Circuitos Lógicos e Organização de Computadores

Capítulo 7 – Flip-Flops e Circuitos Seqüenciais

Ricardo Pannain

pannain@puc-campinas.edu.br

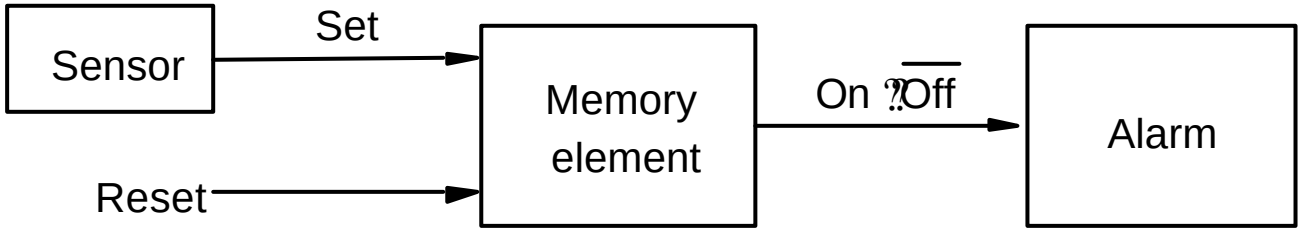
http://docentes.puc-campinas.edu.br/ceatec/pannain/

Circuito Seqüenciais

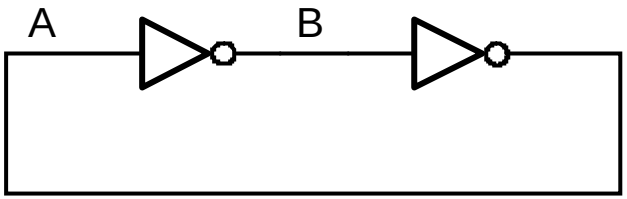
Circuito combinacional ✍ saídas dependem apenas das entradas

Cicuito seqüencial ✍ saídas dependem das entradas e do comportamento anterior do circuito

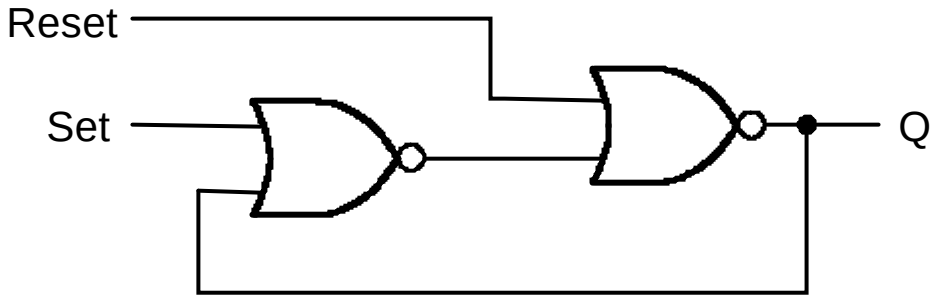
Exemplo - Controle de Sistema de Alarme



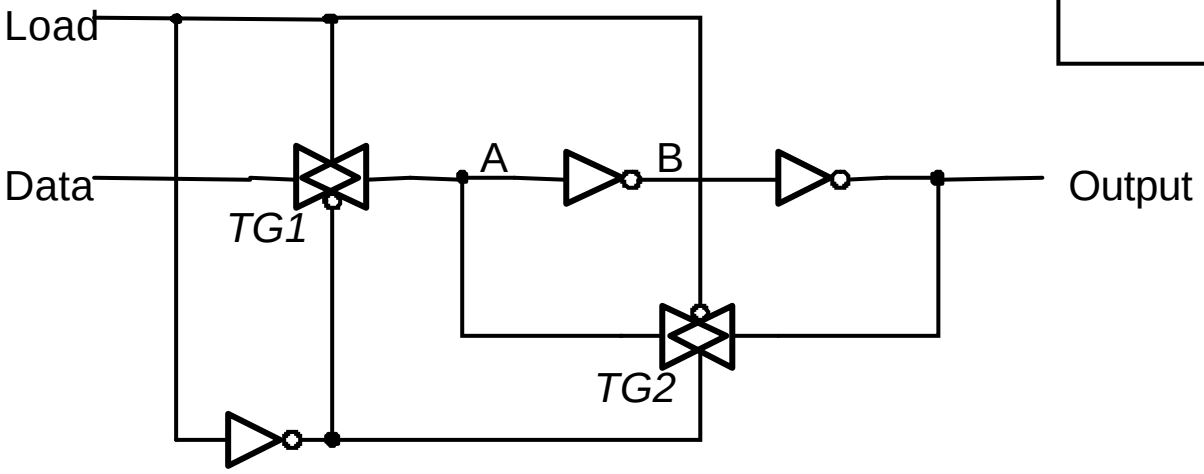
Elemento de memória simples



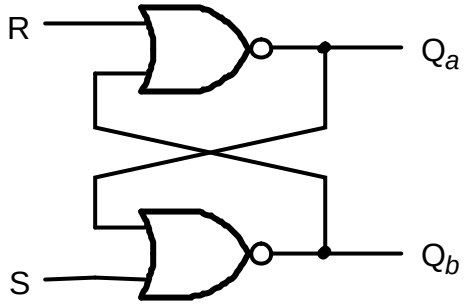
Elemento de memória com portas NOR



Elemento de memória controlado



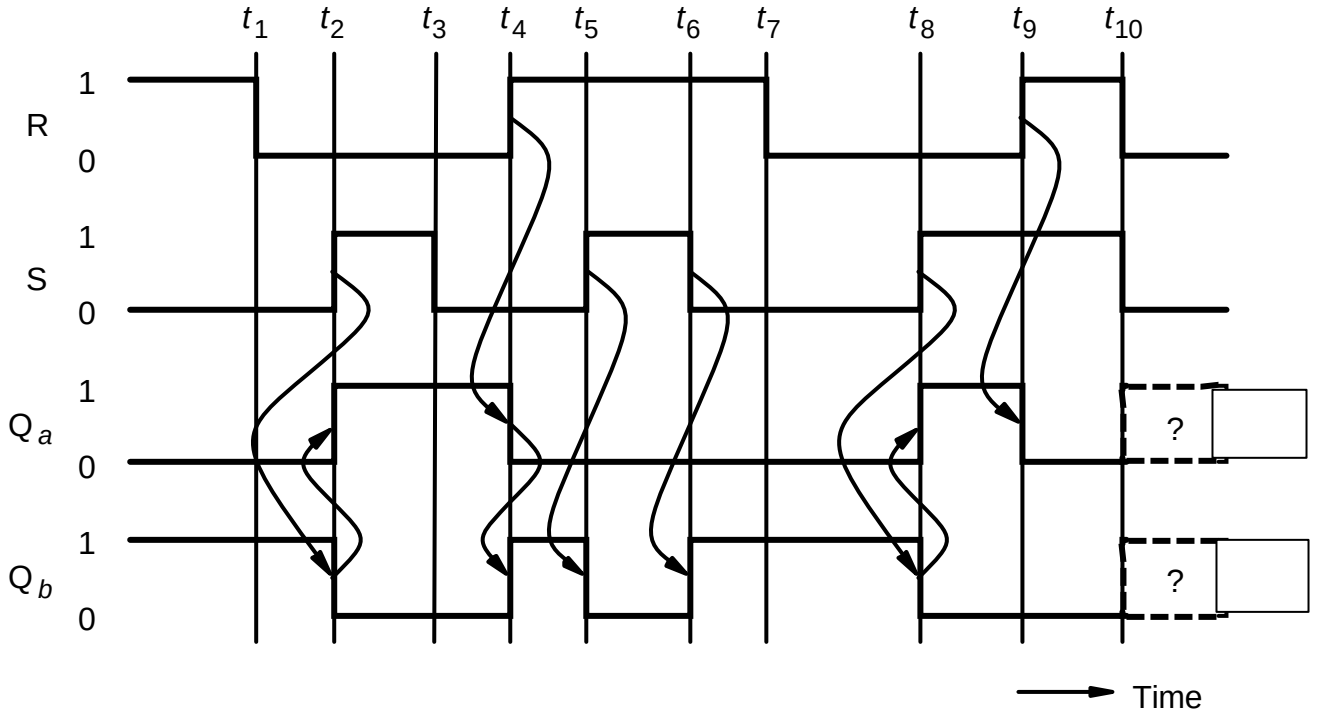
Latch construído com portas NOR



(a) Circuito

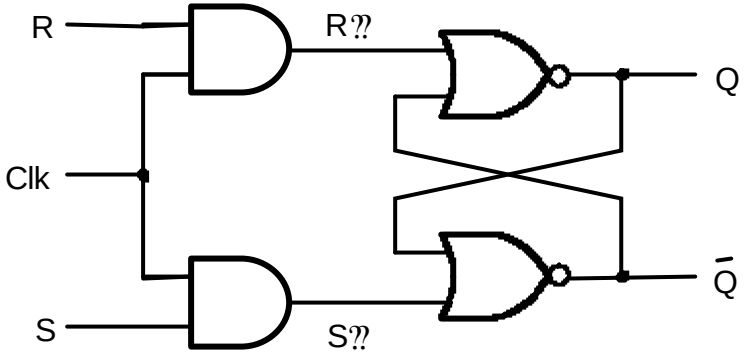
S	R	Q_a	Q_b	
0	0	0/1	1/0	(sem alteração)
0	1	0	1	
1	0	1	0	
1	1	0	0	

(b) Tabela Verdade



(c) Diagrama de Tempo

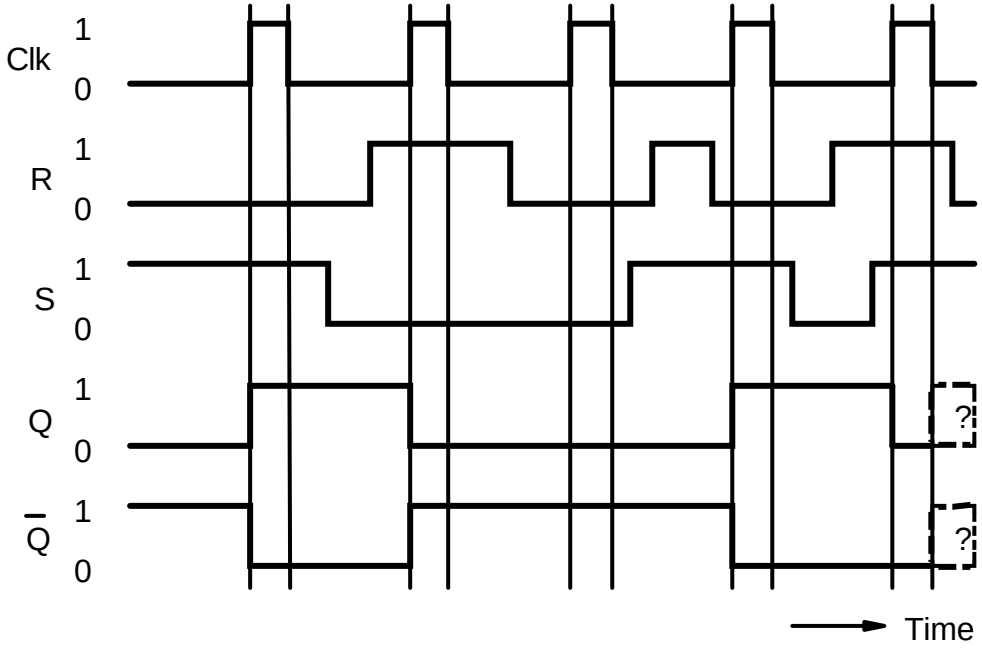
Latch SR com clock (gated)



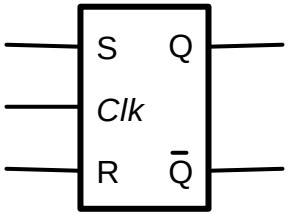
(a) Circuito

Clk	S	R	Q?? + 1??
0	x	x	Q(*t*) (sem alteração)
1	0	0	Q(*t*) (sem alteração)
1	0	1	0
1	1	0	1
1	1	1	x

