ;
    Stage2: muxdff PORT MAP ( Q(3), R(2), L, Clock, Q(2) ) ;
    Stage1: muxdff PORT MAP ( Q(2), R(1), L, Clock, Q(1) ) ;
    Stage0: muxdff PORT MAP ( Q(1), R(0), L, Clock, Q(0) ) ;
END Structure ;
```

Shift Register com Carga Paralela



IC-UNICAMP

Shift Register com Carga Paralela

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

ENTITY shift IS
    PORT ( Clock      : IN   STD_LOGIC ;
           Reset      : IN   STD_LOGIC ;
           Shiftin, Load    : IN   STD_LOGIC ;
           R          : IN   STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           Q          : OUT  STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift ;

ARCHITECTURE Structure OF shift IS
BEGIN
    instance: lpm_shiftreg
       GENERIC MAP (LPM_WIDTH => 4, LPM_DIRECTION =>
"RIGHT")
       PORT MAP (data => R, clock => Clock, aclr => Reset,
          load => Load, shiftin => Shiftin, q => Q ) ;
END Structure ;
```

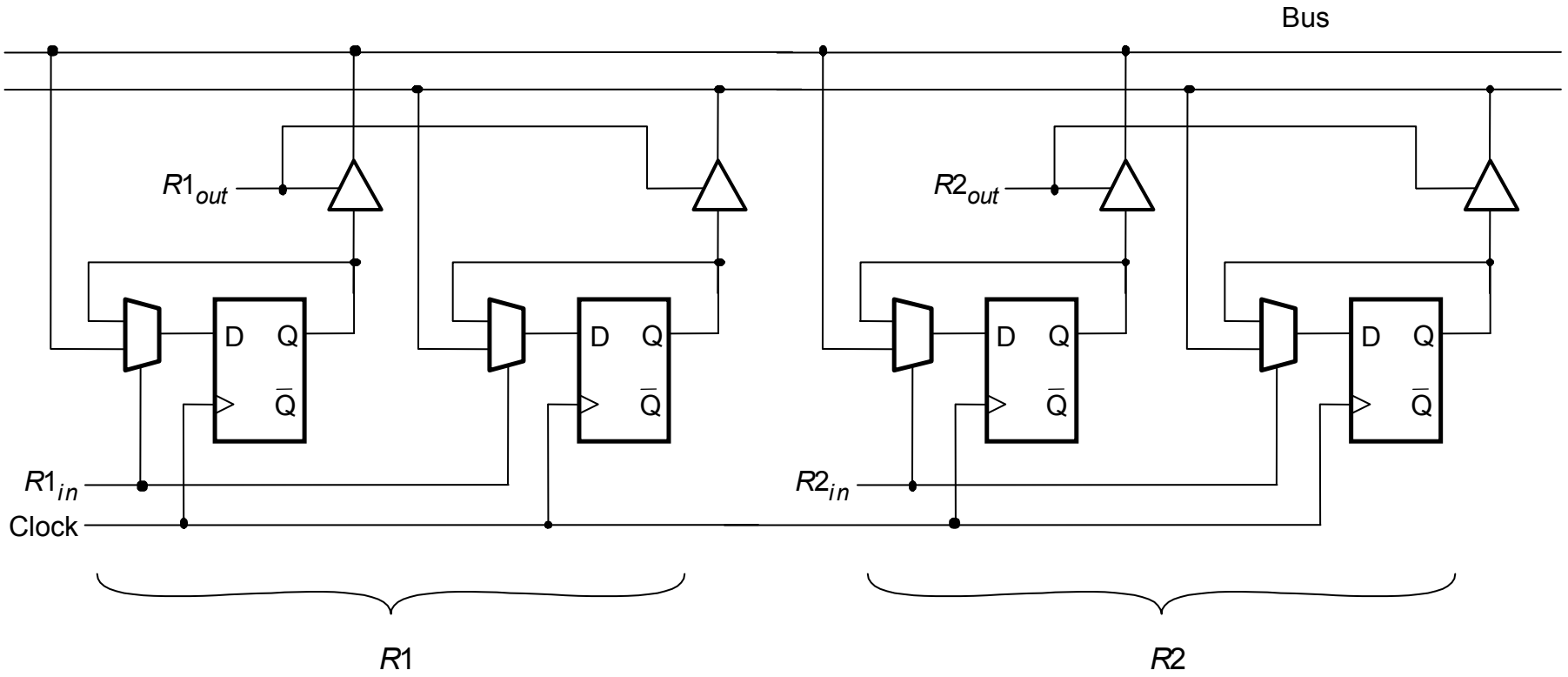
Shift Register Universal

- Entrada Serial
  - Deslocamento a Esquerda
  - Deslocamento a Direita
- Carga Paralela
- Saída Paralela

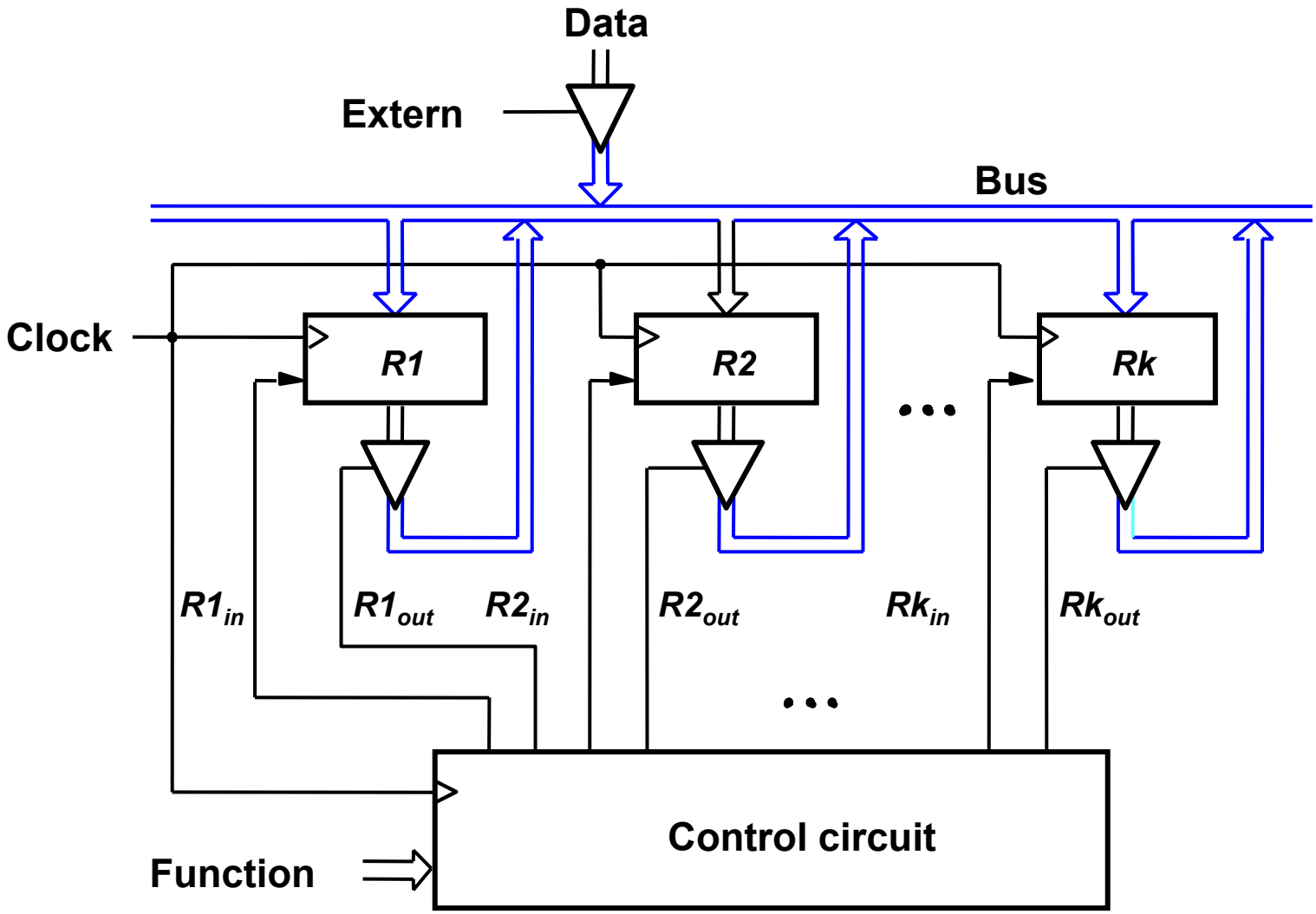
Exercício:

Diagrama do Shift Register Universal de 4 bits

Registradores em um Barramento



Registradores em um Barramento



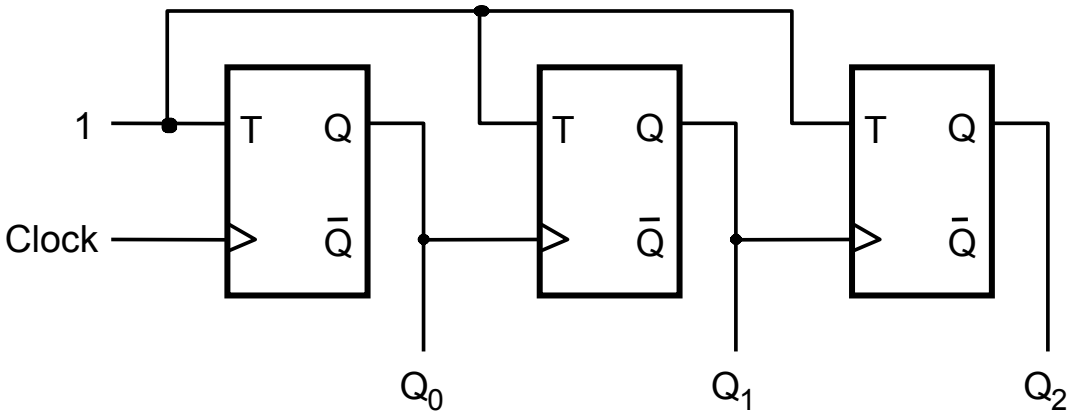
Tópicos de Contadores

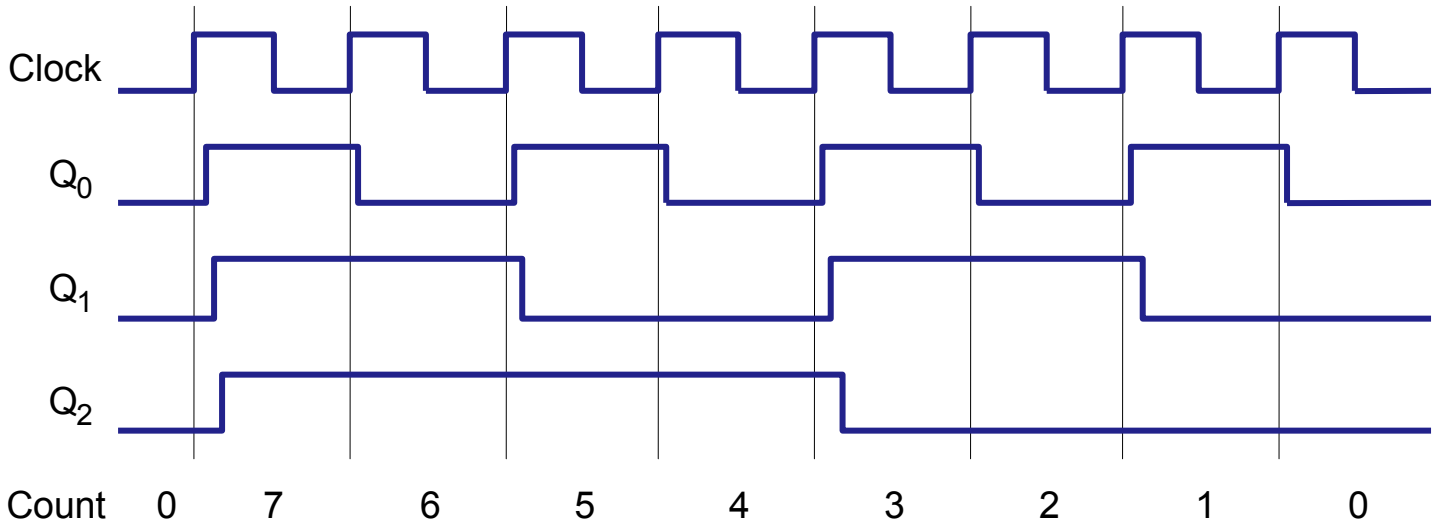
- Contadores síncronos e assíncronos
- Contadores de módulo configurável
- Contadores em anel e Johnson
- Preset e Clear síncronos e assíncronos

Contadores síncronos / assíncronos

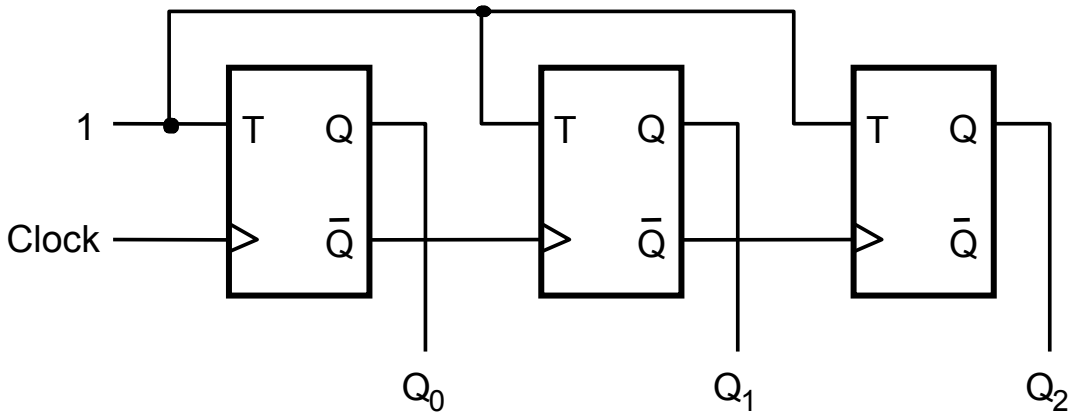
- Contadores assíncronos
  - Entrada de clock dos FFs recebe saída de estágios anteriores
  - Estado do contador: transições dos estágios não simultâneas (em ripple)
  - Circuito mínimo mas requer cuidados com decodificação
- Contadores síncronos
  - Entrada de clock dos FF recebe apenas sinal externo de clock
  - Estado do contador: transições sincronizadas (razoavelmente simultâneas)

