# PCS 3115

# Sistemas Digitais I

## Módulo 20 – Registradores e Contadores

*Prof. Dr. Marcos A. Simplicio Jr.*

*Atualização: Prof. Dr. Marco Túlio Carvalho de Andrade*

*versão: 5.0 (Maio/2020)*

# Registradores: Conceito

- Associação de 2 ou mais Flip-Flops, controla-dos por um *clock* comum, onde cada Flip-Flop armazena 1 só bit – O registrador armazena um conjunto de bits;
- Algumas entradas comuns, síncronas e/ou assíncronas – *Set/Preset, Reset/ Clear, Load;*
- Entradas de alguns Flip-Flops ligadas ao exterior (*load* – carga externa).
- Saídas de alguns Flip-Flops ligadas a entradas de outros (*shift* – deslocamento).

# Registradores: Funções & Usos

- Armazenar conjunto de bits:
  - **Correlacionados**: bits de **dados**.
    - Ex.: armazenar número “12” ➔ 1100
  - **Não relacionados**: bits de **controle**
    - Ex.: armazenar pressionamento dos botões 3 e 2 (1 e 0 não pressionados) ➔ 1100
  - Obs.: Diferença meramente **semântica**!
- Única restrição real: todos os bits devem seguir o **mesmo sinal de *clock*** (nível e/ou borda) quando de sua atualização/registro.

# Registradores: Tipos

- **Síncronos** – Ativados por **borda** (subida ou descida) ou **nível** (alto ou baixo) de sinal de ***clock***.

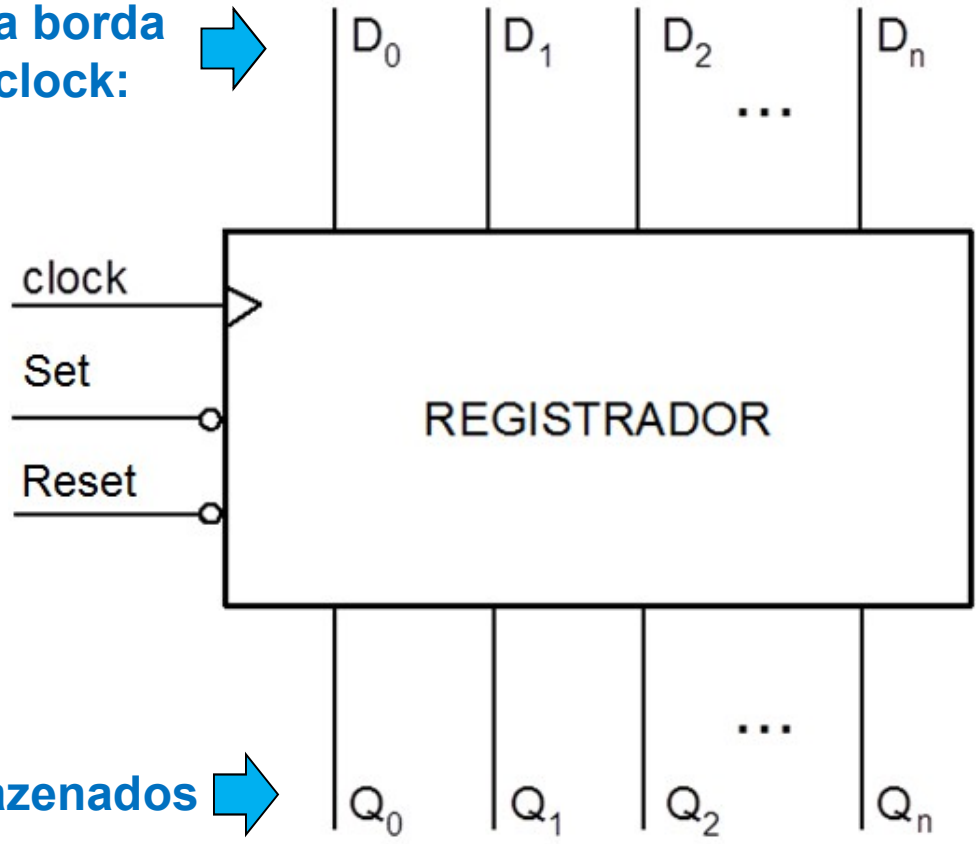
- **Assíncronos** – Ativados **independentemente** da existência de um sinal de ***clock***.
  - Obs. – Registradores síncronos têm, comumente, algumas de suas entradas de controle assíncronas.

# Registradores: Sinais de Controle

- ***Clock* –** Sincroniza atualizações/registros de bits;
- ***Enable* –** Habilita outros sinais (ex.: *clock*);
- ***Set* (*Preset*) –** Carrega todos bits em 1;
- ***Reset* (*Clear*) –** Carrega todos os bits em 0;
- ***Load* –** Carga (registro/armazenamento) de uma determinada cadeia de bits passada como entrada;
- ***Output Enable* –** Controle *tri-state* da saída do registrador;
- ***Shift* (*Right/Left*) & *Hold* –** Deslocamento (Direita/ Esquerda) ou manutenção da informação armazenada (deslocadores & registradores).

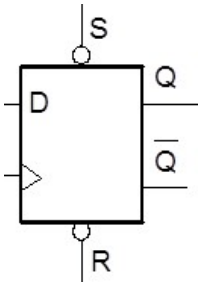
# Registradores: Símbolo Lógico

**Carregados na borda de subida do clock:**



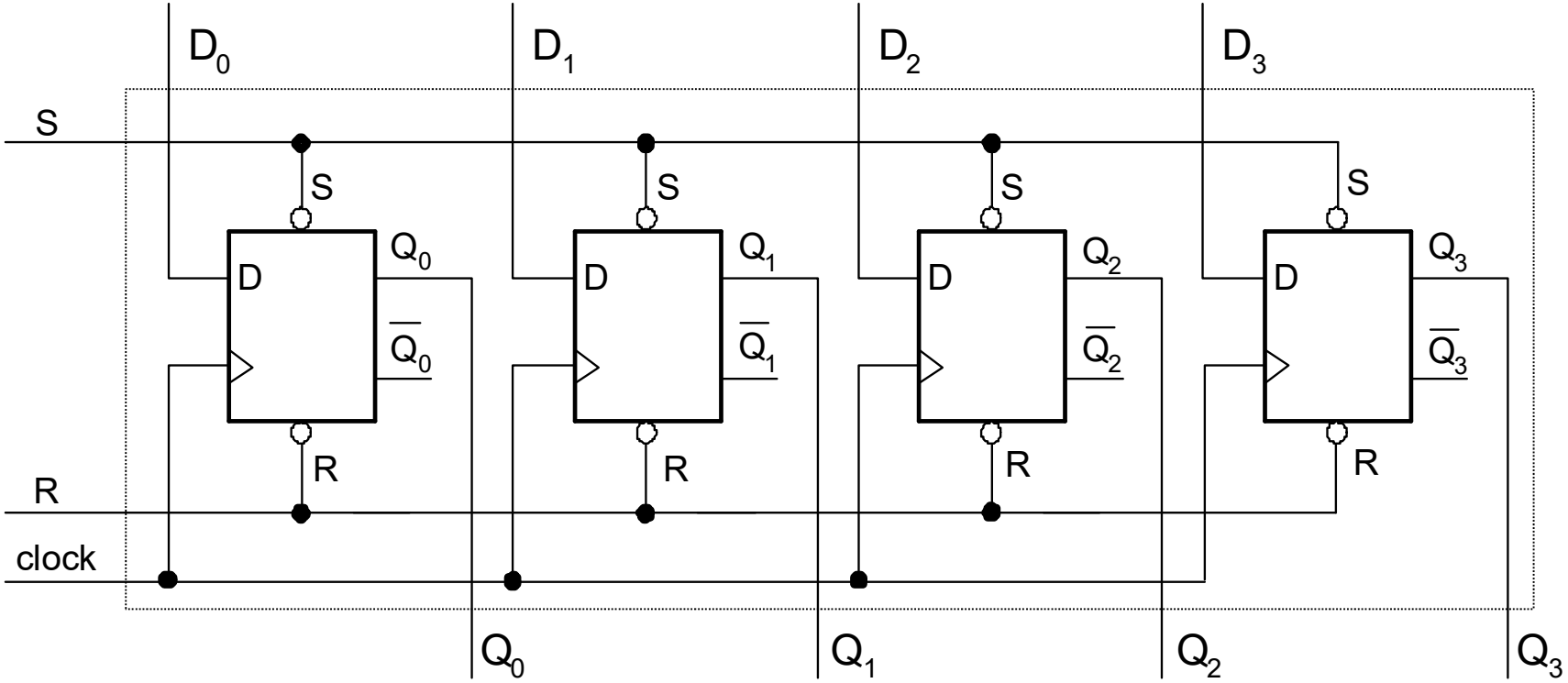

**Valores armazenados**

- **Pergunta**: como implementar com FF tipo D?



# Registradores: por dentro

- Ex.: registrador de 4 bits com flip-flops tipo D

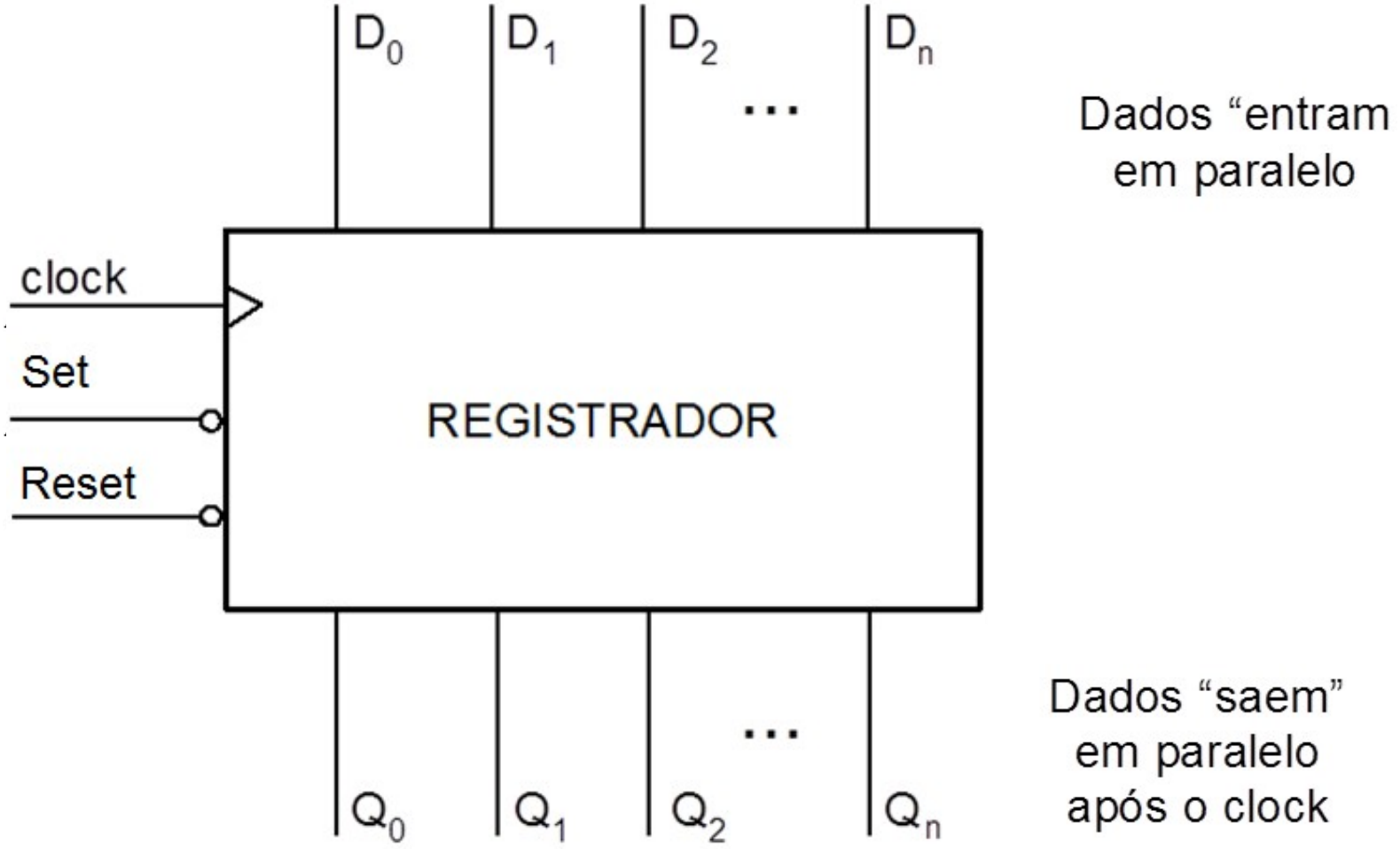


# Registradores: Entradas e Saídas

- Entradas e saídas podem ser de diferentes formas:
  - **Paralela –** Todos os bits simultaneamente;
  - **Série –** Apenas 1 bit por vez.
- Tipos suportados por registradores:
  - Entrada paralela e saída paralela;
  - Entrada série e saída paralela;
  - Entrada paralela e saída série;
  - Entrada série e saída série.

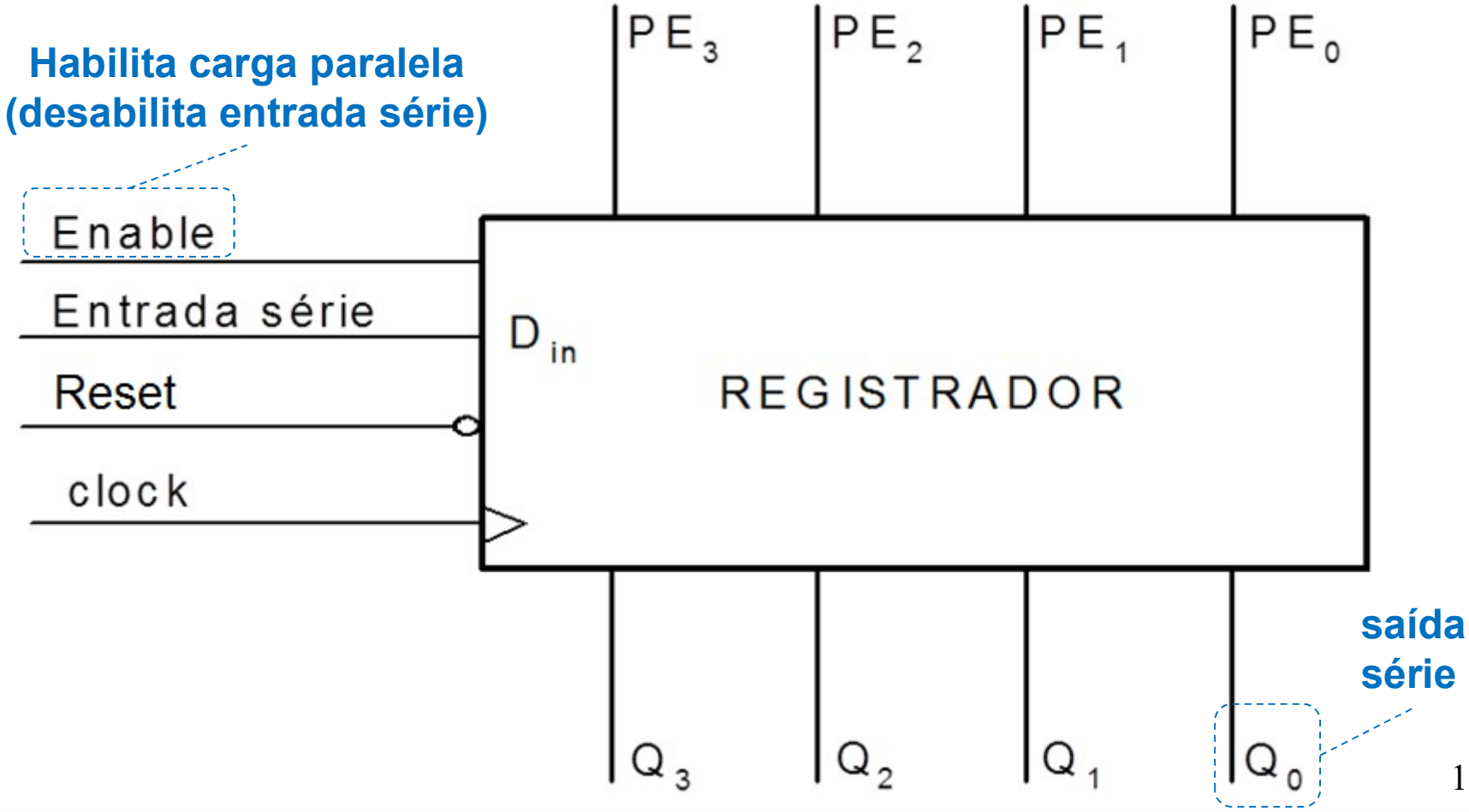
# Registradores

- **Entrada paralela e saída paralela** (exemplo anterior)



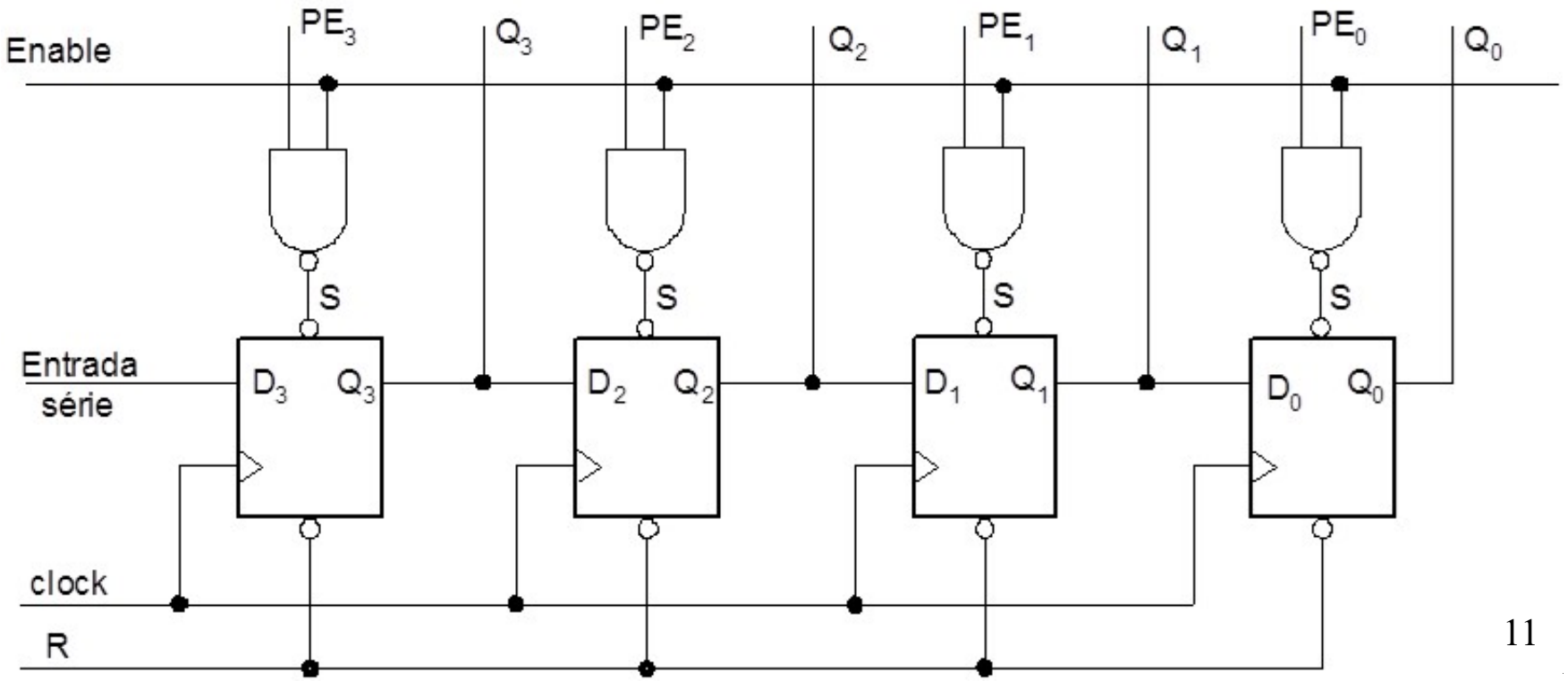
# Registradores

- Entrada série e/ou paralela e saída série ($Q_0$) e/ou paralela



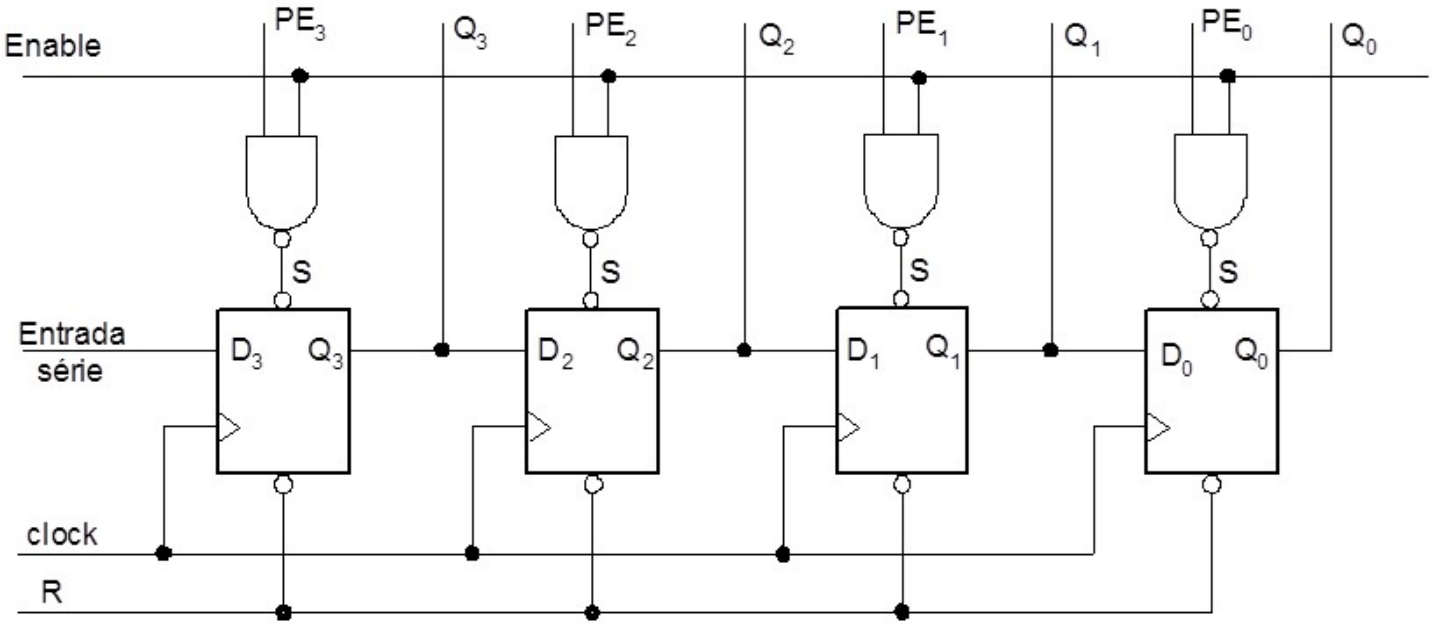
# Registradores

- Entrada série, saída série (Q0) e/ou paralela e carga paralela assíncrona
  - Nota: carga em duas etapas – (I) reset e (II) load



# Registradores

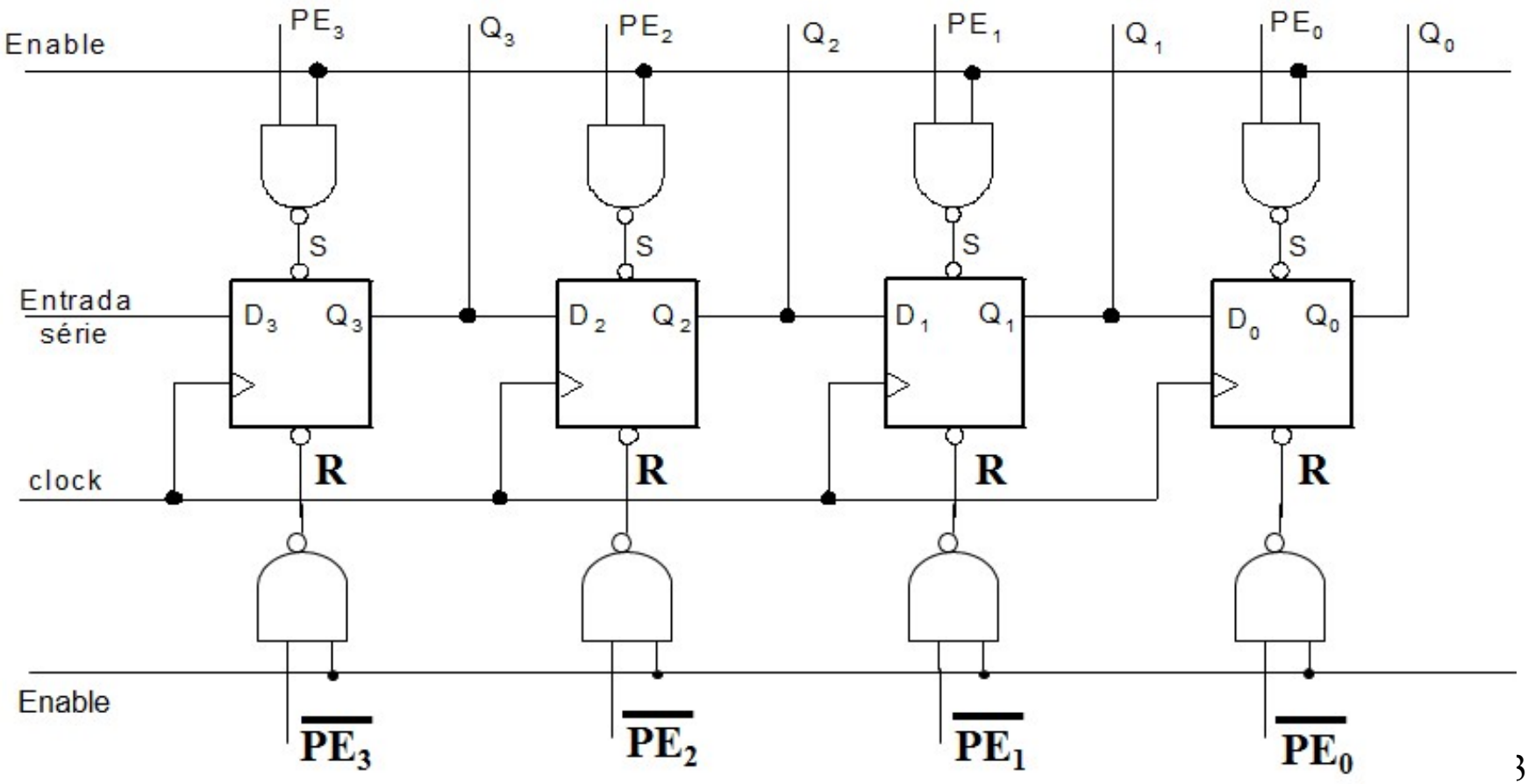
- Entrada série,  saída série (Q0) e/ou paralela e carga paralela assíncrona
  - Nota: carga em duas etapas – (I) reset e (II) load




- **Desafio**: como implementar carga assíncrona com 1 única etapa? (dica: usar o reset, substituindo-o)

# Registradores

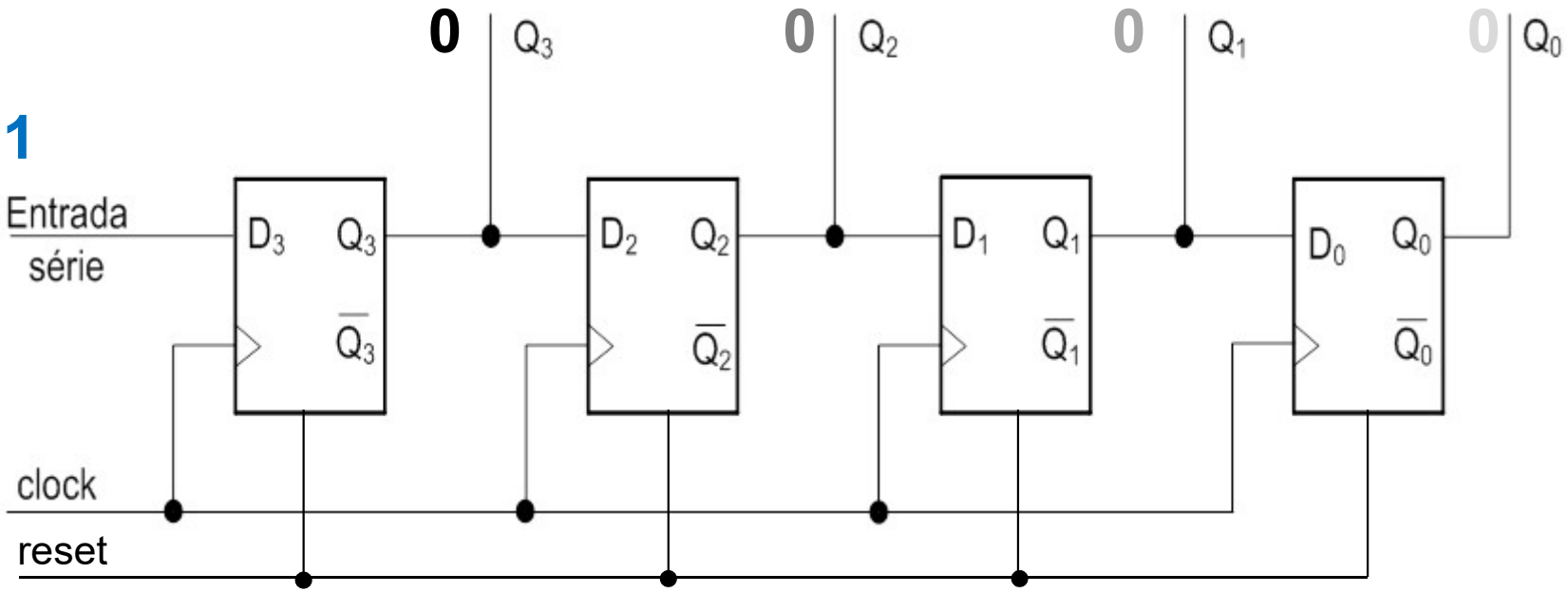
- Entrada série, saída série ($Q_0$) e/ou paralela e **carga paralela assíncrona** (1 só etapa)



# Registradores de deslocamento

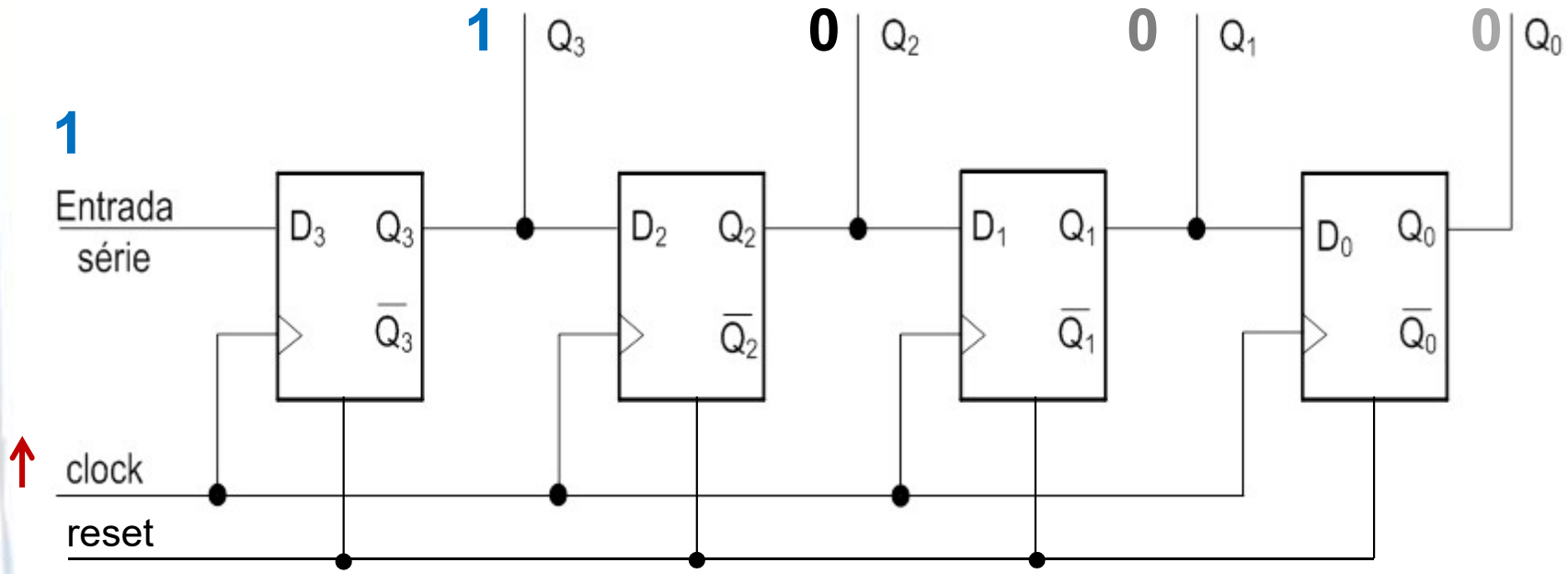
- O que acontece se a **saída de um FF** tipo D for **ligada à entrada do próximo** FF?
  - Suponha que seja feito reset inicial

(Obs.: entrada série e saída paralela)



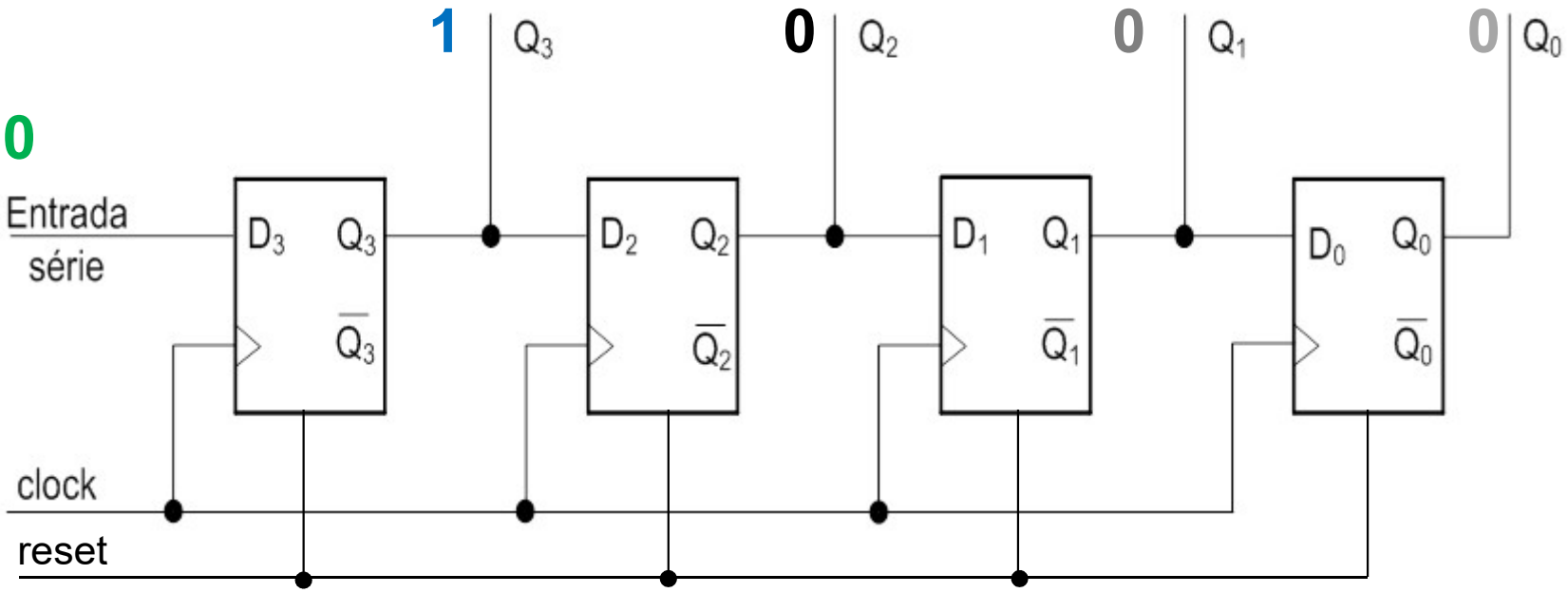
# Registradores de deslocamento

- O que acontece se a **saída de um FF** tipo D for **ligada à entrada do próximo** FF?
  - Clock: carga da entrada e **deslocamento** dos bits



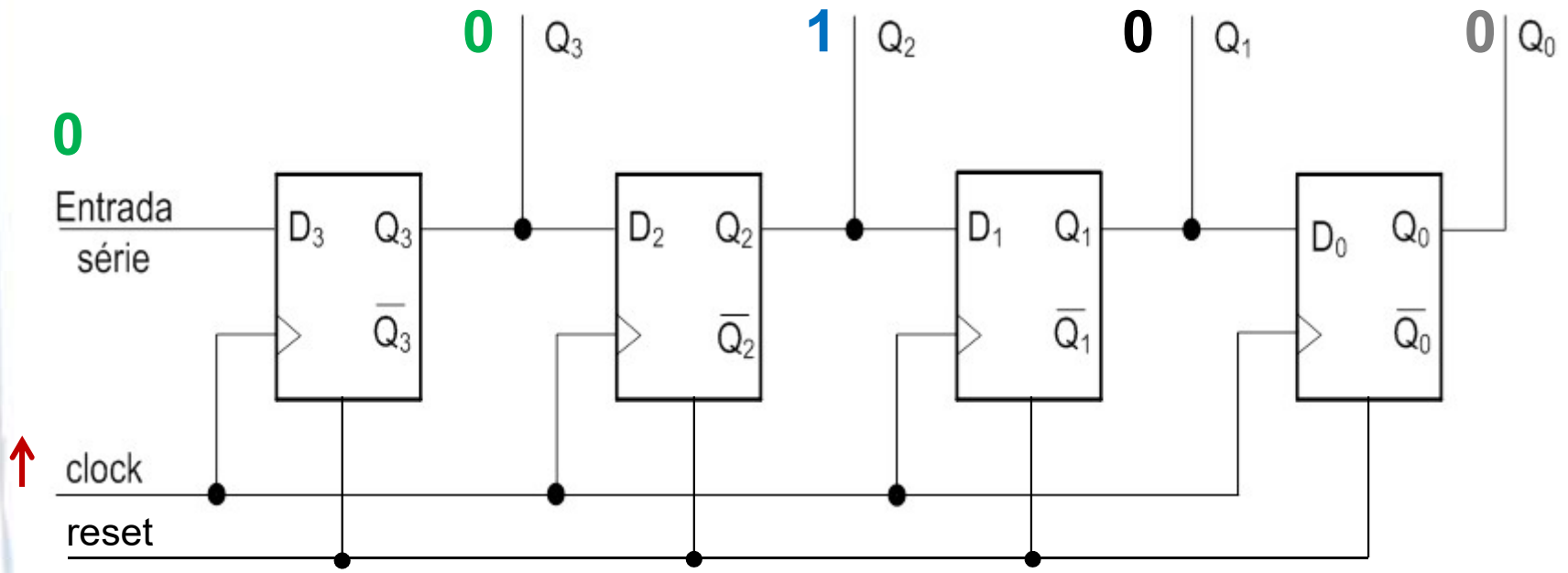
# Registradores de deslocamento

- O que acontece se a **saída de um FF** tipo D for **ligada à entrada do próximo** FF?
  - Entrada agora é **0**



# Registradores de deslocamento

- O que acontece se a **saída de um FF** tipo D for **ligada à entrada do próximo** FF?
  - Clock: carga da entrada e **deslocamento** dos bits



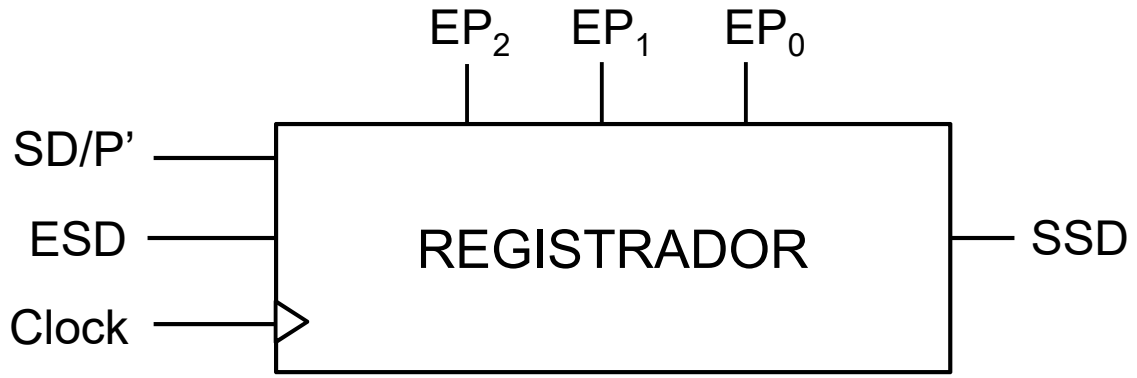
# Registradores de deslocamento

- O que acontece se a **saída de um FF** tipo D for **ligada à entrada do próximo** FF?
  - Tem-se o equivalente a uma operação de **deslocamento da cadeia de bits**.
- Neste caso o registrador é denominado Registrador Deslocador, ou **Registrador de Deslocamento**.
  - Em inglês: *shift register*

# Registradores: exercício 1

- Como dotar registrador de entrada série ou paralela?

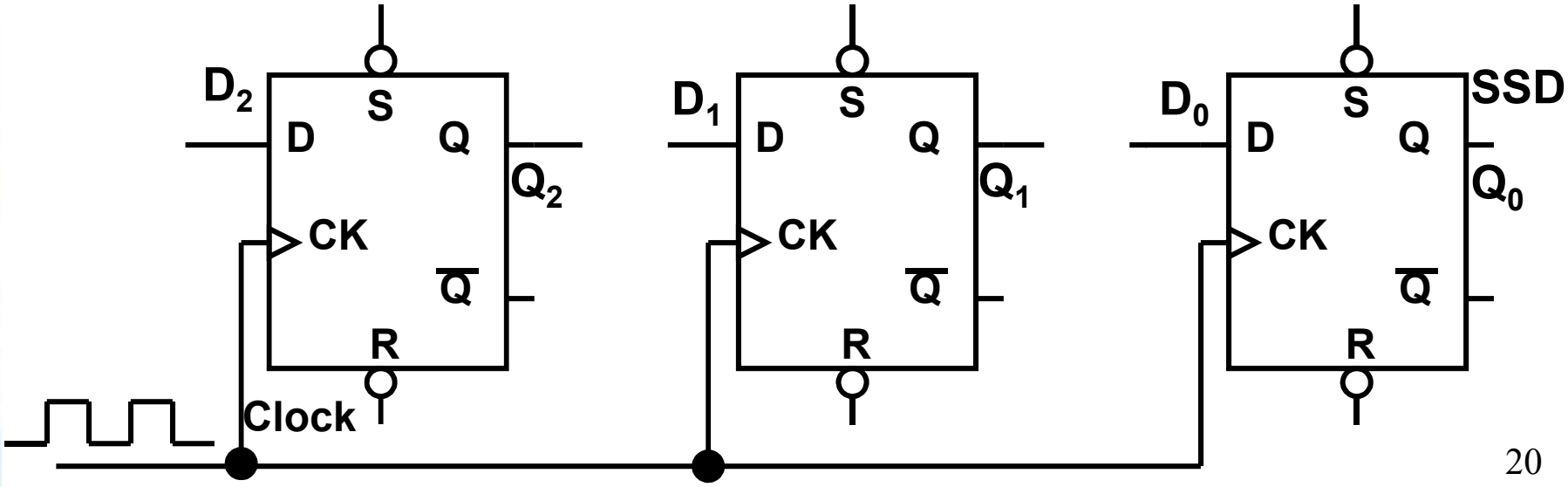
(E/S)SD: Entrada/Saída serial para deslocamento p/ direita;
SD/P': 1 deslocamento p/ direita; 0 p/ carga paralela;
$Ep_n$: Entrada de dados p/ carga paralela.



# Registradores: exercício 1

- Como dotar registrador de entrada série ou paralela?

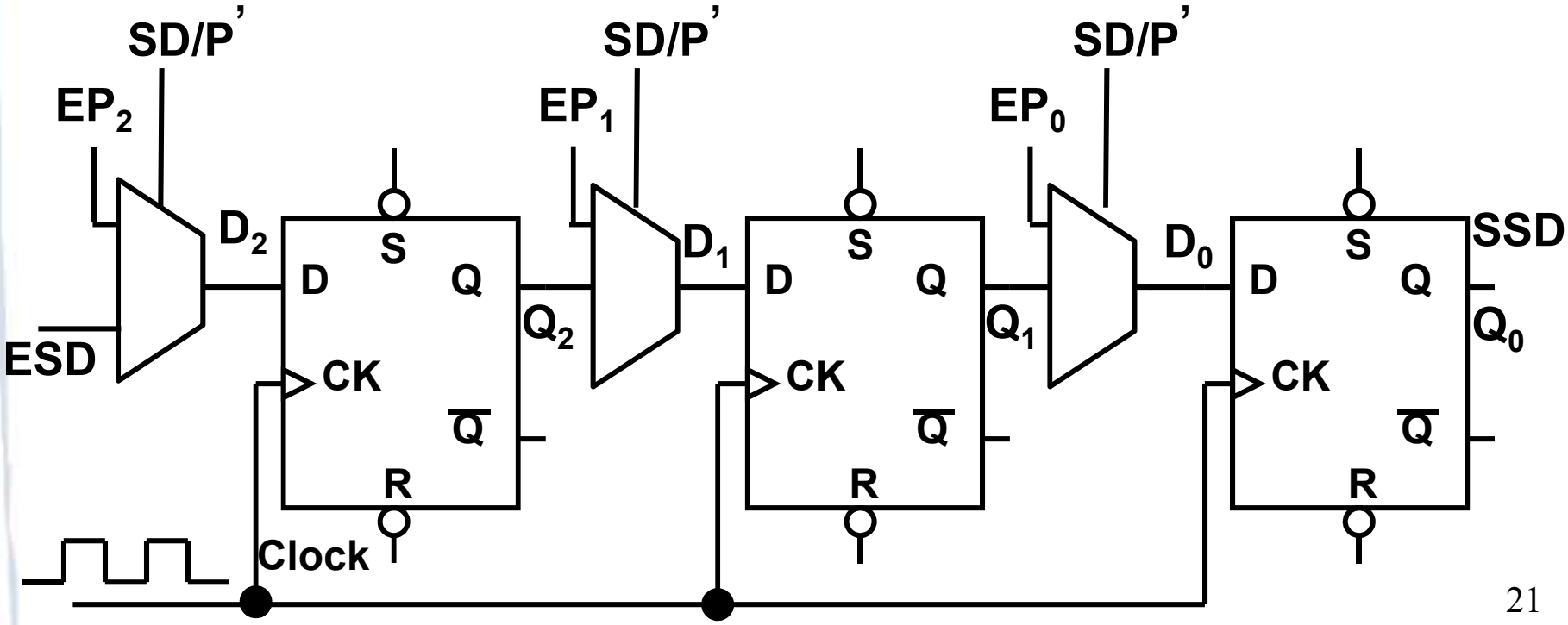
(E/S)SD: Entrada/Saída serial para deslocamento p/ direita;
SD/P': 1 deslocamento p/ direita; 0 p/ carga paralela;
$Ep_n$: Entrada de dados p/ carga paralela.



# Registradores: exercício 1

- Como dotar registrador de entrada série ou paralela?