(b) Tabela Verdade

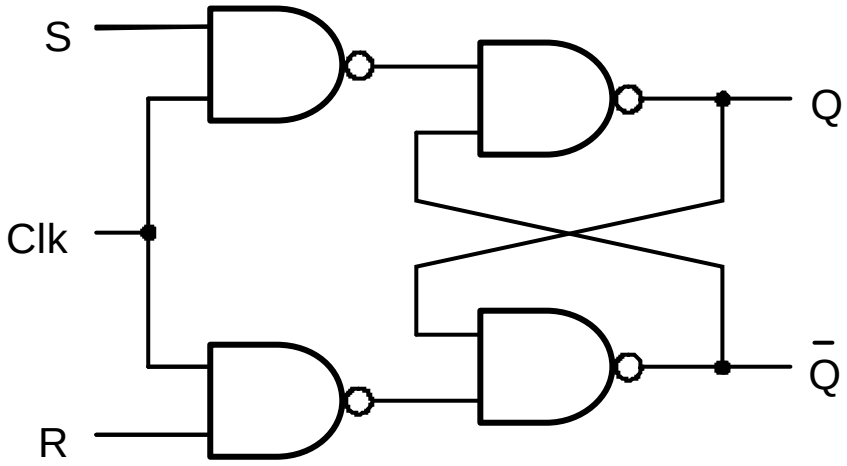


(c) Timing diagram

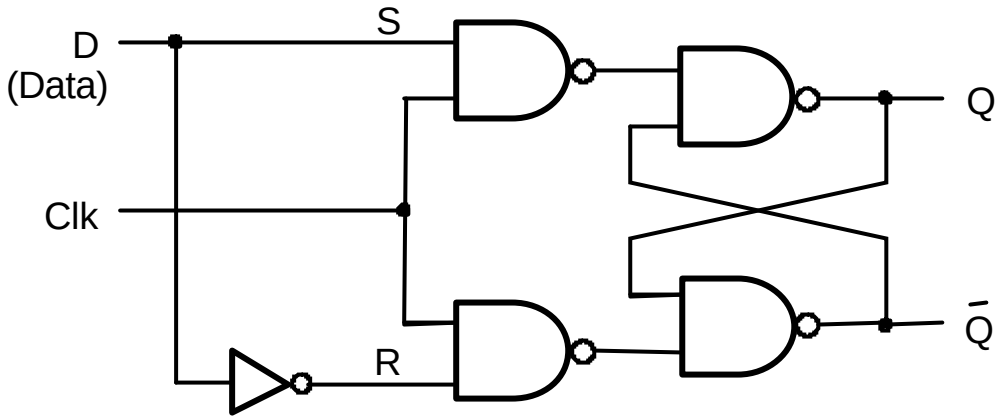


(d) Símbolo Gráfico

Latch SR Gated SR com portas NAND



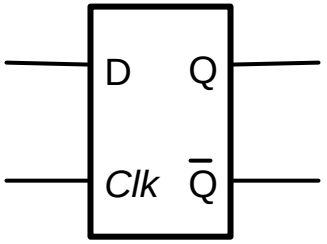
Latch D Gated



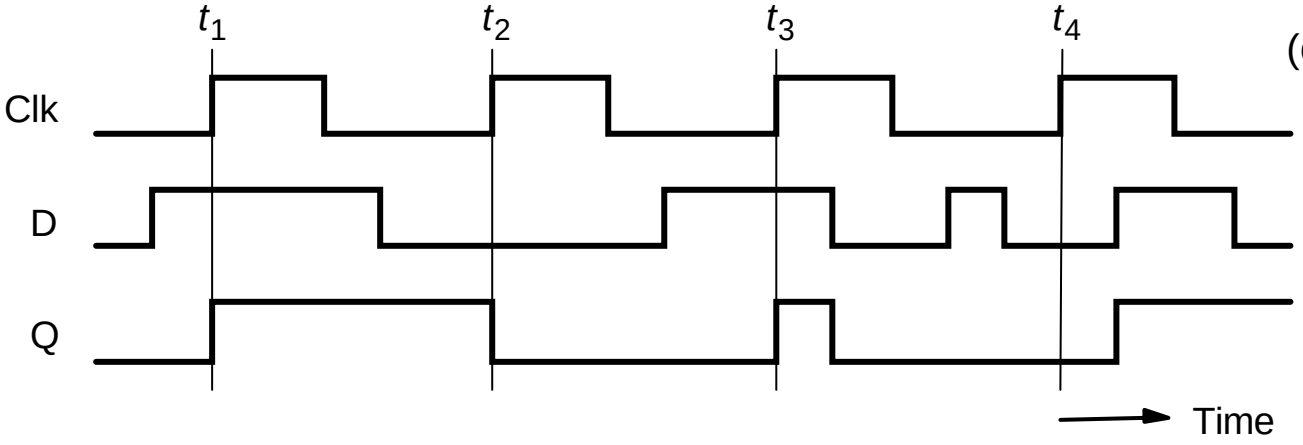
(a) Circuito

Clk	D	Q_{t+1}
0	x	Q_t
1	0	0
1	1	1

(b) Tabela Verdade

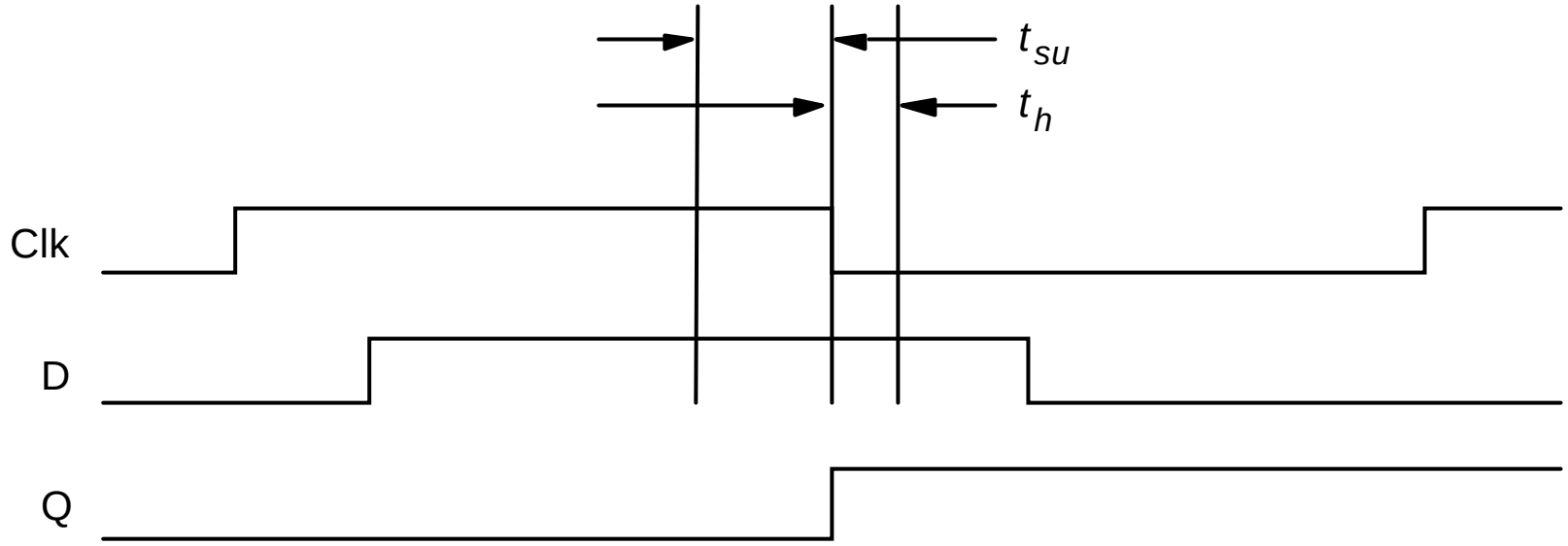


(c) Símbolo Gráfico



(d) Diagrama de Tempo

Setup and hold times

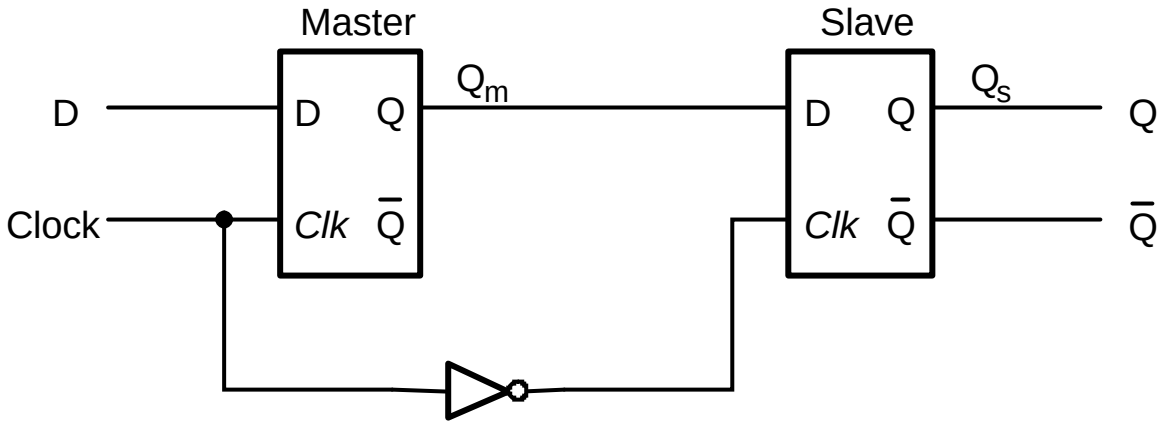


Setup time ✍ tempo mínimo que D deve estar estável antes da descida do clock

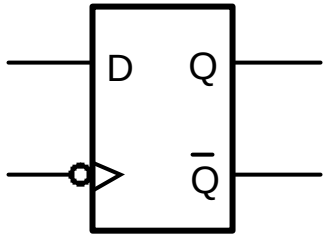
Hold time ✍ temp mínimo que D deve ser mantido estável após a descida do clock

Valores típos (CMOS): t_{su} = 3ns e t_h = 2ns

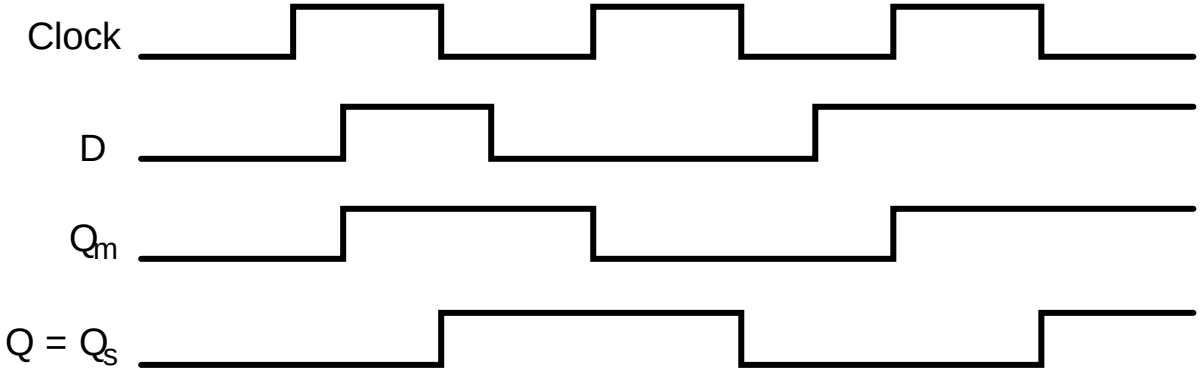
Flip-Flop D Master-slave



(a) Circuito



(c) Símbolo Gráfico

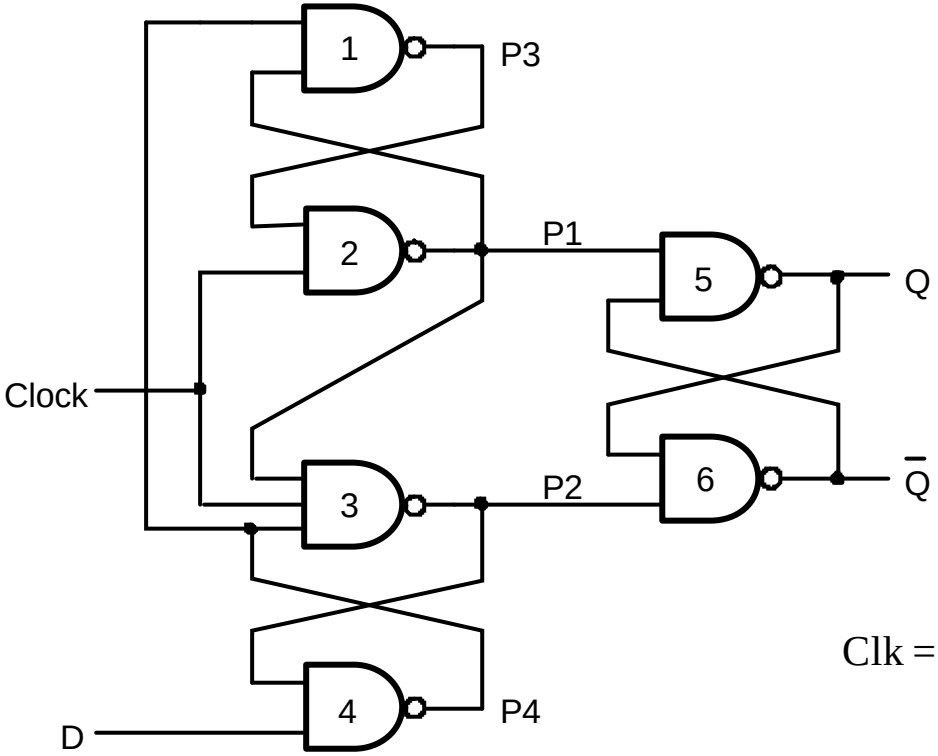


(b) Diagrama de tempo

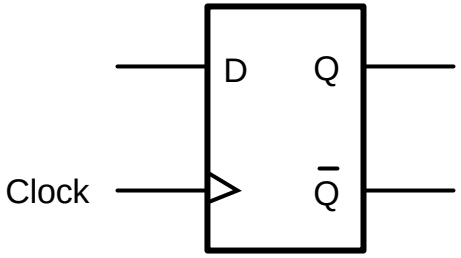
CLK = 1 master armazena
e slave não muda
CLK = 0 master não muda
e slave armazena

Sensível à borda de descida

Flip-Flop D sensível à borda de subida



(a) Circuito



(b) Símbolo Gráfico

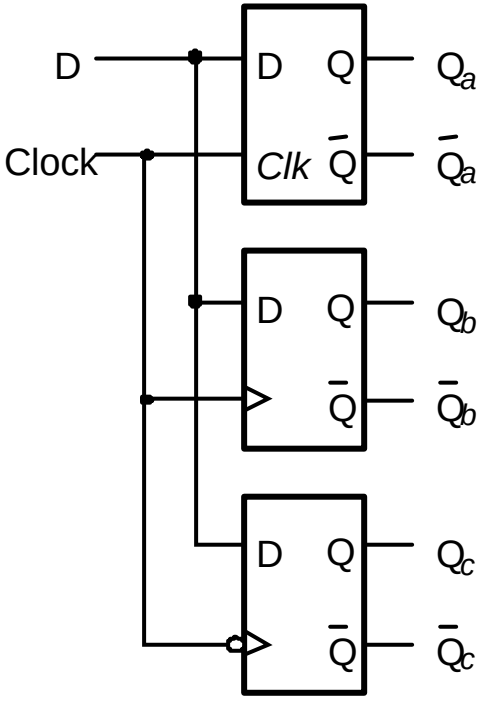
Clk = 0 ✍ P1 = P2 = 1
P3 = D
P4 = D

Clk =1 ✍ P3 e P4 são transmitidos através de 2 e 3

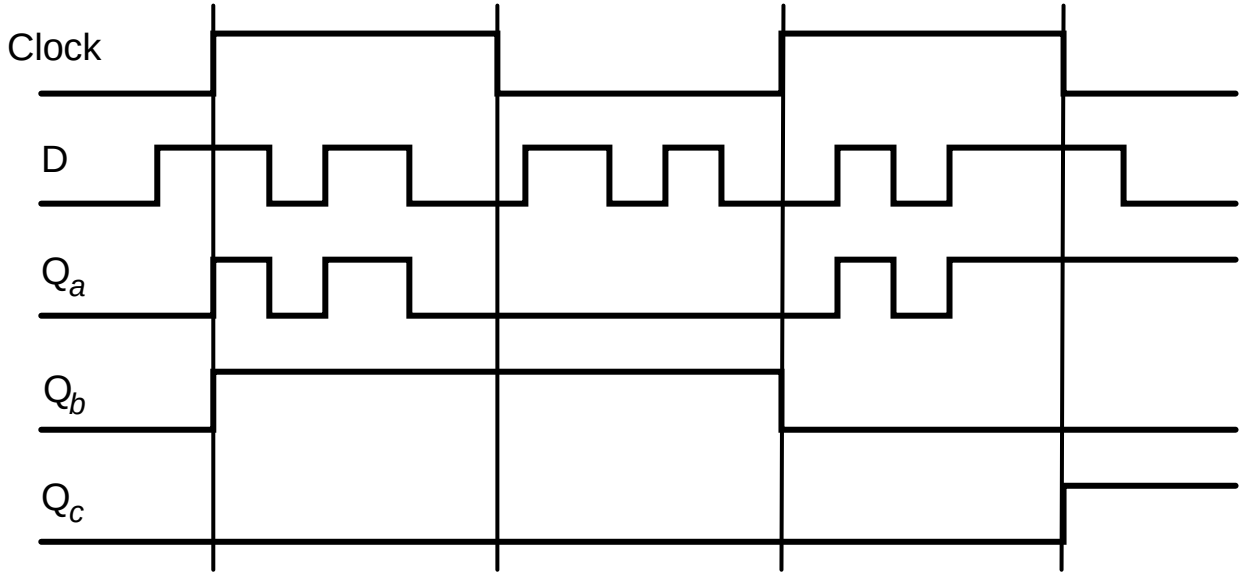
✍P1 = $\overline{D}$ e P2 = D ✍ Q = D e $\overline{Q}$ = $\overline{D}$