Contador assíncrono: 3 bits DOWN

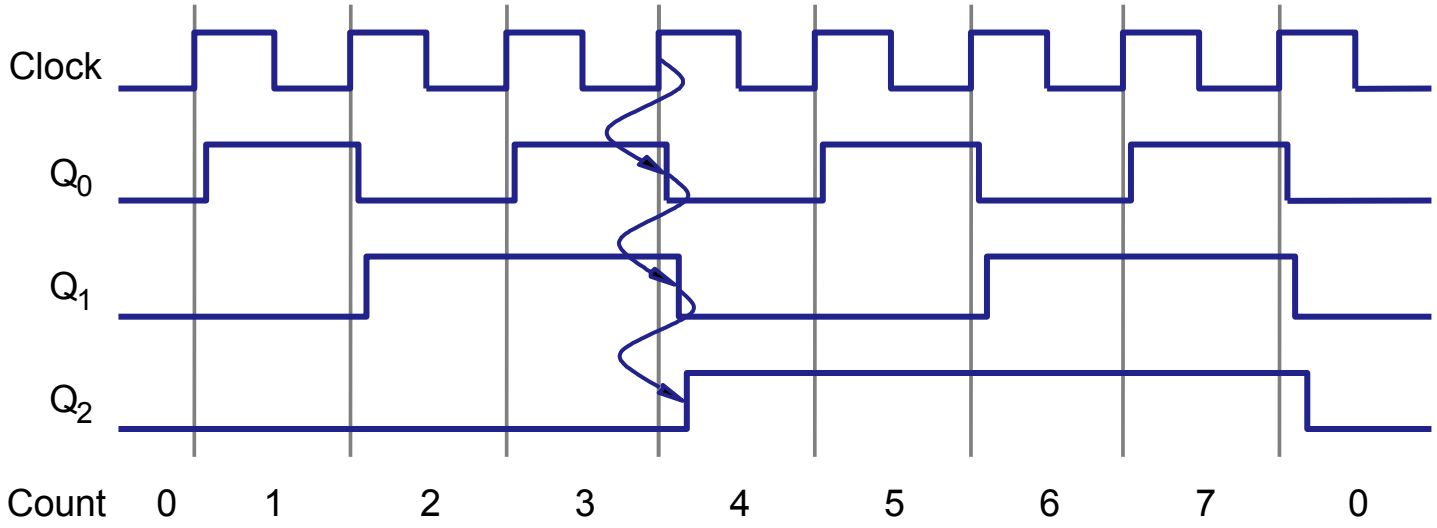




Contador assíncrono: 3 bits UP

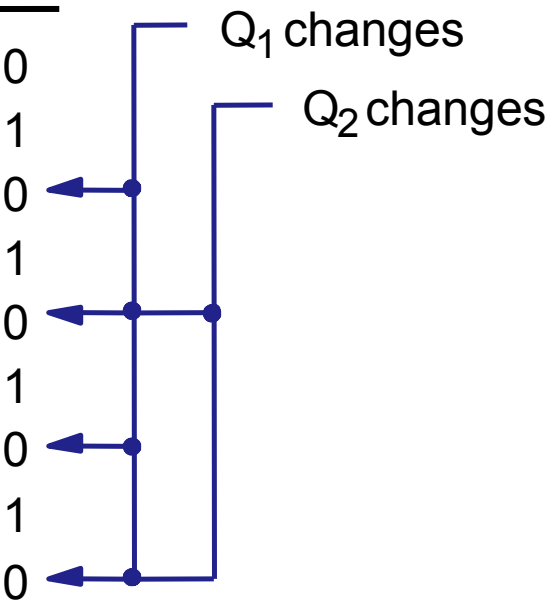


Alternativas para UP/DOWN:
- FF sens. borda de descida
- saída = ~Q

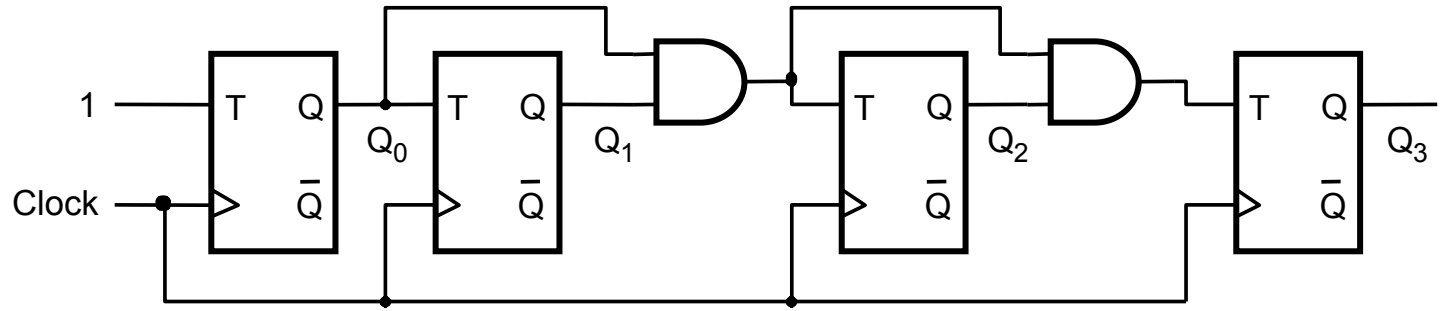


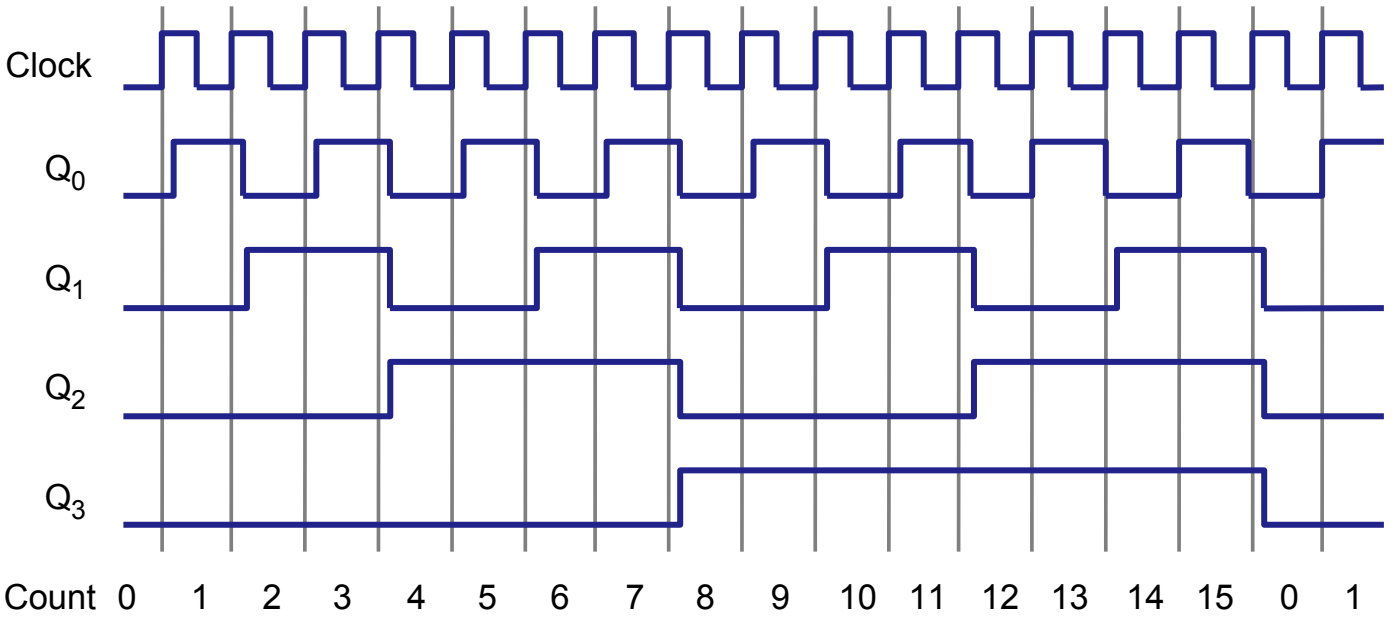
Projeto de contador síncrono: 3 bits

Clock cycle	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	**1**	0
3	0	1	1
4	**1**	**0**	0
5	1	0	1
6	1	**1**	0
7	1	1	1
8	**0**	**0**	0

