

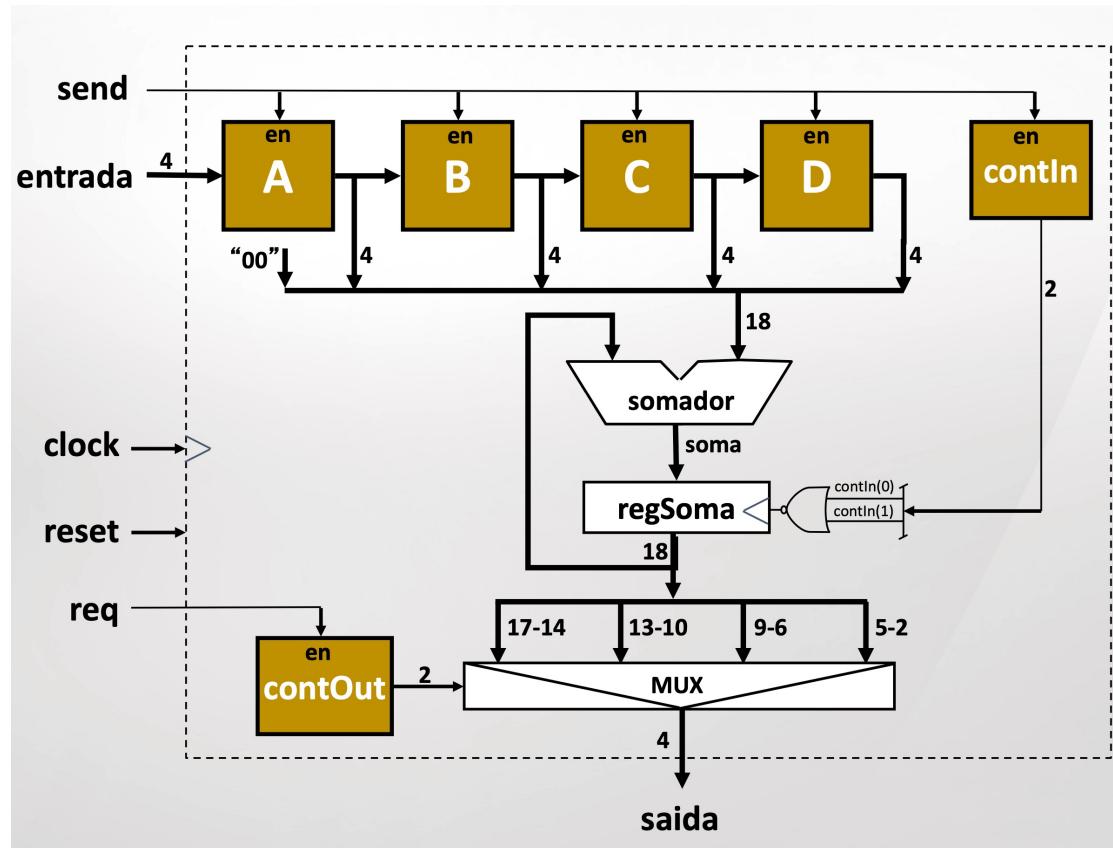
Fundamentos de Sistemas Digitais

CIRCUITOS SEQUENCIAIS

Prof. Fernando G. Moraes

Planejamento

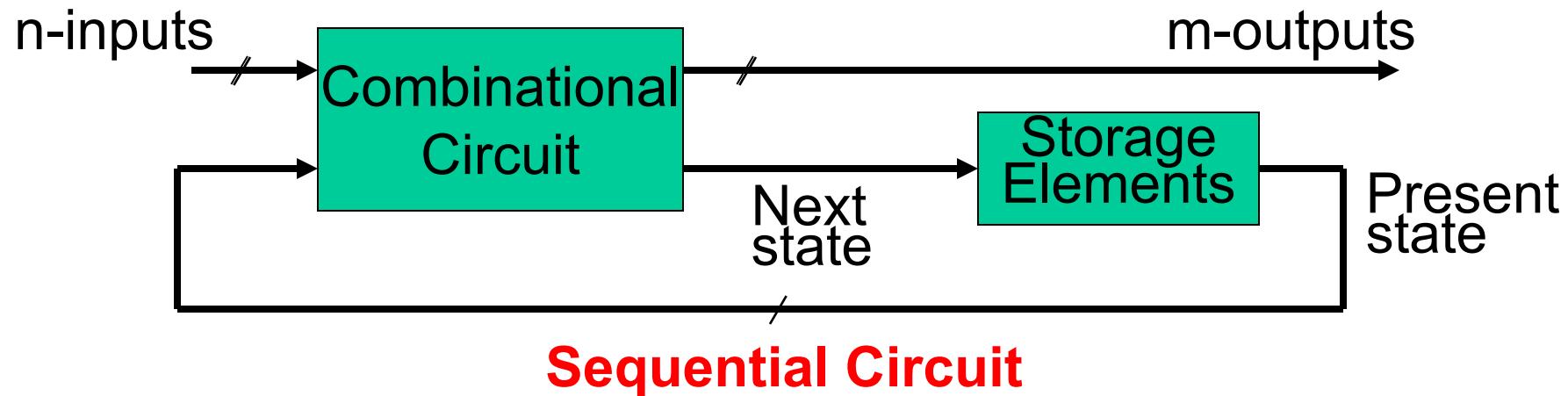
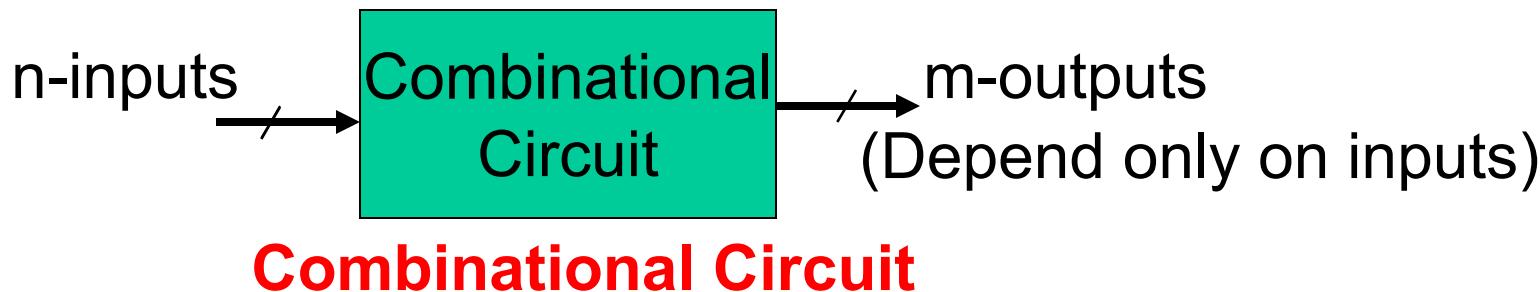
- Aula 17 – 29/04 - Flip-flops: latch e mestre-escravo
- Aula 18 – 04/05 - Registradores
- Aula 19 – 06/05 – Projeto RTL – transferência de dados entre registradores
- Aula 20 – 11/05 – Projeto RTL



PARTE I

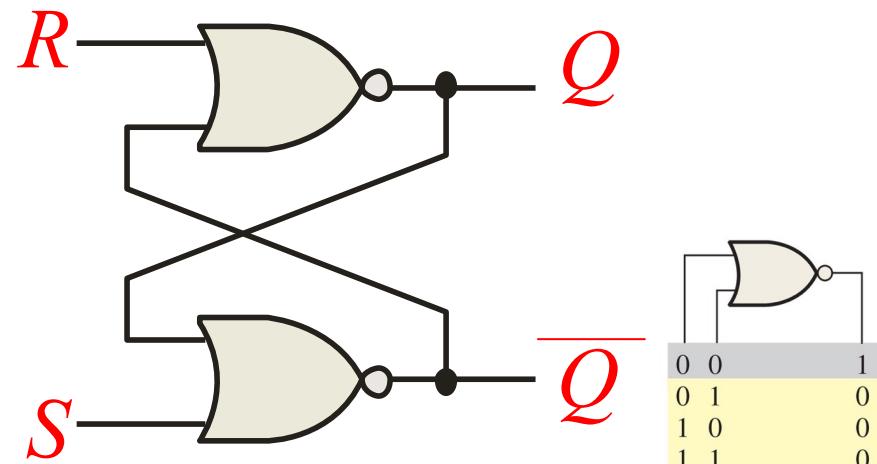
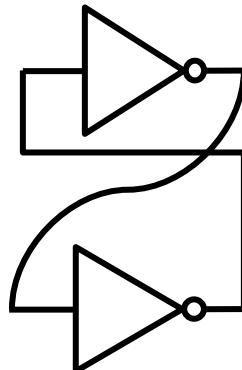
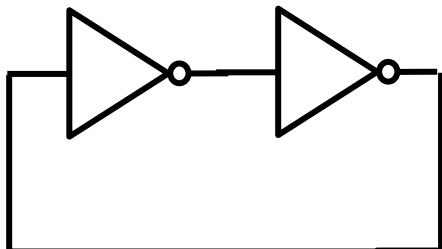
- Latch
- FF Mestre-Escravo

Circuitos Combinacionais versus Sequenciais



LATCH

- Um **latch** é um dispositivo de armazenamento de dados
- É a forma básica de memória
- A **latch S-R** (Set-Reset) é o tipo mais básico
 - Pode ser construído a partir de portas NOR ou NAND
 - Com portas **NOR**, a latch responde a entradas **altas**
 - Com portas **NAND**, a latch responde a entradas **baixa**



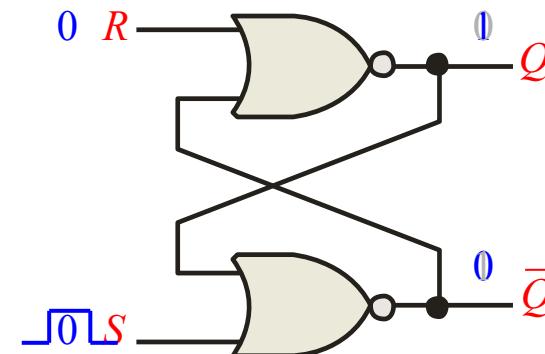
Notar: uma NOR com uma dada entrada em '0' comporta-se como um inversor. 5

LATCH - NOR

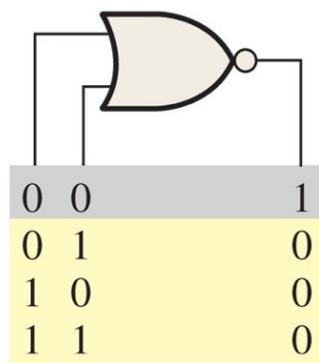
- O **latch** com NOR está em uma condição estável (dado armazenado) quando ambas as entradas estão no nível lógico ZERO

Assumir inicialmente **S** e **R** em '0'.

Para configurar $Q = 1$, um sinal ALTO é aplicado à entrada S enquanto o R permanece BAIXO.



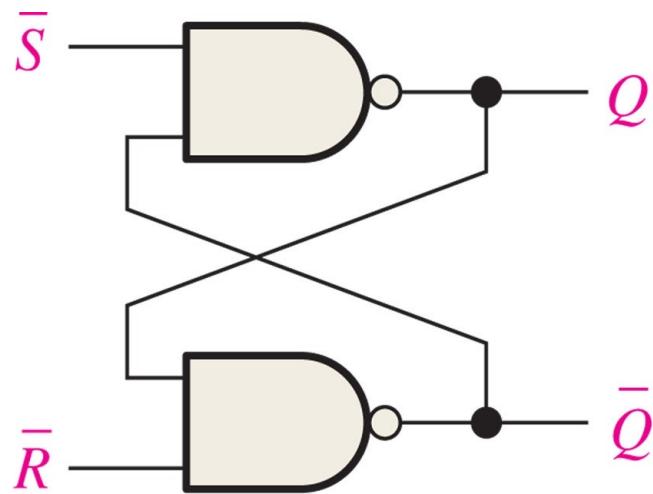
Para RESET, um sinal ALTO é aplicado à entrada R enquanto o S permanece BAIXO.



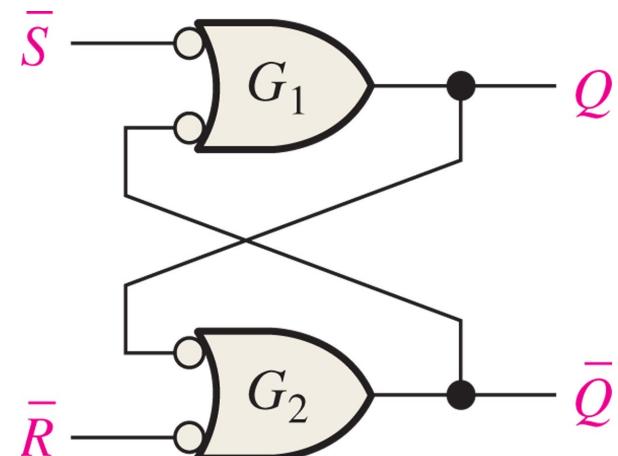
Notar: uma NOR com uma dada entrada em '0' comporta-se como um inversor.

LATCH - NAND

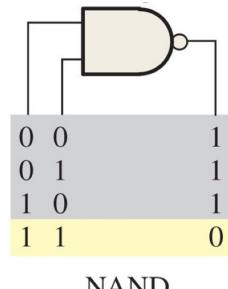
- O **latch** com NAND está em uma condição estável (dado armazenado) quando ambas as entradas estão no nível lógico UM



OU



(b) Active-LOW input \bar{S} - \bar{R} latch



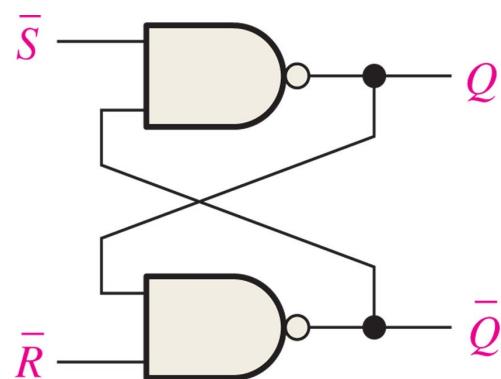
Notar: uma NAND com uma dada entrada em '1' comporta-se como um inversor.

LATCH - NAND

TABLE 7-1

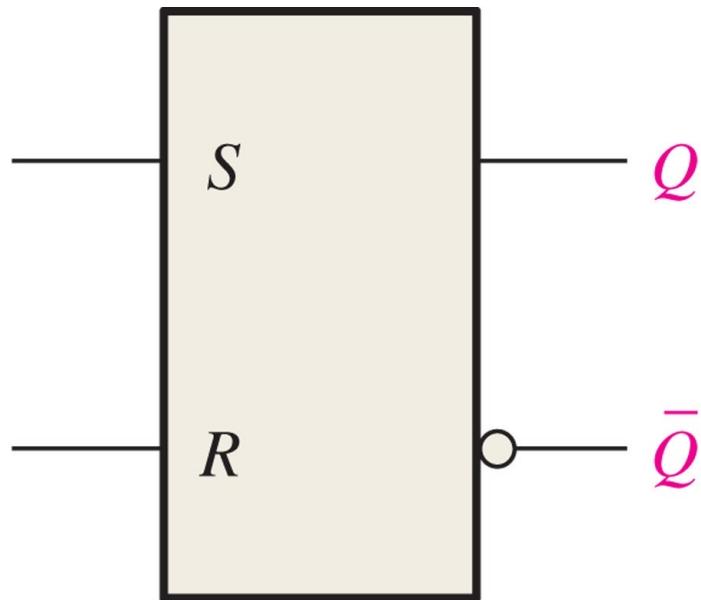
Truth table for an active-LOW input \bar{S} - \bar{R} latch.

Inputs		Outputs		Comments
\bar{S}	\bar{R}	Q	\bar{Q}	
1	1	NC	NC	No change. Latch remains in present state.
0	1	1	0	Latch SET.
1	0	0	1	Latch RESET.
0	0	1	1	Invalid condition

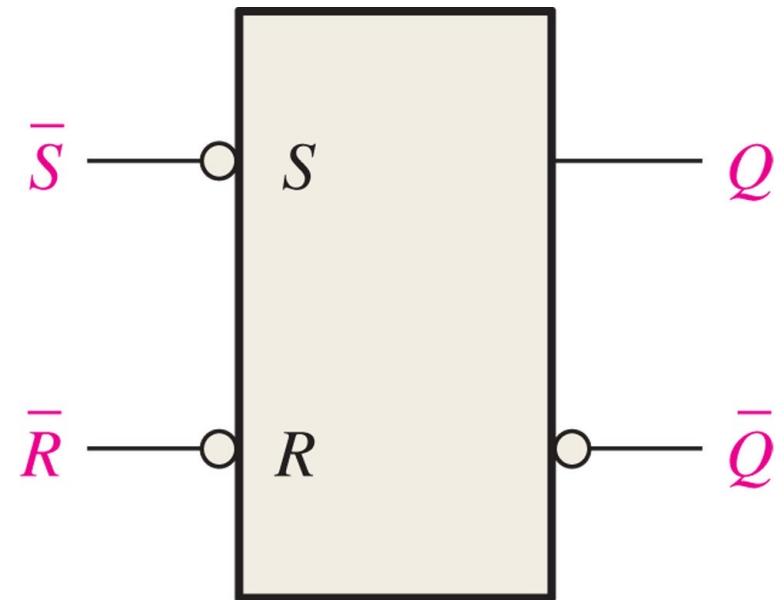


(b) Active-LOW input \bar{S} - \bar{R} latch

Símbolos lógicos para as latches



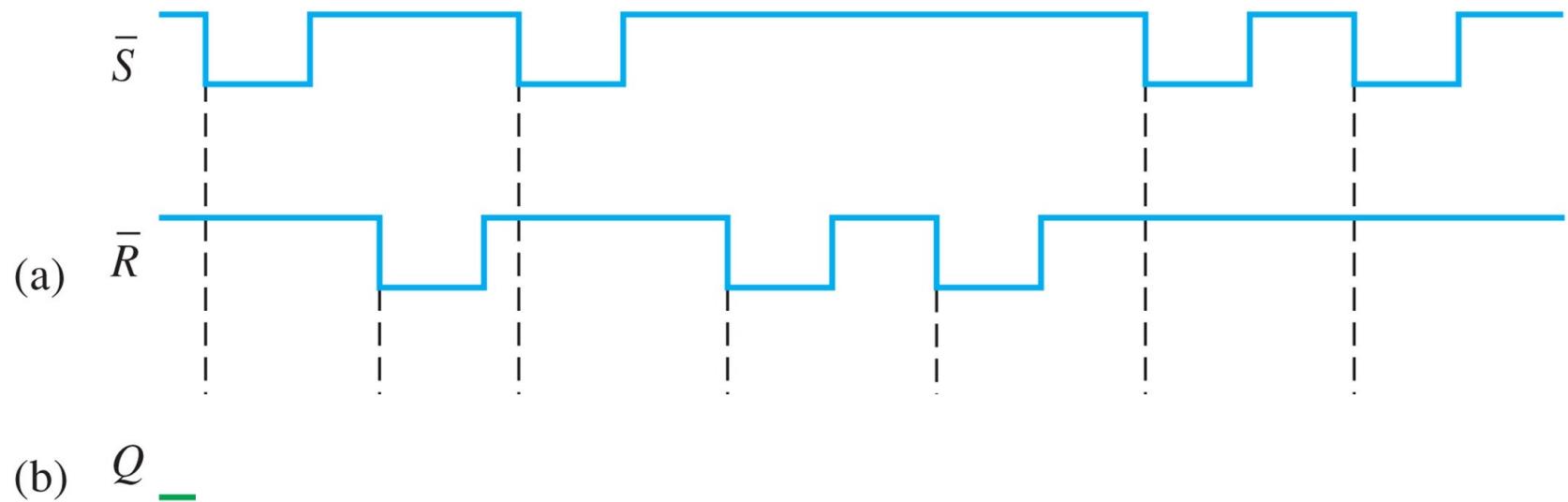
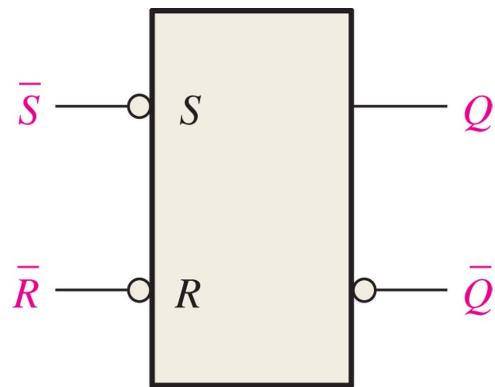
(a) Active-HIGH input
S-R latch