Obs - P4 e P3 DEVEM ESTAR ESTÁVEIS QUANDO CLK MUDA PARA 1

Comparação de Flip-Flops D sensíveis a nivel e sensíveis a borda

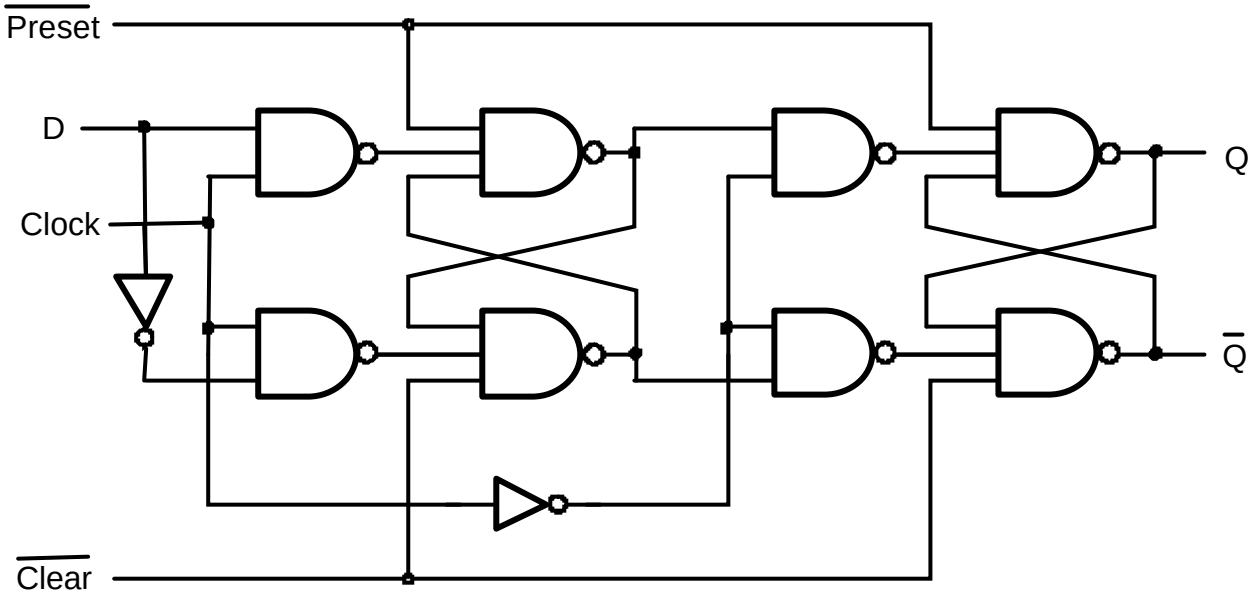


(a) Circuito

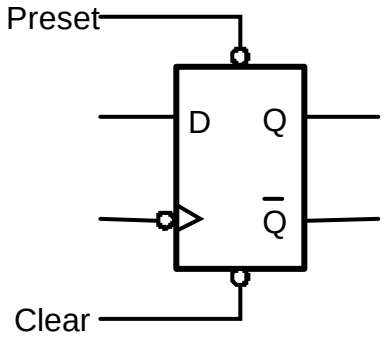


(b) Diagrama de Tempo

Flip-Flop Master-slave tipo D com *Clear* e *Preset*

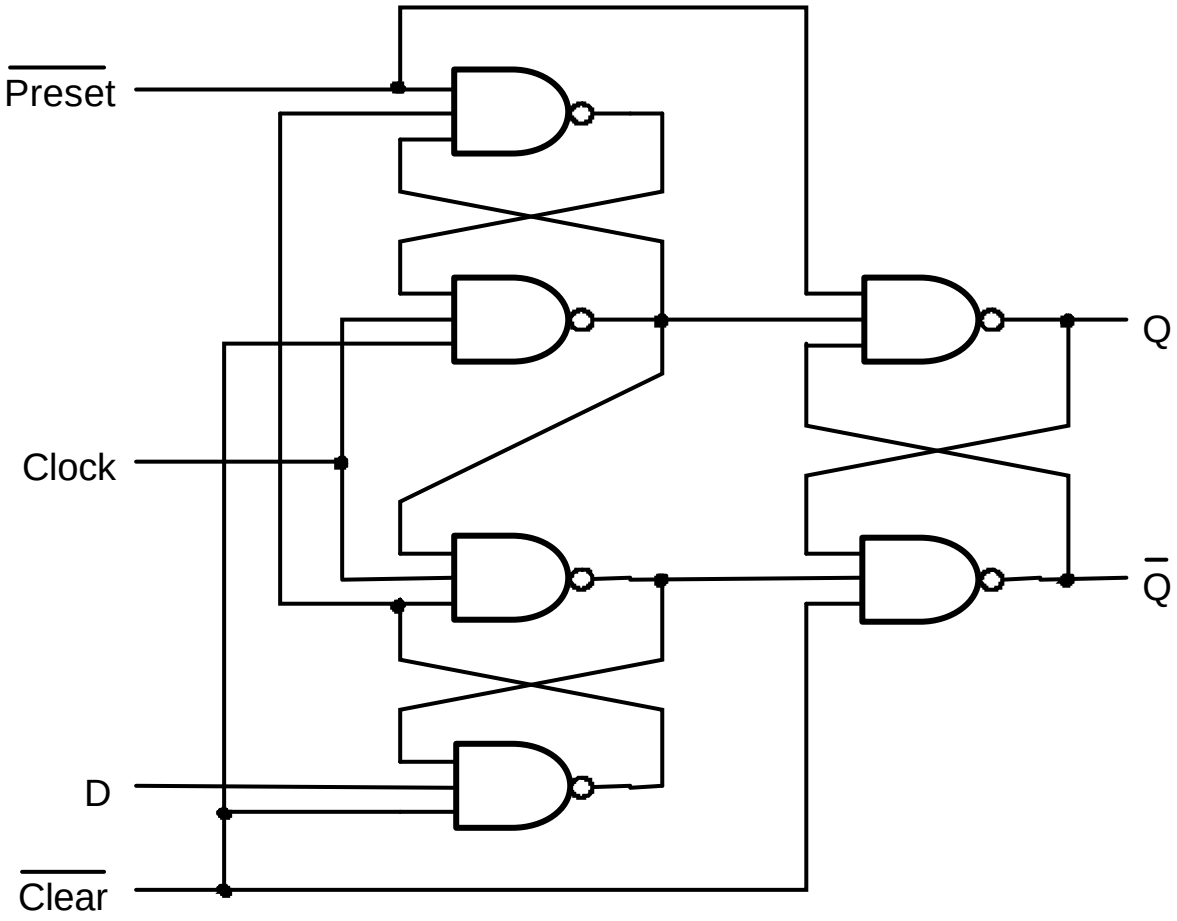


(a) Circuito

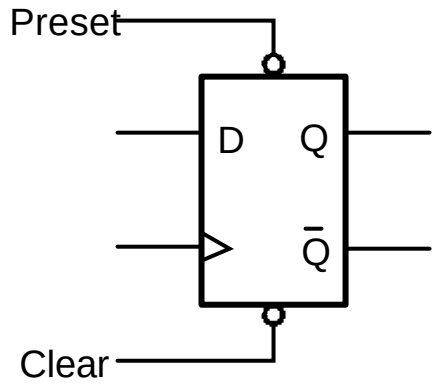


(b) Símbolo Gráfico

Flip-Flop D sensível à borda de subida com *Clear* e *Preset*

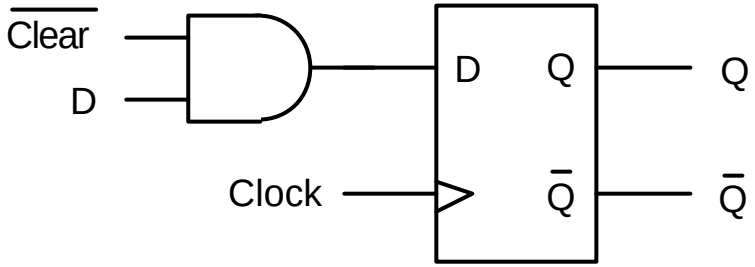


(a) Circuito

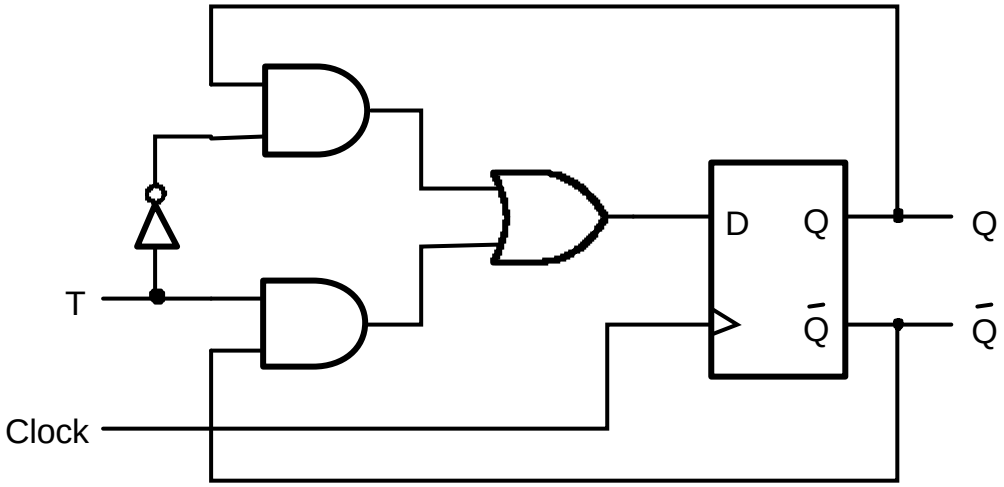


(b) Símbolo Gráfico

Flip-Flop D com reset síncrono



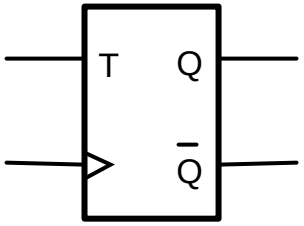
Flip-Flop tipo T (toogle)



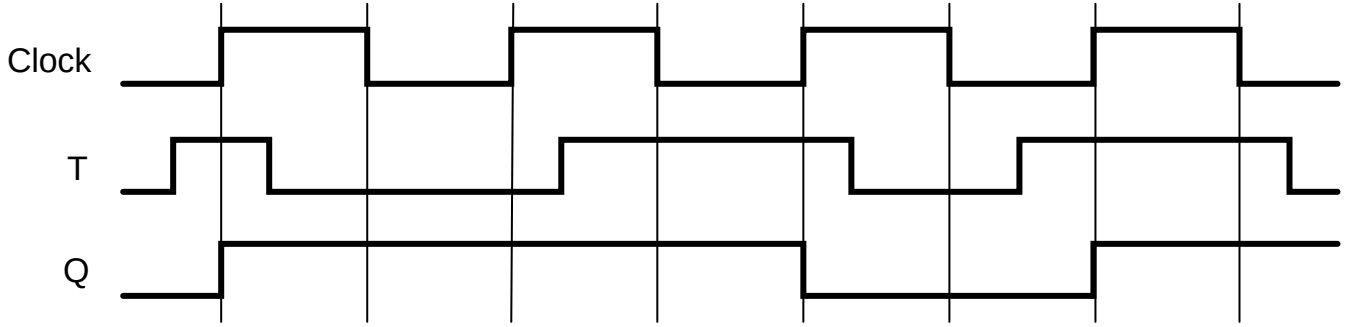
(a) Circuito

T	Q(t+1)
0	Q(t)
1	$\overline{Q}$(t)

(b) Tabela Verdade

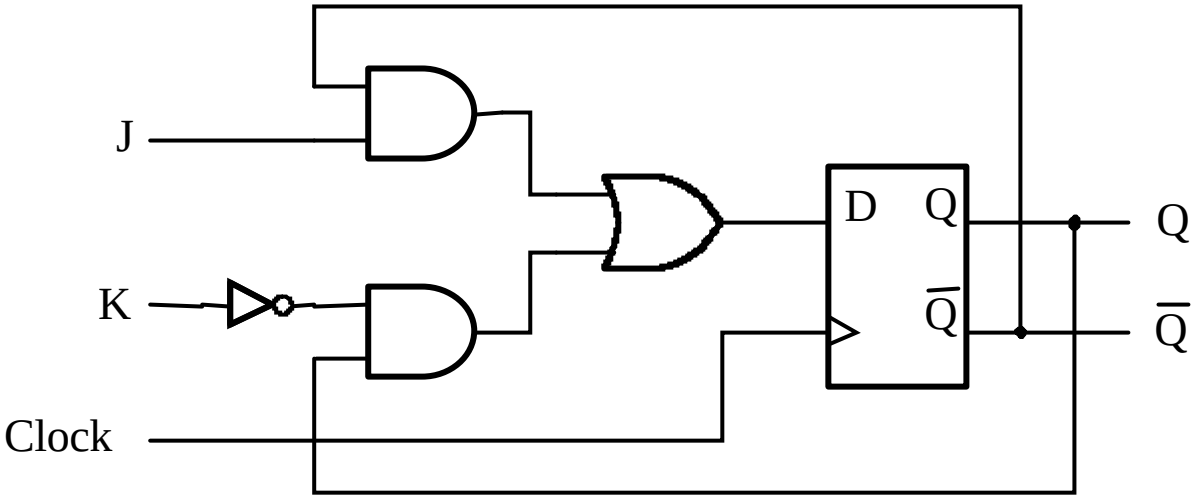


(c) Símbolo Gráfico



(d) Diagrama de tempo

Flip-Flop JK



(a) Circuito

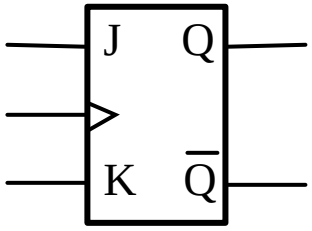
$D = J\overline{Q} + \overline{K}Q$

J e K ✍ entradas
De controle

FFs SR e T juntos

J	K	Q (t+1)
0	0	Q (t)
0	1	0
1	0	1
1	1	$\overline{Q}$ (t)

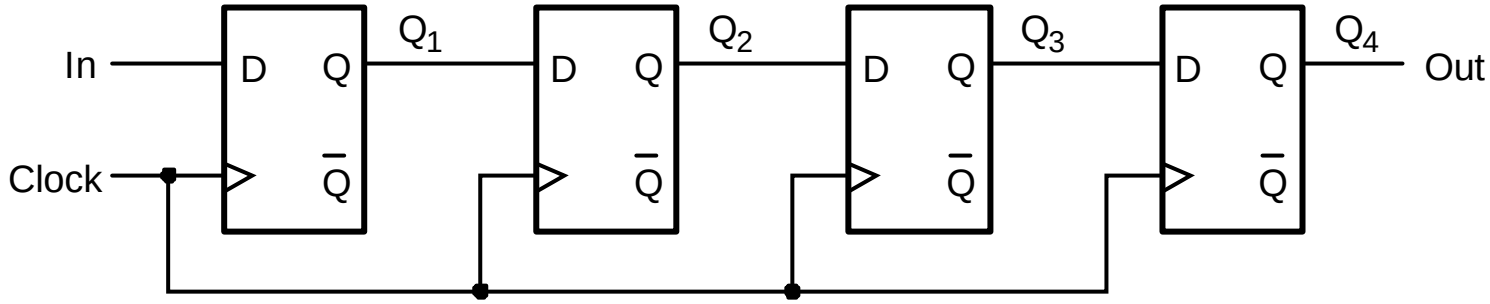
(b) Tabela Verdade



(c) Símbolo Gráfico

Registrador de deslocamento com entrada e saída serial

Registrador é um conjunto de n Flip-Flops



(a) Circuito

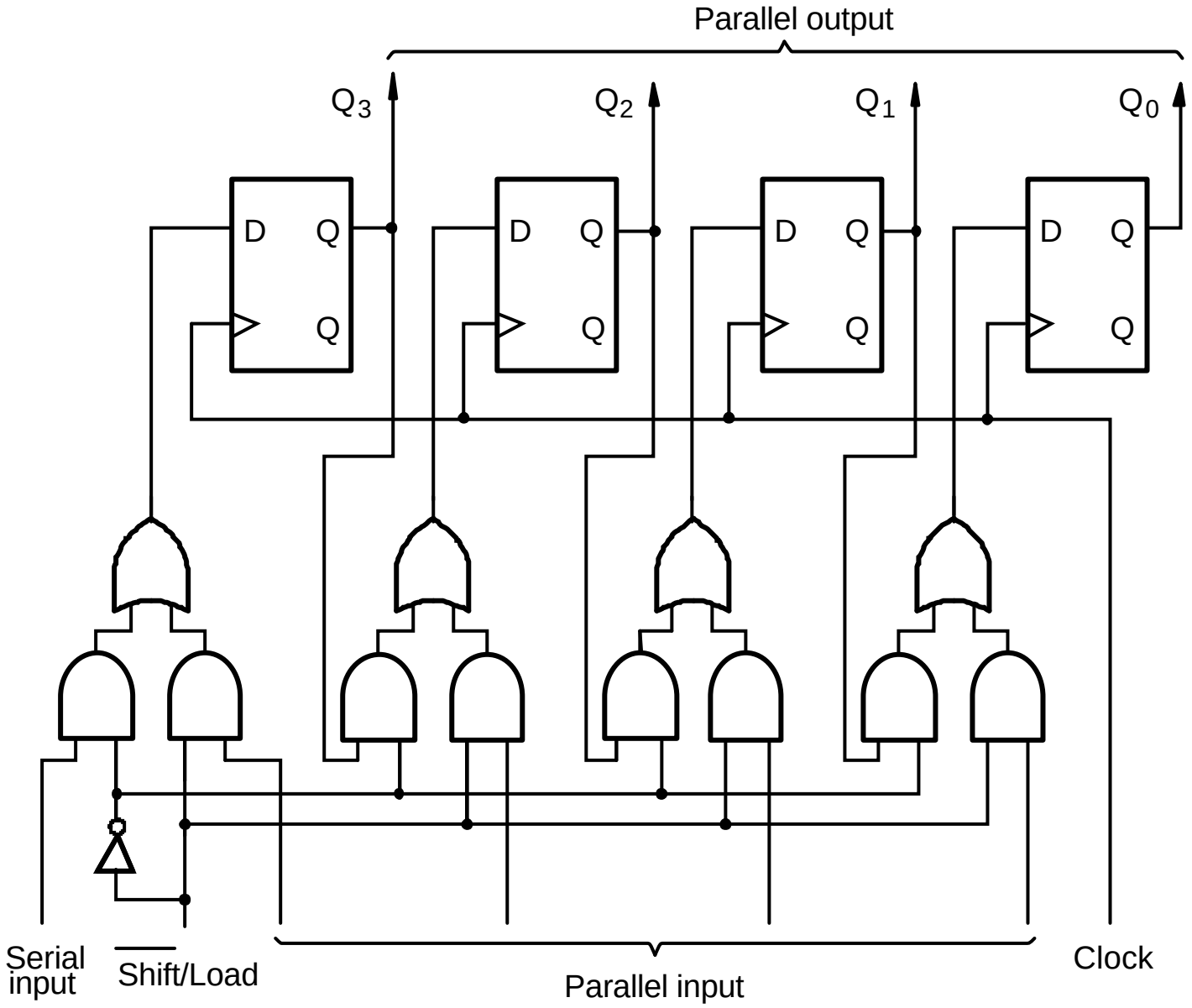
	In	Q_1	Q_2	Q_3	Q_4 = Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

(b) Exemplo de uma seqüência

FFs master-slave
ou sensível à borda.

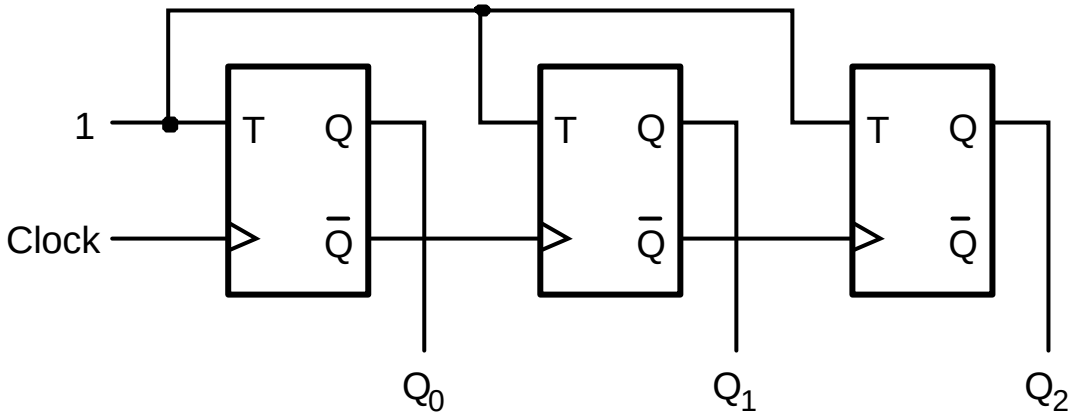
Porque não
sensível a nível ?

