

CIRCUITOS SEQUENCIAIS

Latch, Flip-Flops e Modelagem RTL

Estes slides serão utilizados em 5 aulas

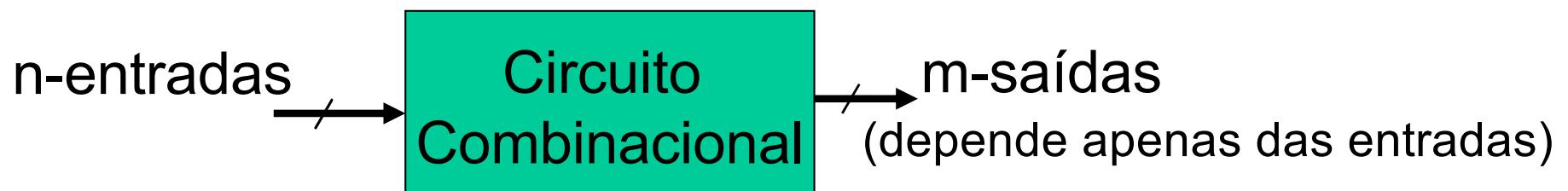
19	08/05/25 5AB	Circuitos Sequenciais. Introdução, latches (armazenamento no nível) e FF mestre-escravo
20	13/05/25 3AB	Circuitos Sequenciais. Modelagem e simulação de latches e FF mestre-escravo en HDL
21	15/05/25 5AB	Circuitos Sequenciais: registradores, contadores, temporizadores (relação tempo e frequência),
22	20/05/25 3AB	Circuitos Sequenciais. Modelagem de sistemas digitais no nível RTL de abstração (1/2)
23	22/05/25 5AB	Circuitos Sequenciais. Modelagem de sistemas digitais no nível RTL de abstração (2/2)

Latches

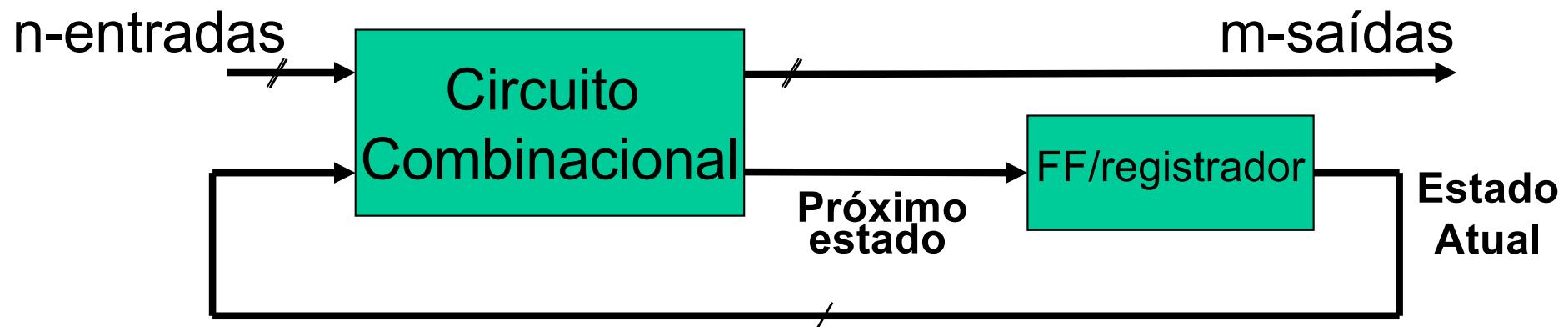
e

Flip-flops MS

Circuitos Combinacionais versus Sequenciais



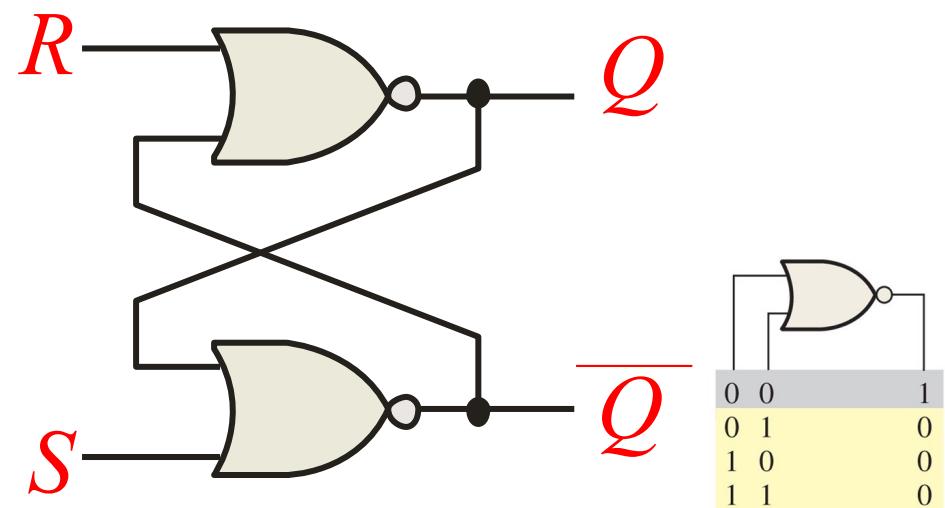
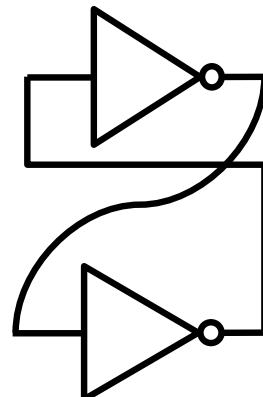
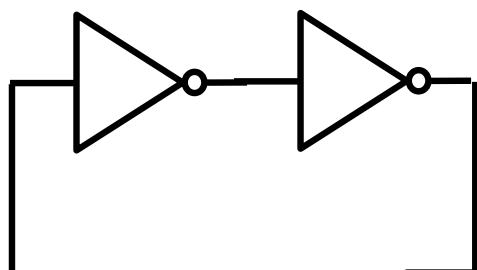
Circuito Combinacional



Circuito Sequencial

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**



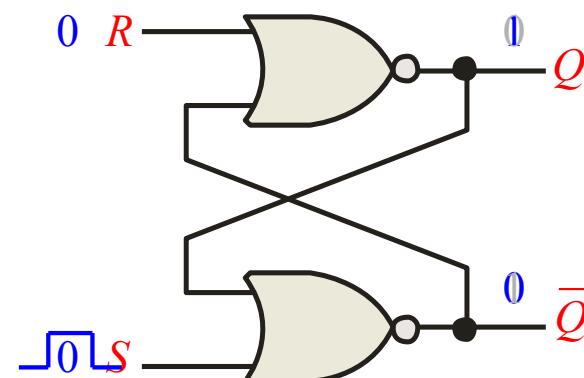
Notar: uma NOR com uma dada entrada em '0' comporta-se como um inversor. 5

LATCH - NOR

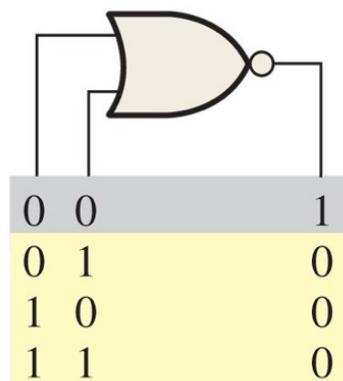
- O **latch com NOR** está em uma condição estável (dado armazenado) quando ambas as entradas estão no nível lógico ZERO

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.



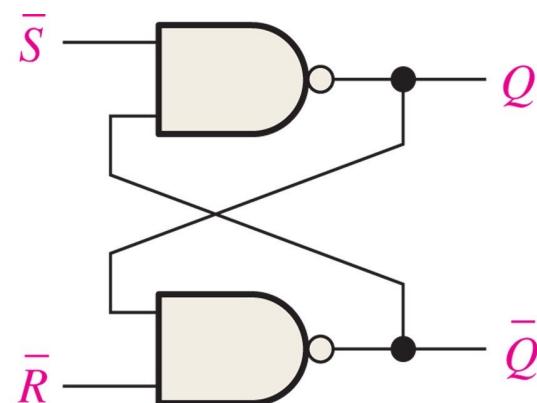
Notar: uma NOR com uma dada entrada em '0' comporta-se como um inversor.

LATCH - NAND

TABLE 7-1

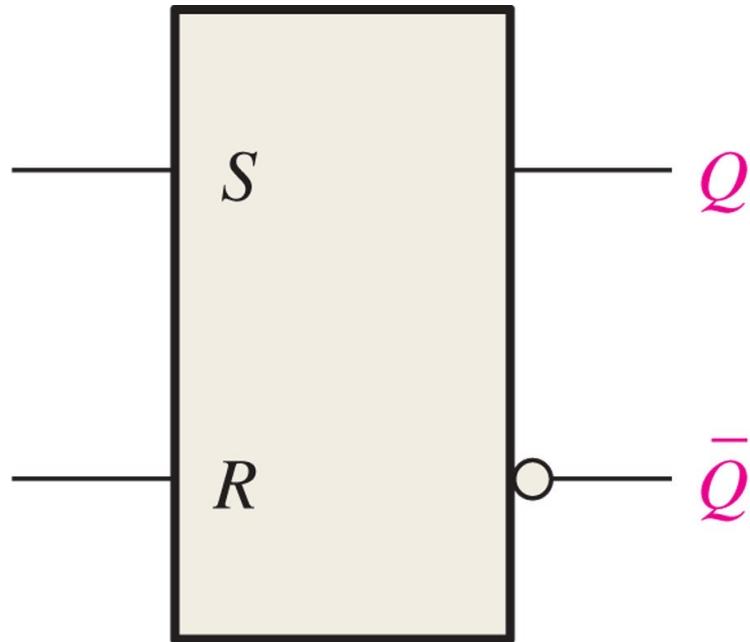
Truth table for an active-LOW input \bar{S} - \bar{R} latch.

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

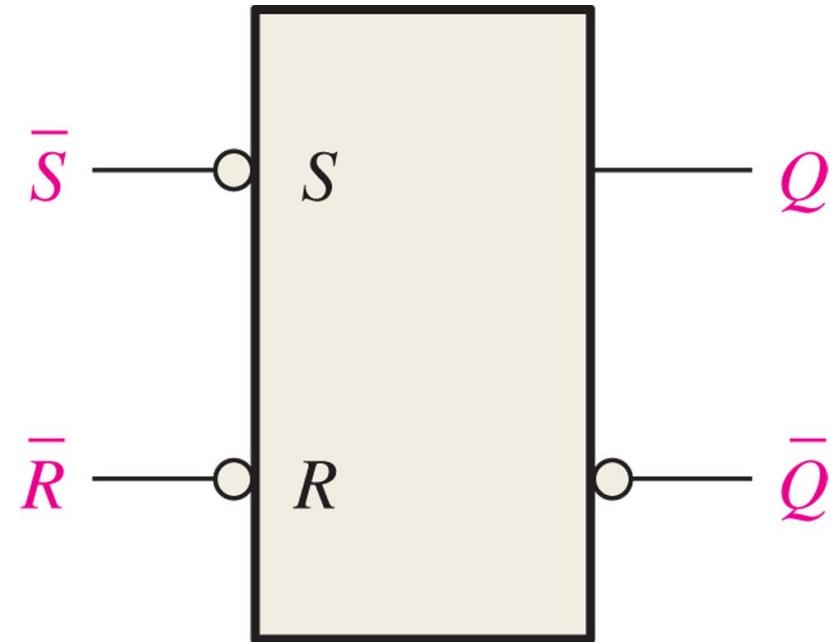


(b) Active-LOW input \bar{S} - \bar{R} latch

Símbolos lógicos para as latches



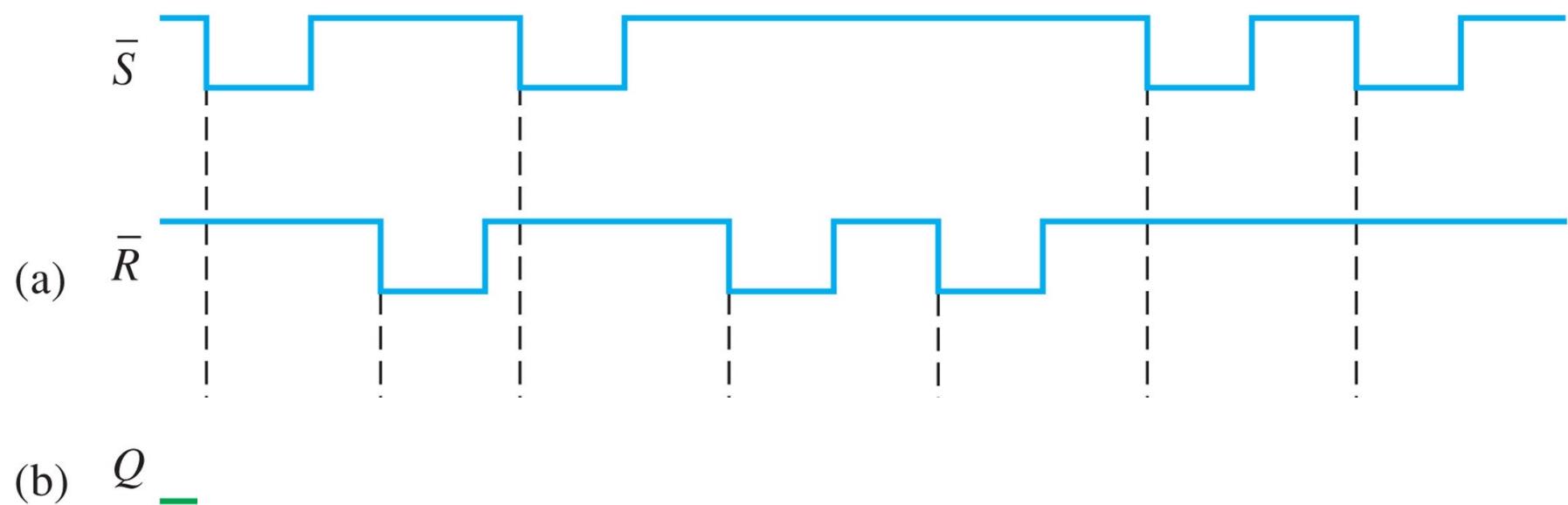
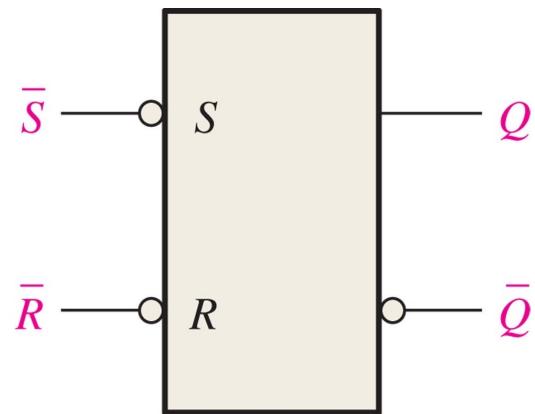
(a) Active-HIGH input
S-R latch



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

FIGURE 7-4

Comportamento da latch



Latch 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.

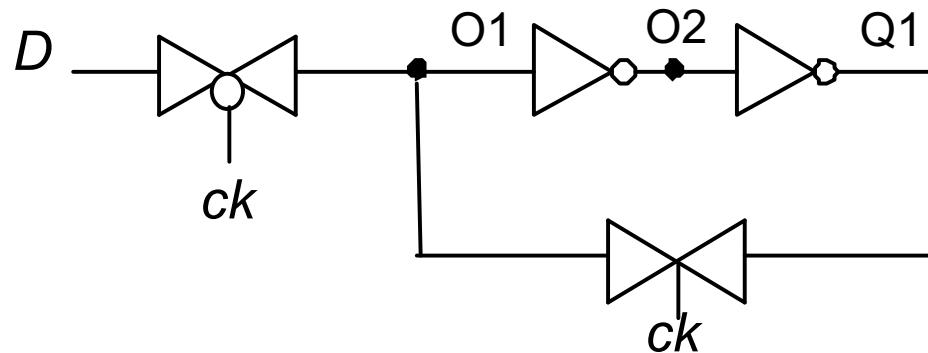
EN: usaremos **clock (clk ou ck)**

Se $ck=1$ a latch esta “**transparente**”

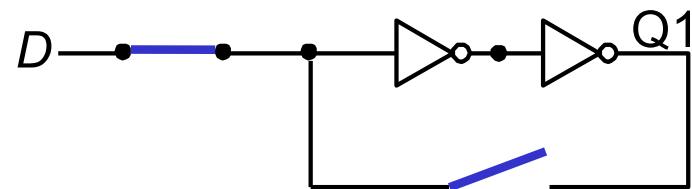
Lach é sensível ao **nível** do ck

Latch D –implementação com chaves

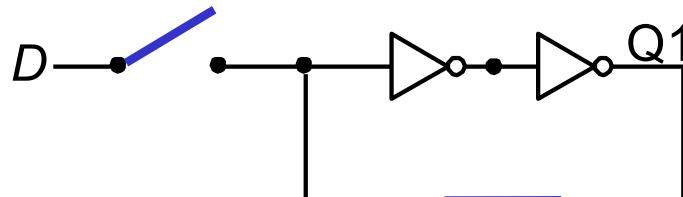
Latch D → bit de memória para caches, por exemplo



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

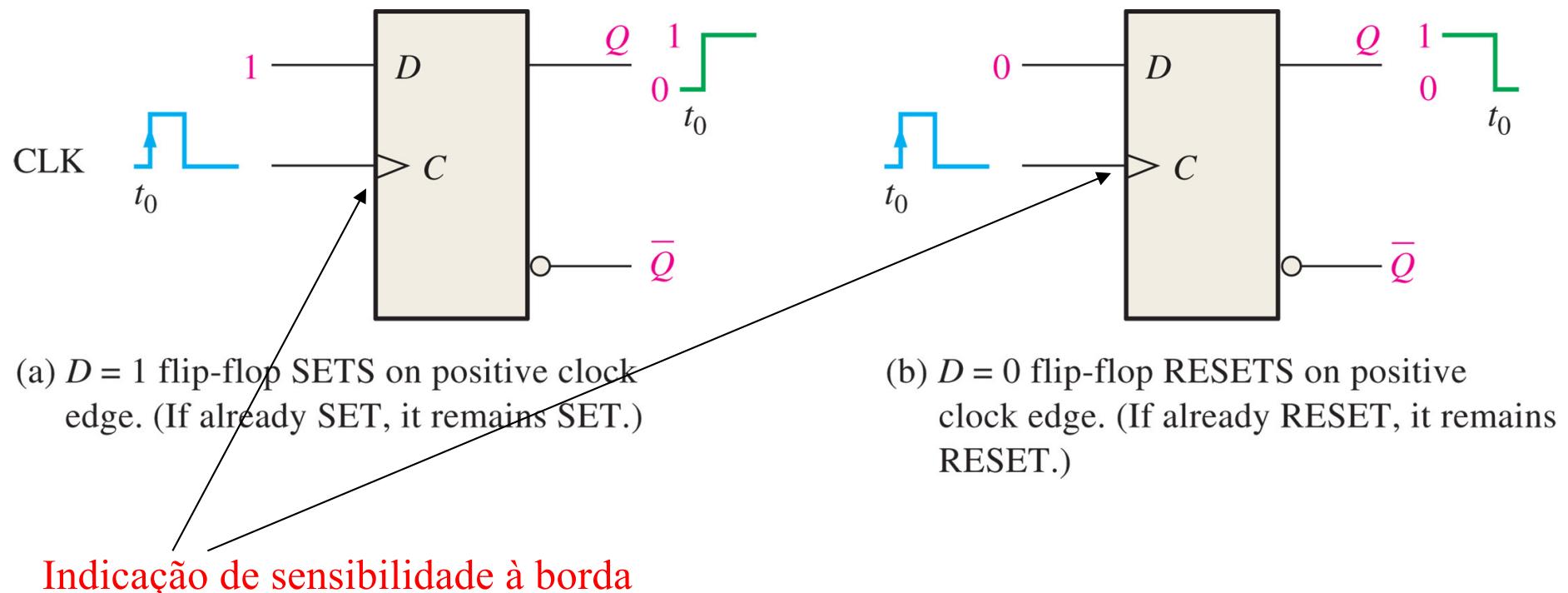


FIGURE 7-14 Operation of a positive edge-triggered D flip-flop.