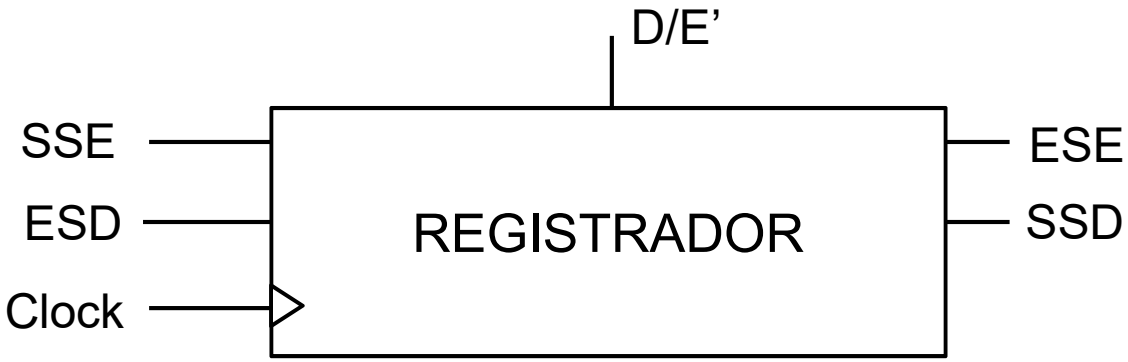
(E/S)SD: Entrada/Saída serial para deslocamento p/ direita;
SD/P': 1 deslocamento p/ direita; 0 p/ carga paralela;
$Ep_n$: Entrada de dados p/ carga paralela.



# Registradores: exercício 2

- E para fazer deslocamento para direita ou esquerda?

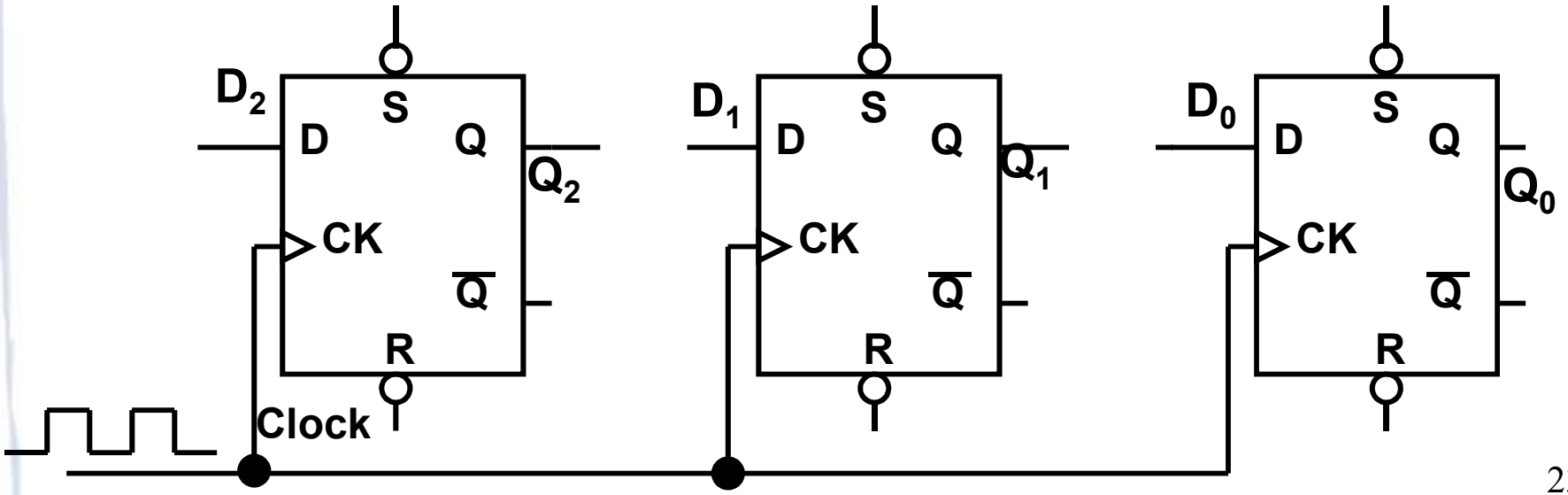
(E/S)SD: Entrada/Saída serial p/ deslocamento p/ direita;
(E/S)SE: Entrada/Saída serial p/ deslocamento p/ esquerda;
D/E': 1 = deslocamento p/ direita; 0 = deslocamento p/ esquerda



# Registradores: exercício 2

- E para fazer deslocamento para direita ou esquerda?

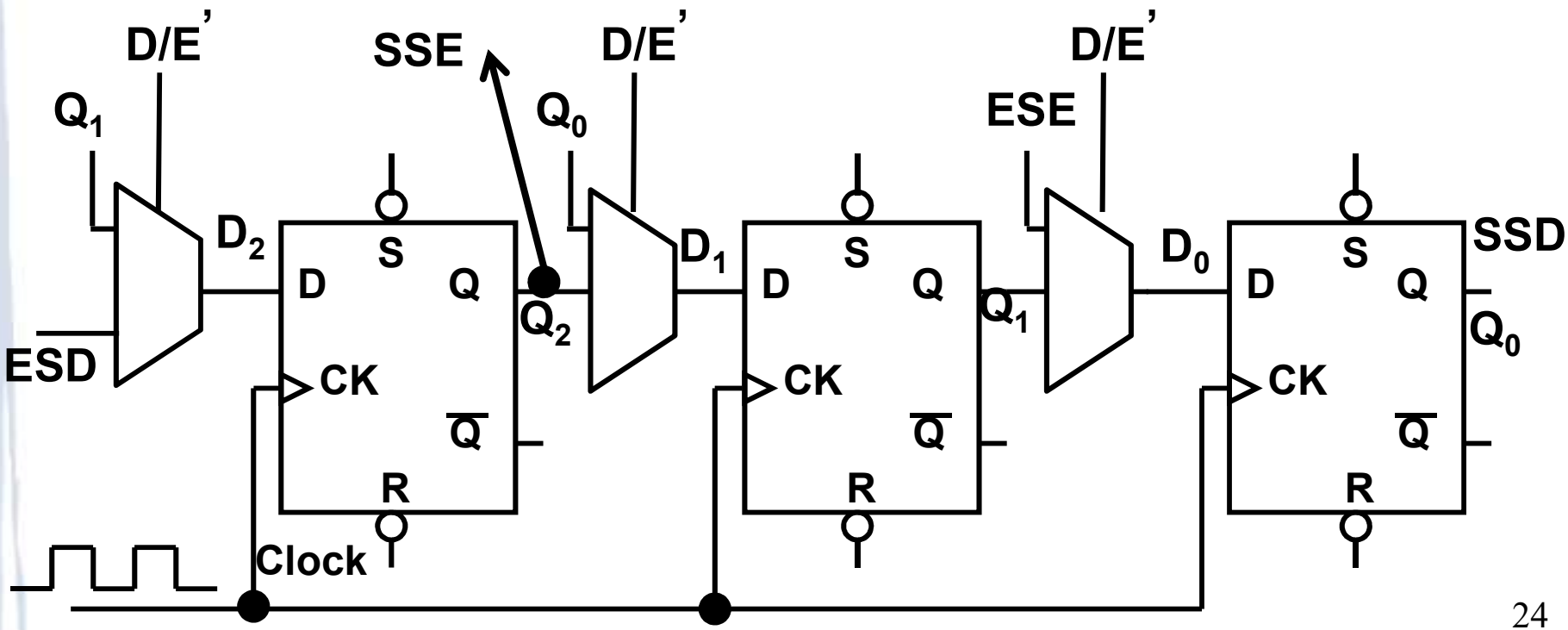
(E/S)SD: Entrada/Saída serial p/ deslocamento p/ direita;
(E/S)SE: Entrada/Saída serial p/ deslocamento p/ esquerda;
D/E': 1 = deslocamento p/ direita; 0 = deslocamento p/ esquerda



# Registradores: exercício 2

- E para fazer deslocamento para direita ou esquerda?

(E/S)SD: Entrada/Saída serial p/ deslocamento p/ direita;
(E/S)SE: Entrada/Saída serial p/ deslocamento p/ esquerda;
D/E': 1 = deslocamento p/ direita; 0 = deslocamento p/ esquerda

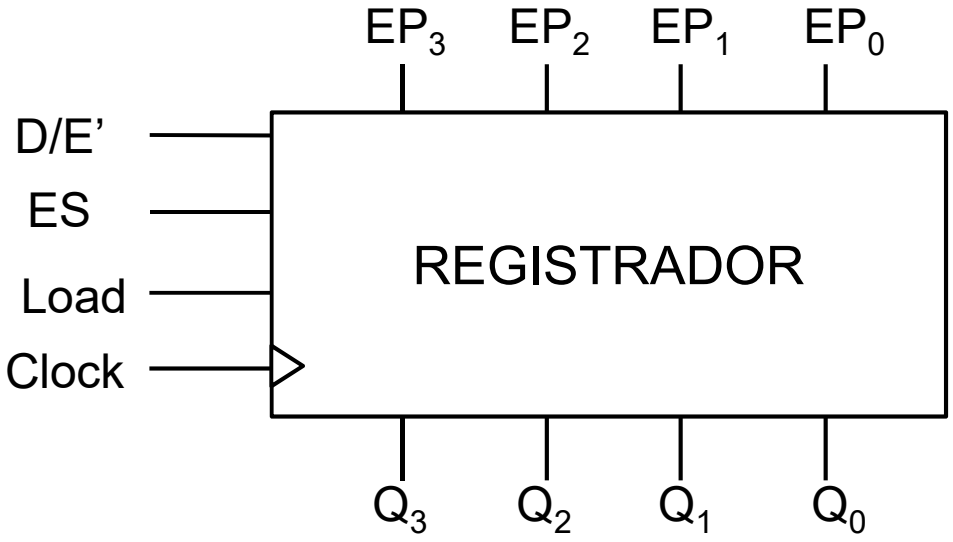


# Registradores: exercício 3

- Dado um registrador de deslocamento para direita/esquerda, implemente um circuito capaz de multiplicar o dividir por 2 a entrada carregada

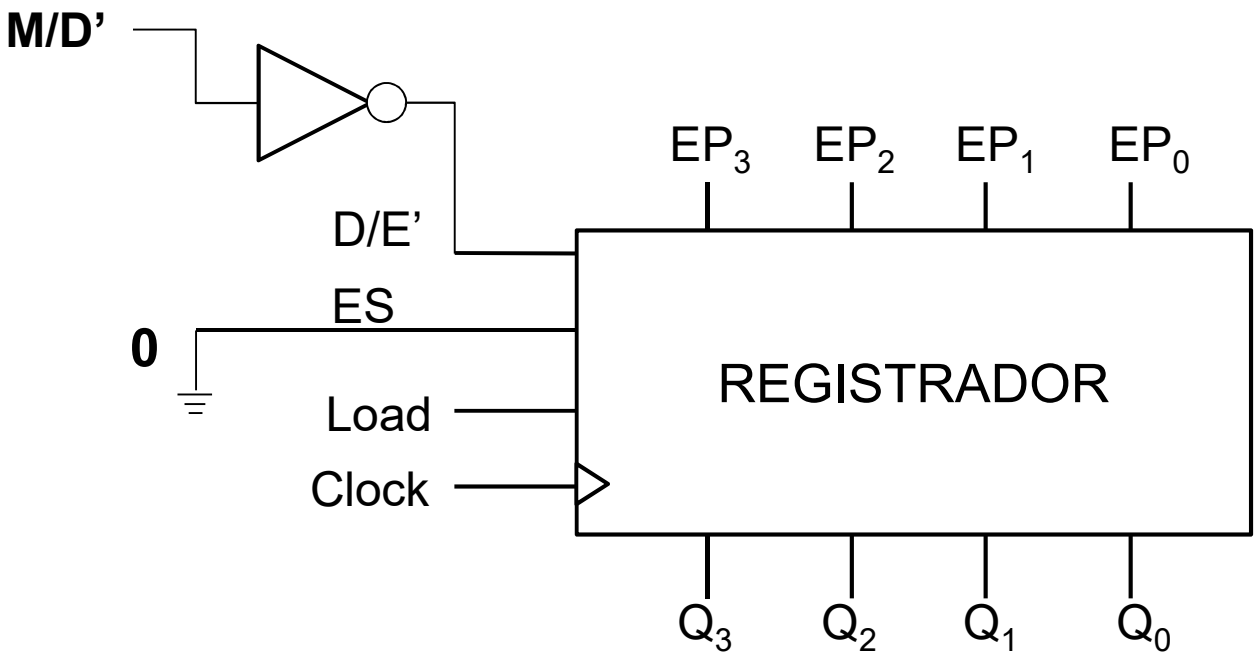
D/E’

M/D’ ?



# Registradores: exercício 3

- Dado um registrador de deslocamento para direita/ esquerda com entrada serial/paralela, implemente circuito que multiplica ou divide por 2 valor carregado



# Contadores

- **Conceito –** Grupos de Flip-Flops acionados por sinal comum de *clock*, que permanecem mudando de estado de acordo com uma sequência pré-estabelecida:
  - Dispensa, às vezes, outros tipos de sinais de entrada – Máquina de *Moore* (só o sinal de clock como uma pseudo-entrada)
  - Basicamente – Circuito Sequencial síncrono que possui uma sequência principal de **transição de estados cíclica, pré-definida.**

# Contadores

- **Classificação:**
  - **Sequência de Contagem** – Tipo de sequência que contador percorre: binária, BCD, código Gray, etc.;
  - **Módulo** – Quantidade de estados da sequência principal que o contador percorre em um ciclo completo de contagem:
    - **Cálculo do Módulo**:

      (Magnitude do Estado Final) menos
      (Magnitude do Estado Inicial) mais 1

# Contadores

- **Síncronos** – Para todos os Flip-Flops:
  - ✓ O *clock* é o mesmo sinal físico;
  - ✓ As saídas (estados) são atualizadas no mesmo instante (mesma borda de subida ou descida).

# Contadores

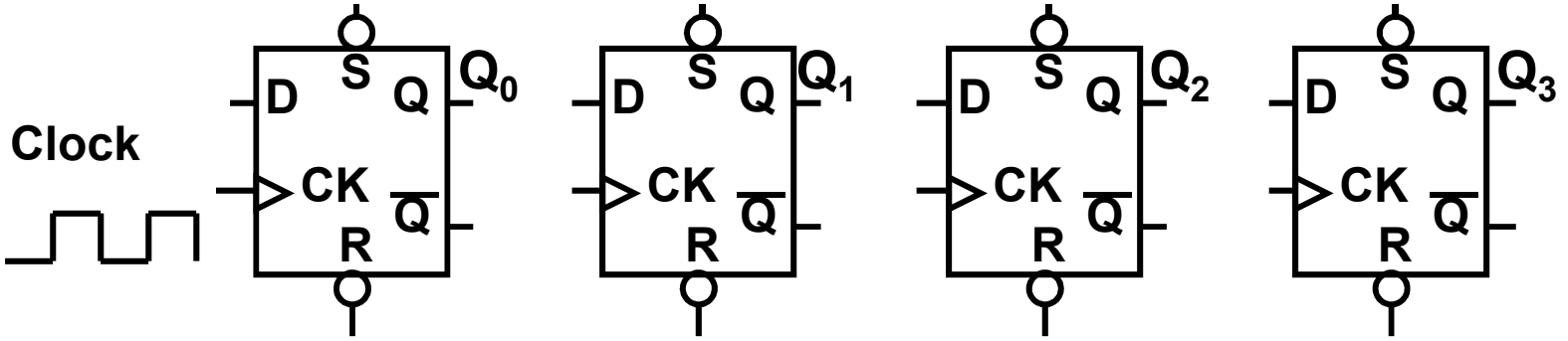
- **Assíncronos** (**via propagação – *ripple counters***):
  - ✓ Atualização dos Flip-Flops dos bits menos significativos **gera as bordas de *clock*** que atualizam os Flip-Flops dos bits mais significativos;
  - ✓ Normalmente usam **menos portas lógicas** que síncronos, porém fornecem **menor velocidade;**
  - ✓ Faz uso de **técnicas de projeto mais intuitivas e menos formais**, dependendo da experiência do projetista.

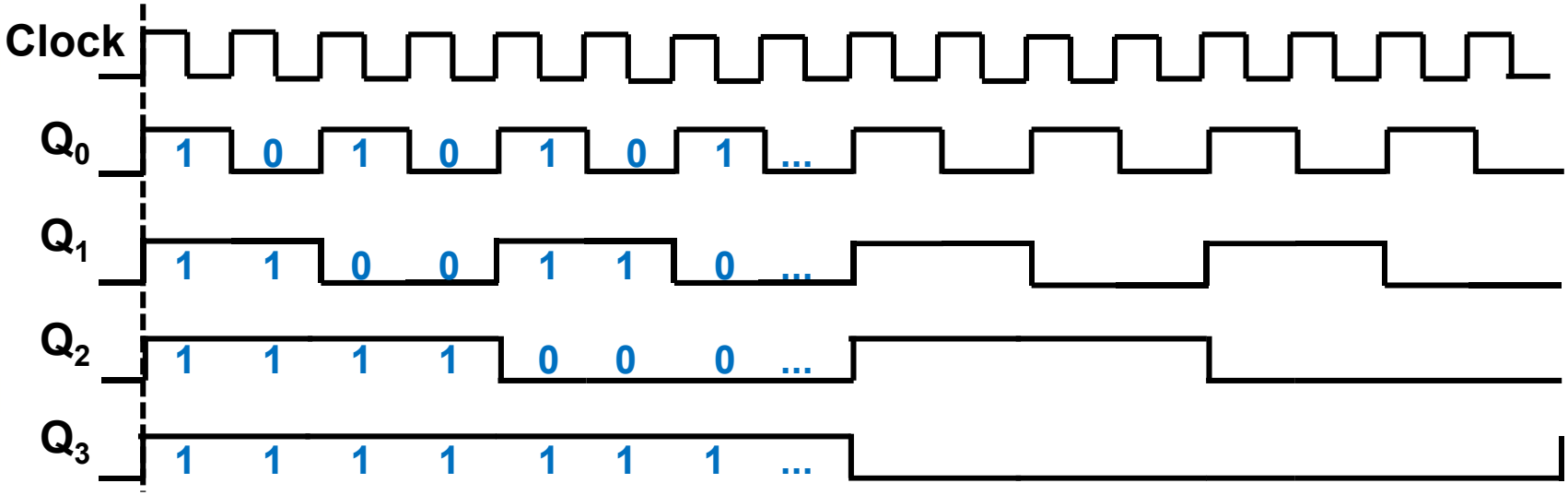
# Contadores

## O contador não precisa de entradas, mas ...

- Para maior **versatilidade**, são comuns algumas **entradas de sinais de controle**:
  - ***Set* e *Reset*** – Carga de zeros ou uns;
  - ***Load*** – Carga de valor específico de contagem;
  - ***Hold*** – Interrompe contagem (ignora *clock*);
  - ***Output Enable*, *Clock Enable*** – Habilitação de saída e *clock*;
  - ***Up/Down*** – Dita a sequência de contagem (se é de baixo para cima ou de cima para baixo).

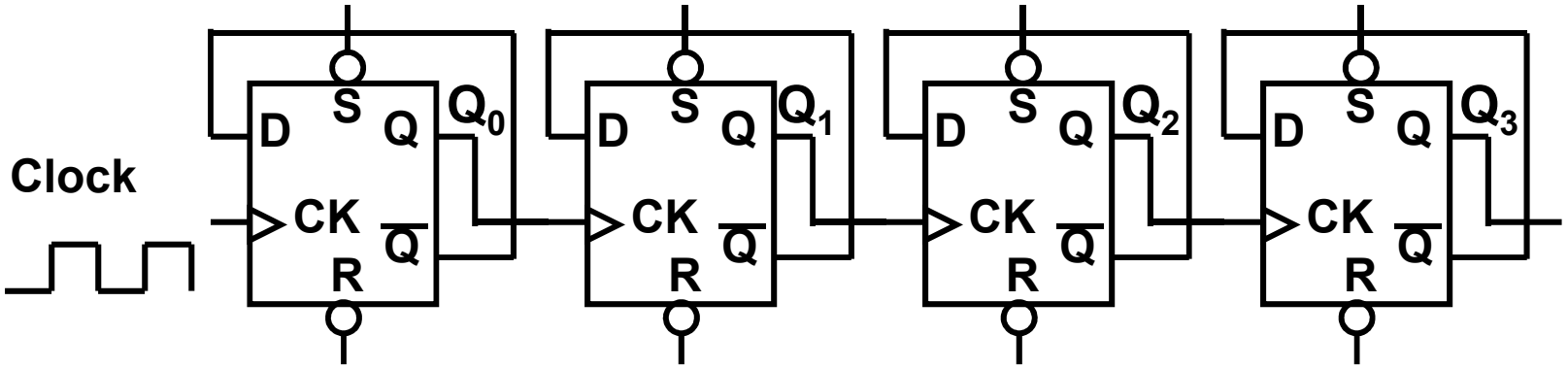
# Contadores Assíncronos

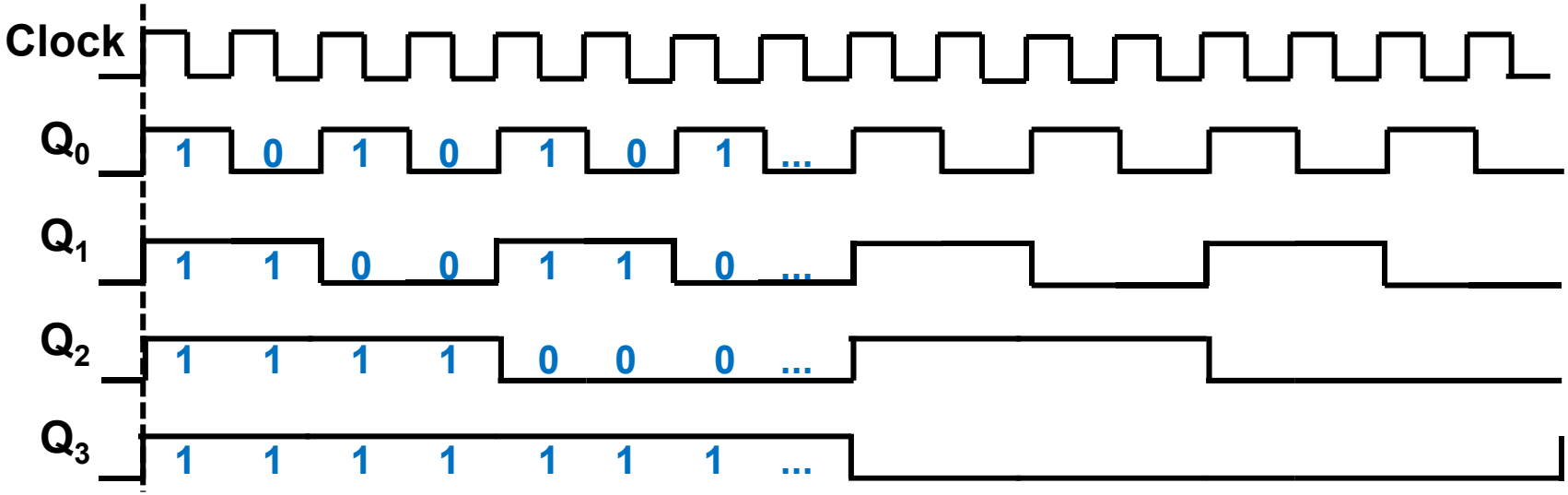



- **Pergunta**: projeto de contador assíncrono decrescente.

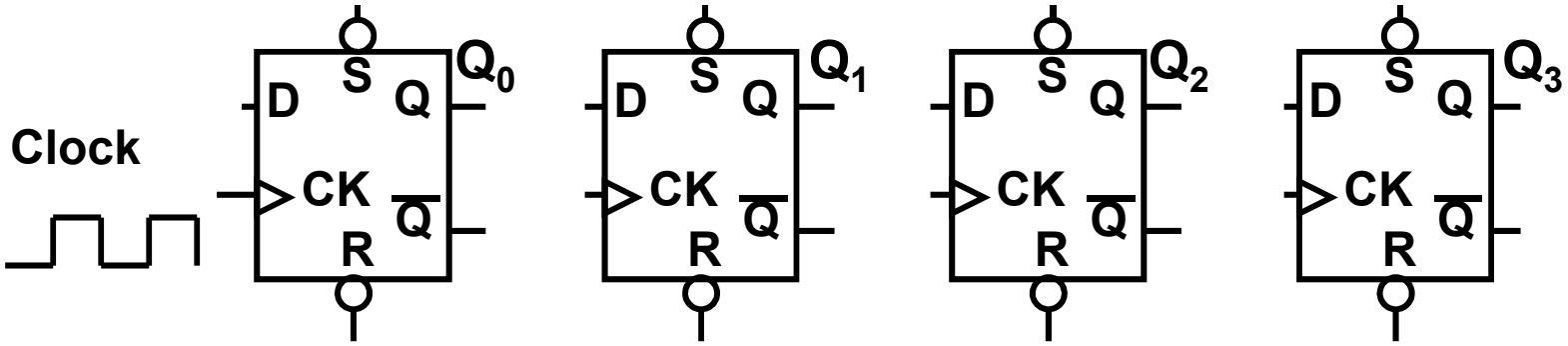
# Contadores Assíncronos

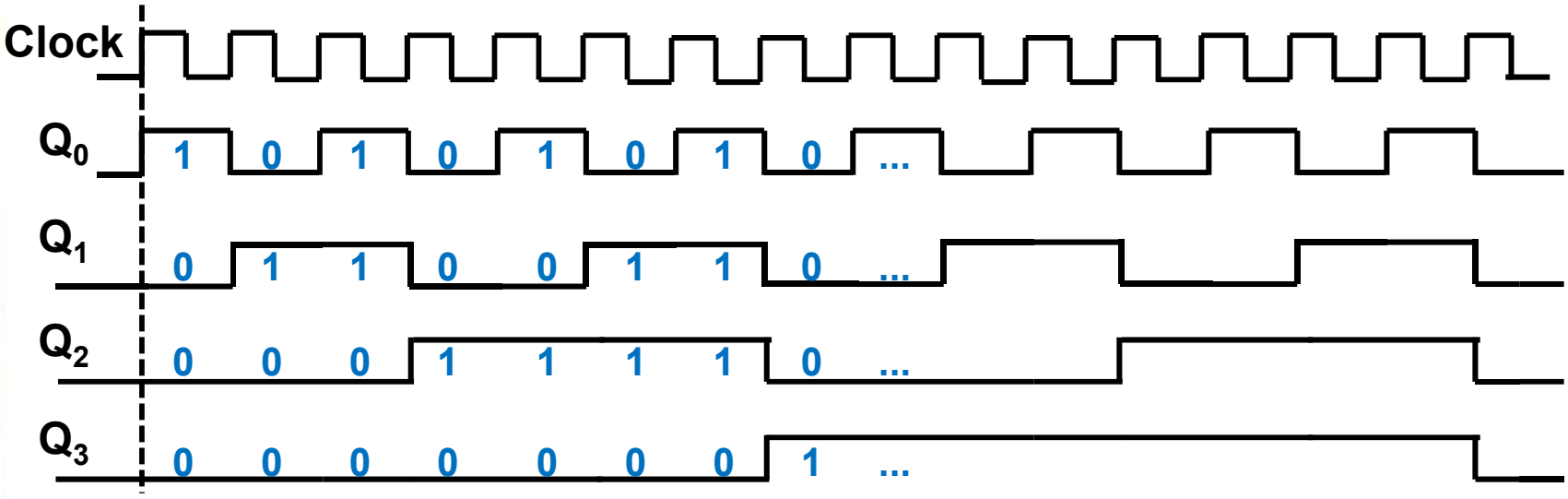



- **Resposta**: clock a partir de borda de Q

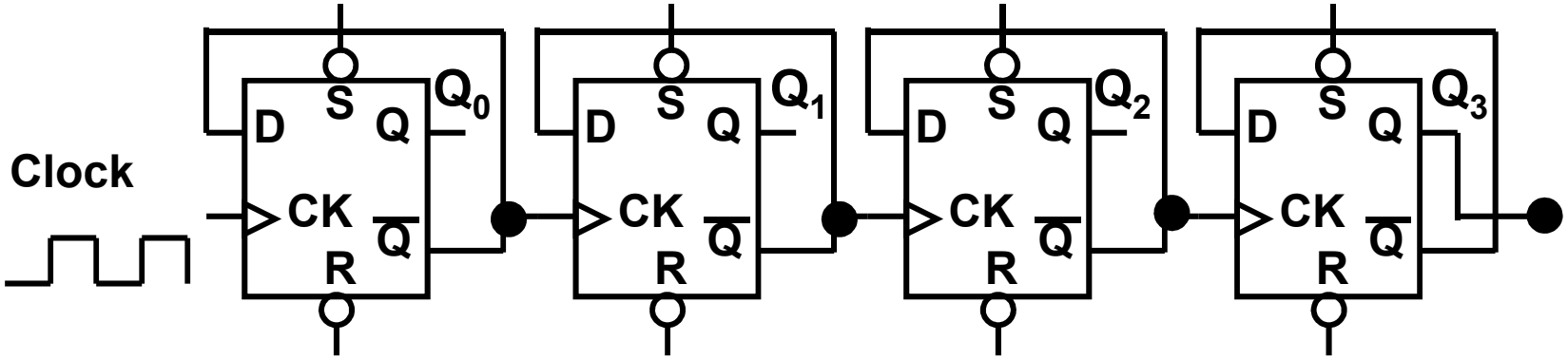
# Contadores Assíncronos

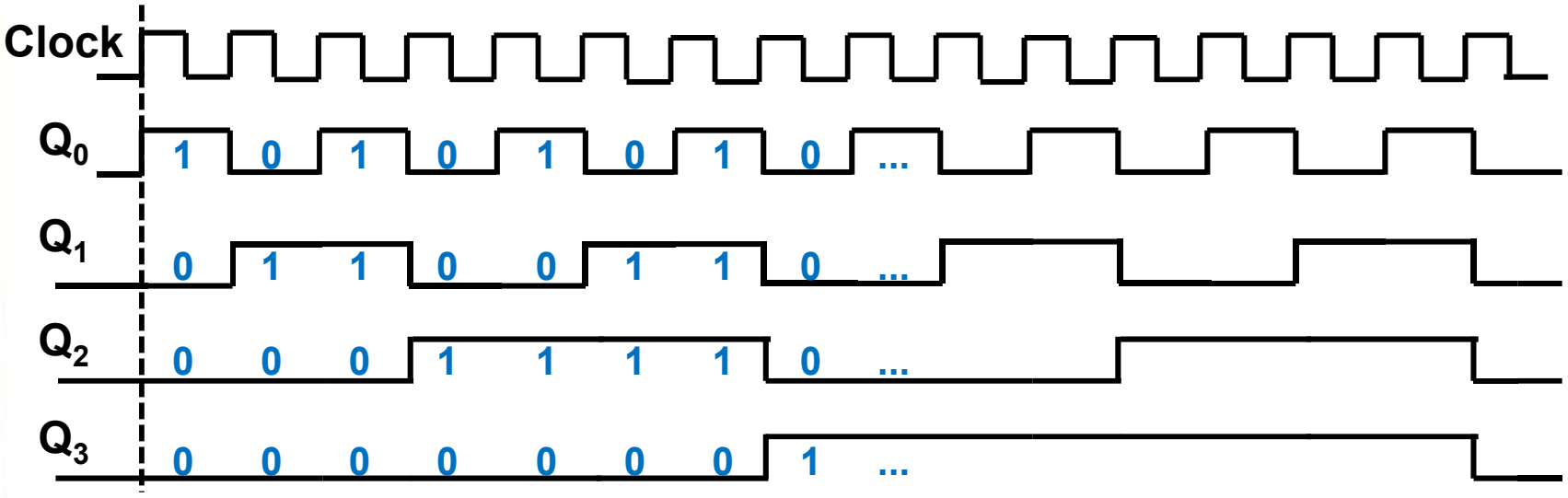



- **Pergunta**: projeto de contador assíncrono crescente.

# Contadores Assíncronos

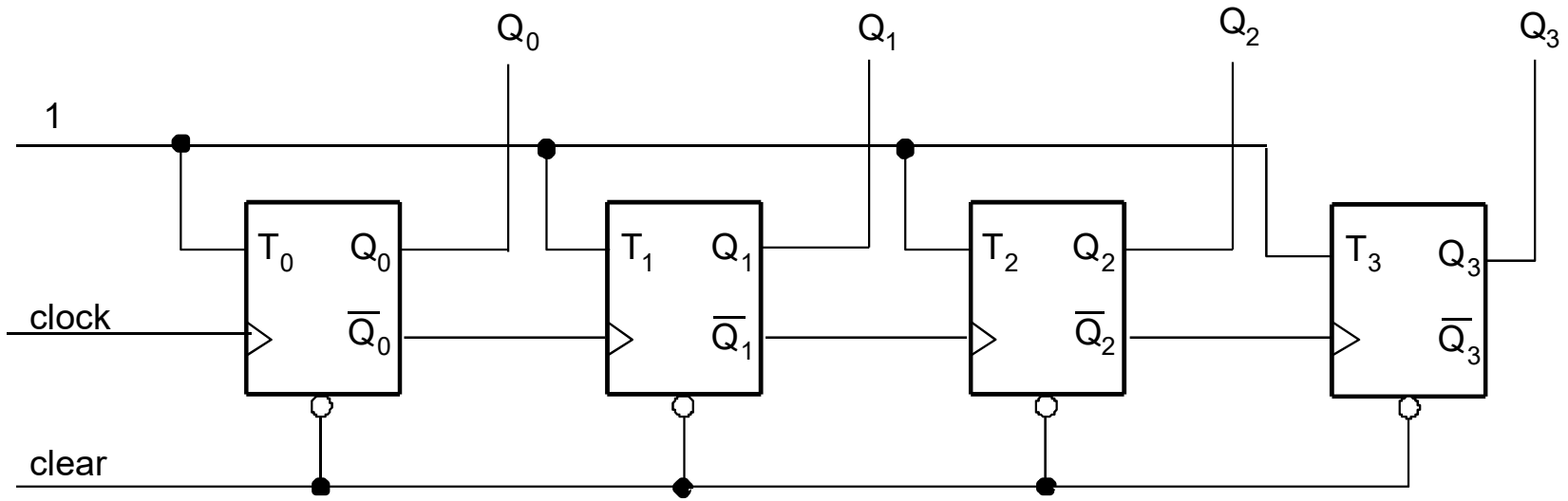



- **Resposta**: clock a partir de borda de Q'

# Contadores Assíncronos

- **<u>Outro exemplo:</u>** Contador assíncrono crescente com **Flip-Flops tipo T**.



# Contadores Assíncronos

- Contador de década, BCD, ou divisor de frequência por 10:

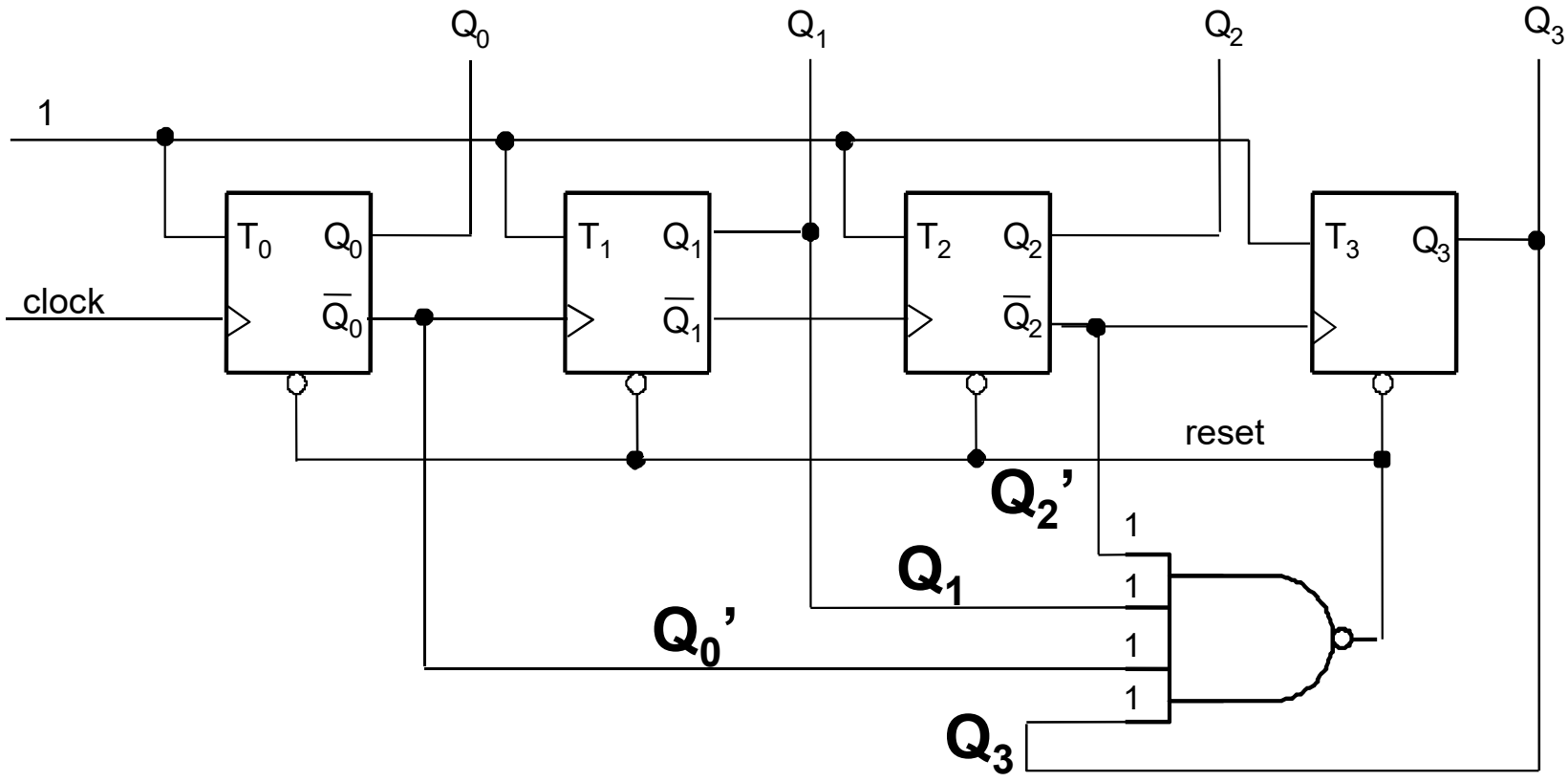
| Q3 | Q2 | Q1 | Q0 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |

**reset**

- Conta de 0 a 9: deve permanecer em cada estado por pelo menos um período de *clock* completo;
- Ao detectar o estado 10, carrega estado 0 e retoma a contagem assíncrona
- **Nota**: O contador permanece no estado 10 por um período de tempo igual ao necessário para sinal *Reset* atuar.

# Contadores Assíncronos

- Contador de década: conta de 0 a 9
  - Obs.: reset síncrono com detector de “9” teria efeito similar



# Contadores Assíncronos

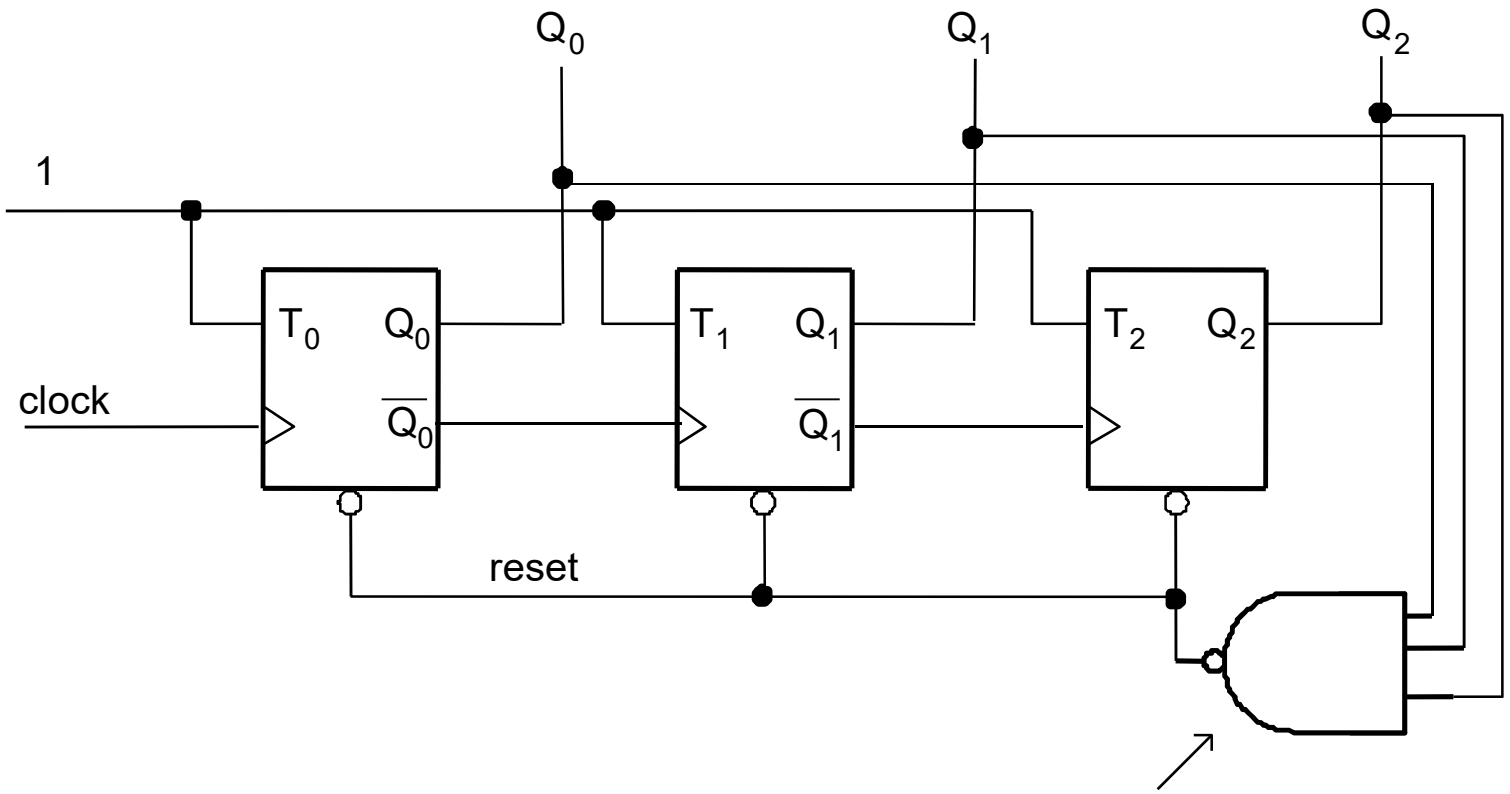
- **Exercício 1:** como seria um **contador de 0 a 6?**
  - Dica: a lógica é semelhante à de 0-9…

- **Exercício 2:** como seria um **contador módulo 16 decrescente** com **Flip Flops tipo T?**
  - Dica: $16 = 2^4$

# Contadores Assíncronos

**Exercício 1:** como seria um **contador de 0 a 6?**

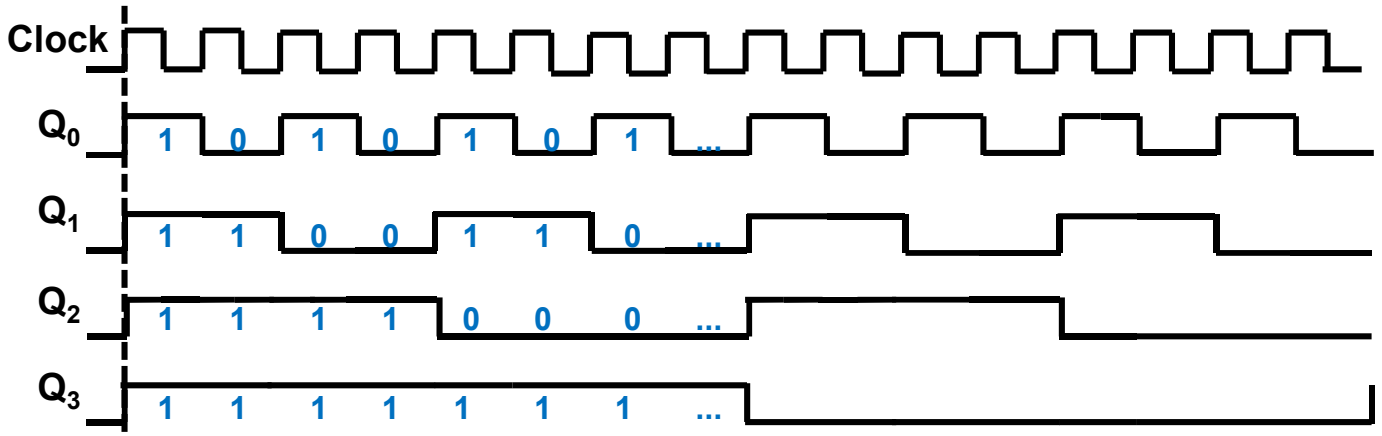
**Solução: contagem crescente**

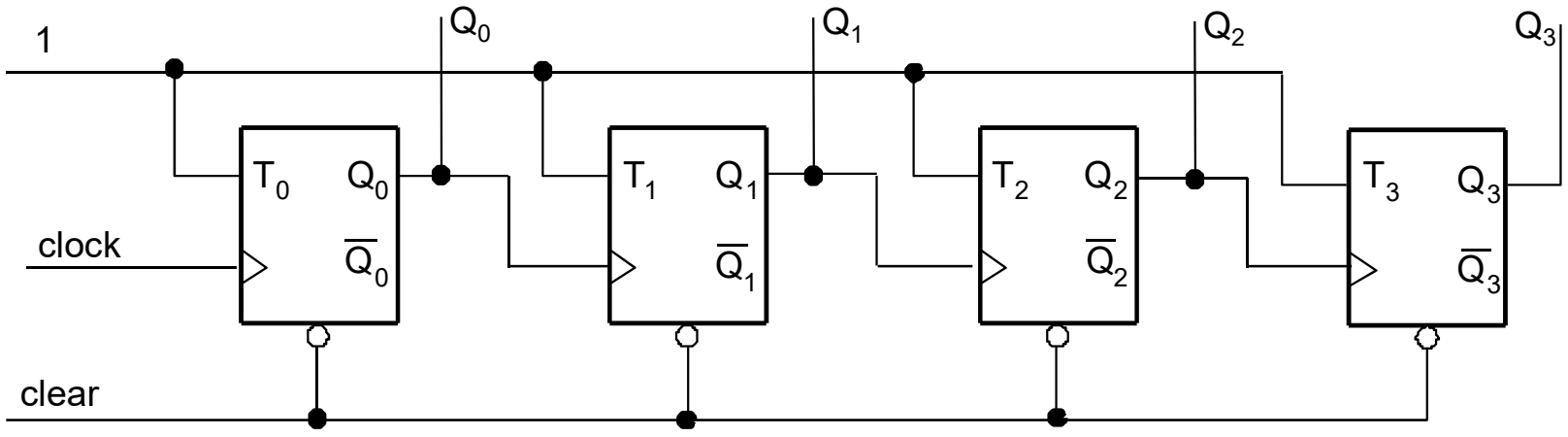



**Detecta o estado 7: “111”.**

# Contadores Assíncronos

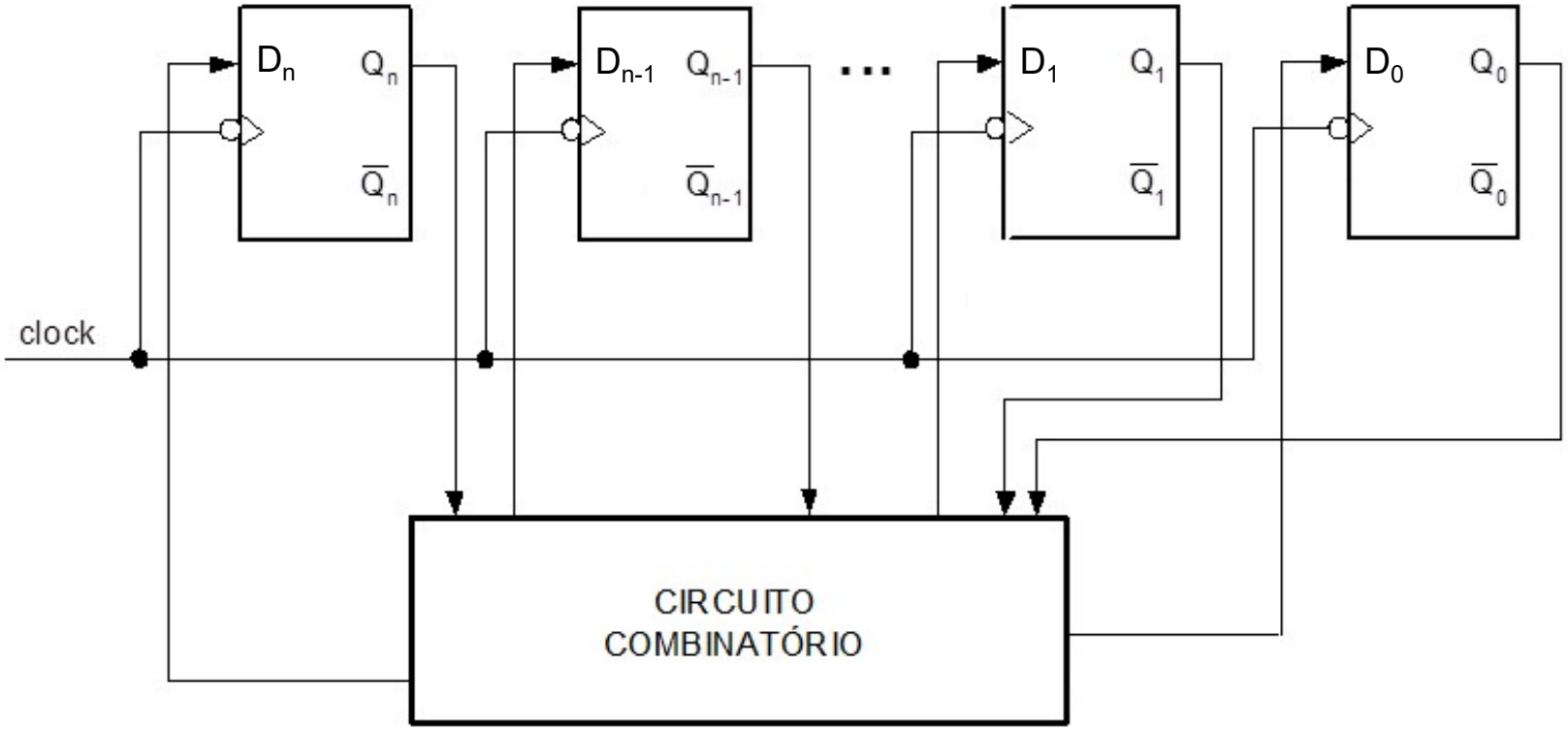
## Ex. 2: contador módulo 16 decrescente