Registrador de deslocamento com entrada paralela e serial e saída paralela

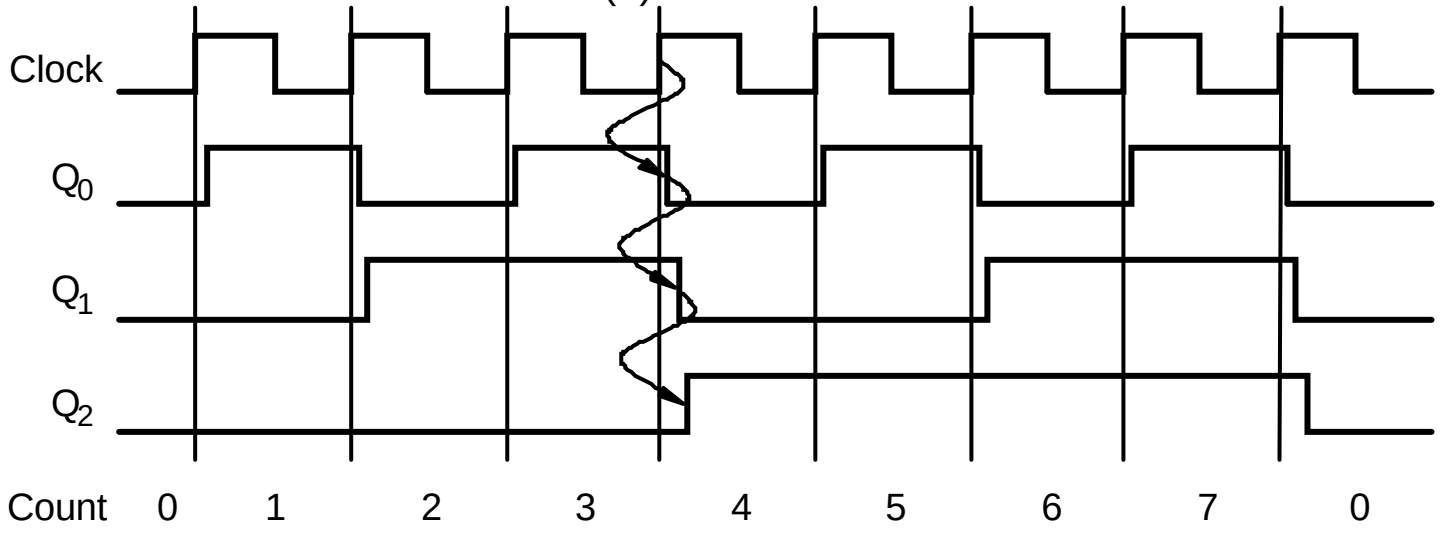


Contadores

Contador de 3 bits up-counter

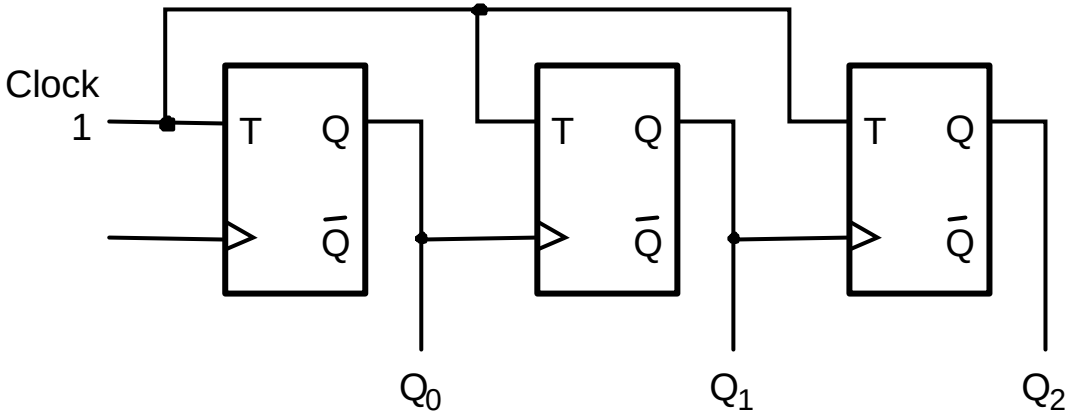


(a) Circuito

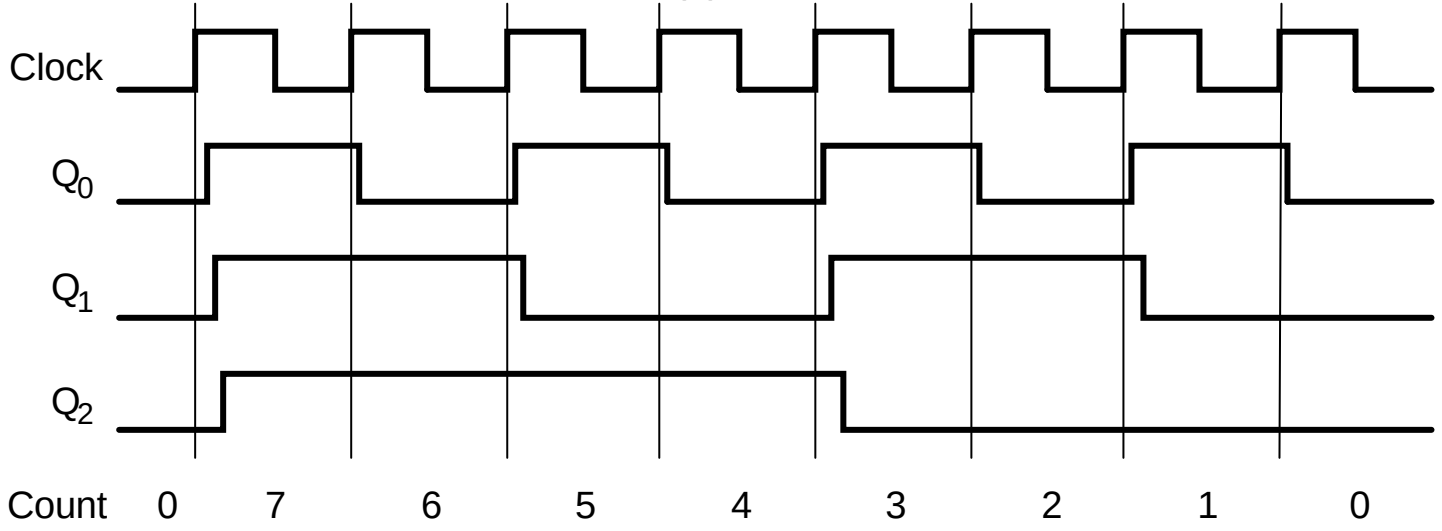


(b) Diagrama de tempo

Contador de 3 bits down-counter



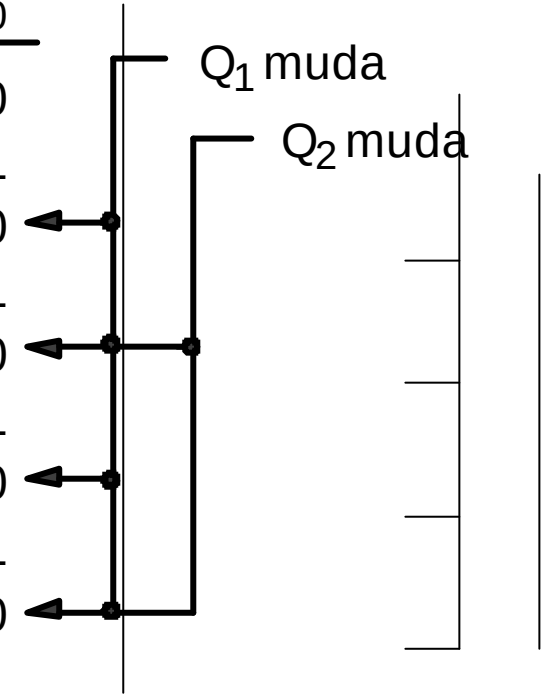
(a) Circuit



(b) Timing diagram

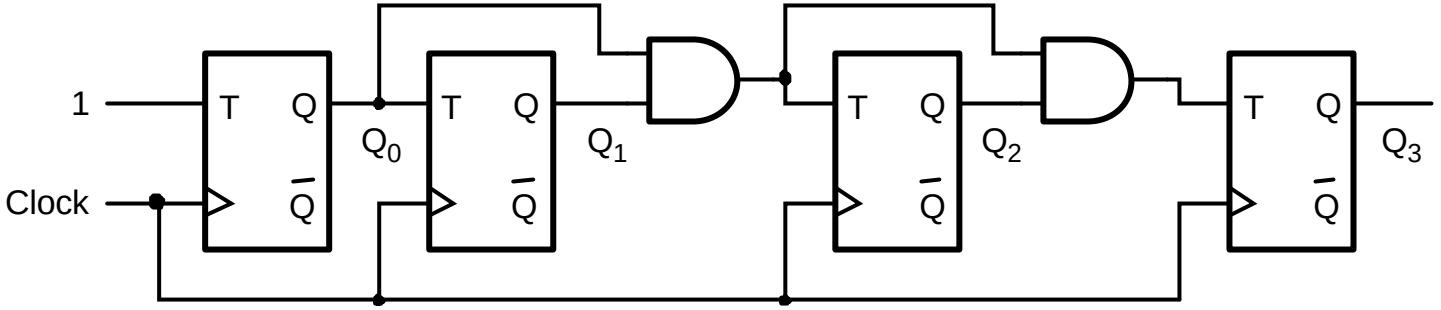
Derivação de contador síncrono up-counter