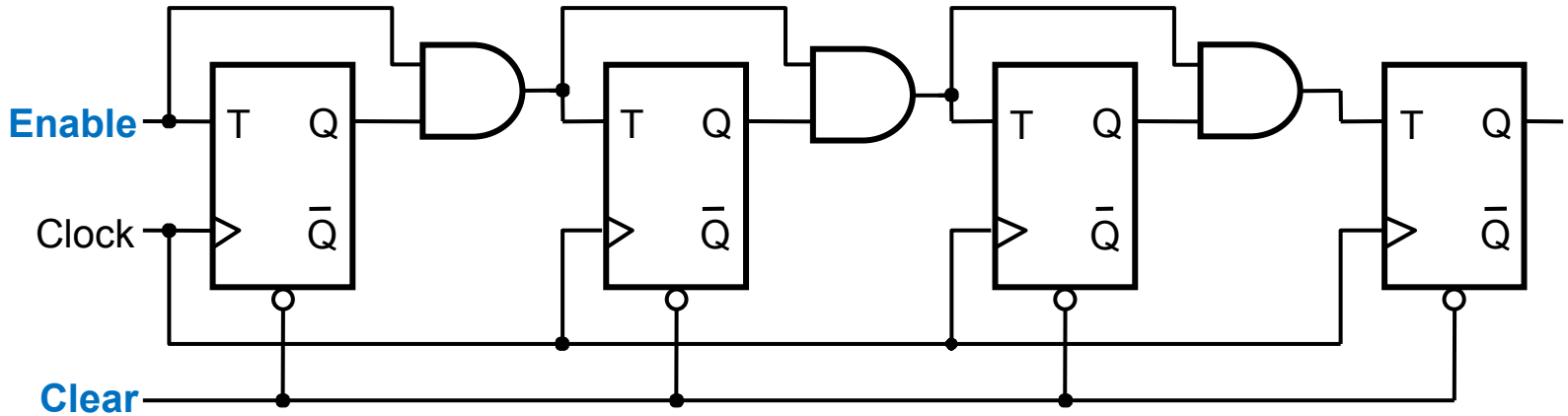


Contador síncrono: 4 bits



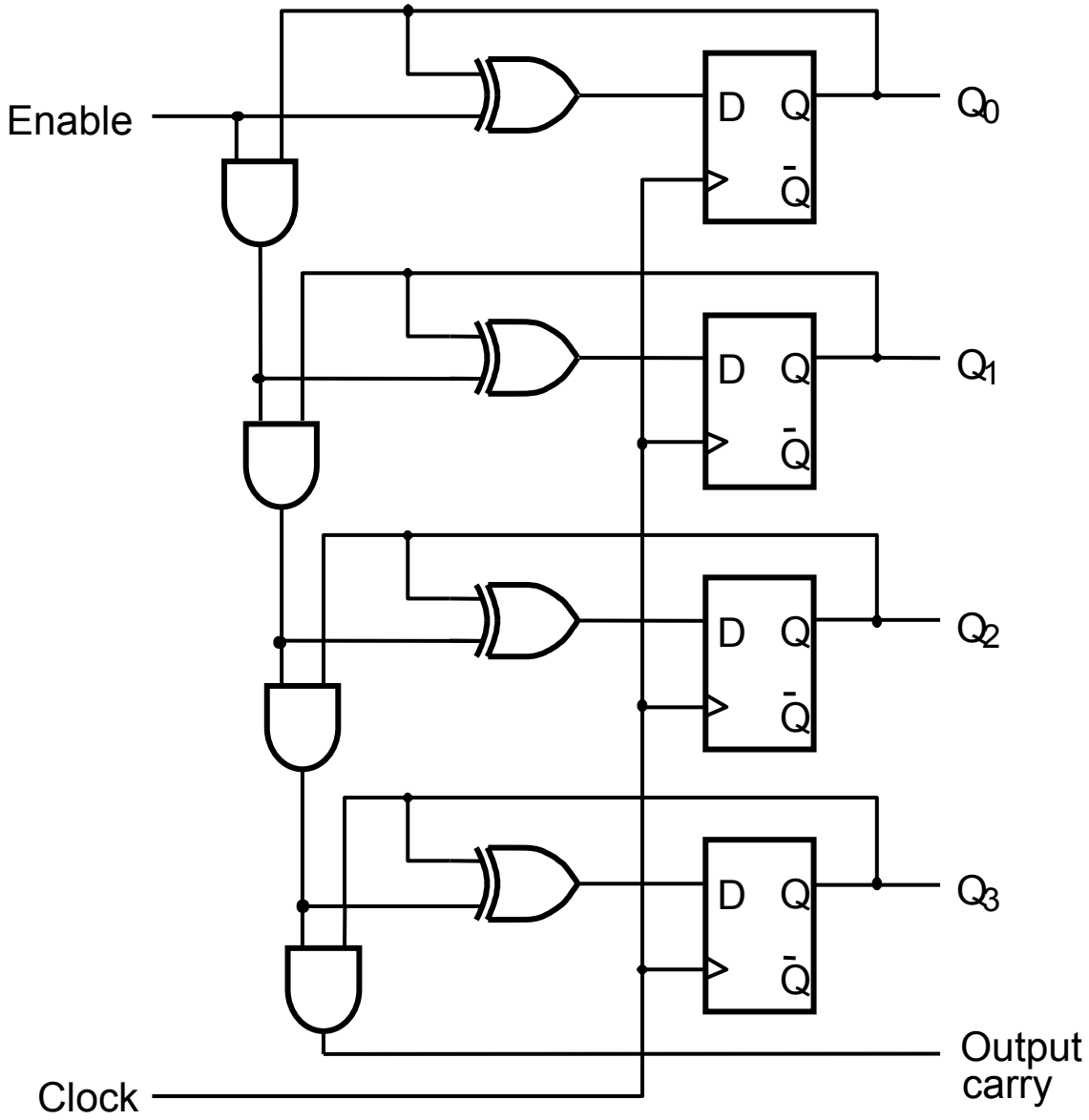


Inclusão de Enable e Clear



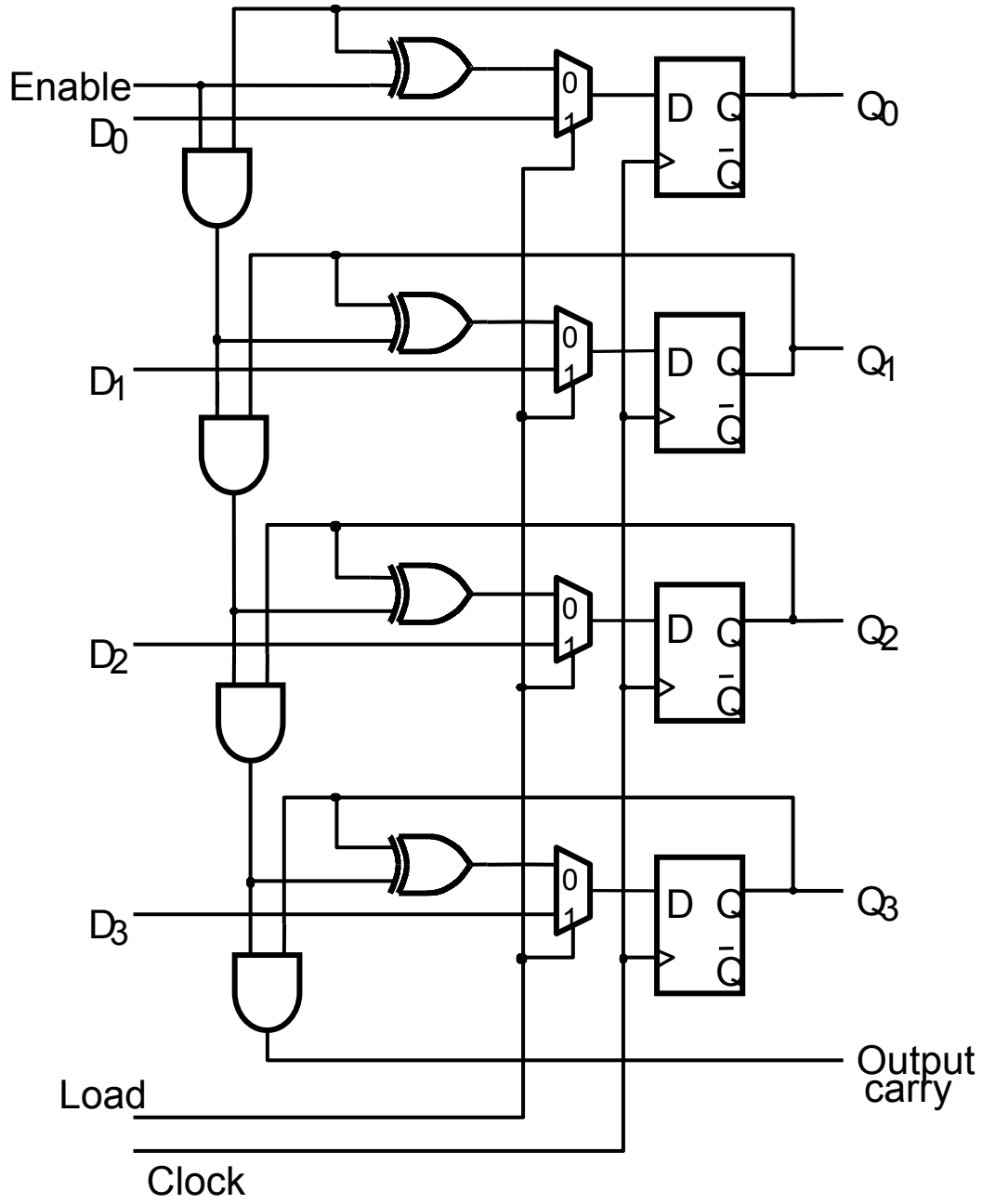
Obs: o clear é assíncrono