(b) Active-LOW input
 \bar{S} - \bar{R} latch

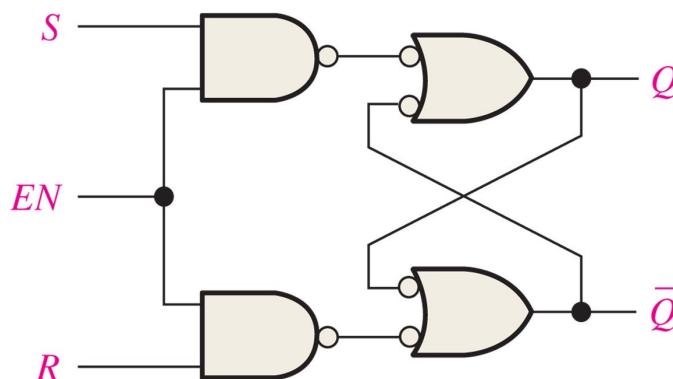
FIGURE 7-4

Comportamento da latch

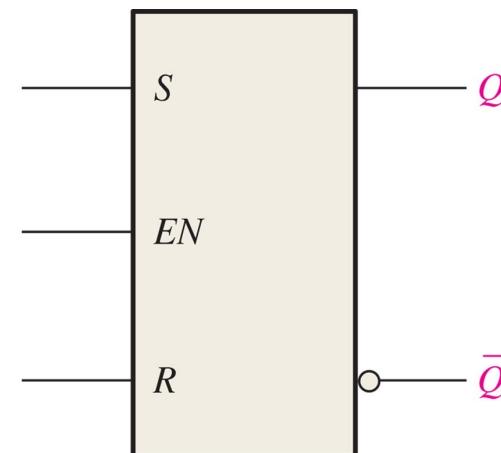


Latch com sinal de habilitação

- Esta latch possui uma entrada adicional, chamada de **habilitação (EN)** que deve estar em nível ALTO, para que a latch responda às entradas S e R

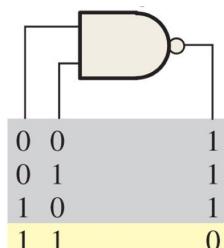


(a) Logic diagram



(b) Logic symbol

FIGURE 7-8 A gated S-R latch.



NAND

Uma NAND com uma dada entrada em '0' tem em sua saída '1' → logo S/R iguais a '1' impedindo alterações na saída Q/nQ

Latch com sinal de habilitação

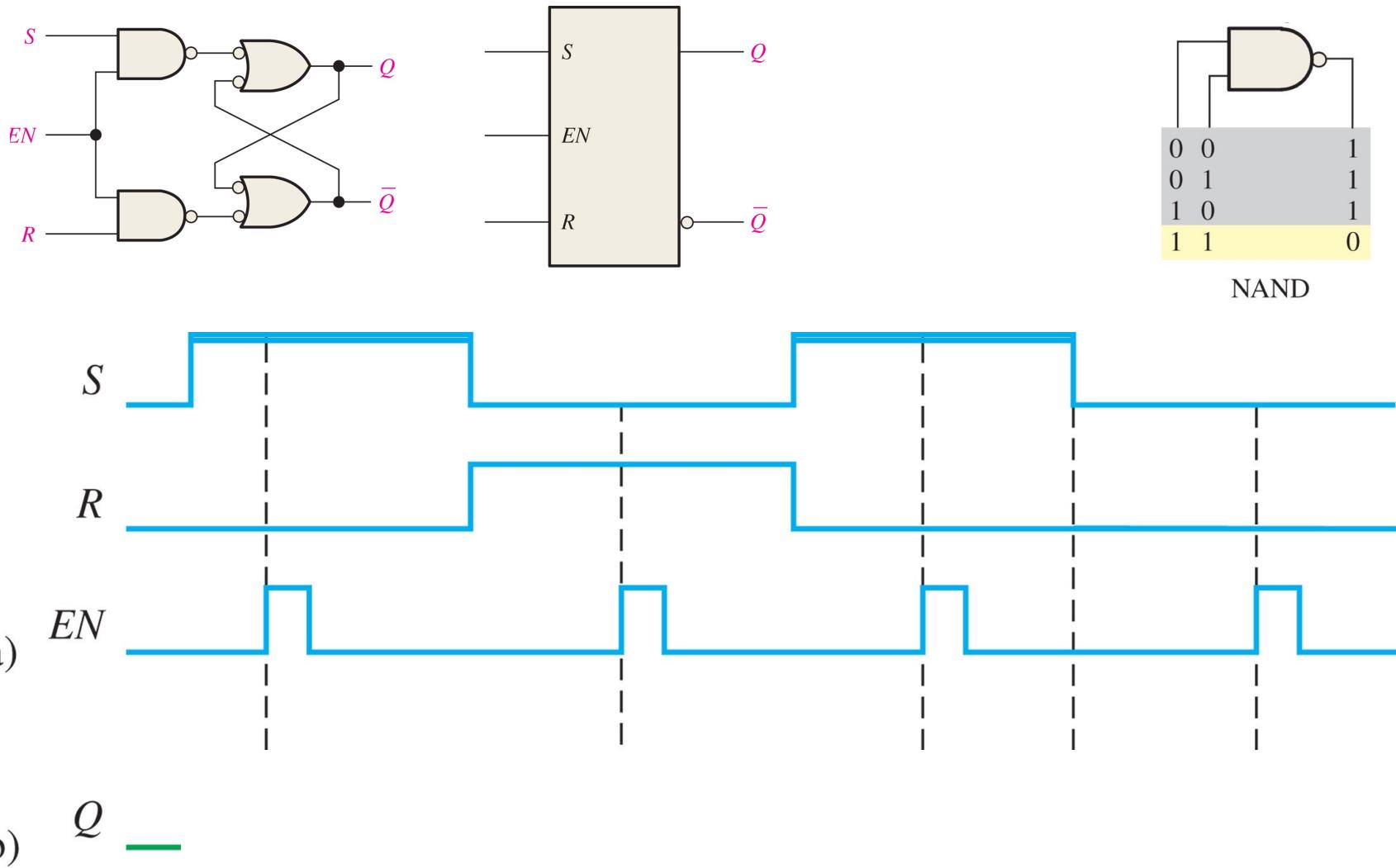
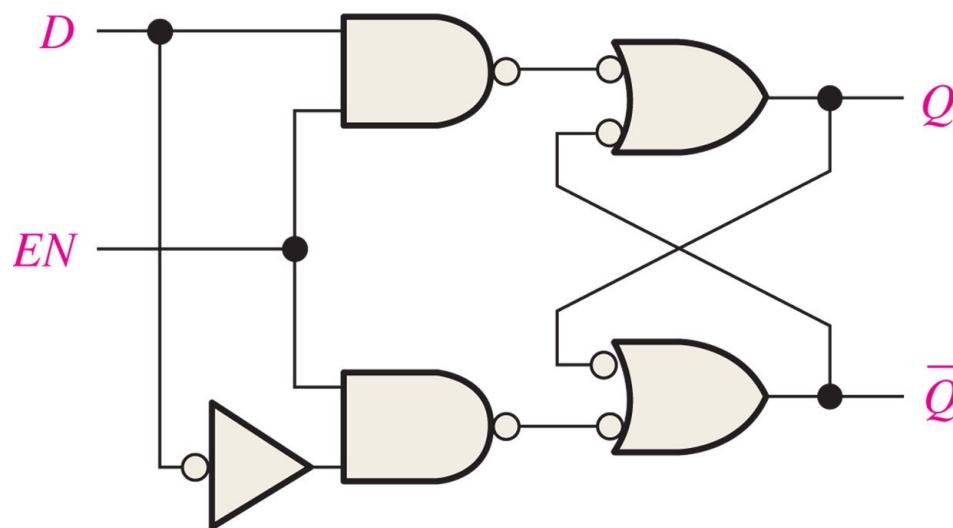


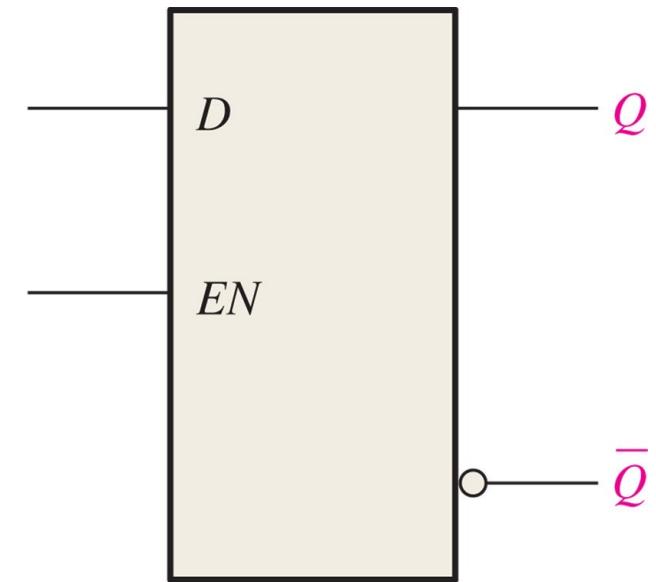
FIGURE 7-9

Latch D com sinal de habilitação

- A **latch D** é uma variação do latch S-R, combinando as entradas S e R em uma única entrada D



(a) Logic diagram



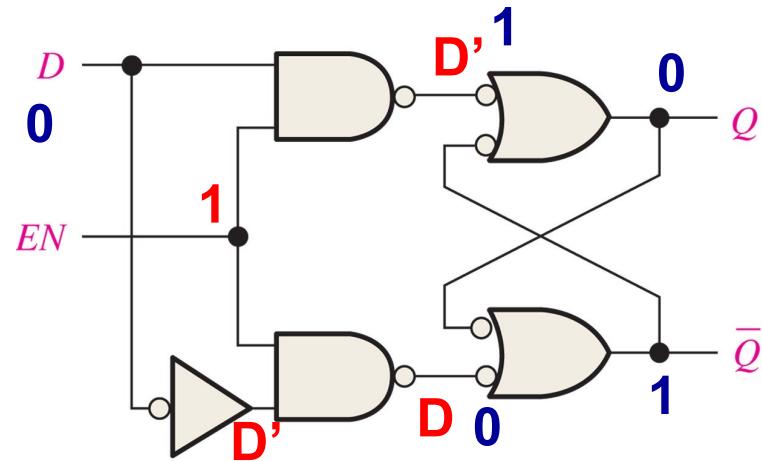
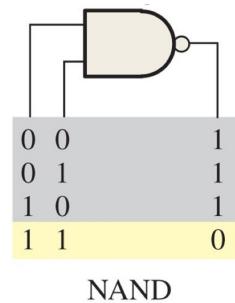
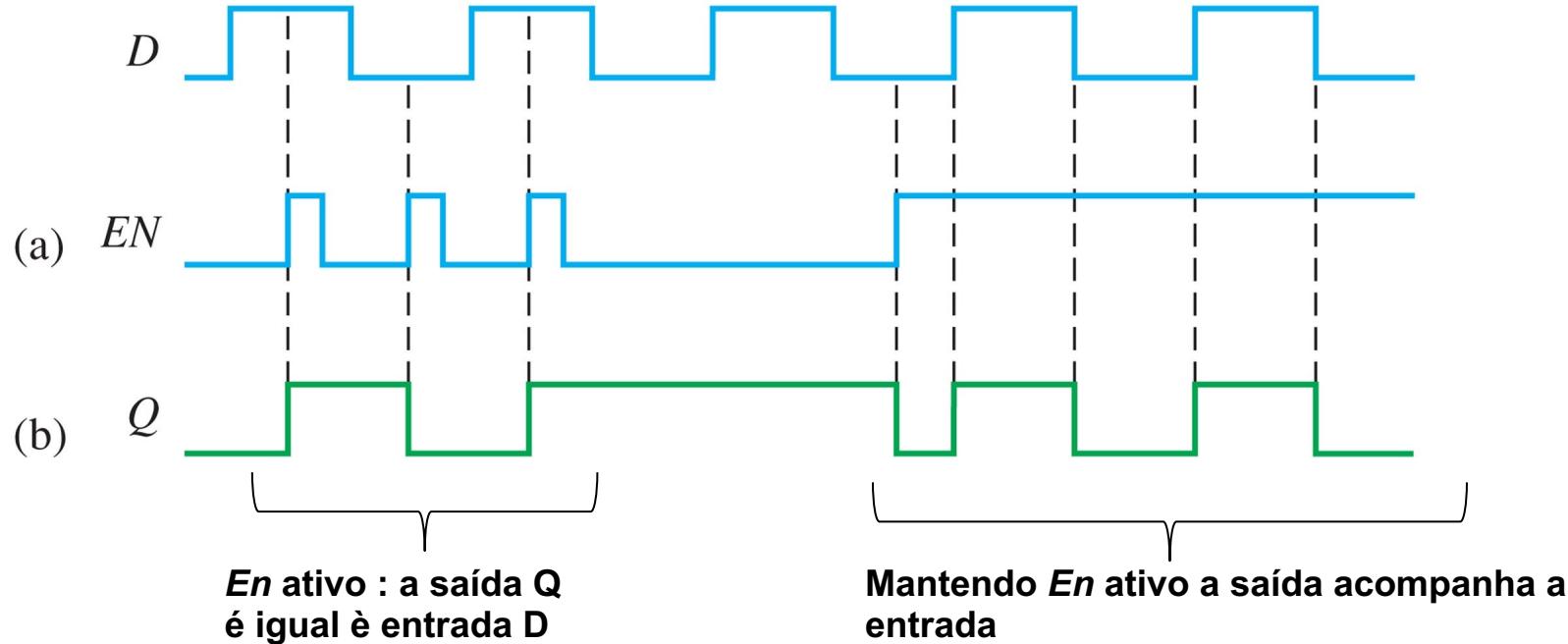
(b) Logic symbol

FIGURE 7-10 A gated D latch.

Uma regra simples para a latch D:
→ Q segue D quando o EN está ativo

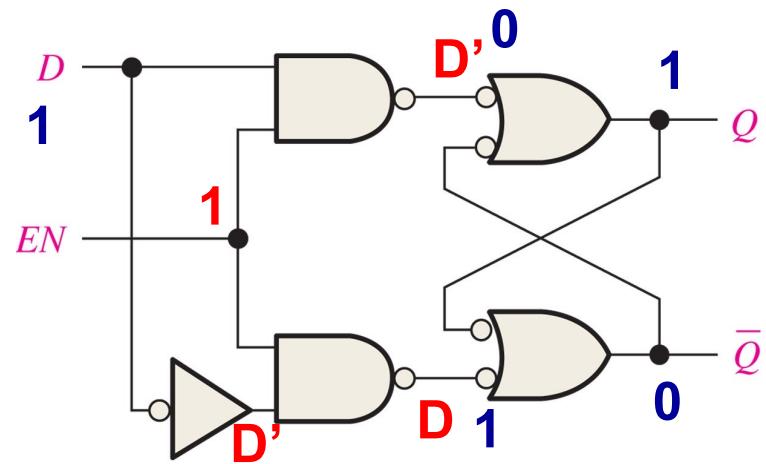
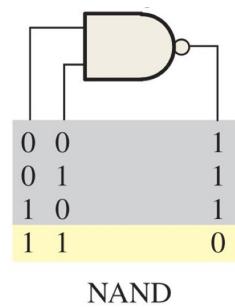
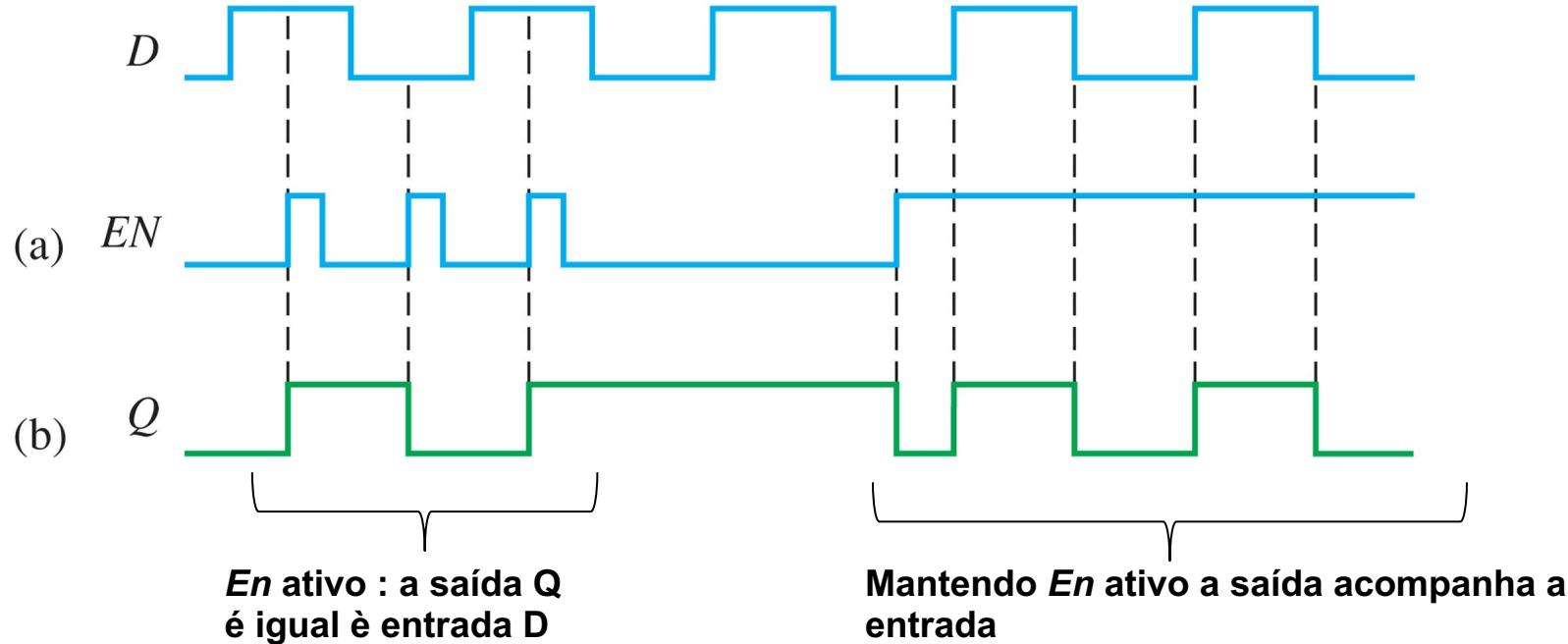
Latch D - comportamento

FIGURE 7-11



Latch D - comportamento

FIGURE 7-11



Latch D – tabela verdade

A tabela de verdade para a latch D é apresentada abaixo

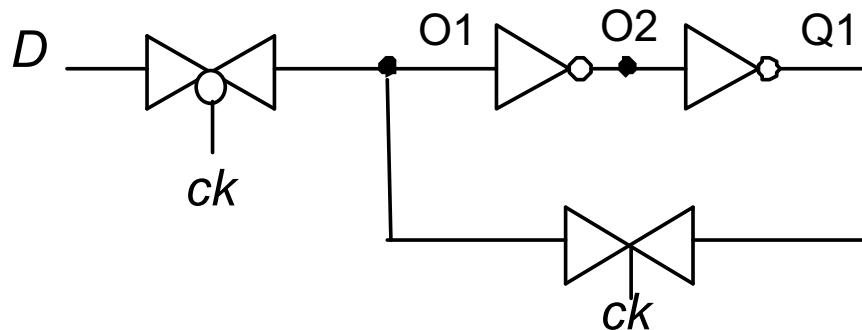
- se EN=1, a saída é igual à entrada D
- se EN=0 não há alteração na saída, e o dado está armazenado

Inputs		Outputs		Comments
D	EN	Q	\bar{Q}	
0	1	0	1	RESET
1	1	1	0	SET
X	0	Q_0	\bar{Q}_0	No change

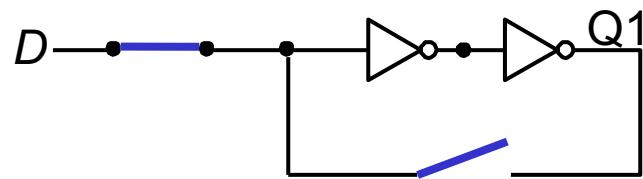
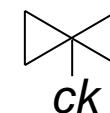
Note: Q_0 is the prior output level before the indicated input conditions were established.