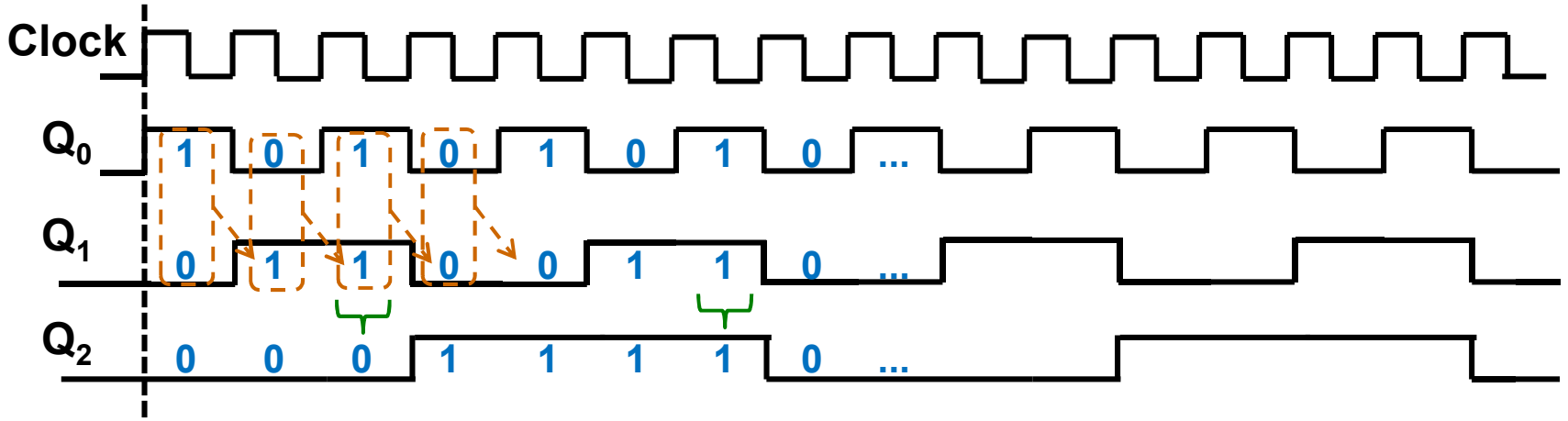
# Contadores Síncronos

- Todos os FFs – (1) compartilham um sinal de *clock* e (2) têm suas saídas (estados) atualizadas no mesmo instante (mesma borda) ➔ **Projeto de Circuito Síncrono!**



# Contadores Síncronos: projeto

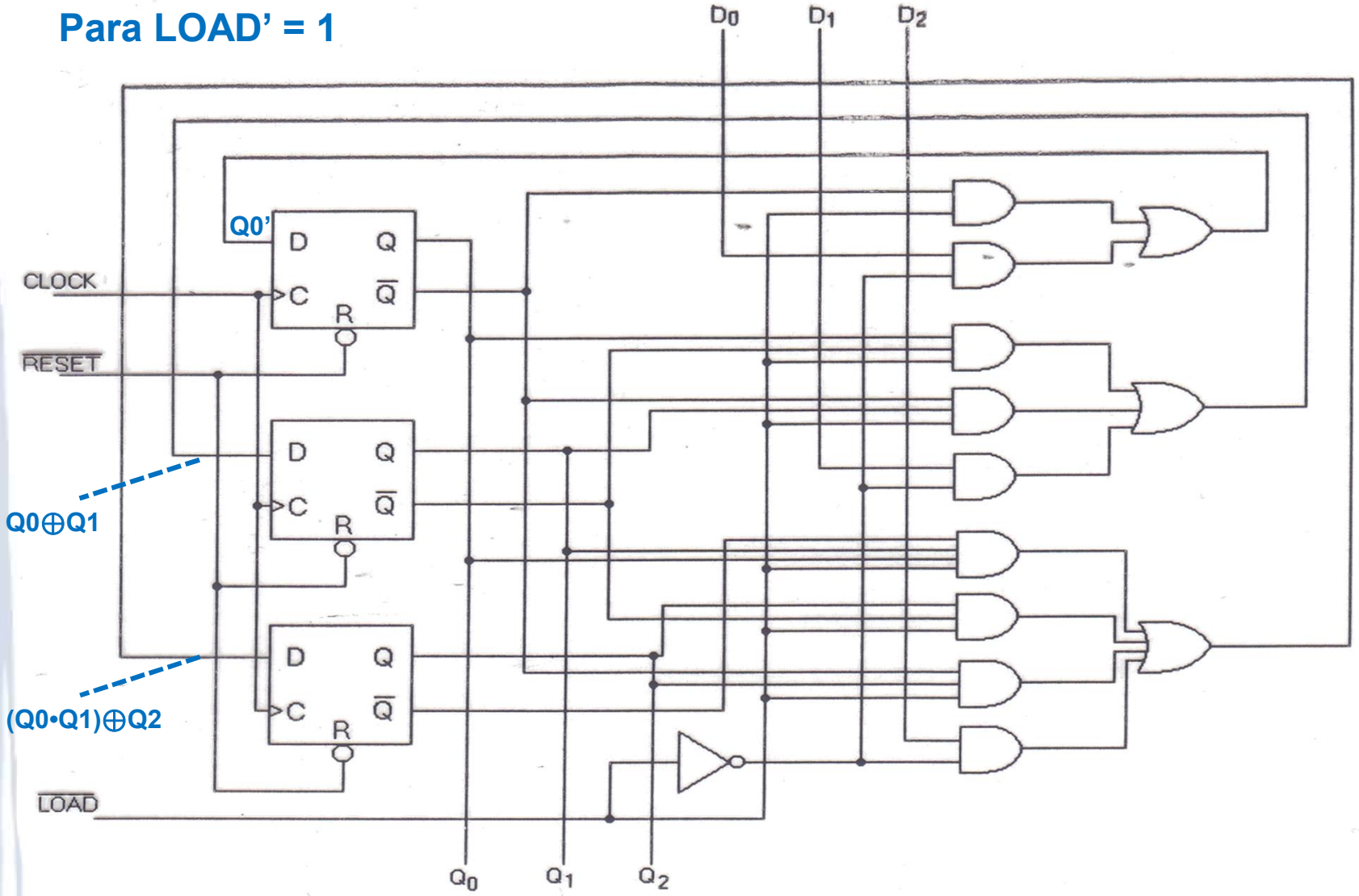
- Flip-Flops tipo D – Como seria a alimentação em cada $D_i$ para obter comportamento síncrono ...?



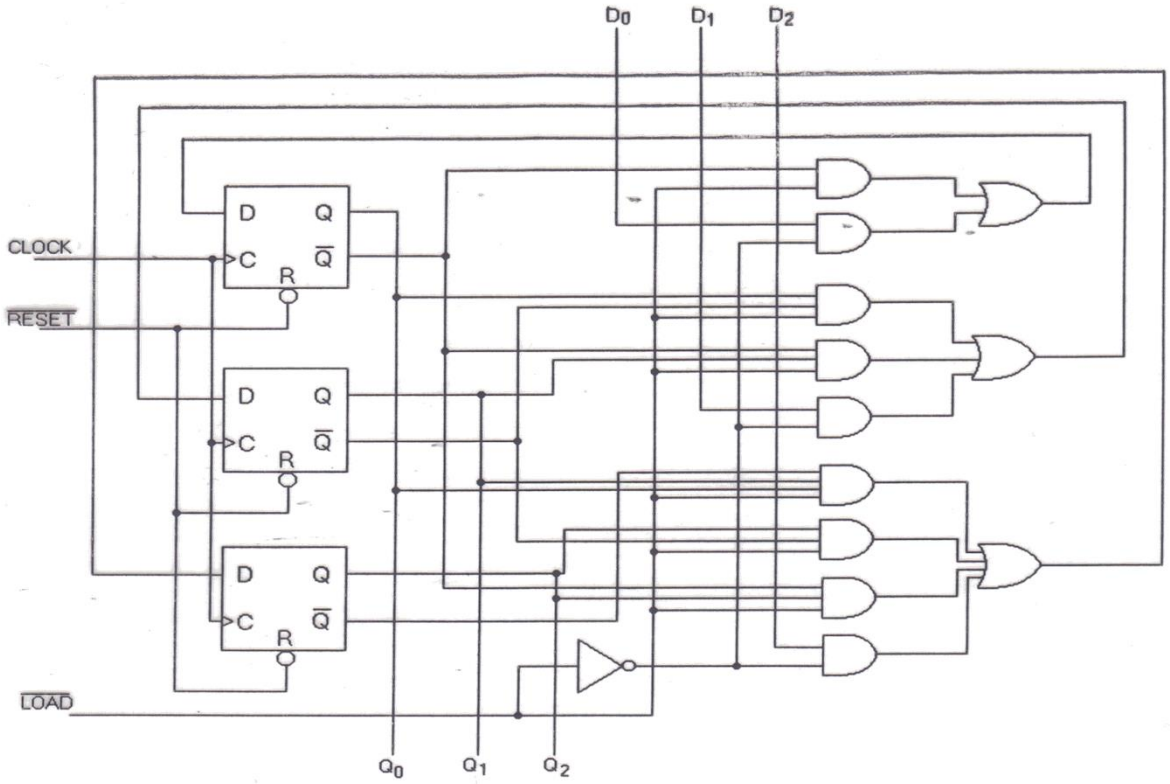

```
D0 <= Q0'
D1 <= Q0 xor Q1  -- se Q0 é 1, inverte Q1
D2 <= (Q0 and Q1) xor Q2 -- se Q0 e Q1 são 1,
                            inverte Q2 ...
                         -- se anteriores forem 1,
                             inverte Qn
```

# Contadores Síncronos: projeto

Para LOAD' = 1



# Contadores Síncronos: projeto




**Load' = 0 ➔ $IN_0 = D_0$ ; $IN_1 = D_1$ ; $IN_2 = D_2$**

**Load' = 1 ➔**

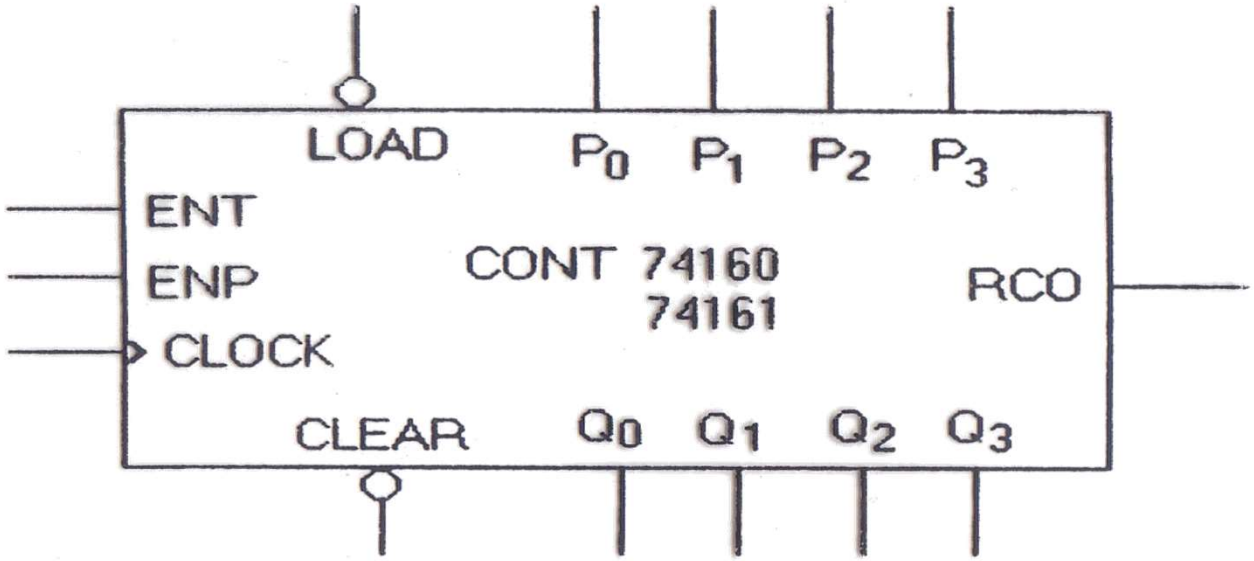
$IN_0 = Q_0'$

$IN_1 = Q_0 \cdot Q_1' + Q_0' \cdot Q_1; = Q_0 \oplus Q_1$

$IN_2 = Q_0 \cdot Q_1 \cdot Q_2' + Q_1' \cdot Q_2 + Q_0' \cdot Q_2 = (Q_0 \cdot Q_1) \oplus Q_2$

# Contadores Síncronos: exemplos

- 74160 – Contador módulo 10, sequência BCD;
- 74161 – Contador binário módulo 16 (hexa).
  - ENT e ENP: enable
  - RCO: ripple carry out (“vai um”)



# Contadores Síncronos: exemplos

| ENTRADAS | | | | | FUNÇÃO ASSOCIADA | EFEITO |
|---|---|---|---|---|---|---|
| $\overline{CLEAR}$ | $\overline{LOAD}$ | ENP | ENT | CLOCK | | $Q_0 - Q_3$ |
| 0 | X | X | X | X | ANULA | 0 – 0 |
| 1 | 0 | X | X | ↑ | CARREGA | $P_0 - P_3$ |
| 1 | 1 | 0 | X | X | INIBE | $Q_0 - Q_3$ |
| 1 | 1 | X | 0 | X | INIBE | $Q_0 - Q_3$ |
| 1 | 1 | 1 | 1 | ↑ | CONTA | $(Q_0 - Q_3)_{n+1}$ |

# OBS Nº1:

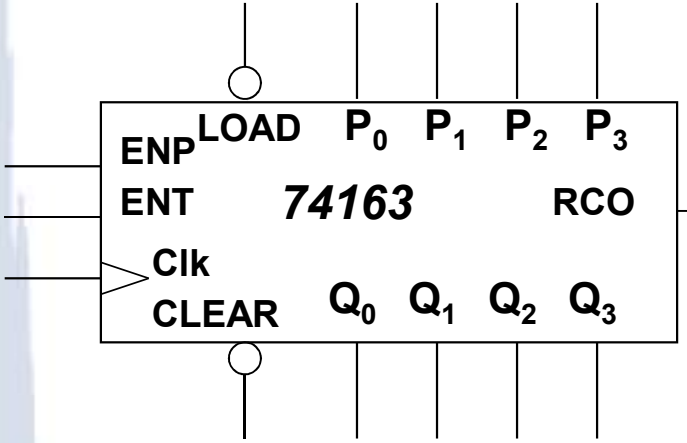
(i) 7416*: CONTADORES 74160 OU 74161

Decimal: (ii) 74160: $RCO = Q_0 \cdot \overline{Q_1} \cdot \overline{Q_2} \cdot Q_3 \cdot ENT$ **Inibe**

Hexa: (iii) 74161: $RCO = Q_0 \cdot Q_1 \cdot Q_2 \cdot Q_3 \cdot ENT$ **RCO**

# Contadores Síncronos: exercício

- Projetar **contador módulo 11** (conta dos Estados 0 até 10) com 74163.

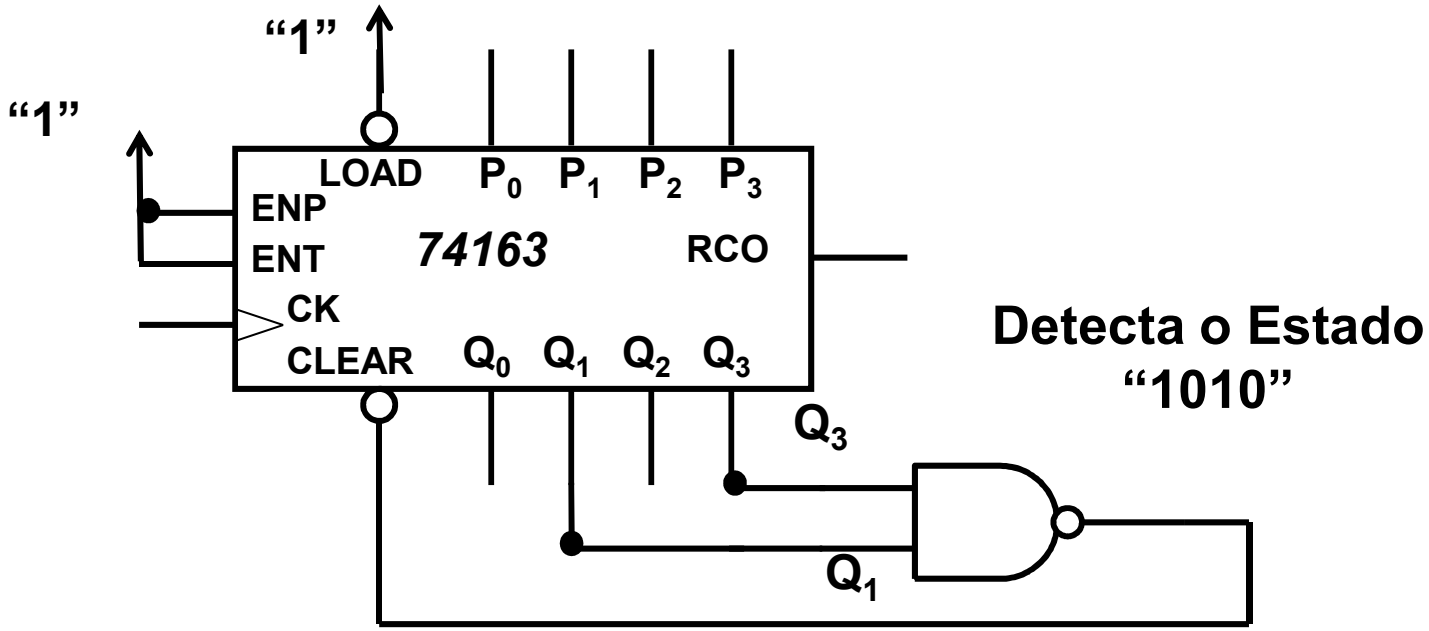



| ENTRADAS | | | | | FUNÇÃO | EFEITO |
|---|---|---|---|---|---|---|
| CLEAR | LOAD | ENP | ENT | CLOCK | ASSOCIADA | $Q_0 - Q_3$ |
| 0 | X | X | X | | ANULA | $Q_0 - Q_3$ |
| 1 | 0 | X | X | | CARREGA | $Q_0 - Q_3$ |
| 1 | 1 | 0 | X | X | INIBE | $Q_0 - Q_3$ |
| 1 | 1 | X | 0 | X | INIBE | $Q_0 - Q_3$ |
| 1 | 1 | 1 | 1 | | CONTA | $(Q_0 - Q_3)^{n+1}$ |

- **1) 74163: Contador módulo 16 (hexadecimal), sequência binária, com clear síncrono**
- **2) RCO: "Ripple Carry Out";**
- **3) ENT: "Enable Trickle Input";**
- **4) ENP: "Enable Parallel Input";**
- **5) RCO = Q0 . Q1 . Q2 . Q3 . ENT**

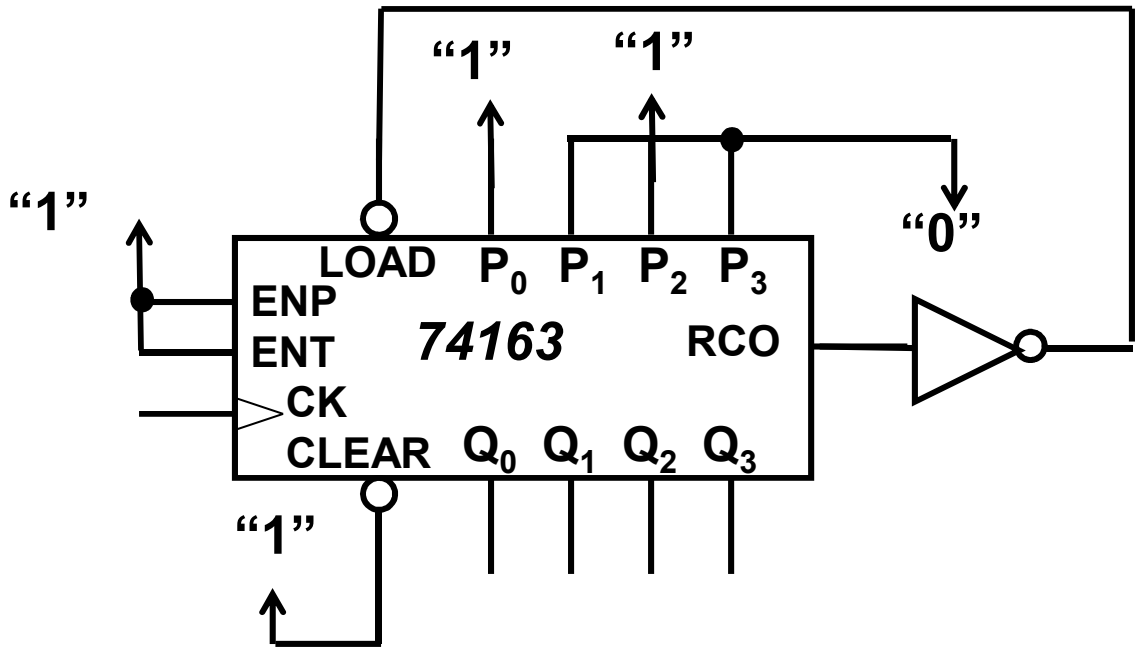
# Contadores Síncronos: exercício

- Projetar **contador módulo 11** (conta dos Estados 0 até 10) com 74163.



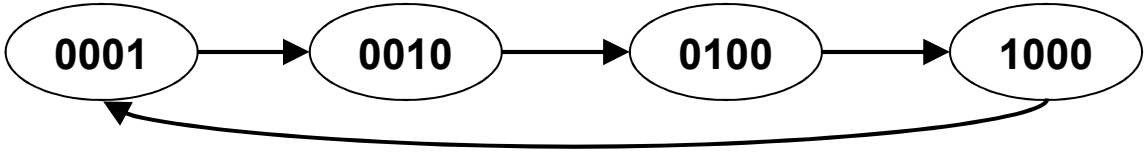
# Contadores Síncronos: exercício

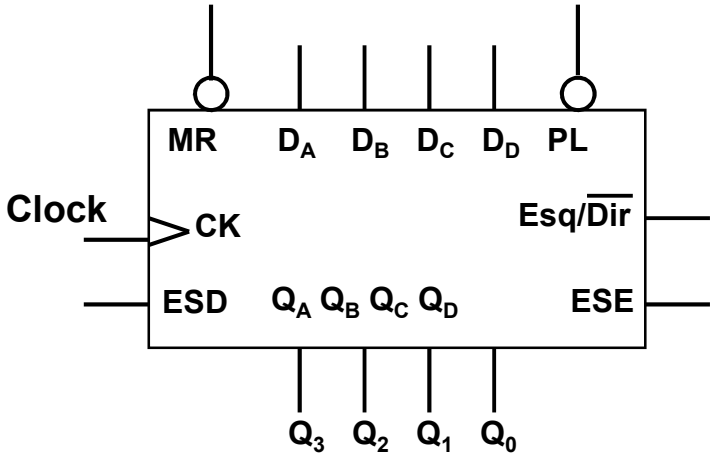
- **Alternativa** – Contador **módulo 11**
  - Conta dos Estados **5 até 15**
  - Usa o sinal de **RCO** para detectar o Estado **15**, ligando-o (invertido) ao **LOAD** para carregar o Estado **5**.



# Exercício

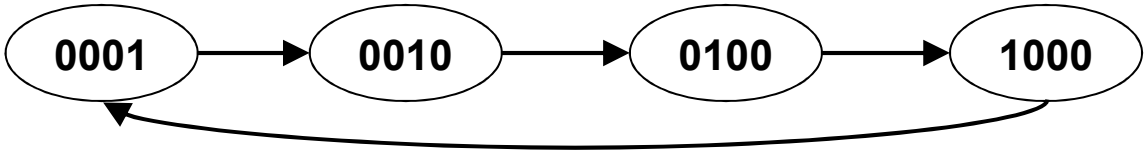
- Usando registrador de deslocamento abaixo, contruir contador síncrono com a sequência de contagem ($Q_3Q_2Q_1Q_0$):

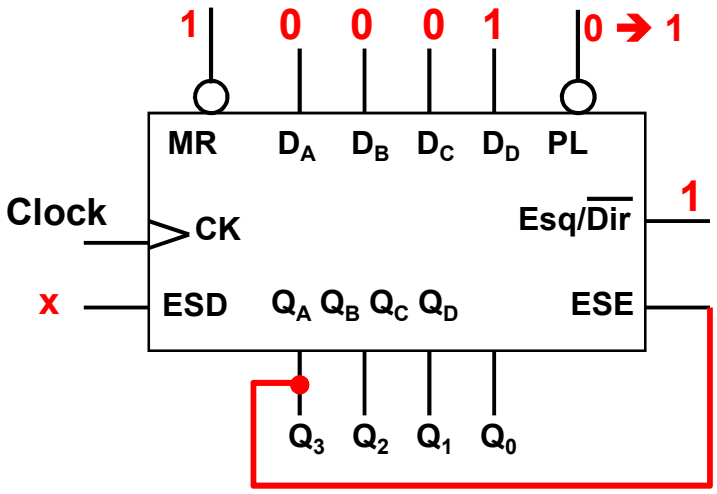


# Exercício

- Usando registrador de deslocamento abaixo, contruir contador síncrono com a sequência de contagem ($Q_3Q_2Q_1Q_0$):
  - Obs.: no início, fazer LOAD de 0001

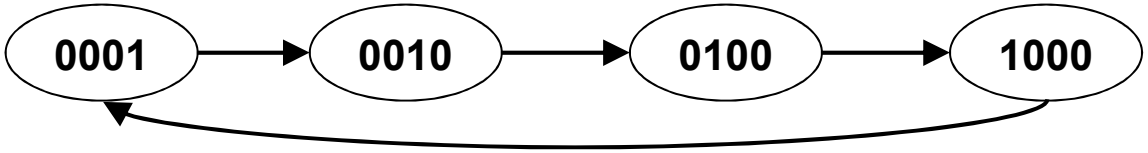


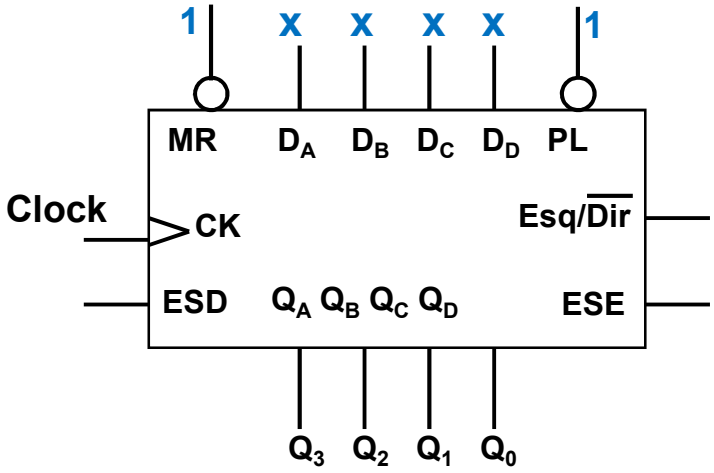




**"Deslocador em anel"**

# Exercício

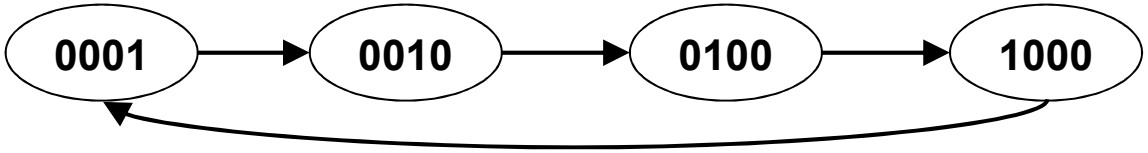
- Reprojetar deslocador em anel do item anterior para se "auto-inicializar": independente de valor inicial, atinge 0001 em algum momento e faz contagem abaixo
  - Ou seja: início pode ser 0000, 1111, ou qualquer outro valor

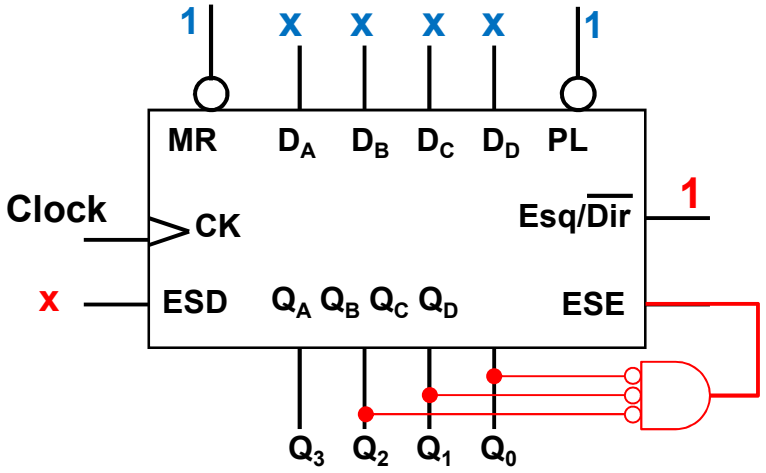


# Exercício

- Reprojetar deslocador em anel do item anterior para se "auto-inicializar": independente de valor inicial, atinge 0001 em algum momento e faz contagem abaixo
  - Ou seja: início pode ser 0000, 1111, ou qualquer outro valor

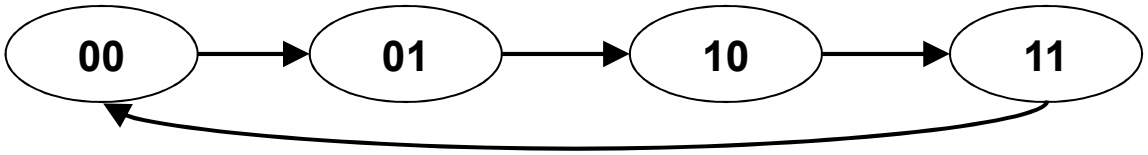


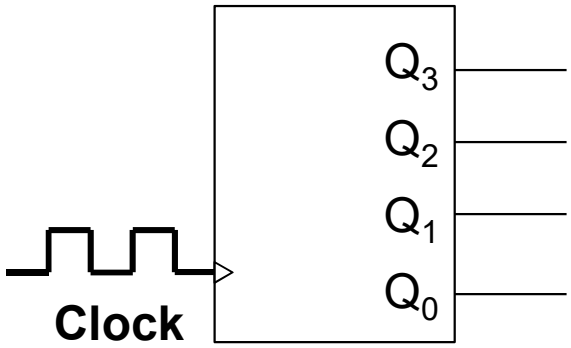




Realimenta: 1 se X000;
0 caso contrário

# Exercício

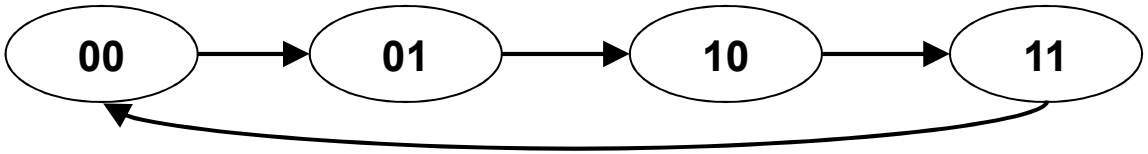
- Usando deslocador em anel do item anterior, criar um contador módulo 4:

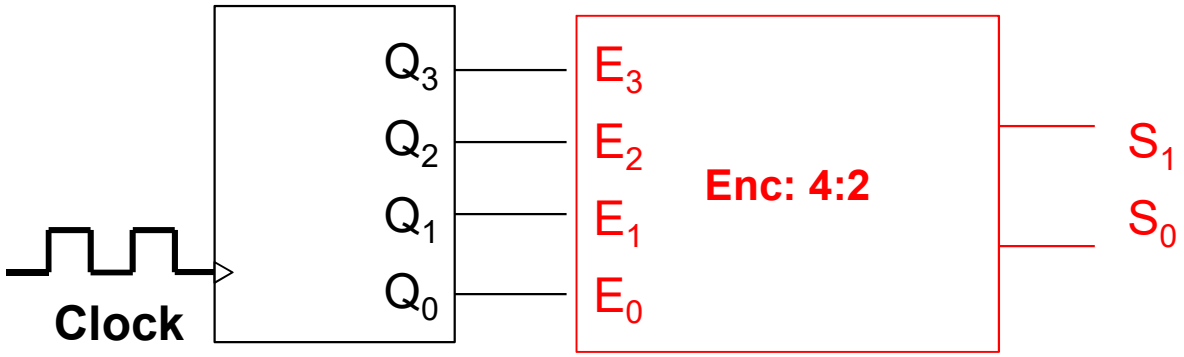


# Exercício

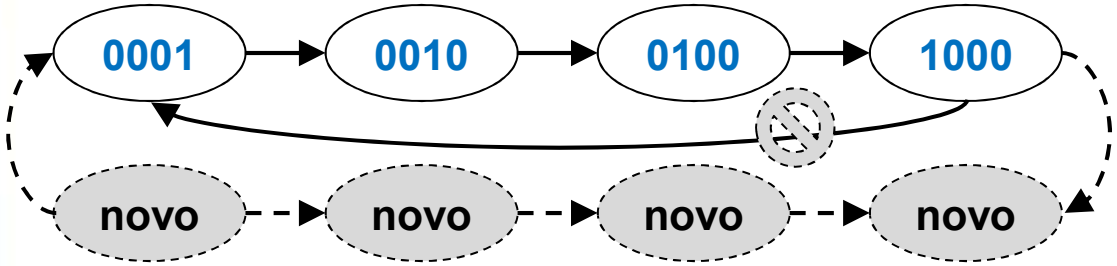
- Usando deslocador em anel do item anterior, criar um contador módulo 4 ($D_1D_0$):



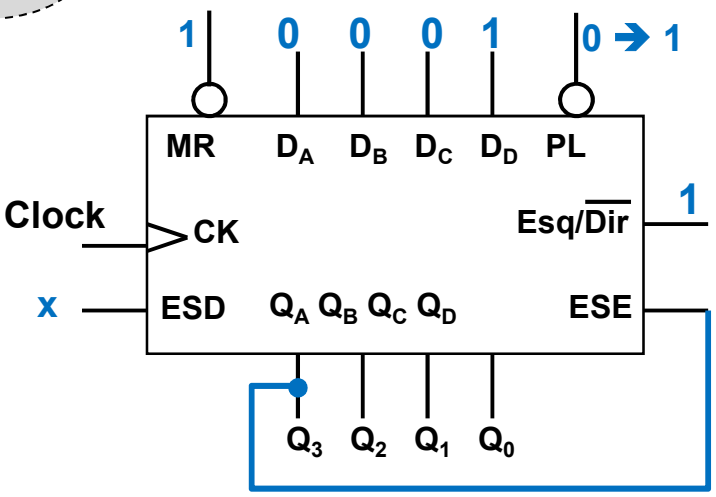



# Exercício

- Reprojetar deslocador em anel original, para duplicar número de estados da contagem. Use no máximo 1 porta lógica adicional (AND, OR ou NOT)
  - Sequência de contagem pode ser diferente da original

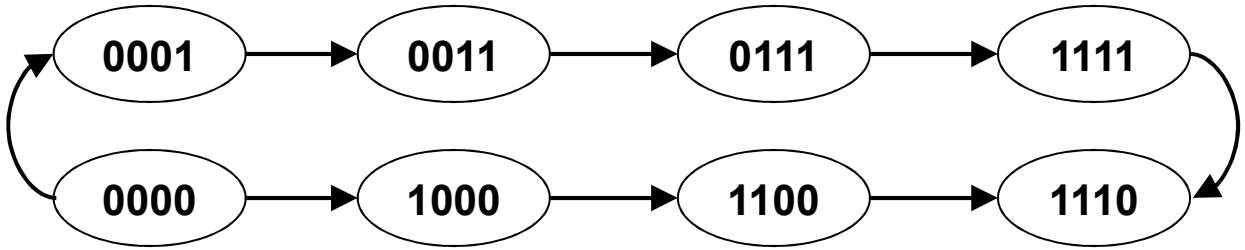



**Original:**

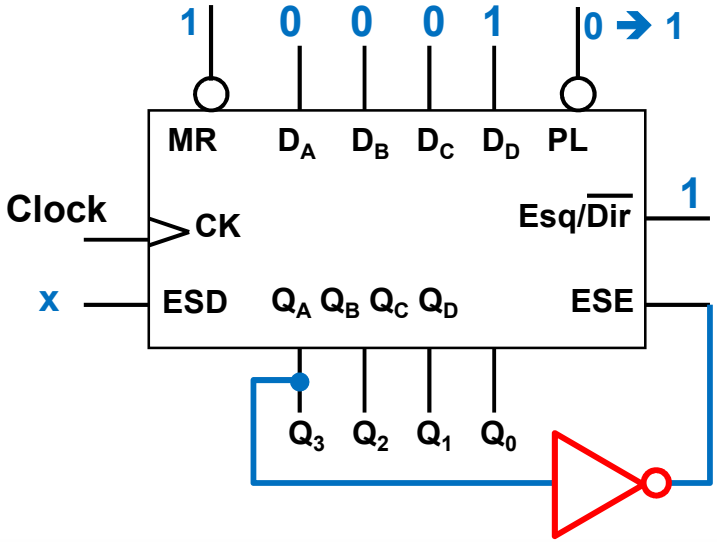


# Exercício

- Reprojetar deslocador em anel original, para dobrar o número de estados da contagem sem lógica adicinal




Original:



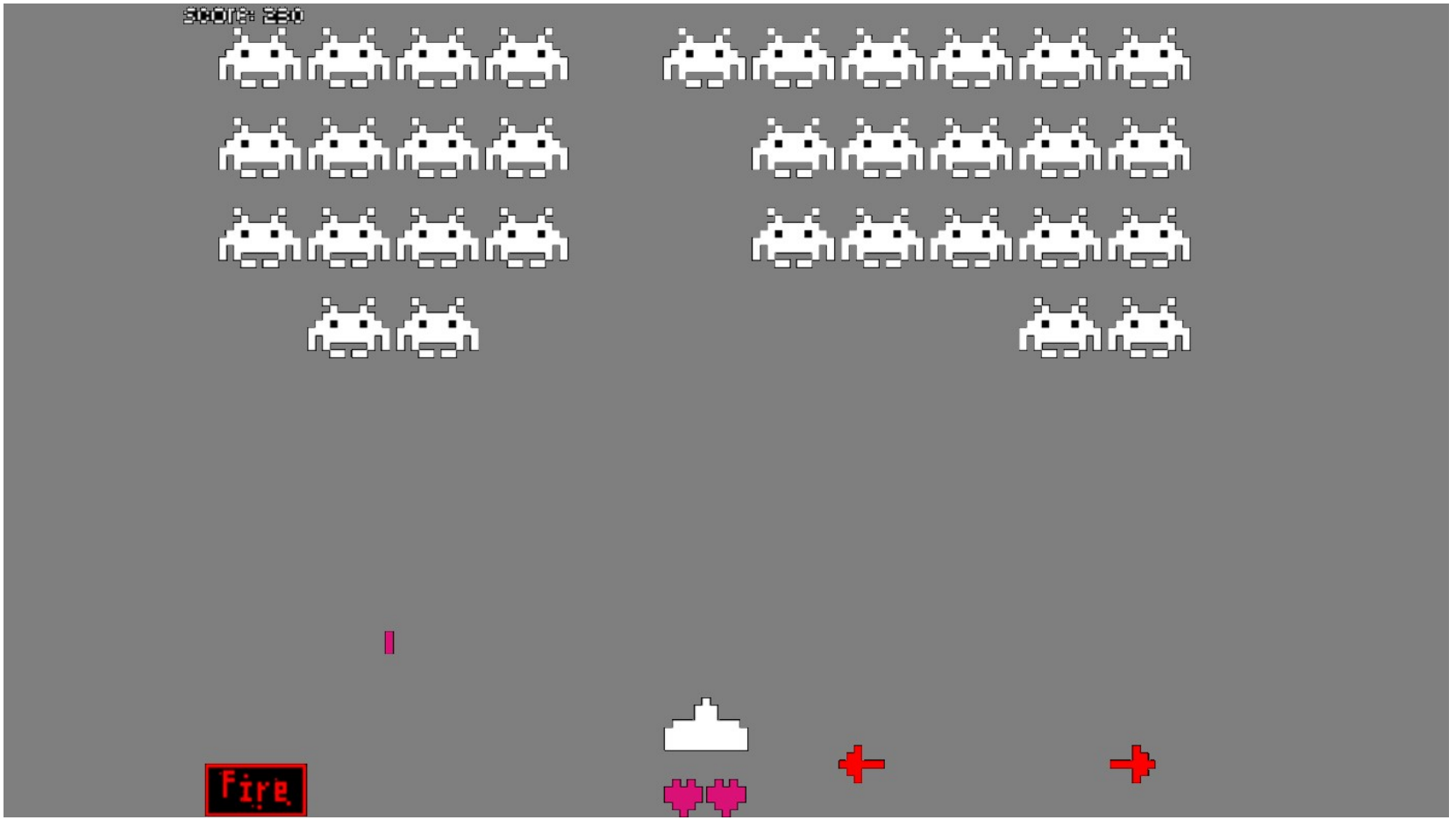

**"Deslocador em anel torcido"**

**ou**

**"Contador Johnson"**

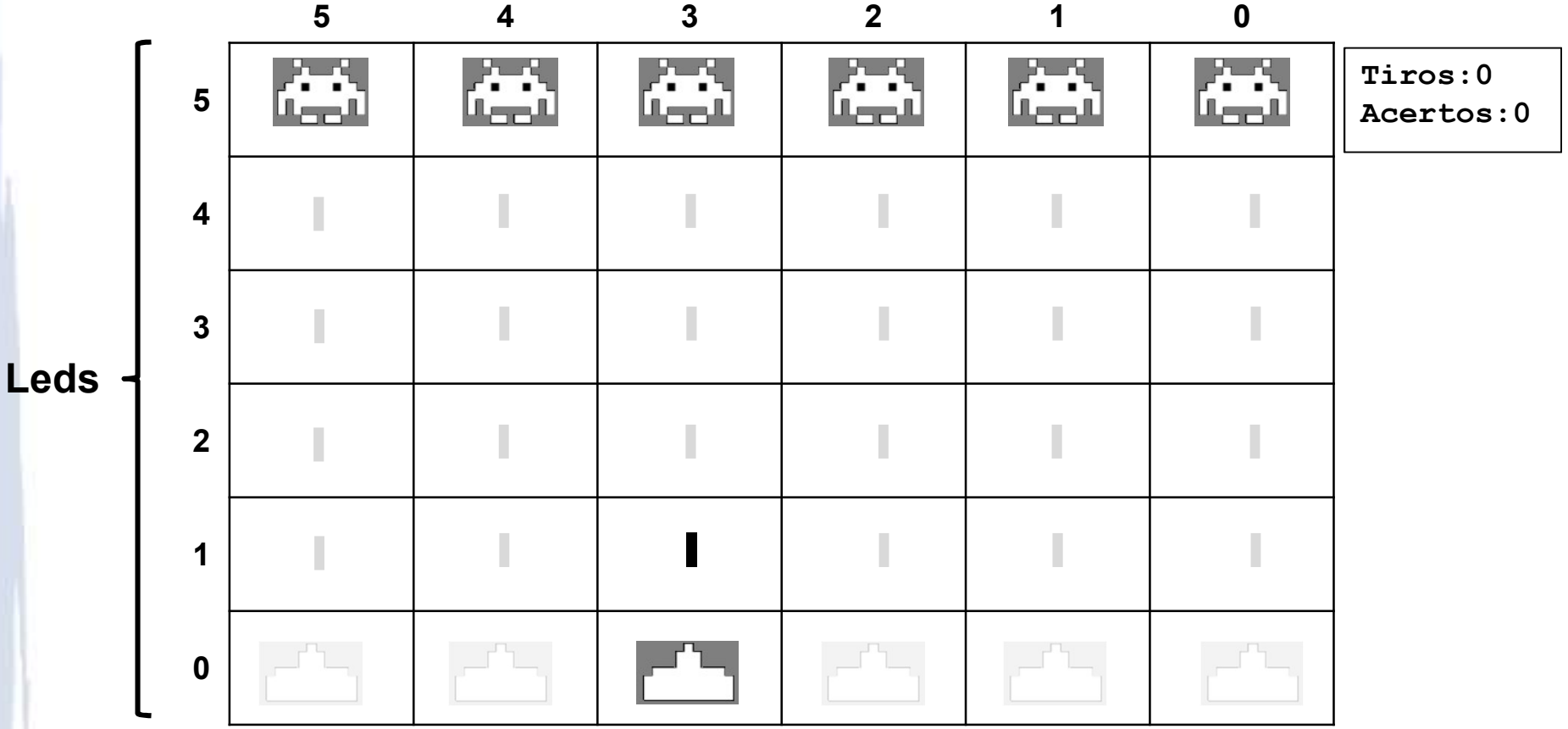
# Exercício 2

- Projeto básico de “Space Invaders” em hardware



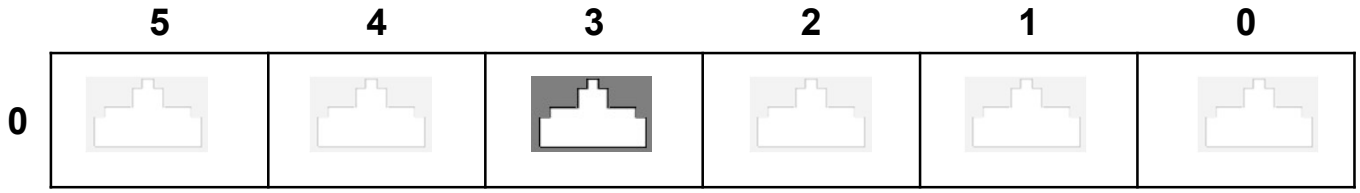
# Exercício 2

- Projeto básico de “Space Invaders” em hardware

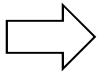


# Exercício 2

- Requisitos: **nave**
  - Início na posição 3, e pode se mover p/ direita ou esquerda
  - Movimento circular: coluna 5 “do lado” da coluna 0