Contador síncrono com FF D

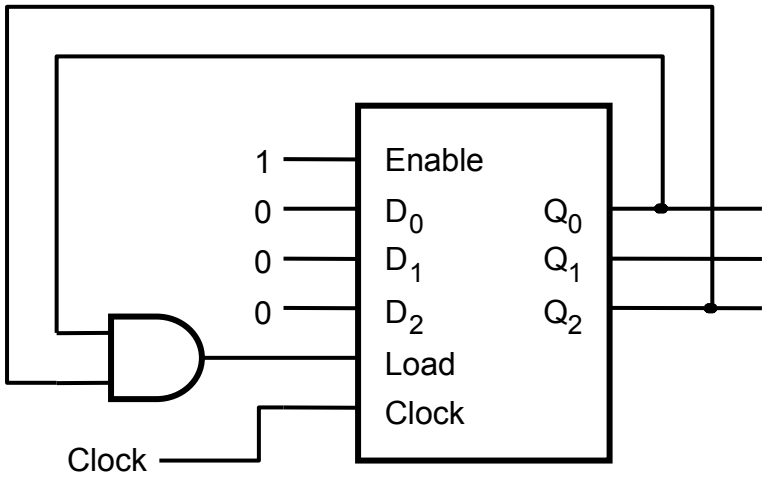


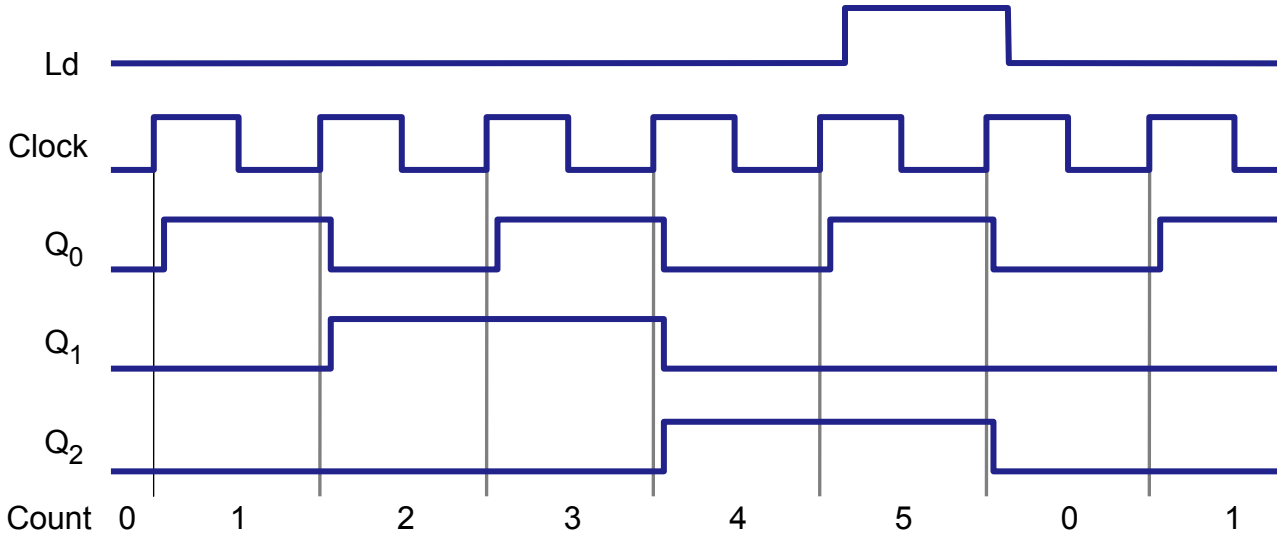
IC-UNICAMP

Inclusão de Load



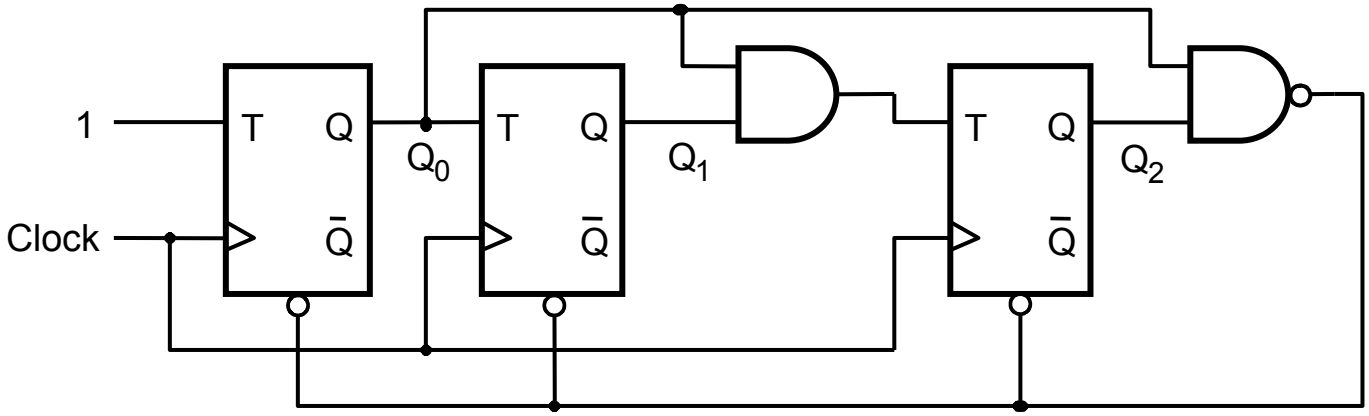