Latch D – uma outra implementação

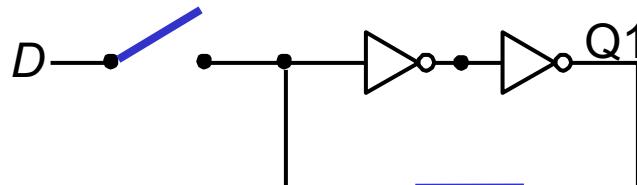
Latch D → bit de memória para caches, por exemplo



2 Inversores em série controlados por chaves



$CK=0 \rightarrow$ saída igual à entrada



$CK=1 \rightarrow$ mantém armazenado o último D lido

Flip-flop D Mestre-Escravo (MS)

- Um flip-flop MS difere de uma latch na forma como ele muda de estados
- Um flip-flop é um dispositivo no qual apenas a **borda** do relógio determina quando um novo bit é armazenado
- A sensibilidade à borda pode ser positiva ou negativa

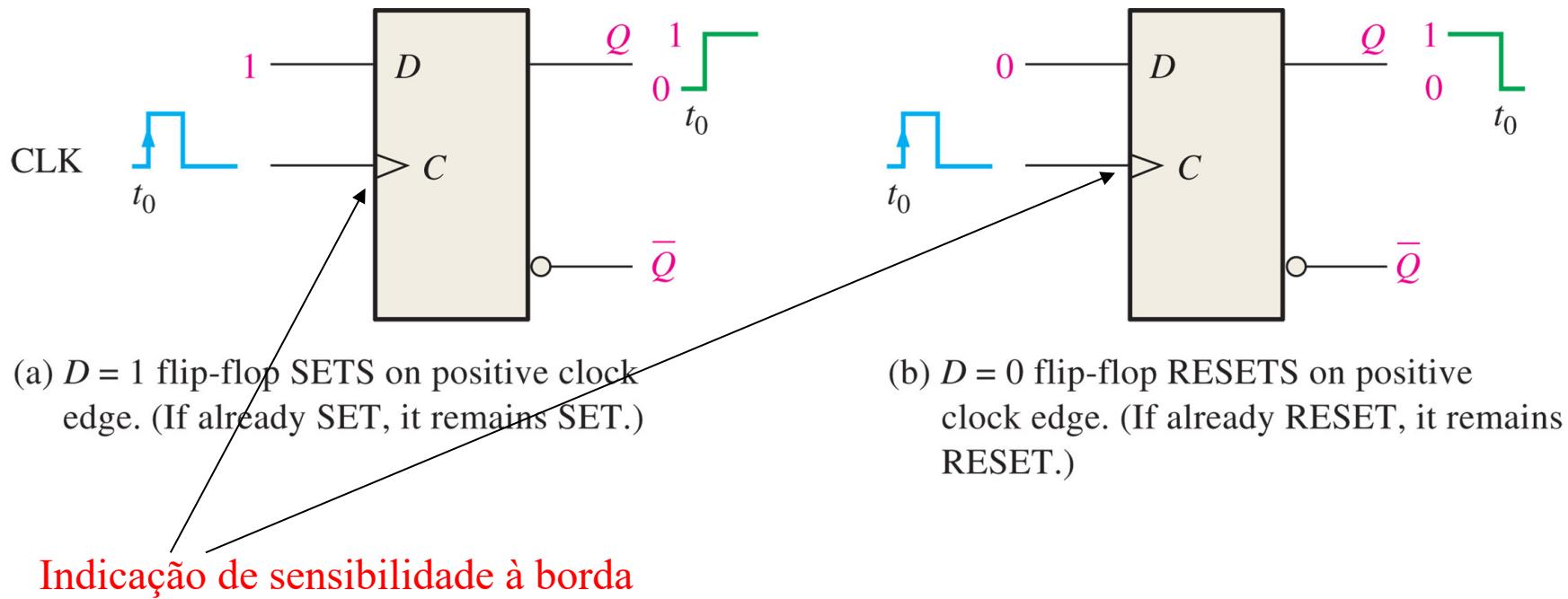


FIGURE 7-14 Operation of a positive edge-triggered D flip-flop.

FFD-MS: Comportamento

TABLE 7-2

Truth table for a positive edge-triggered D flip-flop.

Inputs		Outputs		Comments
D	CLK	Q	\bar{Q}	
0	↑	0	1	RESET
1	↑	1	0	SET

↑ = clock transition LOW to HIGH

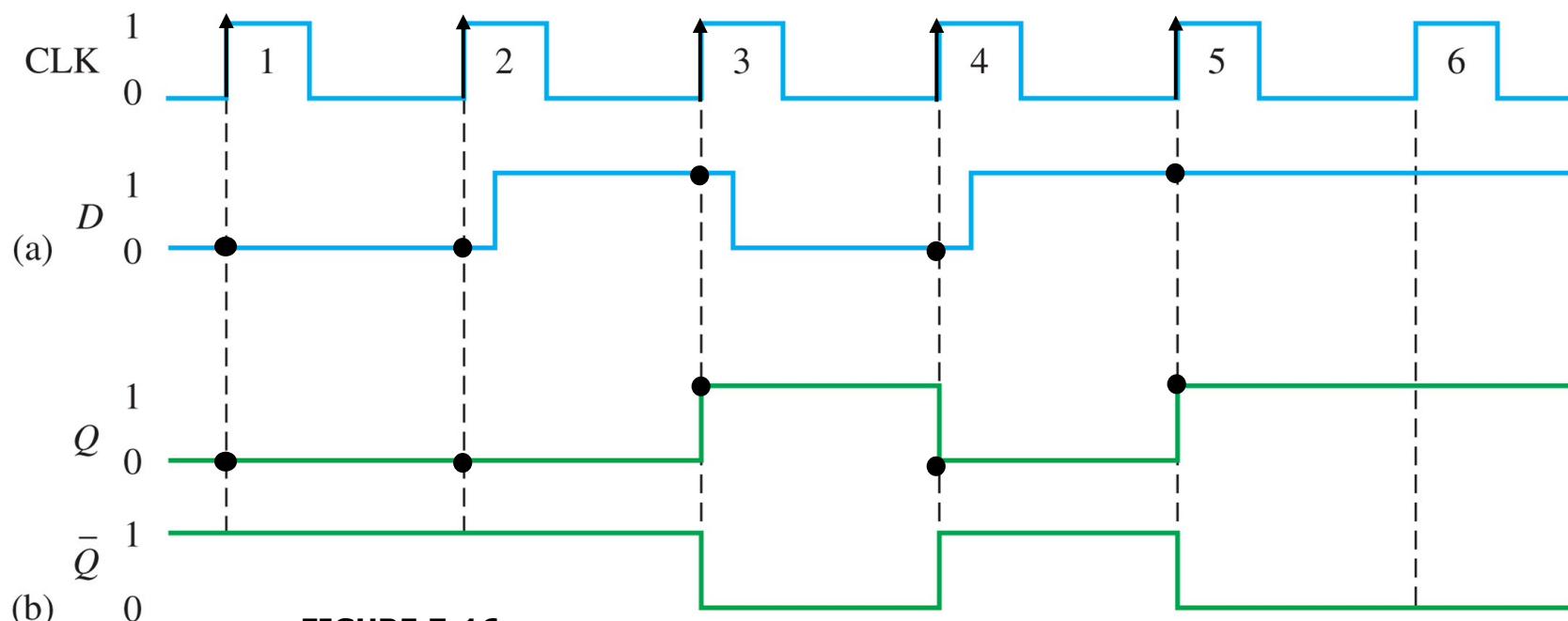
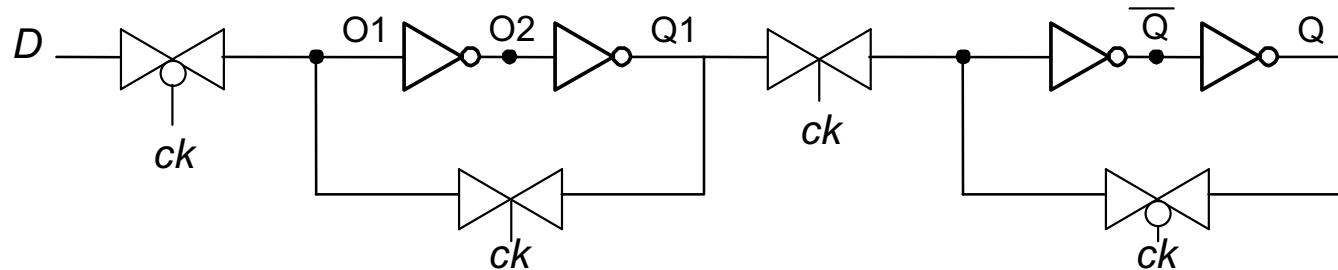


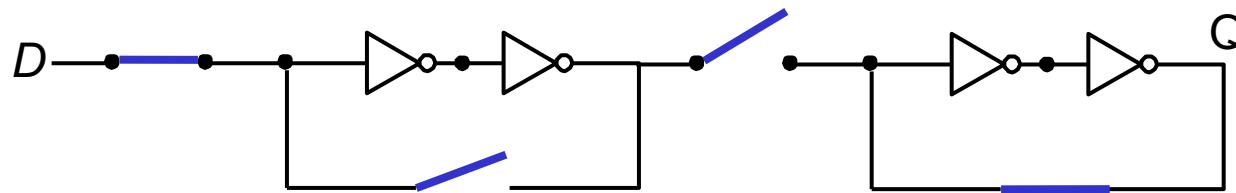
FIGURE 7-16

FFD – Mestre Escravo (implementação com chaves)

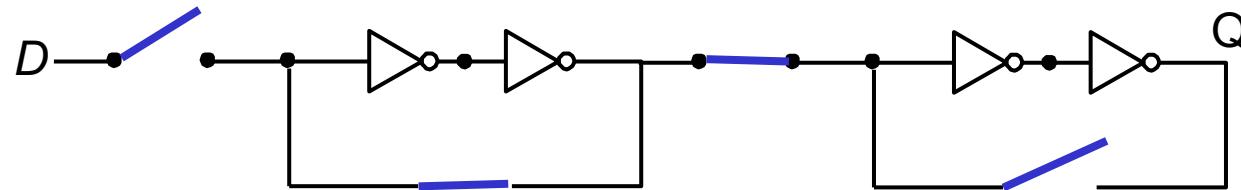
- Implementação utilizando chaves



$CK=0 \rightarrow$ Mestre amostra e escravo mantém o estado anterior



CK sobe \rightarrow Informação transferida para a saída



LATCH S-R EM VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity S_R_latch is
    port ( S, R : in  STD_LOGIC;
           Q, nQ : out STD_LOGIC);
end S_R_latch ;
```

```
architecture a1 of S_R_latch is
    signal qq, nqq: STD_LOGIC;
begin
```

```
    qq  <= R nor nqq;
```

```
    nqq <= S nor qq;
```

```
    nQ  <= nqq;
```

```
    Q   <= qq;
```

Q e notQ são também entradas, logo não podem estar em uma saída

```
end a1;
```

LATCH D EM VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity D_latch is
    port ( D, ck: in  STD_LOGIC;
           Q, nQ : out STD_LOGIC);
end D_latch ;

architecture a1of D_latch is
begin
    process (ck, D) ← LISTA DE SENSITIVIDADE:
    begin
        if (ck = '1') then
            Q <= D;
        end if;
    end process;
end a1;
```

LISTA DE SENSITIVIDADE:

- Ativa o *process* quando um dos sinais da lista mudar de valor

FLIP-FLOP D MESTRE-ESCRAVO EM VHDL

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity D_MS is  
    port ( D, ck: in  STD_LOGIC;  
          Q, nQ : out STD_LOGIC);  
end D_MS ;
```