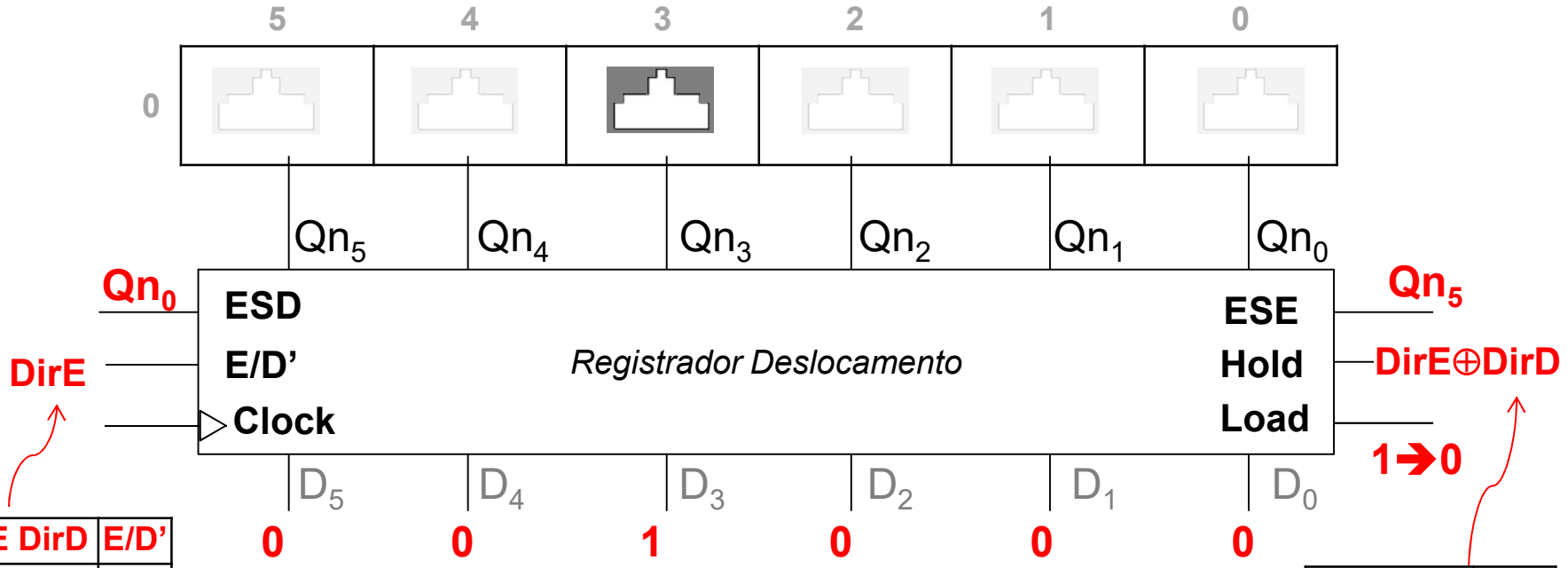



## Como implementar?

**Comandos:** **DirE**  **DirD**  

# Exercício 2

- Requisitos: **nave**
  - Início na posição 2, e pode se mover p/ direita ou esquerda
  - Movimento circular: coluna 5 “do lado” da coluna 0




| DirE | DirD | E/D’ |
|---|---|---|
| 0 | 0 | * |
| 0 | 1 | 0 |
| 1 | 1 | * |
| 1 | 0 | 1 |

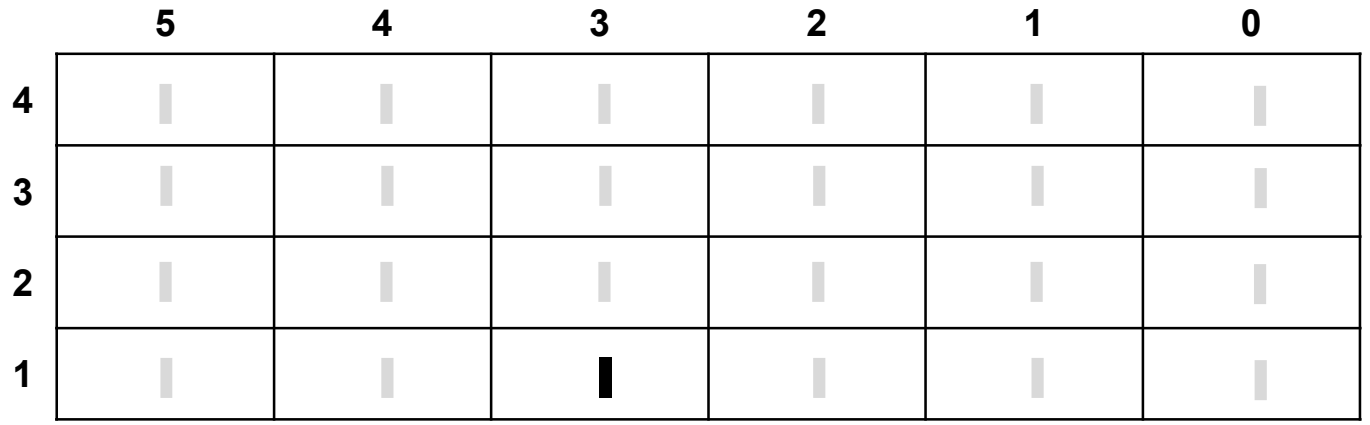
“Don’t care”: devido a Hold

| DirE | DirD | Hold |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |

**Comandos:** DirE DirD F

# Exercício 2

- Requisitos: **tiros da nave**
  - Gerados por F na linha 1, sobem até encontrar linha 4

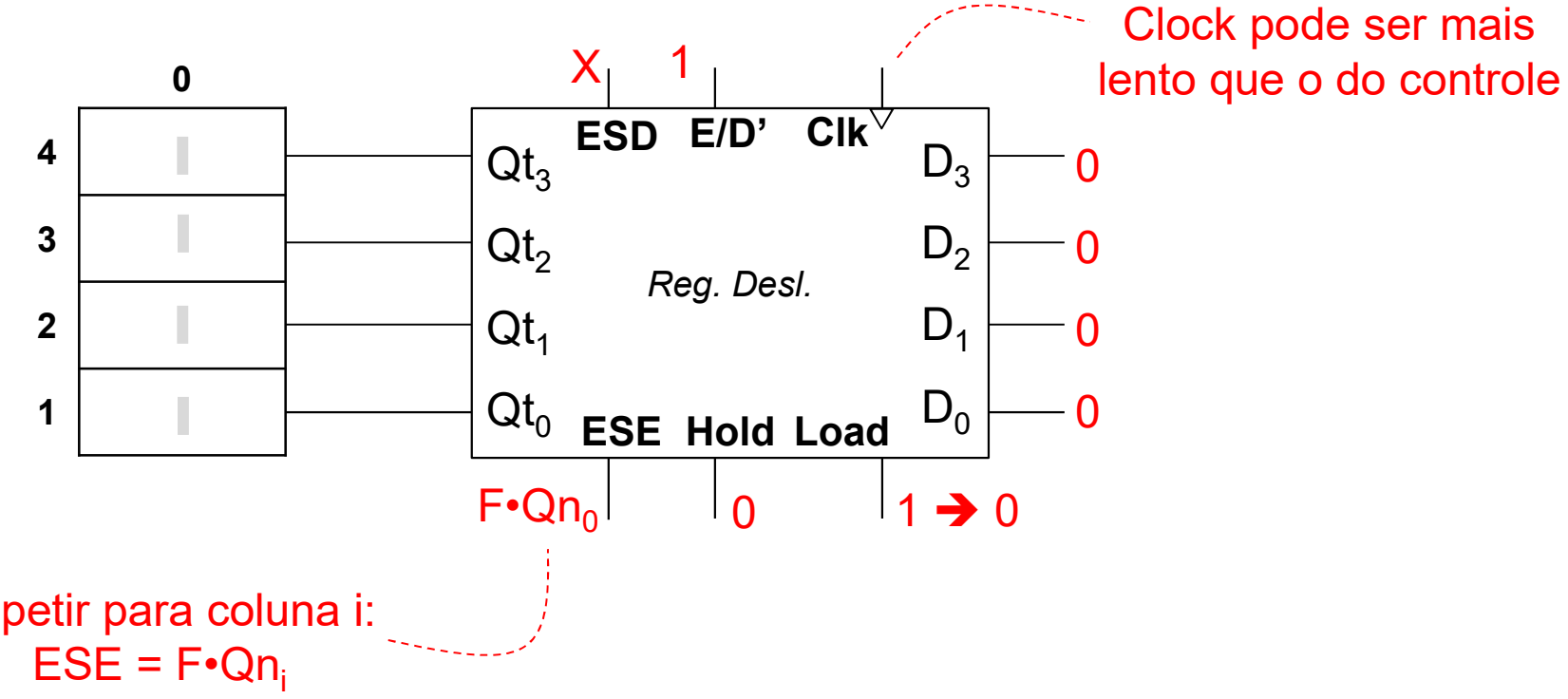



## Como implementar?

**Comandos:** DirE ⇦ DirD ⇨ F

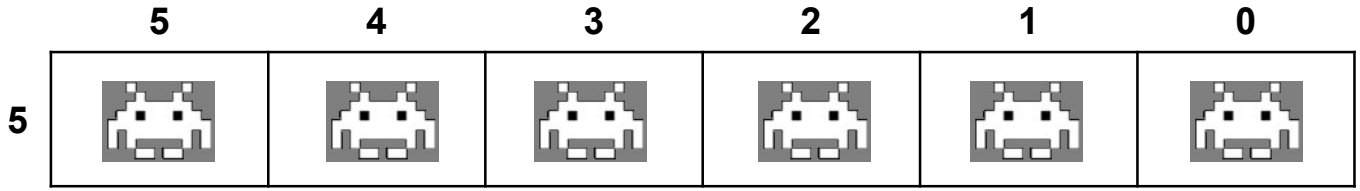
# Exercício 2

- Requisitos: **tiros da nave**
  - Gerados por F na linha 1, sobem até encontrar linha 1



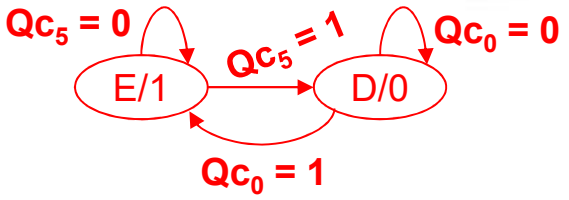
# Exercício 2

- Requisitos: **criaturas do espaço**
  - No início, todos presentes
  - Movem-se para um lado até grupo atingir borda, e então mudam de direção




## Como implementar?

# Exercício 2

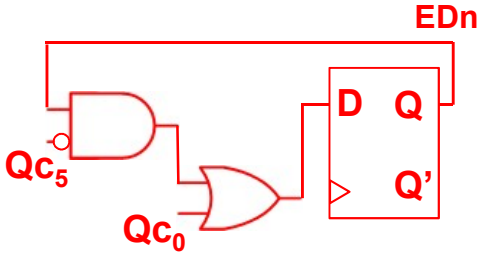



- Requisitos: **criaturas do espaço**
  - No início, todos presentes
  - Vão p/ um lado até atingir borda, e então mudam de direção

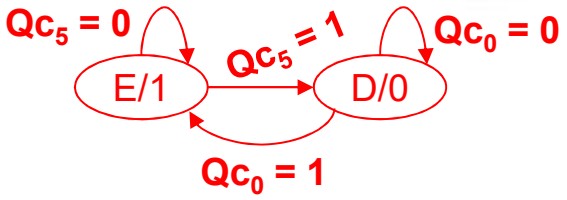

| $Qc_5$ $Qc_0$ | EDn |
|---|---|
| 0 0 | mantém |
| 0 1 | 1 |
| 1 1 | * |
| 1 0 | 0 |

| | $EDn^t$ | |
|---|---|---|
| $Qc_5$ $Qc_0$ | 0 | 1 |
| 0 0 | 0 | 1 |
| 0 1 | 1 | 1 |
| 1 1 | * | * |
| 1 0 | 0 | 0 |

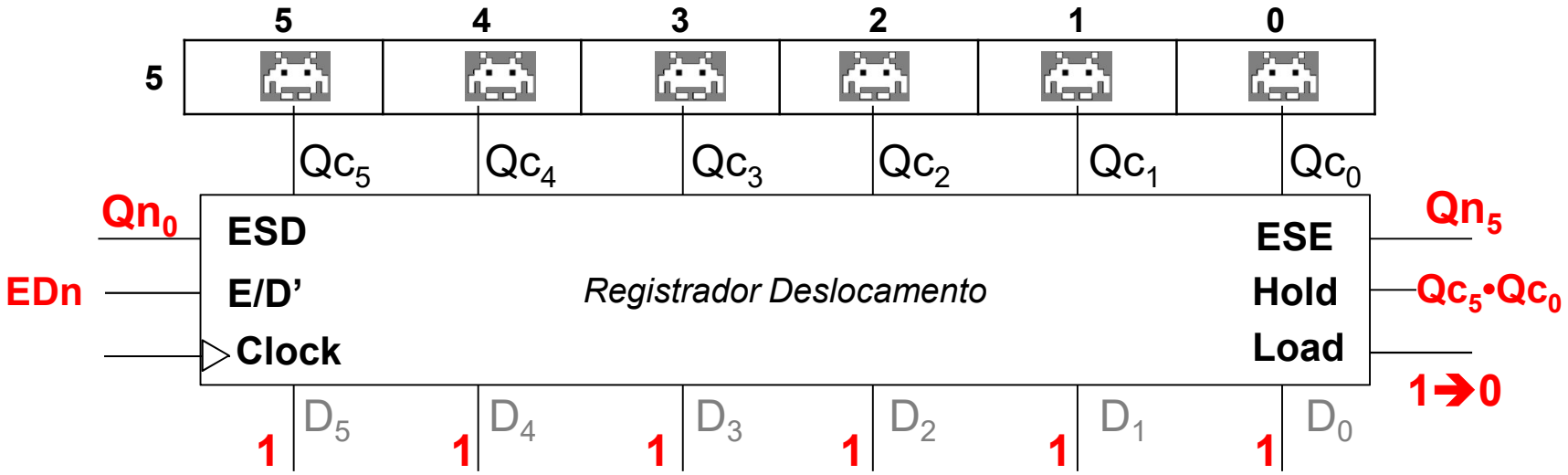
$EDn^{t+1} = Qc_0 + EDn^t \bullet Qc_5'$



# Exercício 2




- Requisitos: **criaturas do espaço**
  - No início, todos presentes, mas **desaparecem ao levar tiro**




## Como implementar?

# Exercício 2

- Requisitos: **criaturas do espaço**
  - No início, todos presentes, mas **desaparecem ao levar tiro**


**Live$_i$** = $Qc_i \bullet Qt_{3-i}'$

**Kill:** $Qc_5 \bullet Qt_{3-5} + Qc_4 \bullet Qt_{3-4} + \ldots + Qc_0 \bullet Qt_{3-0}$

Obs.: nesse projeto, deslocamento não ocorre em clock que invasor é abatido

# Exercício 2

- Requisitos: **pontuação**
  - Contagem de tiros
  - Contagem de acertos: jogo termina se 6 invasores forem abatidos

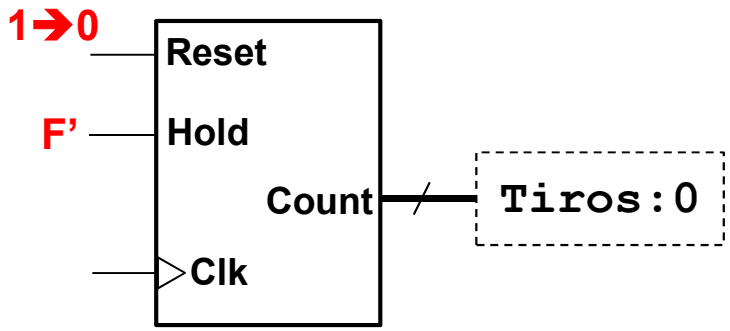
```
Tiros:0
Acertos:0
```

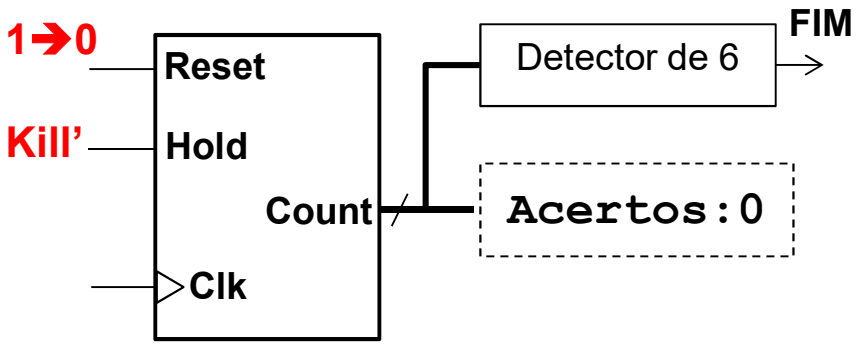
## Como implementar?

**Comandos:** **DirE** ⇦ **DirD** ⇨ (F)

# Exercício 2

- Requisitos: **pontuação**
  - Contagem de tiros
  - Contagem de acertos: jogo termina se 6 invasores forem abatidos




Obs.: projeto aproveita fato de não ser possível abater dois ou mais invasores no mesmo clock

# EXEMPLOS EM VHDL

Extraídos do livro-texto (Wakerly)

Ref.: seções 8.2.7, 8.4.6, 8.5.8.

# Registradores em VHDL

Registrador de 16 bits com *clear assíncrono*, *enable* e saída *tri-state*

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vreg16 is
  port (CLK, CLKEN, OE_L, CLR_L: in STD_LOGIC;
        D: in STD_LOGIC_VECTOR(1 to 16);       -- Input bus
        Q: out STD_ULOGIC_VECTOR (1 to 16) ); -- Output bus (three-state)
end Vreg16;

architecture Vreg16 of Vreg16 is
signal CLR, OE: STD_LOGIC;  -- active-high versions of signals
signal IQ: STD_LOGIC_VECTOR(1 to 16); -- internal Q signals
begin
process(CLK, CLR_L, CLR, OE_L, OE, IQ)
  begin
    CLR <= not CLR_L;  OE <= not OE_L;
    if (CLR = '1') then IQ <= (others => '0');
    elsif (CLK'event and CLK='1') then
      if (CLKEN='1') then IQ <= D; end if;
    end if;
    if OE = '1' then Q <= To_StdULogicVector(IQ);
    else Q <= (others => 'Z'); end if;
  end process;
end Vreg16;
```

Table 8-8

VHDL model of a 16-bit register with many features.

# Registradores em VHDL

- Tabela funcional de um deslocador universal

| Function | Inputs | | | Next state | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | S2 | S1 | S0 | Q7* | Q6* | Q5* | Q4* | Q3* | Q2* | Q1* | Q0* |
| Hold | 0 | 0 | 0 | Q7 | Q6 | Q5 | Q4 | Q3 | Q2 | Q1 | Q0 |
| Load | 0 | 0 | 1 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| Shift right | 0 | 1 | 0 | RIN | Q7 | Q6 | Q5 | Q4 | Q3 | Q2 | Q1 |
| Shift left | 0 | 1 | 1 | Q6 | Q5 | Q4 | Q3 | Q2 | Q1 | Q0 | LIN |
| Shift circular right | 1 | 0 | 0 | Q0 | Q7 | Q6 | Q5 | Q4 | Q3 | Q2 | Q1 |
| Shift circular left | 1 | 0 | 1 | Q6 | Q5 | Q4 | Q3 | Q2 | Q1 | Q0 | Q7 |
| Shift arithmetic right | 1 | 1 | 0 | Q7 | Q7 | Q6 | Q5 | Q4 | Q3 | Q2 | Q1 |
| Shift arithmetic left | 1 | 1 | 1 | Q6 | Q5 | Q4 | Q3 | Q2 | Q1 | Q0 | 0 |

Table 8-36

Function table for an extended-function 8-bit shift register.



73

# Registradores em VHDL

- Deslocador universal

```
entity Vshftreg is
    port (
        CLK, CLR, RIN, LIN: in STD_LOGIC;
        S: in STD_LOGIC_VECTOR (2 downto 0); -- function select
        D: in STD_LOGIC_VECTOR (7 downto 0); -- data in
        Q: out STD_LOGIC_VECTOR (7 downto 0) -- data out
    );
end Vshftreg;

architecture Vshftreg_arch of Vshftreg is
signal IQ: STD_LOGIC_VECTOR (7 downto 0);
begin
process (CLK, CLR, IQ)
  begin
    if (CLR='1') then IQ <= (others=>'0'); -- Asynchronous clear
    elsif (CLK'event and CLK='1') then
      case CONV_INTEGER(S) is
        when 0 => null;                          -- Hold
        when 1 => IQ <= D;                       -- Load
        when 2 => IQ <= RIN & IQ(7 downto 1);    -- Shift right
        when 3 => IQ <= IQ(6 downto 0) & LIN;    -- Shift left
        when 4 => IQ <= IQ(0) & IQ(7 downto 1); -- Shift circular right
        when 5 => IQ <= IQ(6 downto 0) & IQ(7); -- Shift circular left
        when 6 => IQ <= IQ(7) & IQ(7 downto 1); -- Shift arithmetic right
        when 7 => IQ <= IQ(6 downto 0) & '0';    -- Shift arithmetic left
        when others => null;
      end case;
    end if;
    Q <= IQ;
  end process;
end Vshftreg_arch;
```

Table 8-37

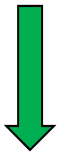
VHDL code for an extended-function 8-bit shift register.

# Contadores em VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity counter3bits is
   port (  Clock, Reset : in std_logic;
           Count: out std_logic_vector (2 downto 0);
           RCO: out std_logic);
end entity counter3bits;

architecture arch of counter3bits is
   signal counter : unsigned (2 downto 0); -- sinal interno
begin
   p0: process (Clock, Reset) is
   begin
        if (Reset = '1') then
            counter <= "000"; -- valor inicial
        elsif rising_edge(Clock) then
            counter <= counter + 1;
            if (counter = "111") then RCO <= '1'; else RCO <= '0';
            end if;
        end if;
        Count <= counter; -- sinal interno carregado na saída
   end process p0;
end architecture arch;
```

# Contadores em VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity counter3bits is
   port (  Clock, Reset : in std_logic;
           Count: out std_logic_vector (2 downto 0);
           RCO: out std_logic);
end entity counter3bits;

architecture arch of counter3bits is
begin
   p0: process (Clock, Reset) is
        variable counter : unsigned (2 downto 0); -- variável
   begin
        if (Reset = '1') then
            counter := "000"; -- valor inicial
        elsif rising_edge(Clock) then
            counter := counter + 1;
            if (counter = "111") then RCO <= '1'; else RCO <= '0';
            end if;
        end if;
        Count <= std_logic_vector(counter); -- "cast" de variable
   end process p0;
end architecture arch;
```



Usando variável

# Contadores em VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity counter26 is   -- contagem alternativa: 2 a 6
   port (  Clock, Reset : in std_logic;
           Count : out std_logic_vector (2 downto 0));
end entity counter26;

architecture arch of counter26 is
begin
   p0: process (Clock, Reset) is
        variable counter : unsigned (2 downto 0);
   begin
    if (Reset = '1') then
           counter := "010"; -- valor inicial
        elsif rising_edge(Clock) then
           if (counter = "110") then
              counter := "010"; -- após "6" volta a "2" no clock
           else
              counter := counter + 1;
           end if;
        end if;
        Count <= std_logic_vector(counter); -- "cast" de variable
   end process p0;
end architecture arch;
```

# Contadores em VHDL

- CI 74x163

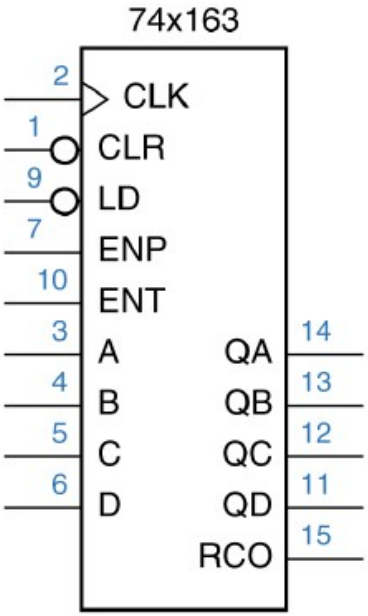



Figure 8-27

Traditional logic symbol for the 74×163.

| Inputs | | | | Current State | | | | Next State | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CLR_L | LD_L | ENT | ENP | QD | QC | QB | QA | QD* | QC* | QB* | QA* |
| 0 | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 |
| 1 | 0 | x | x | x | x | x | x | D | C | B | A |
| 1 | 1 | 0 | x | x | x | x | x | QD | QC | QB | QA |
| 1 | 1 | x | 0 | x | x | x | x | QD | QC | QB | QA |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

Table 8-13

State table for a 74×163 4-bit binary counter.

# Contadores em VHDL

- CI 74x163

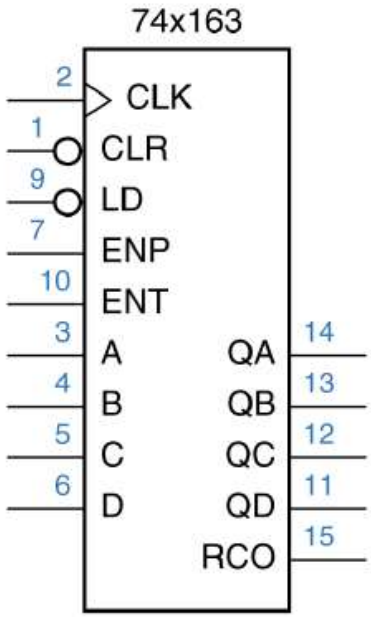



Figure 8-27

Traditional logic symbol for the 74×163.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity V74x163 is
    port ( CLK, CLR_L, LD_L, ENP, ENT: in STD_LOGIC;
           D: in UNSIGNED (3 downto 0);
           Q: out UNSIGNED (3 downto 0);
           RCO: out STD_LOGIC );
end V74x163;