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

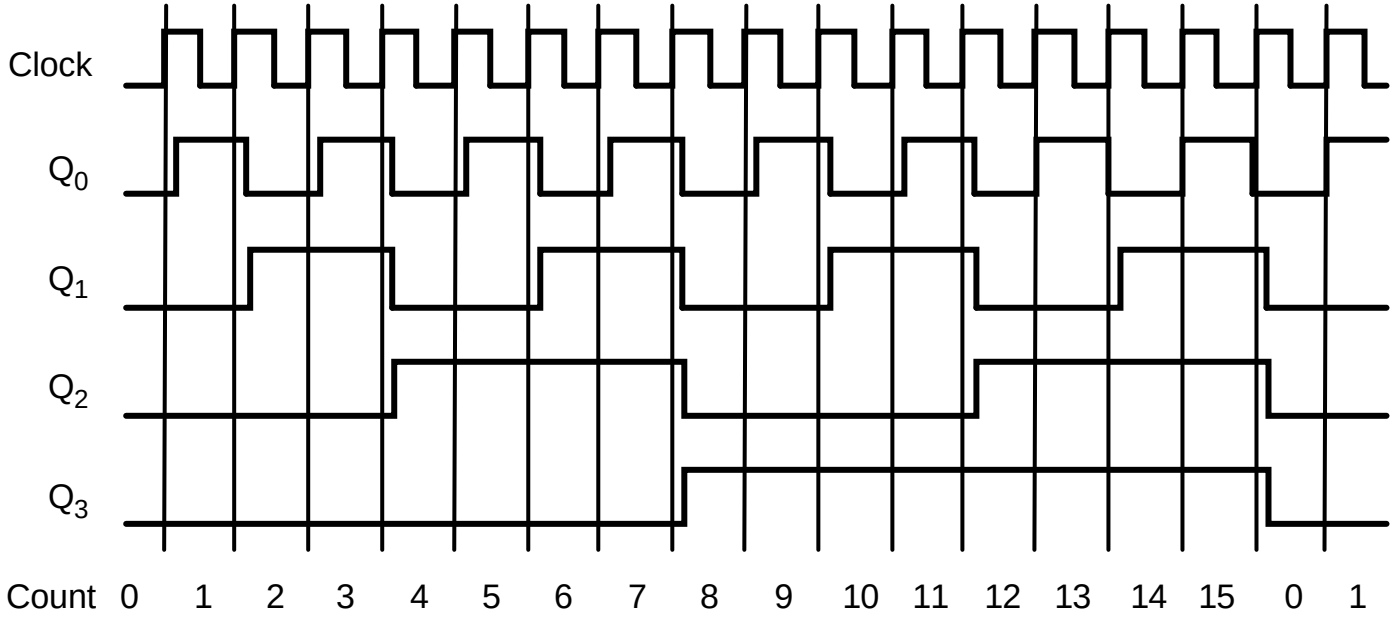


T0 = 1
T1 = Q0
T2 = Q0 Q1
T3 = Q0 Q1 Q2
Tn = Q0 Q1 ... Qn

Contador síncrocno de quatro-bits up-counter

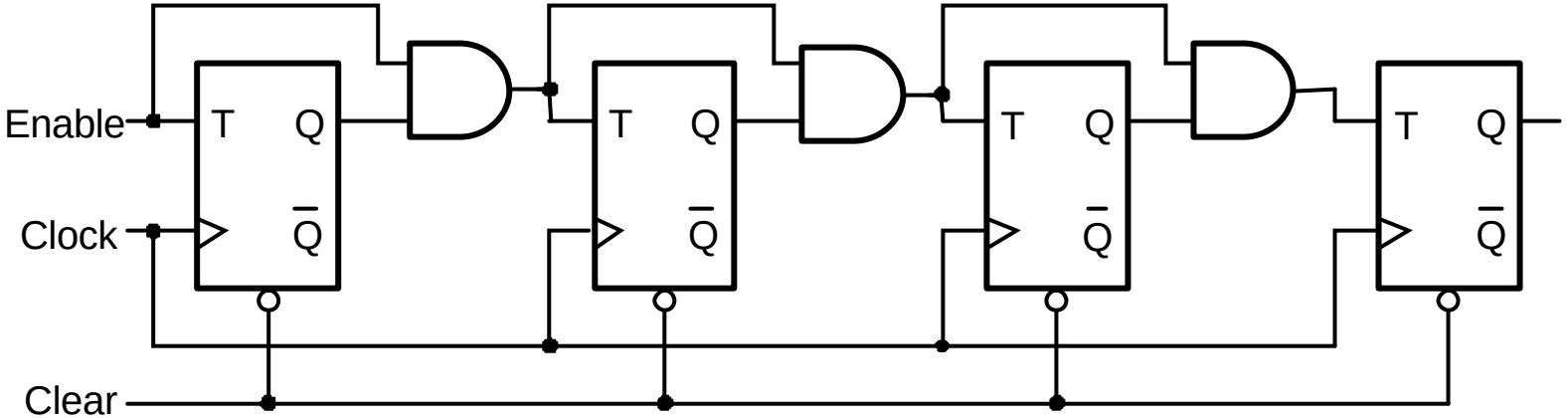


(a) Circuito

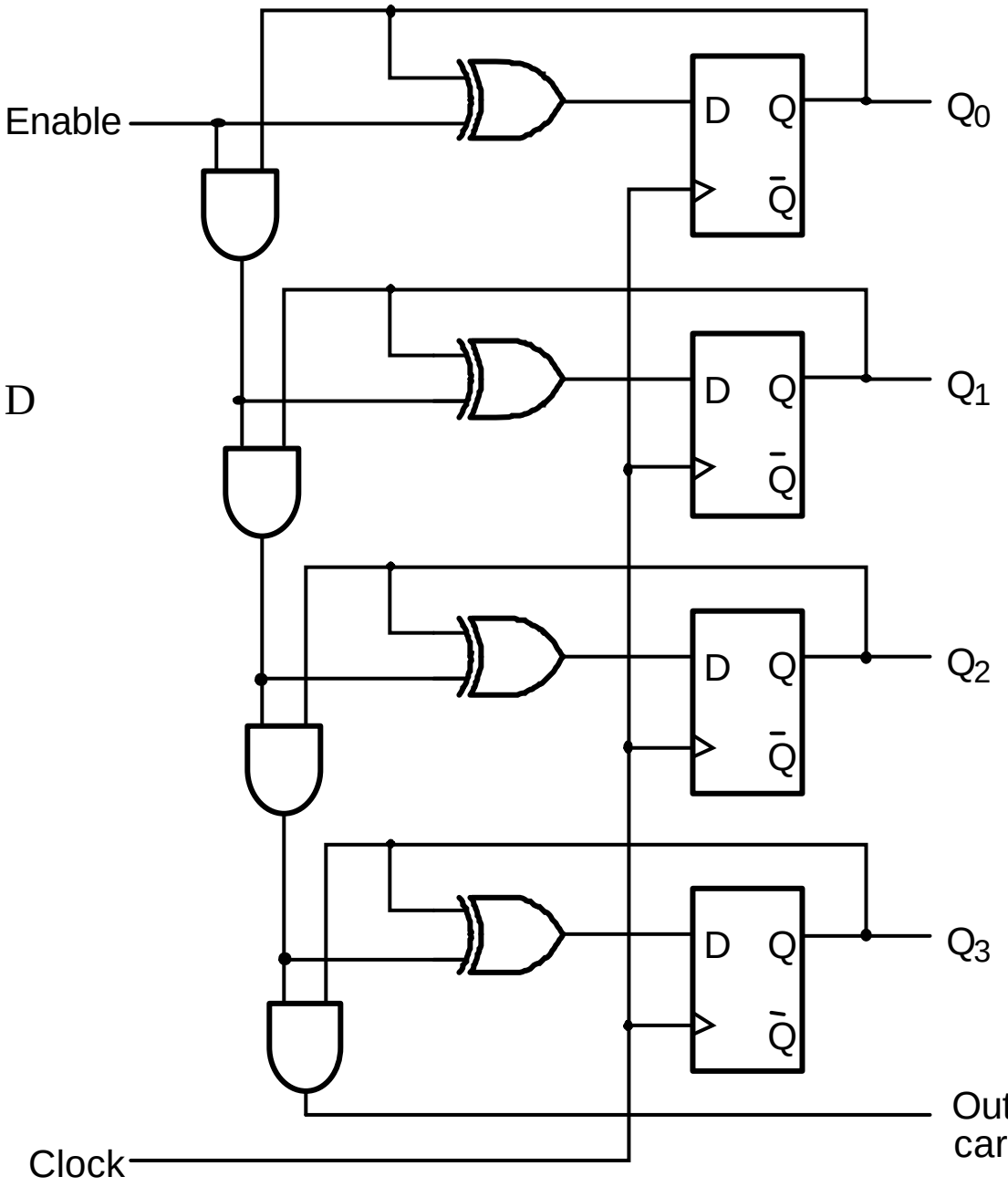


(b) Diagrama de tempo

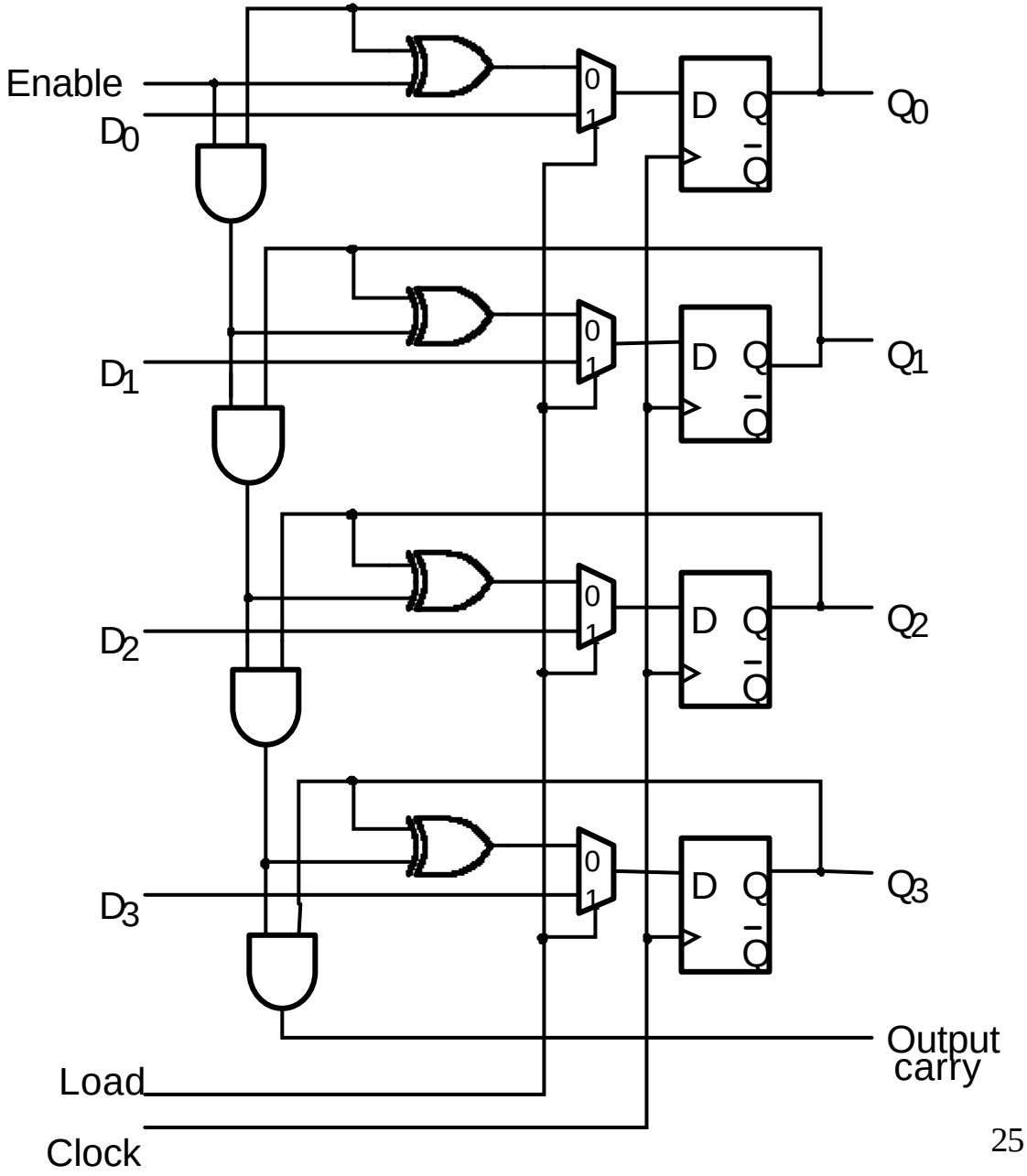
Inclusão de sinais de enable e clear



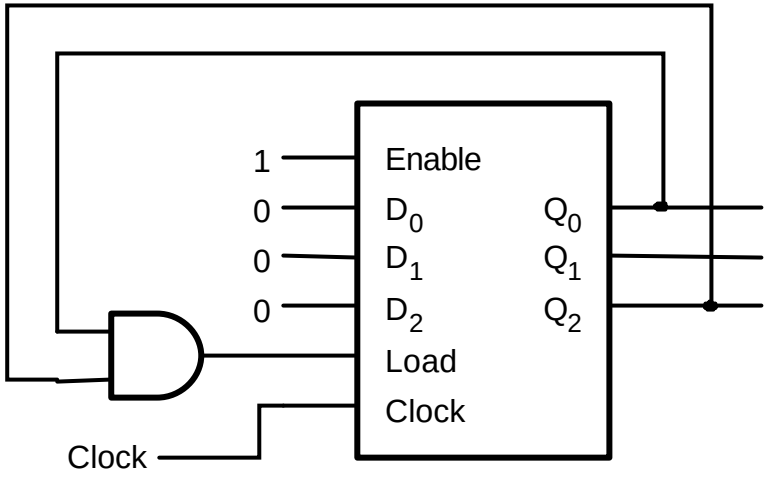
Contador de 4 bits com FF D



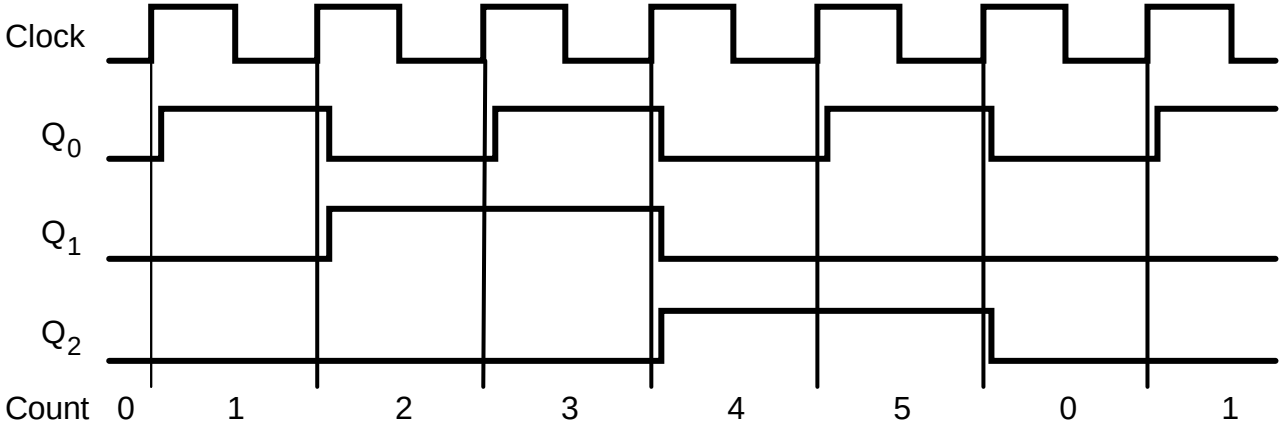
Contador com entrada paralela



Contador modulo-6 com reset síncrono

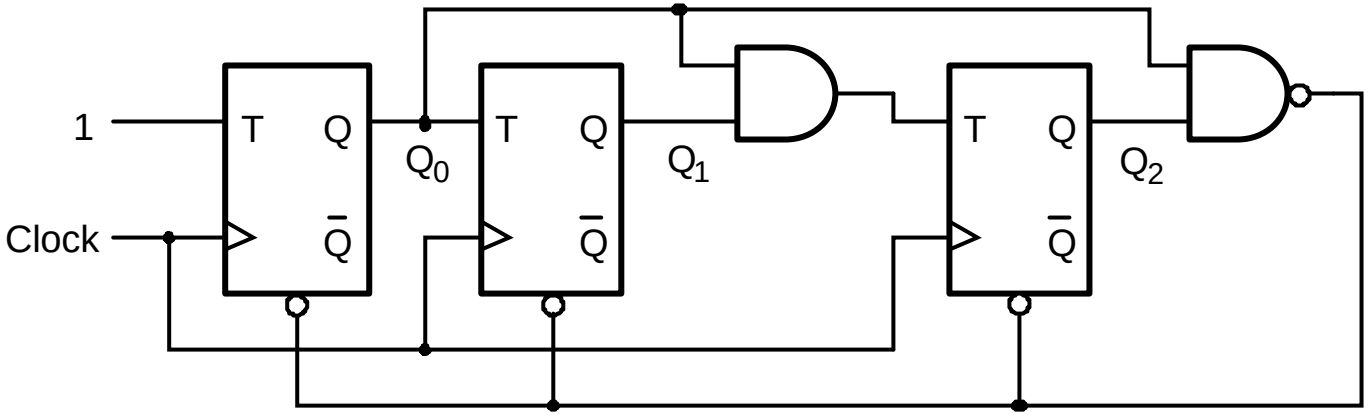


(a) Circuito

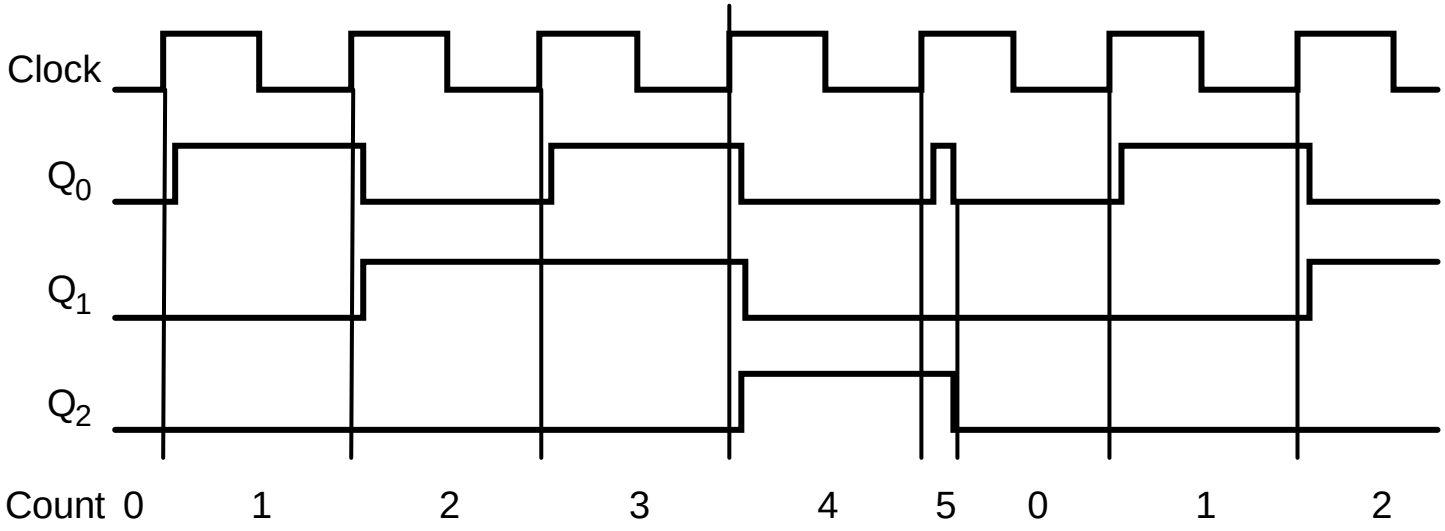


(b) Diagrama de tempo

Contador modulo-6 com reset síncrono

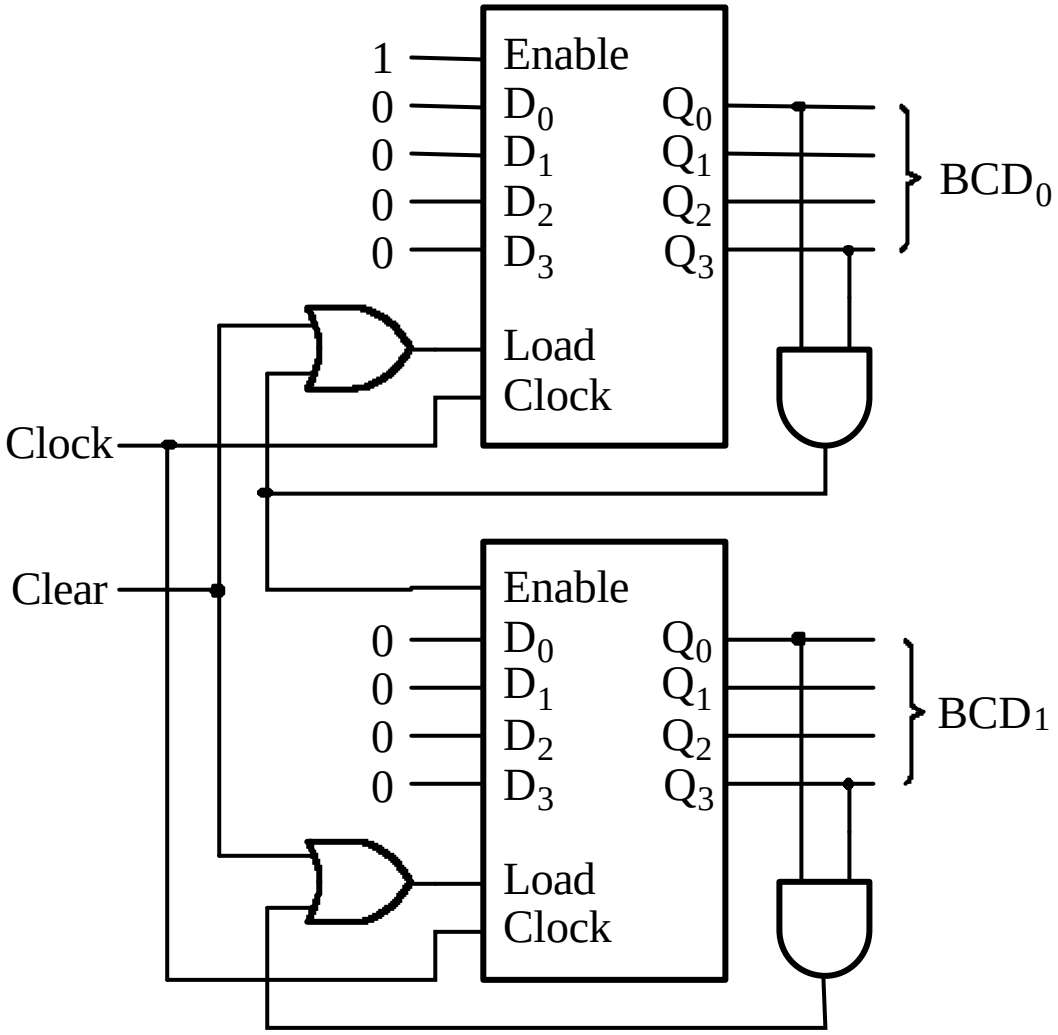


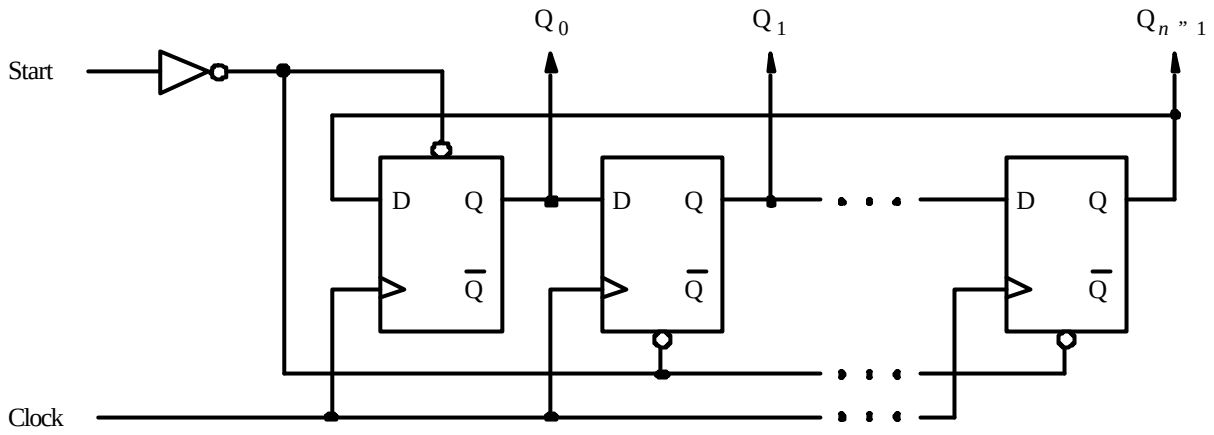
(a) Circuito



(b) Diagrama de tempo

Contador BCD de 2 dígitos

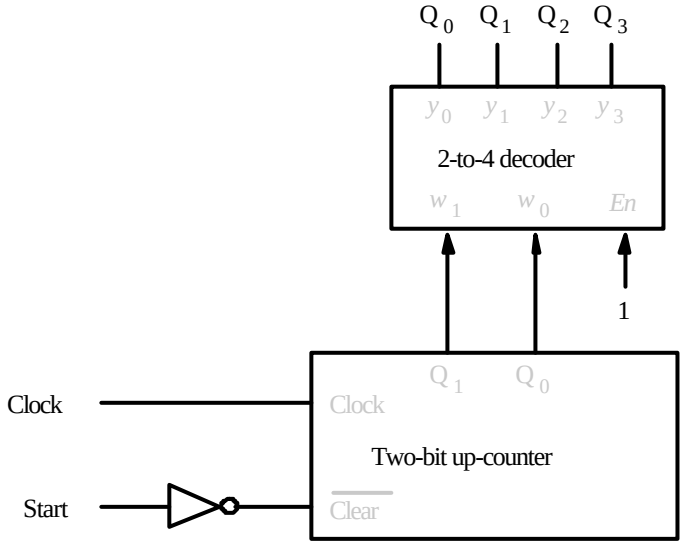




(a) An *n*-bit ring counter

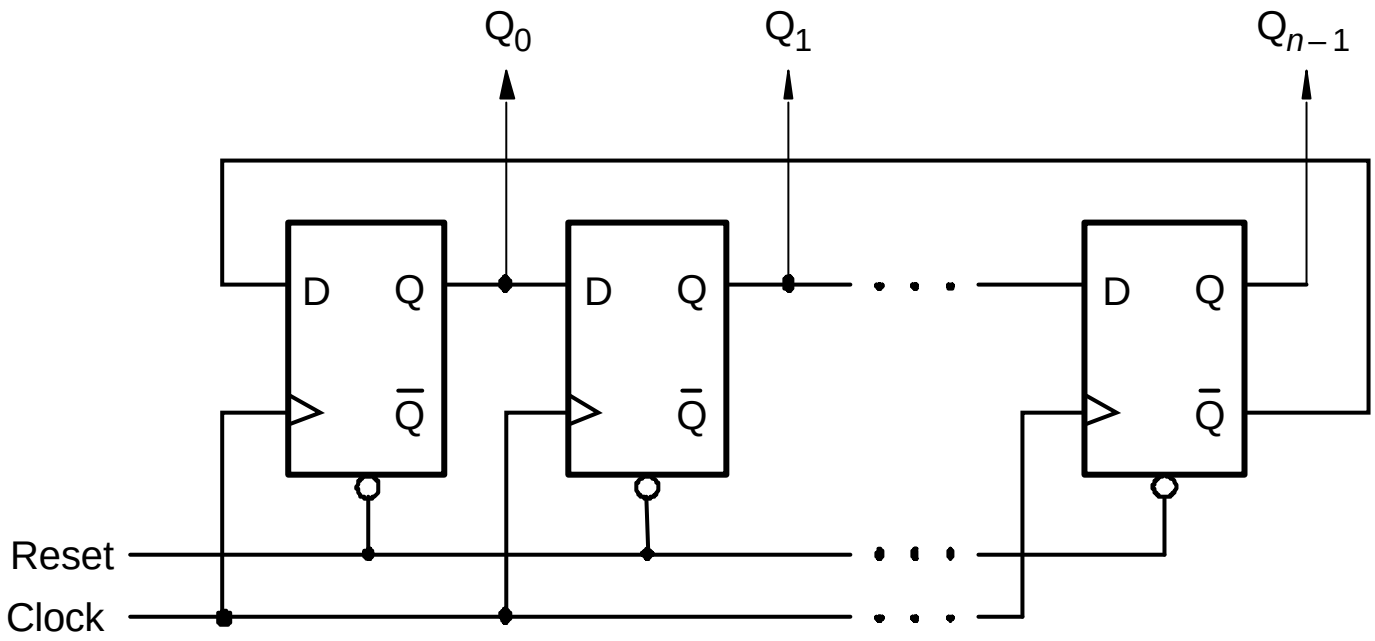
Contador em Anel

1000, 0100, 0010, 0001



(b) A four-bit ring counter

Contador Johnson



Inatalação de um FF D de um package

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
LIBRARY altera ;
USE altera.maxplus2.all ;