```
architecture a1of D_MS is
```

```
begin  
    process (ck) ←  
    begin  
        if rising_edge(ck) then  
            Q <= D;  
        end if;  
    end process;  
end a1;
```

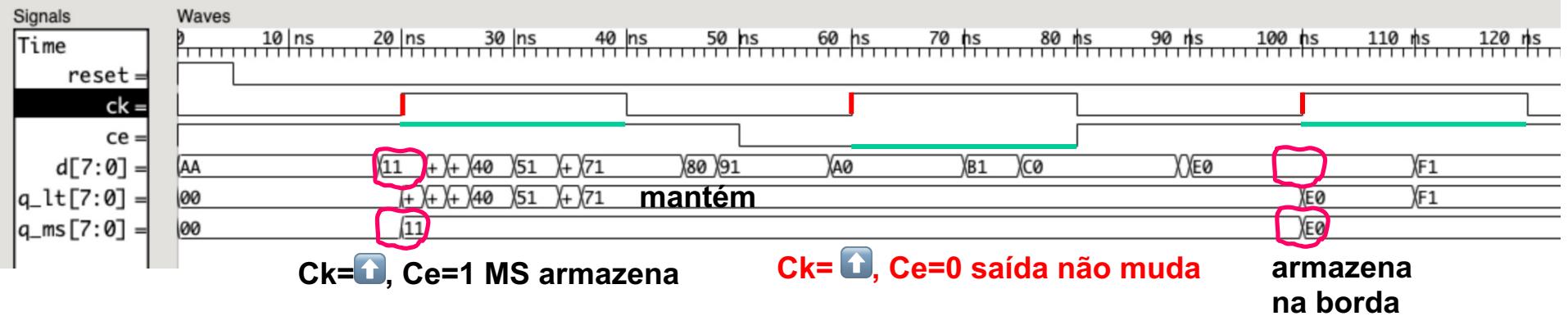
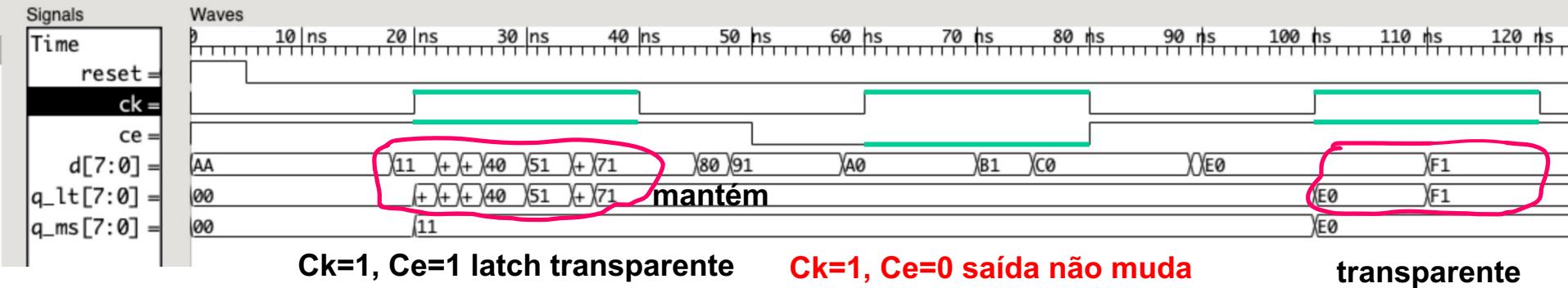
rising_edge(clk) = clock 'event and clock='1'

LISTA DE SENSITIVIDADE:

- Apenas o ck na lista

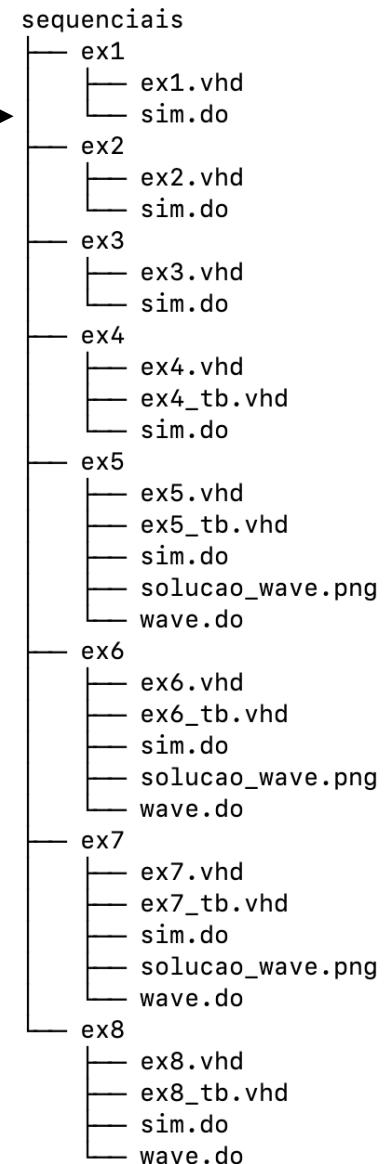
Observe que o bloco **process** é sensível a qualquer alteração no valor de **ck**, no entanto, a função **rising_edge** (definida na biblioteca IEEE) verifica a alteração do valor quando ocorre uma **borda de subida** (evento), executando as instruções do bloco **if**.

LATCH e MS – diferença de comportamento



Exercício de Simulação 1

- Baixe do Moodle o arquivo **sequenciais.zip**
 - Em “material de apoio” → “Material de apoio a VHDL” → **sequenciais.zip**
- Abra o arquivo “**ex1.vhd**” e o analise
 - Quais os módulos de hardware que estão sendo modelados nesta descrição?
- Simule este arquivo executando **vsim -do sim.do**
 - Explique a diferença observado entre as saídas ‘q_ms’ e ‘q_lt’.
 - Qual sinal é sensível ao nível?
 - Qual sinal é sensível à borda?
- Remova o sinal ‘D’ da linha 46: **process(rst, ck, D)** (na latch)
 - Explique o novo comportamento do sinal ‘q_lt’. Está correto?
 - Restaure a linha 46 após a simulação.



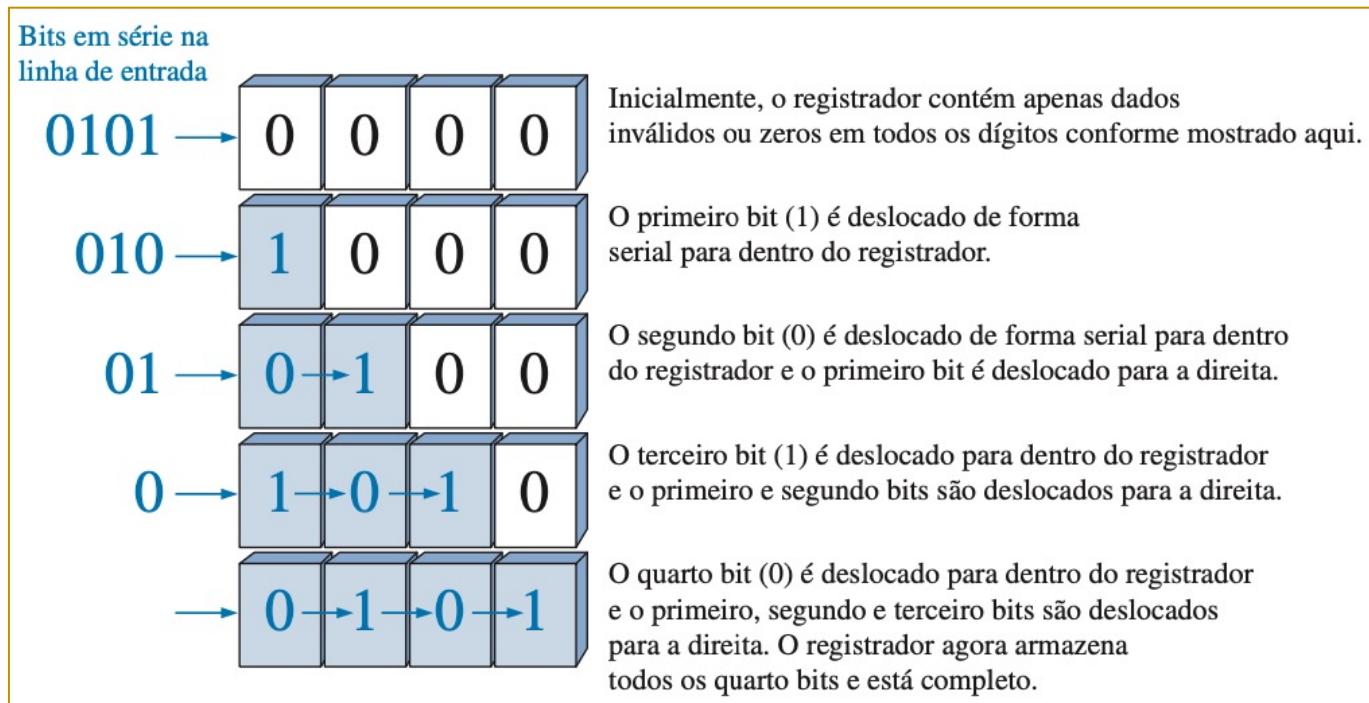
PARTE II

- Registradores
- Contadores

REGISTRADOR

Um **registraror** é formado pela combinação de vários **flip-flops** de forma que um grupo de bits possa ser armazenado.

- ✓ Exemplo de **registraror com entrada serial** de 4 bits que é construído a partir de quatro **flip-flops**:



REGISTRADOR

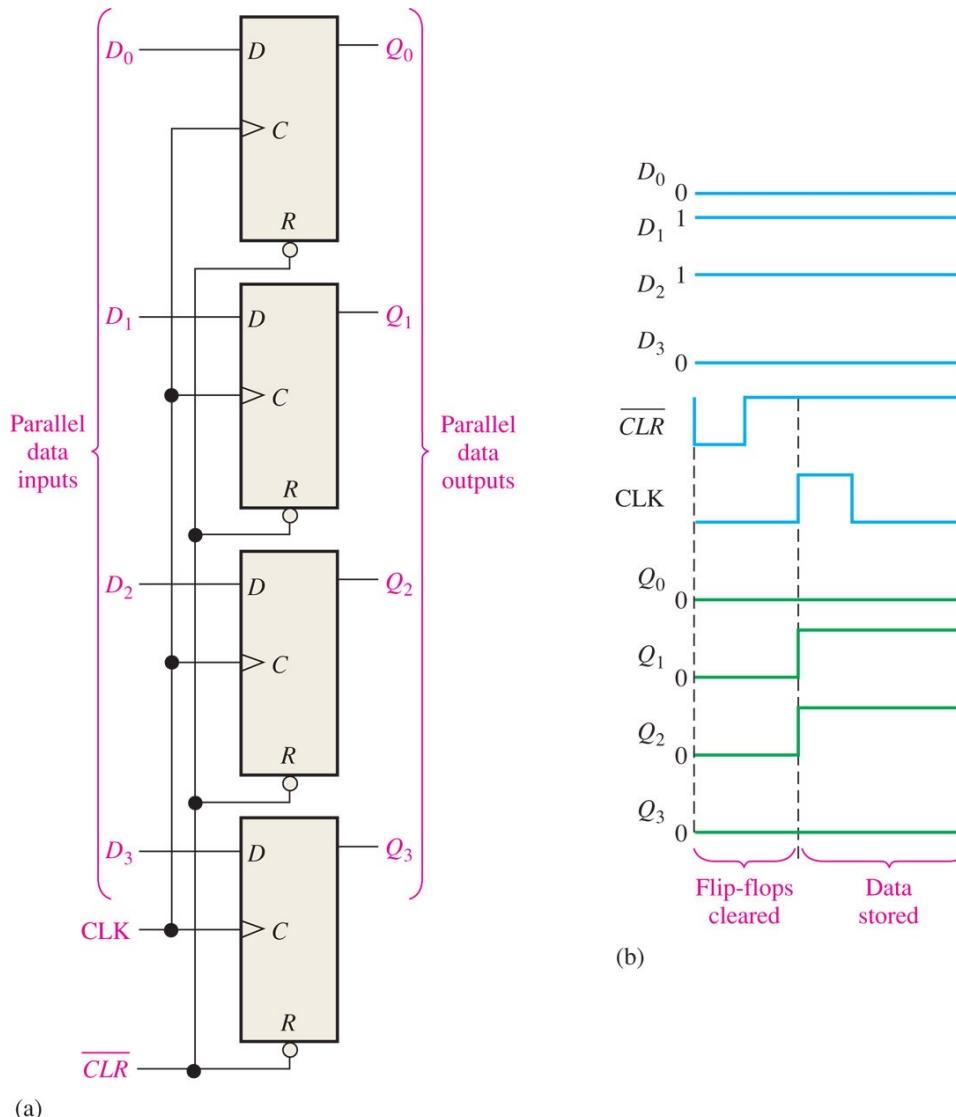


FIGURE 7-35 Example of flip-flops used in a basic register for parallel data storage.

REGISTRADOR

- registradores são basicamente sinais declarados em processos com sinal de sincronismo (exemplo: *clock*).
- para efeito de síntese e simulação, é aconselhável introduzir um reset assíncrono.

```
process (clock, reset)
begin
    if reset = '1' then
        reg <= (others =>'0');           -- portável;
    elsif rising_edge(clock) then
        reg <= barramento_A;
    end if;
end process;
```

→ Como introduzir um sinal de “enable” no registrador, para habilitar a escrita?

REGISTRADOR

- Registrador com largura de palavra parametrizável, com ‘ce’:

library

```
entity regnbit is
    generic(N : integer := 16); ← generic define um parâmetro do módulo
    port( ck, rst, ce : in std_logic;
          D : in STD_LOGIC_VECTOR (N-1 downto 0);
          Q : out STD_LOGIC_VECTOR (N-1 downto 0) );
end regnbit;

architecture regn of regnbit is
begin

process(ck, rst)
begin
    if rst = '1' then
        Q <= (others => '0');
    elsif rising_edge(ck) then
        if ce = '1' then
            Q <= D;
        end if;
    end if;
end process;
end regn;
```

Uso:

rx: entity work.regnbit generic map(8)
port map(ck => ck, rst => rst, ce => ce,
D => D , Q => Q);

Simulando o Registrador

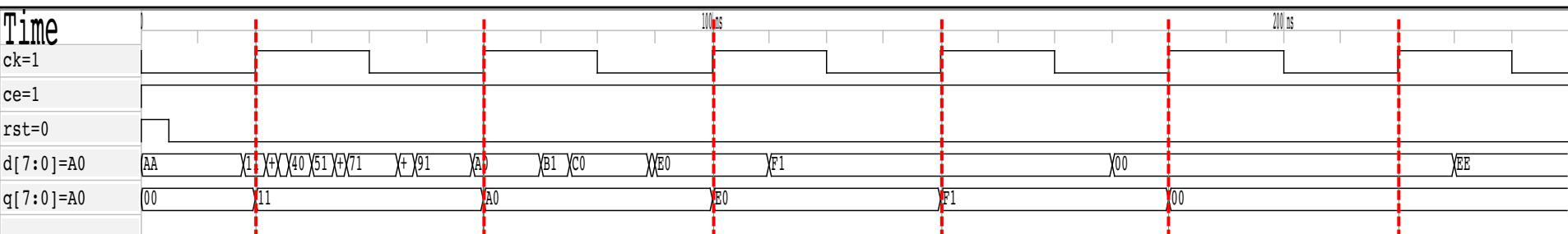
```
reg1: entity work.regnbit generic map(8)
      port map(ck => ck, rst => reset, ce => ce, D => D, Q => Q1);

reset <= '1', '0' after 5 ns;

ck <= not ck after 20 ns;

D <=  x"AA", x"11" after 18 ns, x"20" after 22 ns, x"31" after 24 ns, x"40" after 26 ns, x"51" after 30 ns,
       x"60" after 34 ns, x"71" after 36 ns, x"80" after 45 ns, x"91" after 48 ns, x"A0" after 58 ns,
       x"B1" after 70 ns, x"C0" after 75 ns, x"D1" after 89 ns, x"E0" after 90 ns, x"F1" after 110 ns,
       x"00" after 170 ns, x"EE" after 230 ns;

ce <= '1';
```



- sensibilidade à borda de subida do clock
- tempo de setup respeitado

REGISTRADOR DE DESLOCAMENTO

- exemplo de registrador de deslocamento:

```
process (clock, reset)
begin
    if reset = '1' then
        A <= 0;  B <= 0;  C <= 0;
    elsif rising_edge(clock) then
        A <= entrada;
        B <= A;
        C <= B;
    end if;
end process;
```

- 1) Desenhe o circuito acima utilizando flip-flops
- 2) A ordem das atribuições (A,B,C) é importante ? O que ocorreria se fosse uma linguagem de programação tipo C?

REGISTRADOR DE DESLOCAMENTO

- Atribuição dentro/fora de process:

```
process (clock, reset)
```

```
begin
```

```
    if rising_edge(clock) then
```

```
        A <= entrada;
```

```
        B <= A;
```

```
        C <= B;
```

```
        Y <= B + C;          -- dentro do process
```

```
    end if;
```

```
end process;
```

```
X <= B + C;          -- fora do process
```

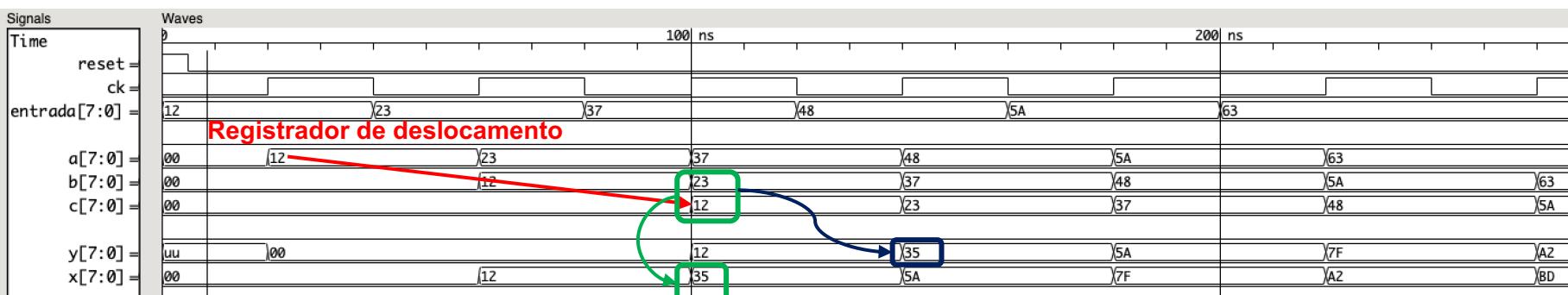
Qual a diferença de comportamento nas atribuições à X e a Y?

- Conclusão:

- sinais atribuídos em processos, com controle de clock, são flip-flops/registradores.

- Sinais fora de processos ou em processos sem variável de sincronismo (clock) serão sintetizados com lógica combinacional.

REGISTRADOR DE DESLOCAMENTO – SIMULAÇÃO



X: soma atual

Y: soma deslocada 1 ciclo de clock

```
process (clock, reset)
begin
    if rising_edge(clock) then
        iA <= entrada;
        iB <= A;
        iC <= B;
        Y <= iB + iC;          -- dentro do process
    end if;
end process;
X <= iB + iC;          -- fora do process
```

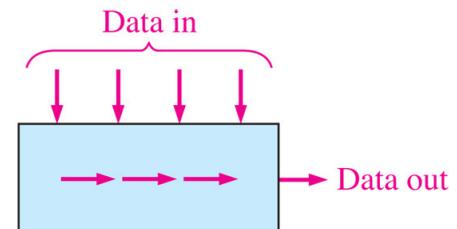
REGISTRADOR – formas de entrada de dados



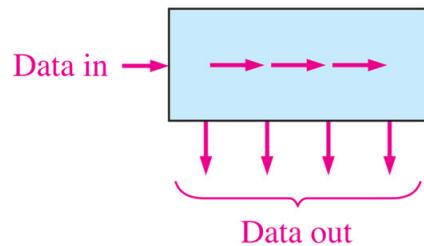
(a) Serial in/shift right/serial out



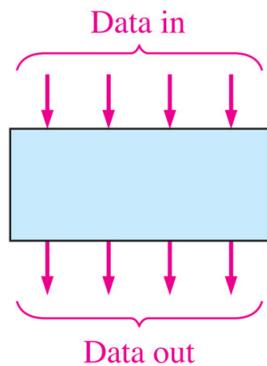
(b) Serial in/shift left/serial out



(c) Parallel in/serial out



(d) Serial in/parallel out



(e) Parallel in/parallel out



(f) Rotate right



(g) Rotate left

FIGURE 8-2 Basic data movement in shift registers. (Four bits are used for illustration. The bits move in the direction of the arrows.)

REGISTRADOR – carga serial

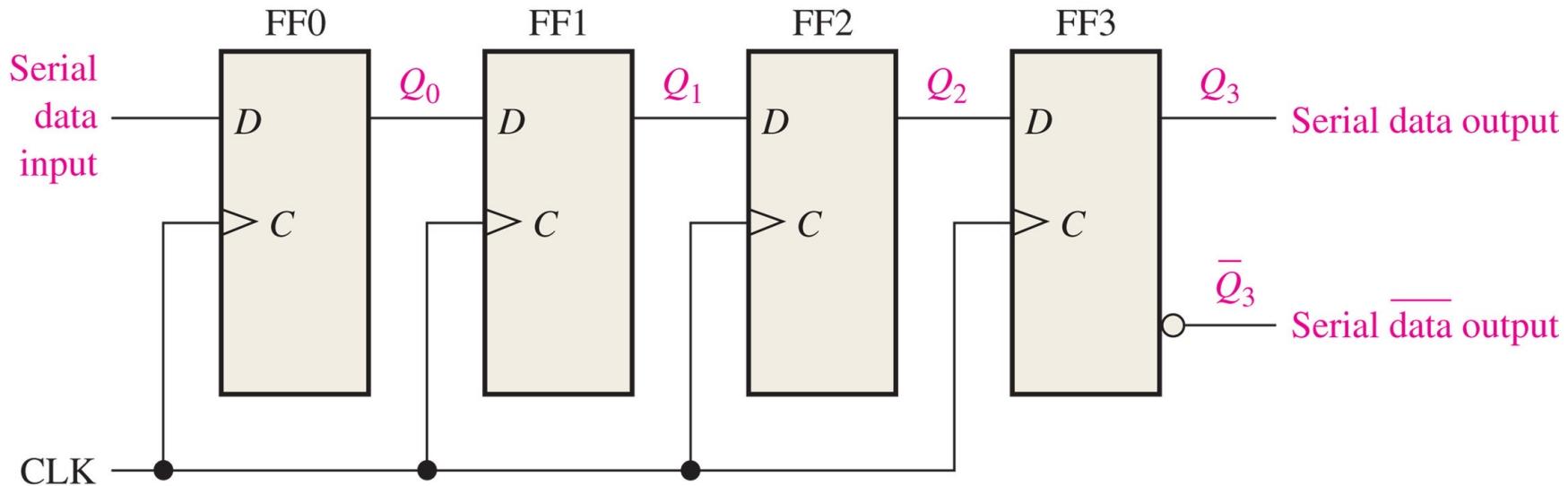


TABLE 8-1

Shifting a 4-bit code into the shift register in Figure 8–3.
Data bits are indicated by a beige screen.

CLK	FF0 (Q_0)	FF1 (Q_1)	FF2 (Q_2)	FF3 (Q_3)
Initial	0	0	0	0
1	0	0	0	0
2	1	0	0	0
3	0	1	0	0
4	1	0	1	0

REGISTRADOR – carga serial

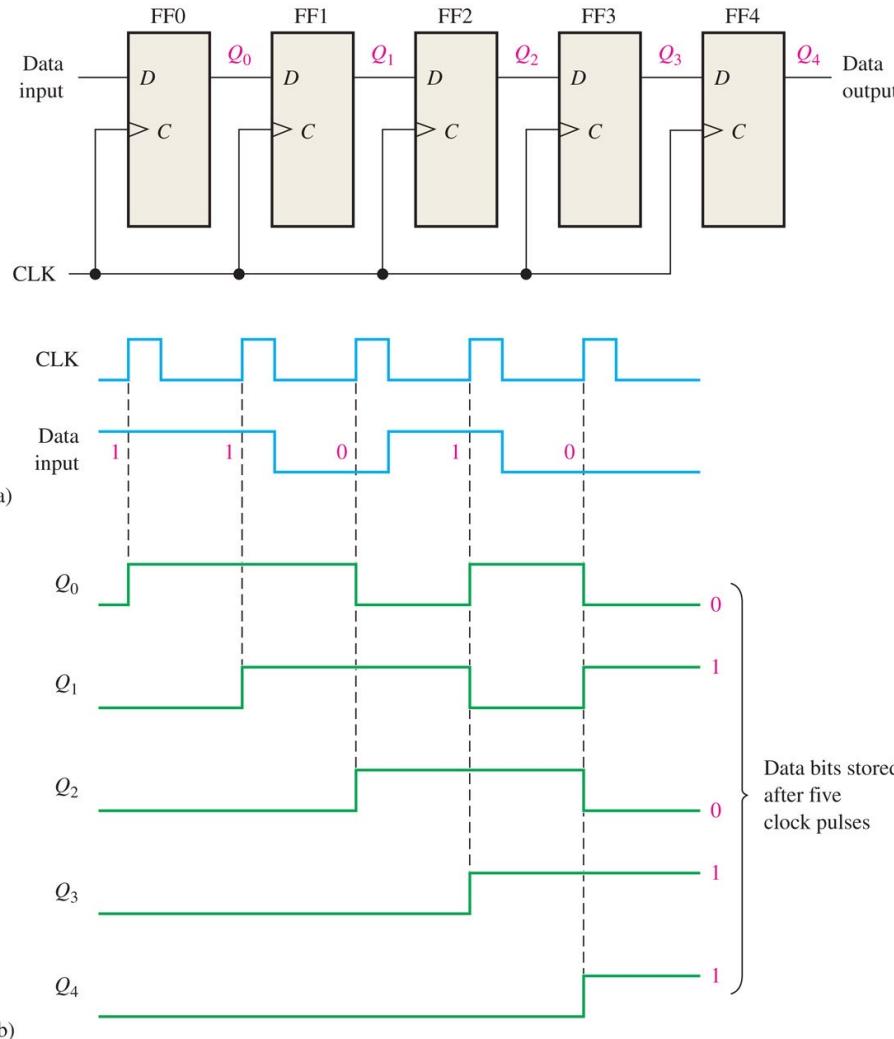
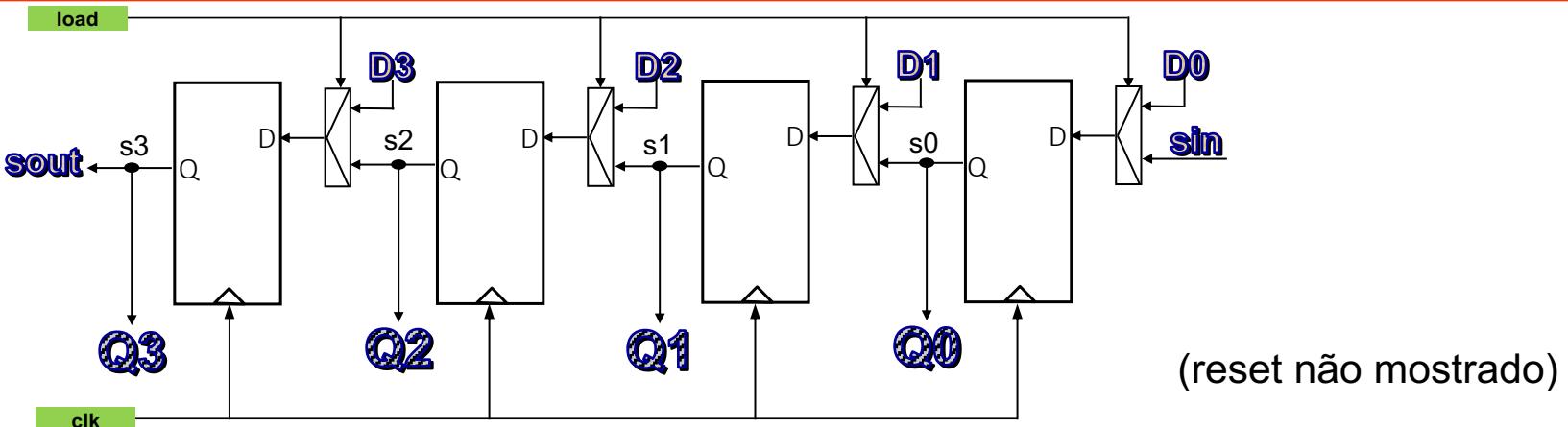


FIGURE 8-4

REGISTRADOR CARGA PARALELA E SERIAL

PARA
4 BITS