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

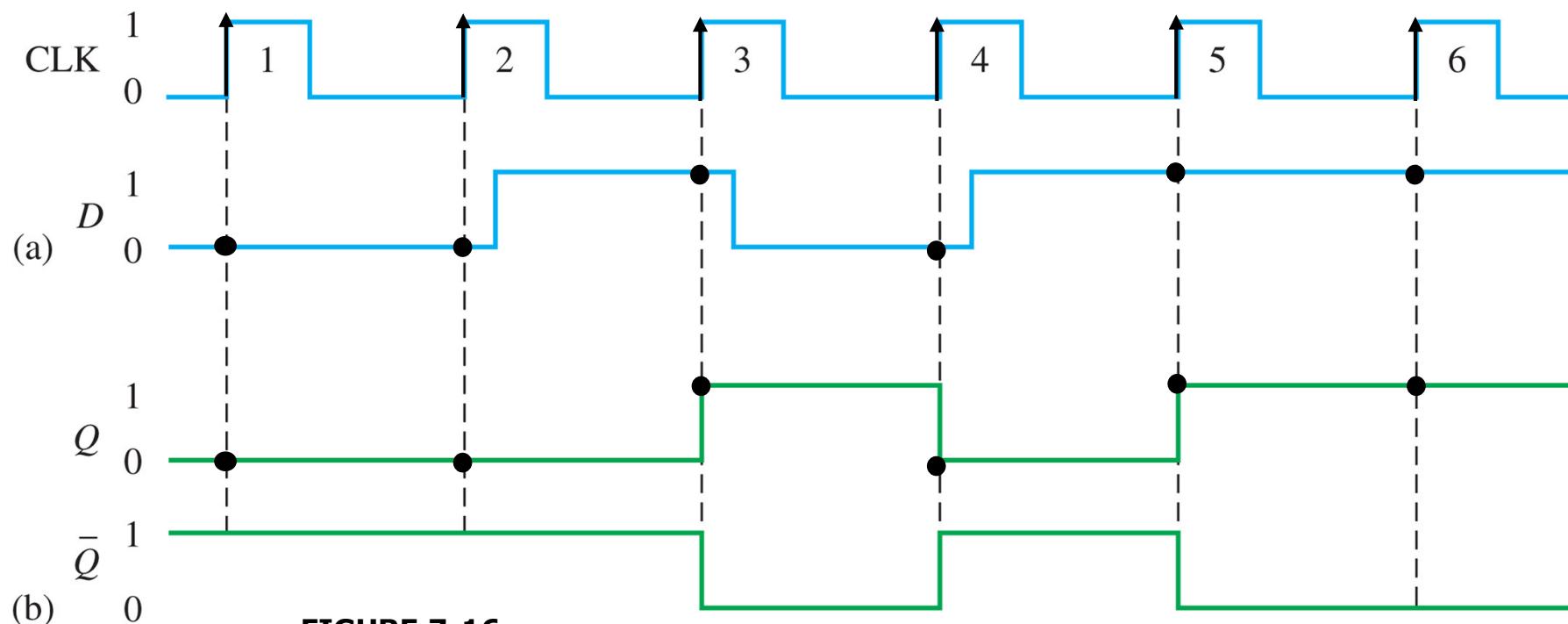
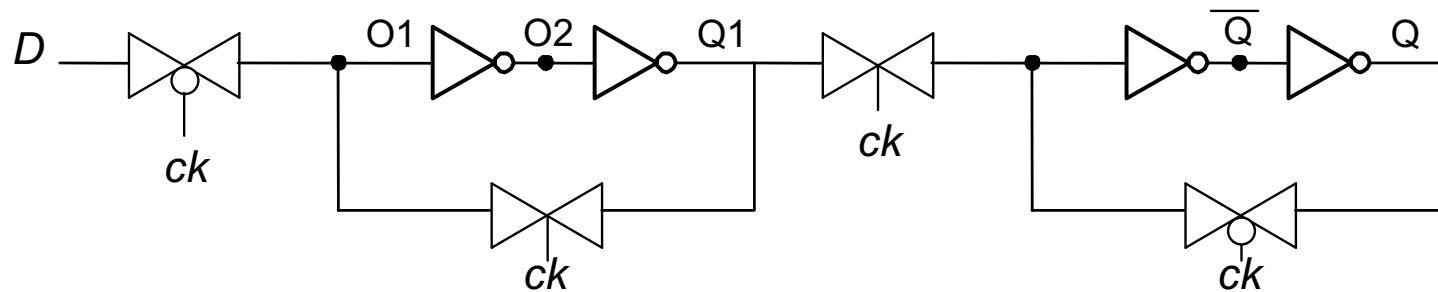


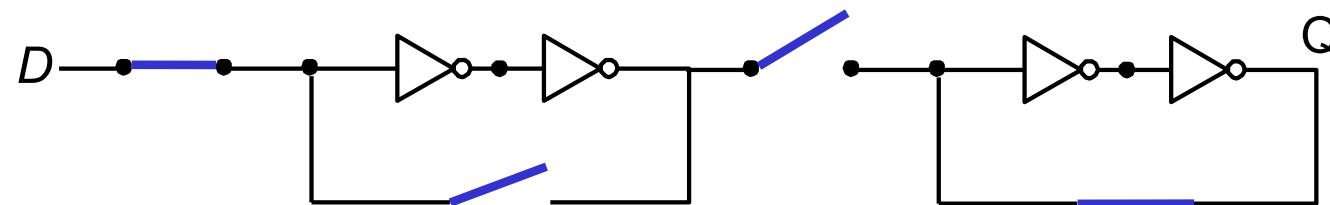
FIGURE 7-16

FFD – Mestre Escravo (implementação com chaves)

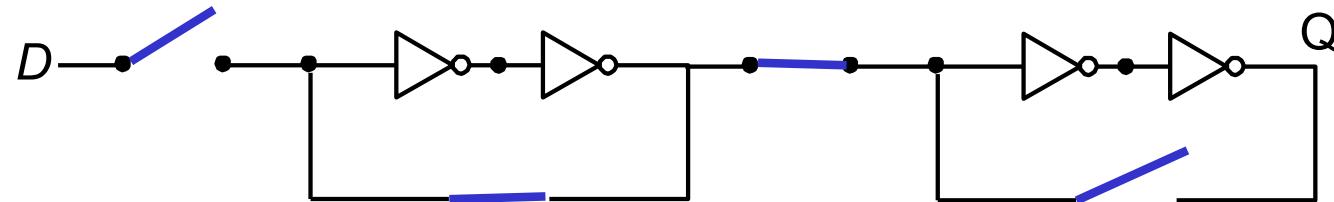
- Implementação utilizando chaves



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



CK sobe \rightarrow Informação transferida para a saída

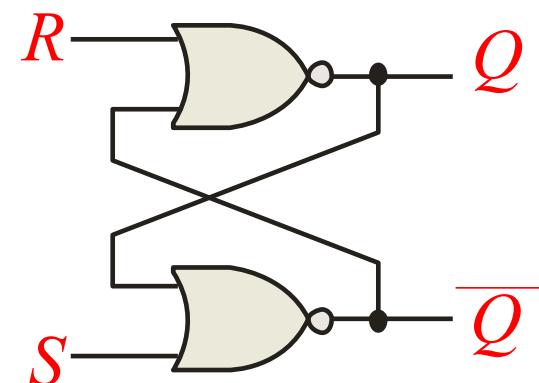


LATCH S-R EM SV

```
module sr_latch (
    input logic S, R,
    output logic Q, nQ
);

    always_comb begin
        Q = ~(R | nQ);
        nQ = ~(S | Q);
    end

endmodule
```



LATCH D EM SV

```
module latch #(parameter int N = 16)
( input logic clk,
  input logic rst,
  input logic ce,
  input logic [N-1:0] D,
  output logic [N-1:0] Q
);
```

```
  always_latch begin
    if (rst)
      Q = '0;
    else if (clk && ce)
      Q = D;
  end
endmodule
```

O comando **always_latch** é usado para descrever um elemento de memória que **lê a entrada enquanto uma condição de controle** (como o clock) **está ativa**.

FLIP-FLOP D MESTRE-ESCRAVO EM SV

```
module ffd_ms #(parameter int N = 16)
( input logic clk,
  input logic rst,
  input logic ce,
  input logic [N-1:0] D,
  output logic [N-1:0] Q
);
  always_ff @ (posedge clk or posedge rst) begin
    if (rst)
      Q <= '0;
    else if (ce)
      Q <= D;
  end
endmodule
```

Define a **sensibilidade do bloco**:

- **posedge rst** → reage na **borda de subida** do sinal de reset (ou seja, assim que rst se torna 1). Isso define um **reset assíncrono**: o bloco será executado imediatamente quando rst subir, independentemente do clock.
- **posedge clk** → reage na **borda de subida** do sinal de clock.

@: **event control**. Execute o bloco sempre que ocorrer **um evento** nos sinais listados na sensibilidade.

FLIP-FLOP D MESTRE-ESCRAVO EM SV

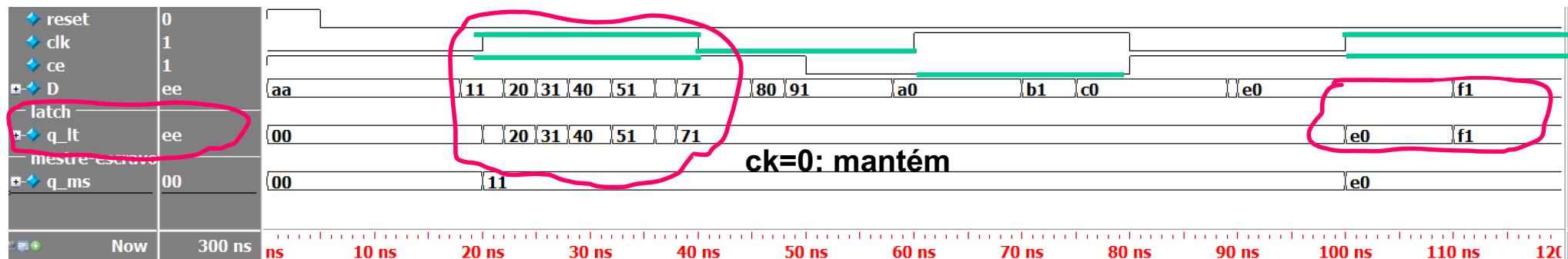
```
module ffd_ms #(parameter int N = 16)
( input logic clk,
  input logic rst,
  input logic ce,
  input logic [N-1:0] D,
  output logic [N-1:0] Q
);

  always_ff @(posedge clk or posedge
    if (rst)
      Q <= '0;
    else if (ce)
      Q <= D;
  end
endmodule
```

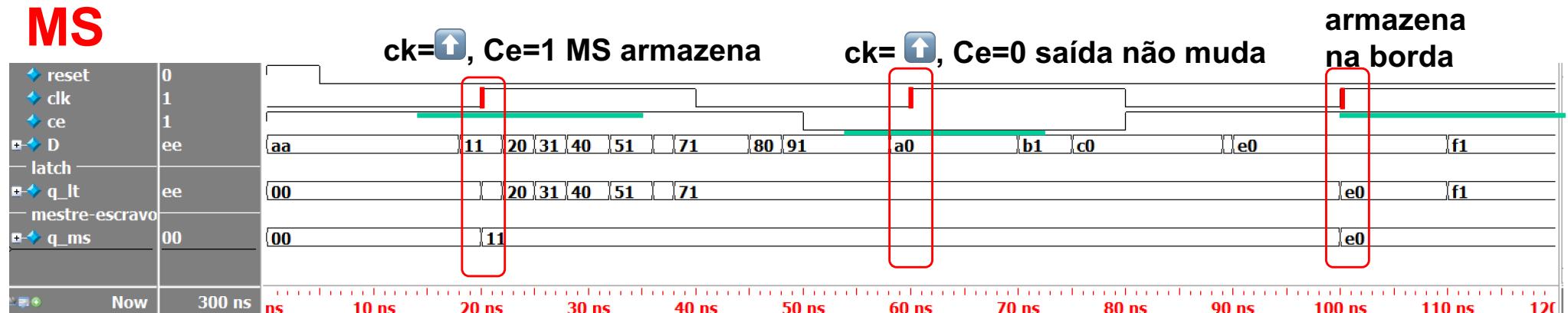
- Em SystemVerilog, a atribuição **não-bloqueante** `<=` é usada em lógica sequencial (dentro de `always_ff`). Ela permite que múltiplas atribuições ocorram de **forma paralela em um mesmo ciclo de clock**.
- Por outro lado, **a atribuição = (bloqueante)** é usada dentro de blocos `always_comb`, que modelam lógica combinacional, garantindo que a execução siga ordem sequencial dentro do bloco.

LATCH e MS – diferença de comportamento

LATCH



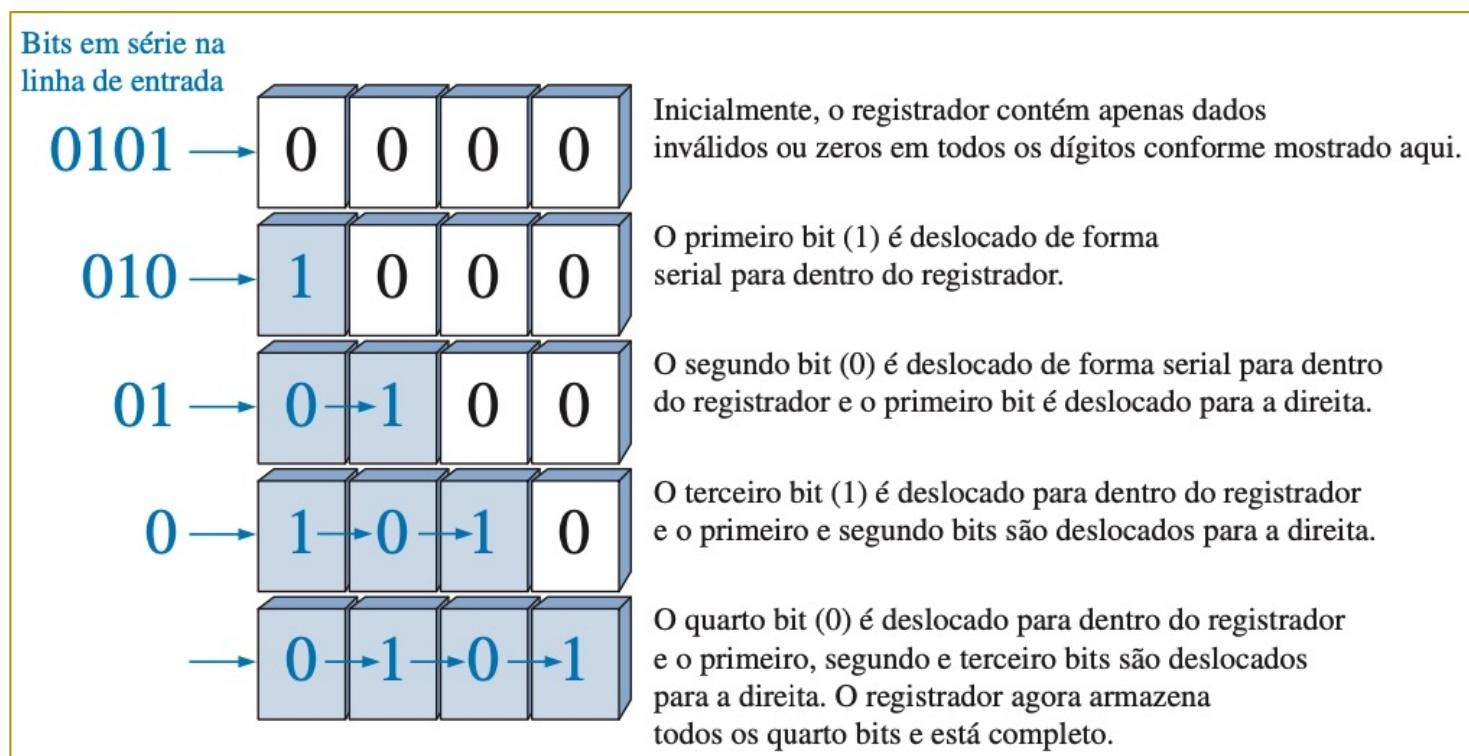
MS



REGISTRADOR

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

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



REGISTRADOR

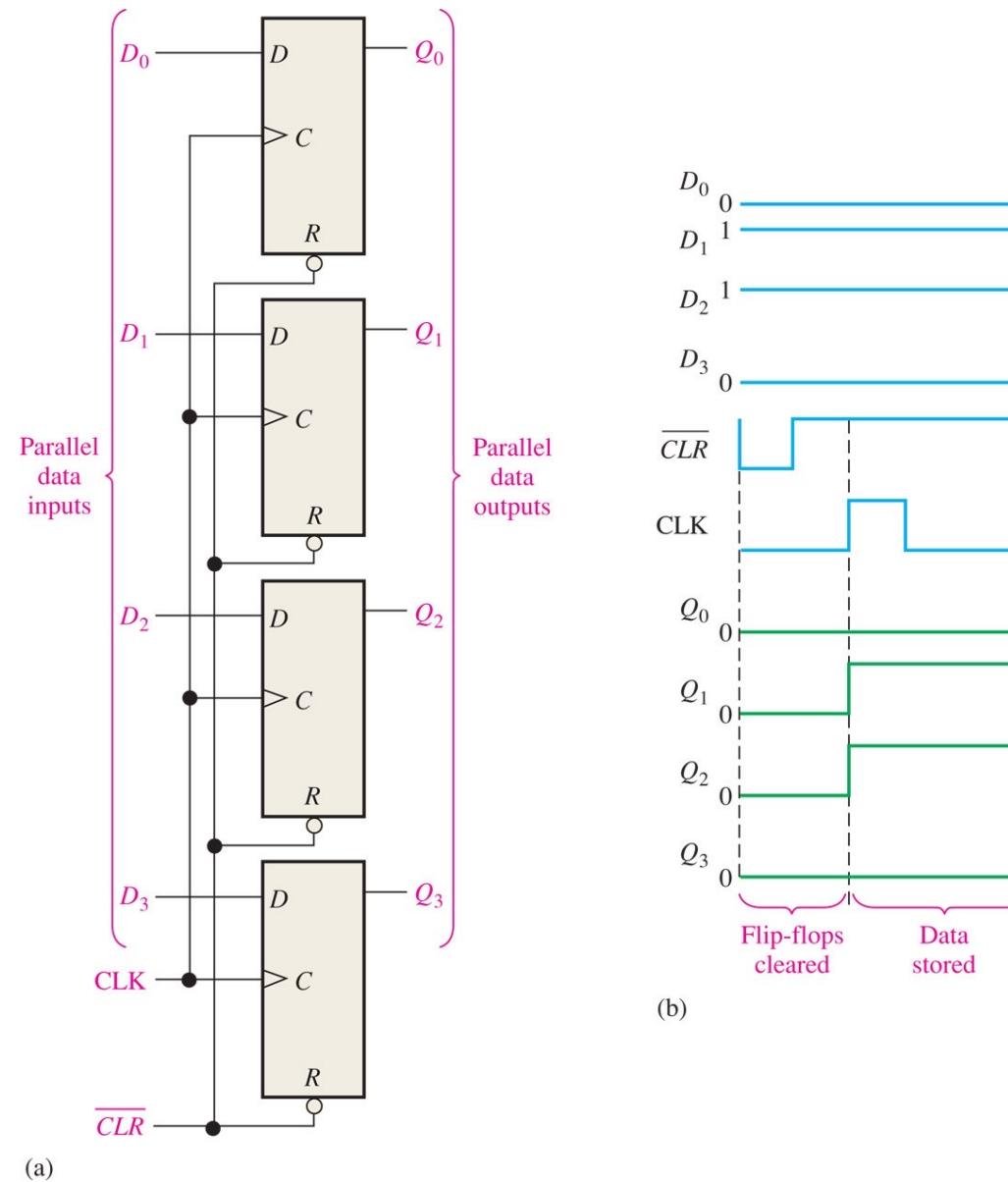


FIGURE 7-35 Example of flip-flops used in a basic register for parallel data storage.