ENTITY flipflop IS
      PORT (   D, Clock                  : IN      STD_LOGIC ;
               Resetn, Presetn           : IN      STD_LOGIC ;
               Q                         : OUT     STD_LOGIC ) ;
END flipflop ;

ARCHITECTURE Structure OF flipflop IS
BEGIN
      dff_instance: dff PORT MAP ( D, Clock, Resetn, Presetn, Q ) ;
END Structure ;
```

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

ENTITY implied IS
      PORT (   A, B       : IN      STD_LOGIC ;
               AeqB       : OUT     STD_LOGIC ) ;
END implied ;

ARCHITECTURE Behavior OF implied IS
BEGIN
      PROCESS ( A, B )
      BEGIN
            IF A = B THEN
                  AeqB <= '1' ;
            END IF ;
      END PROCESS ;
END Behavior ;
```

Codigo para um latch D gated

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

ENTITY latch IS
      PORT (   D, Clk     : IN        STD_LOGIC ;
               Q          : OUT       STD_LOGIC) ;
END latch ;

ARCHITECTURE Behavior OF latch IS
BEGIN
      PROCESS ( D, Clk )
      BEGIN
            IF Clk = '1' THEN
                  Q <= D ;
            END IF ;
      END PROCESS ;
END Behavior ;
```

Código para um FF D

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

ENTITY flipflop IS
      PORT (   D, Clock  : IN      STD_LOGIC ;
               Q         : OUT   STD_LOGIC) ;
END flipflop ;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
      PROCESS ( Clock )
      BEGIN
           IF Clock'EVENT AND Clock = '1' THEN
                Q <= D ;
           END IF ;
      END PROCESS ;
END Behavior ;
```

Código para um FF D usando WAIT UNTIL

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

ENTITY flipflop IS
      PORT (  D, Clock  : IN       STD_LOGIC ;
              Q         : OUT      STD_LOGIC ) ;
END flipflop ;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
      PROCESS
      BEGIN
            WAIT UNTIL Clock'EVENT AND Clock = '1' ;
            Q <= D ;
      END PROCESS ;
END Behavior ;
```

FF D com reset assíncrono

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

ENTITY flipflop IS
      PORT (   D, Resetn, Clock    : IN       STD_LOGIC ;
               Q                   : OUT      STD_LOGIC) ;
END flipflop ;

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

FF D com reset síncrono

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

ENTITY flipflop IS
      PORT (   D, Resetn, Clock    : IN       STD_LOGIC ;
               Q                   : OUT      STD_LOGIC) ;
END flipflop ;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
      PROCESS
      BEGIN
            WAIT UNTIL Clock'EVENT AND Clock = '1' ;
            IF Resetn = '0' THEN
                  Q <= '0' ;
            ELSE
                  Q <= D ;
            END IF ;
      END PROCESS ;
END Behavior ;
```

Instanciação do módulo *lpm_shiftreg*

```
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 ;
```

Código um registrador de 8 bits com clear assíncrono

```
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 ;
```

Código um registrador de *n*-bit com clear assíncrono

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

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

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

Código para um FF D com um multiplexador 2-para-1 na entrada *D*

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

ENTITY muxdff IS
    PORT (  D0, D1, Sel, Clock  : IN    STD_LOGIC ;
            Q                   : OUT   STD_LOGIC ) ;
END muxdff ;

ARCHITECTURE Behavior OF muxdff IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF Sel = '0' THEN
            Q <= D0 ;
        ELSE
            Q <= D1 ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Código hierárquico para um shift-register de 4 bits

```
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 ;
```

Alternativa para o 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 ;
```

Contador up-counter quatro-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 ;
```

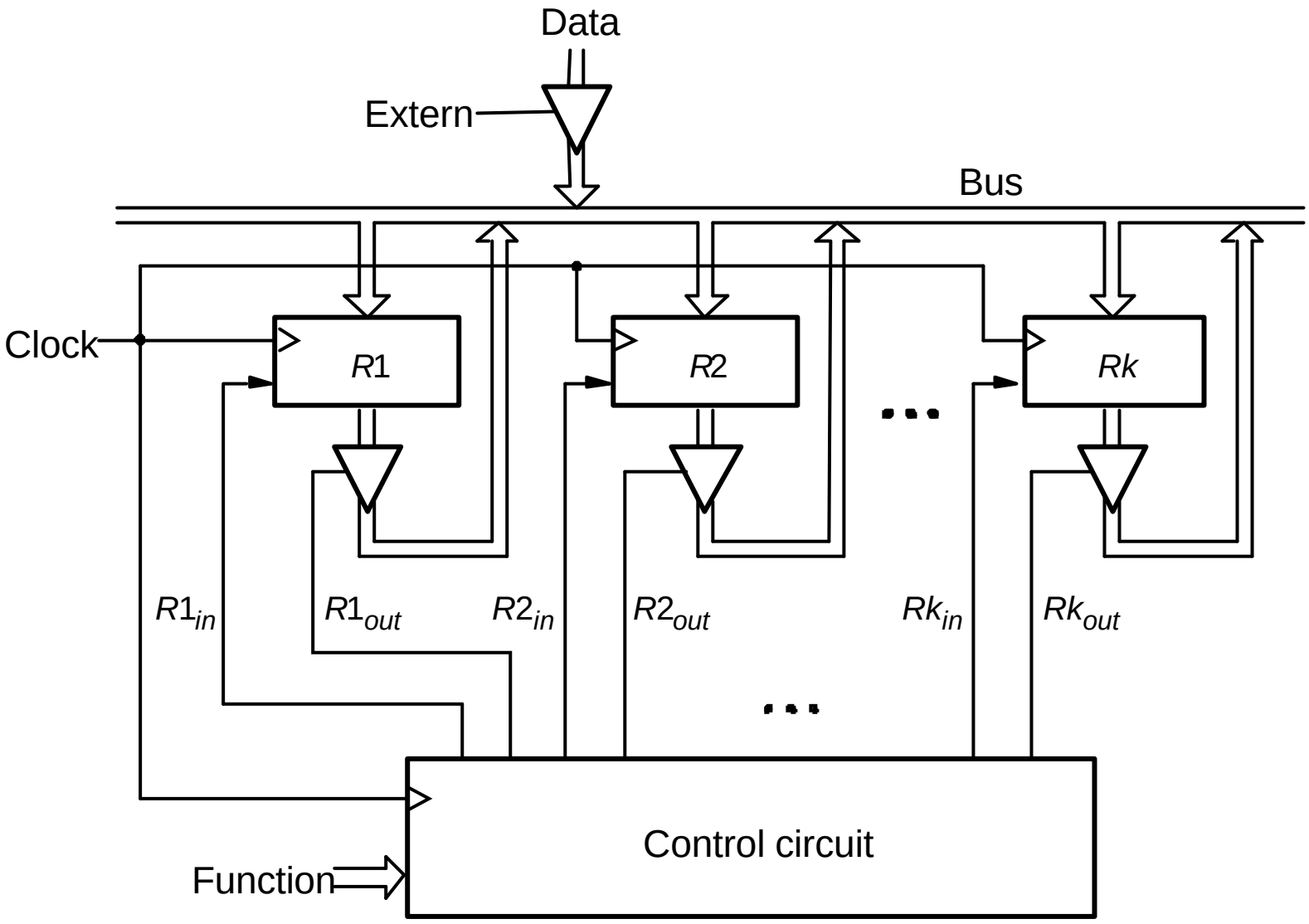
Contador de 4 bits com carga paralela usando sinais INTEGER

```
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;
```

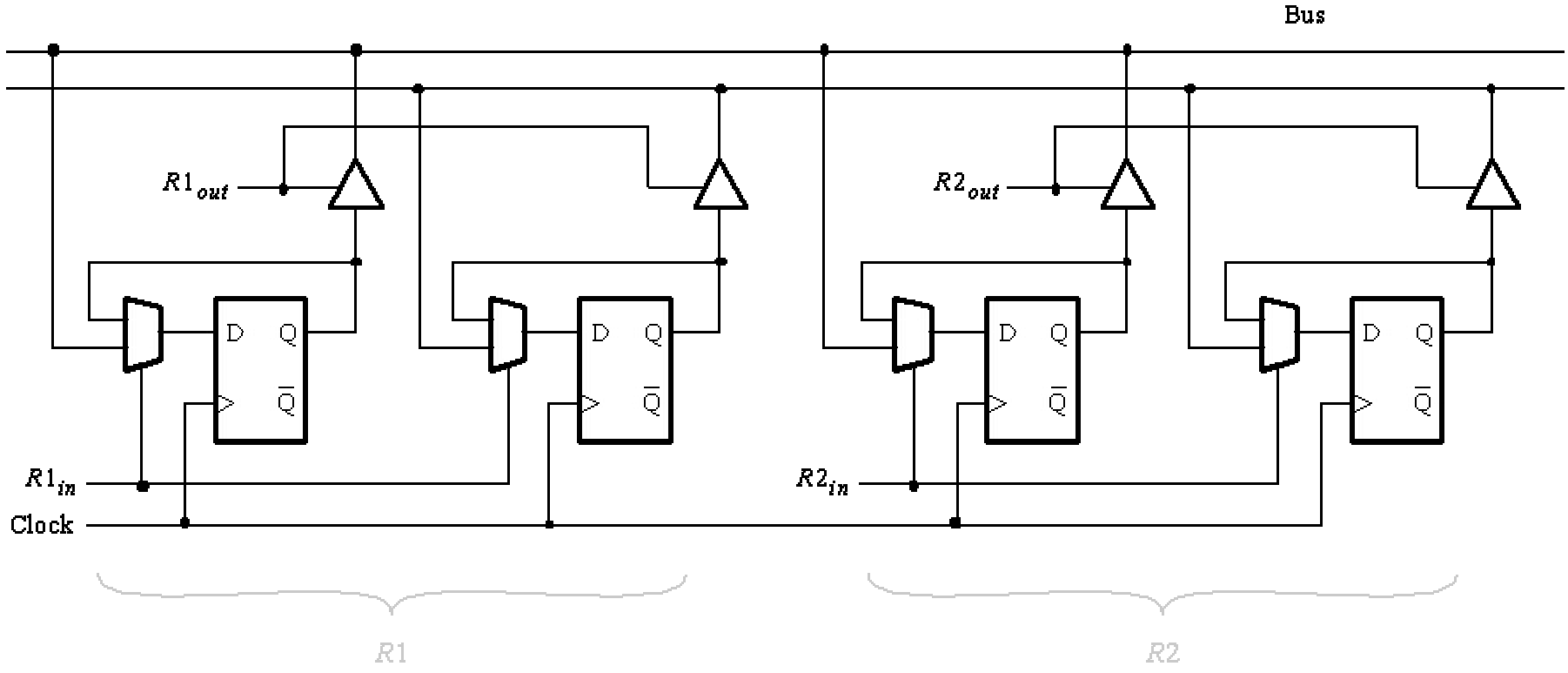
Contador down-counter

```
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 ;
                        ELSE
                                Count <= Count-1 ;
                        END IF ;
                END IF ;
        END PROCESS;
        Q <= Count ;
END Behavior ;
```