```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.Std_Logic_unsigned.all;
4 entity serial_in is
5     generic(N : integer := 8);
6     port( clk, load, reset : in std_logic;
7           sin : in std_logic;
8           sout : out std_logic;
9           d : in STD_LOGIC_VECTOR (N-1 downto 0);
10          q : out STD_LOGIC_VECTOR (N-1 downto 0)
11      end serial_in;
12
13 architecture a1 of serial_in is
14     signal s: STD_LOGIC_VECTOR (N-1 downto 0);
15 begin

```

```

13     architecture a1 of serial_in is
14         signal s: STD_LOGIC_VECTOR (N-1 downto 0);
15     begin
16
17         process (clk, reset)
18         begin
19             if reset = '1' then
20                 s <= (others=>'0');
21             elsif rising_edge(clk) then
22                 if load = '1' then
23                     s <= d; carga paralela
24                 else
25                     s <= s(n-2 downto 0) & sin; carga serial
26                 end if;
27             end if;
28         end process;
29
30         q <= s; saída paralela
31
32         sout <= s(n-1); saída serial
33
34     end a1;

```

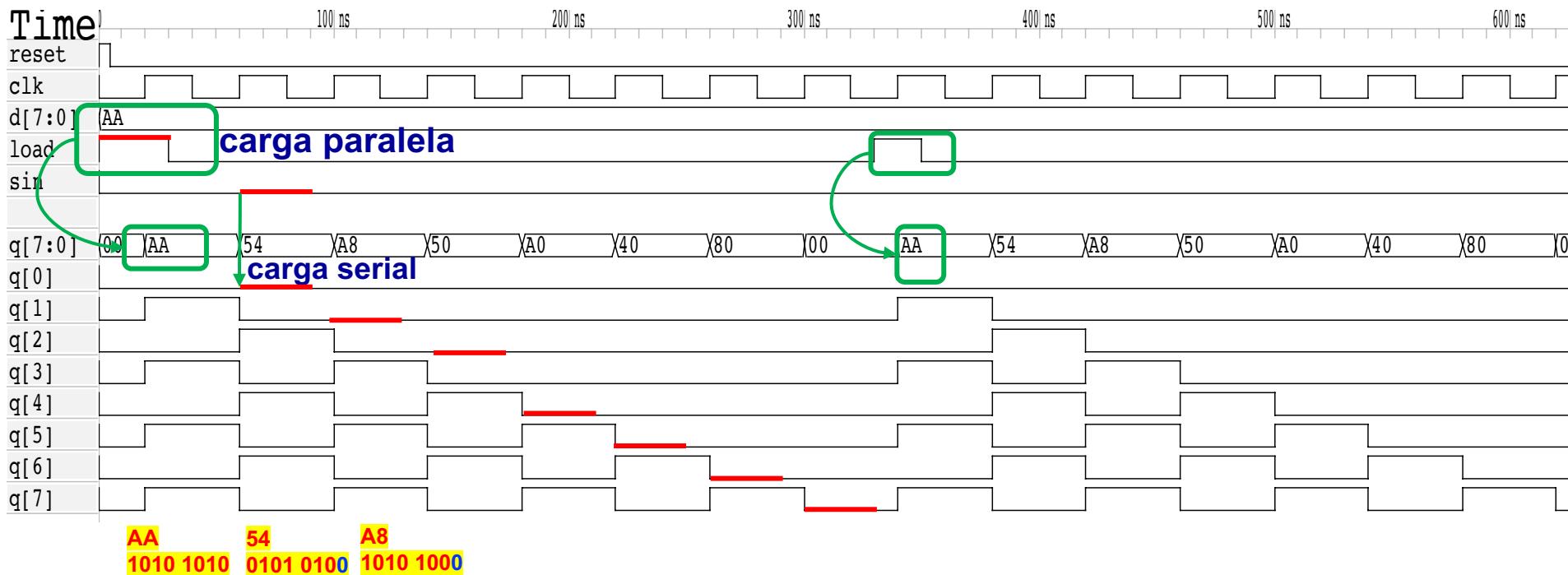
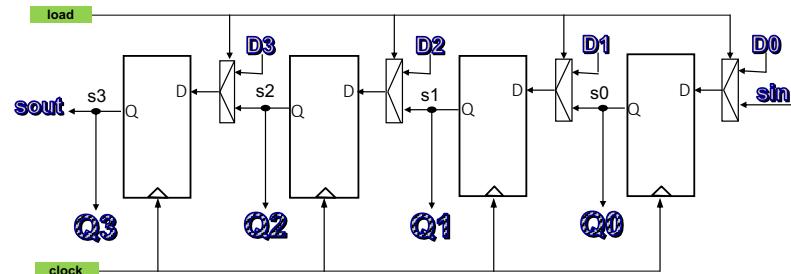
REGISTRADOR CARGA PARALELA E SERIAL

Deslocamento à esquerda multiplica por 2
 $AA \rightarrow (1) 54 \rightarrow A8 \rightarrow (1)50 \rightarrow A0 \dots$

$DD \leq x"AA";$

$sin \leq '0';$

$load \leq '1', '0' \text{ after } 30 \text{ ns}, '1' \text{ after } 330 \text{ ns}, '0' \text{ after } 350 \text{ ns};$



CONTADOR– 2ⁿ estados

```
entity contup is
    port (
        clock, reset, Load, Enable: In std_logic;
        DATABUS : In Std_logic_Vector (5 downto 0);
        Upcount2 : Out Std_logic_Vector (5 downto 0));
end contup;

architecture RTL of contup is
    Signal Upcount : std_logic_Vector (5 downto 0);
begin
    Upcount2 <= Upcount;

    Upcounter : Process (clock, reset)
    begin
        if reset = '1' then
            Upcount <= "000000";
        elsif rising_edge(clock) then
            if ENABLE = '1' then
                if LOAD = '1'  then Upcount <= DATABUS;
                                else Upcount <= Upcount + 1;
                end if;
            end if;
        end if;
    end process Upcounter;
end RTL;
```

- (1) Determine o comportamento deste contador, fazendo um diagrama de tempos.
- (2) O reset é prioritário em relação ao clock? Por quê?
- (3) Como modificar o contador para realizar contagem crescente/decrescente?

CONTADOR GRAY – 2ⁿ estados

- Código gray: sequência onde de um estado para outro há apenas a variação de um bit:
 $000 \rightarrow 001 \rightarrow 011 \rightarrow 010 \rightarrow 110 \rightarrow 111 \rightarrow 101 \rightarrow 100 \rightarrow 000 \rightarrow \dots$
- Uma forma de implementar este código, que não apresenta uma sequência regular, é utilizar uma técnica tipo “máquina de estados”, onde em função do estado atual do contador, determina-se o próximo estado.

```
architecture RTL of graycounter is
    signal clock, reset : std_logic; signal graycnt : std_logic_vector (2 downto 0);
begin
gray : process (clock,reset)
begin
    if reset = '1' then
        graycnt <= "000";      -- reset assíncrono
    elsif rising_edge(clock) then
        case graycnt is
            when "000" => graycnt <= "001";
            when "001" => graycnt <= "011";
            when "010" => graycnt <= "110";
            when "011" => graycnt <= "010";
            when "100" => graycnt <= "000";
            when "101" => graycnt <= "100";
            when "110" => graycnt <= "111";
            when "111" => graycnt <= "101";
            when others => null;
        end case; end if;
    end process gray;
end RTL;
```

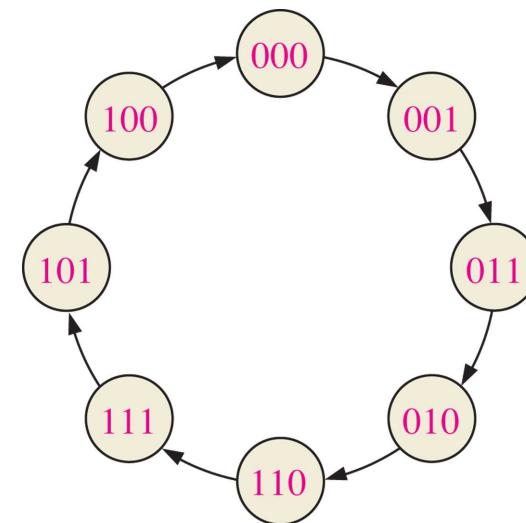


FIGURE 9-26 State diagram for a 3-bit Gray code counter.

CONTADOR Johnson – $2n$ estados

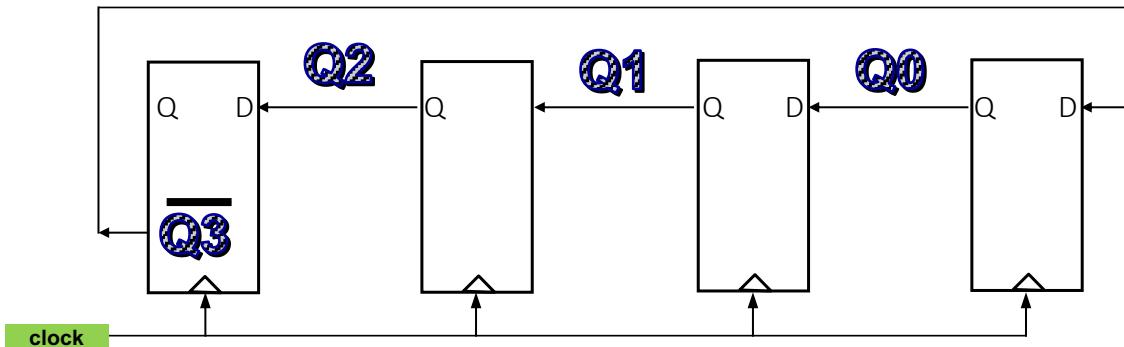


TABLE 8-3

Four-bit Johnson sequence.

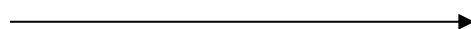
Clock Pulse	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

- Contador JOHNSON, utilizando um registrador de deslocamento:

```
if reset = '1' then
    Q <= "0000";
elsif rising_edge(clock) then
    Q <= Q(2 downto 0) & not (Q(3));      -- CONCATENAÇÃO
end if;
```

Simulação 2

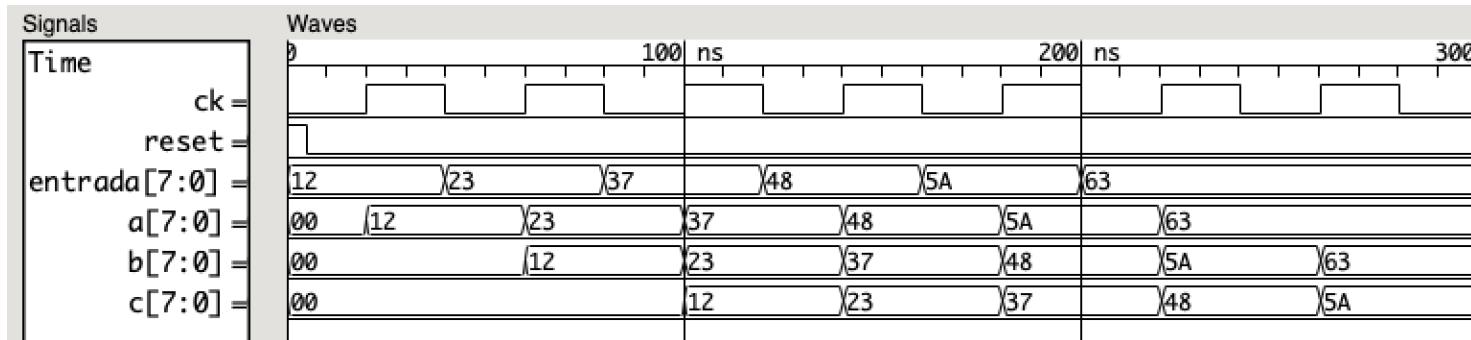
- Se não baixou ainda, baixe do Moodle o arquivo **sequenciais.zip**
 - Em “material de apoio” → “Material de apoio a VHDL” → **sequenciais.zip**



```
sequenciais
├── ex1
│   ├── ex1.vhd
│   └── sim.do
├── ex2
│   ├── ex2.vhd
│   └── sim.do
├── ex3
│   ├── ex3.vhd
│   └── sim.do
├── ex4
│   ├── ex4.vhd
│   ├── ex4_tb.vhd
│   └── sim.do
├── ex5
│   ├── ex5.vhd
│   ├── ex5_tb.vhd
│   ├── sim.do
│   ├── solucao_wave.png
│   └── wave.do
├── ex6
│   ├── ex6.vhd
│   ├── ex6_tb.vhd
│   ├── sim.do
│   ├── solucao_wave.png
│   └── wave.do
├── ex7
│   ├── ex7.vhd
│   ├── ex7_tb.vhd
│   ├── sim.do
│   ├── solucao_wave.png
│   └── wave.do
└── ex8
    ├── ex8.vhd
    ├── ex8_tb.vhd
    ├── sim.do
    └── wave.do
```

Exercício de Simulação 2

- Abra o arquivo “ex2.vhd” e o analise
 - Qual o módulo de hardware que está sendo modelados nesta descrição?
- Simule este arquivo executando `do sim.do`
 - Explique o comportamento observado nos registradores A/B/C
 - Porque, apesar de estarem em série, o valor da entrada não é transferido para a saída C?
- Altere a linha 47 para ‘`process (ck, rst, entrada, iA, iB)`’, comente a linha 53, e acrescente na linha seguinte ‘`elsif ck='1' then`’:
 - `elsif rising_edge(ck) then`
 - `elsif ck='1' then`
 - Simule novamente. O que ocorreu? Explique o novo comportamento.
 - Restaure o arquivo para sua configuração original.



Exercício de Simulação 2b

- No arquivo “ex2.vhd” faça as seguintes modificações *shift_reg*
 - Acrescente duas novas saídas, X e Y, de mesma largura dos registradores
 - Acrescente as atribuições dentro e fora do *process*:

```
...
    iA <= entrada;
    iB <= iA;
    iC <= iB;
    Y <= iB + iC;
end if;
end process;
X <= iB + iC;
```

- No arquivo “ex2.vhd” faça as seguintes modificações no *tb*
 - Acrescente estes 2 novos sinais entra a *architecture* e o *begin*
 - Acrescente no *port map* do *shift_reg* estes 2 novos sinais
- Simule o arquivo modificado com as novas somas
 - Explique o comportamento observado nos sinais X e Y

PARTE III

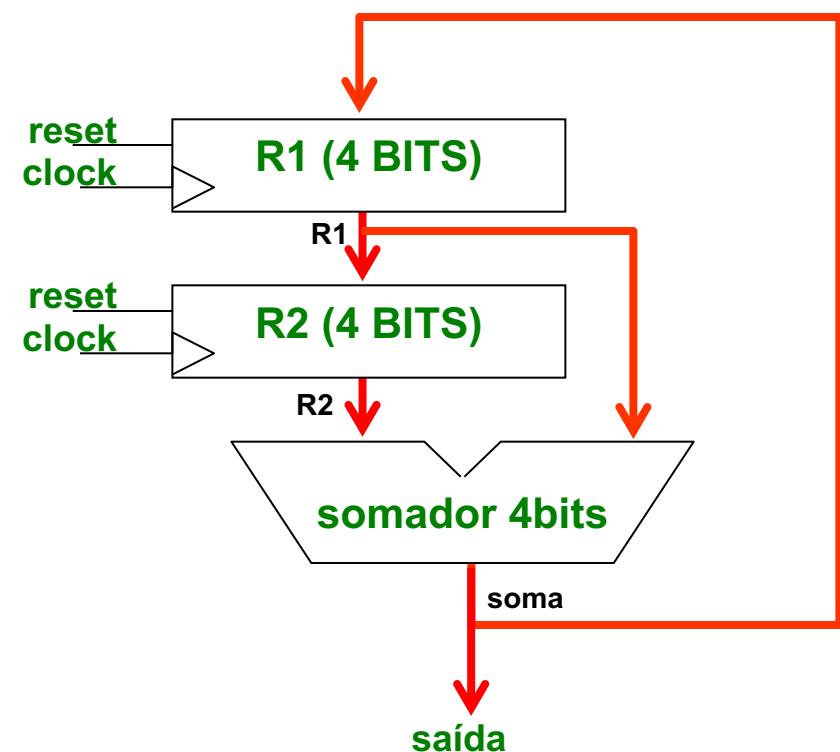
- Projeto RTL

PROJETO RTL

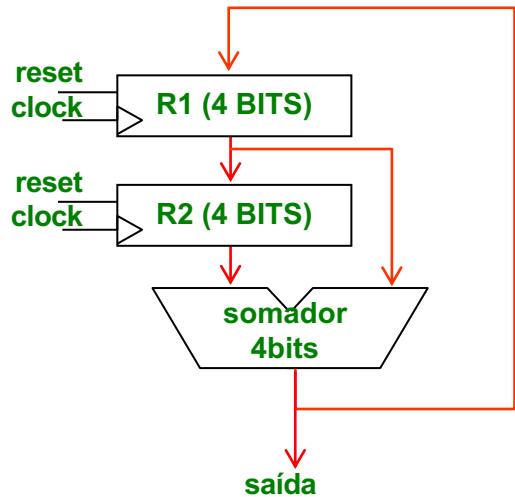
- Nos projetos de sistemas digitais, a descrição da operação de um circuito síncrono digital recebe o nome de **RTL** (do inglês *Register Transfer Level*)
- Em um projeto RTL, o **comportamento** do circuito é descrito em termos do fluxo de sinais, ou transferência de dados entre os registradores presentes no hardware e as operações lógicas conduzidas com estes sinais
- A abstração RTL é usada em linguagens de descrição de hardware como **Verilog** e **VHDL**
- Ferramentas de **simulação lógica** utilizam a descrição RTL para verificar sua funcionalidade
- A descrição RTL é usualmente convertida para a descrição no nível de portas lógicas por uma ferramenta de **síntese lógica**. O resultado da síntese representa a descrição física do circuito

Exercício de Simulação 3

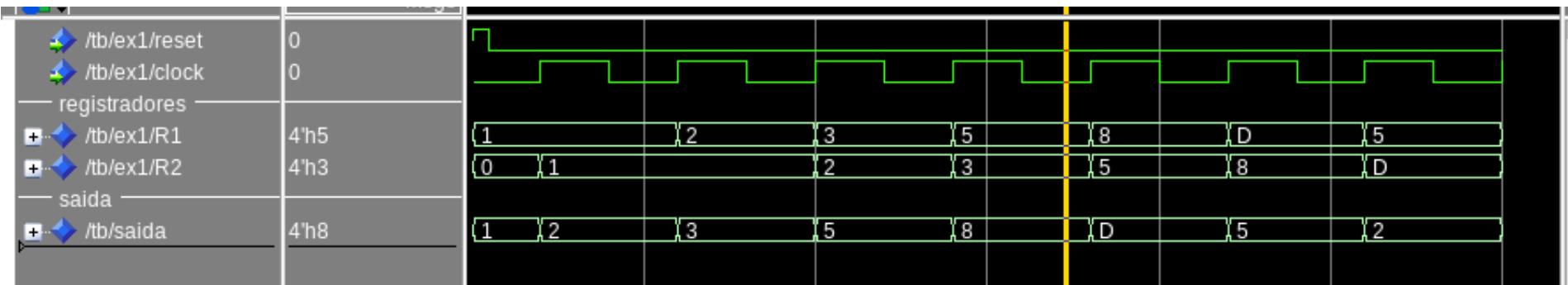
- Quando o sinal de reset for ‘1’ , os registradores R1 e R2 armazenam x“1” e x“0” respectivamente. A saída corresponde aos valores atuais de R1 e R2.



EXERCÍCIO DE SIMULAÇÃO 3



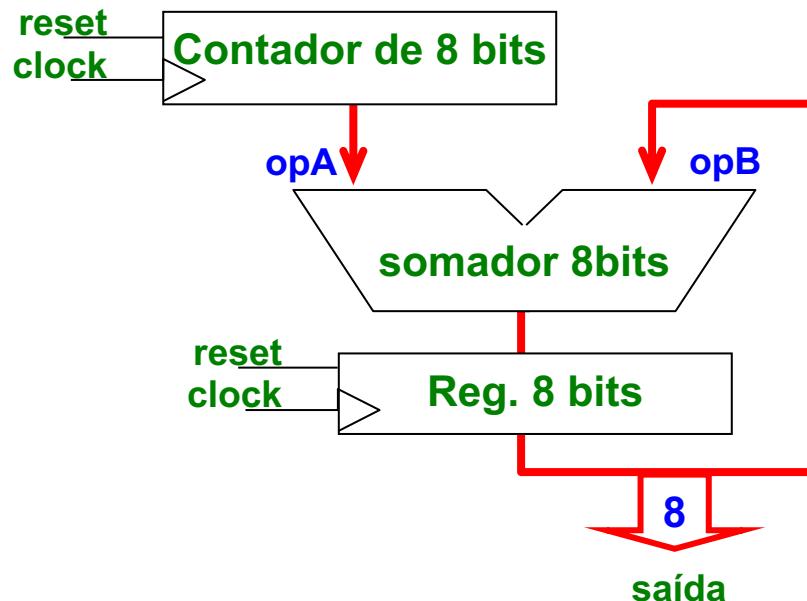
- Execute o script **do sim.do** (pasta ex3)
- Explique o comportamento observado



R1: 1 → 1 → 2 → 3
R2: 0 → 1 → 1 → 2
Saída: 1 → 2 → 3 → 5

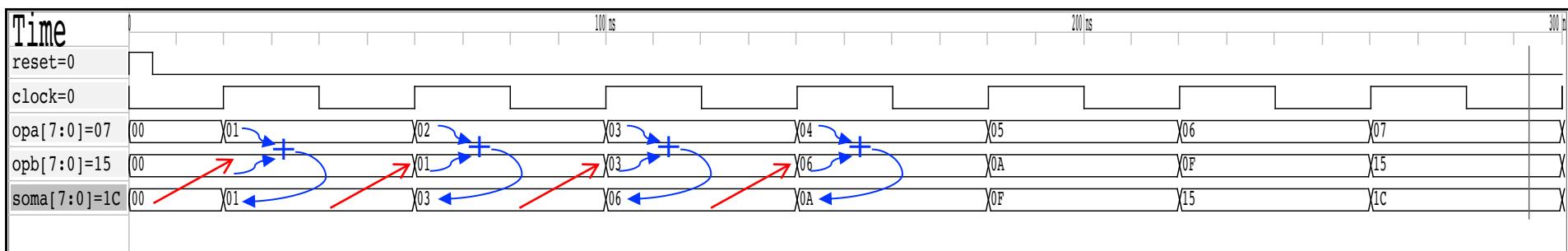
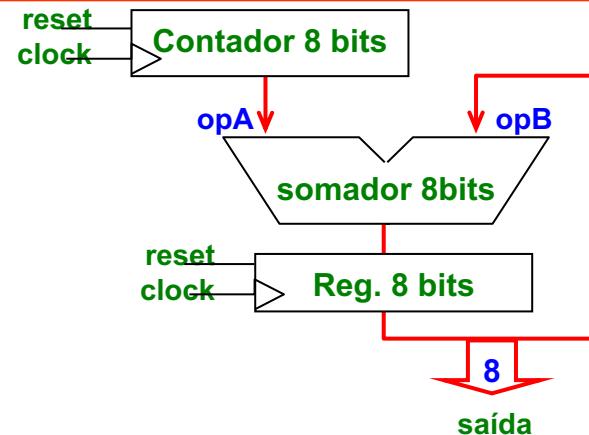
Exercício de Simulação 4

- Descreva o circuito abaixo em VHDL:
 - Um só processo para opA e opB, pois as variáveis de controle são as mesmas



```
port(clock, reset: in std_logic;  
      saída : out std_logic_vector (N-1 downto 0) );
```

EXERCÍCIO DE SIMULAÇÃO 4



opA:	0	1	2	3	4	5	6	7
	+ +							
opB:	0	0	1	3	6	10	15	21
soma:	0	1	3	6	10	15	21	28 51

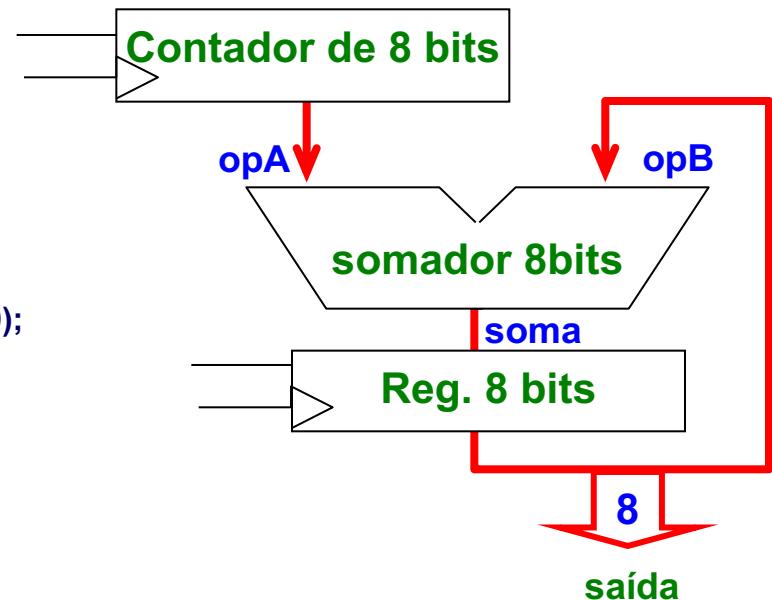
EXERCÍCIO DE SIMULAÇÃO 4

```
library IEEE;
use IEEE.Std_Language.all;
use IEEE.Std_Language_unsigned.all;

entity exercicio4 is
    generic(N : integer := 8);
    port(  clock, reset : in std_logic;
            saida : out STD_LOGIC_VECTOR (N-1 downto 0) );
end exercicio4;

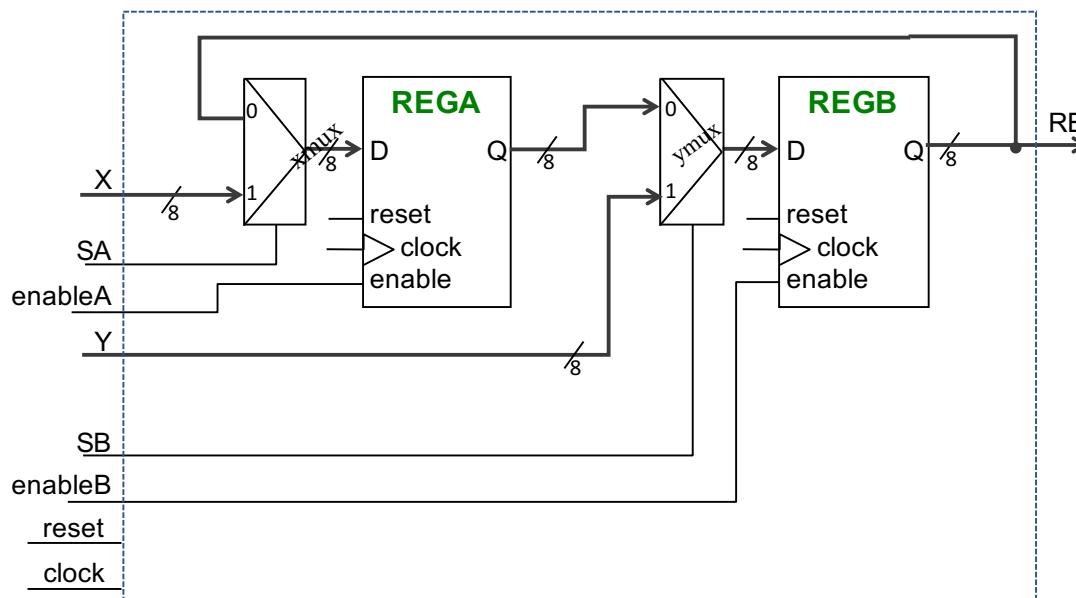
architecture a1 of exercicio4 is
    signal opA, opB, soma: STD_LOGIC_VECTOR (N-1 downto 0);
begin
    process (clock, reset)
    begin
        if reset = '1' then
            opA <= ....
            opB <= ....
        elsif rising_edge(clock) then
            opA <= ....
            opB <=....
        end if;
    end process;

    soma <= opA + opB
    saida <= soma;
end a1;
```

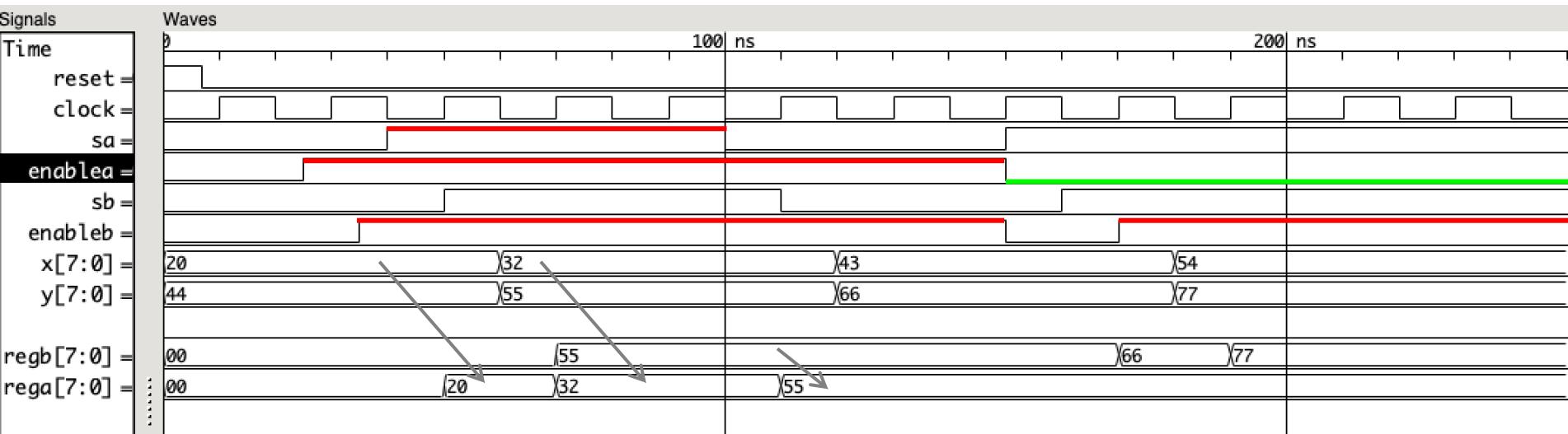
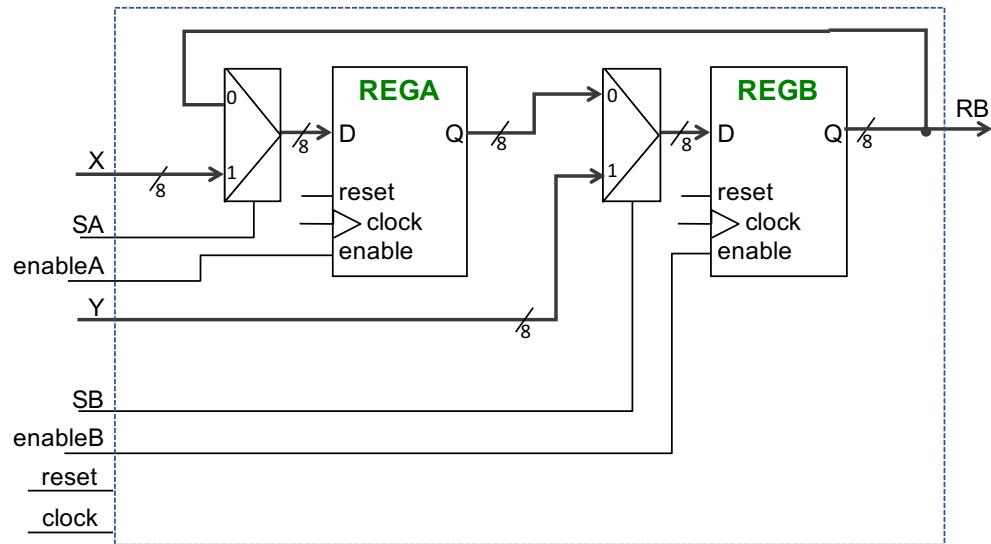


Exercício de Simulação 5

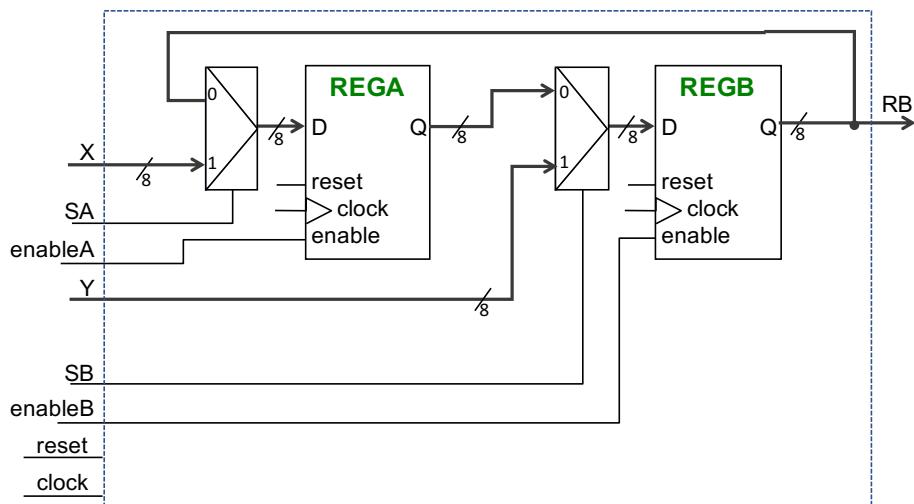
1. (2,5 pts) **Projeto RTL.** Considere o circuito abaixo composto por 2 registradores de 8 bits e dois multiplexadores 2x1. O circuito possui 8 **entradas**: dois sinais de 8 bits: X e Y , habilitação dos registradores: $enableA$ e $enableB$, seleção dos multiplexadores: SA e SB , e o sinal de $clock$ e $reset$ para os registradores. Como **saída** o circuito possui o valor do registrador RB . Descreva em VHDL este diagrama RTL (completo, porém detalhes de sintaxe são irrelevantes). Descreva o circuito utilizando somente um par entidade-arquitetura, ou seja, não façam instanciação (*port map*) de componentes.



Exercício de Simulação 5



5 - Solução