REGISTRADOR

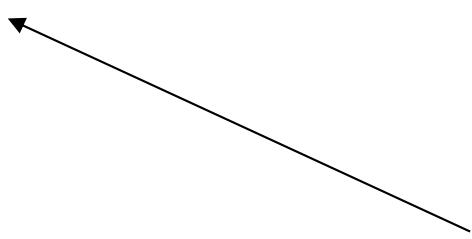
- registradores são sinais declarados em `always_ff` com sinal de sincronismo (exemplo: *clock*)
- para efeito de síntese e simulação, é aconselhável introduzir um reset assíncrono
- ce: *enable*

```
always_ff @(posedge clock or posedge reset)
begin
    if (reset)
        reg <= '0; // zera reg
    else if (ce)
        reg <= barramento_A;
end
```

@: **event control.** Execute o bloco sempre que ocorrer **um evento** nos sinais listados na sensibilidade.

REGISTRADOR

- Registrador com largura de palavra parametrizável

```
module regnbit #(parameter int N = 16)   
  (input logic ck,  
   input logic rst,  
   input logic ce,  
   input logic [N-1:0] D,  
   output logic [N-1:0] Q  
);  
  
  always_ff @ (posedge ck or posedge rst) begin  
    if (rst)  
      Q <= '0; // zera  
    else if (ce)  
      Q <= D;  
  end  
endmodule
```

Parameter N
define o
número de bits
do registrador

Uso:

```
regnbit #(.N(8)) rx (.ck(ck), .rst(rst),  
                      .ce(ce), .D(D), .Q(Q));
```

```
module tb;
    logic ck = 0;
    logic reset, ce;
    logic [7:0] D, Q1;

    regnbit #(.N(8)) rx (.ck(ck), .rst(reset), .ce(ce), .D(D), .Q(Q1));

    always #5 ck = ~ck;

    initial begin
        reset = 1;
        #4 reset = 0;
    end

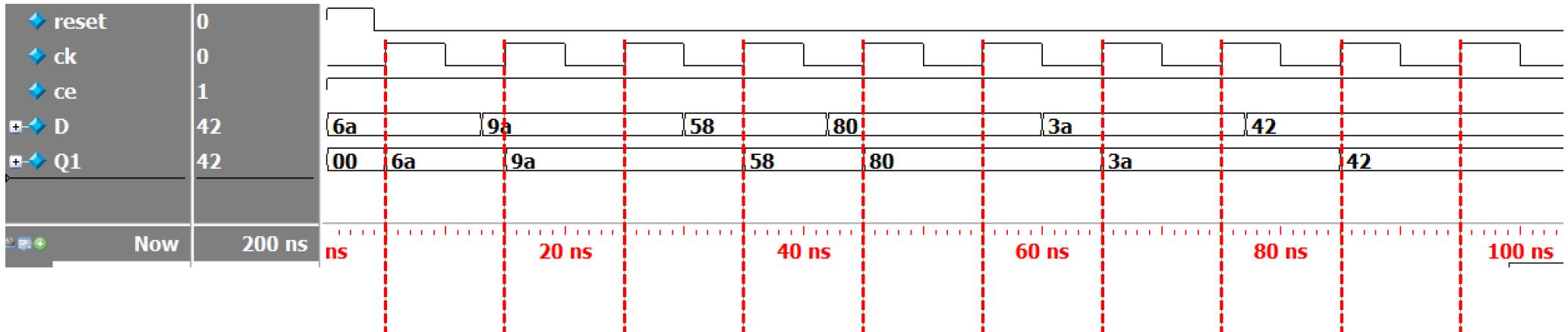
    initial begin // Enable sempre ativo
        ce = 1;
    end

    // Estímulo com valores e tempos aleatórios entre 8'h00 e 8'hAA
    initial begin
        int unsigned i;
        for (i = 0; i < 20; i++) begin
            D = $urandom_range(8'h00, 8'hAA);
            #($urandom_range(5, 20)); // intervalos random entre 5 ns e 20 ns
        end
        $finish;
    end

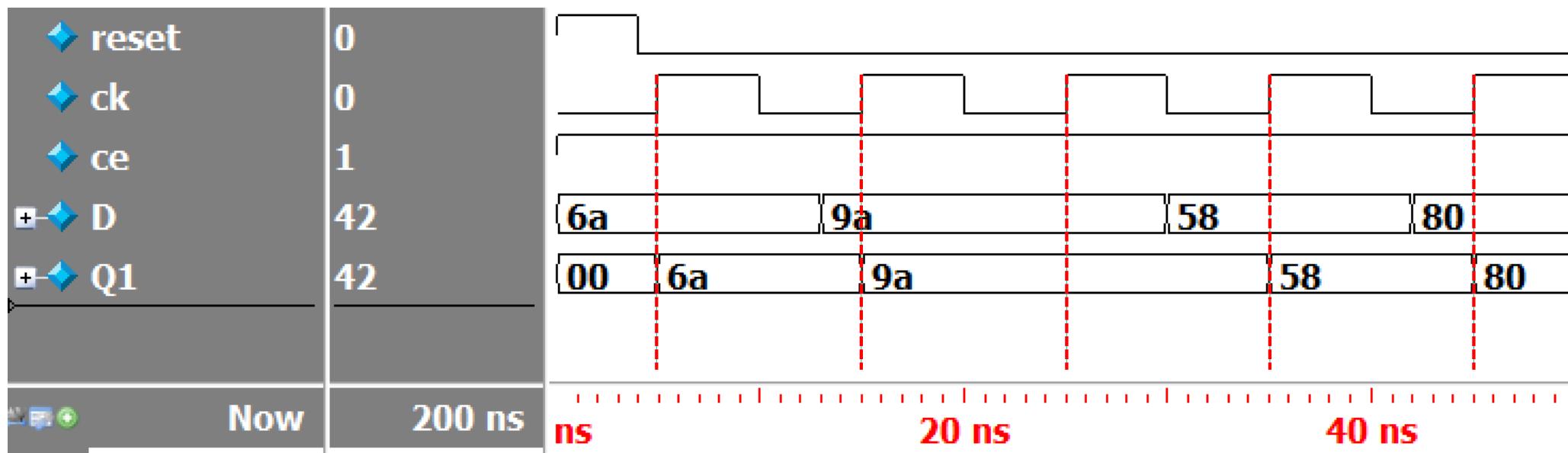
endmodule
```

Simulando o Registrador

Simulando o Registrador



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



REGISTRADOR DE DESLOCAMENTO

Exemplo de registrador de deslocamento:

```
module shift_reg #(parameter int N = 16)
  ( input logic ck, rst,
    input logic [N-1:0] in,
    output logic [N-1:0] out
  );

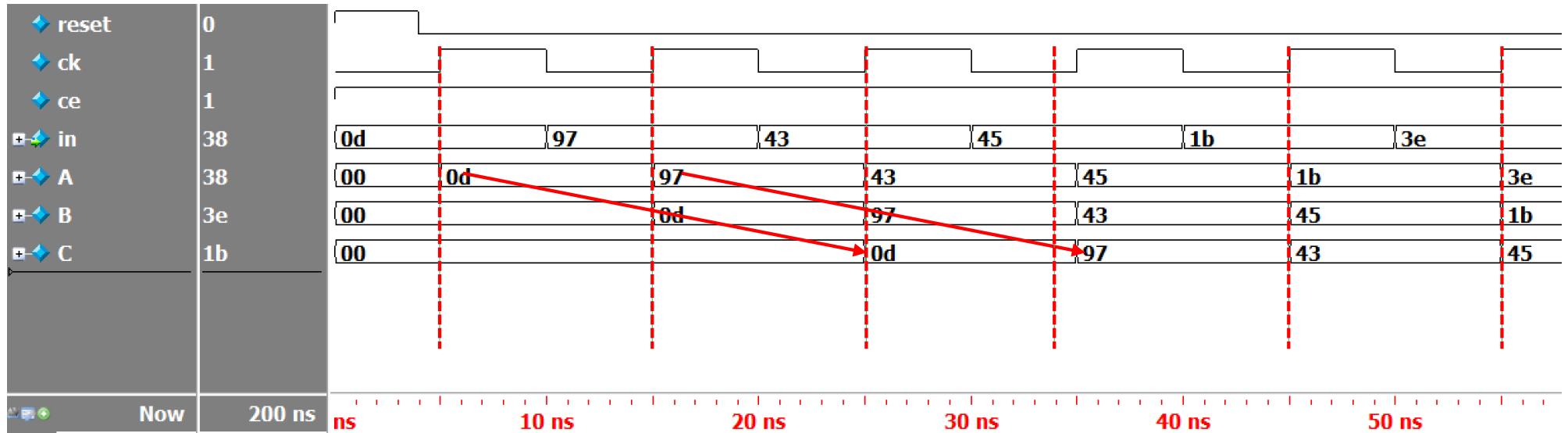
  logic [N-1:0] A, B, C;

  assign out = C;

  always_ff @ (posedge ck or posedge rst) begin
    if (rst) begin
      A <= '0;
      B <= '0;
      C <= '0;
    end
    else begin
      A <= in;
      B <= A;
      C <= B;
    end
  end
endmodule
```

- 1) Desenhe este 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



```
always_ff @(posedge ck or posedge rst) begin
    if (rst) begin
        A <= '0;
        B <= '0;
        C <= '0;
    end
    else begin
        A <= in;
        B <= A;
        C <= B;
    end
end
```

REGISTRADOR DE DESLOCAMENTO

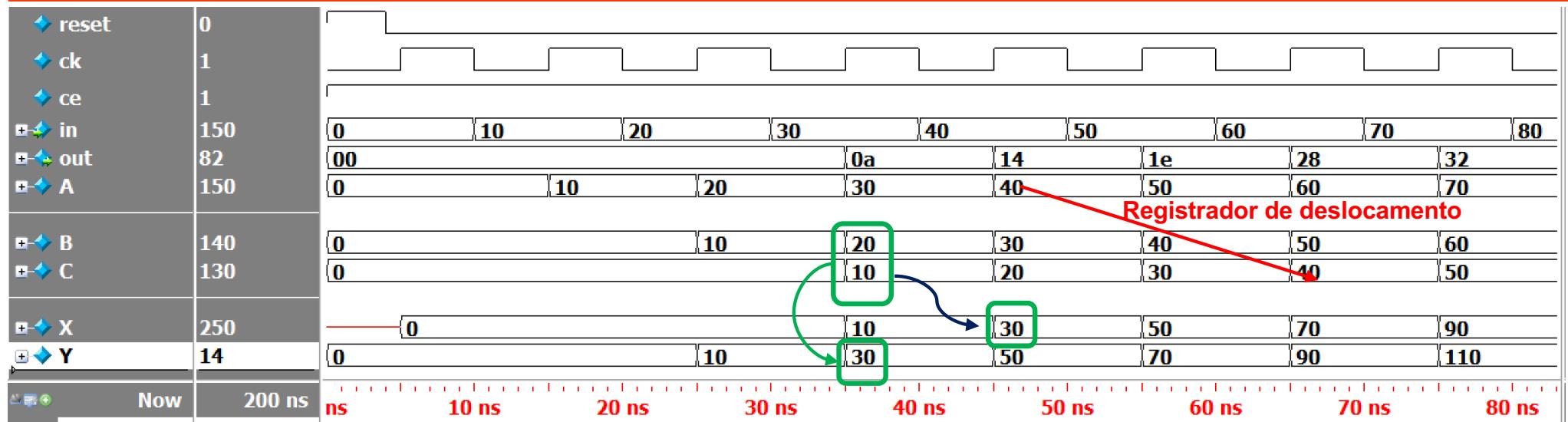
- Atribuição dentro/fora de *always_ff*

```
always_ff @(posedge ck or posedge rst) begin
    if (rst) begin
        A <= '0;
        B <= '0;
        C <= '0;
    end
    else begin
        A <= in;
        B <= A;
        C <= B;
        X <= B + C;
    end
end
assign Y = B + C;
```

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

- sinais atribuídos em *always_ff* são sempre flip-flops/registradores
- Sinais com lógica combinacional (*assign*) representam o valor atual

REGISTRADOR DE DESLOCAMENTO – SIMULAÇÃO



```

always_ff @(posedge ck or posedge rst) begin
    if (rst) begin
        A <= '0;
        B <= '0;
        C <= '0;
    end
    else begin
        A <= in;
        B <= A;
        C <= B;
        X <= B + C;
    end
end
assign Y = B + C;

```

REGISTRADOR – formas de entrada de dados

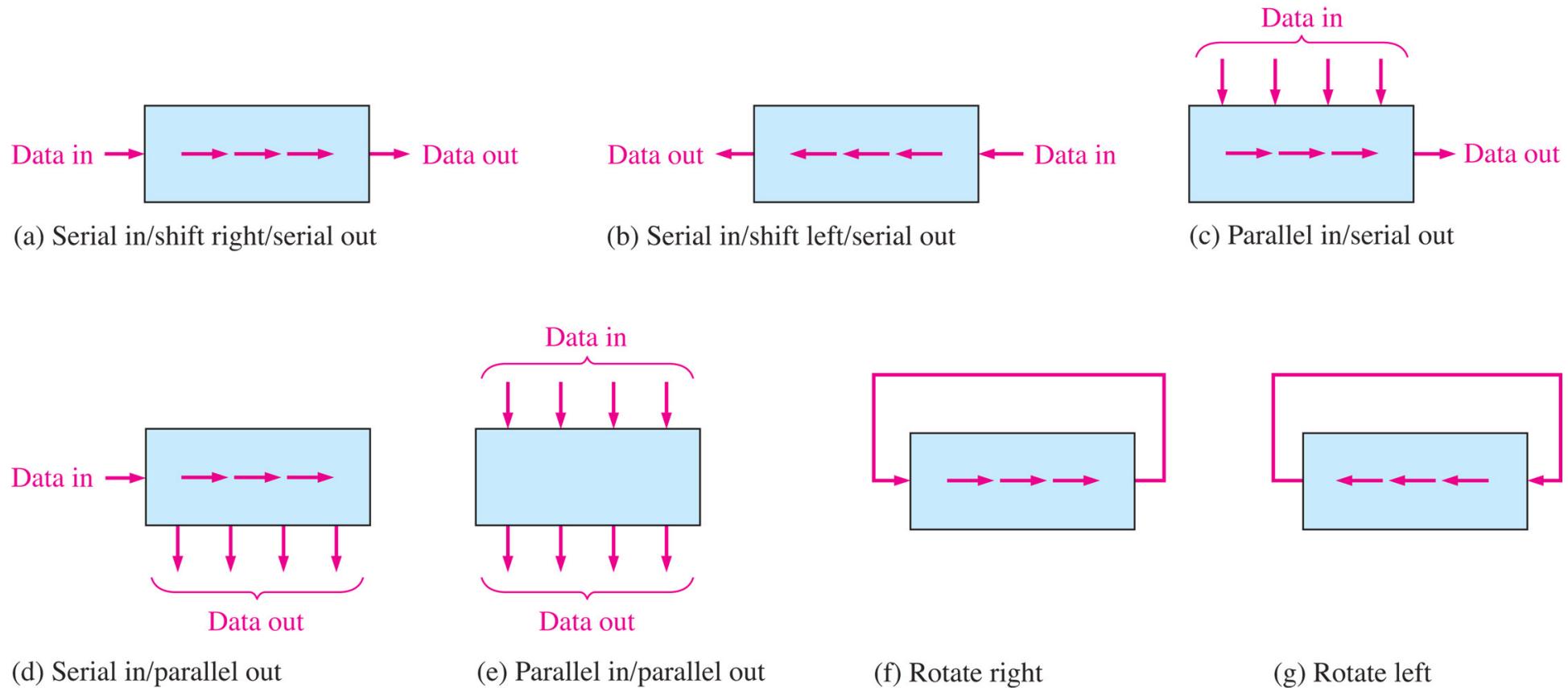


FIGURE 8-2 Basic data movement in shift registers. (Four bits are used for illustration. The bits move in the direction of the arrows.)