architecture V74x163_arch of V74x163 is
signal IQ: UNSIGNED (3 downto 0);
begin
process (CLK, ENT, IQ)
  begin
    if (CLK'event and CLK='1') then
      if CLR_L='0' then IQ <= (others => '0');
      elsif LD_L='0' then IQ <= D;
      elsif (ENT and ENP)='1' then IQ <= IQ + 1;
      end if;
    end if;
    if (IQ=15) and (ENT='1') then RCO <= '1';
    else RCO <= '0';
    end if;
    Q <= IQ;
  end process;
end V74x163_arch;
```

Table 8-16

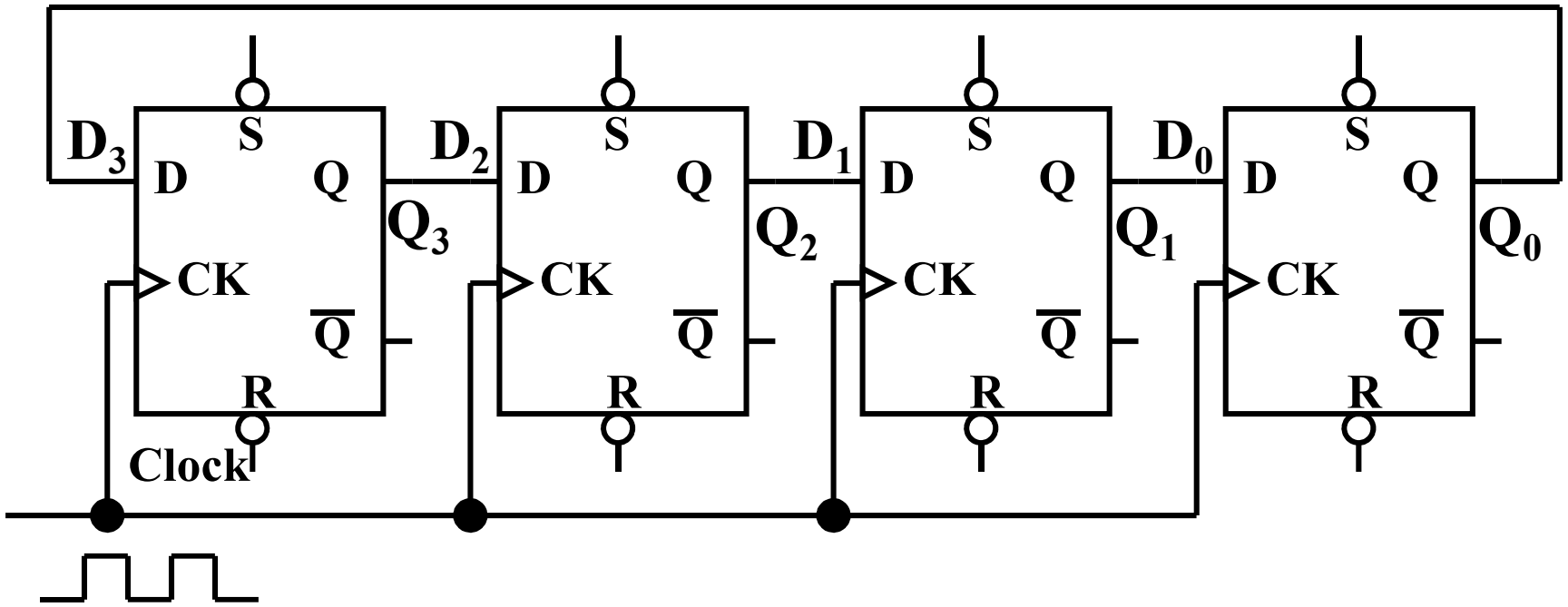
VHDL program for a 74×163-like 4-bit binary counter.

# APÊNDICE

Registradores em anel

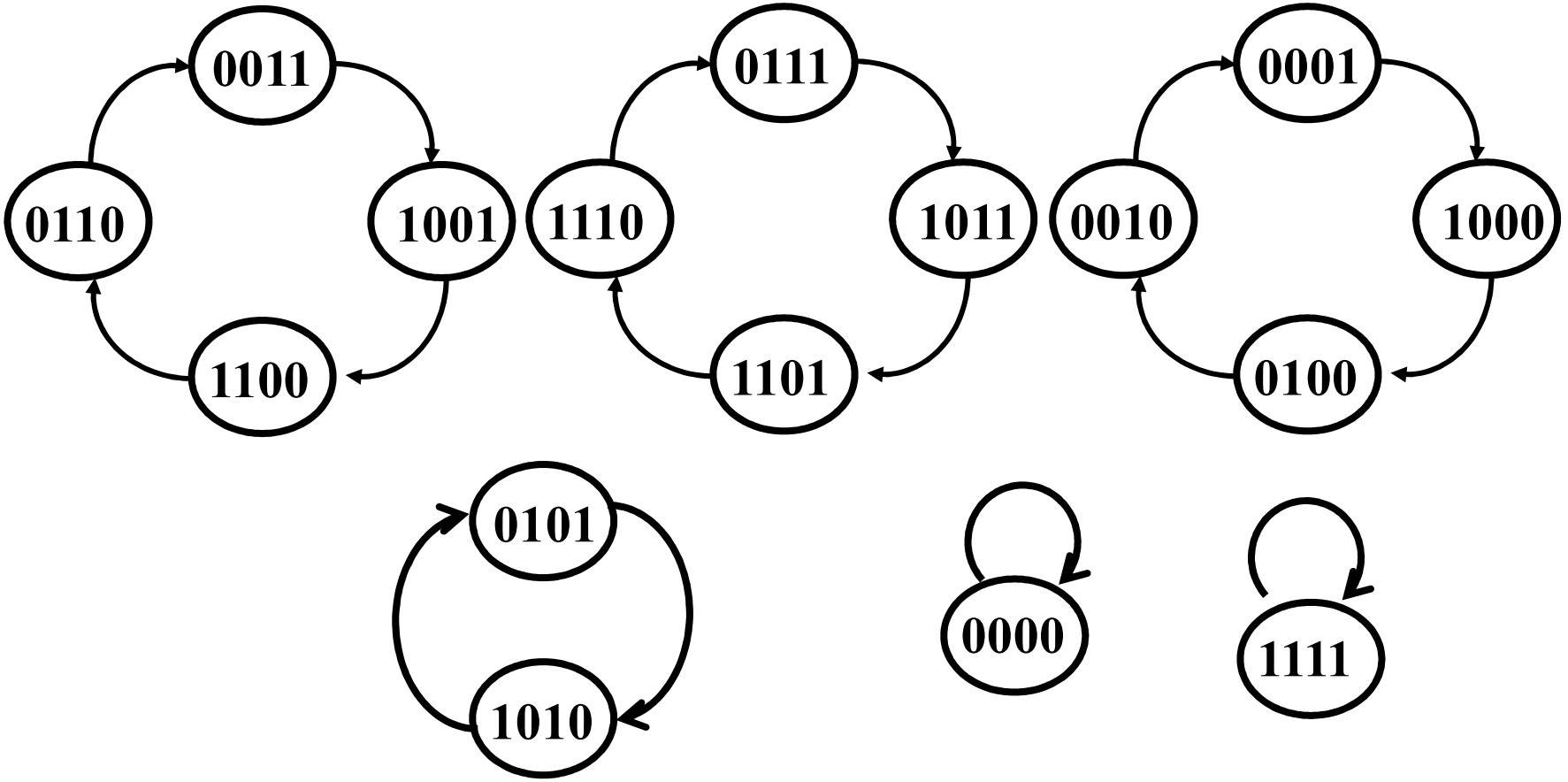
# Deslocador em Anel

- **Deslocador em Anel:** Obtém-se ao realimentar a saída $Q_0$ para a entrada $D_{n-1}$, para deslocamento para a direita.



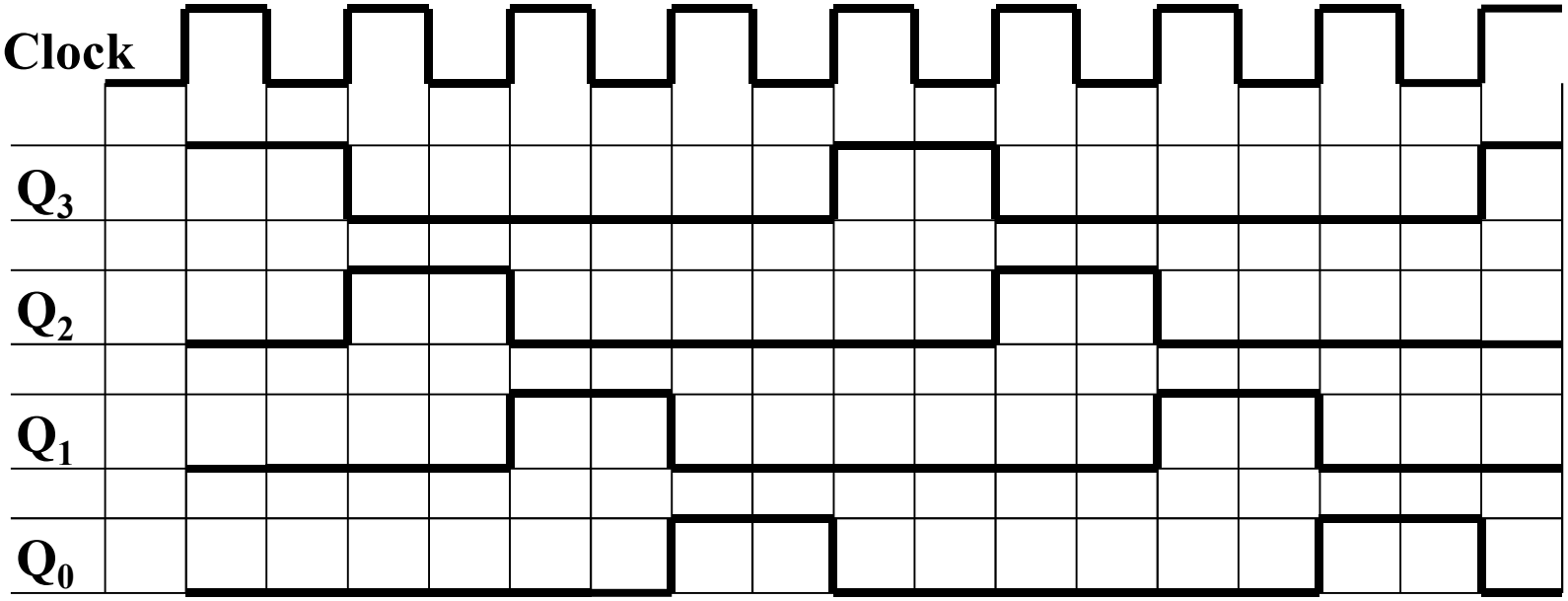
# Deslocador em Anel

- **Deslocador em Anel:** Sequências de transição de estados (desloc. direita):



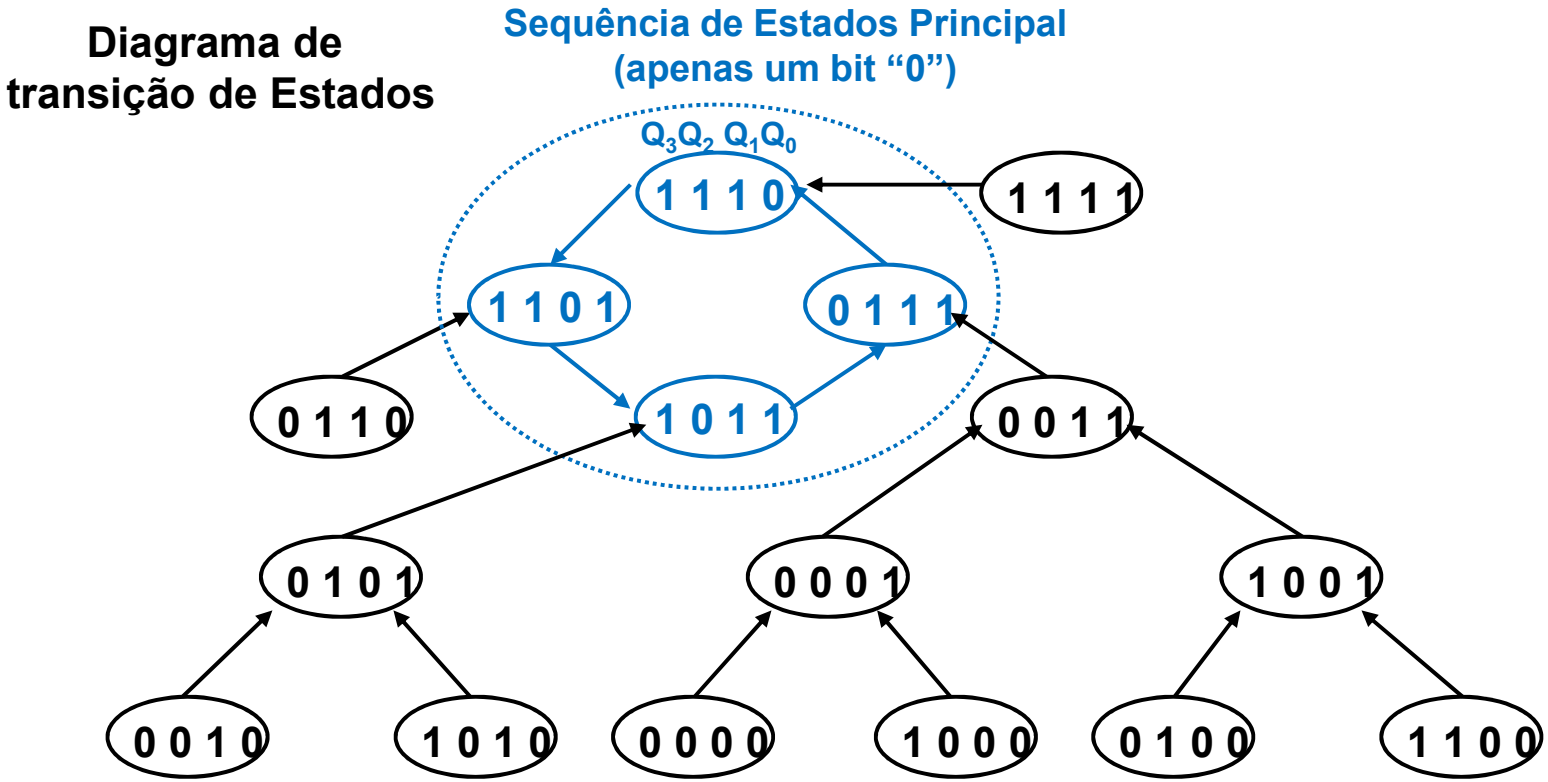
# Deslocador em Anel

- **Deslocador em Anel** – Diagrama de Temporização para a sequência que começa no Estado 1000 ($Q_3Q_2Q_1Q_0$):



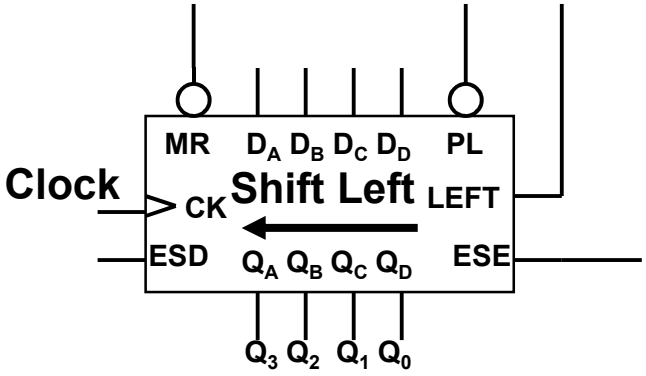
# Deslocador em Anel

- **Deslocador em Anel:** Circuito de auto-correção (apenas 1 bit ZERO)
  - Exemplo – deslocamento à esquerda



# Deslocador em Anel

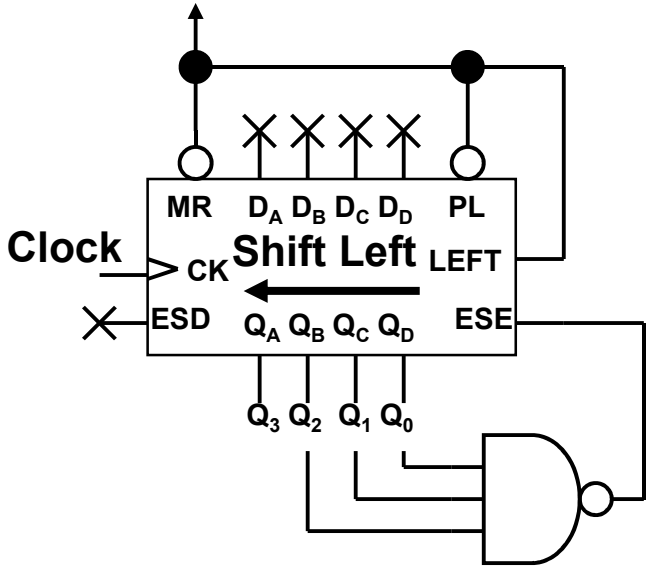
- **Deslocador em Anel:** Circuito de auto-correção (apenas 1 bit ZERO)
  - Exemplo – deslocamento à esquerda




PL: carga paralela
MR: reset mestre

# Deslocador em Anel

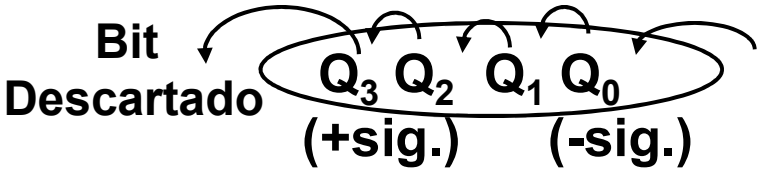
- **Deslocador em Anel:** Circuito de auto-correção (apenas 1 bit ZERO)
  - Exemplo – deslocamento à esquerda




PL: carga paralela
MR: reset mestre

**Tabela de transição de Estados**

$Q_A\ Q_B\ Q_C\ Q_D$




$Q_3^{t+1} = Q_2^t$ ou $Q_A^{t+1} = Q_B^t$

$Q_2^{t+1} = Q_1^t$ ou $Q_B^{t+1} = Q_C^t$

$Q_1^{t+1} = Q_0^t$ ou $Q_C^{t+1} = Q_D^t$

$Q_0^{t+1} = ESE$ ou $Q_D^{t+1} = ESE$

# Deslocador em Anel

- **Deslocador em Anel:** Circuito de auto-correção (apenas 1 bit UM)
  - Exemplo – deslocamento à esquerda

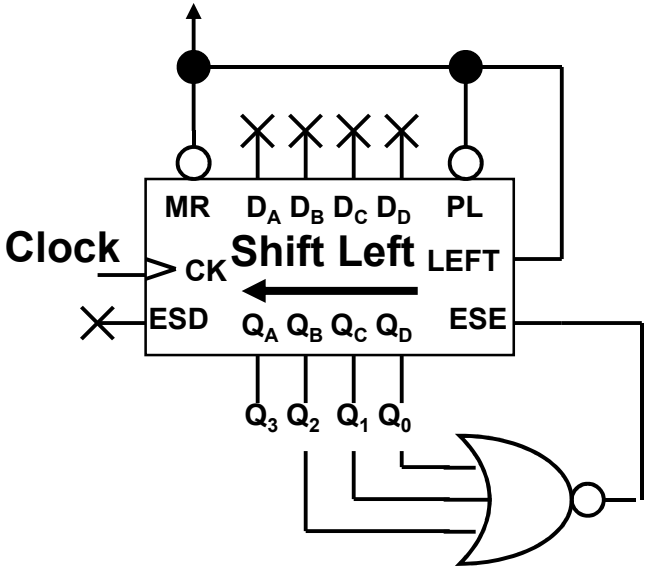
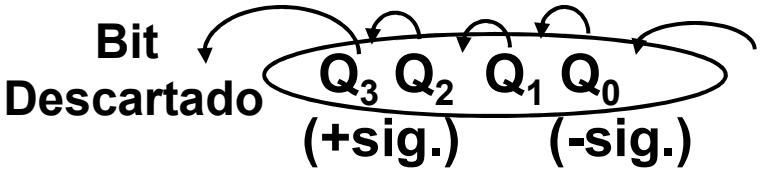



**Tabela de transição de Estados**

$Q_A\ Q_B\ Q_C\ Q_D$




$Q_3^{t+1} = Q_2^t$ ou $Q_A^{t+1} = Q_B^t$

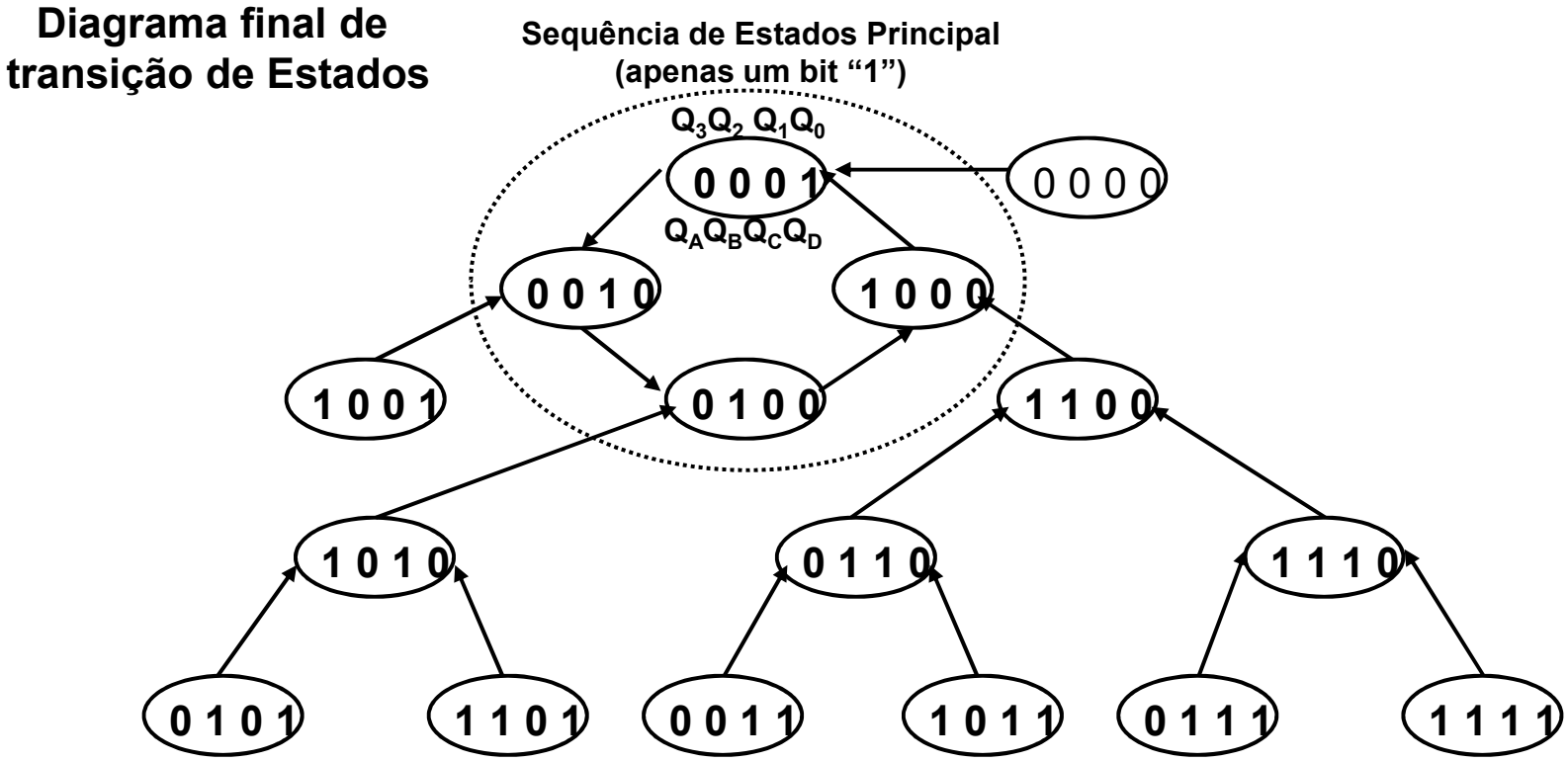
$Q_2^{t+1} = Q_1^t$ ou $Q_B^{t+1} = Q_C^t$

$Q_1^{t+1} = Q_0^t$ ou $Q_C^{t+1} = Q_D^t$

$Q_0^{t+1} = ESE$ ou $Q_D^{t+1} = ESE$

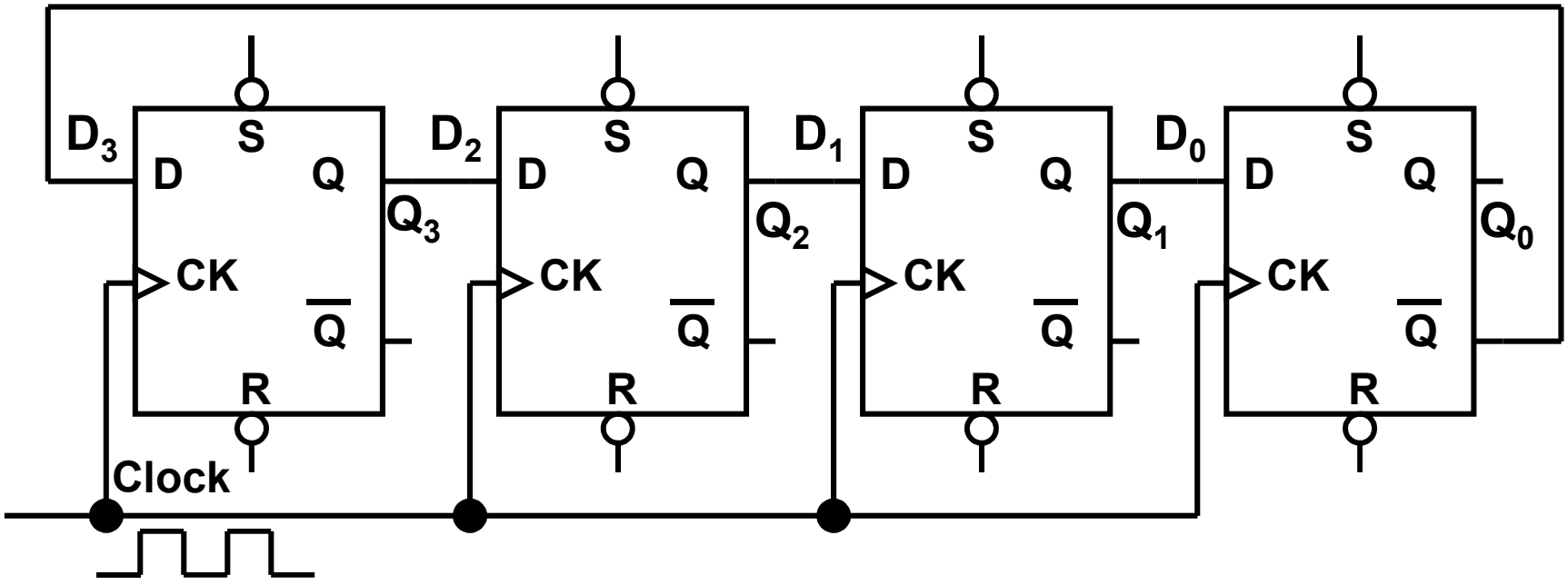
# Deslocador em Anel

- **Deslocador em Anel:** Circuito de auto-correção (apenas 1 bit UM)
  - Exemplo – deslocamento à esquerda



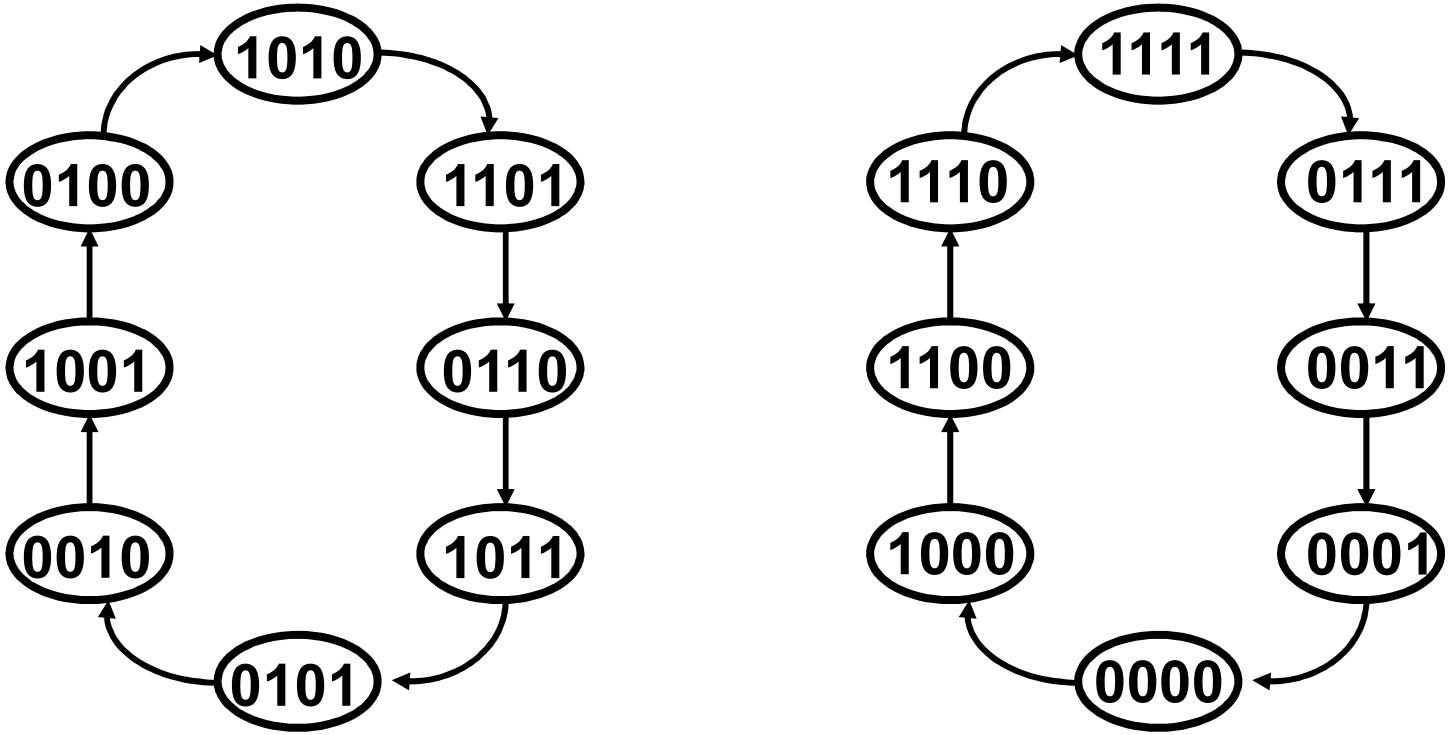
# Deslocador em Anel Torcido

- **Deslocador em Anel Torcido:** saída $\mathbf{Q_0}$ é **complementada e realimentada** p/ entrada $\mathbf{D_{n-1}}$
  - 2n estados para n Flip-Flops (“Contador Johnson”)



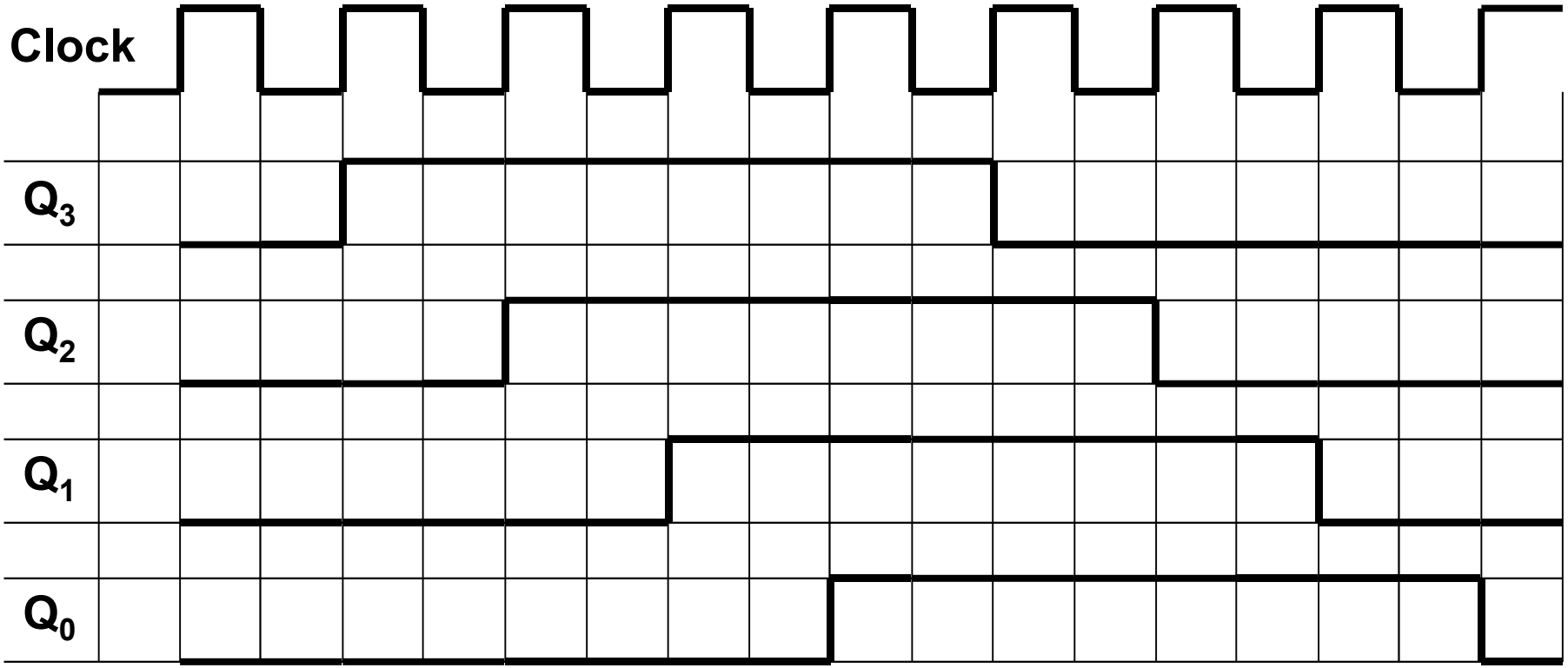
# Deslocador em Anel Torcido

- **Deslocador em Anel Torcido:** Duas Sequências de transição de 8 Estados:



# Deslocador em Anel Torcido

- **Deslocador em Anel Torcido** – Diagrama de temporização para a sequência que começa no Estado 0000 ($Q_3Q_2Q_1Q_0$):

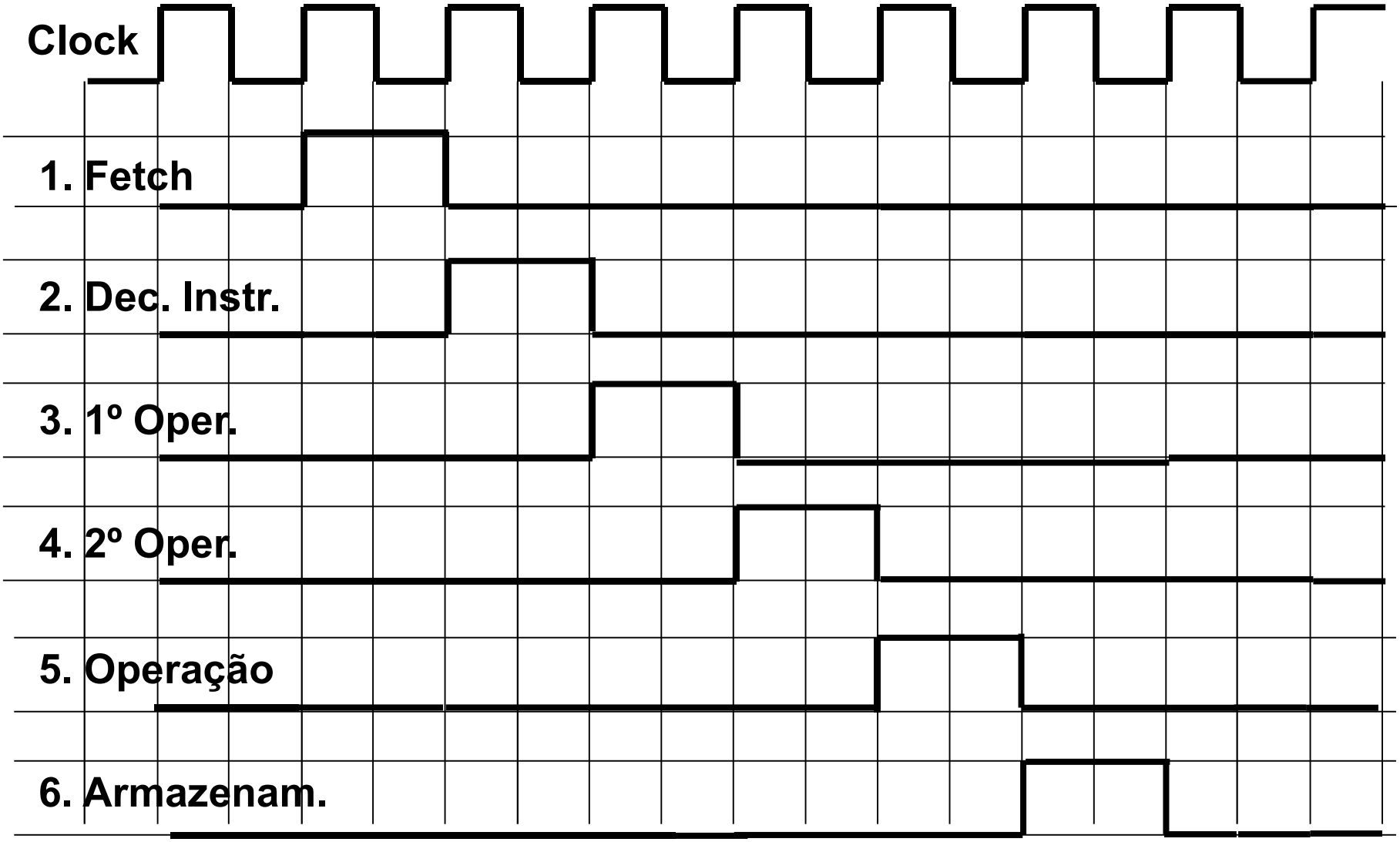


# Deslocador em Anel Torcido

- **Deslocador em Anel Torcido** – Exemplo de **geração de sinais de controle** para **execução de instrução** aritmética em um **microprocessador** – Seis etapas:

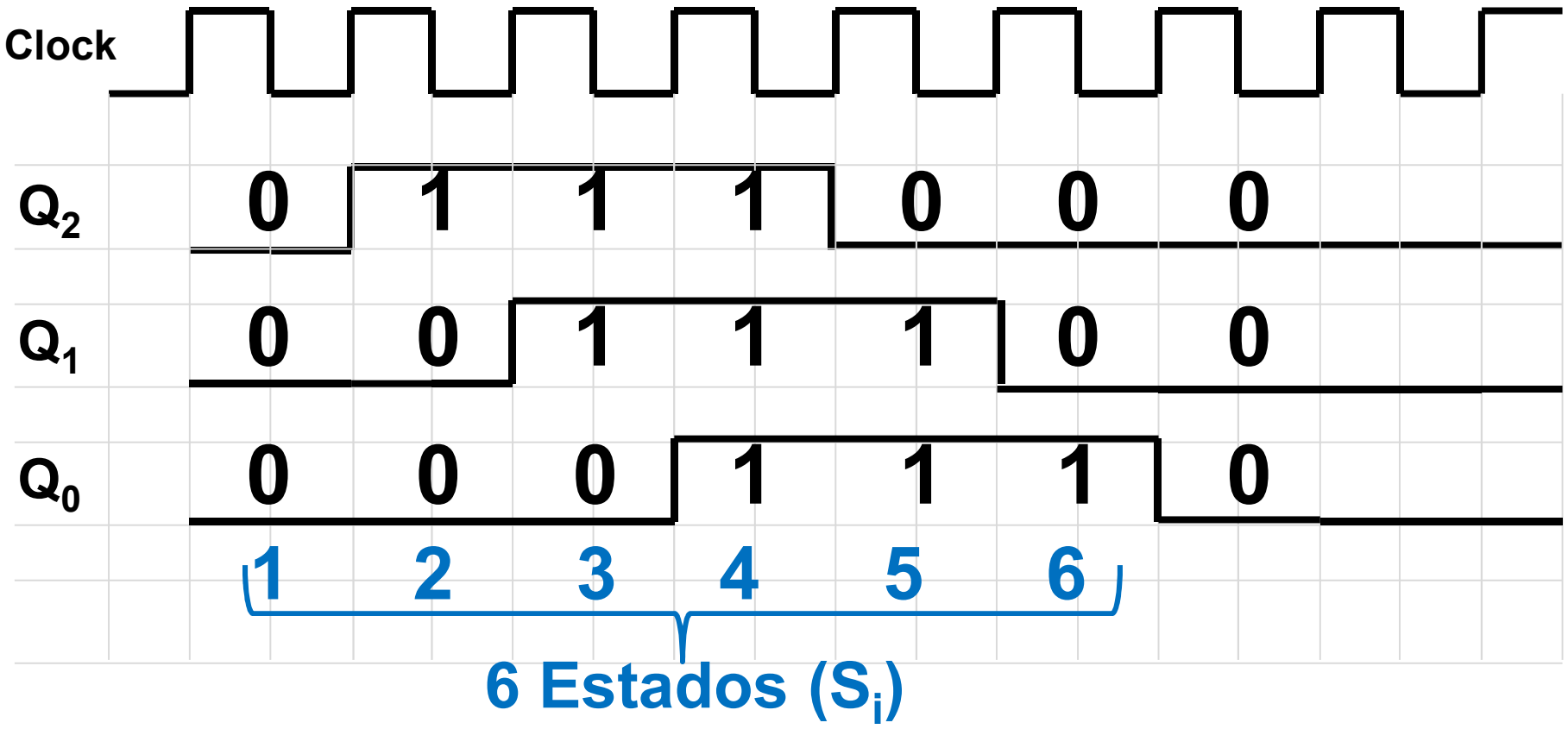
1. *Fetch* (busca da Instrução);
2. Decodificação da Instrução;
3. Obtenção do Primeiro Operando;
4. Obtenção do Segundo Operando;
5. Realização da Operação;
6. Armazenamento do Resultado.

# Deslocador em Anel Torcido

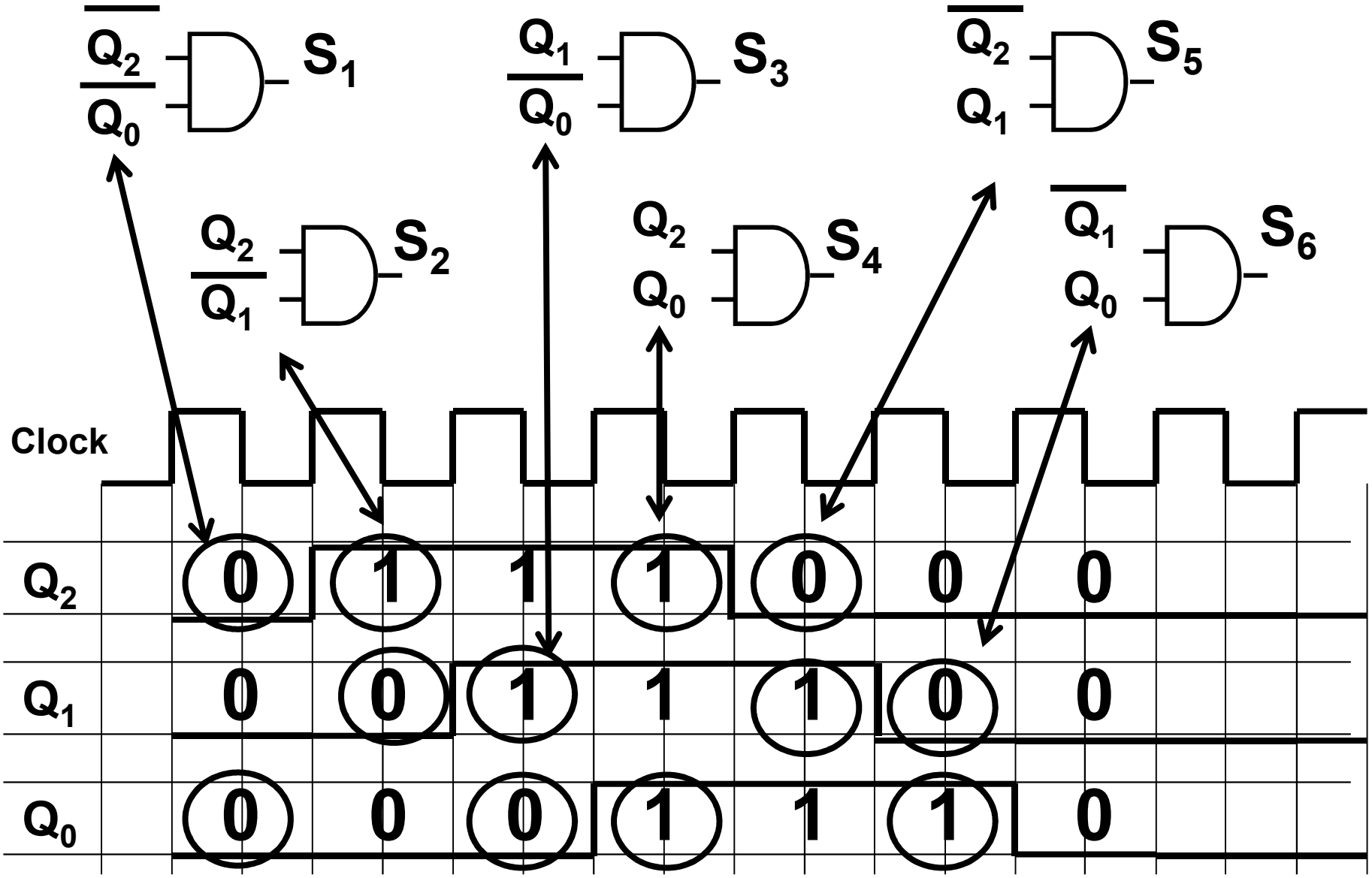


# Deslocador em Anel Torcido

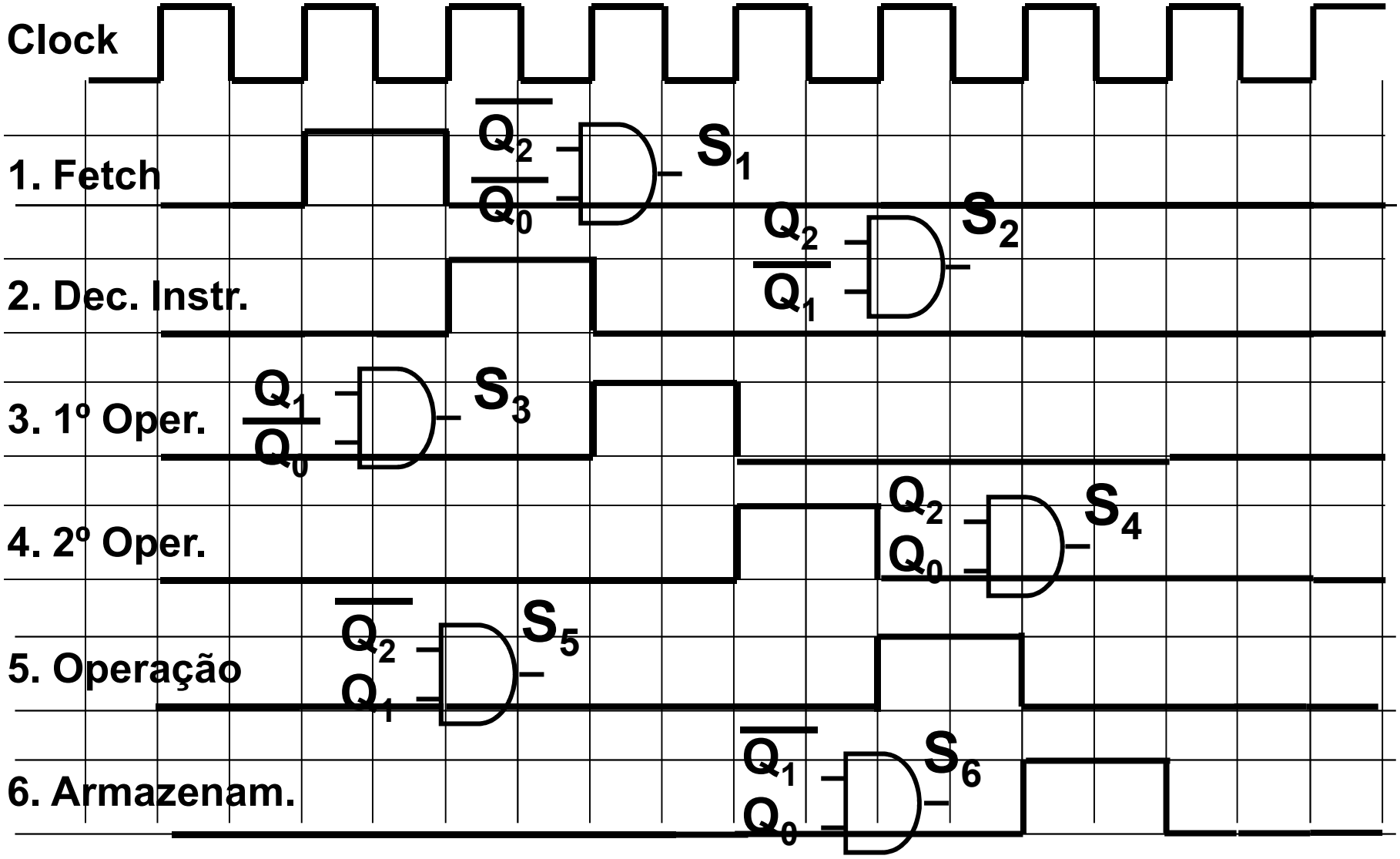
- **Deslocador Anel Torcido:** Utilizando 3 Flip-Flop's ($Q_2Q_1Q_0$) mais decodificação:



# Deslocador em Anel Torcido

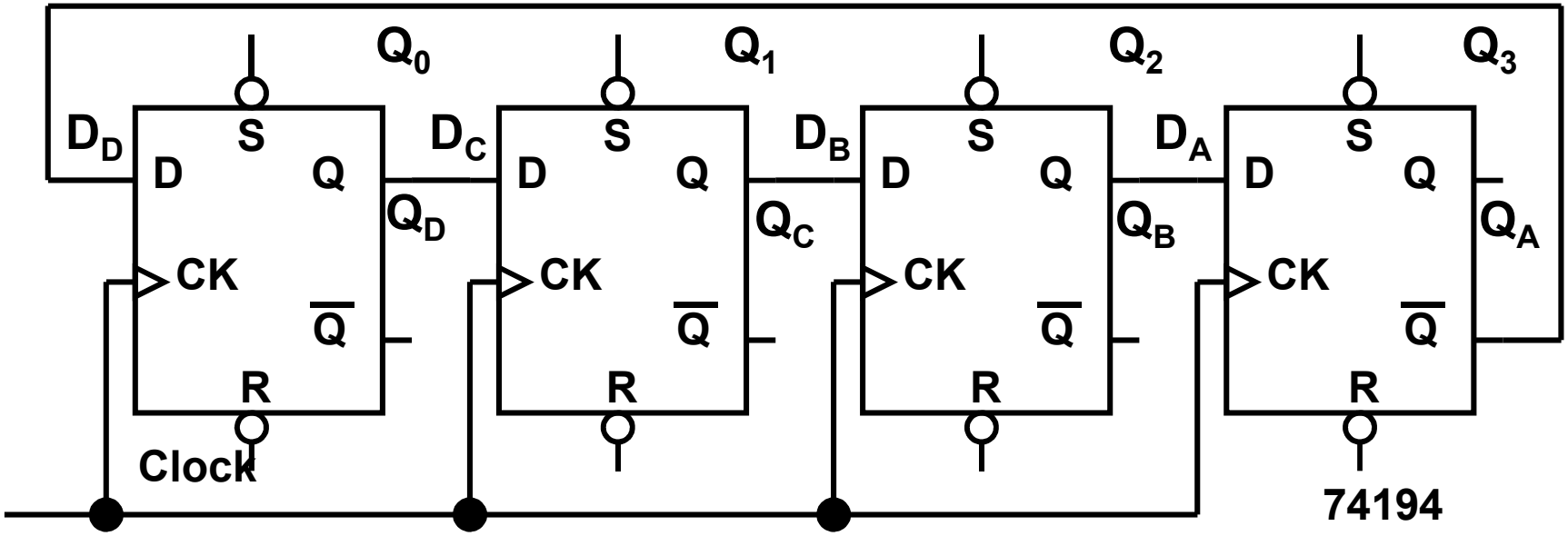


# Deslocador em Anel Torcido



# Deslocador em Anel Torcido

- **Deslocador em Anel Torcido:** permite simplificar lógica de decodificação
  - 8 estados, decodificados com portas NAND2
  - Com contador normal, seriam 3 FF + portas NAND3



# Deslocador em Anel Torcido

- **Deslocador em Anel Torcido:** permite simplificar a Lógica de Decodificação:

| $Q_0$ | $Q_1$ | $Q_2$ | $Q_3$ | Decodificação |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | $Q_0 \cdot \overline{Q_3}$ |
| 1 | 1 | 0 | 1 | $Q_0 \cdot Q_1$ |
| 0 | 1 | 1 | 0 | $Q_1 \cdot Q_2$ |
| 1 | 0 | 1 | 1 | $Q_2 \cdot Q_3$ |
| 0 | 1 | 0 | 1 | $\overline{Q_0} \cdot Q_3$ |
| 0 | 0 | 1 | 0 | $\overline{Q_0} \cdot \overline{Q_1}$ |
| 1 | 0 | 0 | 1 | $\overline{Q_1} \cdot \overline{Q_2}$ |
| 0 | 1 | 0 | 0 | $\overline{Q_2} \cdot \overline{Q_3}$ |

# Deslocador em Anel Torcido

- Se **n** é igual ao **número** de **Flip-Flop's**:

| **Contador em Anel** | **Contador em Anel Torcido** | **Contador Binário** |
|---|---|---|
| **n Estados** | **< 2n Estados <** | **$2^n$ Estados** |

# *Linear Feedback Shift Register* – LFSR

- *Linear Feedback Shift Register* (LFSR) **registrador com realimentação linear**