```
library IEEE;
use IEEE.Std_Language.all;
use IEEE.Std_Language_unsigned.all;

entity ex5 is
    port(reset, clock, SA, enableA, SB, enableB : in std_logic;
          X, Y : in std_logic_vector (7 downto 0);
          RB : out std_logic_vector (7 downto 0));
end ex5;

architecture ex5 of ex5 is
    signal xmux, ymux : std_logic_vector (7 downto 0);
    signal regA, regB : std_logic_vector (7 downto 0);
begin
    RB  <= regB;

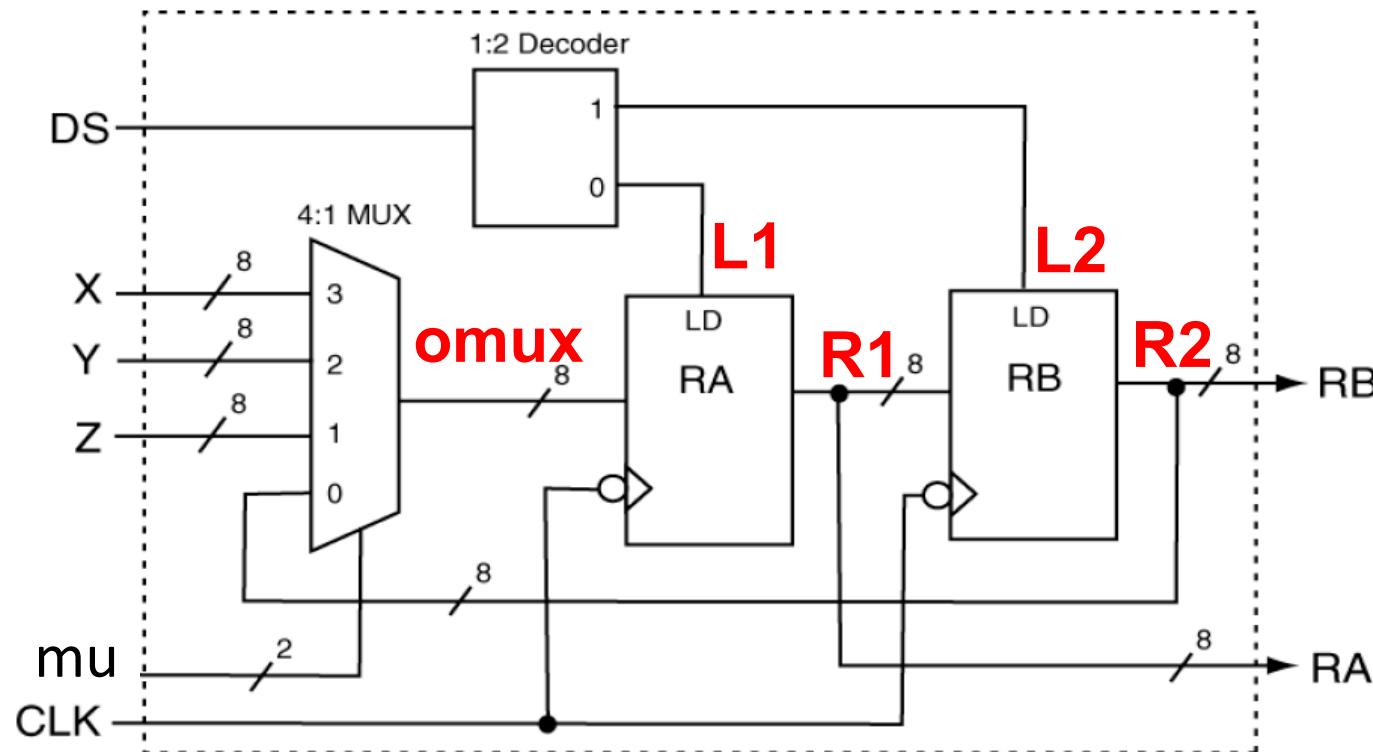
    xmux <= X when SA = '1' else regB;
    ymux <= Y when SB = '1' else regA;

    process (reset, clock)
    begin
        if reset = '1' then
            regA <= (others => '0');
            regB <= (others => '0');
        elsif rising_edge(clock) then
            if enableA = '1' then
                regA <= xmux;
            end if;
            if enableB = '1' then
                regB <= ymux;
            end if;
        end if;
    end process;

end ex5;
```

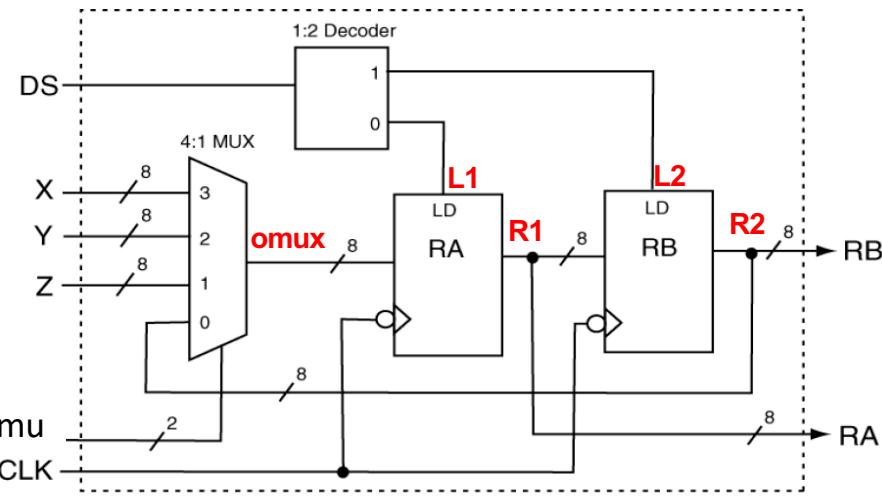
Exercício de Simulação 6

- Descreve e simular em VHDL



O **reset** não está mostrado no desenho e deve ser incluído no circuito para inicializar os registradores

Saída esperado para o ex6



/tb/dut/reset	0													
/tb/dut/clock	0													
entradas														
/tb/dut/X	8'hA0	00	10	20	30	40	50	60	70	80	90	A0	B0	C0
/tb/dut/Y	8'h32	00	05	0A	0F	14	19	1E	23	28	2D	32	37	3C
/tb/dut/Z	8'h14	00	02	04	06	08	0A	0C	0E	10	12	14	16	18
saída mux		0	1	2	3	0	1	2	3	0	1	2	3	0
/tb/dut/mu	2'h2	00	02	0A	30	0A		1E	70	1E	12	32	80	32
/tb/dut/OMUX	8'h32												1A	46
registadores														
/tb/DS	1													
/tb/dut/L1	0													
/tb/dut/RA	8'h32	00	02	0A				1E			12	32		1A
/tb/dut/L2	1													46
/tb/dut/RB	8'h1E	00			0A				1E				32	

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Std_Logic_unsigned.all;

entity ex6 is
    port(clock, reset, DS : in std_logic;
        mu :         in std_logic_vector(1 downto 0);
        X, Y, Z :     in std_logic_vector (7 downto 0);
        RA, RB :     out std_logic_vector (7 downto 0));
end ex6;

architecture a1 of ex6 is
    signal OMUX : std_logic_vector (7 downto 0);
    signal R1, R2: std_logic_vector (7 downto 0);
    signal L1, L2: std_logic;
begin
    RA <= R1;
    RB <= R2;

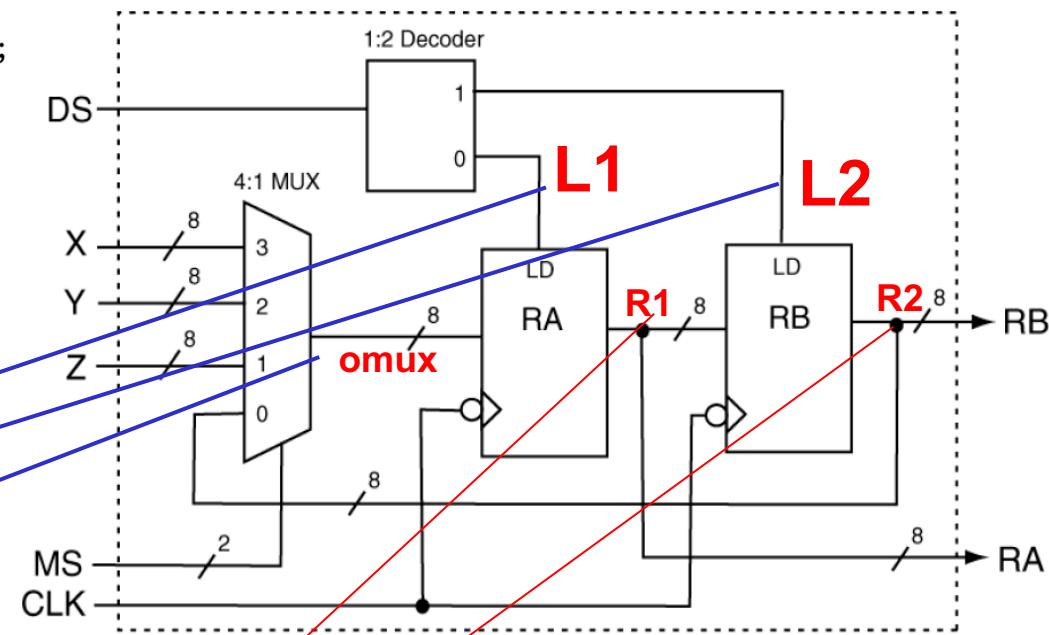
    L1 <= '1' when ds='0' else '0';
    L2 <= '1' when ds='1' else '0';

    OMUX <= X  when mu="11" else
              Y  when mu="10" else
              Z  when mu="01" else
                  R2;

    process (clock, reset)
    begin
        if reset='1' then
            R1 <= (others => '0');
            R2 <= (others => '0');
        elsif falling_edge(clock) then
            if L1='1' then R1 <= OMUX; end if;
            if L2='1' then R2 <= R1;   end if;
        end if;
    end process;
end a1;

```

ex6 - solução



Test Bench – ex6

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

entity tb is
end tb;

architecture tb of tb is
    signal reset, ck, DS : std_logic := '0';
    signal mu : std_logic_vector(1 downto 0) := "00";
    signal X, Y, Z, RA, RB : std_logic_vector(7 downto 0) := (others=>'0');
begin
dut: entity work.ex6
port map(reset=>reset, clock => ck, mu => mu, DS=>DS,
         X=>X, Y=>Y, Z=>Z, RA=>RA, RB=>RB);

ck <= not ck after 10 ns;
reset <= '1', '0' after 5 ns;

process (ck)
begin
    if rising_edge(ck) then
        mu <= mu + 1;
        X <= X + 16;
        Y <= Y + 5;
        Z <= Z + 2;
    end if;
end process;

DS <= not DS after 40 ns;

end tb;
```

Ex. de Simulação 7

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity divisor is
    port( fin, rst : in std_logic;
          fout : out std_logic );
end divisor;

architecture a1 of divisor is
    signal A, B, C: std_logic;
begin

process (fin, rst)
begin
    if rst = '1' then
        A <= '0';
    elsif rising_edge(fin) then
        A <= not A;
    end if;
end process;

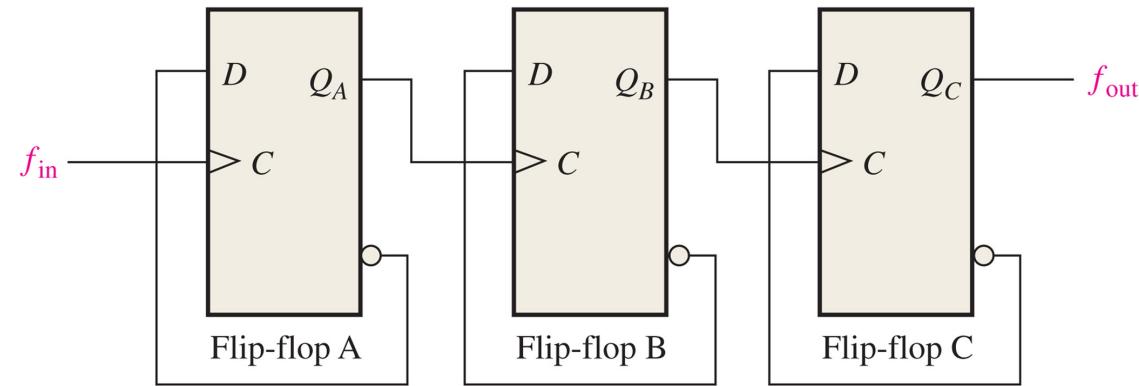
process (A, rst)
begin
    if rst = '1' then
        B <= '0';
    elsif rising_edge(A) then
        B <= not B;
    end if;
end process;

process (B, rst)
begin
    if rst = '1' then
        C <= '0';
    elsif rising_edge(B) then
        C <= not C;
    end if;
end process;

fout <= C;

end a1;
```

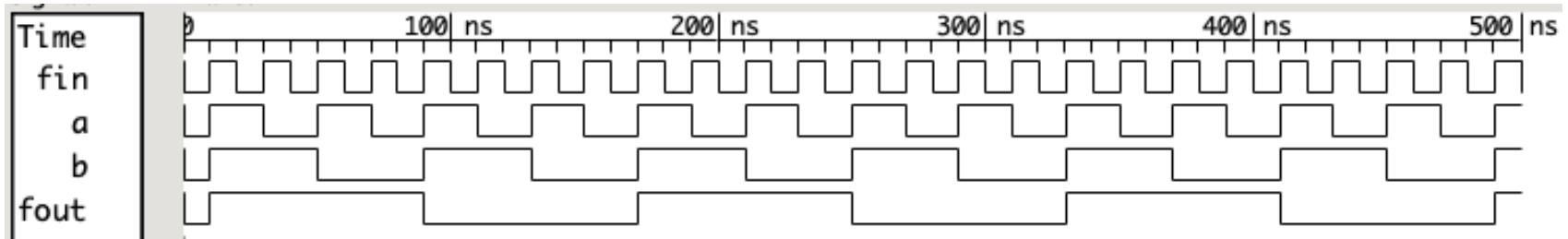
- Qual a saída esperada para *fout*?



Este é um circuito
divisor de clock!

O reset não está mostrado no desenho e deve ser incluído no circuito para inicializar os flip-flops

EXERCÍCIO 7 – Divisor de clock – test bench (2/2)



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

entity tb is
end tb;

architecture arch of tb is
    signal reset : std_logic;
    signal fin : std_logic := '0';
    signal fout: std_logic;
begin

    ex2: entity work.divisor
        port map(fin => fin, rst => reset, fout => fout);

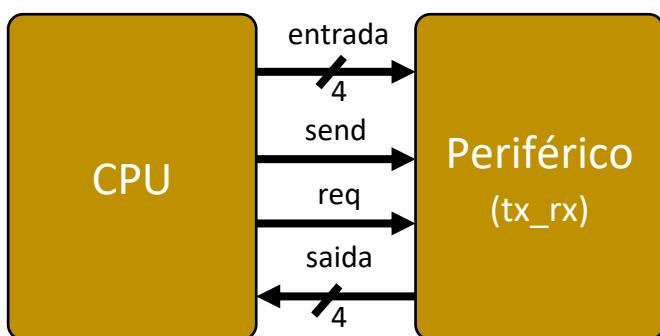
    reset <= '1', '0' after 5 ns;

    fin <= not fin after 10 ns;

end arch;
```

Exercício de Simulação 8

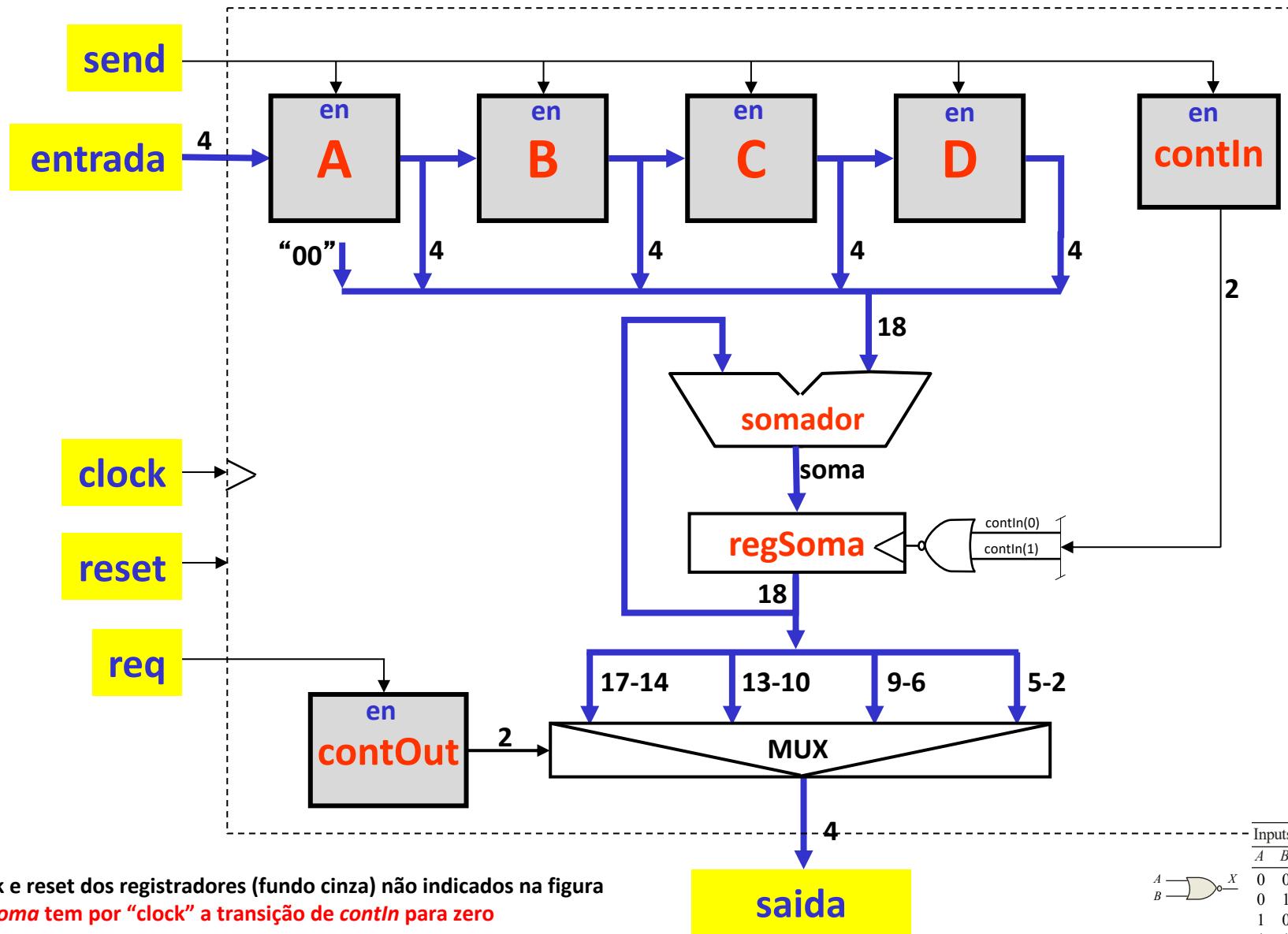
COMUNICAÇÃO ENTRE CPU E PERIFÉRICO



Função do Periférico:

- 1. Recebe 4 palavras de 16 bits**
 - ✓ A CPU deve enviar 16 palavras de 4 bits (cada **send** avisa que tem 4 bits disponíveis em **entrada**)
- 2. Somar as 4 palavras de 16 bits, sem perder precisão**
 - ✓ Para isto o somador deve ter 18 bits
- 3. Calcular a média aritmética das 4 palavras, sem utilizar divisão**
 - ✓ Emprego de deslocamento à direita
- 4. Enviar para a CPU a média (16 bits) em pacotes de 4 bits**
 - ✓ A CPU ativa **req** e o periférico devolve em **saída** parte da resposta
 - ✓ A CPU gera 4 **reqs**

Arquitetura – Visão Geral



clock e reset dos registradores (fundo cinza) não indicados na figura
regSoma tem por "clock" a transição de contIn para zero