Contador mod-6 com Reset síncrono

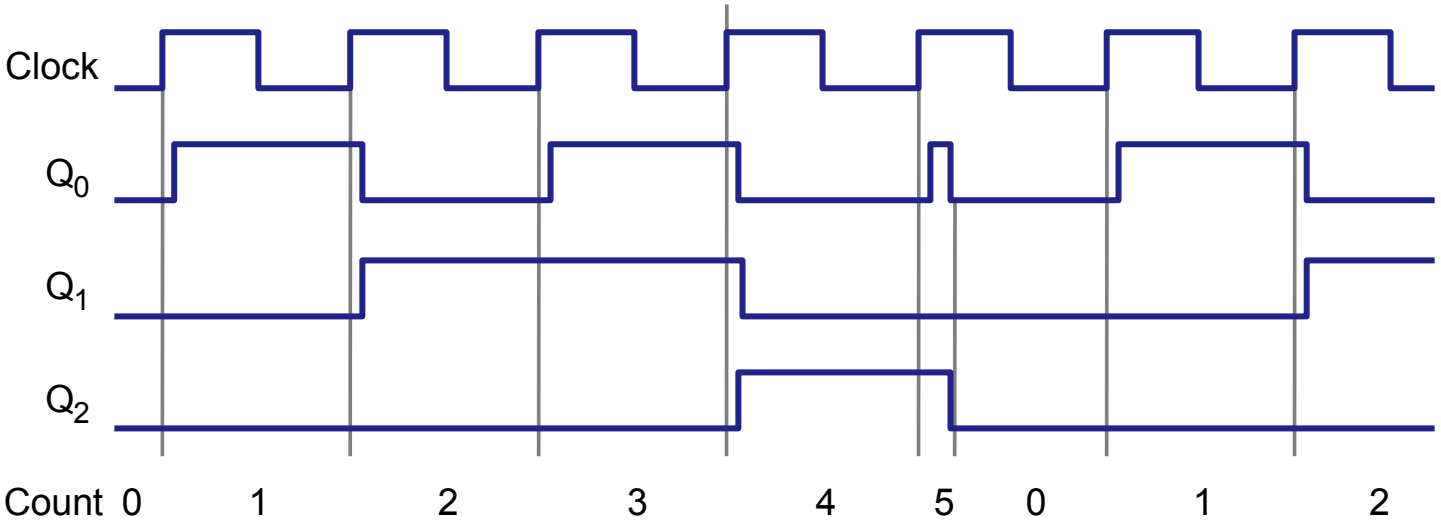




Obs: o Reset é obtido carregando-se 000 via sinal de load síncrono

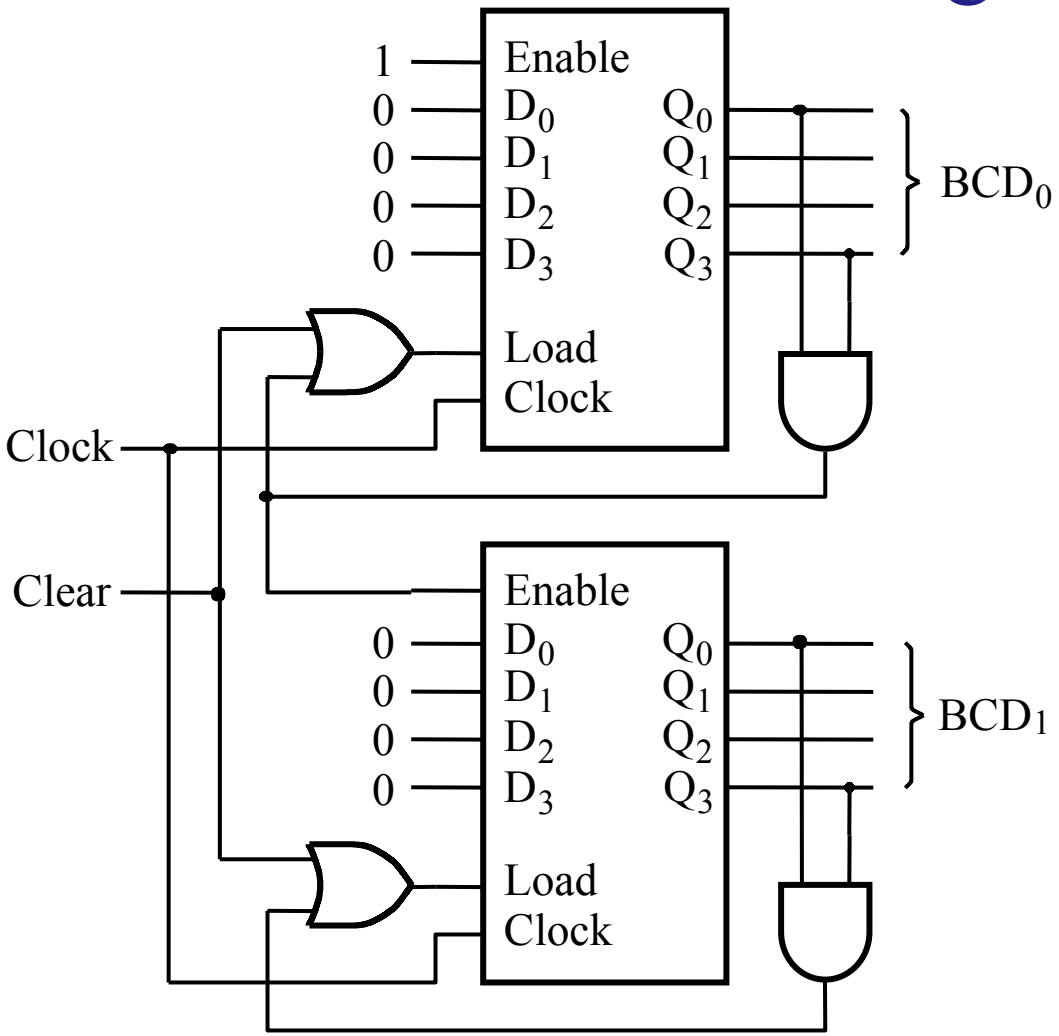
Contador mod-5 com Reset assíncrono