  - Realimenta uma função linear das saídas para a entrada em série.
- Teoria de Corpos [Evariste Galois – 1832]:
  - Se função de realimentação é uma função booleana (usa apenas operadores XOR) escolhida de maneira apropriada, o contador LFSR resultante tem uma sequência principal de $2^n - 1$ estados
    - » n = número de Flip-Flop's
- Exemplos de aplicação: códigos de transmissão e criptografia

# *Linear Feedback Shift Register* – LFSR

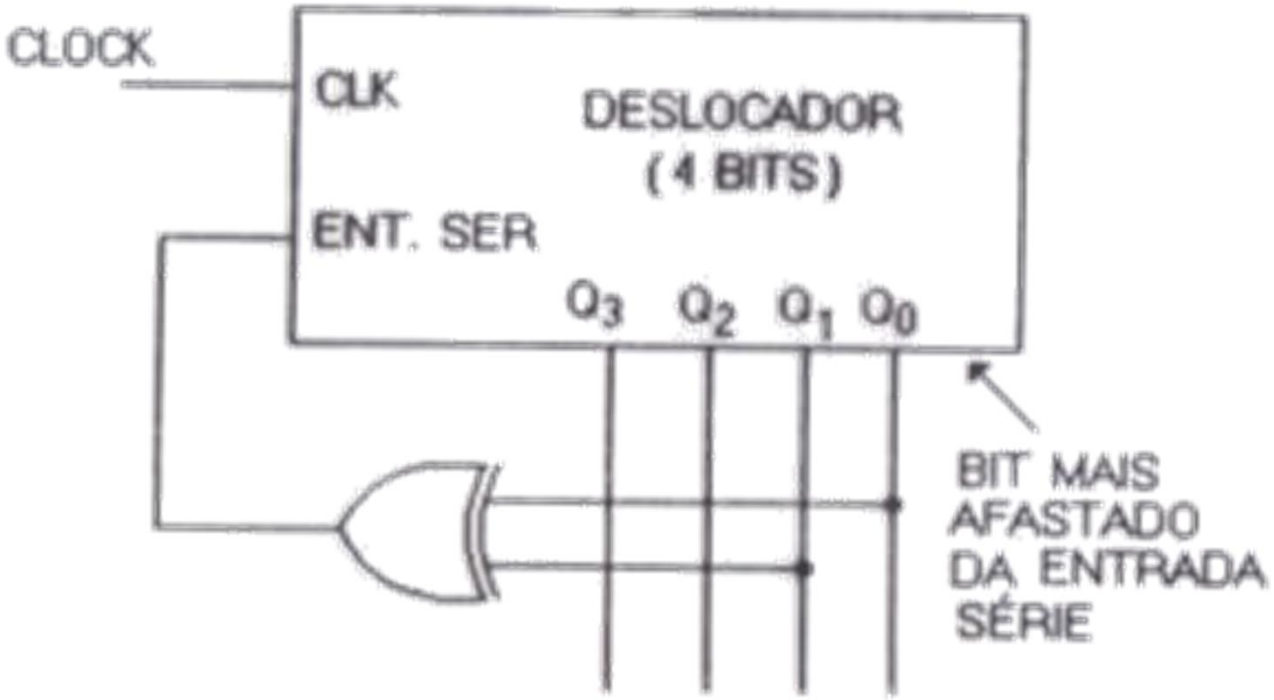
- *Linear Feedback Shift Register* (LFSR): exemplos de funções lineares que passam por $2^n - 1$ Estados (onde n = número de Flip-Flop's e $\oplus$ = XOR)

| n | Função |
|---|---|
| 2 | $Q_1 \oplus Q_0$ |
| 3 | $Q_1 \oplus Q_0$ |
| 4 | $Q_1 \oplus Q_0$ |
| 5 | $Q_2 \oplus Q_0$ |

| n | Função |
|---|---|
| 6 | $Q_1 \oplus Q_0$ |
| 7 | $Q_3 \oplus Q_0$ |
| 8 | $Q_4 \oplus Q_3 \oplus Q_2 \oplus Q_0$ |
| 12 | $Q_6 \oplus Q_4 \oplus Q_1 \oplus Q_0$ |

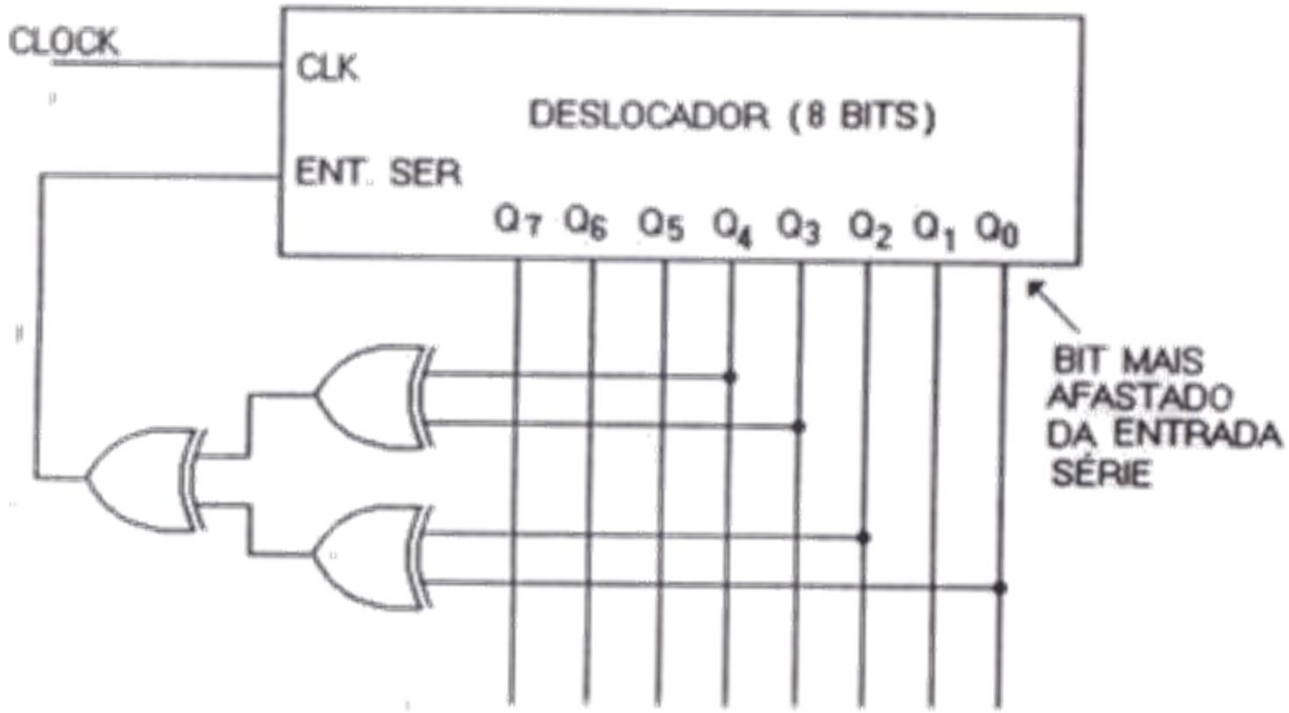
# *Linear Feedback Shift Register* – LFSR

- *Linear Feedback Shift Register* (LFSR) – Exemplos de circuitos que implementam funções lineares (com $2^n - 1$ estados na sequência):



# *Linear Feedback Shift Register* – LFSR

- *Linear Feedback Shift Register* (LFSR): Exemplos de circuitos que implementam funções lineares (com $2^n$ – 1 Estados na sequência):

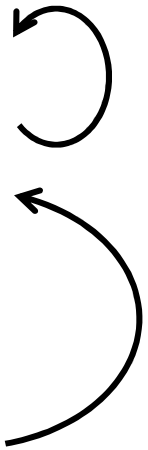


# *Linear Feedback Shift Register* – LFSR

- *Linear Feedback Shift Register* (LFSR): exemplos de funções lineares com $2^n - 1$ estados:

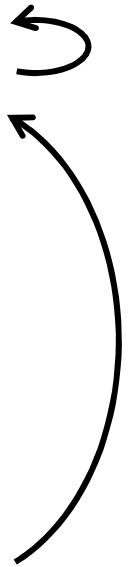
**n = 2; Função $Q_1 \oplus Q_0$**

| | $Q_1$ | $Q_0$ |
|---|---|---|
| | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |



**n = 3; Função $Q_1 \oplus Q_0$**

| | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|
| | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 |
| 4 | 1 | 0 | 1 |
| 5 | 1 | 1 | 0 |
| 6 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 |



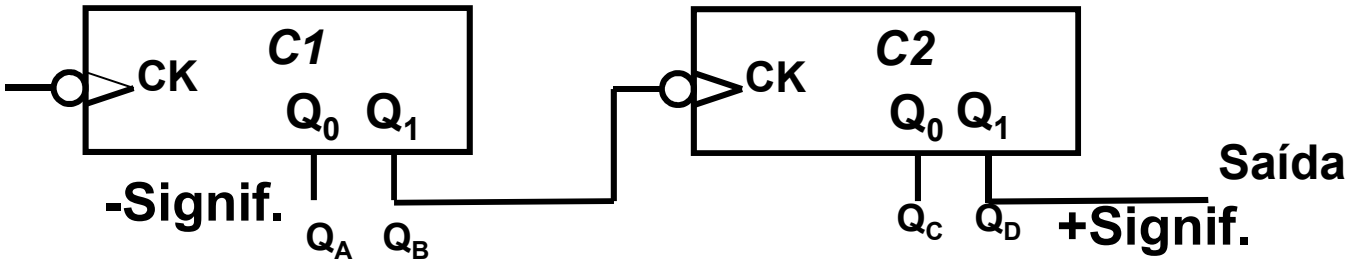
# APÊNDICE

Aplicações diversas de contadores

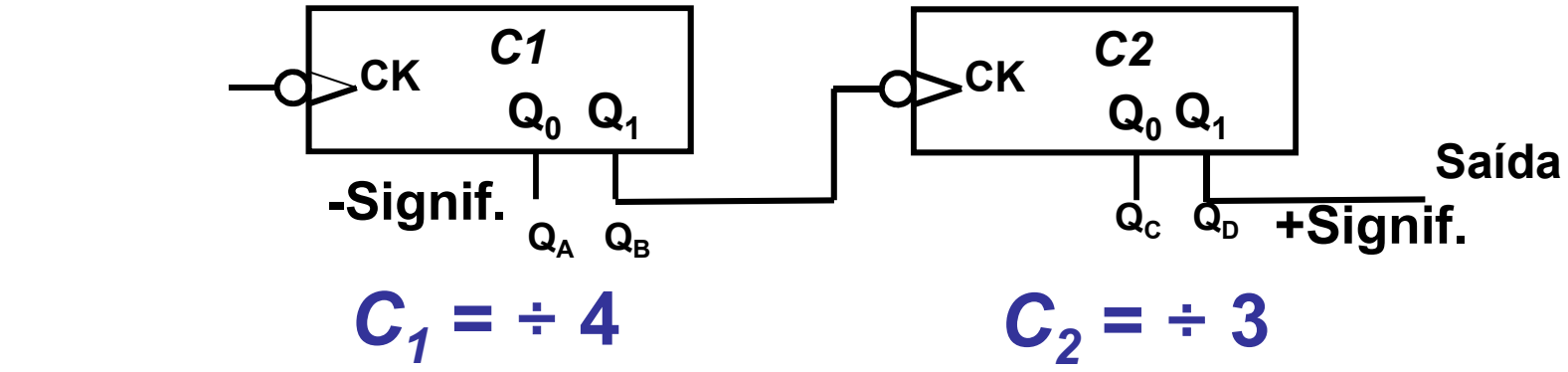
# Associação Assíncrona de Contadores

- **Objetivo:** divisão por $C_1 * C_2$
  - Cada contador registra parte da palavra de dados
  - Contadores podem ser síncronos e/ou assíncronos.
- Funcionamento:
  - **Associação é assíncrona**
  - **Bit mais significativo da parte menos significativa: gera clock** para parte mais significativa

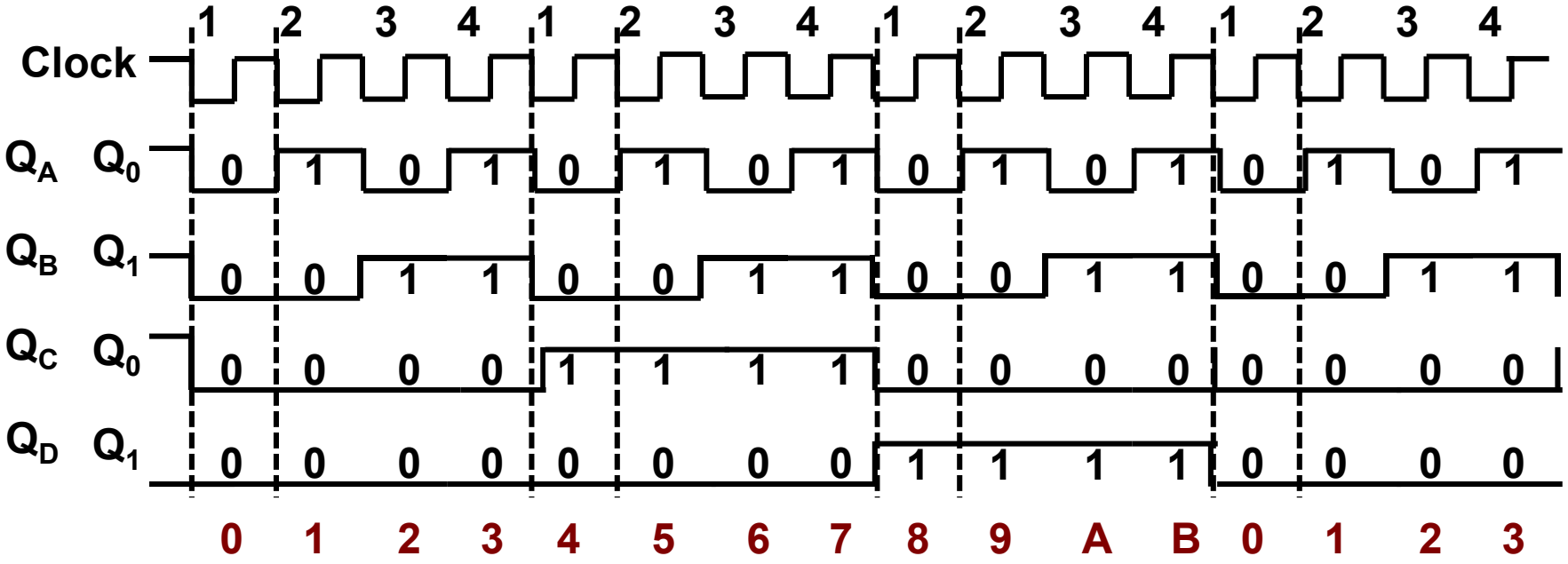
**Exemplo**



# Associação Assíncrona de Contadores




$C_1 = \div 4$ $C_2 = \div 3$



# Associação Assíncrona de Contadores

**$C_2$ Módulo $M_2 \neq 2^n$**

**$C_2 = \div 3$ (+Significativo)**

**Saída: Onda Retangular, Módulo 12 ($f_{CK} \div 12$)**

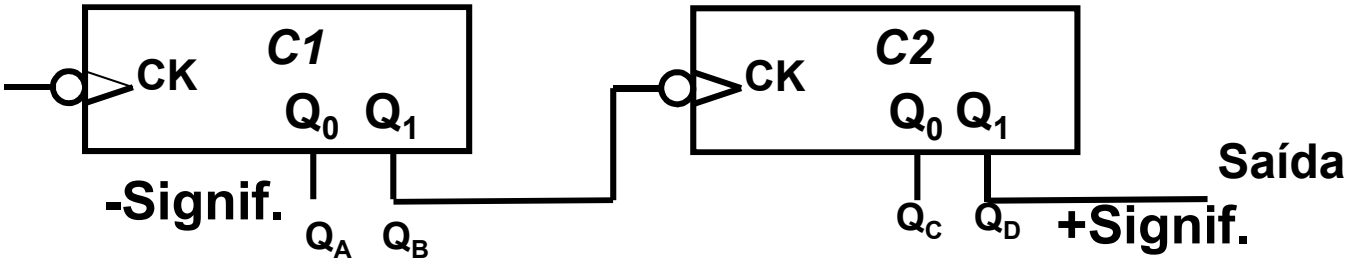
| $Q_D$ | $Q_C$ | $Q_B$ | $Q_A$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |

**$C_1$ Módulo $M_1 = 2^n$**

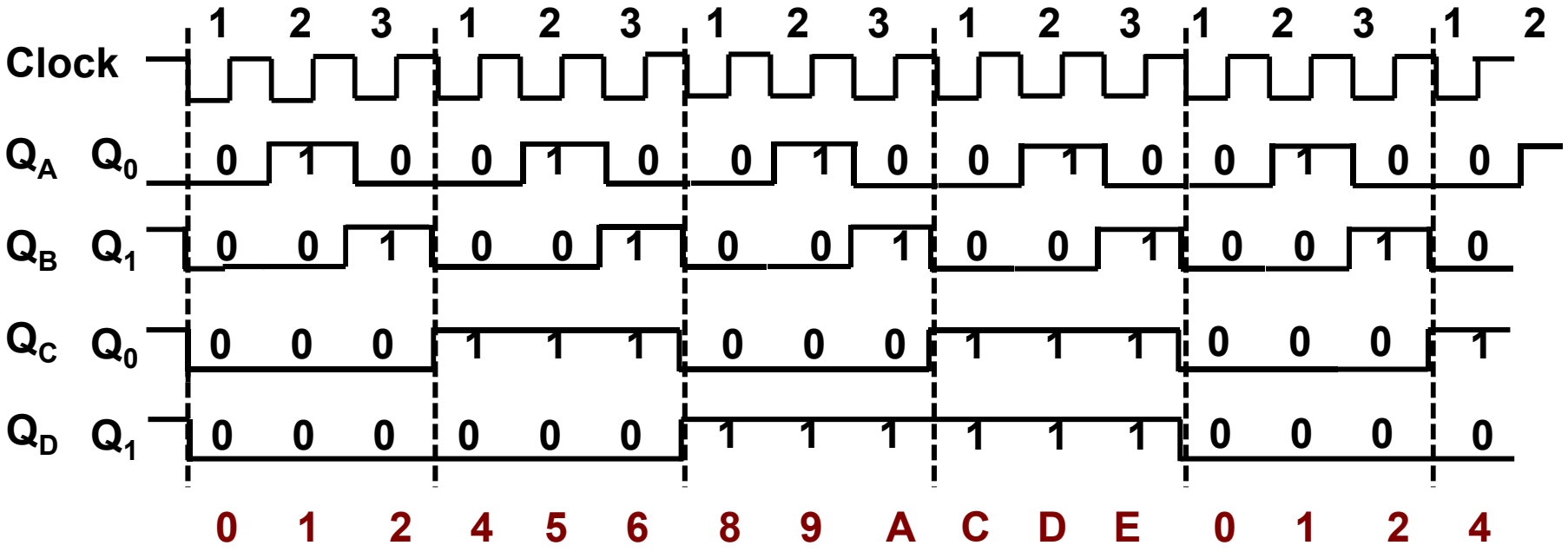
**$C_1 = \div 4$ (-Significativo)**

**Sequência Binária**

# Associação Assíncrona de Contadores




$C_1 = \div 3$ $C_2 = \div 4$



# Associação Assíncrona de Contadores

**$C_2$ Módulo $M_2 = 2^n$**

**$C_2 = \div 4$ (+Significativo)**

**Saída: Onda Quadrada, Módulo 12 ($f_{CK} \div 12$)**

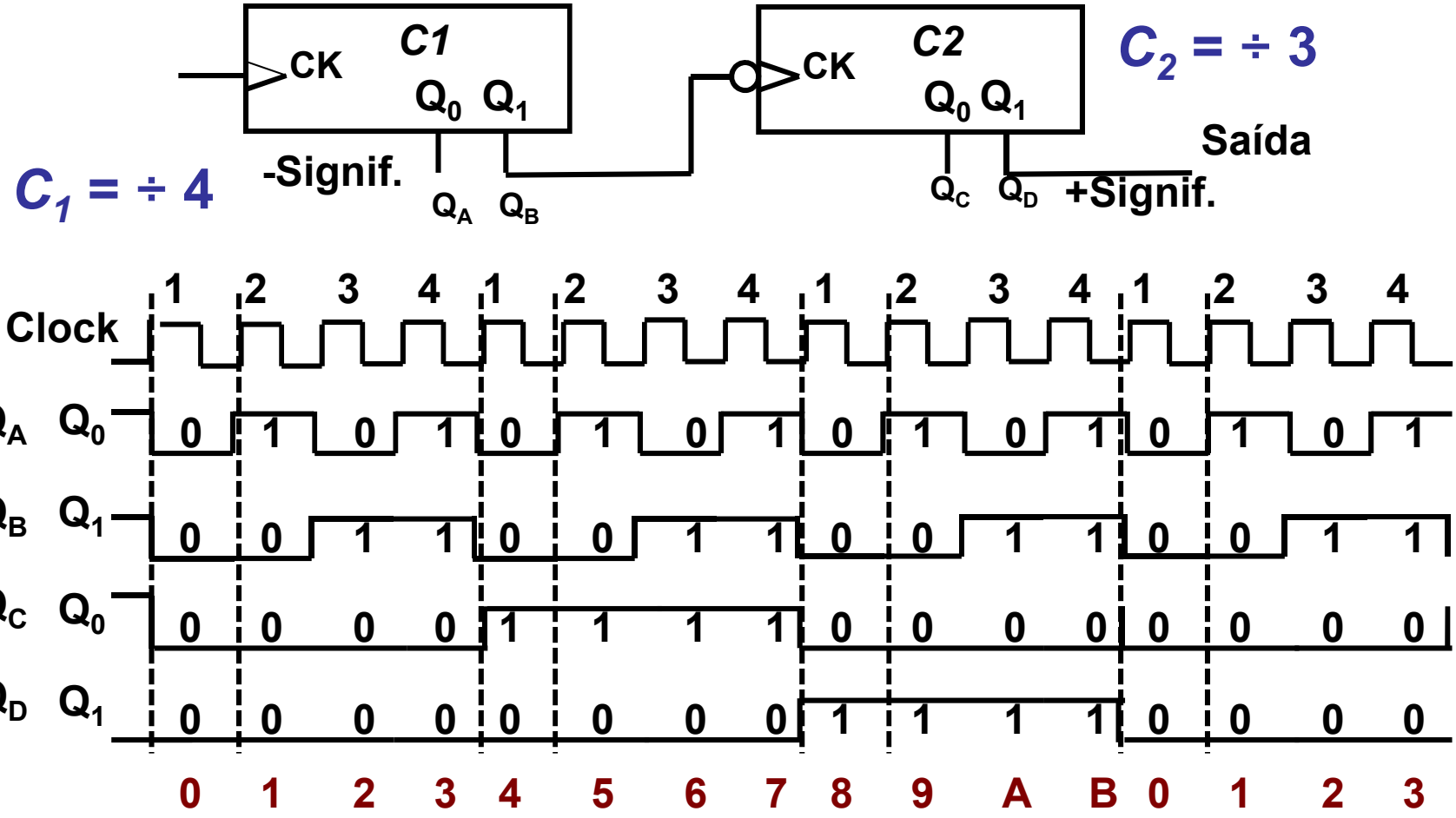
| $Q_D$ | $Q_C$ | $Q_B$ | $Q_A$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

**$C_1$ Módulo $M_1 \neq 2^n$**

**$C_1 = \div 3$ (-Significativo)**

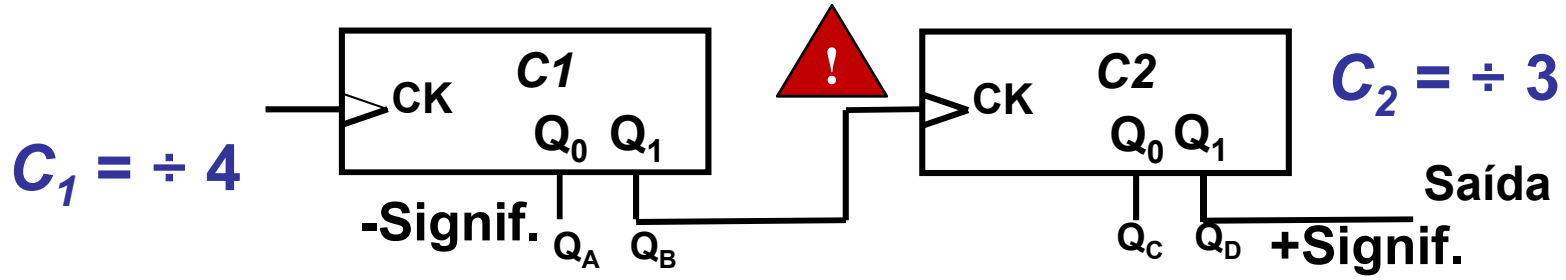
**Sequência NÃO Binária**

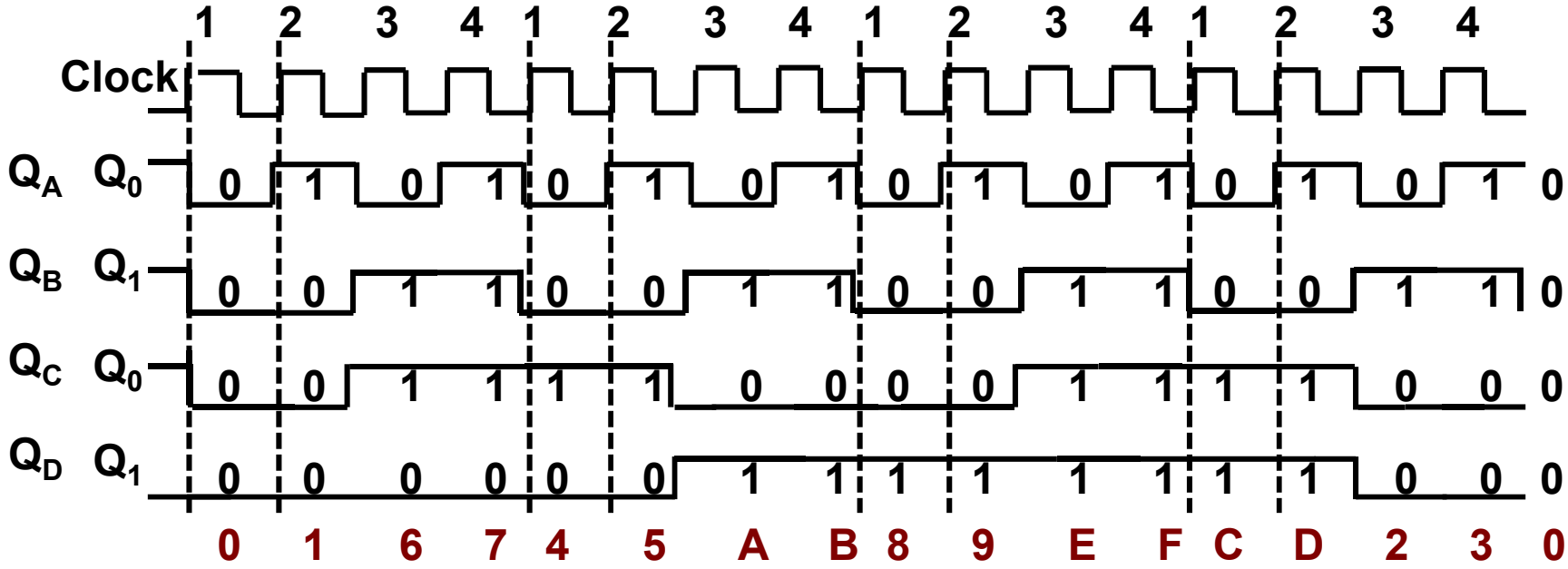
**Pergunta**: e se, neste caso, tornássemos o contador $C_1$ sensível à borda de subida do *clock*?




**Resposta**: as formas de onda seriam as mesmas, apenas havendo um deslocamento de meio período de *clock*.

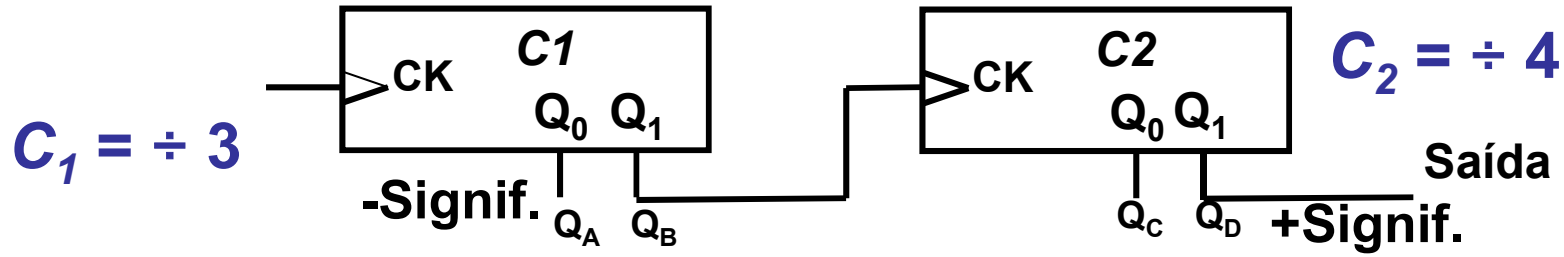
**Pergunta**: e se fizéssemos o *clock* de $C_2$ também sensível à borda de subida?

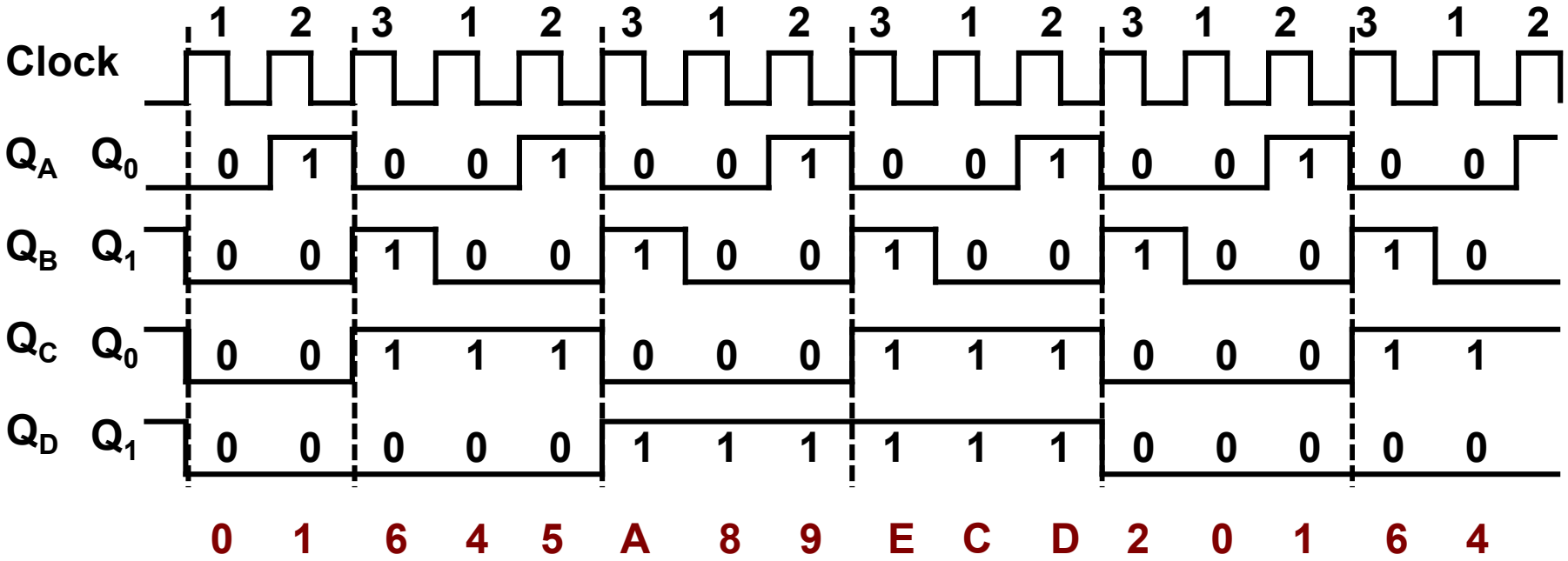



**Resposta**: As formas de onda se alteram! A sequência não é binária, o contador é módulo 16, e a onda é quadrada

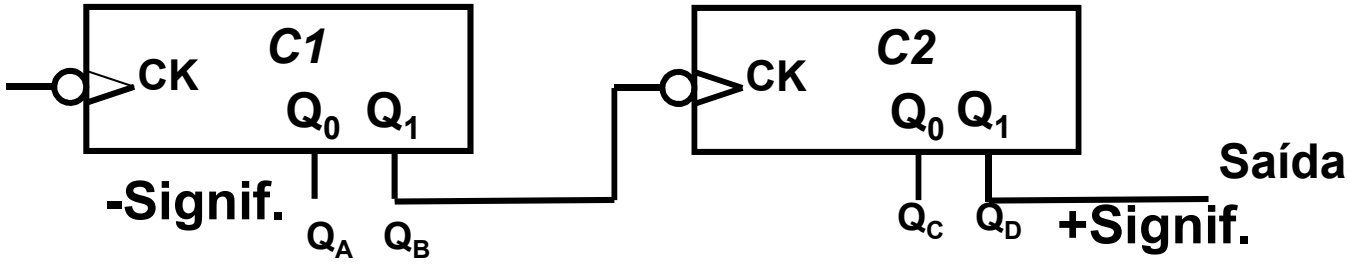
**Pergunta**: e se C1 for módulo impar, podemos fazer o *clock* de $C_2$ sensível à borda de subida?




**Resposta**: A sequência continua não sendo binária, mas a onda é quadrada, com a frequência dividida por 12

# Associação Assíncrona de Contadores




| Contador C1 | Contador C2 | Código Sequência Contagem | *Duty Cicle* *Bit* Saída + Significativo | Módulo da Contagem |
|---|---|---|---|---|
| Binário Módulo $M = 2^n$ | Binário Módulo $M \neq 2^n$ | Binário | Onda Retangular $f=f_{CK} / 2^n.M$ | $2^n . M$ |
| Binário Módulo $M \neq 2^n$ | Binário Módulo $M = 2^n$ | Não Binário | Onda Quadrada $f=f_{CK} / 2^n.M$ | $2^n . M$ |

# Contadores – Geração de Onda Quadrada

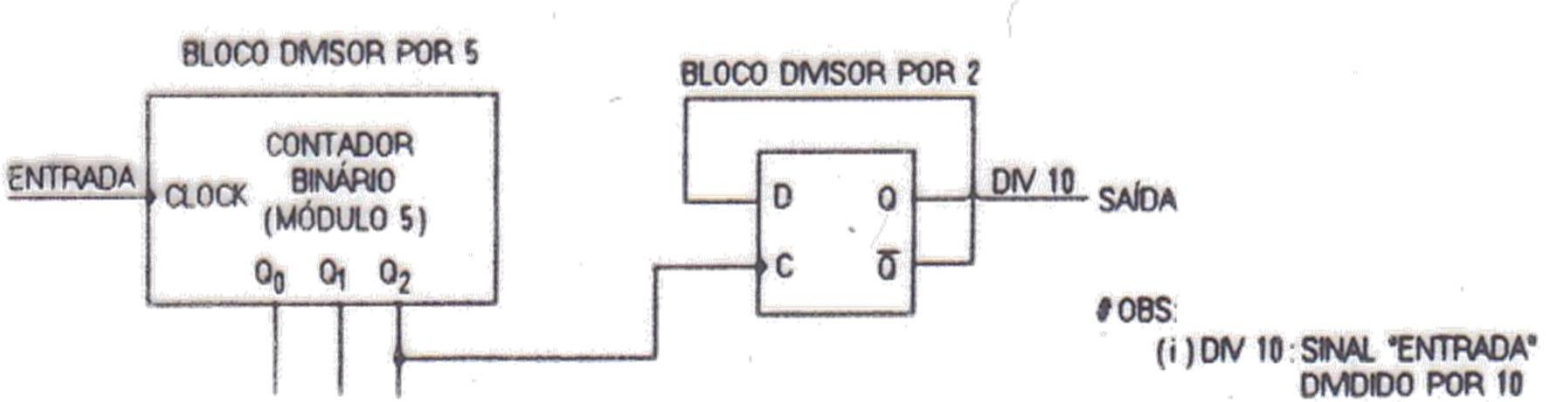
- **Problema**: gerar onda quadrada com frequência menor do que clock original
- Por que quadrada?
  - Reduz nível DC do sinal
    - » Transmissão de sinais – Diminui perdas;
    - » Conversão digital-analógico – Previne saturação de capacitores;
    - » Aplicações diversas – Qualquer luz piscante em que o tempo total da luz apagada é igual ao tempo total da luz acesa ...