REGISTRADOR – carga serial

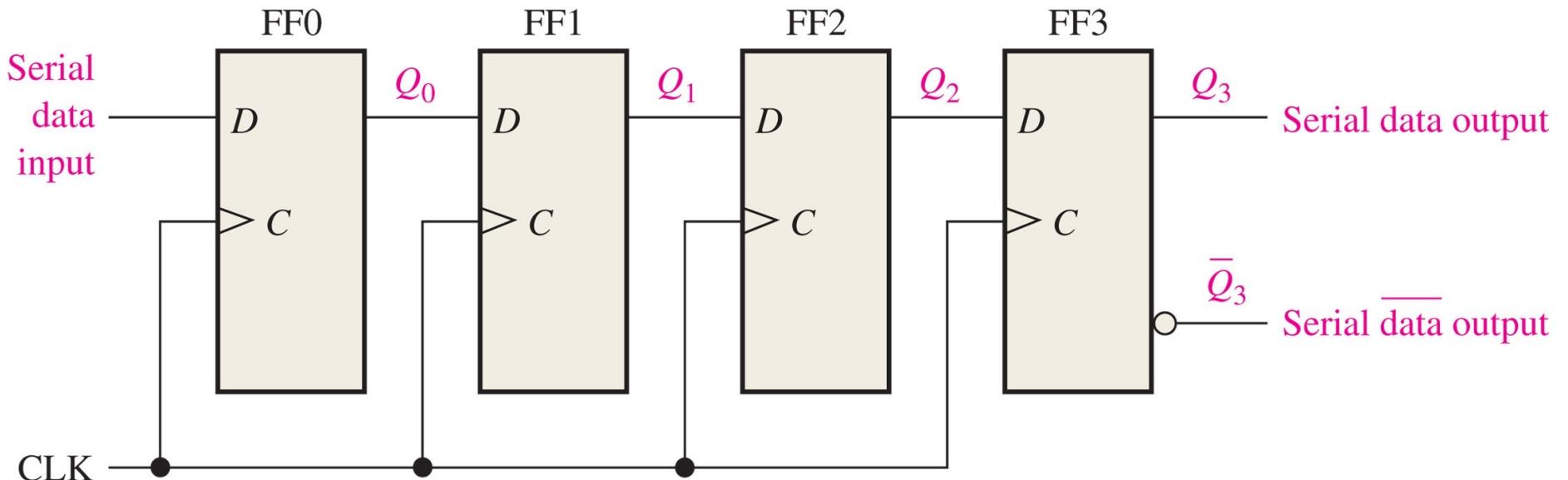


TABLE 8–1

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

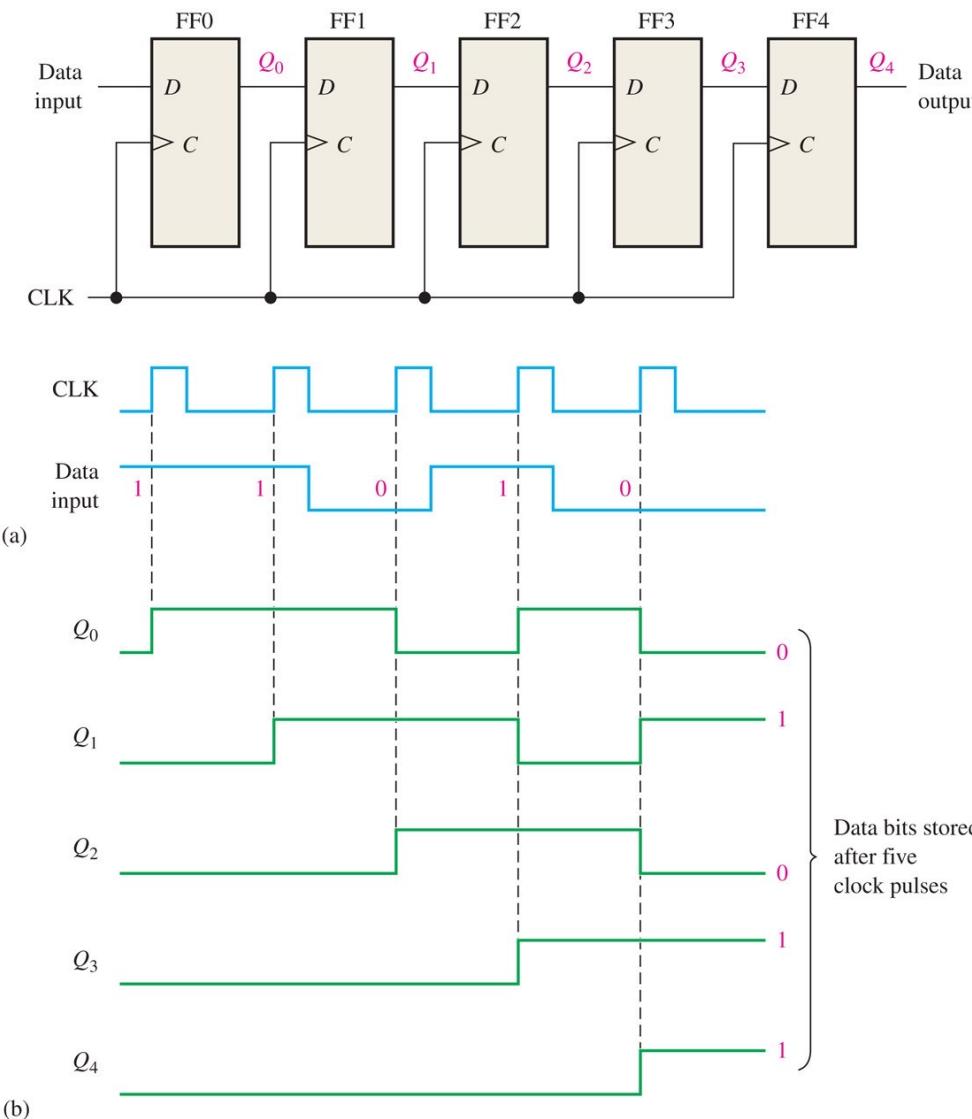
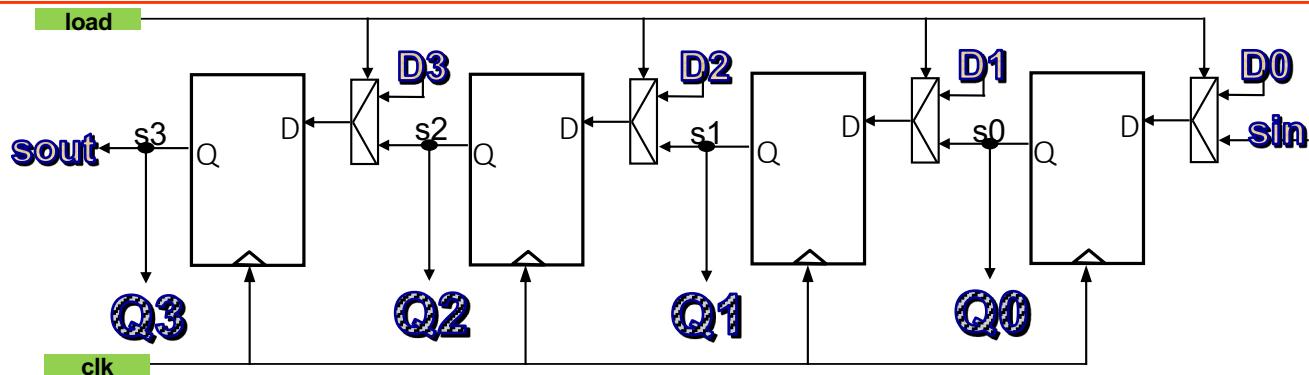


FIGURE 8-4

REGISTRADOR CARGA PARALELA E SERIAL

PARA
4 BITS



```
module serial_in #(parameter int N = 8)
( input logic clk, reset, load, sin,
  input logic [N-1:0] d,
  output logic [N-1:0] q, saída paralela
  output logic sout      saída serial
);

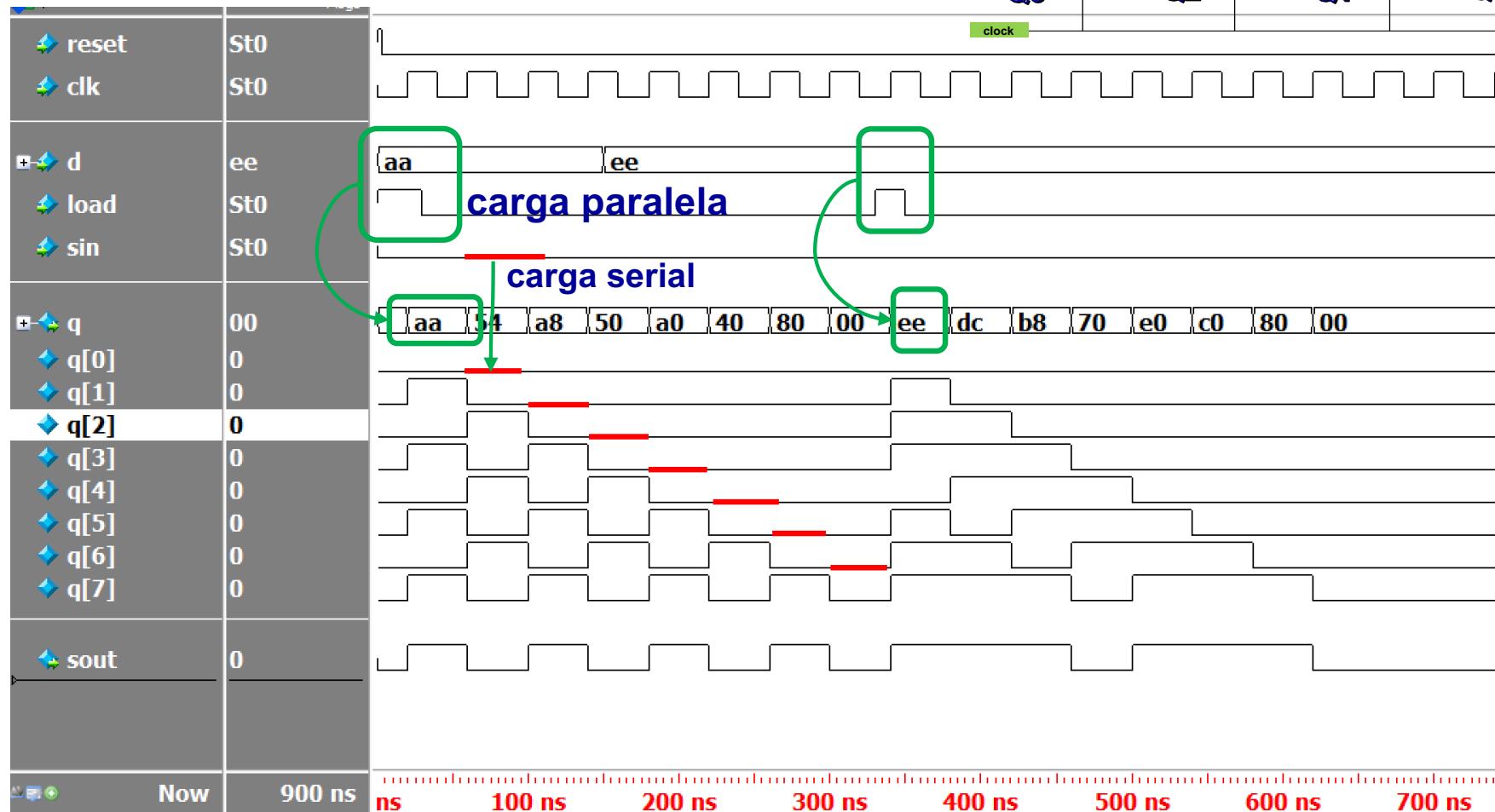
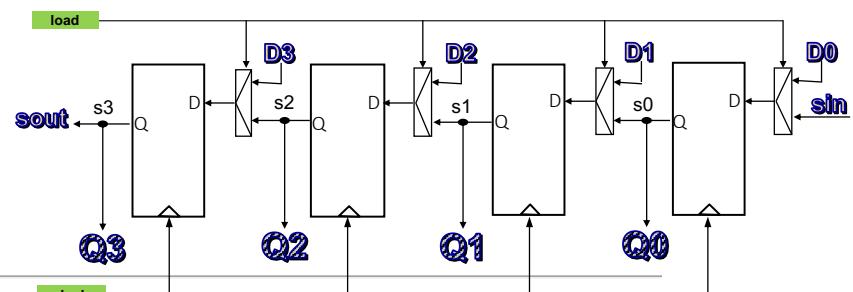
  always_ff @(posedge clk or posedge reset) begin
    if (reset)
      q <= '0;
    else
      q <= load ? d : {q[N-2:0], sin}; // multiplexador
  end

  assign sout = q[N-1]; // MSB

endmodule
```

REGISTRADOR CARGA PARALELA E SERIAL

**Deslocamento à esquerda multiplica por 2
AA → (1) 54 → A8 → (1)50 →A0**



AA **54** **A8**
1010 1010 **0101 0100** **1010 1000**

CONTADOR– 2ⁿ estados

```
module contup (
    input logic clock,reset, Load, Enable,
    input logic [5:0] DATABUS,
    output logic [5:0] Upcount
);

    always_ff @(posedge clock or posedge reset) begin
        if (reset)
            Upcount <= '0;
        else if (Enable) begin
            Upcount <= Load ? DATABUS : Upcount + 1;
        end
    end
endmodule
```

(1) Determine o comportamento deste contador, fazendo um diagrama de tempos.

(2) Como modificar o contador para realizar contagem crescente/decrescente?

```
else if (Enable)
    Upcount <= Load ? DATABUS :
    UpDown ? Upcount + 1 :
    Upcount - 1;
```

Exercício de Simulação 1

- Baixe do Moodle o arquivo **sequenciais_sv.zip** →
 - Em “material de apoio” → “Material de apoio a SV” → **sequenciais_sv.zip**
- Abra o arquivo **ex1.sv** 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?

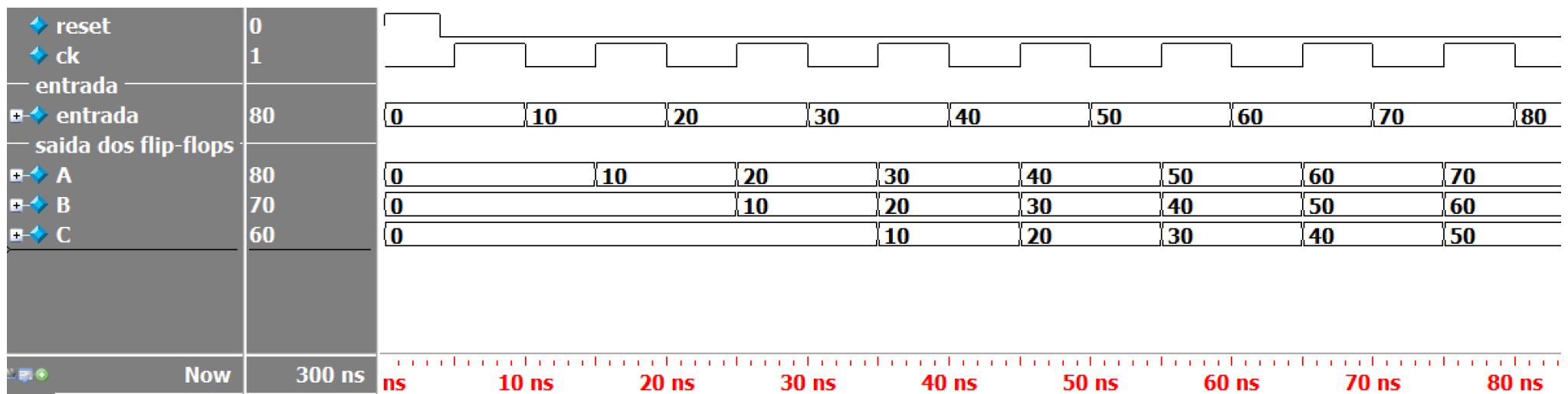


Material de Apoio Simulação SystemVerilog (SV)
[Material de Apoio para Simulação de Flip-Flops e RTL](#) ←
[Material de Apoio para Simulação de FSMs](#)
[Material de Apoio para a Simulação da NanoCPU](#)

```
sequenciais_sv/
  └── ex1
      ├── ex1.sv
      └── sim.do
  └── ex2
      ├── ex2.sv
      └── sim.do
  └── ex3
      ├── ex3.sv
      └── sim.do
  └── ex4
      ├── ex4_tb.sv
      ├── ex4.sv
      └── sim.do
  └── ex5
      ├── ex5_tb.sv
      ├── ex5.sv
      └── sim.do
      └── wave.do
  └── ex6
      ├── ex6_tb.sv
      ├── ex6.sv
      └── sim.do
      └── wave.do
  └── ex7
      ├── ex7_tb.sv
      ├── ex7.sv
      └── sim.do
      └── wave.do
  └── ex8
      ├── ex8_tb.sv
      ├── ex8.sv
      └── sim.do
      └── wave.do
```

Exercício de Simulação 2

- Abra o arquivo **ex2.sv** 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?