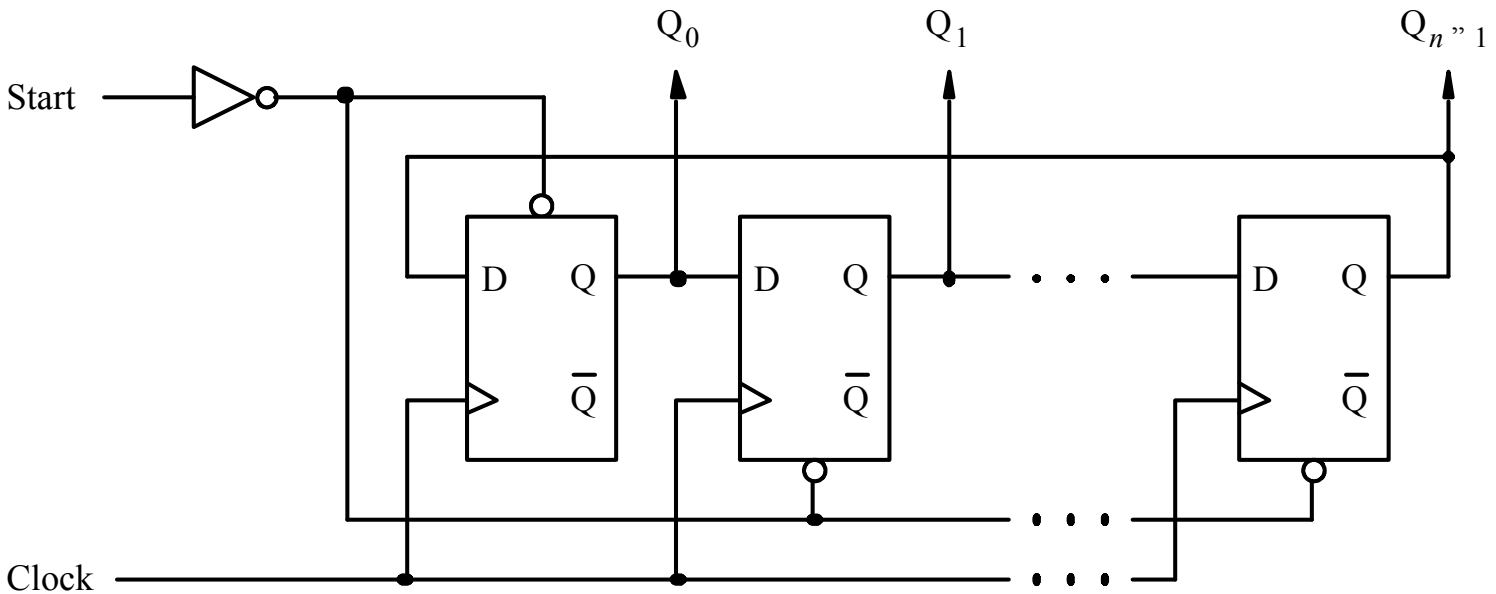


Obs: apesar do estado final a ser detectado ser o mesmo do caso anterior, estado 101 (5), o módulo deste contador é 5 (contagem 0 1 2 3 4 0 1 2 3 4)