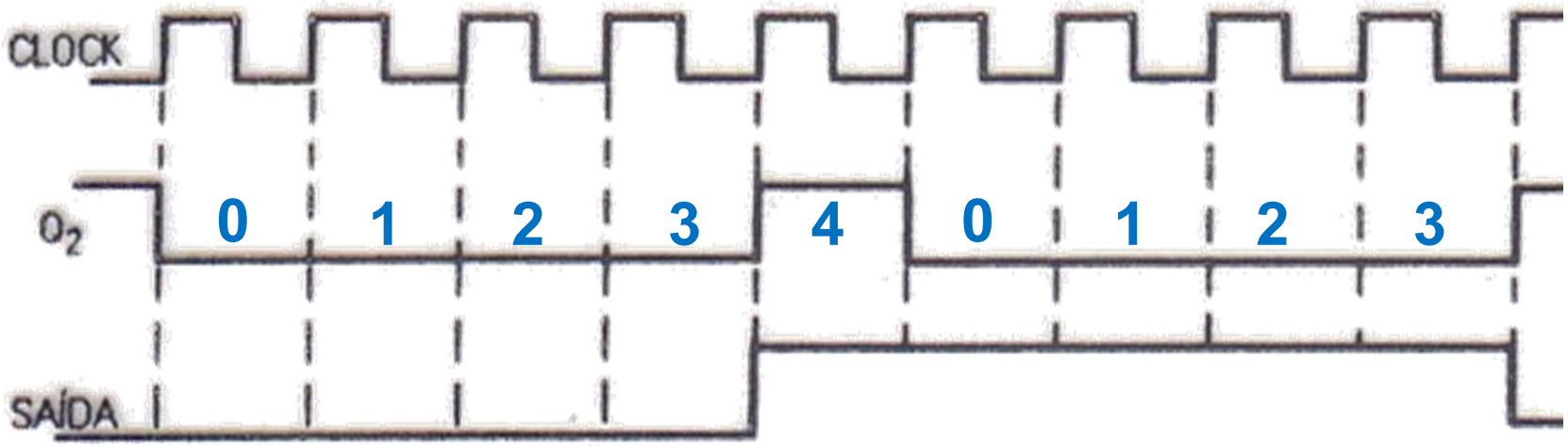
# Contadores – Geração de Onda Quadrada

- **Problema**: gerar onda quadrada com frequência menor do que clock original
- **Sub-problema 1**: divisão por **número par**
- Solução possível: associação já discutida...
  - Contador módulo n/2, que não precisa ser gerador de onda quadrada
  - Saída mais significativa usada como *clock* para Flip-Flop em configuração divisor por 2
    - » Pode ser borda de subida ou descida
  - $\mathbf{f_{CK\text{-}SAÍDA} = [f_{CK\text{-}ENTRADA}/(n/2)] \cdot 1/2}$

# Contadores – Geração de Onda Quadrada

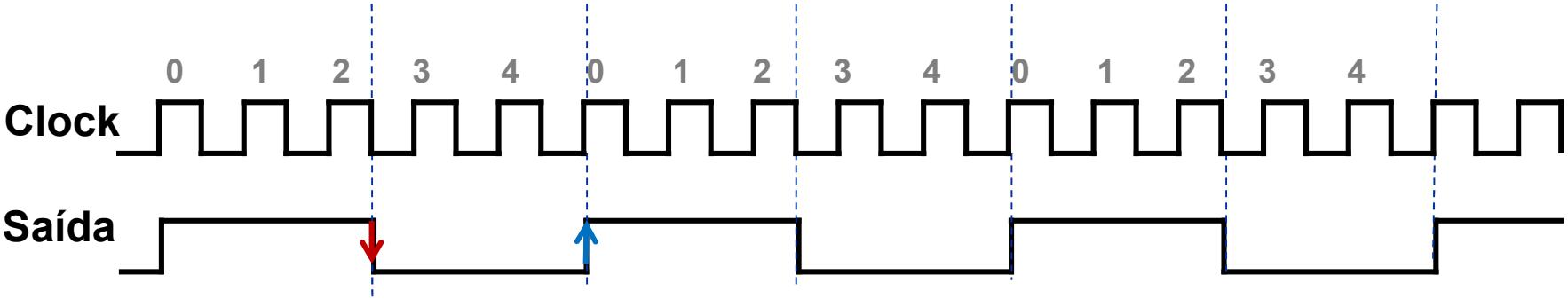
- Para divisão por número par:



# Contadores – Geração de Onda Quadrada

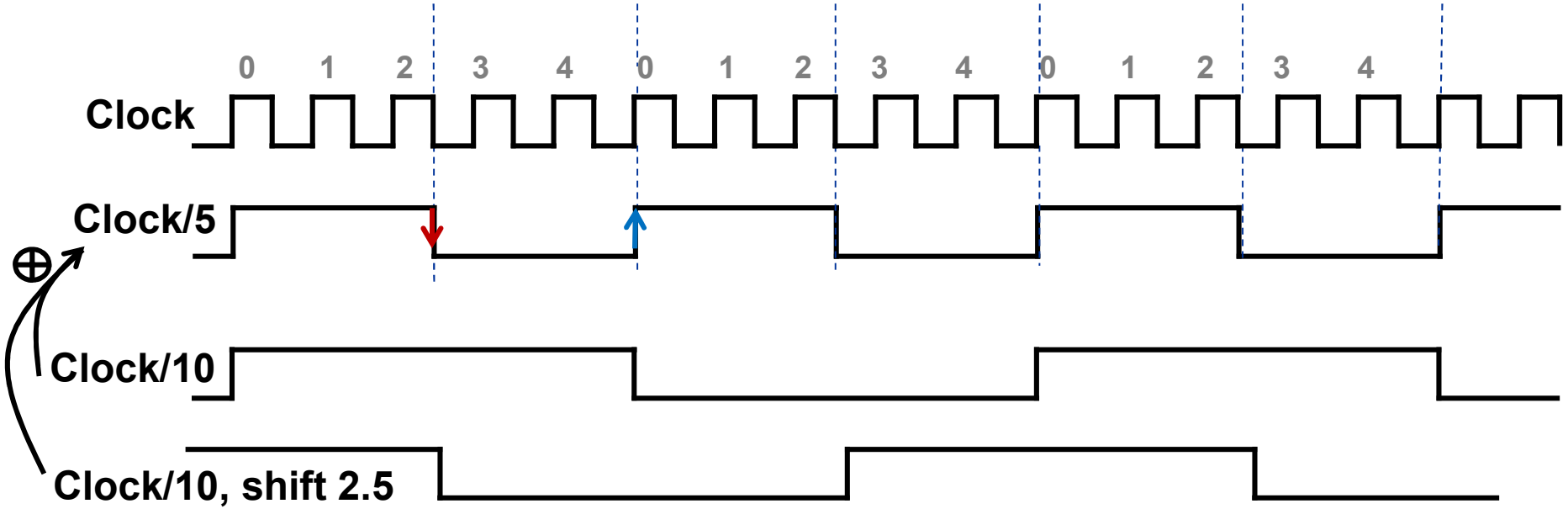
- **<u>Problema</u>**: gerar onda quadrada com frequência menor do que clock original
- **<u>Sub-problema 2</u>**: divisão por **número ímpar**




- Solução:
  - São necessárias mudanças, dentro do contador, tanto na borda de subida como de descida, associadas a lógica adicional, para gerar a onda quadrada na saída.

# Contadores – Geração de Onda Quadrada

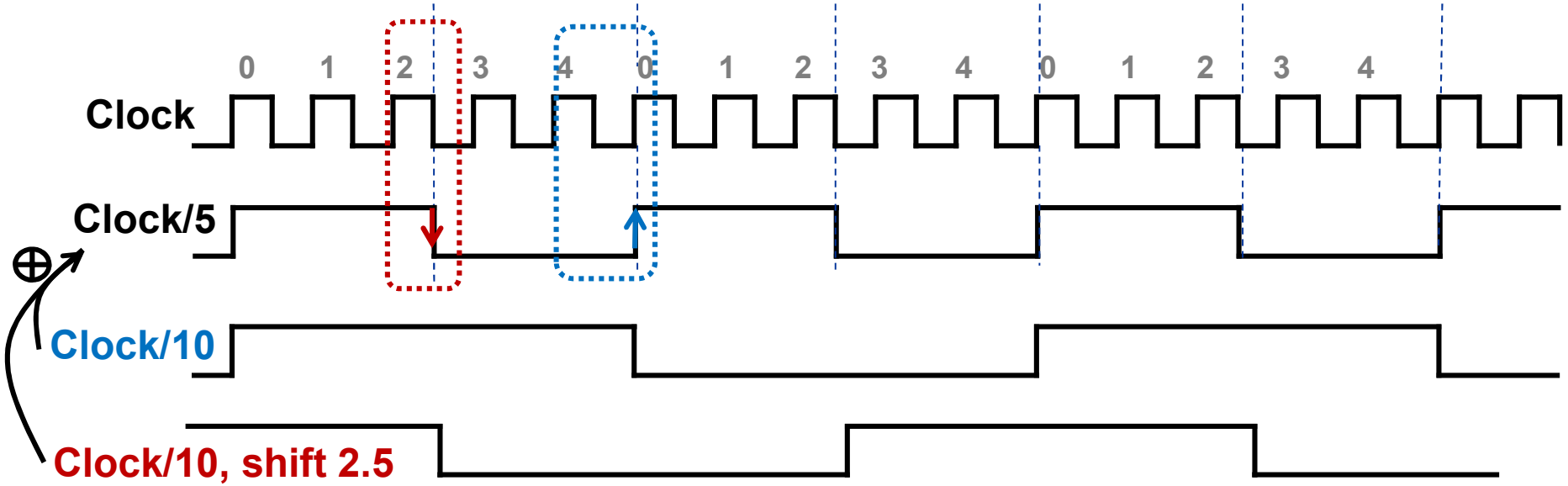
- **<u>Problema</u>**: onda quadrada, divisão por **n = 5 (ímpar)**




- Ideia:
  - Sabemos dividir por 2n = 10 com onda quadrada (5*2)
  - E podemos deslocar essa onda para sua transição aparecer após no “n/2 = 2.5 clock”, nas bordas de descida
  - Agora basta um XOR ou XNOR entre esses dois sinais!

# Contadores – Geração de Onda Quadrada

- **<u>Problema</u>**: onda quadrada, divisão por **n = 5 (ímpar)**



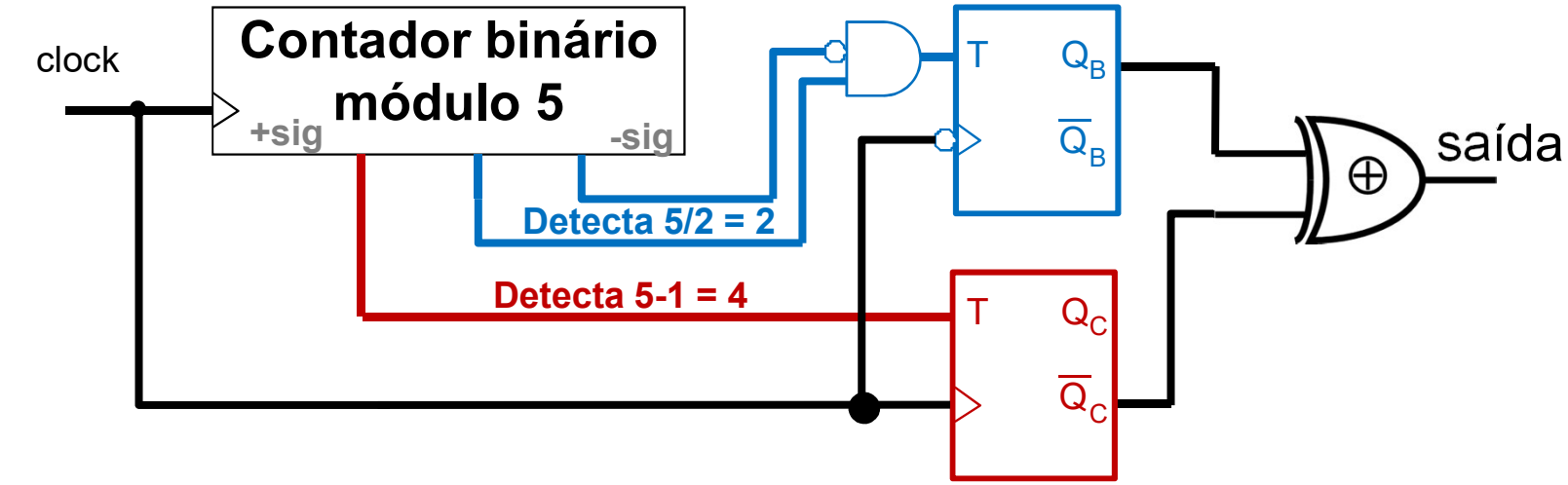

- Ideia:
  - Sabemos dividir por 2n=10 com onda quadrada (5*2)
    - » Basta fazer a inversão ao detectar n-1, com borda de subida
  - E podemos deslocar essa onda para sua transição aparecer após no “2,5 clock”, nas bordas de descida
    - » Inverter Flip-Flop ao detectar n/2 = 2 e for borda de descida

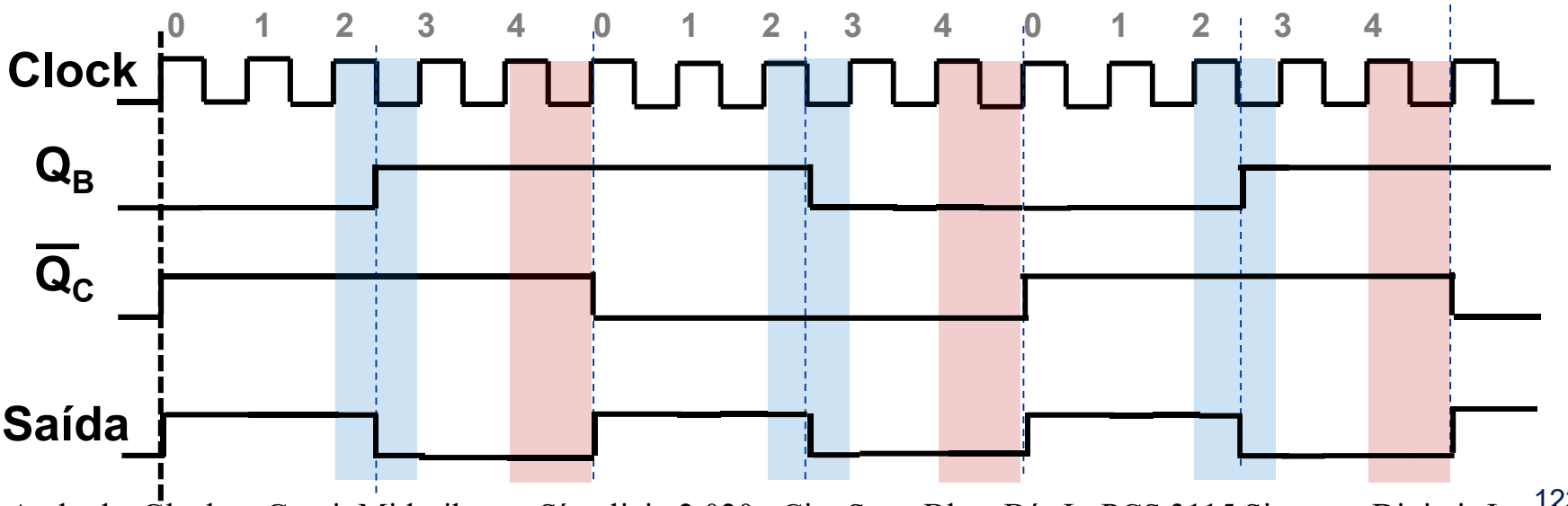
# Contadores – Geração de Onda Quadrada

- Uma estratégia para divisão por n ímpar:
  - **Bloco A**: um **contador módulo n** ativo na **borda de subida**
  - **Bloco B**: um **detector** ativado quando a saída do Bloco A for $\lfloor \mathbf{n/2} \rfloor$, ligado a um **Flip-Flop tipo T** ativo na **borda de descida**
  - **Bloco C**: um **detector** ativado quando a saída do Bloco A for **n-1**, ligado a um **Flip-Flop tipo T** ativo na **borda de subida**
  - **Saída**: operação de "**XOR**" ou "**XNOR**" entre a saída dos blocos **B e C**

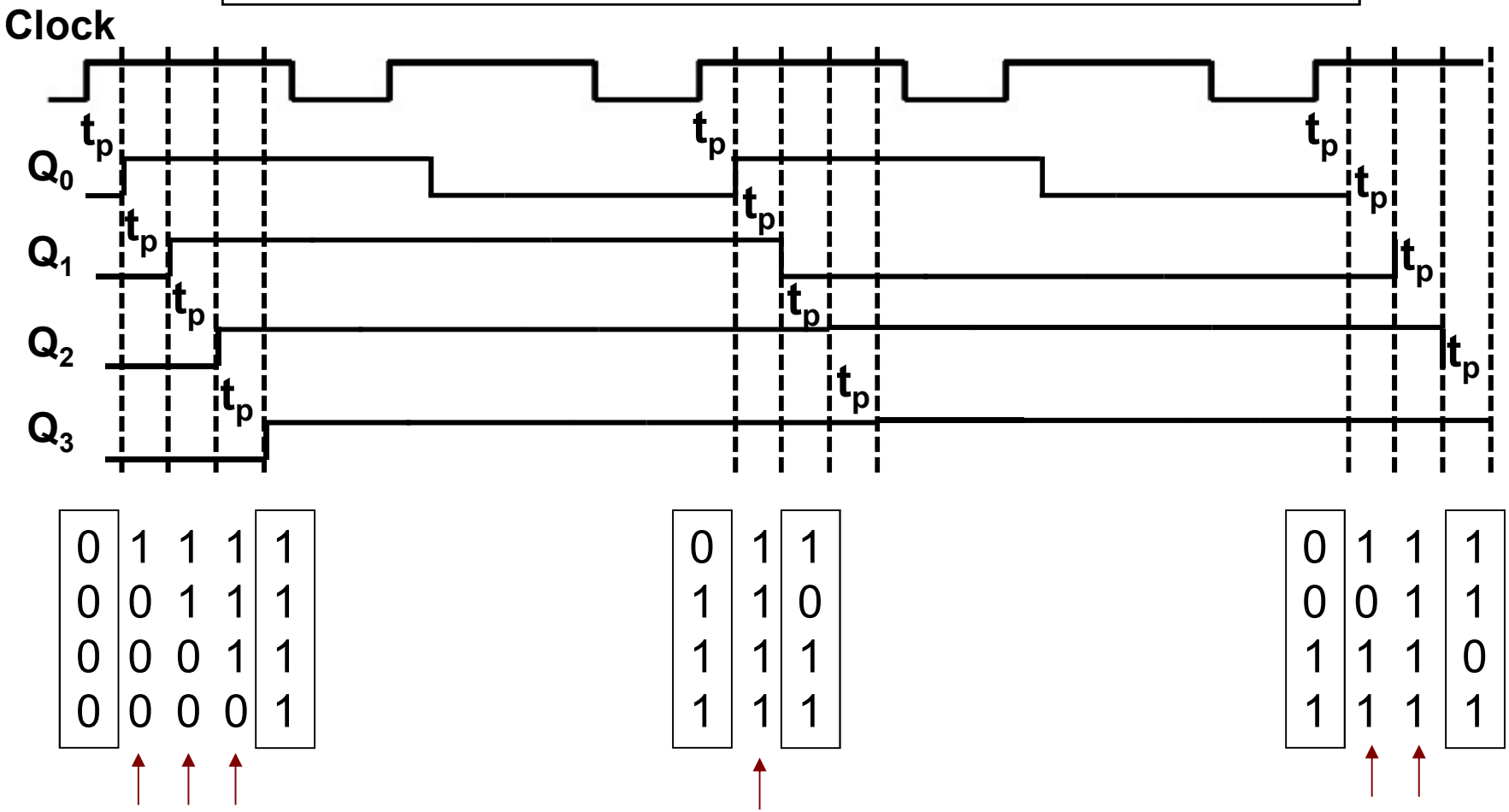
# Contadores – Geração de Onda Quadrada

- Exemplo usando método descrito: divisão por 5



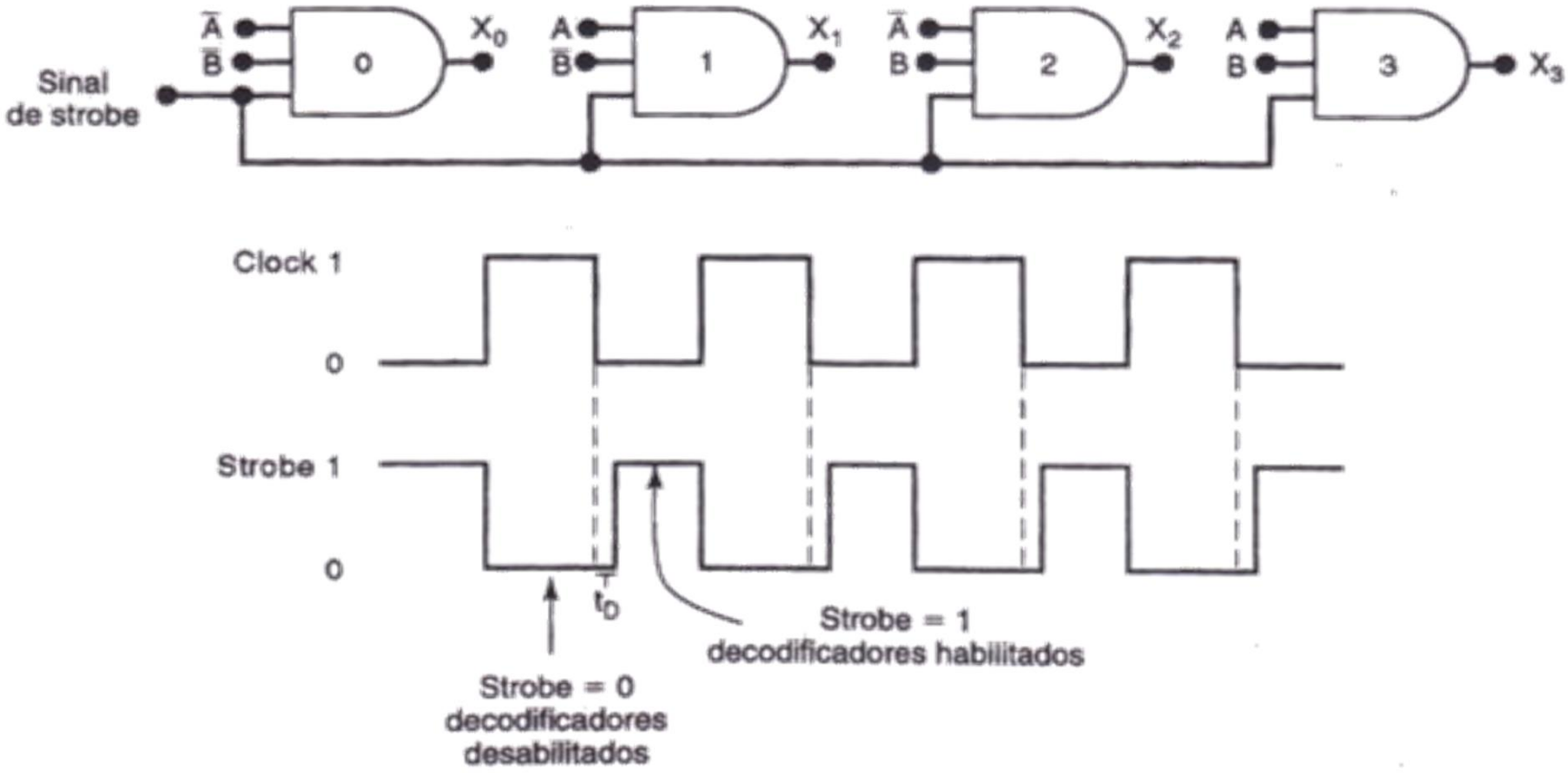



# Contadores Assíncronos



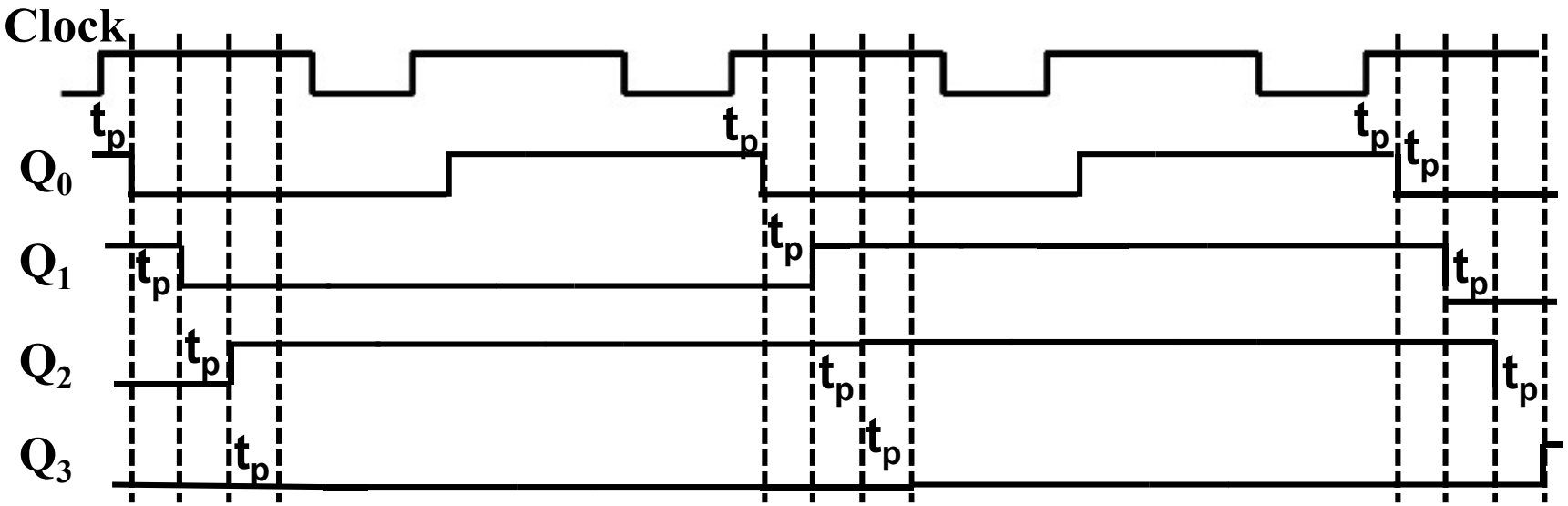

- Contador Assíncrono Decrescente: estados espúrios devido a $t_p$ dos Flip-Flop's ➔ **<u>Como eliminar</u>**?

# Contadores Assíncronos




- Eliminando glitches de decodificação: decodificadores desabilitados no intervalo relativo aos estados espúrios provocados pelos $t_p$ dos FFs
  - Uso de sinal de “strobe” (inverso do clock)

# Contadores Assíncronos




| | | | | |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | |
| 1 | 1 | 0 | 0 | |
| 0 | 0 | 0 | 1 | |
| 0 | 0 | 0 | 0 | |

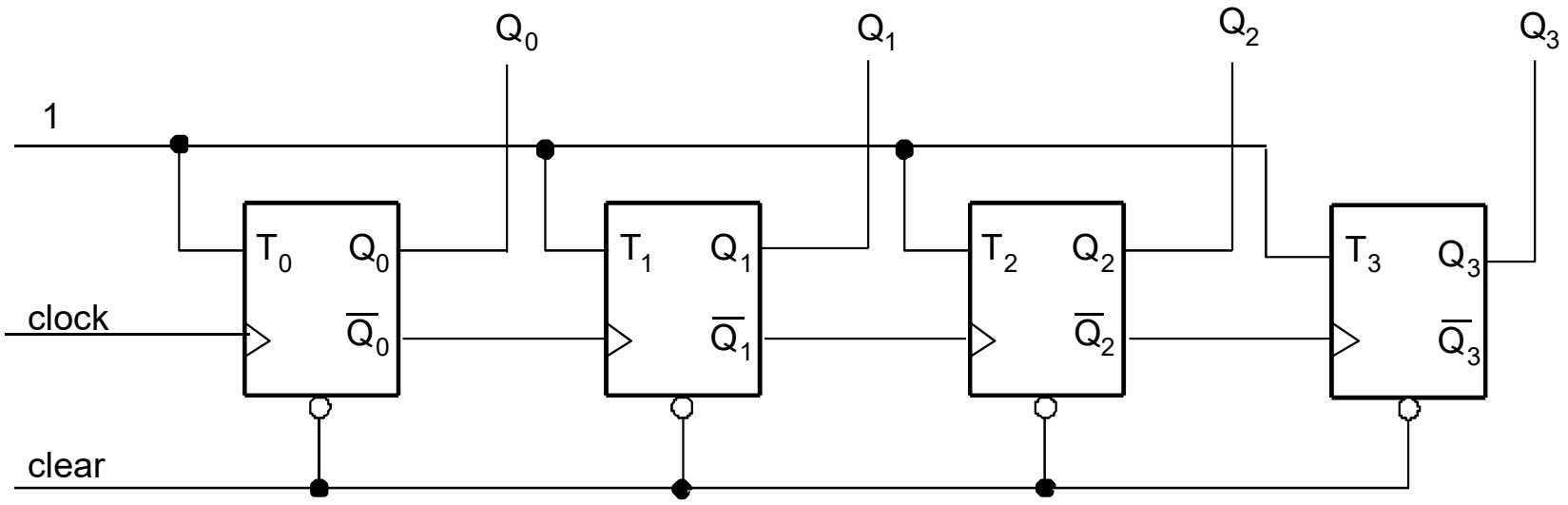
| | | | |
|---|---|---|---|
| 1 | 0 | 0 | |
| 0 | 0 | 1 | |
| 1 | 1 | 1 | |
| 0 | 0 | 0 | |

| | | | | |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 |

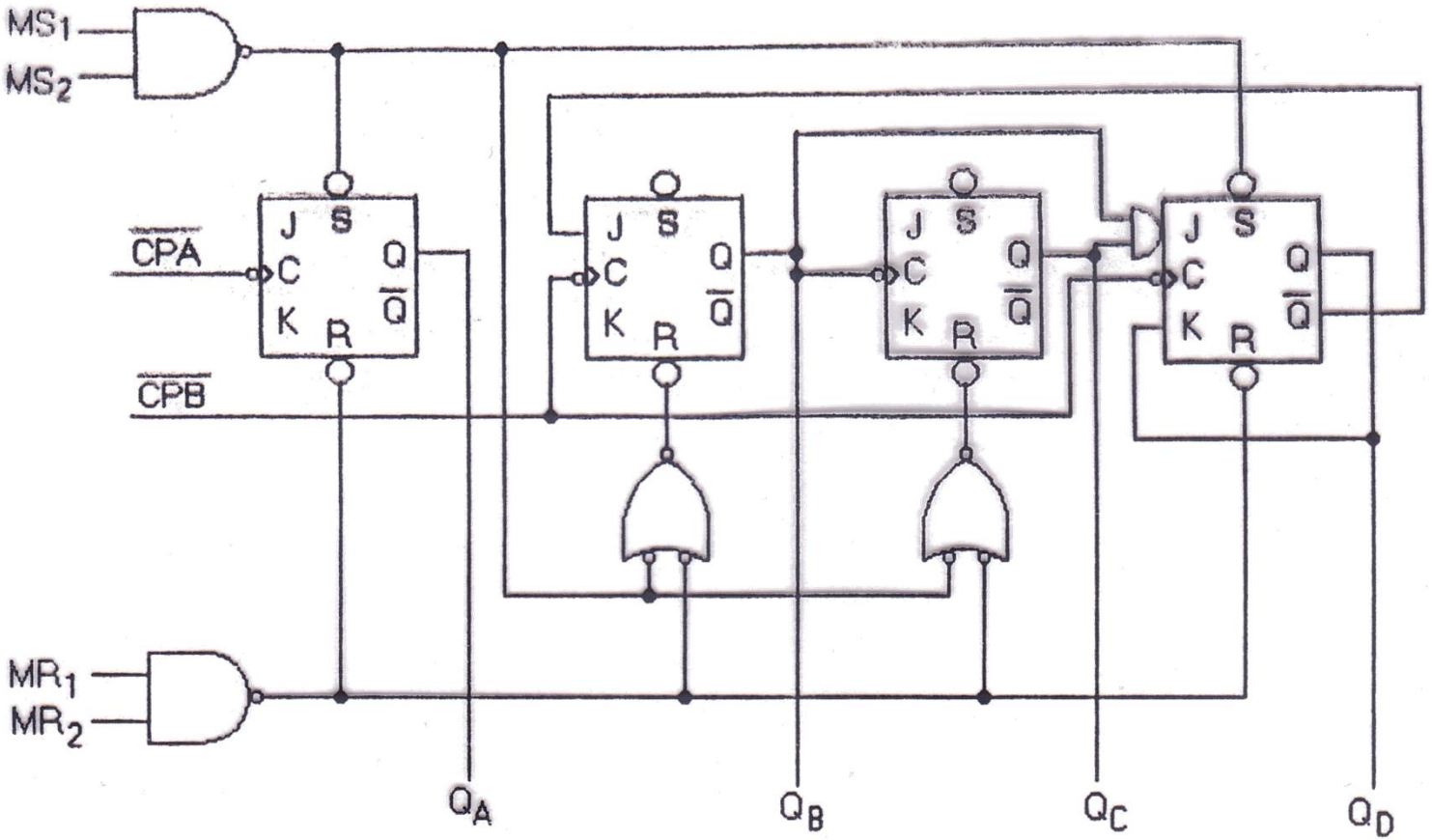
- Contador Assíncrono Crescente: **estados espúrios** devido ao $t_p$ dos FFs: **strobe** na decodificação também se aplica aqui

# Contadores Assíncronos

- Outro exemplo – Contador assíncrono crescente com Flip-Flops tipo T.



# Contadores Assíncronos: Exemplos

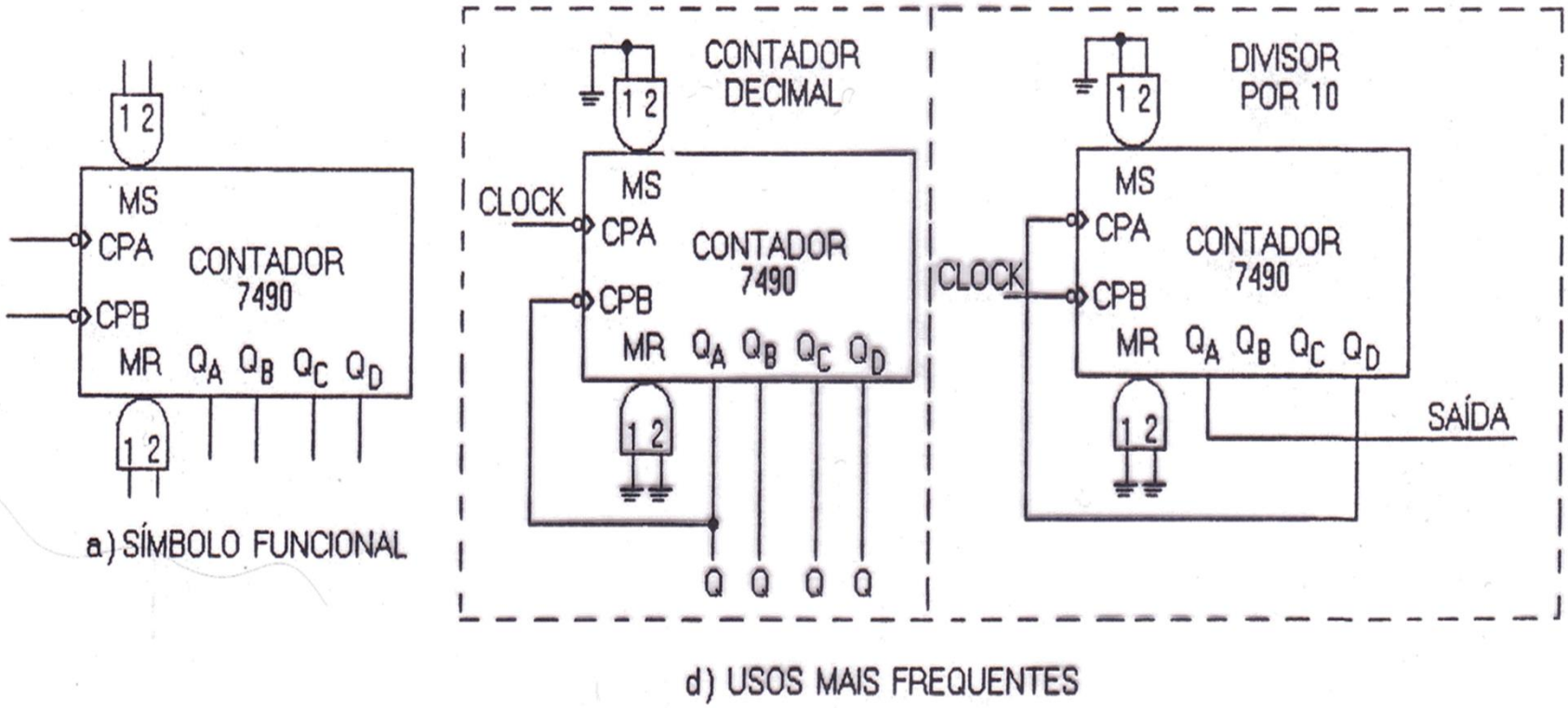



MS: Master Set CPA: clock para divisão por 2

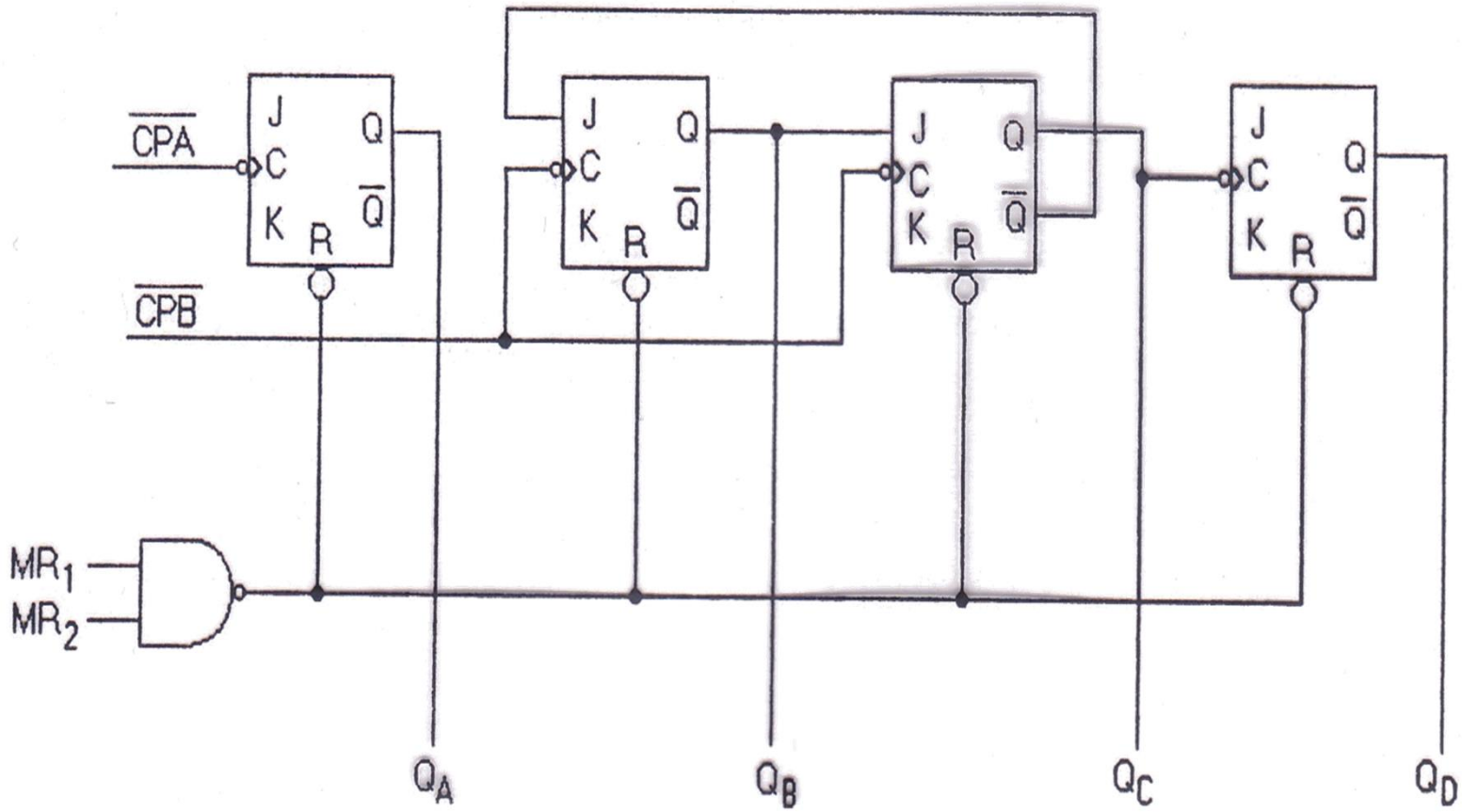
MR: Master Reset CPB: clock para divisão por 5

# Contadores Assíncronos: Exemplos

- Contador 7490 (usos frequentes):



# Contadores Assíncronos: Exemplos

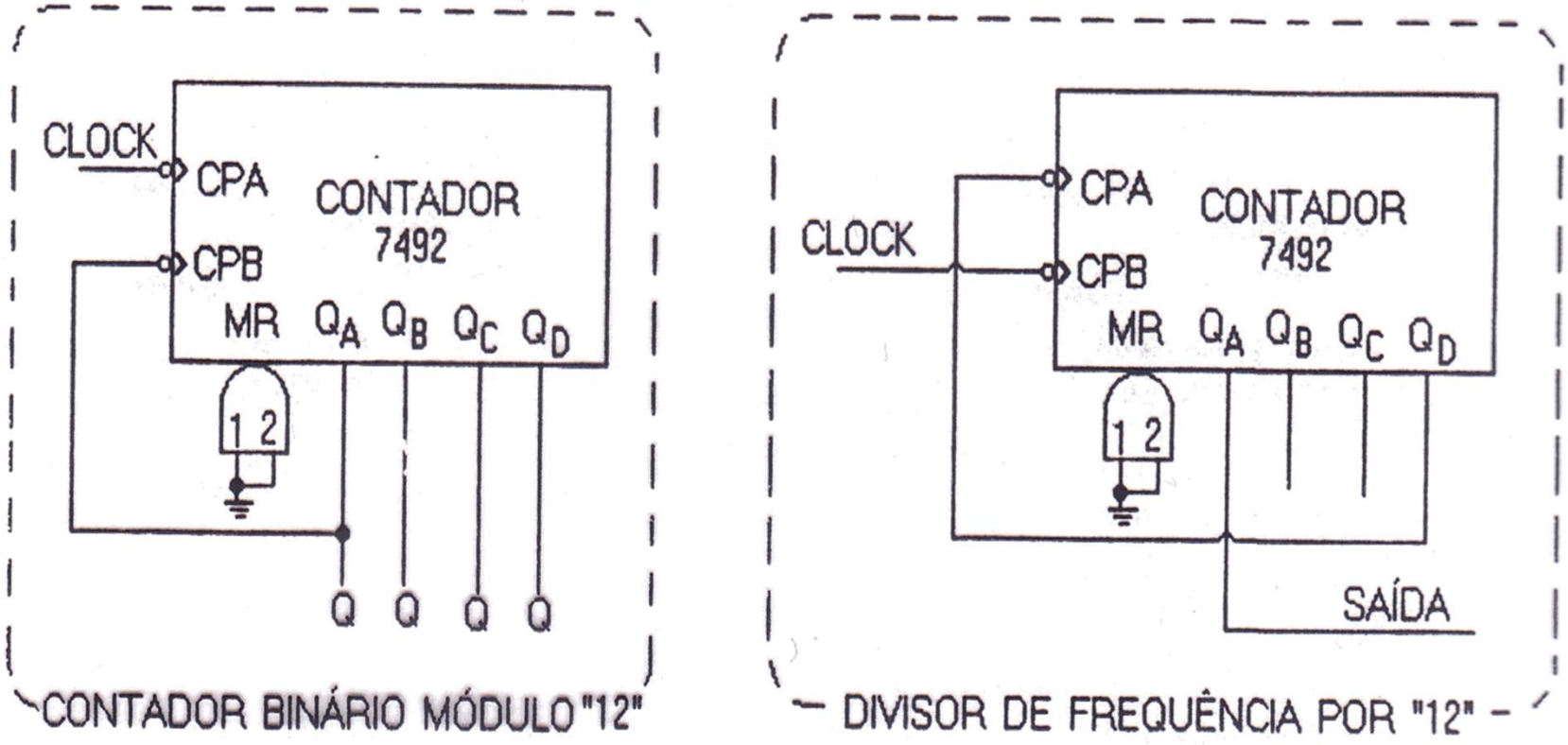



CPA: clock para divisão por 2

MR: Master Reset CPB: clock para divisão por 6

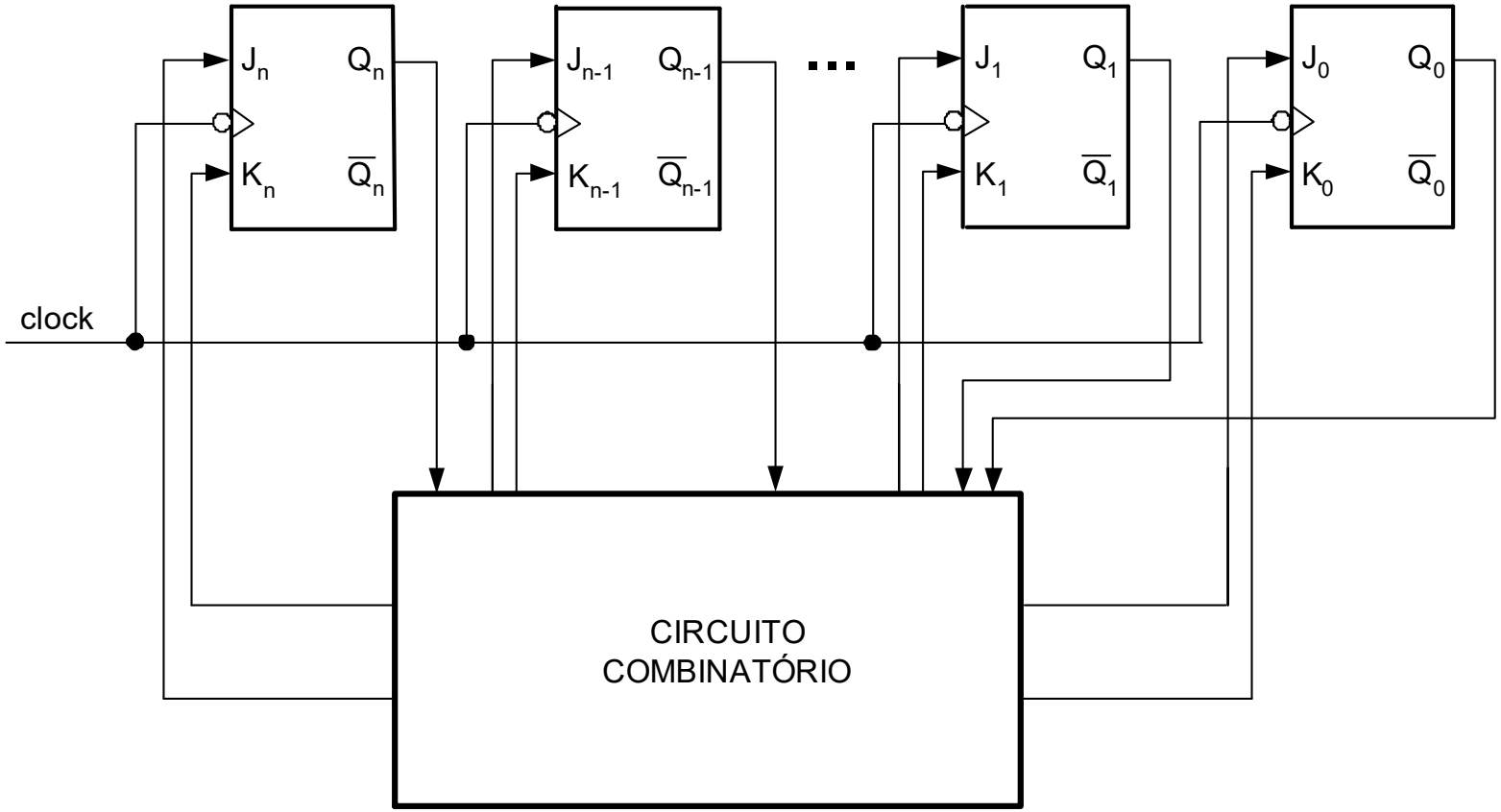
# Contadores Assíncronos: Exemplos

- Contador 7492 (usos frequentes):

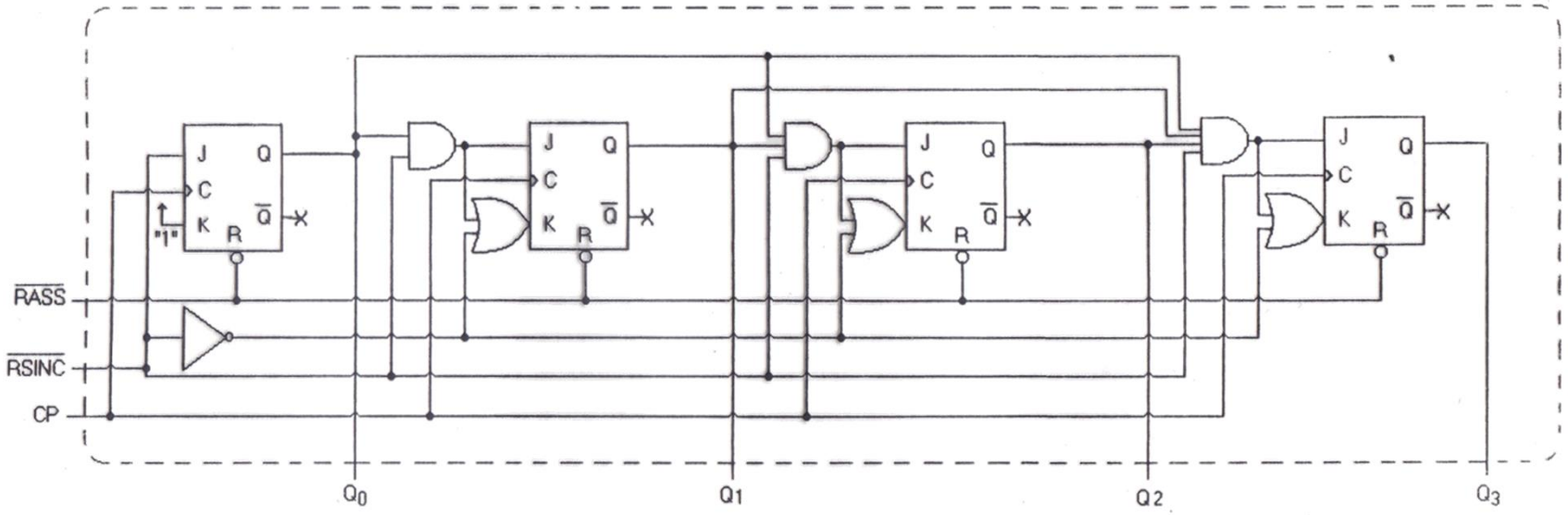


# Contadores Síncronos

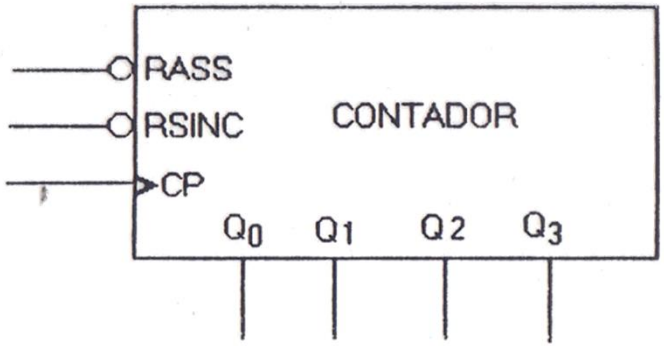
- Todos os FFs: (1) compartilham o mesmo clock e (2) têm usas saídas (estados) atualizadas no mesmo instante (mesma borda).



# Contadores Síncronos: Estruturas




a) CIRCUITO DO CONTADOR




b) SÍMBOLO FUNCIONAL

**Reset assíncrono**

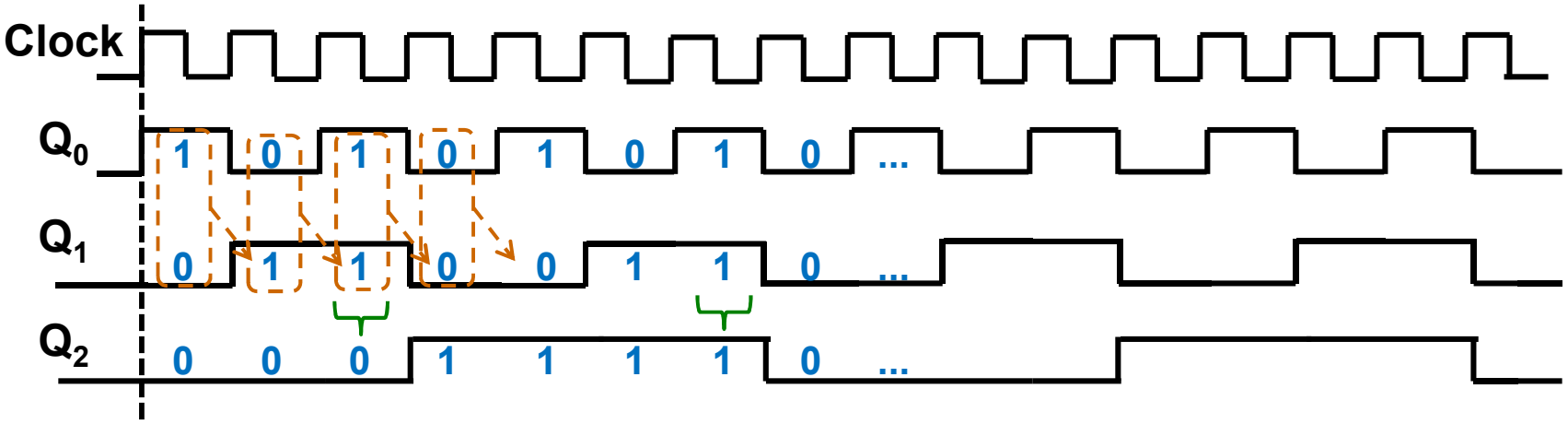
**Reset síncrono**

| ENTRADAS | | | FUNÇÃO ASSOCIADA |
|---|---|---|---|
| $\overline{RASS}$ | $\overline{RSINC}$ | CP | |
| 0 | X | X | **Anula** |
| 1 | 0 | ↑ | **Anula** |
| 1 | 1 | ↑ | **Conta** |

c) TABELA FUNCIONAL

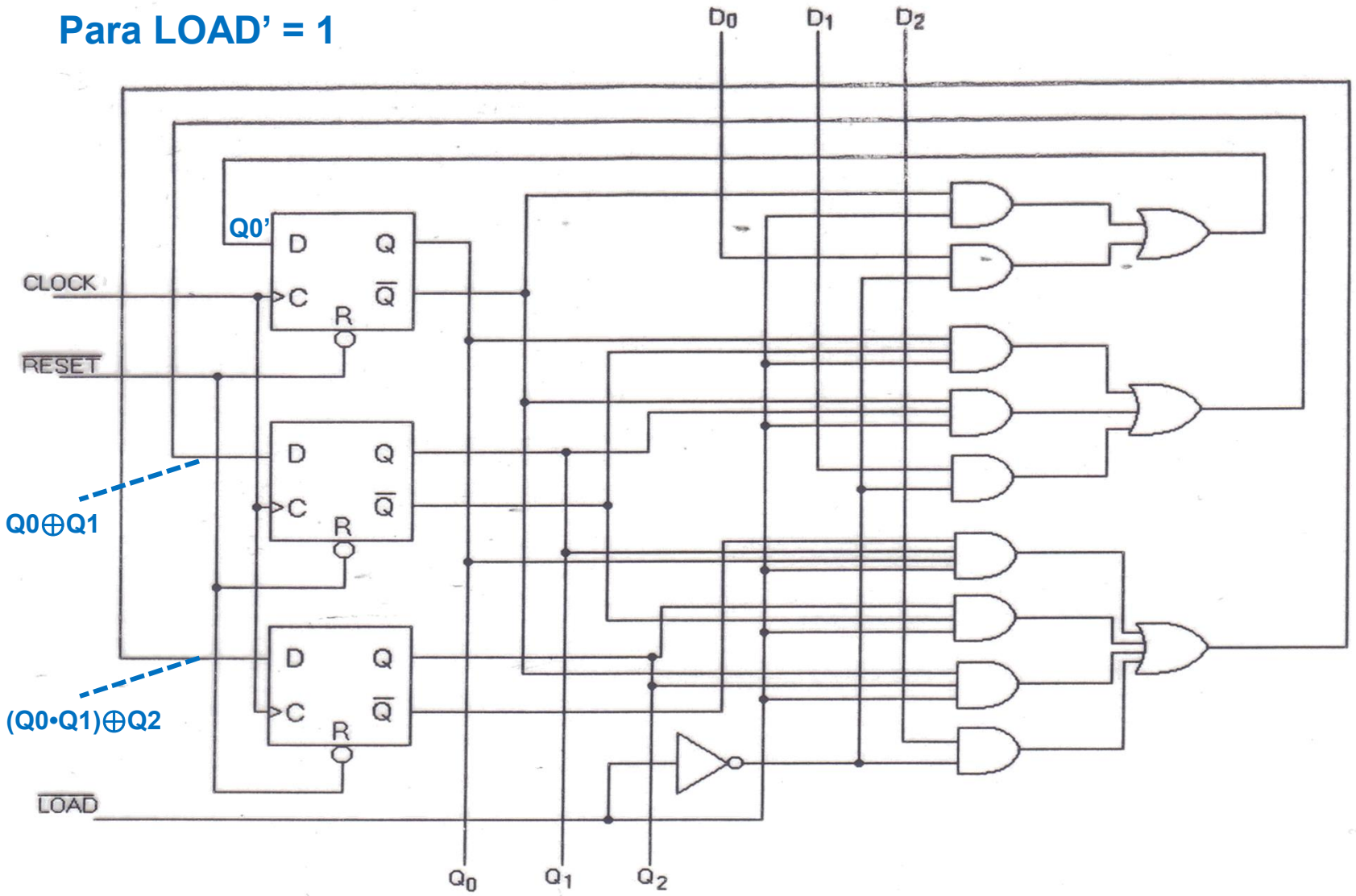
# Contadores Síncronos: Estruturas

- Flip-Flops tipo D: como seria a alimentação em cada Di para obter comportamento síncrono...?

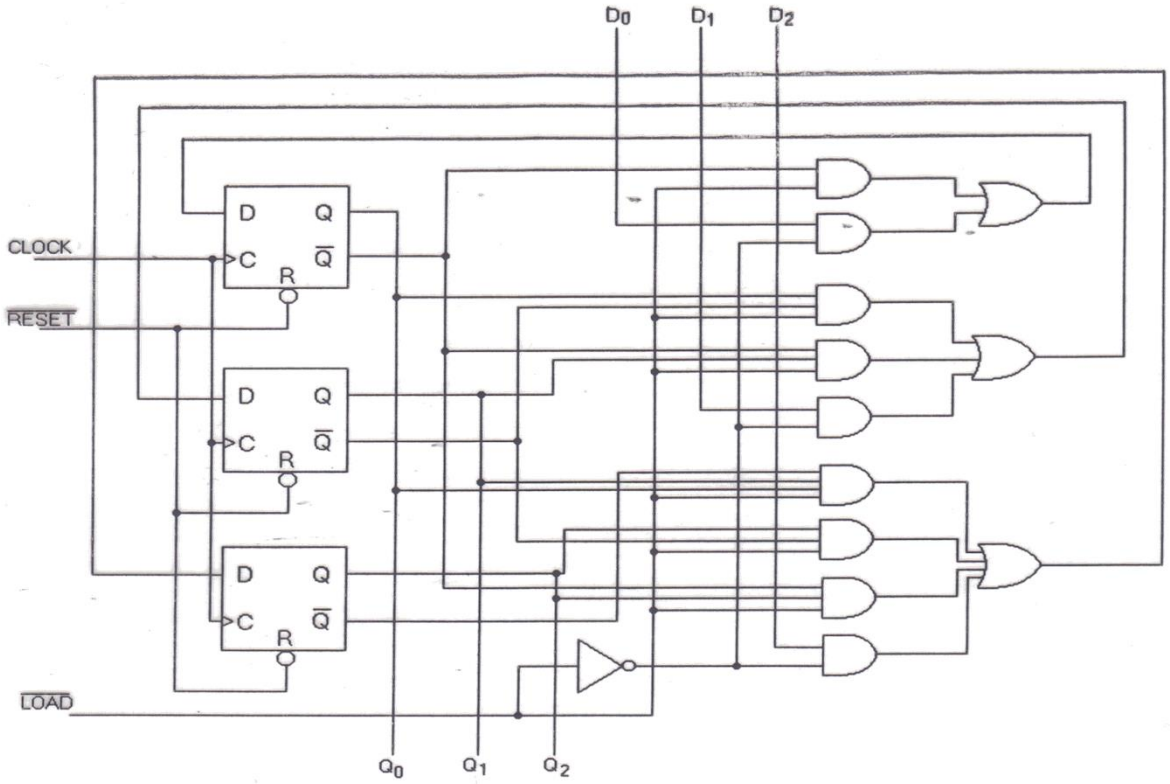