Exercício de Simulação 2b

- No arquivo `ex2.sv` faça as seguintes modificações `shift_reg`
 - Acrescente os sinais, X e Y, de mesma largura dos registradores
 - Acrescente as atribuições dentro e fora do `always_ff`.

```
always_ff @(posedge ck or posedge rst) begin
    if (rst) begin
        A <= '0;
        B <= '0;
        C <= '0;
    end
    else begin
        A <= in;
        B <= A;
        C <= B;
        X <= B + C;
    end
end
assign Y = B + C;
```

- Simule o arquivo modificado com as novas somas
 - Explique o comportamento observado nos sinais X e Y

Ver resposta na lâmina 29

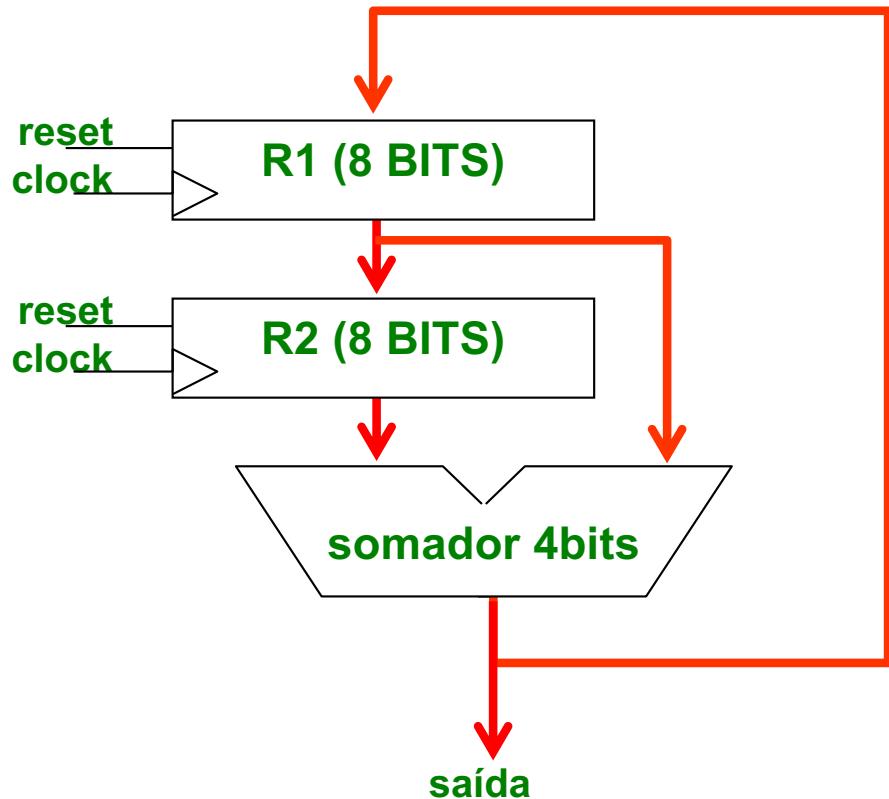
Projeto RTL

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 **SystemVerilog**
- 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.



```
module ex3 (
    input logic clock,
    input logic reset,
    output logic [7:0] saída
);

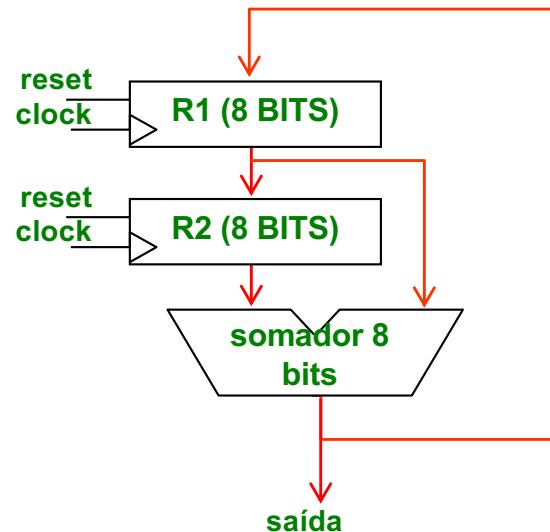
logic [7:0] R1, R2;

always_ff @ (posedge clock or posedge reset) begin
    if (reset) begin
        R1 <= 8'h01;
        R2 <= '0;
    end else begin
        R1 <= saída;
        R2 <= R1;
    end
end

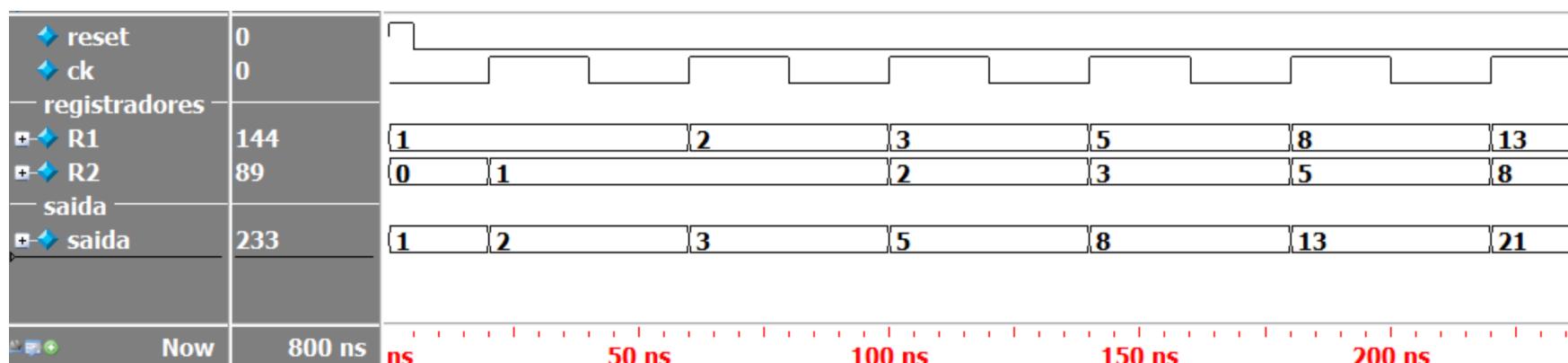
assign saída = R1 + R2;

endmodule
```

EXERCÍCIO DE SIMULAÇÃO 3



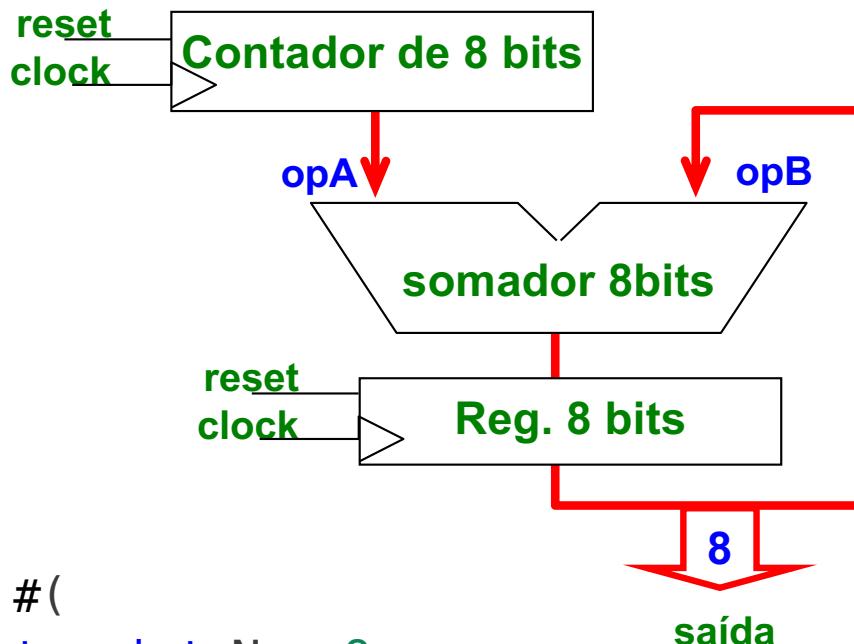
- Execute o script **do sim.do** (pasta ex3)



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

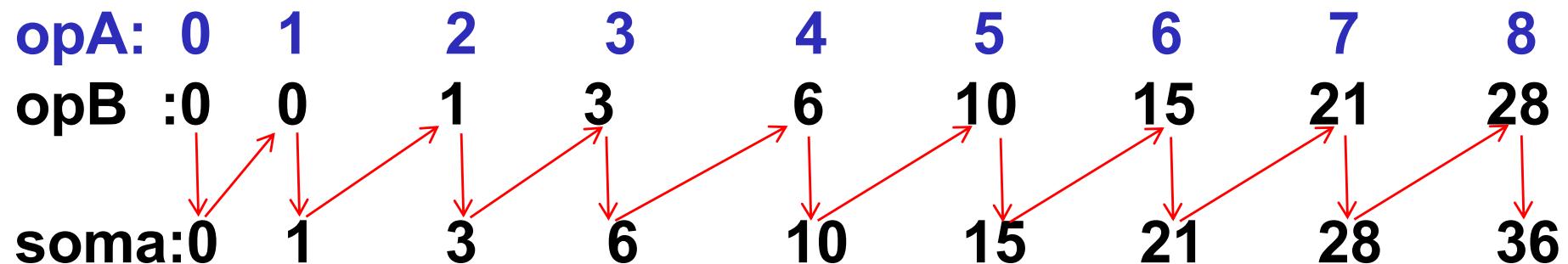
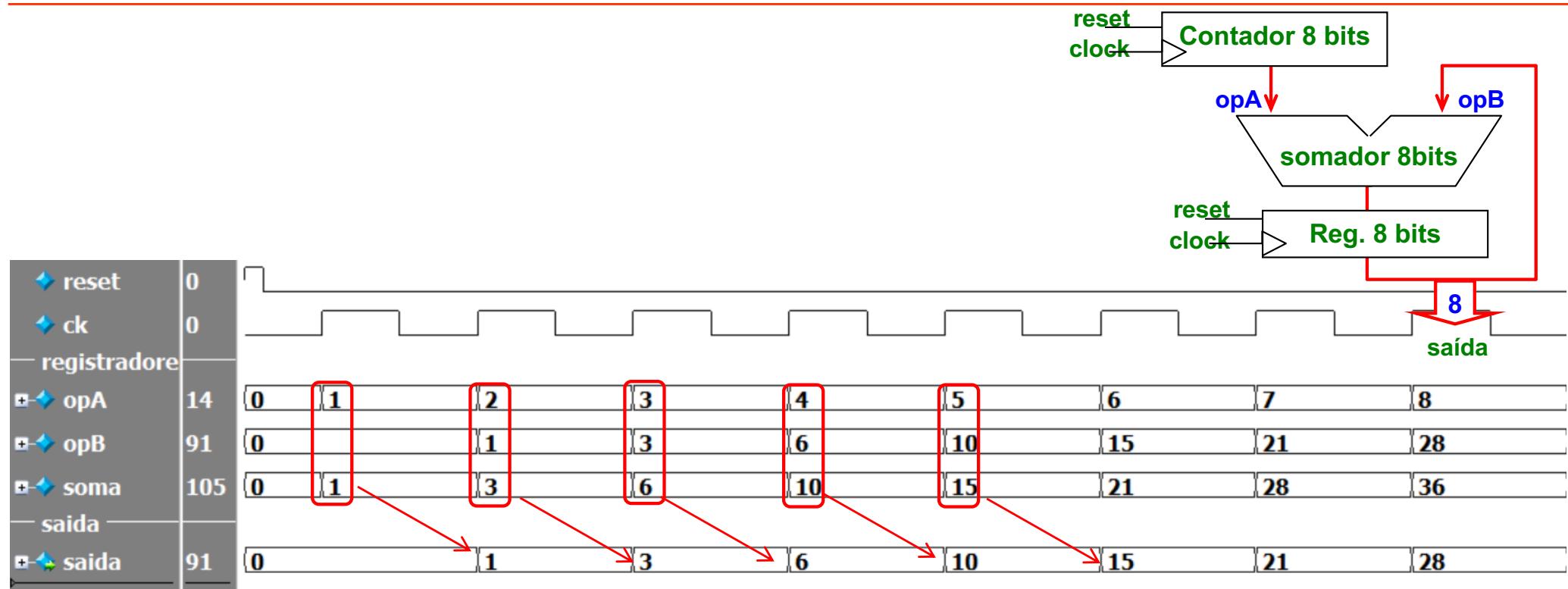
Exercício de Simulação 4

- Descreva o circuito abaixo em SystemVerilog:
 - Um só *always_ff* para opA e opB, pois as variáveis de controle (clock e reset) são as mesmas



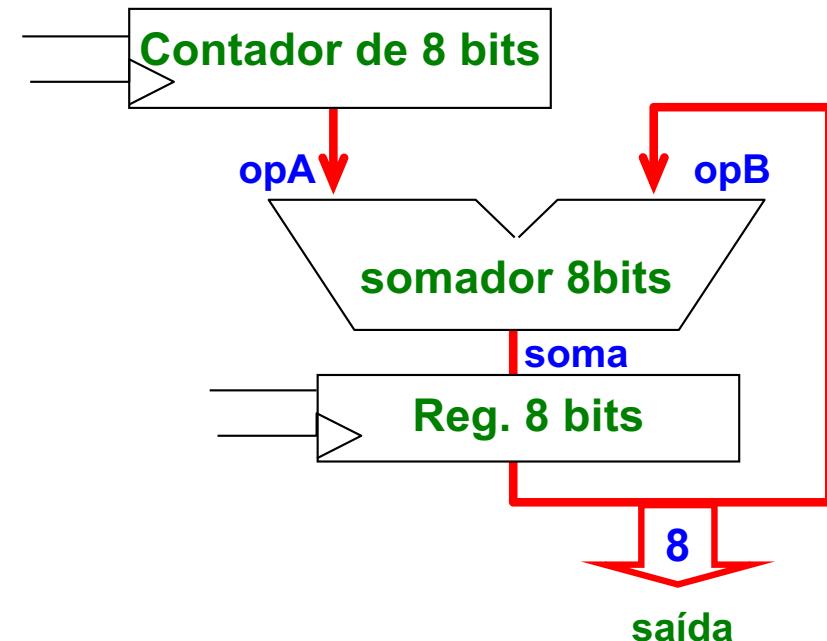
```
module ex4 #(
    parameter int N = 8
) (
    input logic clock,
    input logic reset,
    output logic [N-1:0] saída
);
```

EXERCÍCIO DE SIMULAÇÃO 4



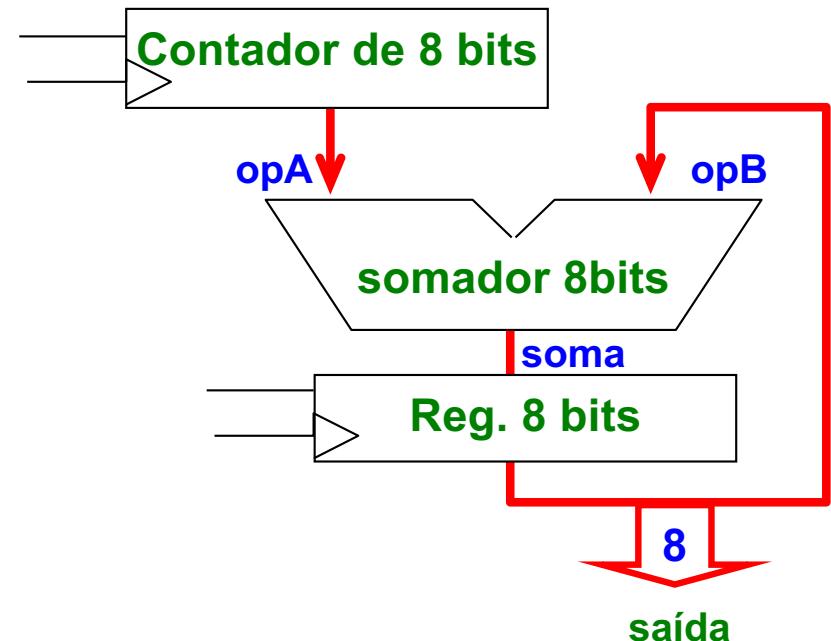
EXERCÍCIO DE SIMULAÇÃO 4

```
module ex4 #( parameter int N = 8)
(
    input logic clock,
    input logic reset,
    output logic [N-1:0] saida
);
    logic [N-1:0] opA, opB, soma;
    assign soma = opA + opB;
    assign saida = opB;
    always_ff @(completar) begin
        if (reset) begin
            opA <= completar;
            opB <= completar;
        end else begin
            opA <= completar;
            opB <= completar;
        end
    end
endmodule
```



Ex 4 - Solução

```
module ex4 #( parameter int N = 8)
(
    input logic clock,
    input logic reset,
    output logic [N-1:0] saida
);
    logic [N-1:0] opA, opB, soma;
    assign soma = opA + opB;
    assign saida = opB;
    always_ff @(completar) begin
        if (reset) begin
            opA <= '0;
            opB <= '0
        end else begin
            opA <= opA + 1;
            opB <= soma;
        end
    end
endmodule
```



Descrição RTL. Considere o circuito descrito pelo código SystemVerilog abaixo utilizando portas lógicas, multiplexadores, registradores e contadores. Indicar no desenho os sinais que são do tipo entrada e saída, e o número de fios em cada sinal

```

module acumulador (
    input logic clock,
    input logic reset,
    input logic enable,
    output logic [7:0] saida
);

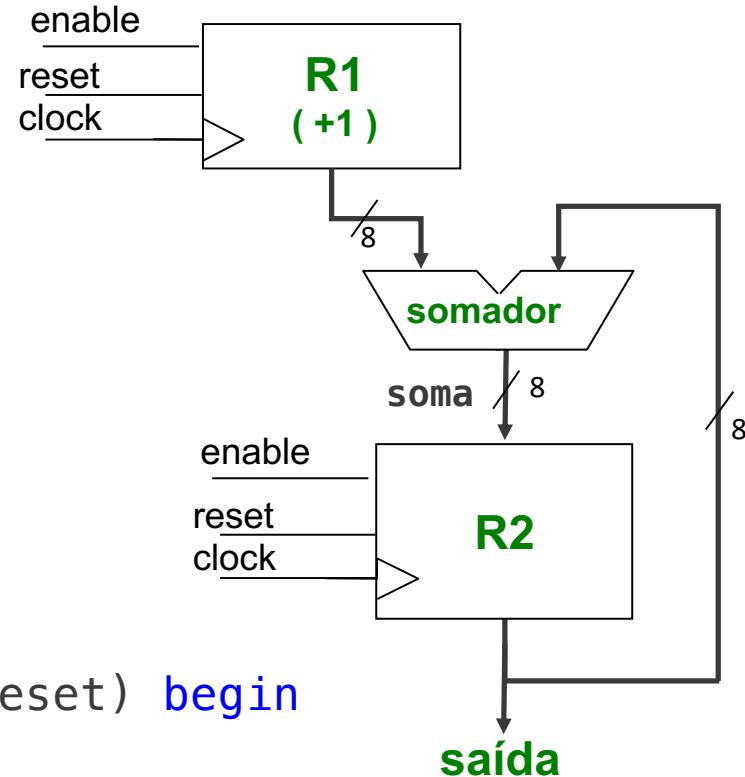
logic [7:0] R1, R2;
logic [7:0] soma;

assign soma = R1 + R2;
assign saida = R2;

always_ff @(posedge clock or posedge reset) begin
    if (reset)
        R1 <= 8'b0;
        R2 <= 8'b0;
    else if (enable)
        R1 <= R1 + 1;
        R2 <= soma;
end

endmodule

```



Descrição RTL. Considere o circuito descrito pelo código SystemVerilog abaixo, que é composto por três registradores de 8 bits, um multiplexador 2x1 e um somador de 8 bits. O circuito possui cinco entradas: dois sinais de 8 bits, chamados **A** e **B**; um sinal de seleção do multiplexador, chamado **sel**; e os sinais de **clock** e **reset**. Como saídas, o circuito gera o valor do registrador REGC, denominado **C**, e a saída.

```
module rtl (
    input logic clock, reset, sel,
    input logic [7:0] A, B,
    output logic [7:0] saida,
    output logic [7:0] C
);
    logic [7:0] REGA, REGB, REGC;
    logic [7:0] op2;

    assign op2 = (sel == 1'b0) ? REGA : B;

    assign saida = REGB + op2;

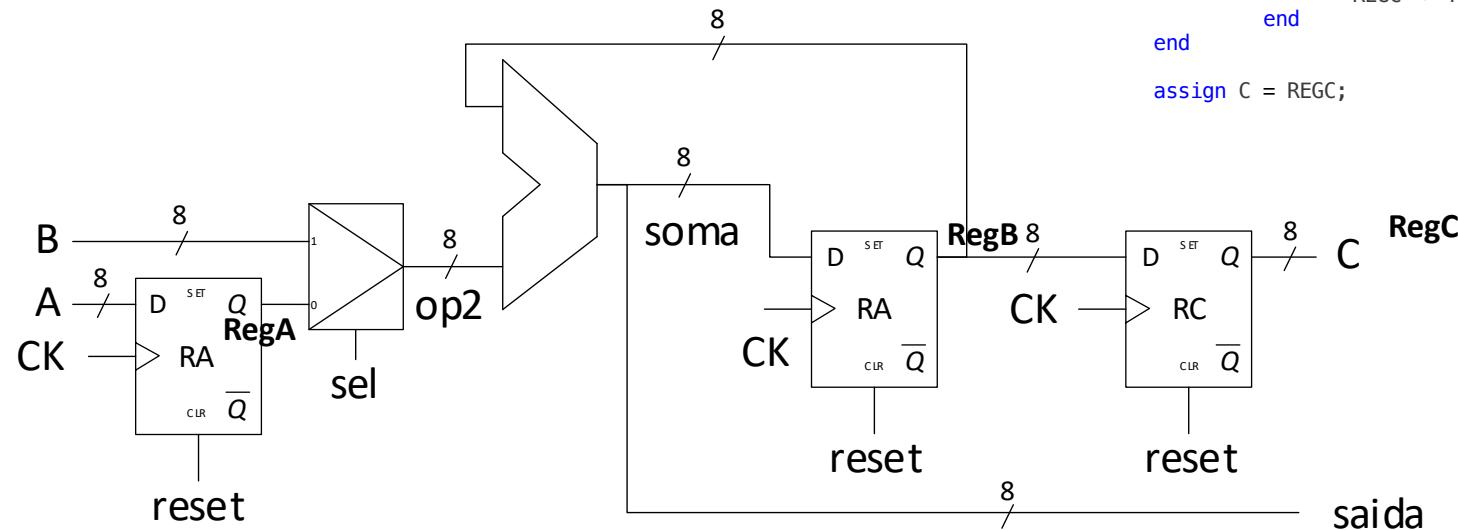
    always_ff @(posedge clock or posedge reset) begin
        if (reset) begin
            REGA <= 8'b0;
            REGB <= 8'b0;
            REGC <= 8'b0;
        end else begin
            REGA <= A;
            REGB <= saida;
            REGC <= REGB;
        end
    end

    assign C = REGC;

endmodule
```

Descrição RTL. Considere o circuito descrito pelo código SystemVerilog abaixo, que é composto por três registradores de 8 bits, um multiplexador 2x1 e um somador de 8 bits. O circuito possui cinco entradas: dois sinais de 8 bits, chamados **A** e **B**; um sinal de seleção do multiplexador, chamado **sel**; e os sinais de **clock** e **reset**. Como saídas, o circuito gera o valor do registrador REGC, denominado **C**, e a **saída**.