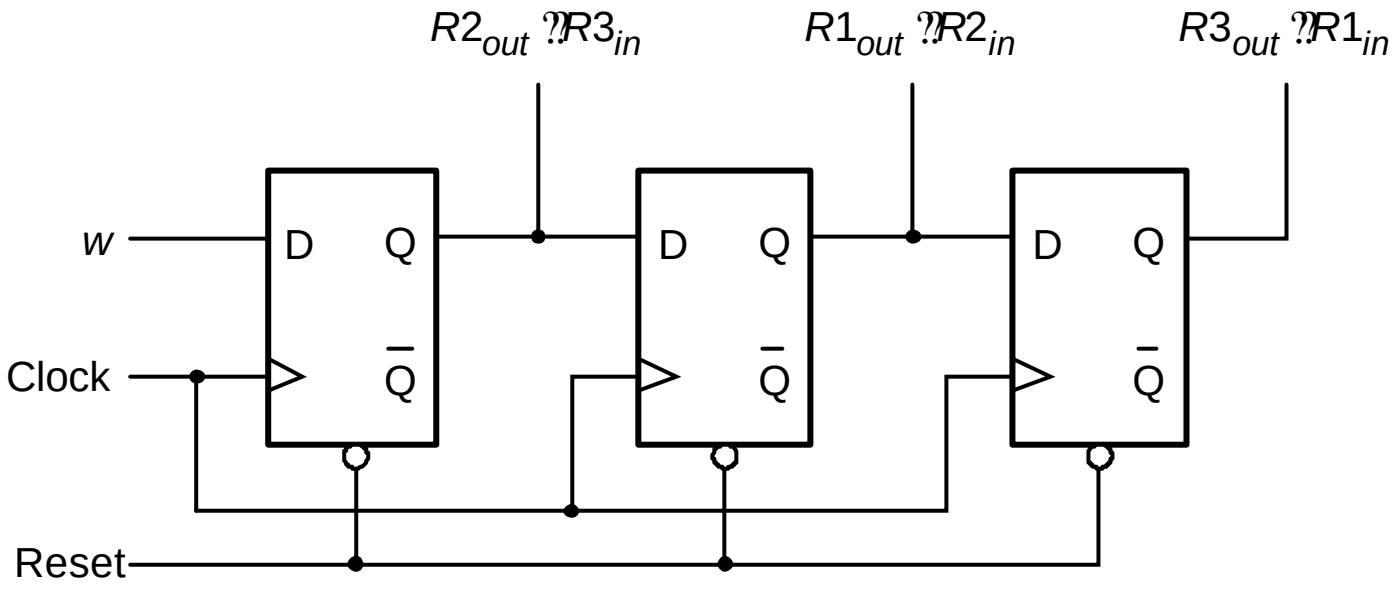
Sistema digital com *k* registradores



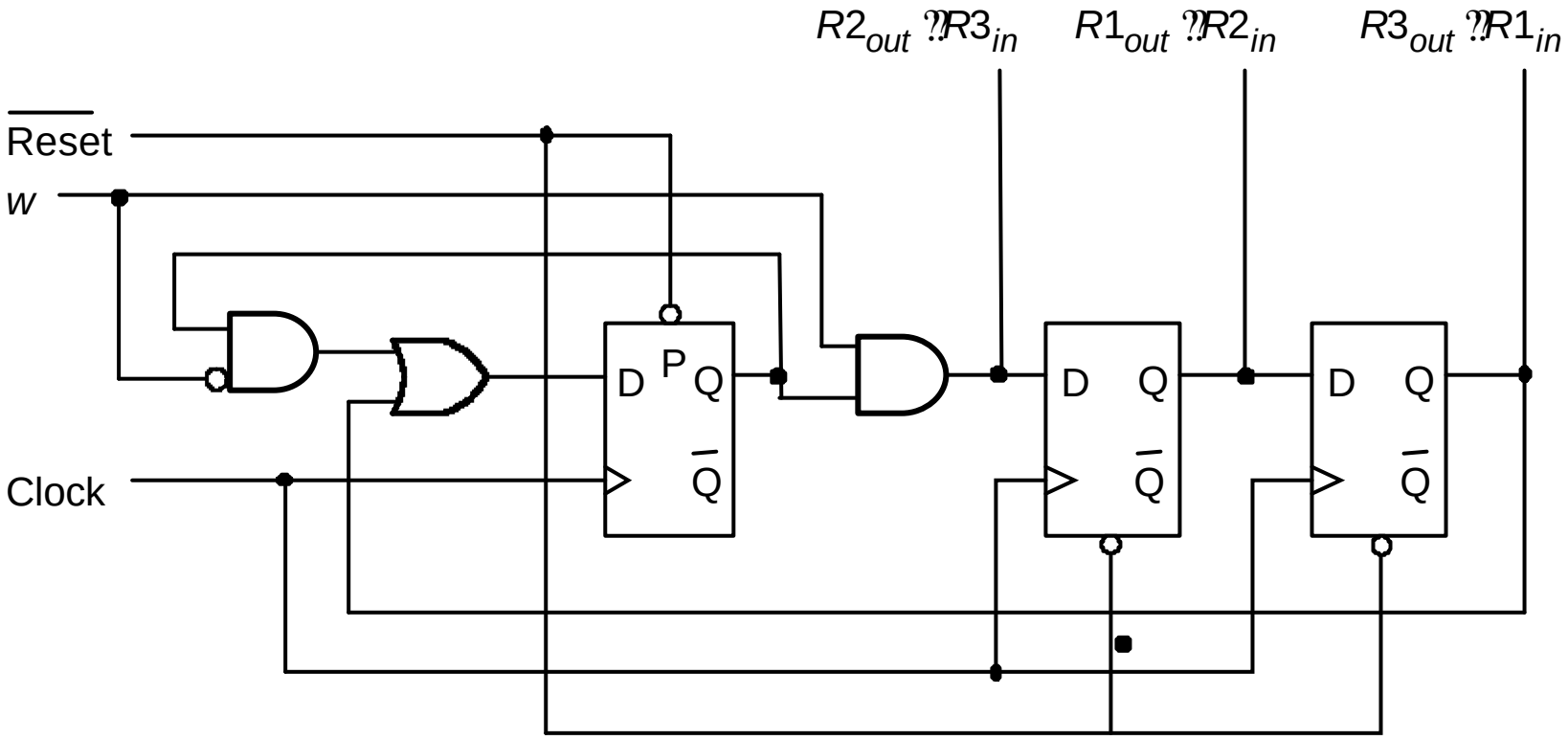
Conexão dos registradoes ao barramento



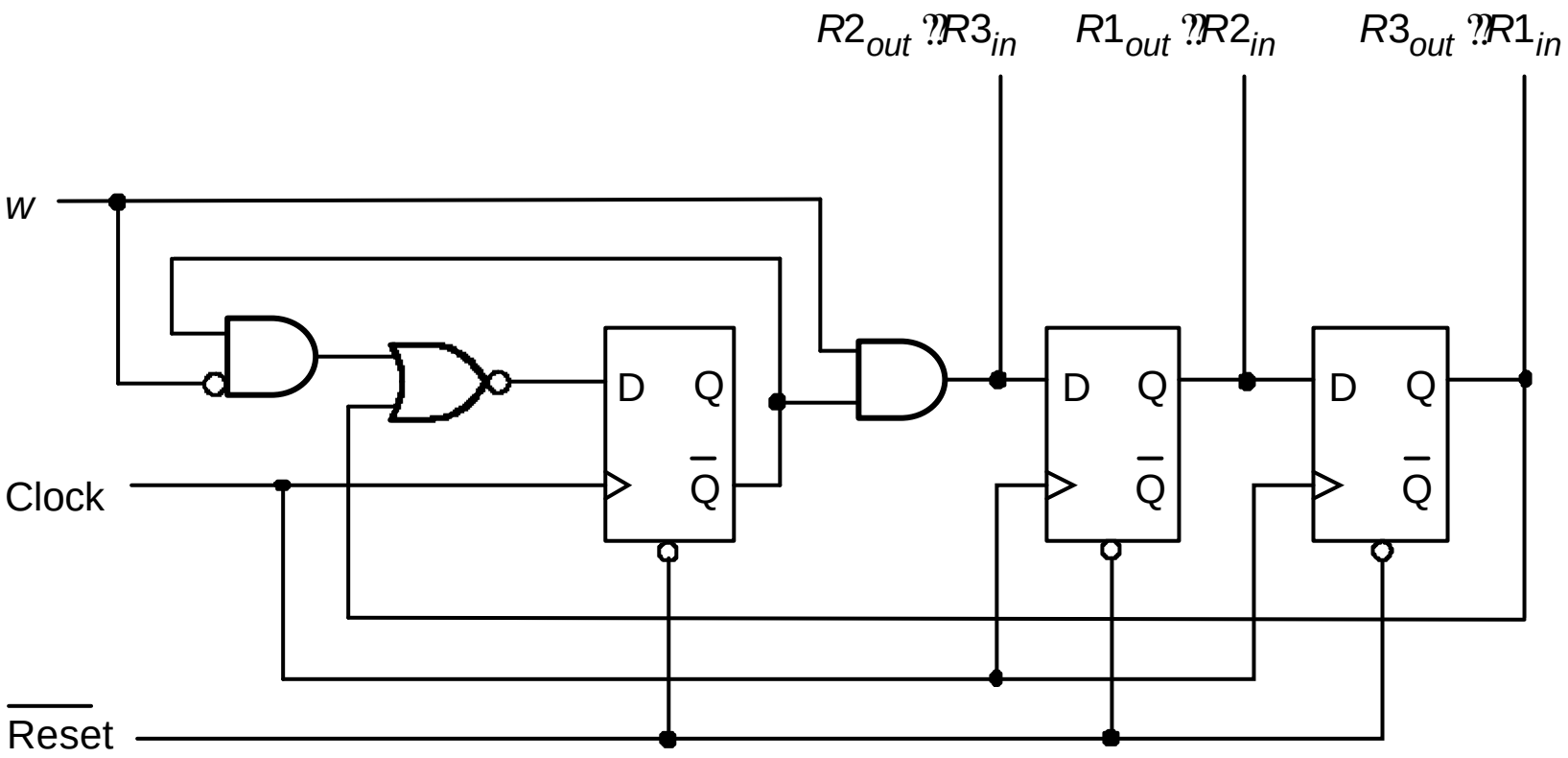
Circuito de controle com um shift-register



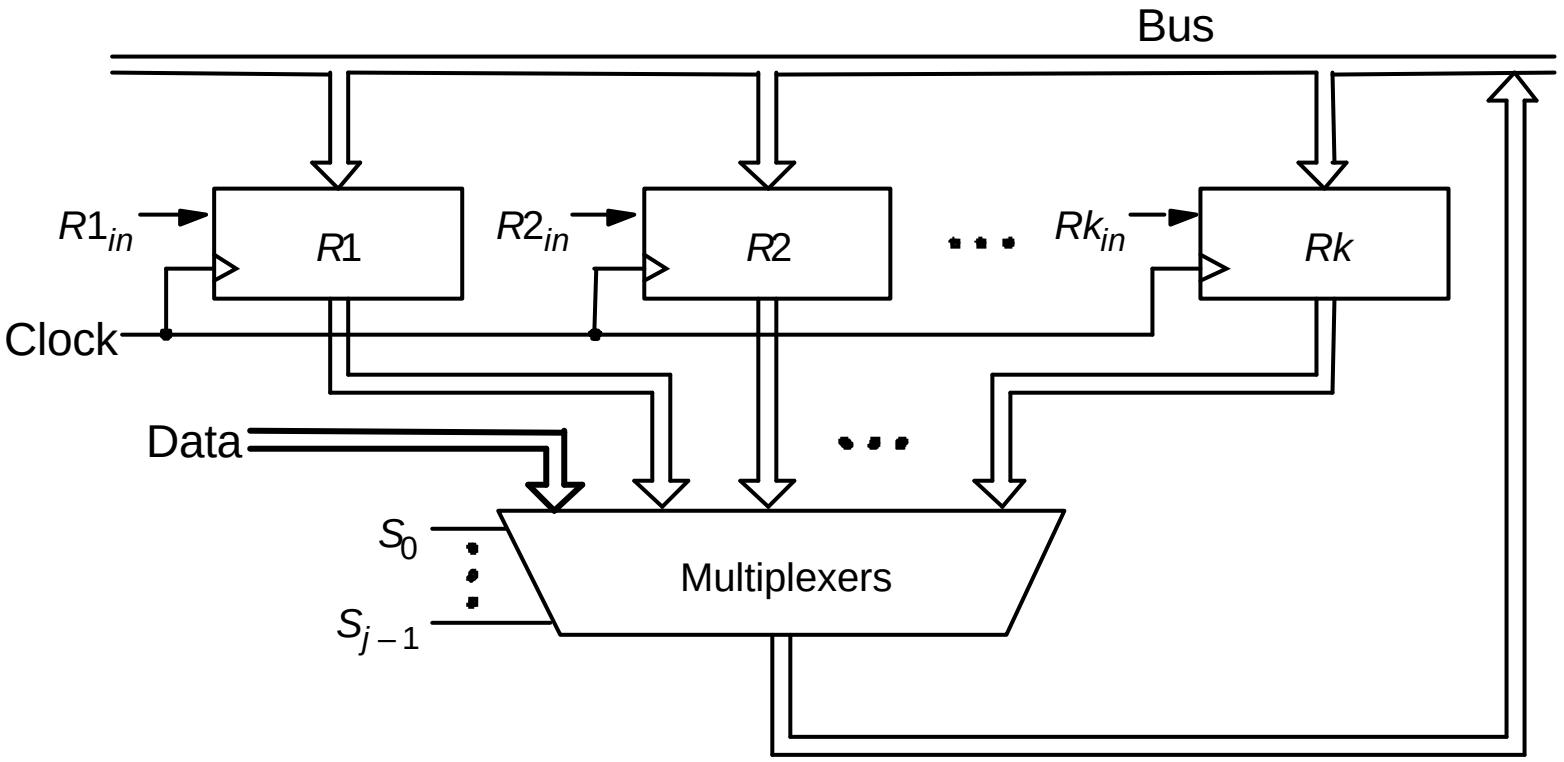
Circuito de controle modificado



Cicuito de controle com FF sem preset



Usando multiplexadores para implementação de um barramento



Código para registrador de *n*-bits com 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 ;
```

Código para buffer tri-sate *n*-bits

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

ENTITY trin IS
      GENERIC ( N : INTEGER := 8 ) ;
      PORT (   X    : IN       STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
               E    : IN       STD_LOGIC ;
               F    : OUT      STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END trin ;

ARCHITECTURE Behavior OF trin IS
BEGIN
      F <= (OTHERS => 'Z') WHEN E = '0' ELSE X ;
END Behavior ;
```

Código para controlador com for the shift-register

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

ENTITY shiftr IS  -- left-to-right shift register with async reset
        GENERIC ( K : INTEGER := 4 ) ;
        PORT (      Resetn, Clock, w        : IN            STD_LOGIC ;
                    Q                       : BUFFER        STD_LOGIC_VECTOR(1 TO K) ) ;
END shiftr ;

ARCHITECTURE Behavior OF shiftr IS
BEGIN
        PROCESS ( Resetn, Clock )
        BEGIN
                IF Resetn = '0' THEN
                        Q <= (OTHERS => '0') ;
                ELSIF Clock'EVENT AND Clock = '1' THEN
                        Genbits: FOR i IN K DOWNTO 2 LOOP
                                Q(i) <= Q(i-1) ;
                        END LOOP ;
                        Q(1) <= w ;
                END IF ;
        END PROCESS ;
END Behavior ;
```

Declaração de package e component

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

PACKAGE components IS
    COMPONENT regn -- register
        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 COMPONENT ;

    COMPONENT shiftr -- left-to-right shift register with async reset
        GENERIC ( K : INTEGER := 4 ) ;
        PORT    (  Resetn, Clock, w    : IN      STD_LOGIC ;
                   Q                   : BUFFER  STD_LOGIC_VECTOR(1 TO K) ) ;
    END component ;

    COMPONENT trin -- tri-state buffers
        GENERIC ( N : INTEGER := 8 ) ;
        PORT (  X  : IN  STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                E  : IN  STD_LOGIC ;
                F  : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
    END COMPONENT ;
END components ;
```

Sistema digital com bus

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

ENTITY swap IS
     PORT (   Data              : IN    STD_LOGIC_VECTOR(7
DOWNTO 0) ;
              Resetn, w         : IN    STD_LOGIC ;
              Clock, Extern     : IN    STD_LOGIC ;
              RinExt            : IN    STD_LOGIC_VECTOR(1 TO 3) ;
              BusWires : INOUT  STD_LOGIC_VECTOR(7
DOWNTO 0) ) ;
END swap ;
```

Sistema digital com bus - continuação

```
ARCHITECTURE Behavior OF swap IS
    SIGNAL Rin, Rout, Q : STD_LOGIC_VECTOR(1 TO 3) ;
    SIGNAL R1, R2, R3 : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
BEGIN
    control: shiftr GENERIC MAP ( K => 3 )
        PORT MAP ( Resetn, Clock, w, Q ) ;
    Rin(1) <= RinExt(1) OR Q(3) ;
    Rin(2) <= RinExt(2) OR Q(2) ;
    Rin(3) <= RinExt(3) OR Q(1) ;
    Rout(1) <= Q(2) ; Rout(2) <= Q(1) ; Rout(3) <= Q(3) ;

    tri_ext: trin PORT MAP ( Data, Extern, BusWires ) ;
    reg1: regn PORT MAP ( BusWires, Rin(1), Clock, R1 ) ;
    reg2: regn PORT MAP ( BusWires, Rin(2), Clock, R2 ) ;
    reg3: regn PORT MAP ( BusWires, Rin(3), Clock, R3 ) ;
    tri1: trin PORT MAP ( R1, Rout(1), BusWires ) ;
    tri2: trin PORT MAP ( R2, Rout(2), BusWires ) ;
    tri3: trin PORT MAP ( R3, Rout(3), BusWires ) ;
END Behavior ;
```

Usando multiplexadores para implementar um bus

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

ENTITY swapmux IS
    PORT (  Data        : IN        STD_LOGIC_VECTOR(7 DOWNTO 0) ;
            Resetn, w   : IN        STD_LOGIC ;
            Clock       : IN        STD_LOGIC ;
            RinExt      : IN        STD_LOGIC_VECTOR(1 TO 3) ;
            BusWires    : BUFFER    STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END swapmux ;

ARCHITECTURE Behavior OF swapmux IS
    SIGNAL Rin, Q : STD_LOGIC_VECTOR(1 TO 3) ;
    SIGNAL S : STD_LOGIC_VECTOR(1 DOWNTO 0) ;
    SIGNAL R1, R2, R3 : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
BEGIN
    control: shiftr GENERIC MAP ( K => 3 )
        PORT MAP ( Resetn, Clock, w, Q ) ;

    … con't
```

Usando multiplexadores para implementar um bus - continuação

```
    Rin(1) <= RinExt(1) OR Q(3) ;
    Rin(2) <= RinExt(2) OR Q(2) ;
    Rin(3) <= RinExt(3) OR Q(1) ;
    reg1: regn PORT MAP ( BusWires, Rin(1), Clock, R1 ) ;
    reg2: regn PORT MAP ( BusWires, Rin(2), Clock, R2 ) ;
    reg3: regn PORT MAP ( BusWires, Rin(3), Clock, R3 ) ;
    encoder:
    WITH Q SELECT
        S <= "00" WHEN "000",
             "10" WHEN "100",
             "01" WHEN "010",
             "11" WHEN OTHERS ;
    muxes: --eight 4-to-1 multiplexers
    WITH S SELECT
        BusWires <= Data WHEN "00",
                    R1 WHEN "01",
                    R2 WHEN "10",
                    R3 WHEN OTHERS ;
END Behavior ;
```

Código simplificado para descrever um barramento

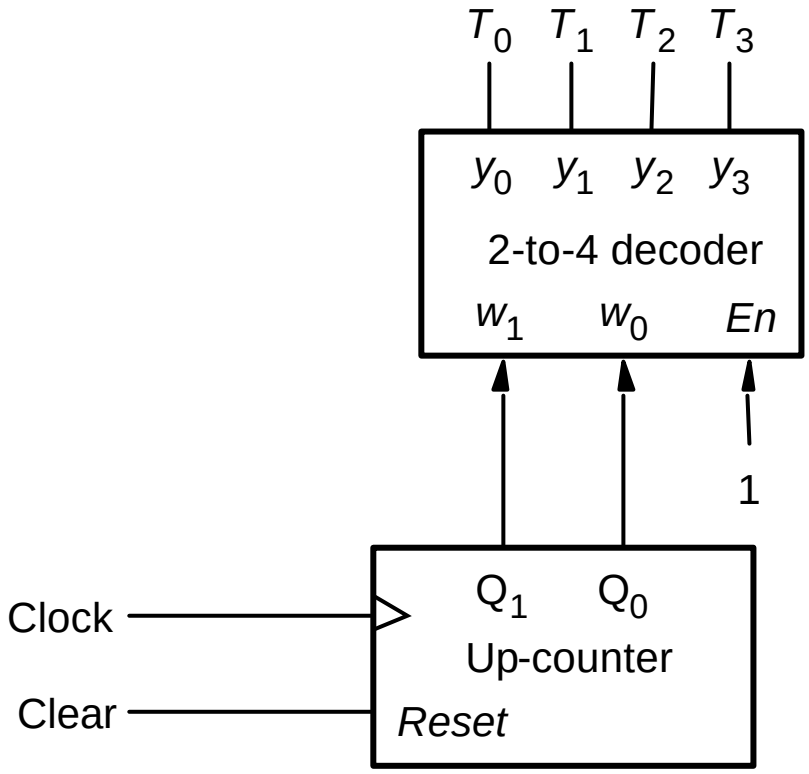
```
(ENTITY declaration not shown)

ARCHITECTURE Behavior OF swapmux IS
        SIGNAL Rin, Q : STD_LOGIC_VECTOR(1 TO 3) ;
        SIGNAL R1, R2, R3 : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
BEGIN
        control: shiftr GENERIC MAP ( K => 3 )
                PORT MAP ( Resetn, Clock, w, Q ) ;
        Rin(1) <= RinExt(1) OR Q(3) ;
        Rin(2) <= RinExt(2) OR Q(2) ;
        Rin(3) <= RinExt(3) OR Q(1) ;

        reg1: regn PORT MAP ( BusWires, Rin(1), Clock, R1 ) ;
        reg2: regn PORT MAP ( BusWires, Rin(2), Clock, R2 ) ;
        reg3: regn PORT MAP ( BusWires, Rin(3), Clock, R3 ) ;

        muxes:
        WITH Q SELECT
                BusWires <=  Data    WHEN "000",
                             R2      WHEN "100",
                             R1      WHEN "010",
                             R3      WHEN OTHERS ;
END Behavior ;
```

Sistema digital implementa um processador simples

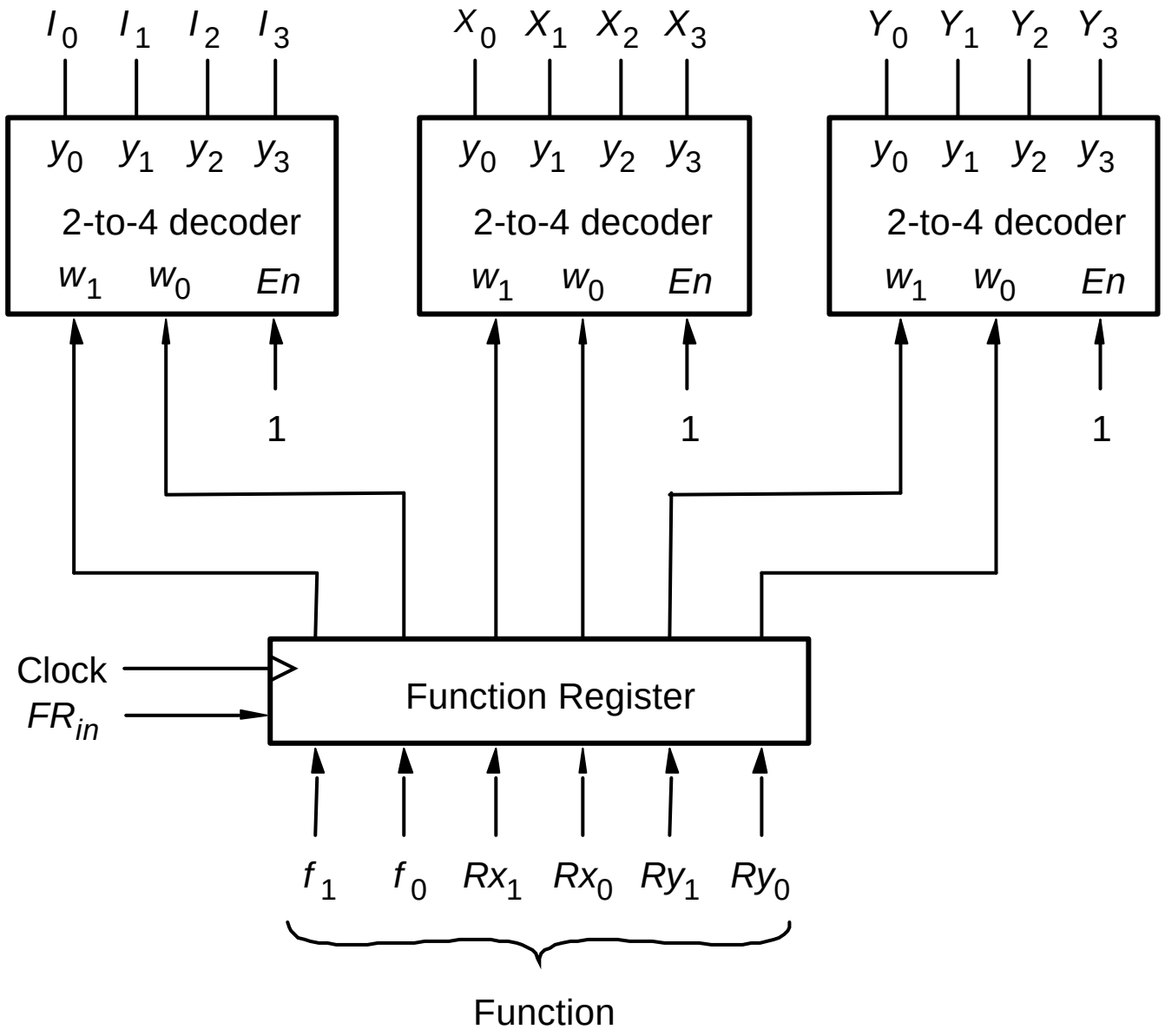
Operacões executadas pelo processador

Operation	Function performed
Load Rx,$Data$	$Rx \leftarrow Data$
Move Rx, Ry	$Rx \leftarrow [Ry]$
Add Rx, Ry	$Rx \leftarrow [Rx] + [Ry]$
Sub Rx, Ry	$Rx \leftarrow [Rx] - [Ry]$