```
D0 <= Q0'
D1 <= Q0 xor Q1            -- se Q0 é 1, inverte Q1
D2 <= (Q0 and Q1) xor Q2 -- se Q0 e Q1 são 1, inverte Q2
...                -- se anteriores forem 1, inverte Qn
```

# Contadores Síncronos: Estruturas



# Contadores Síncronos: Estruturas




**Load' = 0 ➔ $IN_0 = D_0$ ; $IN_1 = D_1$ ; $IN_2 = D_2$**

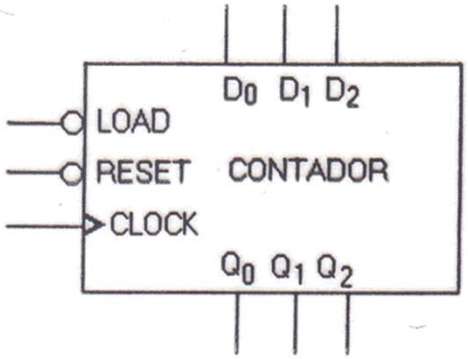
**Load' = 1 ➔**

$IN_0 = Q_0'$

$IN_1 = Q_0 \cdot Q_1' + Q_0' \cdot Q_1 = Q_0 \oplus Q_1$

$IN_2 = Q_0 \cdot Q_1 \cdot Q_2' + Q_1' \cdot Q_2 + Q_0' \cdot Q_2 = (Q_0 \cdot Q_1) \oplus Q_2$

# Contadores Síncronos: Estruturas




a) SÍMBOLO FUNCIONAL

| "FLIP-FLOP" | FUNÇÃO ASSOCIADA<br>ENTRADA "Di" DO I-ÉSIMO "FLIP-FLOP" |
|---|---|
| 0 | $y_0 = LOAD.D_0 + \overline{LOAD}.\overline{Q_0}$ |
| 1 | $y_1 = LOAD.D_1 + \overline{LOAD}.[\overline{Q_1}.Q_0 + Q_1.\overline{Q_0}]$ |
| 2 | $y_2 = LOAD.D_2 + \overline{LOAD}.[\overline{Q_2}.Q_1.Q_0 + Q_2\overline{Q_1} + Q_2\overline{Q_0}]$ |

c) TABELA DAS FUNÇÕES LÓGICAS

| ENTRADAS | | | FUNÇÃO ASSOCIADA |
|---|---|---|---|
| $\overline{RESET}$ | $\overline{LOAD}$ | CLOCK | |
| 0 | X | X | <"ANULA"> |
| 1 | 0 | ↑ | <"CARGA PARALELA"> |
| 1 | 1 | ↑ | <"CONTA"> |

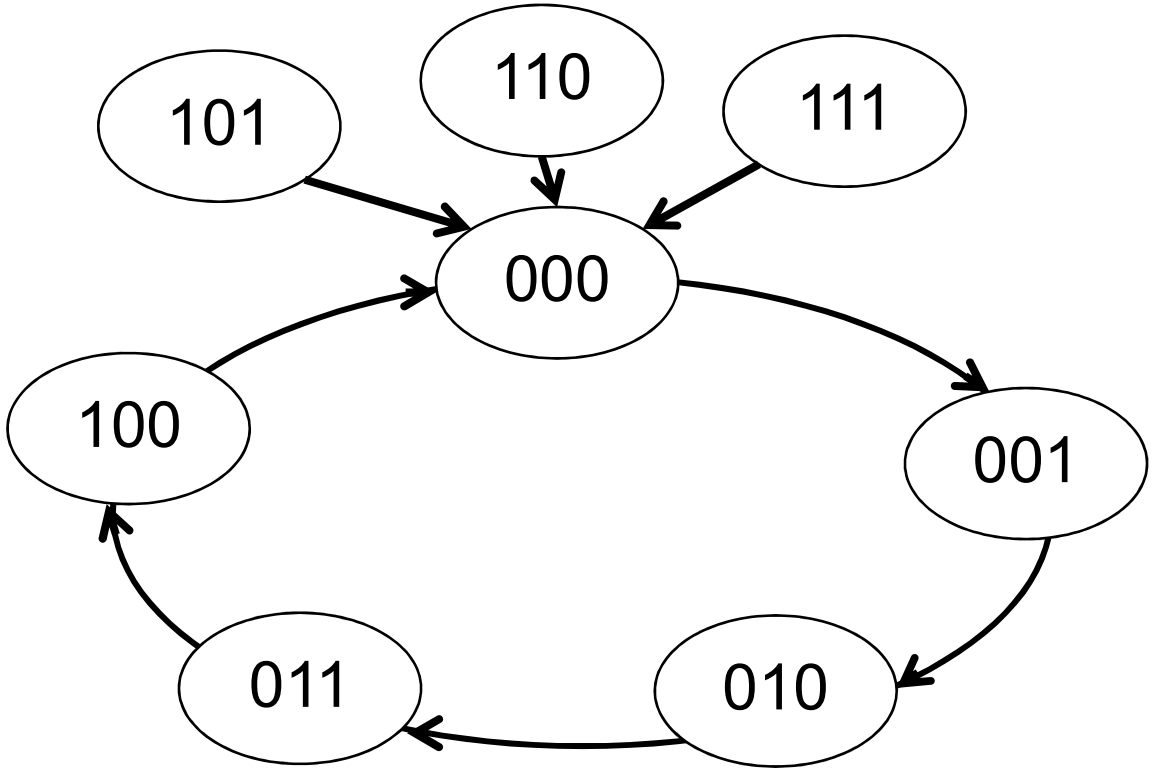
d) TABELA FUNCIONAL

# Contadores Síncronos – Síntese

- Vários métodos possíveis
- O que vamos adotar aqui:
  - Considerar contador como uma máquina combinatório-sequencial baseada no modelo de *Moore*.
  - Todas as considerações sobre síntese de máquinas usando modelo de Moore são válidas e podem ser aplicadas na síntese de contadores.

# Contadores Síncronos – Síntese

- Exemplo: contagem 0-4, usando FFs tipo D
  - São necessários 3 bits (C, B e A): 000 a 100
  - Estados 101, 110 e 111 são indesejáveis: abordagem “segura”



# Contadores Síncronos – Síntese

- Tabela de Transição de Estados.

| Linha | Estado Atual | | | | Próximo Estado | | |
|---|---|---|---|---|---|---|---|
| | C | B | A | | C | B | A |
| 0 | 0 | 0 | 0 | | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | | 0 | 1 | 1 |
| 3 | 0 | 1 | 1 | | 1 | 0 | 0 |
| 4 | 1 | 0 | 0 | | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | | 0 | 0 | 0 |
| 6 | 1 | 1 | 0 | | 0 | 0 | 0 |
| 7 | 1 | 1 | 1 | | 0 | 0 | 0 |

# Contadores Síncronos – Síntese

- Da tabela de transição de estados obtém-se a tabela de excitação.

| # | Estado Atual | | | Próximo Estado | | | $D_C$ | $D_B$ | $D_A$ |
|---|---|---|---|---|---|---|---|---|---|
| | C | B | A | C | B | A | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 3 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

# Contadores Síncronos – Síntese

- Determinar as Funções de Excitação ($D_C$,$D_B$,$D_A$)

| Atual | | | $D_C$ | $D_B$ | $D_A$ |
|---|---|---|---|---|---|
| C | B | A | | | |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |

| | | A | |
|---|---|---|---|
| C | B | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |

| A | |
|---|---|
| 0 | 1 |
| 0 | 1 |
| 1 | 0 |
| 0 | 0 |
| 0 | 0 |

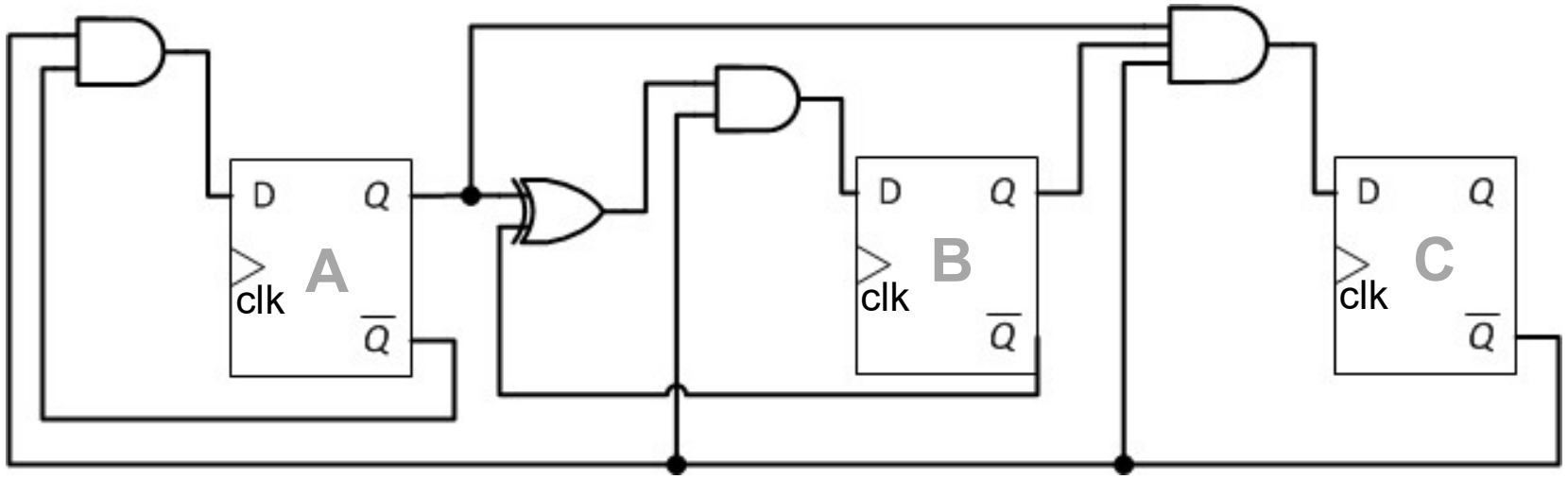
| A | |
|---|---|
| 0 | 1 |
| 0 | 0 |
| 0 | 1 |
| 0 | 0 |
| 0 | 0 |

$$D_A = A'C'$$
$$D_B = C'(A \oplus B')$$
$$D_C = ABC'$$

# Contadores Síncronos – Síntese

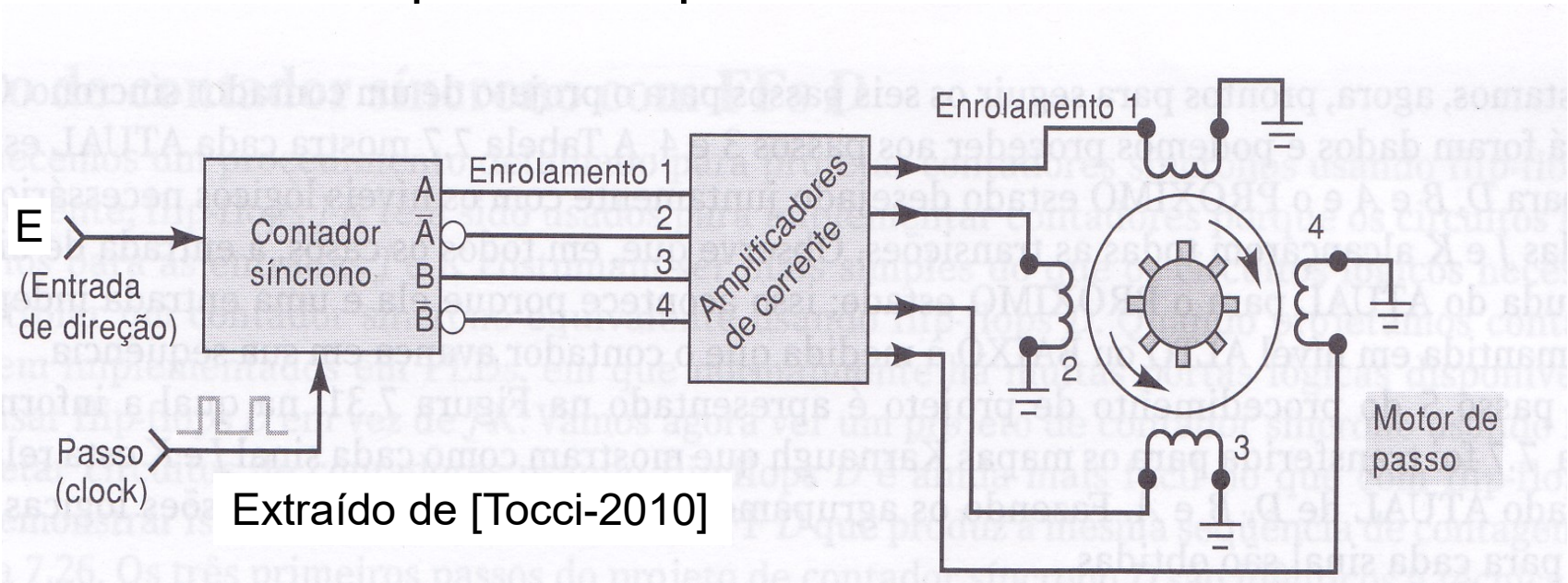
- Construir circuito lógico para ($D_C$,$D_B$,$D_A$)

**$D_A = A'C'$  $D_B = C'(A \oplus B')$  $D_C = ABC'$**



# Contadores Síncronos – Aplicações

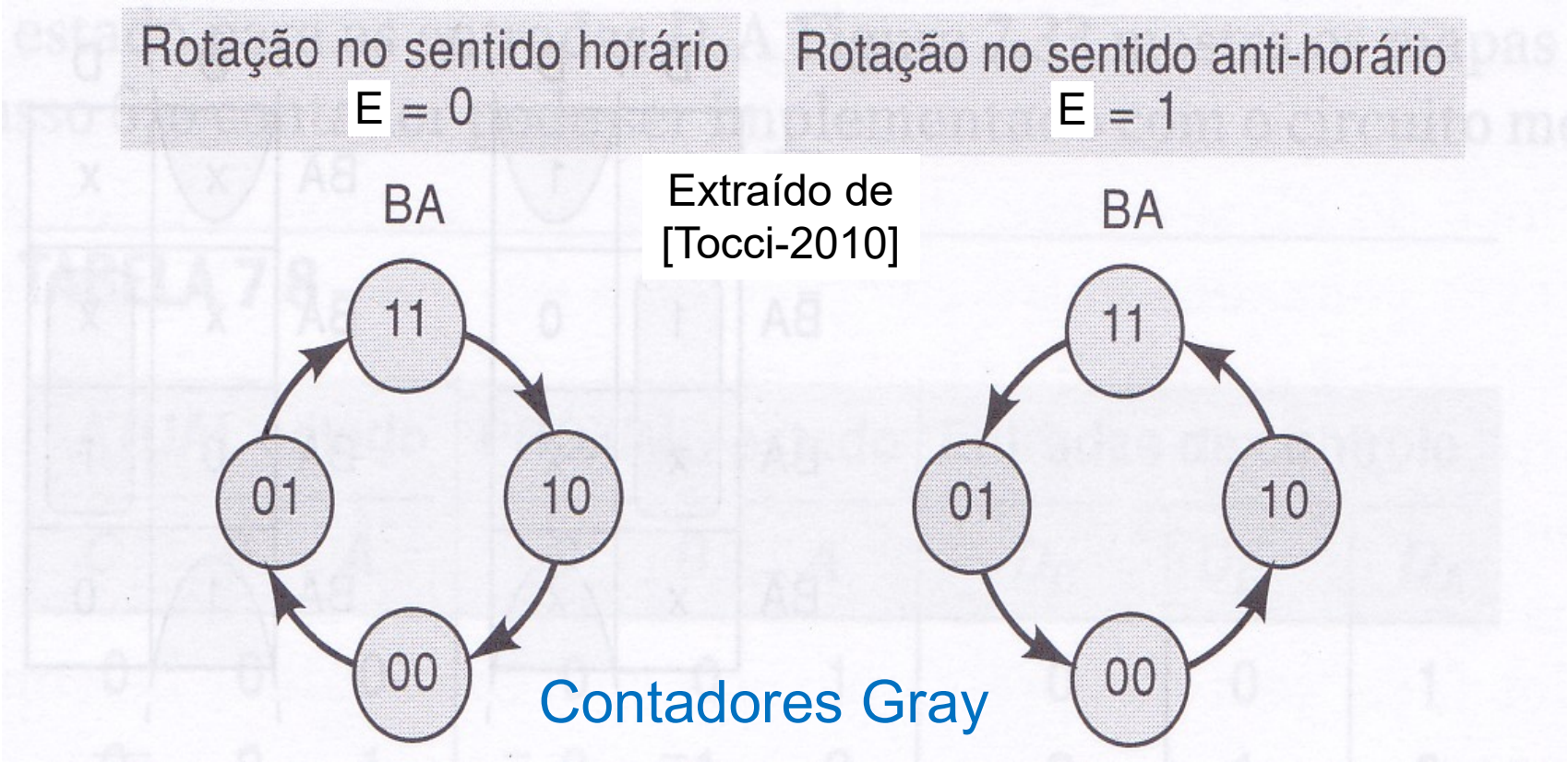
- Aplicaremos esta técnica de projeto em uma situação prática:
  - Controle de um motor de passo que gira em passos discretos de 15º por passo.
  - Os enrolamentos do motor devem ser energizados numa sequência específica.




Extraído de [Tocci-2010]

# Contadores Síncronos – Aplicações

- Os Diagramas de Transição de Estados que fornecem a sequência apropriada (de energização e desenergização) são:



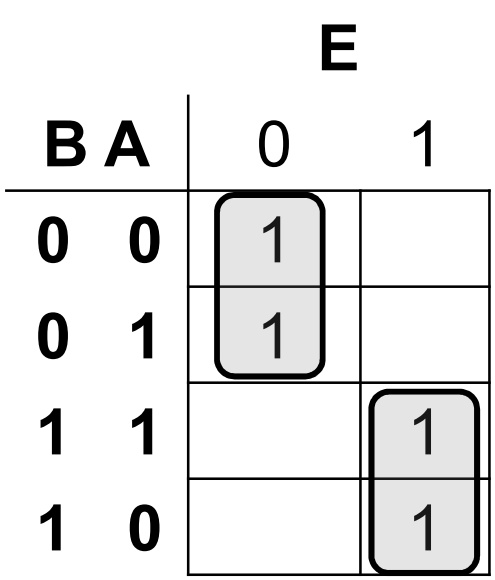
# Contadores Síncronos – Aplicações

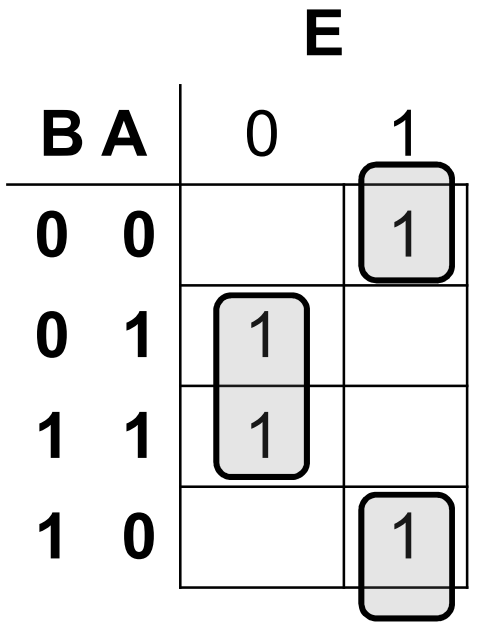
○

| Entrada | Estado Atual | | Próximo Estado | | Variáveis de excitação | |
|---|---|---|---|---|---|---|
| E | B | A | B | A | $D_B$ | $D_A$ |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 |

# Contadores Síncronos – Aplicações

- Funções de Excitação – $D_A$ e $D_B$.



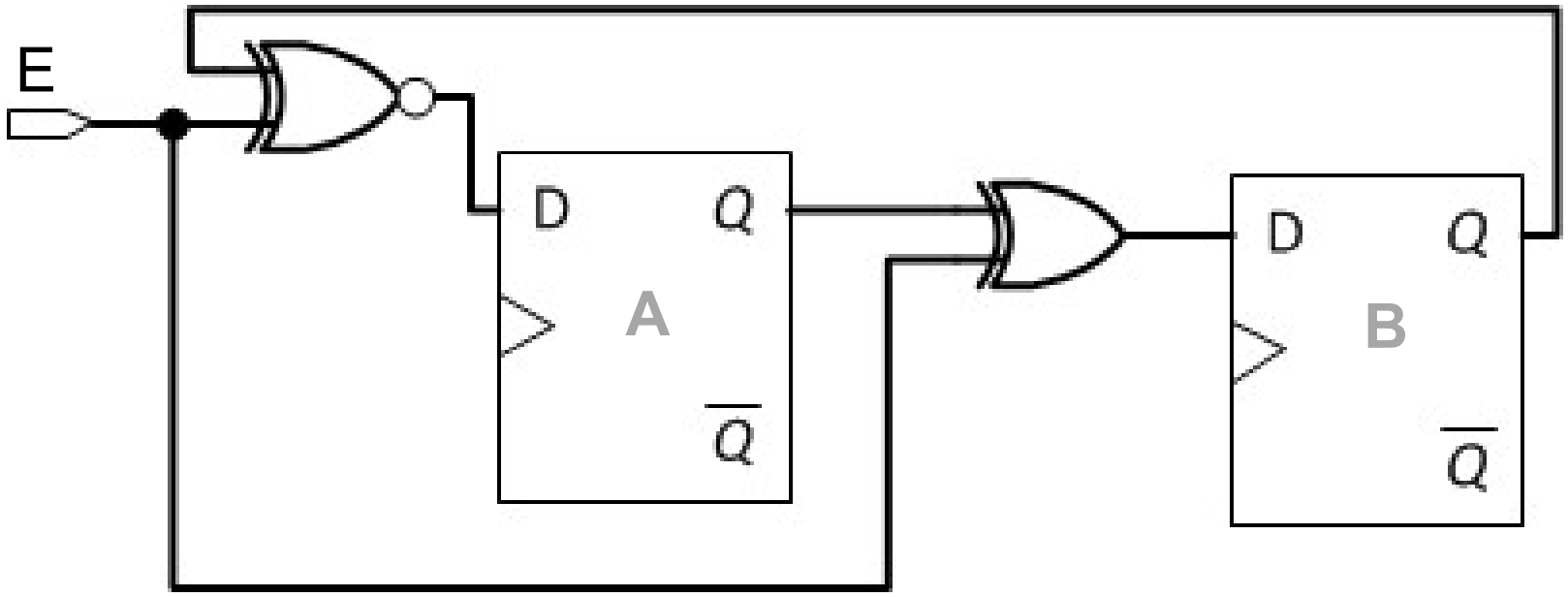




**$D_A$ = E'B' + EB**
**= (E⊕B)'**

**$D_B$ = E'A + EA'**
**= E ⊕ A**

# Contadores Síncronos – Aplicações

- Desenho do Diagrama Lógico do circuito.




$D_A = E'B' + EB$
$= (E \oplus B)'$

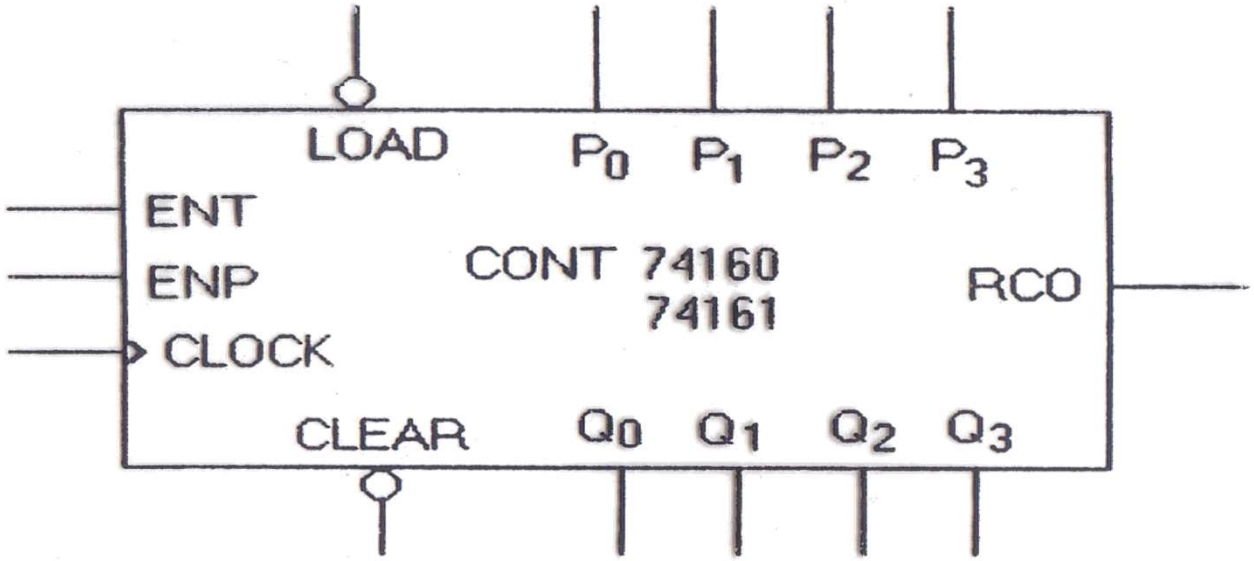
$D_B = E'A + EA'$
$= E \oplus A$

# Exercícios sugeridos

- Projeto de contador hexadecimal síncrono crescente
- Projeto de contador hexadecimal síncrono decrescente
- Projeto de contador octal síncrono crescente ou decrescente dependendo de entrada adicional “dir”
- Projeto de contador octal síncrono seguindo Código de Gray
- Projeto de contador 0-5 síncrono, para uso em display de dezenas de minutos em relógio digital
- Projeto de contador BCD síncrono, para uso em display de minutos em relógio digital

# Contadores Síncronos: Associação em Cascata

- 74160 – Contador módulo 10, sequência BCD;
- 74161 – Contador binário módulo 16 (hexadec.).
  - ENT e ENP: enable
  - RCO: ripple carry out (“vai um”)



# Contadores Síncronos: Associação em Cascata

| ENTRADAS | | | | | FUNÇÃO | EFEITO |
|---|---|---|---|---|---|---|
| $\overline{CLEAR}$ | $\overline{LOAD}$ | ENP | ENT | CLOCK | ASSOCIADA | $Q_0 - Q_3$ |
| 0 | X | X | X | X | ANULA | 0 – 0 |
| 1 | 0 | X | X | ↑ | CARREGA | $P_0 - P_3$ |
| 1 | 1 | 0 | X | X | INIBE | $Q_0 - Q_3$ |
| 1 | 1 | X | 0 | X | INIBE | $Q_0 - Q_3$ |
| 1 | 1 | 1 | 1 | ↑ | CONTA | $(Q_0 - Q_3)_{n+1}$ |

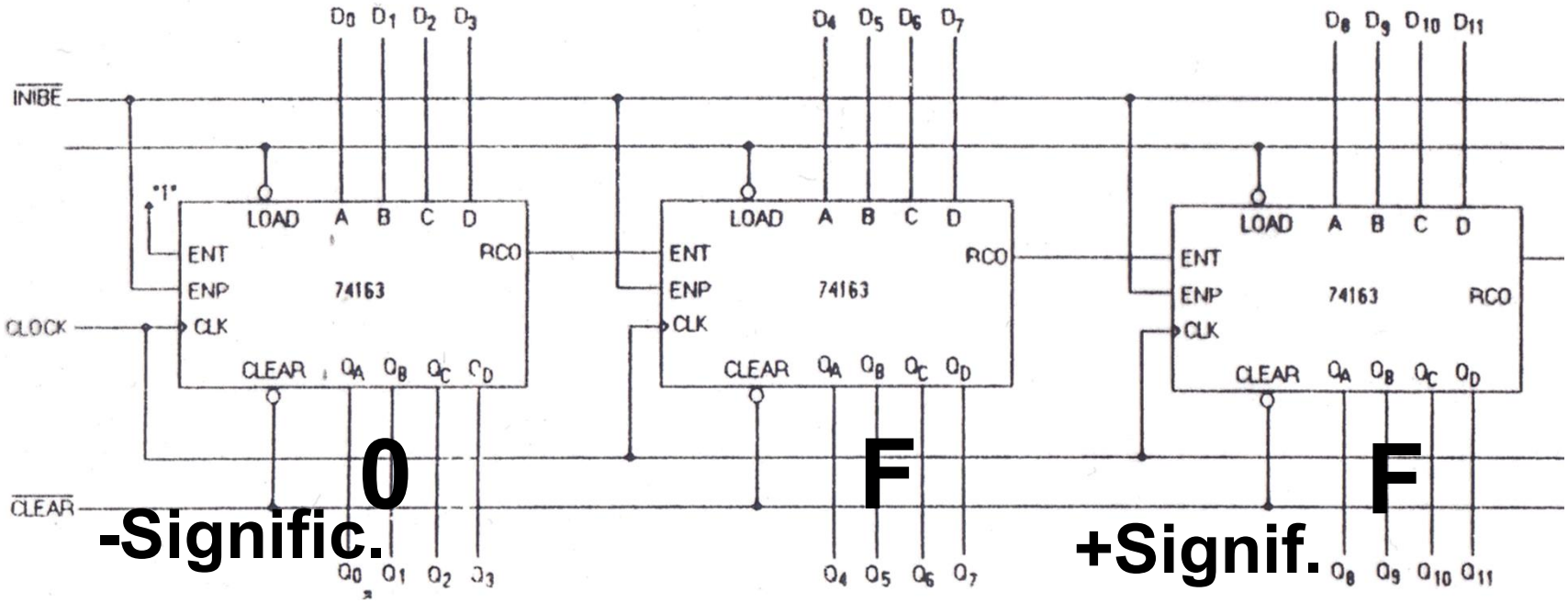
# OBS Nº1:

(i) 7416*: CONTADORES 74160 OU 74161

Decimal: (ii) 74160: $RCO = Q_0 \cdot \overline{Q_1} \cdot \overline{Q_2} \cdot Q_3 \cdot ENT$ **Inibe**

Hexa: (iii) 74161: $RCO = Q_0 \cdot Q_1 \cdot Q_2 \cdot Q_3 \cdot ENT$ **RCO**

# Contadores Síncronos: Associação em Cascata

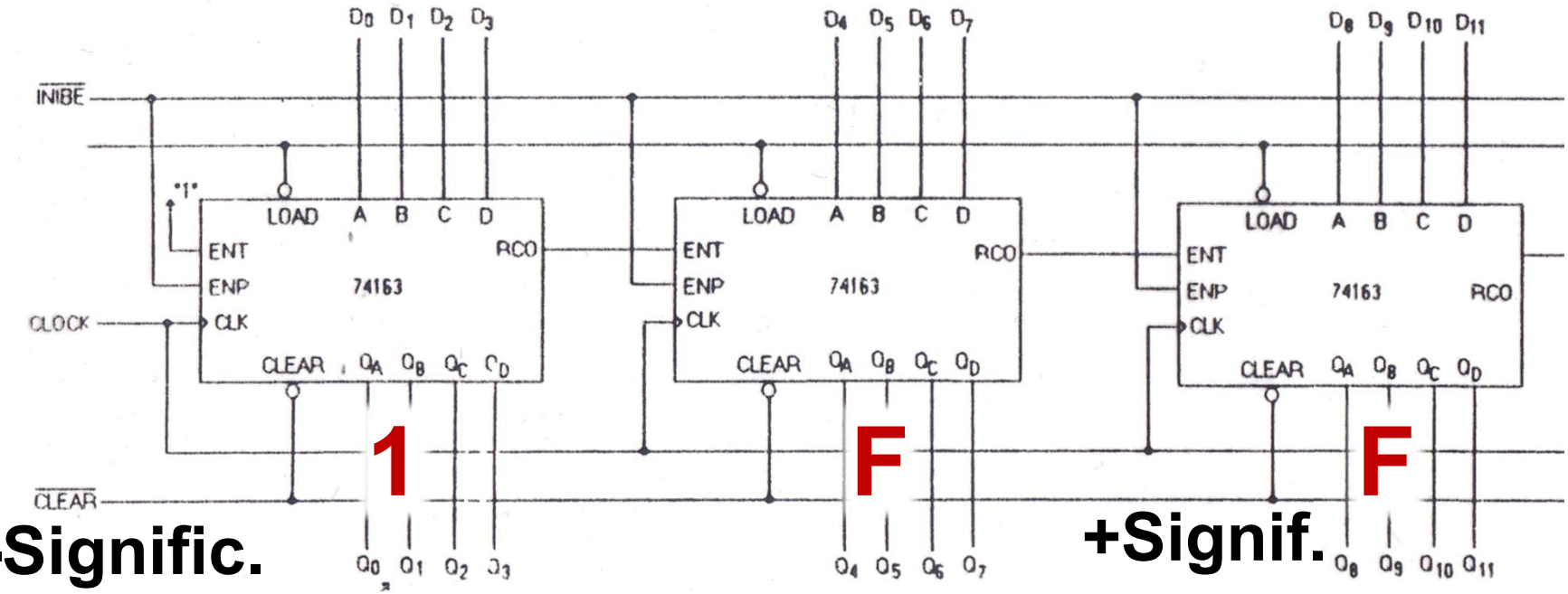



- Se o Estado atual dos contadores fosse:

  (-Sig.)**0FF**(+Sig.)

- Qual seria o Estado após a próxima borda de subida do *clock*?

# Contadores Síncronos: Associação em Cascata




- Resposta: Se o sinal ENT do caractere do meio não desabilitasse seu próprio RCO, o Estado após a próxima borda seria:

  (-Sig.)**1F0**(+Sig.)

- Obs.: o sinal de enable “ENT” deve ser usado, pois o RCO não depende do enable “ENP”

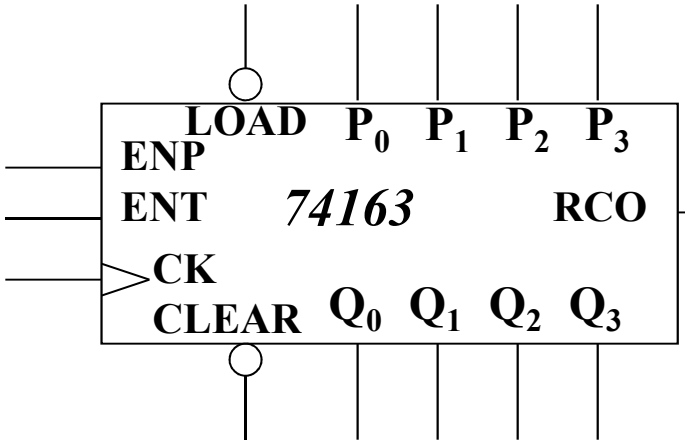
# Contadores Síncronos – Detecção e Carga Síncrona de Estado para Sequência de Módulo ≠ $2^n$

- **Lembrando**: A detecção e **carga assíncrona** de Estado necessitava da detecção do **Estado posterior** ao **último Estado** da **sequência desejada**.

- **Síncrono** – **Detecta-se** o **último Estado** da **sequência desejada**. O contador permanece um **período** de ***clock*** **completo**, **neste Estado**, e, **na próxima borda** do ***clock*** é ***resetado***.

# Contadores Síncronos – Detecção e Carga Síncrona de Estado para Sequência de Módulo ≠ $2^n$

- **Síncrono** – Exemplo: Contador módulo 11 (conta dos Estados 0 até 10) com 74163.

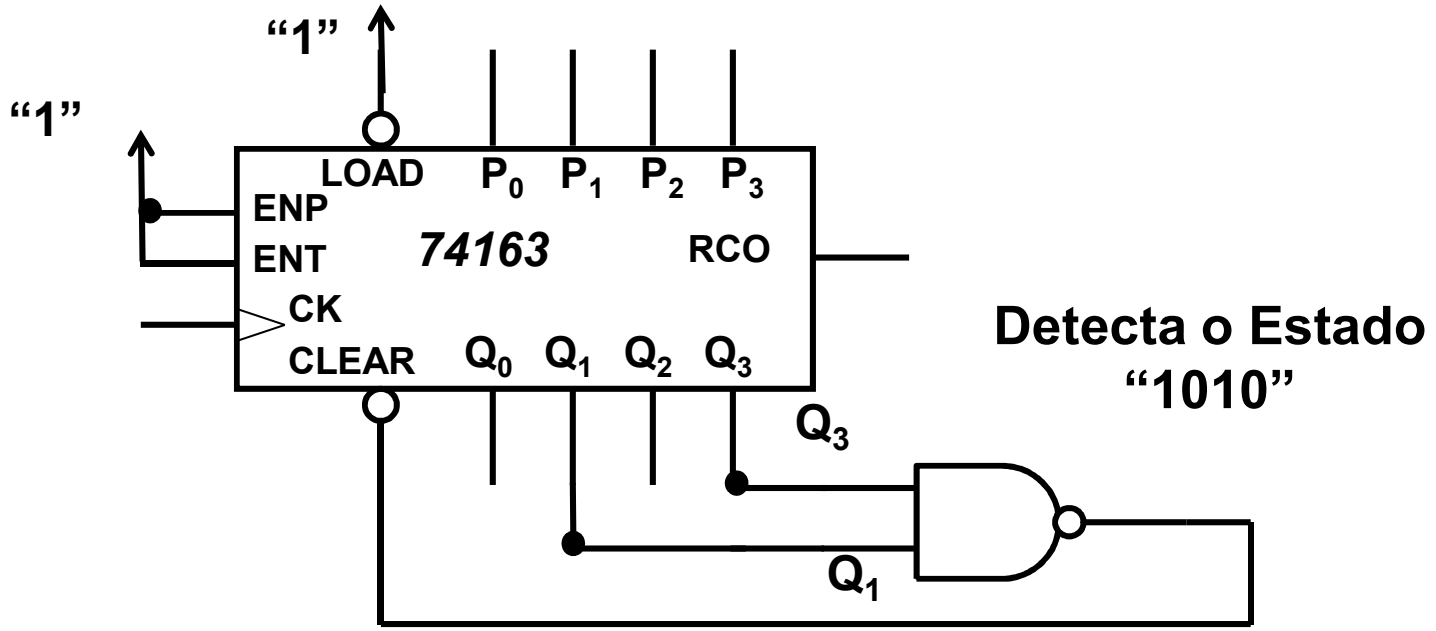



| ENTRADAS | | | | | FUNÇÃO | EFEITO |
|---|---|---|---|---|---|---|
| CLEAR | LOAD | ENP | ENT | CLOCK | ASSOCIADA | $Q_0 - Q_3$ |
| 0 | X | X | X | ↑ | ANULA | $Q_0 - Q_3$ |
| 1 | 0 | X | X | ↑ | CARREGA | $Q_0 - Q_3$ |
| 1 | 1 | 0 | X | X | INIBE | $Q_0 - Q_3$ |
| 1 | 1 | X | 0 | X | INIBE | $Q_0 - Q_3$ |
| 1 | 1 | 1 | 1 | ↑ | CONTA | $(Q_0 - Q_3)^{n+1}$ |

- **1) 74163: Contador módulo 16 (hexadecimal), sequência binária, com clear síncrono**
- **2) RCO: "Ripple Carry Out";**
- **3) ENT: "Enable Trickle Input";**
- **4) ENP: "Enable Parallel Input";**
- **5) RCO = Q0 . Q1 . Q2 . Q3 . ENT**

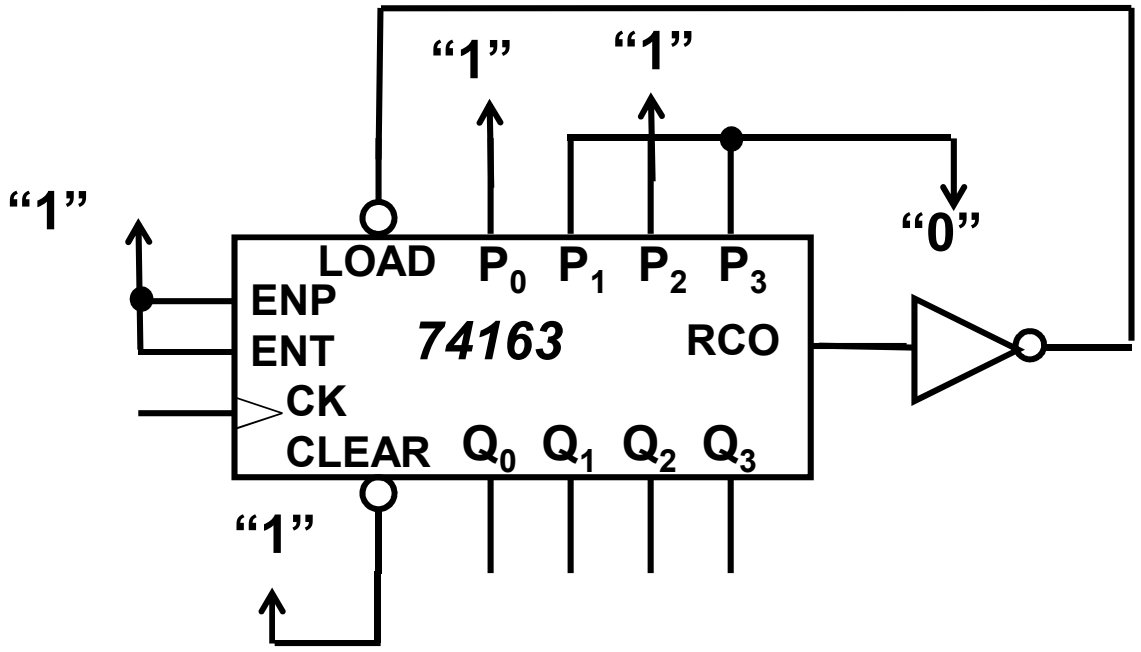
# Contadores Síncronos – Detecção e Carga Síncrona de Estado para Sequência de Módulo ≠ $2^n$

- **Síncrono** – Exemplo: Contador módulo 11 (conta dos Estados 0 até 10) com 74163.

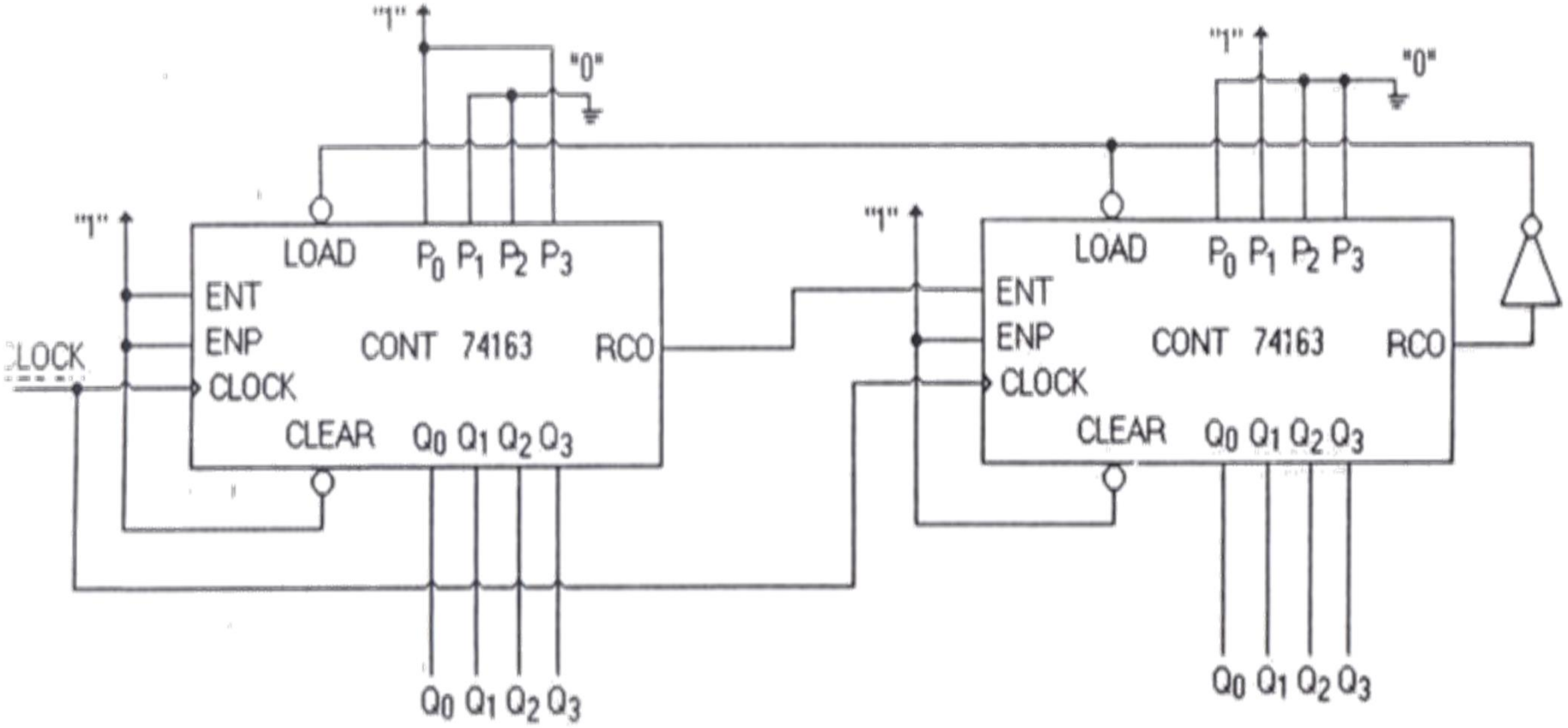


# Contadores Síncronos – Detecção e Carga Síncrona de Estado para Sequência de Módulo ≠ $2^n$

- **Alternativa** – Contador **módulo 11**
  - Conta dos Estados **5 até 15;**
  - Usa o sinal de **RCO** para detectar o Estado **15**, ligando-o (invertido) ao **LOAD** para carregar o Estado **5**.



- **Exercício** – Qual o módulo deste contador?

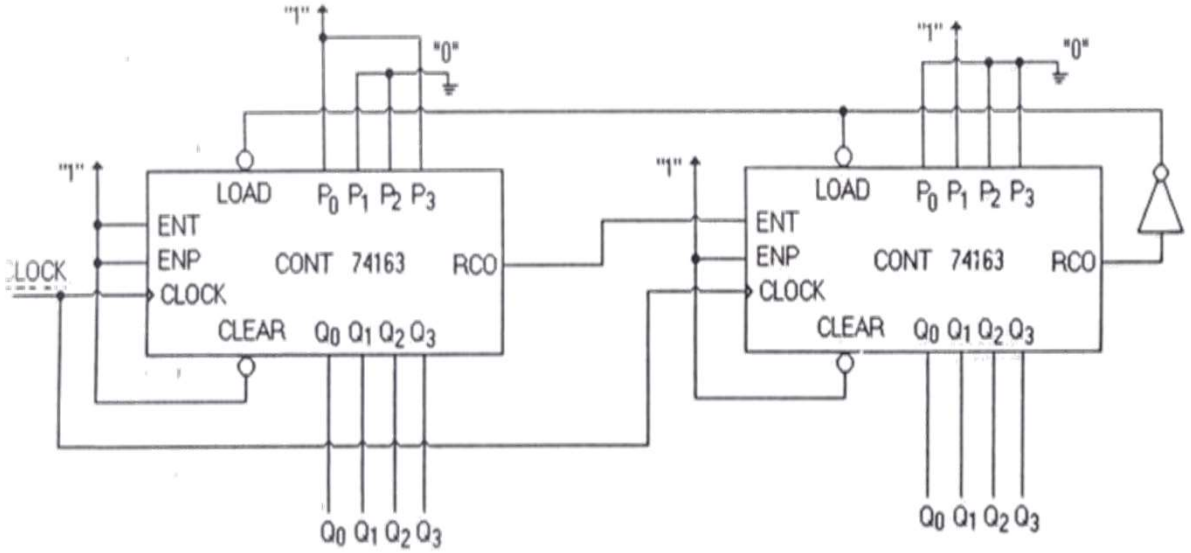


# Contadores Síncronos – Detecção e Carga Síncrona de Estado para Sequência de Módulo ≠ $2^n$

- **Exercício** – Qual o módulo deste contador?

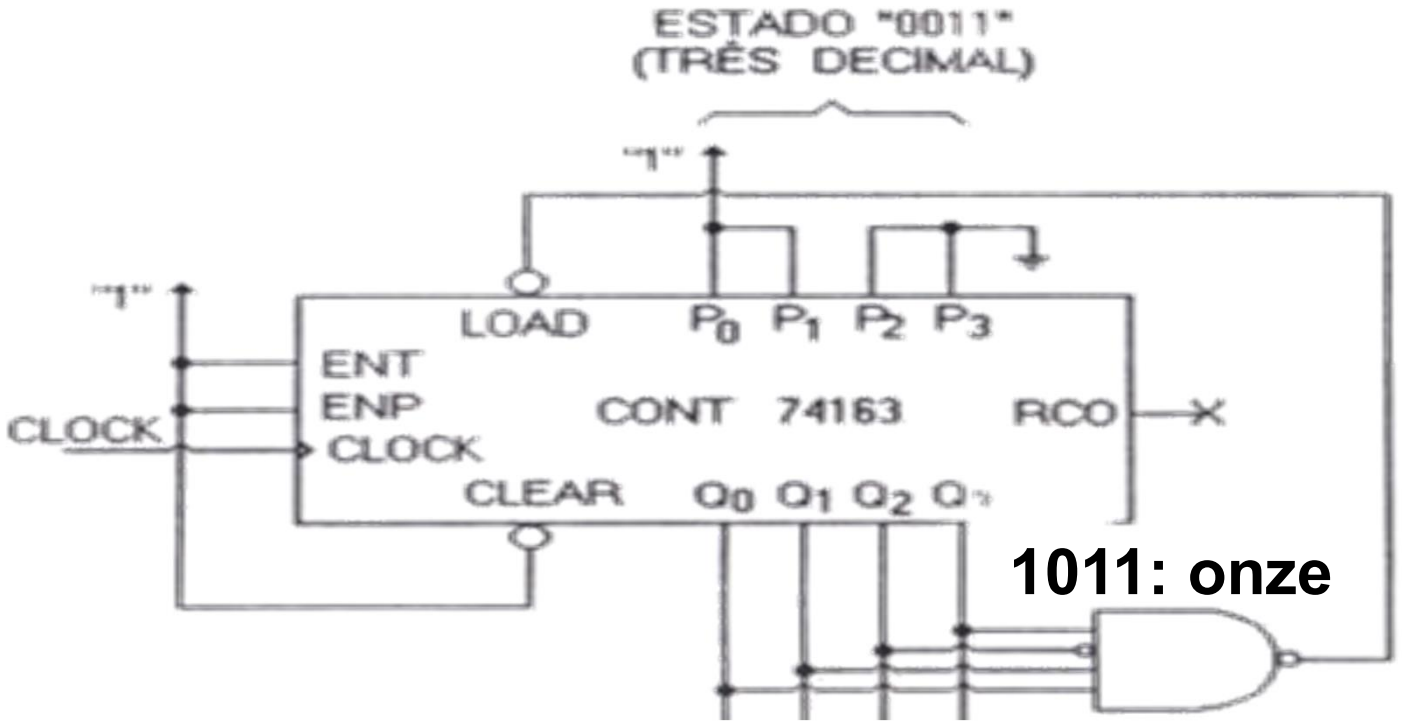
| | C1 | C2 | clocks |
|---|---|---|---|
| Reset | 9 | 2 | +7 |
| | 0 | 3 | +16 |
| | 0 | 4 | +16 |
| | … | … | +16 |
| $RCO_2 = 1 \bullet ENT$ | 0 | F | +16 |

**Total:**
**7+13*16**



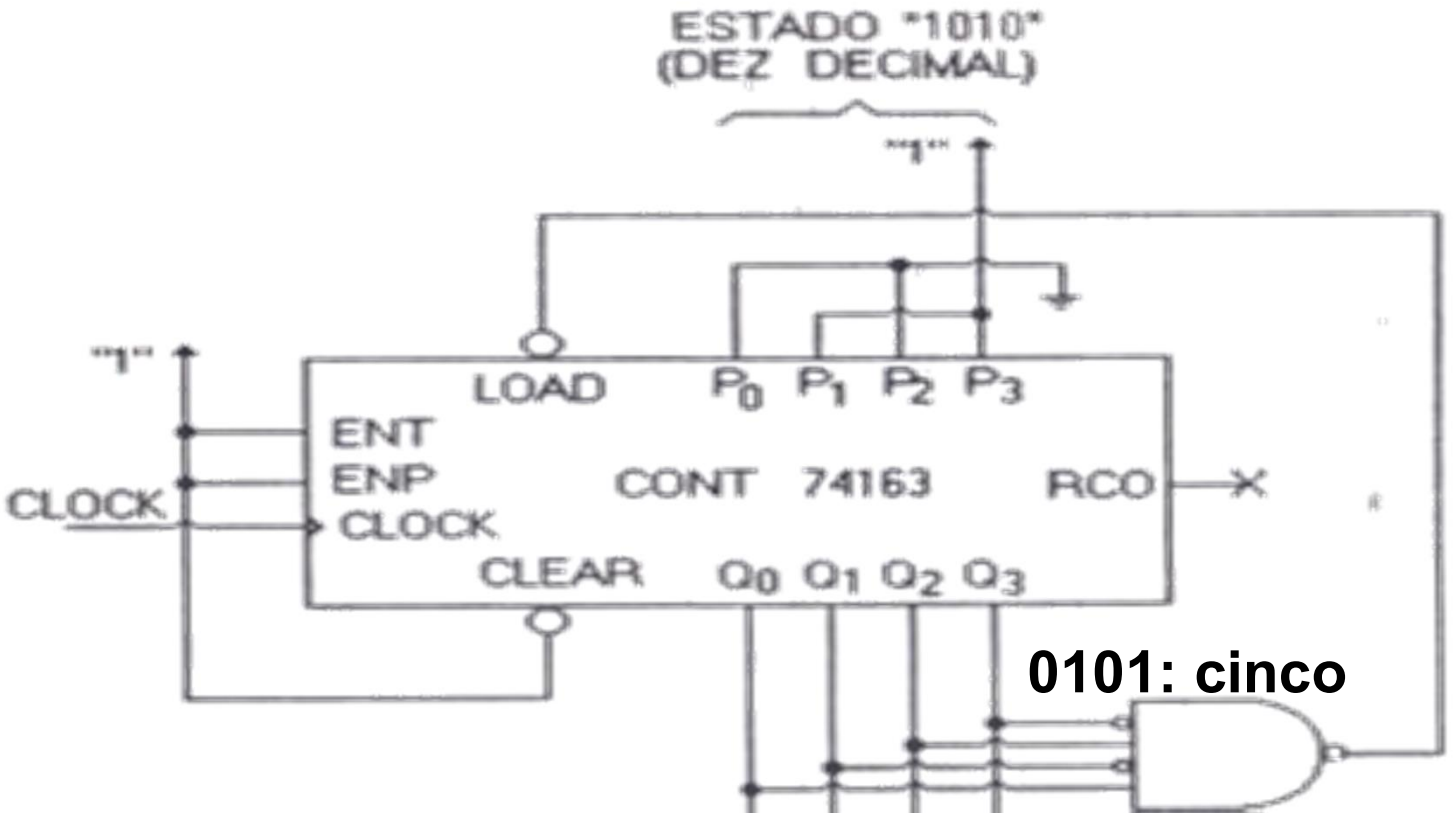
# Contadores Síncronos – Detecção e Carga Síncrona de Estado para Sequência de Módulo ≠ $2^n$

- **Exercício** – Qual o módulo deste contador? Qual a sequência de contagem?
  - **Resposta**: intervalo [3 , 11] ➔ módulo 9



# Contadores Síncronos – Detecção e Carga Síncrona de Estado para Sequência de Módulo ≠ $2^n$

- **Exercício** – Qual o módulo deste contador? Qual a sequência de contagem?
  - **Resposta**: intervalo [10, 15] + [0, 5] ➔ módulo 12



# Contadores Síncronos – Geração de Onda Quadrada para Sequências de Módulo ≠ $2^n$

- Divisão de Frequência por x = 2n: “pulando” alguns estados escolhidos
  - Exemplo: divisão por 6

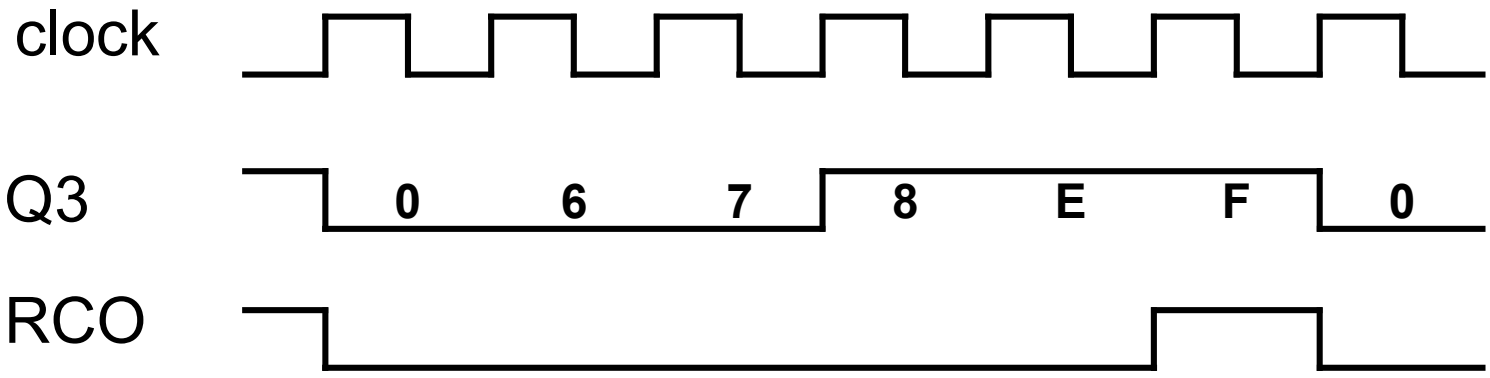
| | Q3 | Q2 | Q1 | Q0 | Estado | Ação | Pulou |
|---|---|---|---|---|---|---|---|
| 3 zeros | 0 | 0 | 0 | 0 | 0 | carrega 6 | 1,2,3,4,5 |
| | 0 | 1 | 1 | 0 | 6 | conta | |
| | 0 | 1 | 1 | 1 | 7 | conta | |
| 3 uns | 1 | 0 | 0 | 0 | 8 | carrega 14 | 9, A, B, C, D |
| | 1 | 1 | 1 | 0 | E | conta | |
| | 1 | 1 | 1 | 1 | F | conta | |

**Carga: Q3,1,0,0**

# Contadores Síncronos – Geração de Onda Quadrada para Sequências de Módulo ≠ $2^n$

- **Divisão de Frequência por 6**

| Q3 | Q2 | Q1 | Q0 | Estado | Ação | Pulou |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | carrega 6 | 1,2,3,4,5 |
| 0 | 1 | 1 | 0 | 6 | conta | |
| 0 | 1 | 1 | 1 | 7 | conta | |
| 1 | 0 | 0 | 0 | 8 | carrega 14 | 9, A, B, C, D |
| 1 | 1 | 1 | 0 | E | conta | |
| 1 | 1 | 1 | 1 | F | conta | |



# Contadores Síncronos – Geração de Onda Quadrada para Sequências de Módulo ≠ $2^n$

- **Divisão de Frequência por 6**