Solução



```

assign op2 = (sel == 1'b0) ? REGA : B;
assign saida = REGB + op2;

always_ff @(posedge clock or posedge reset) begin
    if (reset) begin
        REGA <= 8'b0;
        REGB <= 8'b0;
        REGC <= 8'b0;
    end else begin
        REGA <= A;
        REGB <= saida;
        REGC <= REGB;
    end
end

assign C = REGC;

```

Descrição RTL. Considere o circuito descrito pelo código SystemVerilog abaixo utilizando portas lógicas, multiplexadores, registradores e contadores. Indicar no desenho os sinais que são do tipo entrada e saída, e o número de fios em cada sinal

```
module RTL (
    input logic reset,
    input logic clock,
    input logic [7:0] E0, E1, E2, E3,
    output logic [7:0] S0, S1, S2, S3
);

    logic [1:0] cont;
    logic [7:0] mux;

    assign mux = (cont == 2'b00) ? E0 :
                (cont == 2'b01) ? E1 :
                (cont == 2'b10) ? E2 :
                E3;

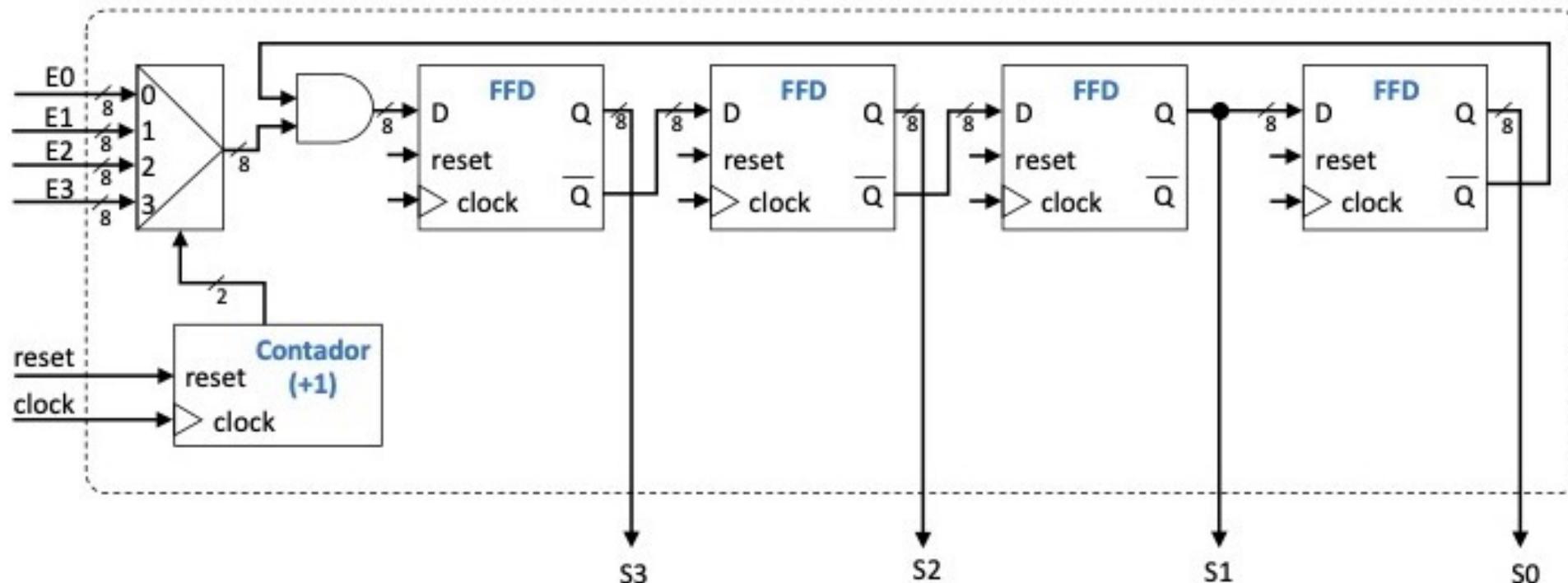
    always_ff @(posedge clock or posedge reset) begin
        if (reset) begin
            S3 <= 8'b0;
            S2 <= 8'b0;
            S1 <= 8'b0;
            S0 <= 8'b0;
            cont <= 2'b00;
        end else begin
            S3 <= mux & ~S0;
            S2 <= ~S3;
            S1 <= ~S2;
            S0 <= S1;
            cont <= cont + 1;
        end
    end
end
endmodule
```

Descrição RTL. Considere o circuito descrito pelo código SystemVerilog abaixo utilizando portas lógicas, multiplexadores, registradores e contadores. Indicar no desenho os sinais que são do tipo entrada e saída, e o número de fios em cada sinal

Solução

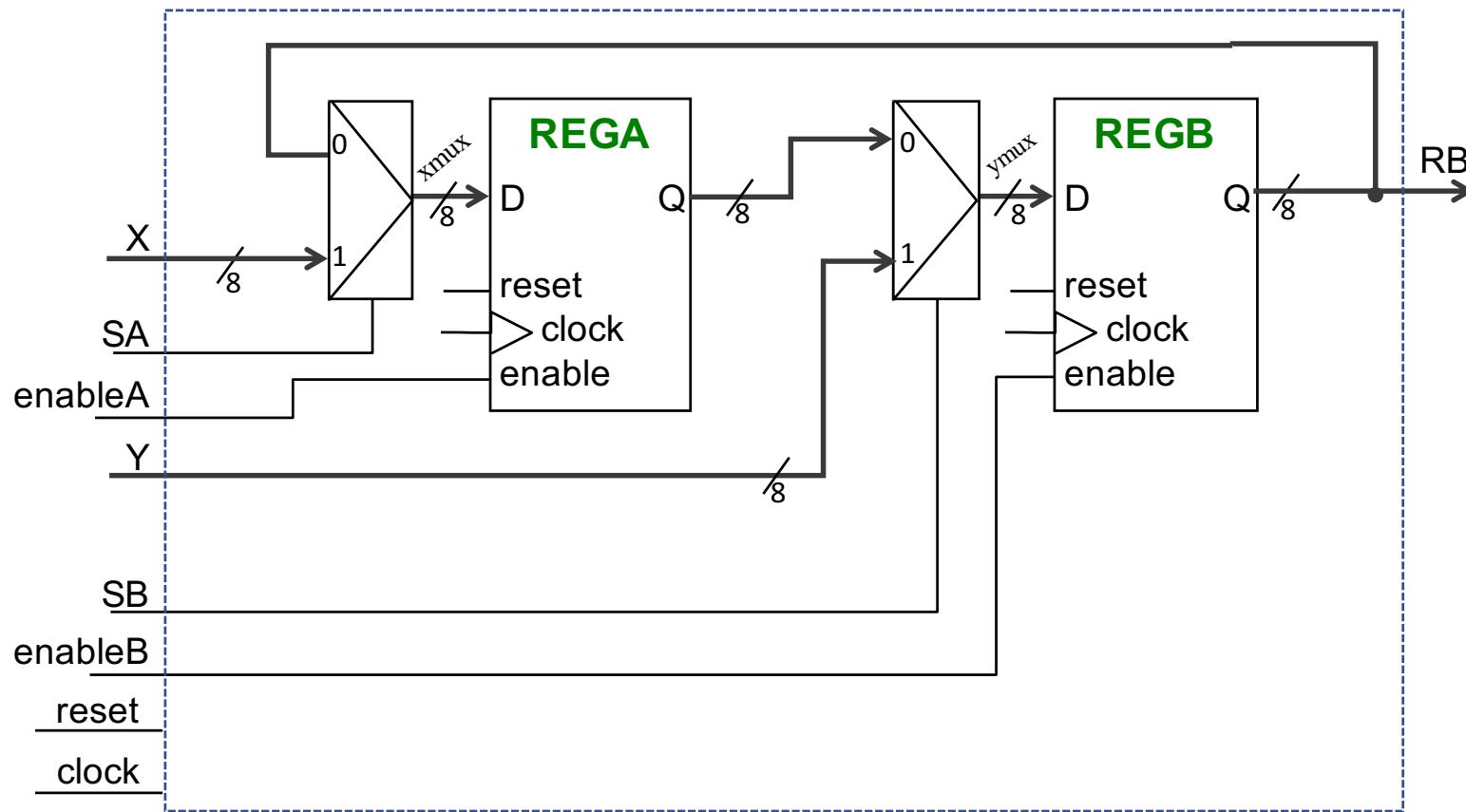
```
assign mux = (cont == 2'b00) ? E0 :  
           (cont == 2'b01) ? E1 :  
           (cont == 2'b10) ? E2 :  
           E3;  
  
end
```

```
always_ff @ (posedge clock or posedge reset) begin  
    if (reset) begin  
        S3 <= 8'b0; S2 <= 8'b0; S1 <= 8'b0; S0 <= 8'b0;  
        cont <= 2'b00;  
    end else begin  
        S3 <= mux & ~S0;  
        S2 <= ~S3;  
        S1 <= ~S2;  
        S0 <= S1;  
        cont <= cont + 1;  
    end  
end
```

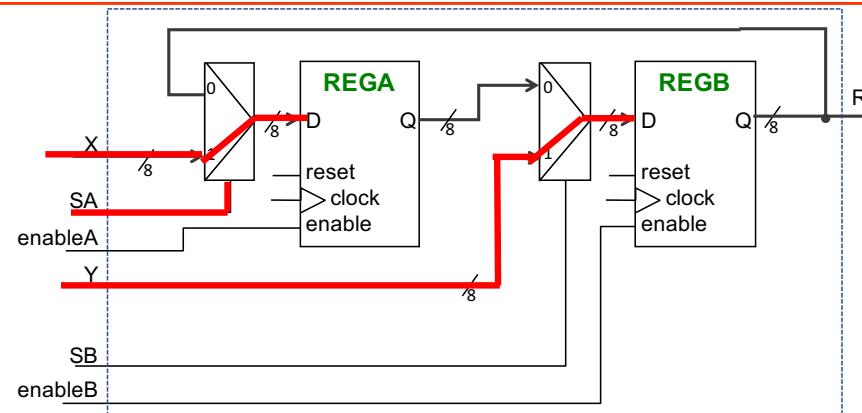


Exercício de Simulação 5

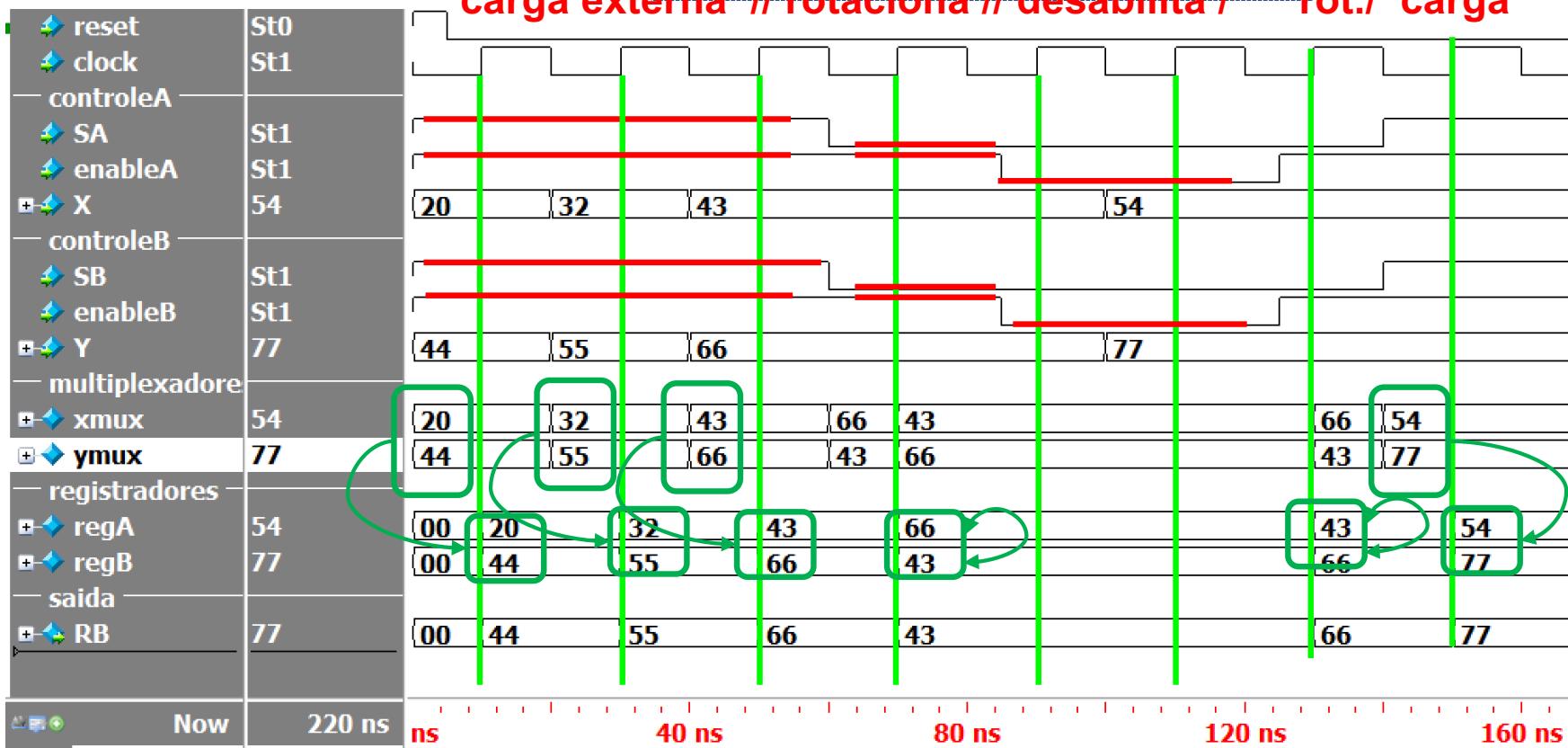
1. (2,5 pts) **Projeto RTL.** Considere o circuito abaixo composto por 2 registradores de 8 bits e dois multiplexadores 2x1. O circuito possui 8 **entradas**: dois sinais de 8 bits: X e Y , habilitação dos registradores: $enableA$ e $enableB$, seleção dos multiplexadores: SA e SB , e o sinal de $clock$ e $reset$ para os registradores. Como **saída** o circuito possui o valor do registrador REGB, RB .