Contador BCD de 2 dígitos



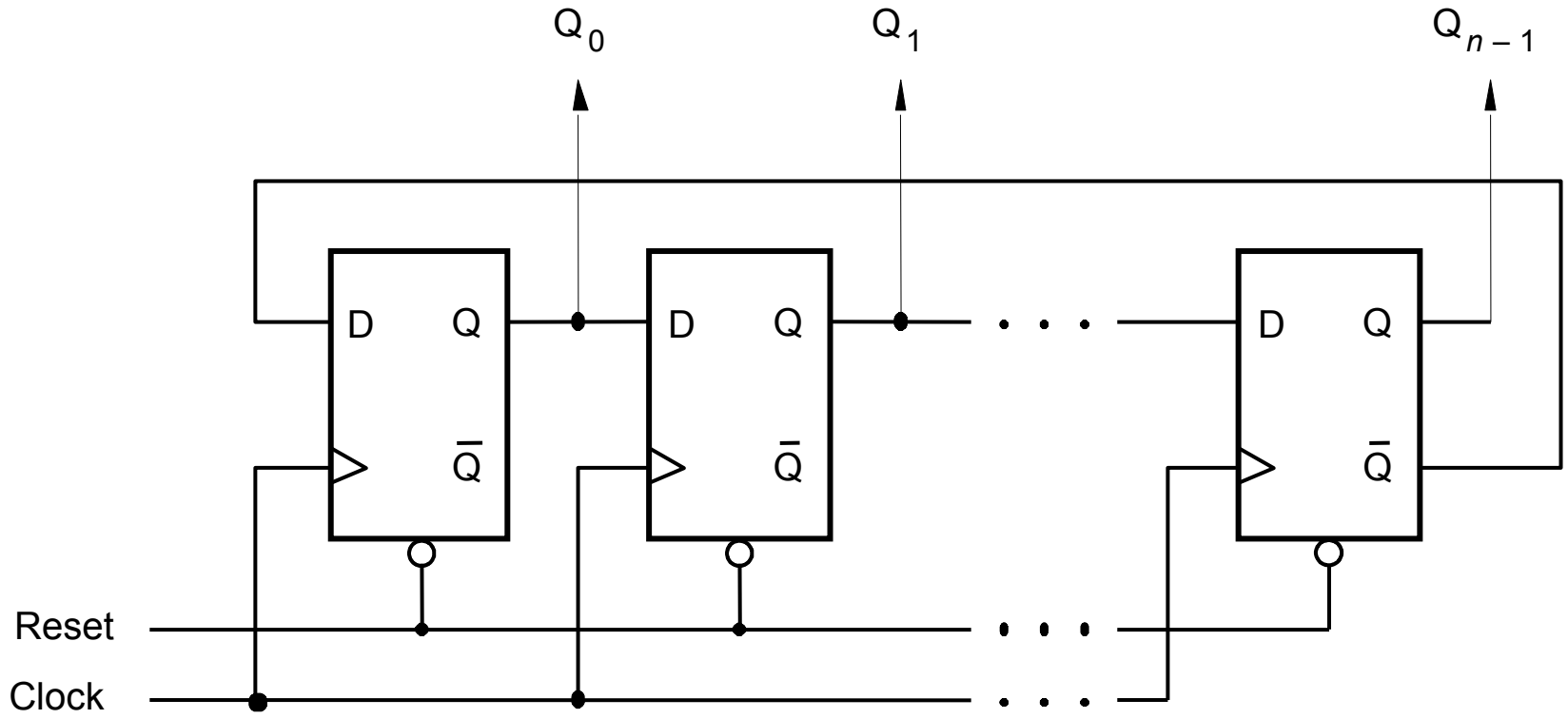
Contador em anel



Contagem p 4 bits: 1000 0100 0010 0001 1000 0100 0010 0001 ....

Módulo?

Contador Johnson



Para um contador de 4 estágios:

- Qual é a sequência de contagem
- Qual é o módulo do contador?

Implementação de contadores em VHDL

IC-UNICAMP

Contador crescente 4 bits

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY upcount IS
    PORT (  Clock, Resetn, E    : IN    STD_LOGIC ;
            Q    : OUT   STD_LOGIC_VECTOR (3 DOWNTO 0)) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
    SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Count <= "0000" ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            IF E = '1' THEN
                Count <= Count + 1 ;
            ELSE
                Count <= Count ;
            END IF ;
        END IF ;
    END PROCESS ;
    Q <= Count ;
END Behavior ;
```

Contador com LD paralelo, c/ sinais inteiros

IC-UNICAMP

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

ENTITY upcount IS
    PORT (  R : IN         INTEGER RANGE 0 TO 15 ;
            Clock, Resetn, L : IN   STD_LOGIC ;
            Q   : BUFFER  INTEGER RANGE 0 TO 15 ) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Q <= 0 ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            IF L = '1' THEN
                Q <= R ;
            ELSE
                Q <= Q + 1 ;
            END IF;
        END IF;
    END PROCESS;
END Behavior;
```

Obs: com o uso do tipo BUFFER,
o sinal count não é necessário

IC-UNICAMP

Contador decrescente

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY downcnt IS
    GENERIC ( modulus : INTEGER := 8 ) ;
    PORT ( Clock, L, E : IN   STD_LOGIC ;
           Q   : OUT INTEGER RANGE 0 TO modulus-1 ) ;
END downcnt ;

ARCHITECTURE Behavior OF downcnt IS
    SIGNAL Count : INTEGER RANGE 0 TO modulus-1 ;
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL (Clock'EVENT AND Clock = '1') ;
        IF E = '1' THEN
            IF L = '1' THEN
                Count <= modulus-1 ; -- carrega c módulo
            ELSE
                Count <= Count-1 ;
            END IF ;
        END IF ;
    END PROCESS;
    Q <= Count ;
END Behavior ;
```

IC-UNICAMP

Contador crescente com Reset Síncrono

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY upcount IS
    PORT ( Clear, Clock: IN    STD_LOGIC ;
           Q   : BUFFER   STD_LOGIC_VECTOR(1 DOWNTO 0) ) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
BEGIN
    upcount: PROCESS ( Clock )
    BEGIN
        IF (Clock'EVENT AND Clock = '1') THEN
            IF Clear = '1' THEN
                Q <= "00" ;
            ELSE
                Q <= Q + '1' ;
            END IF ;
        END IF;
    END PROCESS;
END Behavior ;
```

IC-UNICAMP

Contador BCD de 2 dígitos (entity)

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

ENTITY BCDcount IS
    PORT ( Clock       : IN     STD_LOGIC ;
           Clear, E    : IN   STD_LOGIC ;
           BCD1, BCD0  : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END BCDcount ;
```

IC-UNICAMP

Contador BCD de 2 dígitos (architect.)

```
ARCHITECTURE Behavior OF BCDcount IS
BEGIN
    PROCESS ( Clock )
    BEGIN
        IF Clock'EVENT AND Clock = '1' THEN
            IF Clear = '1' THEN
                BCD1 <= "0000" ; BCD0 <= "0000" ;
            ELSIF E = '1' THEN
                IF BCD0 = "1001" THEN
                    BCD0 <= "0000" ;
                    IF BCD1 = "1001" THEN
                        BCD1 <= "0000";
                    ELSE
                        BCD1 <= BCD1 + '1' ;
                    END IF ;
                ELSE
                    BCD0 <= BCD0 + '1' ;
                END IF ;
            END IF ;
        END IF;
    END PROCESS;

END Behavior ;
```