Circuito de controle do porcessador



Registrador de função e os decodificadores



Valores dos sinais de controle para cada operação/time step

	T_1	T_2	T_3
(Load): I_0	$Extern, R_{in} = X,$ $Done$		
(Move): I_1	$R_{in} = X, R_{out} = Y,$ $Done$		
(Add): I_2	$R_{out} = X, A_{in}$	$R_{out} = Y, G_{in},$ $AddSub = 0$	$G_{out}, R_{in} = X,$ $Done$
(Sub): I_3	$R_{out} = X, A_{in}$	$R_{out} = Y, G_{in},$ $AddSub = 1$	$G_{out}, R_{in} = X,$ $Done$

Código para two-bit up-counter reset assíncrono

```
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 ;
```

Código para o processador

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;
USE work.subccts.all ;

ENTITY proc IS
        PORT (  Data        : IN        STD_LOGIC_VECTOR(7 DOWNTO 0) ;
                Reset, w    : IN        STD_LOGIC ;
                Clock       : IN        STD_LOGIC ;
                F, Rx, Ry   : IN        STD_LOGIC_VECTOR(1 DOWNTO 0) ;
                Done        : BUFFER    STD_LOGIC ;
                BusWires    : INOUT     STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END proc ;

ARCHITECTURE Behavior OF proc IS
        SIGNAL Rin, Rout : STD_LOGIC_VECTOR(0 TO 3) ;
        SIGNAL Clear, High, AddSub : STD_LOGIC ;
        SIGNAL Extern, Ain, Gin, Gout, FRin : STD_LOGIC ;
        SIGNAL Count, Zero : STD_LOGIC_VECTOR(1 DOWNTO 0) ;
        SIGNAL T, I, X, Y : STD_LOGIC_VECTOR(0 TO 3) ;
        SIGNAL R0, R1, R2, R3 : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
        SIGNAL A, Sum, G : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
        SIGNAL Func, FuncReg : STD_LOGIC_VECTOR(1 TO 6) ;
```

… con’t

Código para o processador - continuação

```
BEGIN
    Zero <= "00" ; High <= '1' ;
    Clear <= Reset OR Done OR (NOT w AND T(0)) ;
    counter: upcount PORT MAP (          Clear, Clock, Count ) ;
    decT: dec2to4 PORT MAP ( Count, High, T );
    Func <= F & Rx & Ry ;
    FRin <= w AND T(0) ;
    functionreg: regn GENERIC MAP ( N => 6 )
        PORT MAP ( Func, FRin, Clock, FuncReg ) ;
    decI: dec2to4 PORT MAP ( FuncReg(1 TO 2), High, I ) ;
    decX: dec2to4 PORT MAP ( FuncReg(3 TO 4), High, X ) ;
    decY: dec2to4 PORT MAP ( FuncReg(5 TO 6), High, Y ) ;

    Extern <= I(0) AND T(1) ;
    Done <= ((I(0) OR I(1)) AND T(1)) OR ((I(2) OR I(3)) AND T(3)) ;
    Ain <= (I(2) OR I(3)) AND T(1) ;
    Gin <= (I(2) OR I(3)) AND T(2) ;
    Gout <= (I(2) OR I(3)) AND T(3) ;
    AddSub <= I(3) ;

    … con't
```

Código para o processador - continuação

```
    RegCntl:
    FOR k IN 0 TO 3 GENERATE
        Rin(k) <= ((I(0) OR I(1)) AND T(1) AND X(k)) OR
            ((I(2) OR I(3)) AND T(3) AND X(k)) ;
        Rout(k) <= (I(1) AND T(1) AND Y(k)) OR
            ((I(2) OR I(3)) AND ((T(1) AND X(k)) OR (T(2) AND Y(k)))) ;
    END GENERATE RegCntl ;
    tri_extern: trin PORT MAP ( Data, Extern, BusWires ) ;
    reg0: regn PORT MAP ( BusWires, Rin(0), Clock, R0 ) ;
    reg1: regn PORT MAP ( BusWires, Rin(1), Clock, R1 ) ;
    reg2: regn PORT MAP ( BusWires, Rin(2), Clock, R2 ) ;
    reg3: regn PORT MAP ( BusWires, Rin(3), Clock, R3 ) ;
    tri0: trin PORT MAP ( R0, Rout(0), BusWires ) ;
    tri1: trin PORT MAP ( R1, Rout(1), BusWires ) ;
    tri2: trin PORT MAP ( R2, Rout(2), BusWires ) ;
    tri3: trin PORT MAP ( R3, Rout(3), BusWires ) ;
    regA: regn PORT MAP ( BusWires, Ain, Clock, A ) ;
    alu:
    WITH AddSub SELECT
        Sum <=  A + BusWires WHEN '0',
                A - BusWires WHEN OTHERS ;
    regG: regn PORT MAP ( Sum, Gin, Clock, G ) ;
    triG: trin PORT MAP ( G, Gout, BusWires ) ;
END Behavior ;
```

Alternativa para o código do processador

```
… (ENTITY declaration not shown)
ARCHITECTURE Behavior OF proc IS
        SIGNAL X, Y, Rin, Rout : STD_LOGIC_VECTOR(0 TO 3) ;
        SIGNAL Clear, High, AddSub : STD_LOGIC ;
        SIGNAL Extern, Ain, Gin, Gout, FRin : STD_LOGIC ;
        SIGNAL Count, Zero, T, I : STD_LOGIC_VECTOR(1 DOWNTO 0) ;
        SIGNAL R0, R1, R2, R3 : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
        SIGNAL A, Sum, G : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
        SIGNAL Func, FuncReg, Sel : STD_LOGIC_VECTOR(1 TO 6) ;
BEGIN
        Zero <= "00" ; High <= '1' ;
        Clear <= Reset OR Done OR (NOT w AND NOT T(1) AND NOT T(0)) ;
        counter: upcount PORT MAP (          Clear, Clock, Count ) ;
        T <= Count ;
        Func <= F & Rx & Ry ;
        FRin <= w AND NOT T(1) AND NOT T(0) ;
        functionreg: regn GENERIC MAP ( N => 6 )
                PORT MAP ( Func, FRin, Clock, FuncReg ) ;
        I <= FuncReg(1 TO 2) ;
        decX: dec2to4 PORT MAP ( FuncReg(3 TO 4), High, X ) ;
        decY: dec2to4 PORT MAP ( FuncReg(5 TO 6), High, Y ) ;
        controlsignals: PROCESS ( T, I, X, Y )
        BEGIN
… con't
```

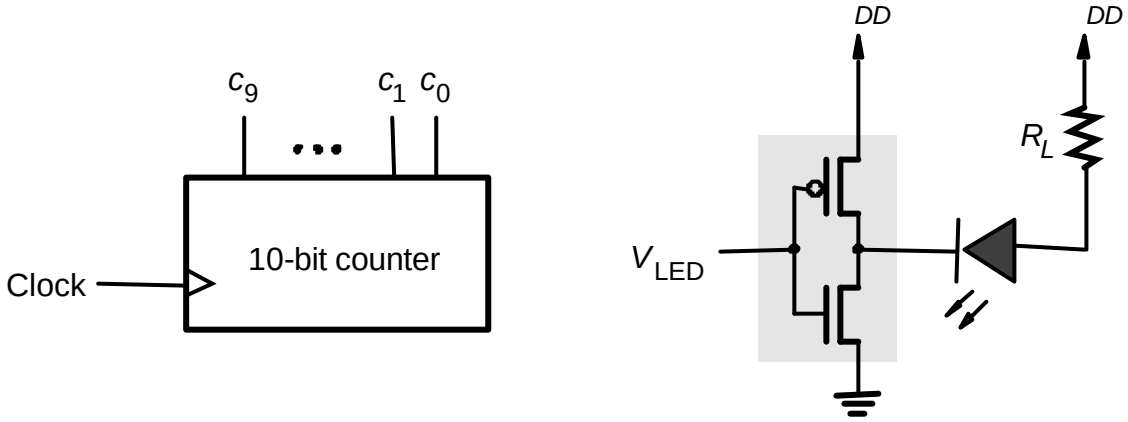
Alternativa para o código do processador - continuação

```
Extern <= '0' ; Done <= '0' ; Ain <= '0' ; Gin <= '0' ;
Gout <= '0' ; AddSub <= '0' ; Rin <= "0000" ; Rout <= "0000" ;
CASE T IS
    WHEN "00" => -- no signals asserted in time step T0
    WHEN "01" => -- define signals asserted in time step T1
        CASE I IS
            WHEN "00" => -- Load
                Extern <= '1' ; Rin <= X ; Done <= '1' ;
            WHEN "01" => -- Move
                Rout <= Y ; Rin <= X ; Done <= '1' ;
            WHEN OTHERS => -- Add, Sub
                Rout <= X ; Ain <= '1' ;
        END CASE ;

    WHEN "10" => -- define signals asserted in time step T2
        CASE I IS
            WHEN "10" => -- Add
                Rout <= Y ; Gin <= '1' ;
            WHEN "11" => -- Sub
                Rout <= Y ; AddSub <= '1' ; Gin <= '1' ;
            WHEN OTHERS =>      -- Load, Move
        END CASE ;
    WHEN OTHERS => -- define signals asserted in time step T3
        CASE I IS
```

Alternativa para o código do processador - continuação

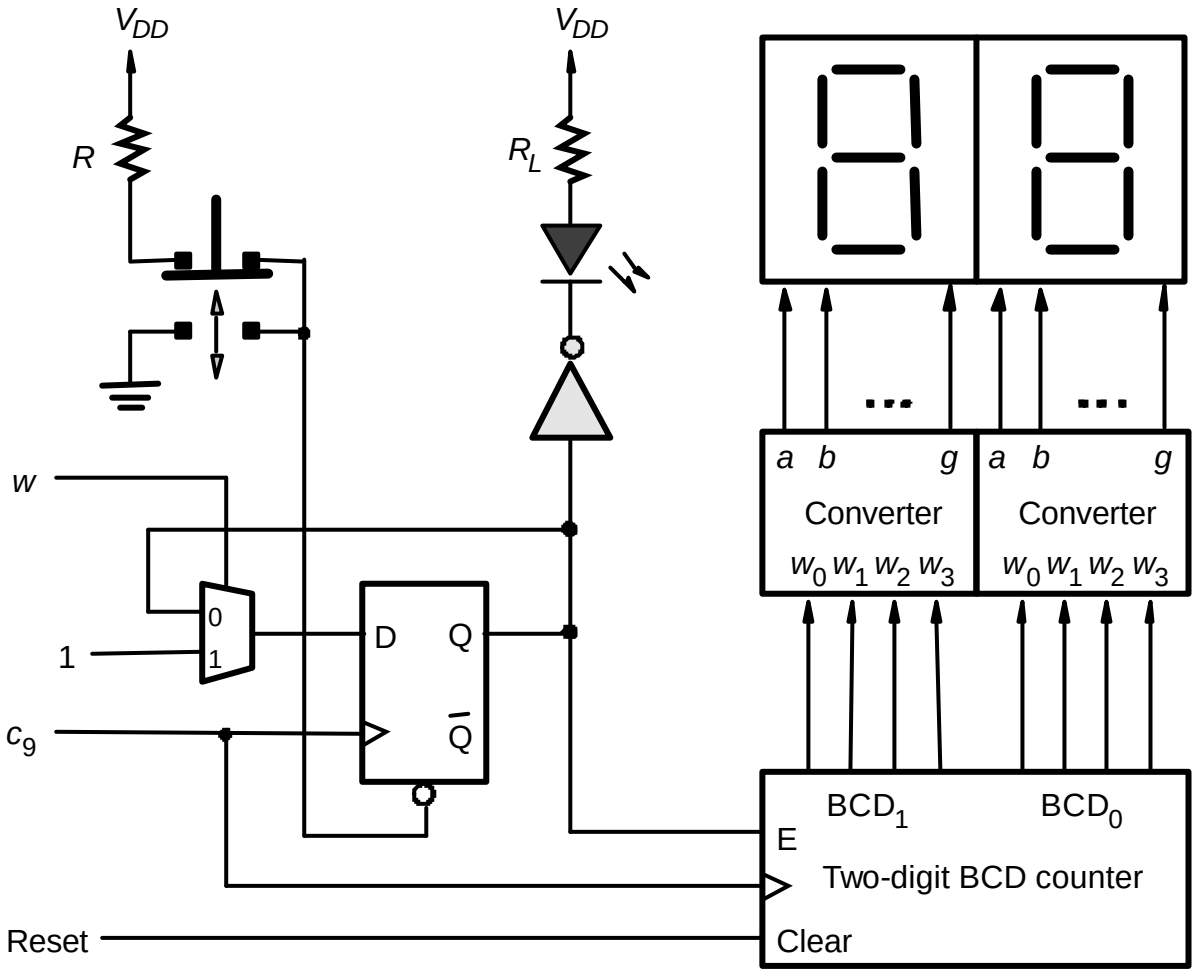
```
                        WHEN "00" => -- Load
                        WHEN "01" => -- Move
                        WHEN OTHERS => -- Add, Sub
                            Gout <= '1' ; Rin <= X ; Done <= '1' ;
                END CASE ;
        END CASE ;
    END PROCESS ;
    reg0: regn PORT MAP ( BusWires, Rin(0), Clock, R0 ) ;
    reg1: regn PORT MAP ( BusWires, Rin(1), Clock, R1 ) ;
    reg2: regn PORT MAP ( BusWires, Rin(2), Clock, R2 ) ;
    reg3: regn PORT MAP ( BusWires, Rin(3), Clock, R3 ) ;
    regA: regn PORT MAP ( BusWires, Ain, Clock, A ) ;
    alu: WITH AddSub SELECT
        Sum <=    A + BusWires WHEN '0',
                  A - BusWires WHEN OTHERS ;
    regG: regn PORT MAP ( Sum, Gin, Clock, G ) ;
    Sel <= Rout & Gout & Extern ;
    WITH Sel SELECT
        BusWires <=    R0    WHEN "100000",
                       R1    WHEN "010000",
                       R2    WHEN "001000",
                       R3    WHEN "000100",
                       G     WHEN "000010",
                       Data  WHEN OTHERS ;
END Behavior ;
```

Um circuito reaction-timer



(a) Clock divider (b) LED circuit

Um circuito reaction-timer



(c) Push-button switch, LED, and 7-segment displays

Código para contador BCD de dois dígitos

```
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 ;

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

… con’t

Código para contador BCD de dois dígitos

```
            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 ;
```

Código para o reaction timer

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

ENTITY reaction IS
        PORT (  c9, Reset       : IN        STD_LOGIC ;
                w, Pushn        : IN        STD_LOGIC ;
                LEDn            : OUT       STD_LOGIC ;
                Digit1, Digit0  : BUFFER    STD_LOGIC_VECTOR(1 TO 7) ) ;
END reaction ;

ARCHITECTURE Behavior OF reaction IS
        COMPONENT BCDcount
                PORT (  Clock           : IN        STD_LOGIC ;
                        Clear, E        : IN        STD_LOGIC ;
                        BCD1, BCD0      : BUFFER    STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
        END COMPONENT ;
        COMPONENT seg7
                PORT (  bcd     : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
                        leds    : OUT   STD_LOGIC_VECTOR(1 TO 7) ) ;
        END COMPONENT ;
        SIGNAL LED : STD_LOGIC ;
        SIGNAL BCD1, BCD0 : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
```

… con’t

Código para o reaction timer (continuação)

```
BEGIN
        flipflop: PROCESS
        BEGIN
                WAIT UNTIL c9'EVENT AND c9 = '1' ;
                IF Pushn = '0' THEN
                    LED <= '0' ;
                ELSIF w = '1' THEN
                    LED <= '1' ;
                END IF ;
        END PROCESS ;

        LEDn <= NOT LED ;
        counter: BCDcount PORT MAP ( c9, Reset, LED, BCD1, BCD0 ) ;
        seg1 : seg7 PORT MAP ( BCD1, Digit1 ) ;
        seg0 : seg7 PORT MAP ( BCD0, Digit0 ) ;
END Behavior ;
```