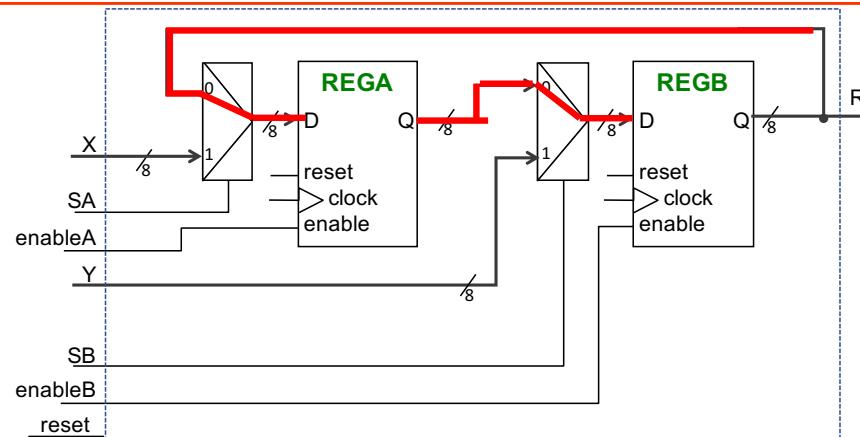
Exercício de Simulação 5



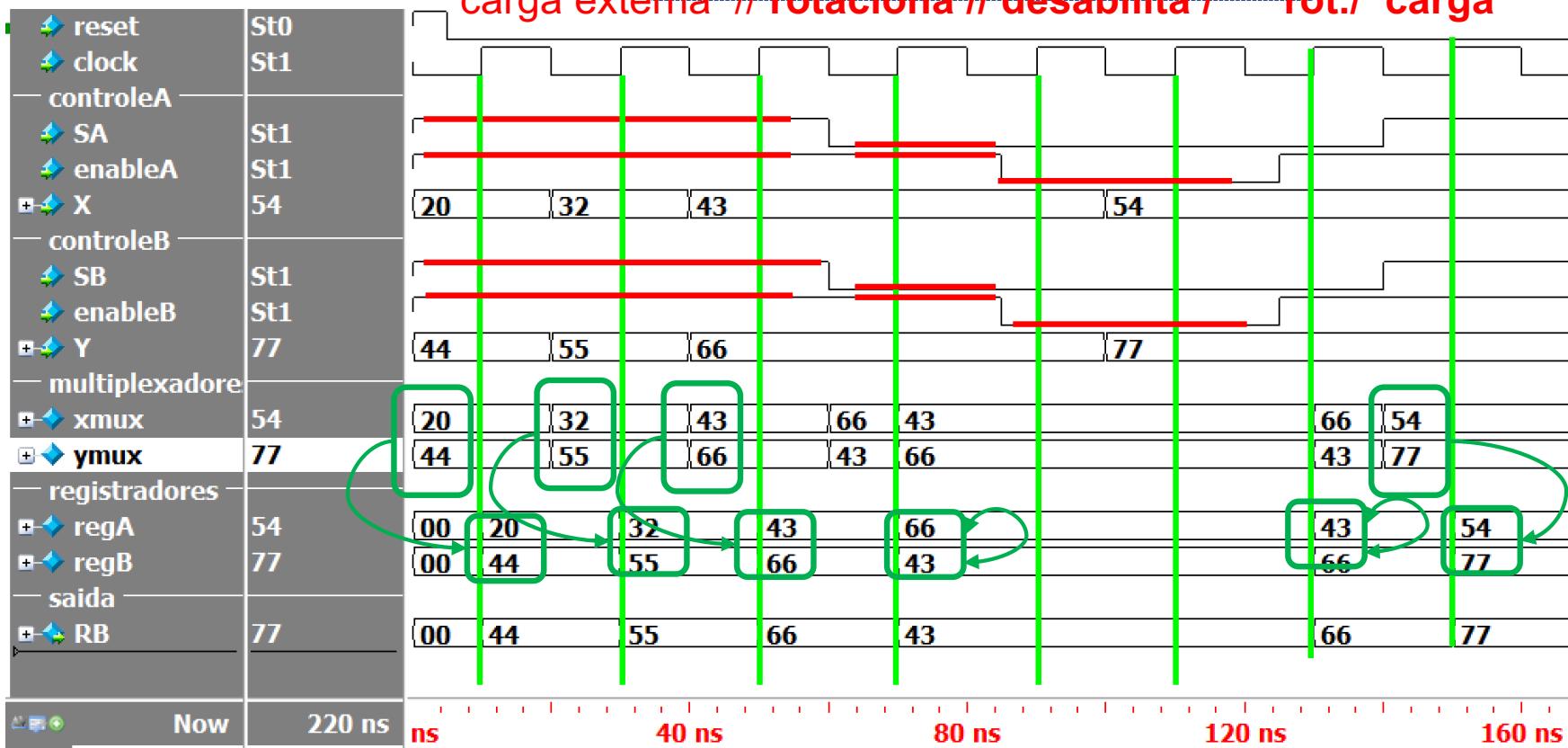
carga externa // rotaciona // desabilita / rot./ carga



Exercício de Simulação 5



carga externa // rotaciona // desabilita / rot./ carga



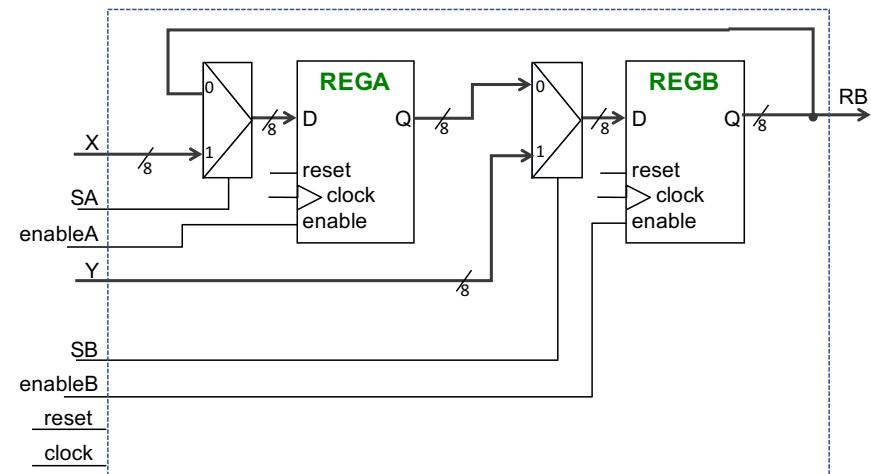
Solução ex5

```
module ex5 (
    input logic reset,clock, SA, enableA, SB, enableB,
    input logic [7:0] X,
    input logic [7:0] Y,
    output logic [7:0] RB
);

    logic [7:0] xmux, ymux;
    logic [7:0] regA, regB;

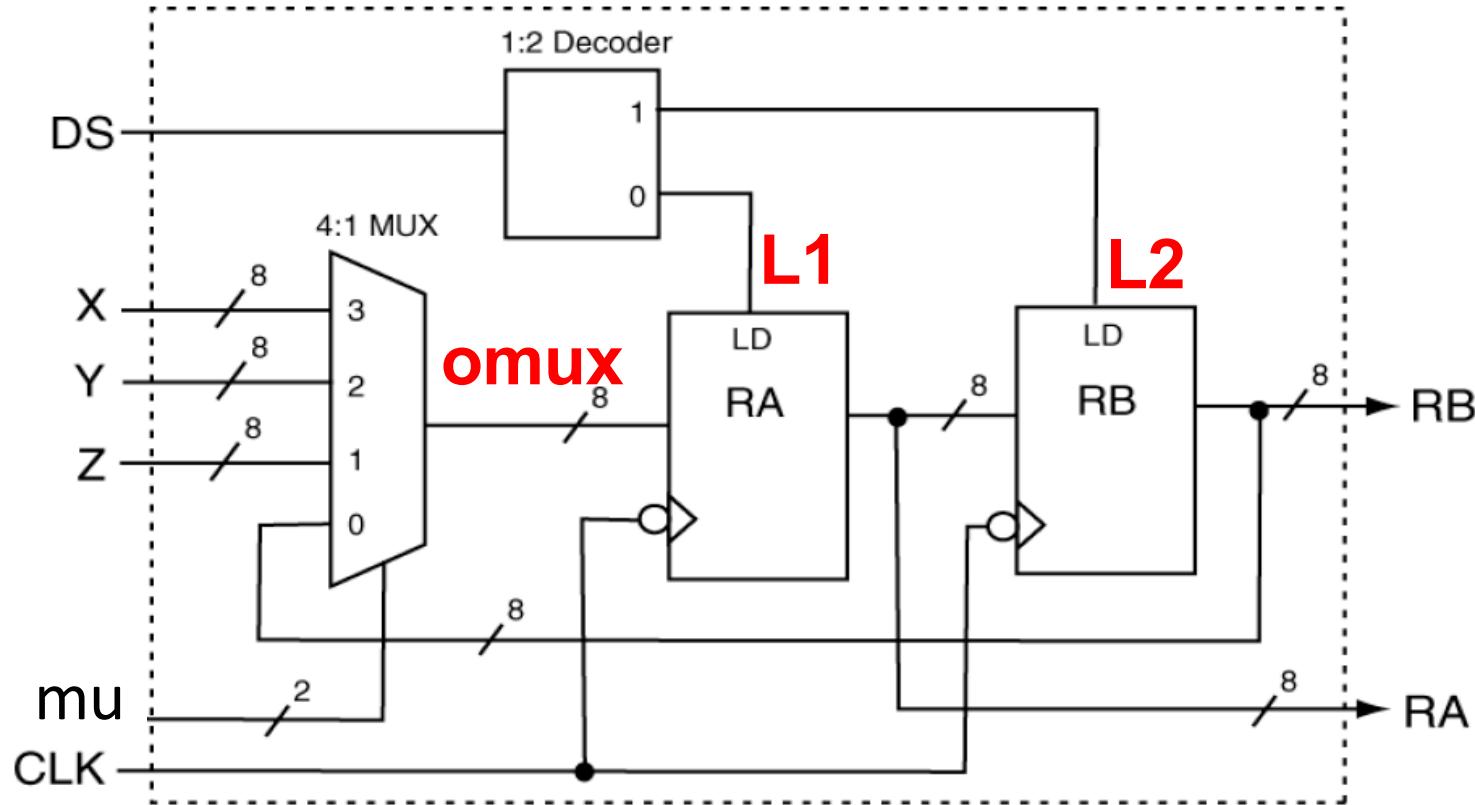
    assign RB = regB;
    assign xmux = SA ? X : regB;
    assign ymux = SB ? Y : regA;

    always_ff @(posedge clock or posedge reset) begin
        if (reset) begin
            regA <= '0;
            regB <= '0;
        end else begin
            if (enableA)
                regA <= xmux;
            if (enableB)
                regB <= ymux;
        end
    end
endmodule
```



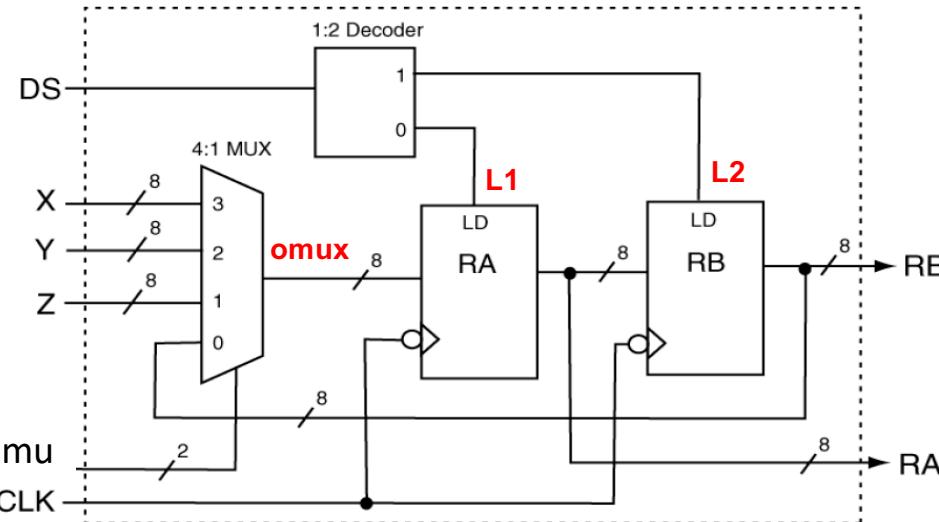
Exercício de Simulação 6

Descreve e simular o circuito abaixo utilizando SystemVerilog



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

Saída esperado para o ex6



```

always_ff @(negedge clock or
posedge reset) begin
    if (reset) begin
        RA <= '0;
        RB <= '0;
    end else begin
        if (L1) RA <= OMUX;
        if (L2) RB <= RA;
    end
end

```

	St0	St1
/tb/dut/reset		
/tb/dut/clock		
entradas		
+ /tb/dut/X	20	10 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0 00 10
+ /tb/dut/Y	5a	05 0a 0f 14 19 1e 23 28 2d 32 37 3c 41 46 4b 50 55
+ /tb/dut/Z	24	02 04 06 08 0a 0c 0e 10 12 14 16 18 1a 1c 1e 20 22
saida mux		
+ /tb/dut/mu	2	1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1
+ /tb/dut/OMUX	5a	02 0a 30 0a 1e 70 0a 12 32 b0 0a 1a 46 f0 46 22
registadores		
/tb/dut/DS	St1	
+ /tb/dut/RA	46	00 02 0a 1e 70 0a 1a 46
+ /tb/dut/RB	46	00 0a 46

ex6 - solução

```
module ex6 (
    input logic clock, reset, DS,
    input logic [1:0] mu,
    input logic [7:0] X, Y, Z,
    output logic [7:0] RA, RB
);
```

```
logic [7:0] OMUX;
logic L1, L2;
```

```
assign L1 = ~DS;
assign L2 = DS;
```

```
always_comb begin
```

```
case (mu)
```

```
2'b11: OMUX = X;
2'b10: OMUX = Y;
2'b01: OMUX = Z;
default: OMUX = RB;
```

```
endcase
```

```
end
```

```
always_ff @(negedge clock or posedge reset)
```

```
begin
```

```
if (reset) begin
```

```
RA <= '0;
```

```
RB <= '0;
```

```
end else begin
```

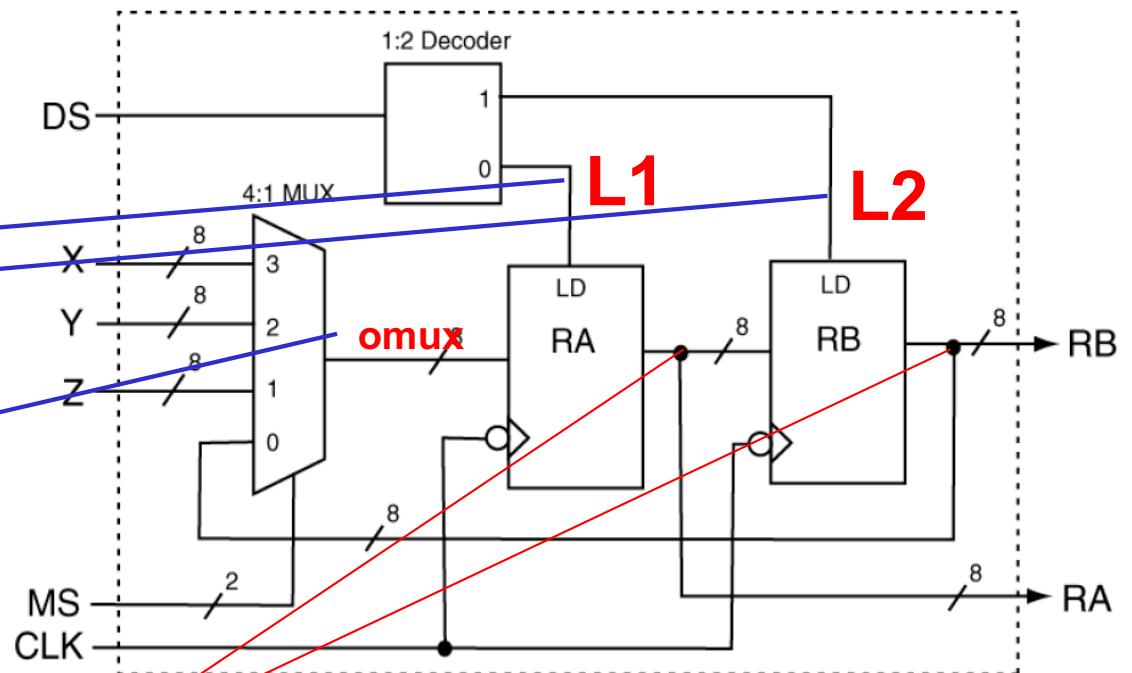
```
if (L1) RA <= OMUX;
```

```
if (L2) RB <= RA;
```

```
end
```

```
end
```

```
endmodule
```



Ex. de Simulação 7

```
module ex7 (
    input logic fin,
    input logic rst,
    output logic fout
);

    logic A, B, C;

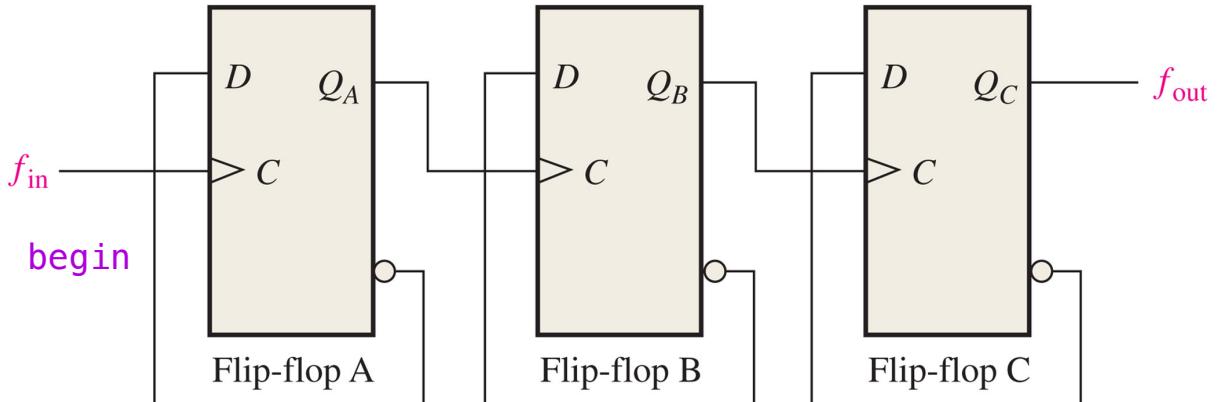
    always_ff @ (posedge fin or posedge rst) begin
        if (rst)
            A <= 0;
        else
            A <= ~A;
    end

    always_ff @ (posedge A or posedge rst) begin
        if (rst)
            B <= 0;
        else
            B <= ~B;
    end

    always_ff @ (posedge B or posedge rst) begin
        if (rst)
            C <= 0;
        else
            C <= ~C;
    end

    assign fout = C;
endmodule
```

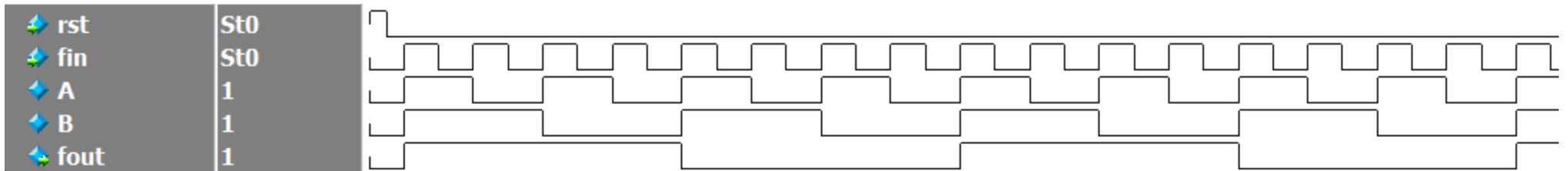
Qual a saída esperada para *f_{out}*?



O **reset** não está mostrado no desenho

Este é um circuito divisor de clock!