```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.Std_Logic_unsigned.all;

entity ex8 is
    port( clock, reset, send, req : in std_logic;
          entrada : .....;
          saida : .... );
end ex8;

architecture a1 of ex8 is
    signal A, B, C, D: STD_LOGIC_VECTOR (3 downto 0);
    signal contIn, contOut: STD_LOGIC_VECTOR (1 downto 0);

begin

```

```

begin
    -- registrador de deslocamento controlado pelo send
    process (clock, reset)

```

```

begin
    if reset = '1' then
        contIn <= (others=>'0');
        A <= ....;
        B <= ....;
        C <= ....;
        D <= ....;
    elsif rising_edge(clock) then
        if send=..... then
            A <= ....;
            B <= ....;
            C <= ....;
            D <= ....;
            contIn <= ....;
        end if;
    end if;
end process;

```

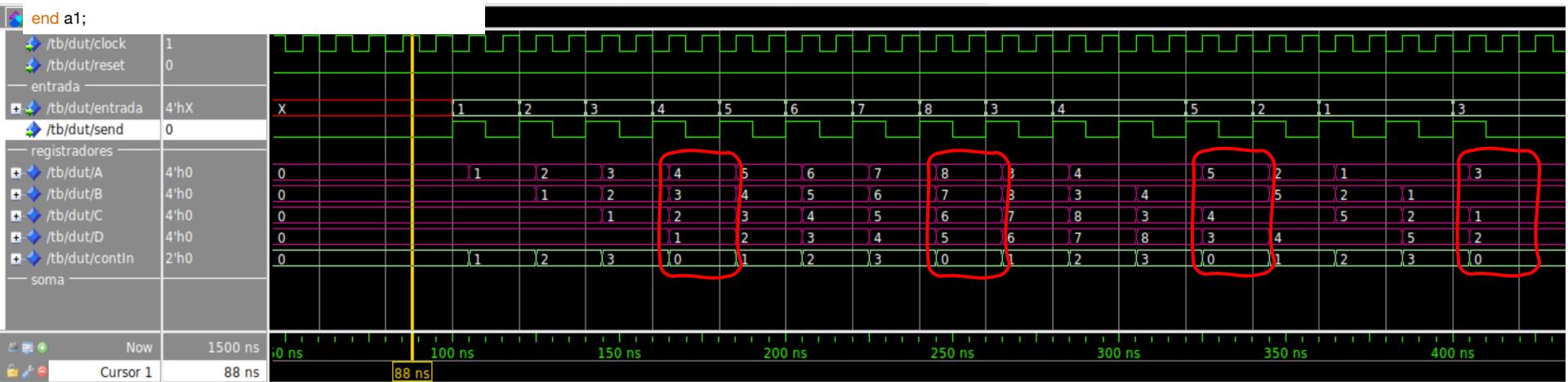
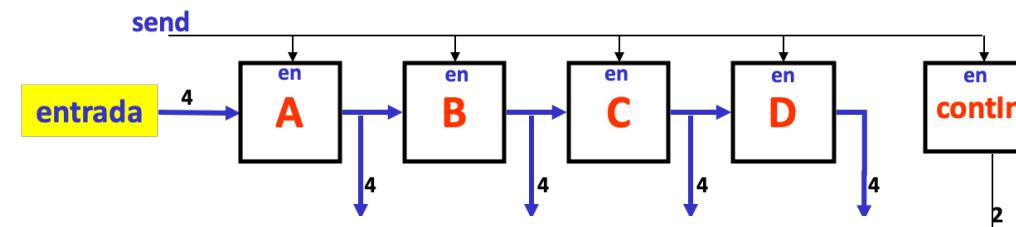
```

end a1;

```

+/tb/dut/clock	1
+/tb/dut/reset	0
entrada	
+ /tb/dut/entrada	4'hX
+ /tb/dut/send	0
registadores	
+ /tb/dut/A	4'h0
+ /tb/dut/B	4'h0
+ /tb/dut/C	4'h0
+ /tb/dut/D	4'h0
+ /tb/dut/contIn	2'h0
soma	
Now	1500 ns
Cursor 1	88 ns

Entrada de Dados (1/5)



Primeiro valor: x4321

x8765

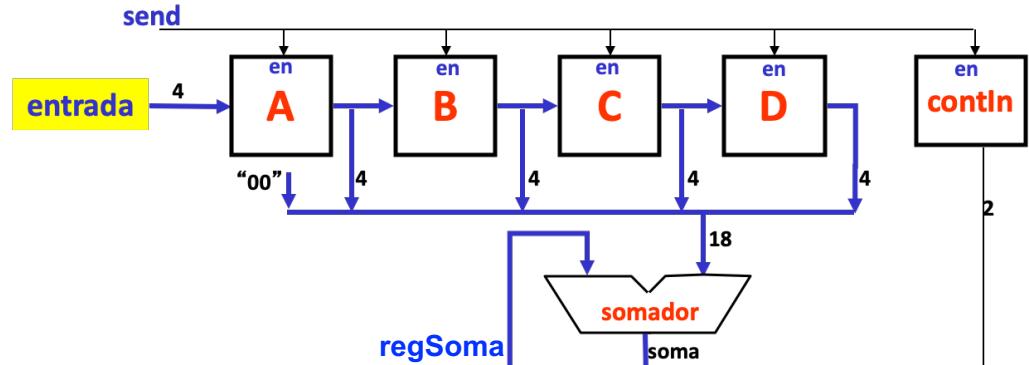
x5443

x3112 64

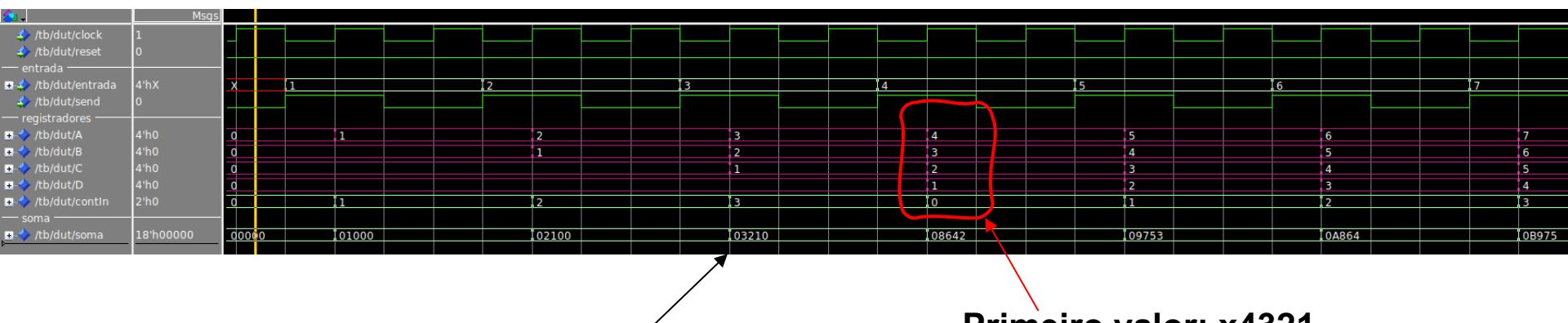
Soma (2/5)

Na declaração inicializar com zero os vetores soma e regSoma

```
signal soma, regSoma: STD_LOGIC_VECTOR (17 downto 0) := (others=>'0');
```



```
soma <= regSoma + ("00" & A & B & C & D);
```



Notar que está sempre somando

Primeiro valor: x4321

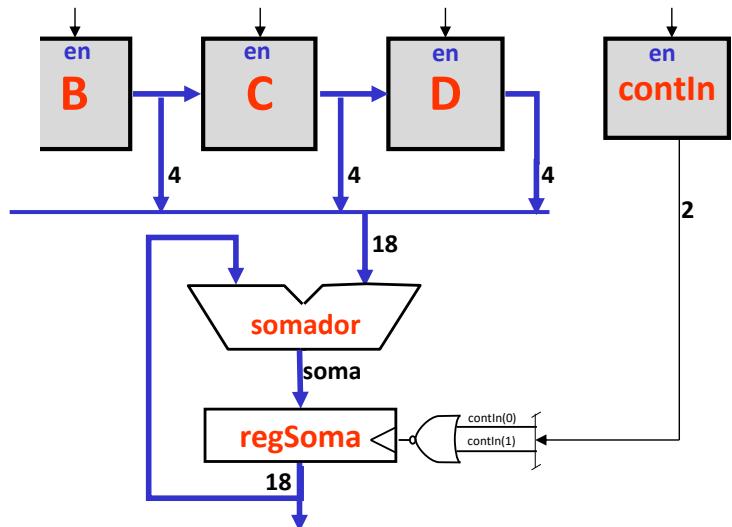
Acumulação (3/5)

Usar um sinal de controle que só permita a acumulação uma única vez por ciclo de 4 entrada de dados

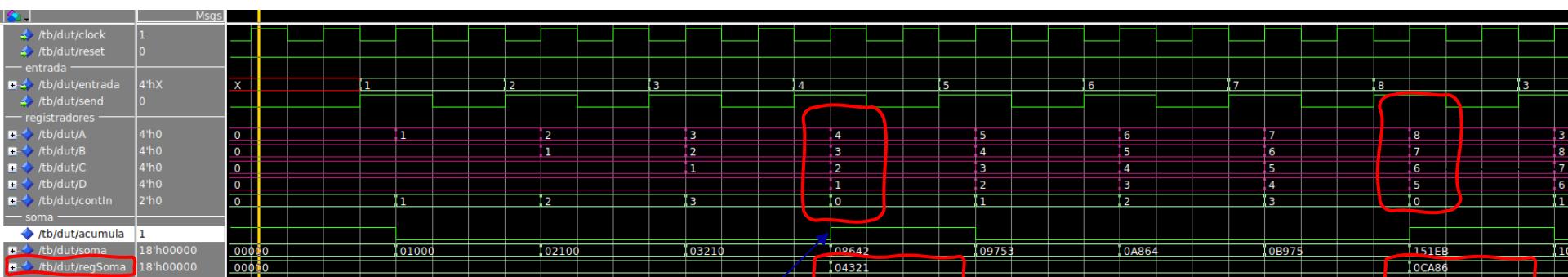
Solução:

acumula <= contIn(0) nor contIn(1); -- '1' quando "00"

```
process (acumula, reset)
begin
  if reset = '1' then
    regSoma <= (others=>'0');
  elsif rising_edge(acumula) then
    regSoma <= soma ;
  end if;
end process;
```



Detecta a transição de *contIn*=“00” para acumular



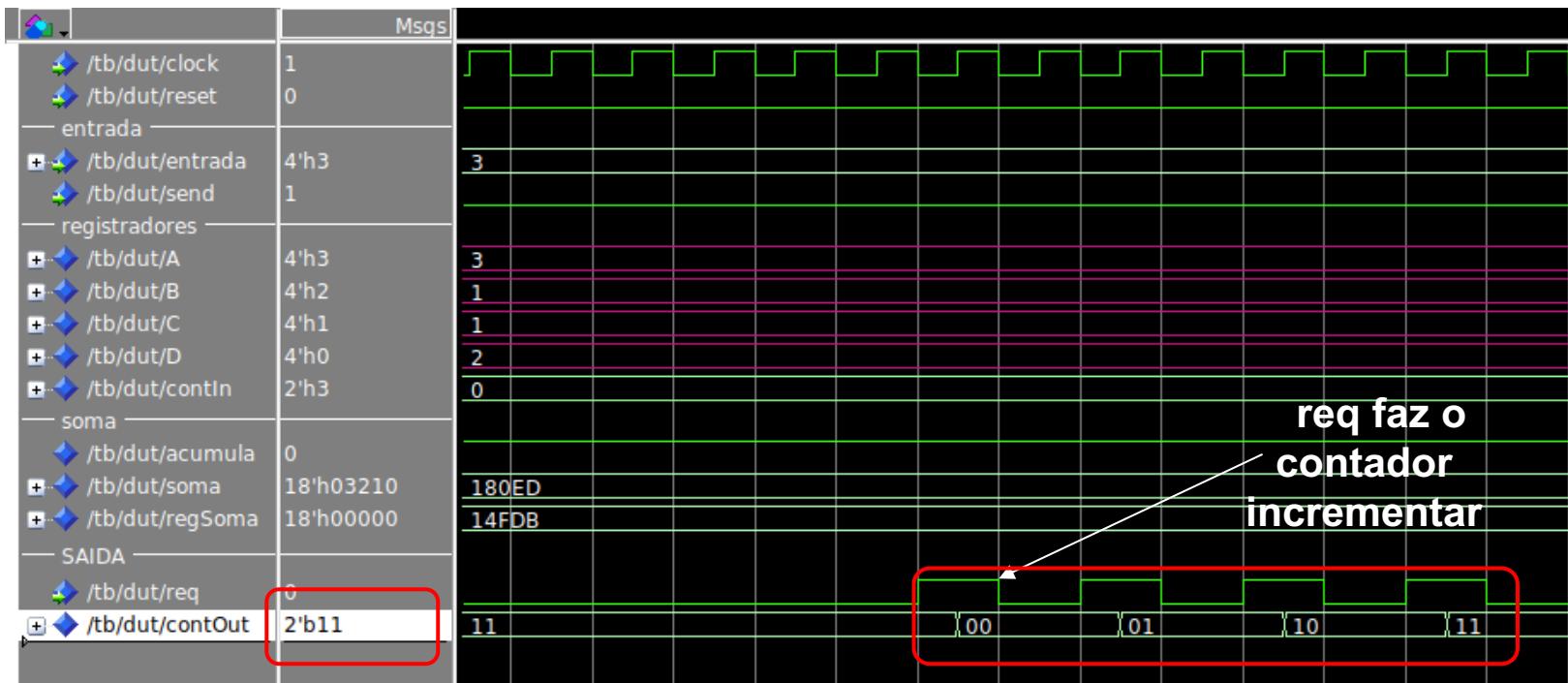
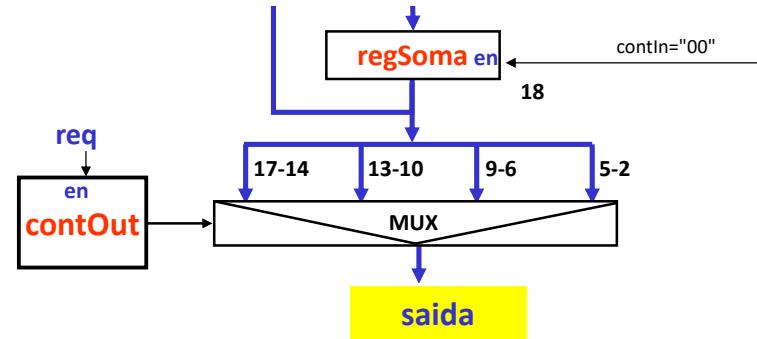
Borda quando o contador
passa para zero

4321
+ 8765
CA86

Contador em função do *request* (4/5)

Contador de 2 bits, *contOut*, incrementado quando *req=1* (*process*)

Cuidado: no reset inicializar em “11” para o primeiro valor ser “00”



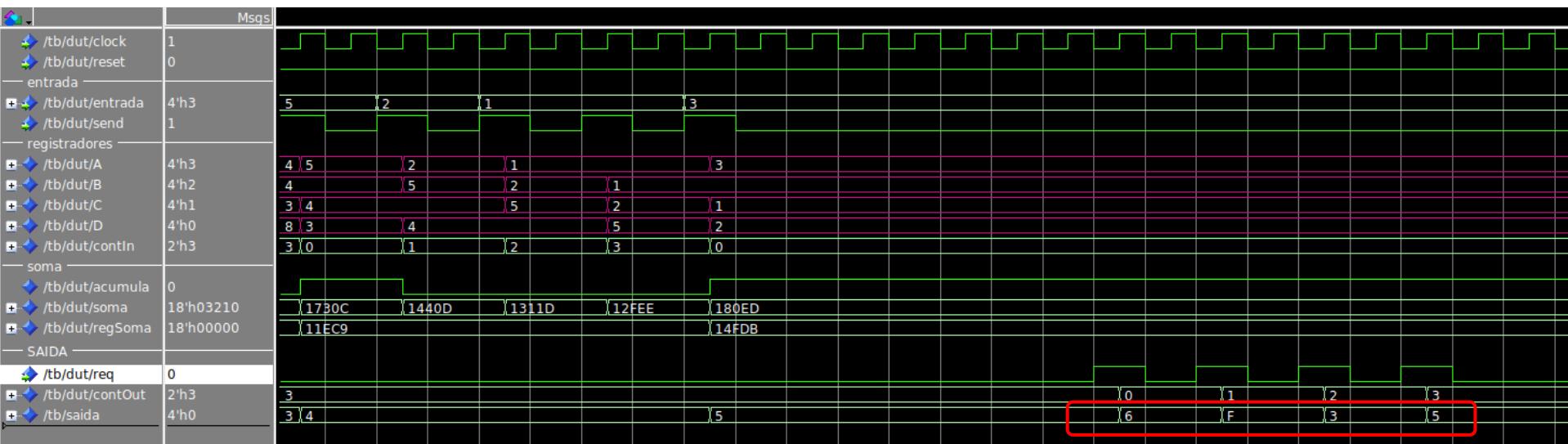
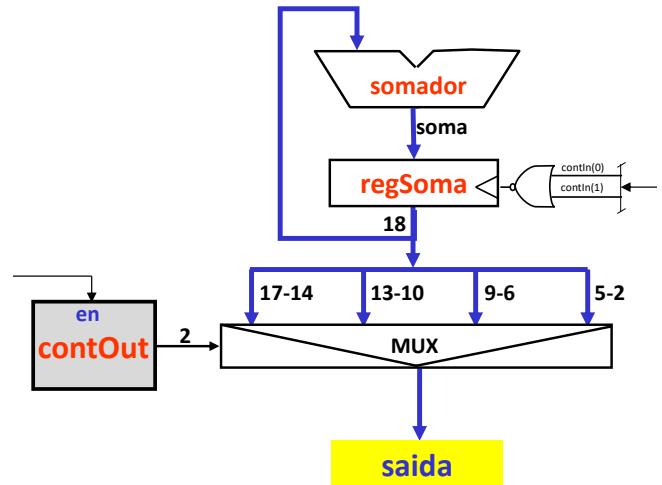
Multiplexador de saída (5/5)

Bloco combinacional

```

saída<= regSoma( 5 downto 2) when contOut="00" else
...
...
regSoma(17 downto 14);

```



$$\text{Média: } 14\text{FDB} / 4 = 53\text{F6}$$