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



```
module tb;

    logic rst = 0;
    logic fin = 0;
    logic fout;

    always #10 fin = ~fin;

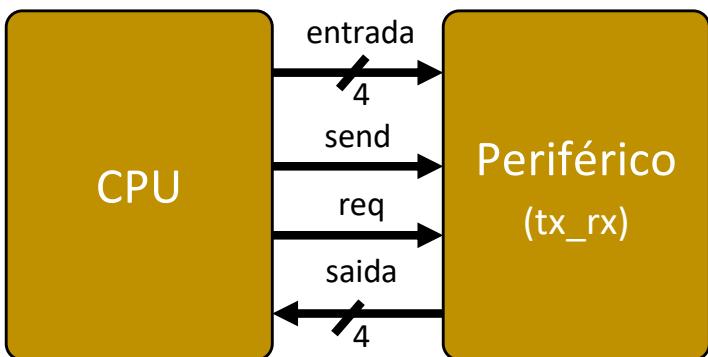
    initial begin
        rst = 1;
        #5 rst = 0;
    end

    ex7 dut ( .rst(rst), .fin(fin), .fout(fout) );

endmodule
```

Exercício de Simulação 8

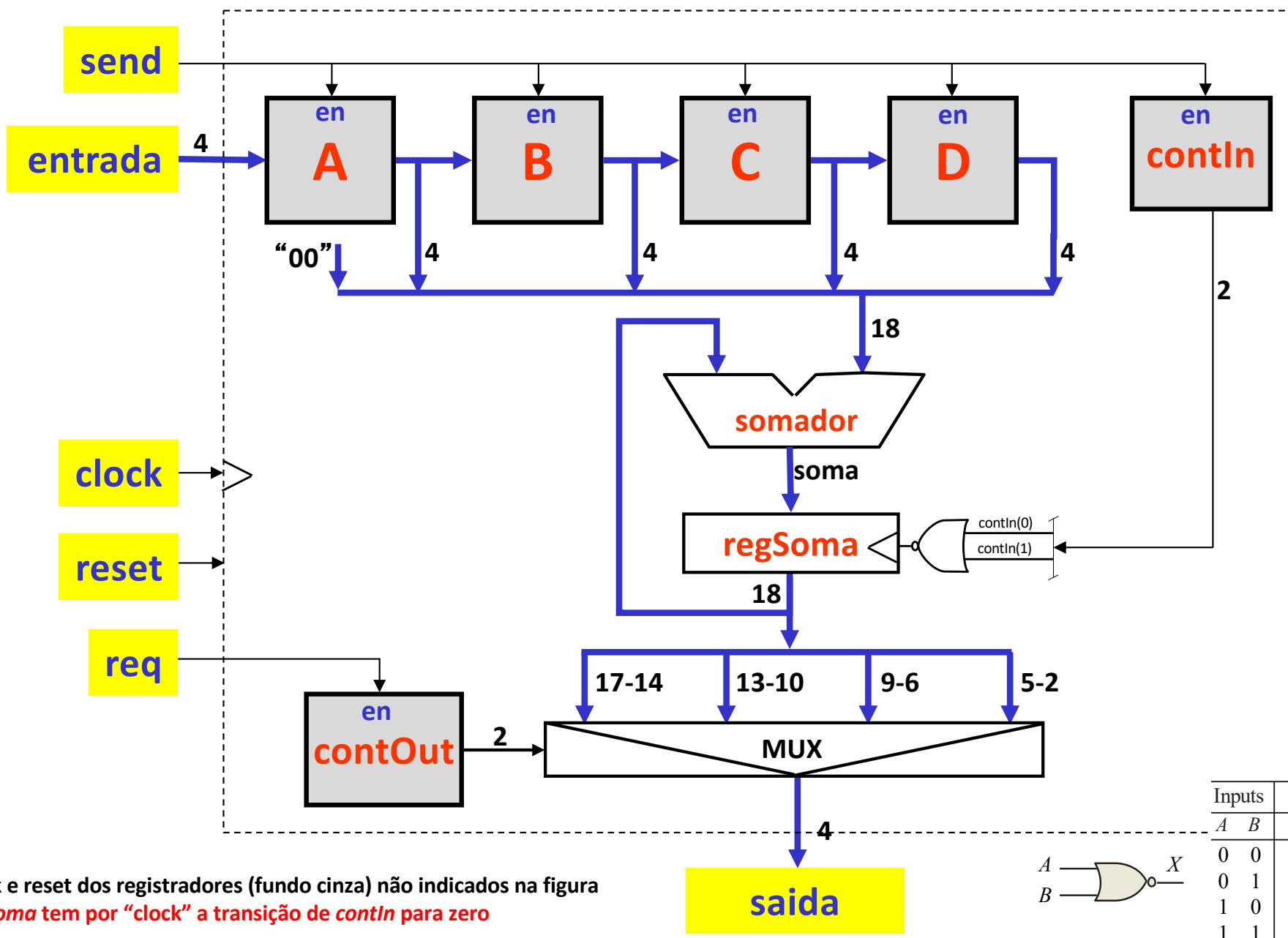
COMUNICAÇÃO ENTRE CPU E PERIFÉRICO



Função do Periférico:

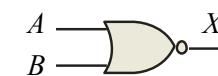
- 1. Recebe 4 palavras de 16 bits**
 - ✓ A CPU deve enviar 16 palavras de 4 bits (cada **send** avisa que tem 4 bits disponíveis em **entrada**)
- 2. Somar as 4 palavras de 16 bits, sem perder precisão**
 - ✓ Para isto o somador deve ter 18 bits
- 3. Calcular a média aritmética das 4 palavras, sem utilizar 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
reqsoma tem por “clock” a transição de *contIn* para zero**

Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0



```

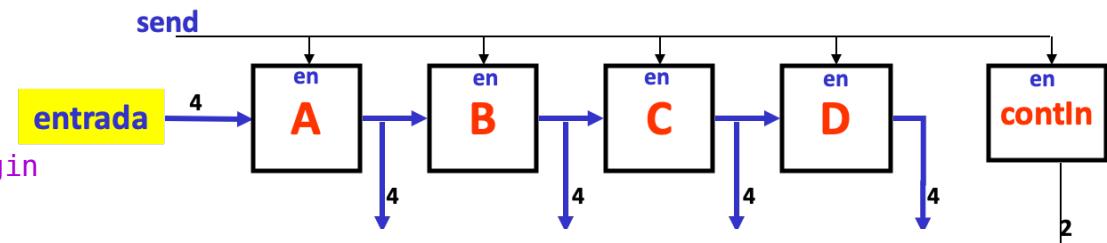
module ex8 (
    input logic clock, reset, send, req,
    input logic [3:0] entrada,
    output logic [3:0] saida
);

logic [3:0] A, B, C, D;
logic [1:0] contIn, contOut;
logic [17:0] vet, regSoma;
logic acumula;

always_ff @(posedge clock or posedge reset) begin
    if (reset) begin
        contIn <= 2'b00;
        A <= '0; B <= '0; C <= '0; D <= '0;
    end else if (send) begin
        contIn <= completar
        A <= completar
        B <= completar
        C <= completar
        D <= completar
    end
end
endmodule

```

Entrada de Dados (1/5)



```

logic [3:0] padrao_de_teste [0:15] = '{  

    4'h1, 4'h2, 4'h3, 4'h4,  

    4'h5, 4'h6, 4'h7, 4'h8,  

    4'h3, 4'h4, 4'h4, 4'h5,  

    4'h2, 4'h1, 4'h1, 4'h3  

};

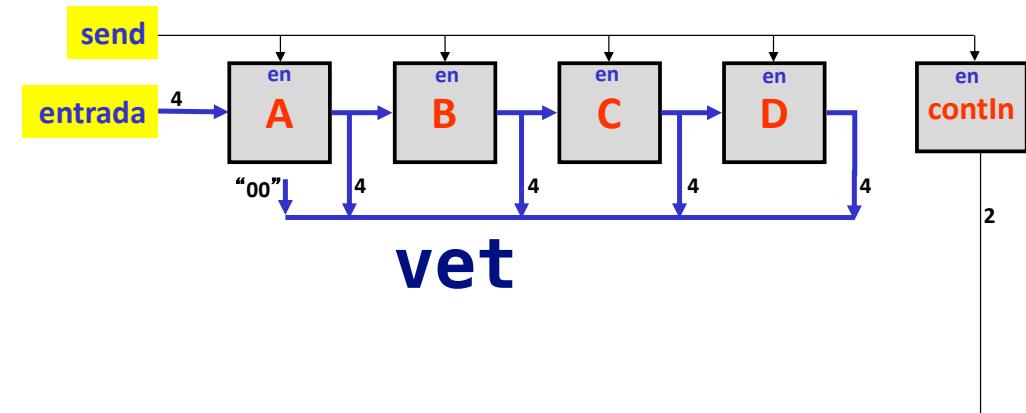
```



Soma (2/5)

Criar o barramento *vet*, de 18 bits via concatenação

assign *vet* = {*A*, *B*, *C*, *D*};



/tb/dut/clock	St1	
/tb/dut/reset	St0	
entrada		
+ /tb/dut/entrada	3	
+ /tb/dut/send	St0	
registradores		
+ /tb/dut/A	3	
+ /tb/dut/B	1	
+ /tb/dut/C	1	
+ /tb/dut/D	2	
+ /tb/dut/contIn	0	
+ /tb/dut/vet	03112	
	01000 02100 03210 04321 05432 06543 07654 08765 03876 04387 04438 05443 02544 01254 01125 03112	

Primeiro valor: x4321

x8765

x5443

x3112

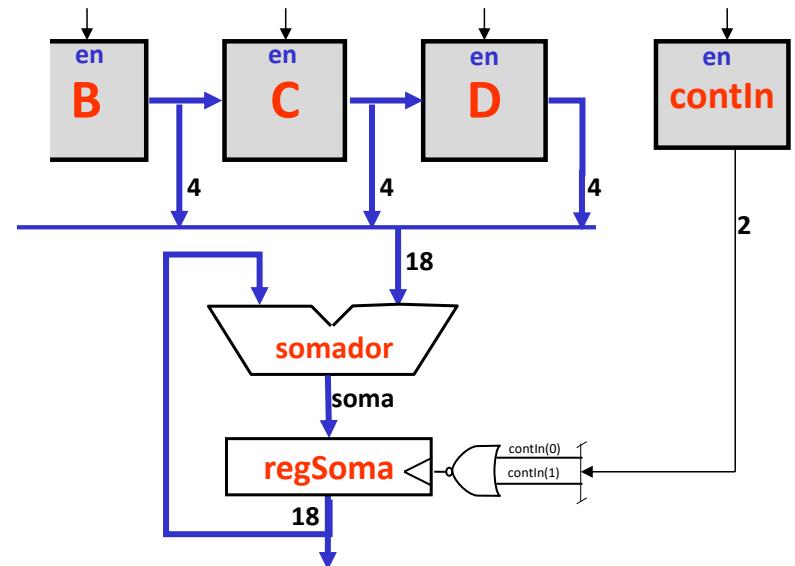
Acumulação (3/5)

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

```
// verdadeiro se contIn == "00"
nor(acumula, contIn[0], contIn[1]);

always_ff @(posedge acumula or posedge reset) begin
    if (reset)
        regSoma <= '0;
    else
        regSoma <= regSoma + vet;
end
```

$$\begin{array}{r}
 4321 \\
 + 8765 \\
 \hline
 \text{CA86}
 \end{array}$$



Detecta a transição de `contIn="00"` para acumular

The figure shows a timing diagram with multiple horizontal timelines representing different signals over time. The signals include:

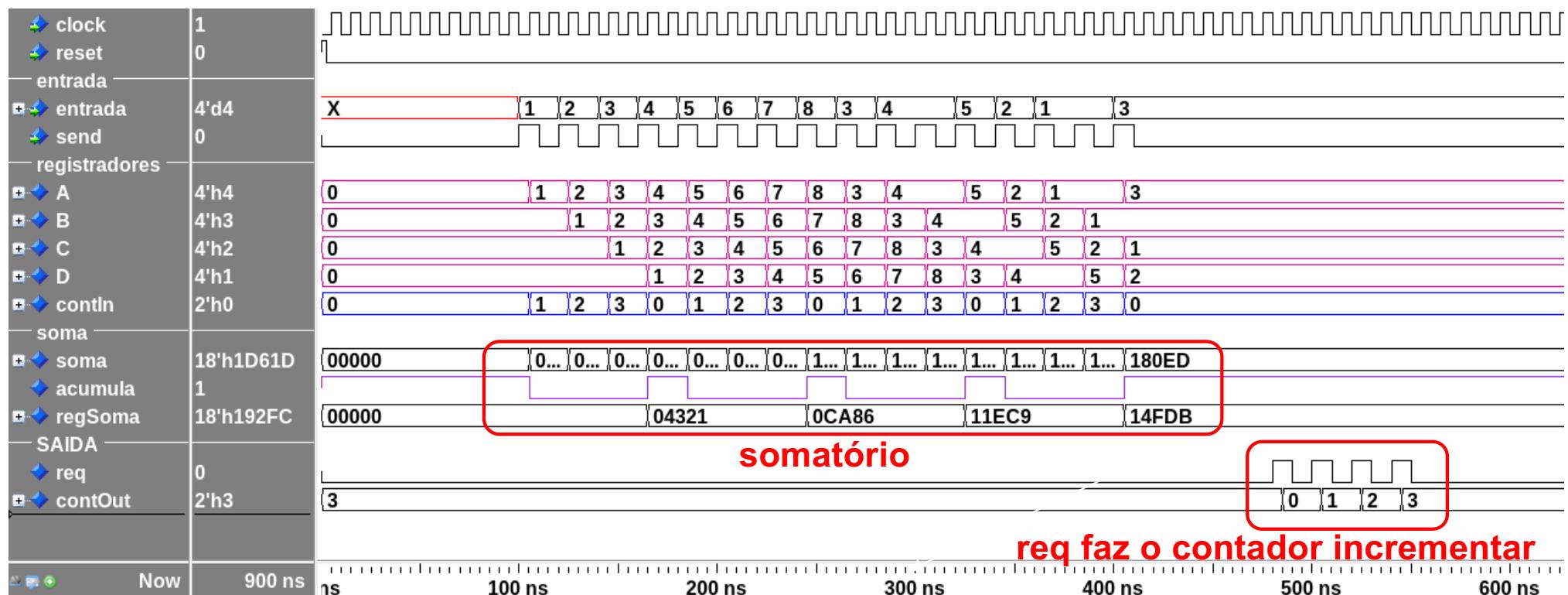
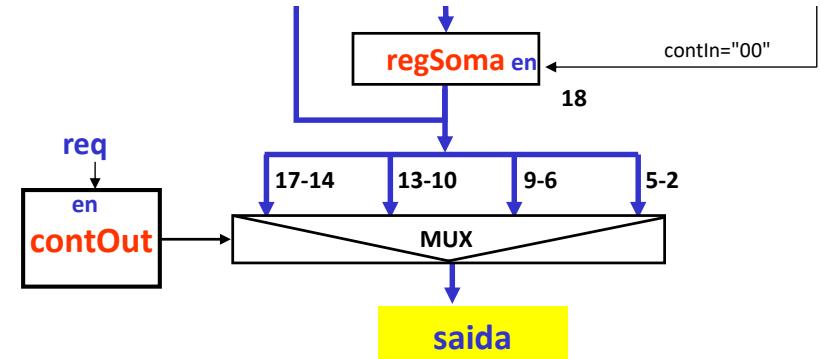
- /tb/dut/clock**: A square wave clock signal.
- /tb/dut/reset**: A pulse signal that initializes the registers.
- entrada**: An input signal with values 1 through 13.
- /tb/dut/send**: A signal indicating when data is ready to be sent.
- registradores**: Four 4-bit registers (**/tb/dut/A**, **/tb/dut/B**, **/tb/dut/C**, **/tb/dut/D**) and a counter (**/tb/dut/contIn**). The registers have values corresponding to the **entrada** signal. The counter **/tb/dut/contIn** has values from 0 to 14.
- soma**: A summing junction that adds the outputs of the registers and the counter. The output is 14fdb.
- /tb/dut/vet**: A reference value of 03112.
- /tb/dut/acumula**: A signal that accumulates the sum.
- /tb/dut/regSoma**: The final accumulated sum value, which is 14fdb.

Red boxes highlight specific transitions in the register data where the counter passes zero, particularly in registers A, B, C, and D.

Contador em função do *request* (4/5)

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

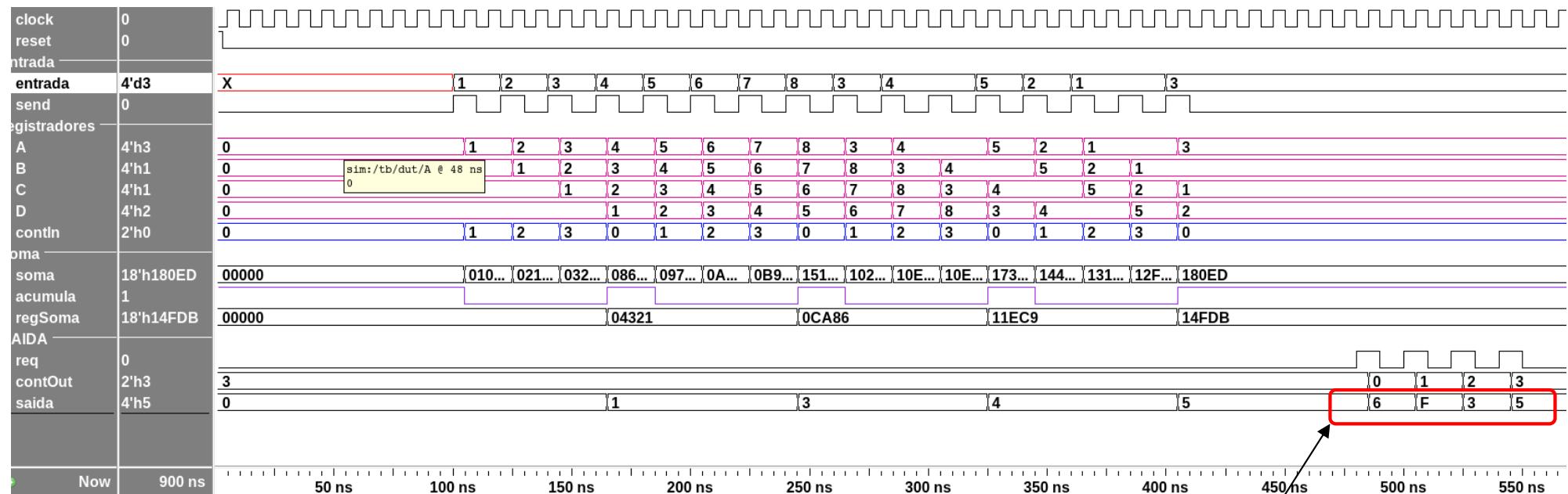
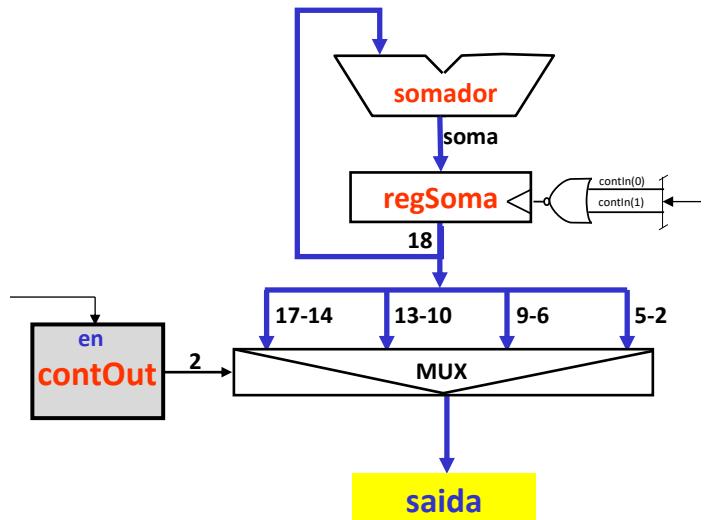
Cuidado: no reset inicializar em “11” para o primeiro valor ser “00”



Multiplexador de saída (5/5)

Bloco combinacional

```
assign saida = (contOut == 2'b00) ? regSoma[5:2] :  
    (contOut == 2'b01) ? regSoma[9:6] :  
    (contOut == 2'b10) ? regSoma[13:10] :  
    regSoma[17:14];
```



Média: $14FDB / 4 = 53F6$

Fim
Parte flip-flops
e projeto RTL