# SD132 – Circuitos Digitais II
# A-201 Latches e Flip-Flops

| Autores | |
|---|---|
| Daniel Muñoz Arboleda (UnB) | Gilmar Silva Beserra (UnB) |
| Nome 3 (INSTITUIÇÃO) | Nome 4 (INSTITUIÇÃO) |

| Histórico de revisões | | |
|---|---|---|
| 07/01/2025 | V1.0 | Versão inicial |
| 09/02/2025 | V1.2 | Revisão dos exemplos em VHDL (revisão em simulação comportamental) |
| 03/03/2025 | V1.2 | Incluídos exemplos em Verilog (revisão em simulação comportamental) |
| 25/03/2025 | V1.2 | Revisão de conteúdo |

# Tópicos

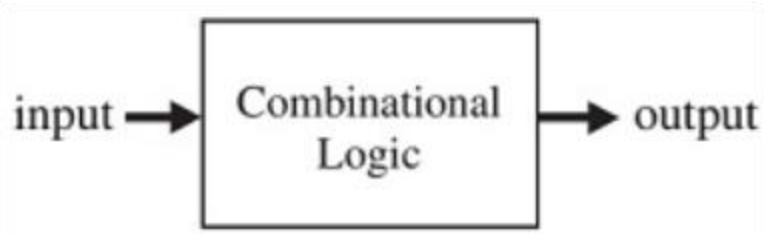
- Definição de sistema digital sequencial
- Latch SR
- Latch SR chaveado
- Latch D chaveado
- Flip-Flop D
- Flip-Flop JK
- Flip-Flop T
- Parâmetros de temporização de um Flip-Flop
- Implementação de Latches e Flip-Flops em VHDL/Verilog

Página em branco.

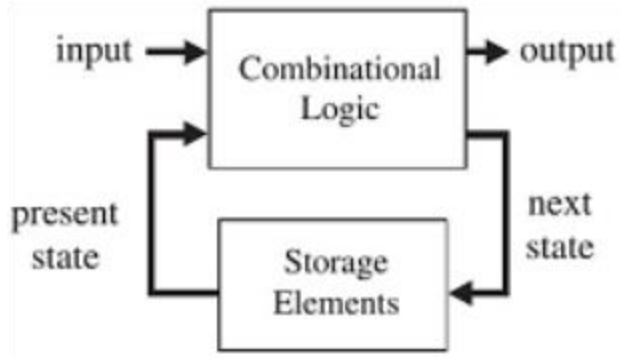
# Aula 1
# Latch SR e Latch D

# Circuitos Sequenciais

- Um sistema digital sequencial é um sistema digital que, em geral, não pode ser descrito exclusivamente pela combinação das entradas.
- Portanto, é um sistema digital que sob as mesmas condições possui mais de um estado, isto é, depende dos valores passados das entradas (memória).



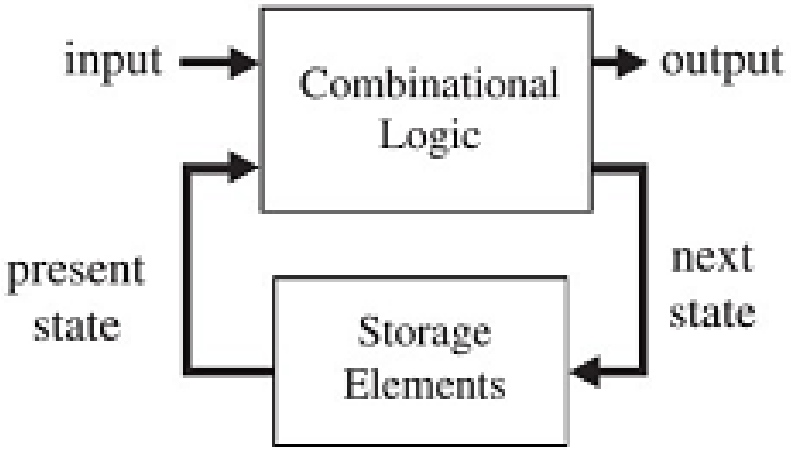

Lógica combinacional




Lógica sequencial

# Circuitos Sequenciais

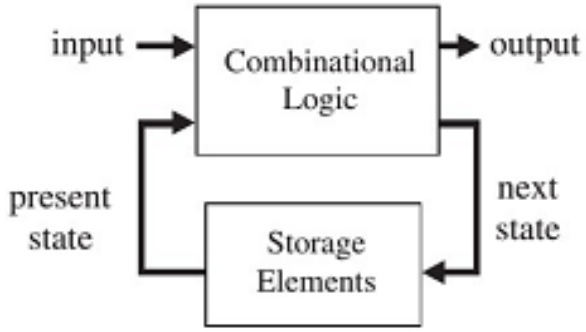
- Um circuito sequencial emprega elementos de armazenamento denominados Latches e Flip-Flops, além de portas lógicas. Os valores das saídas do circuito dependem dos valores das entradas e dos estados dos Latches ou Flip-Flops utilizados.




Lógica sequencial

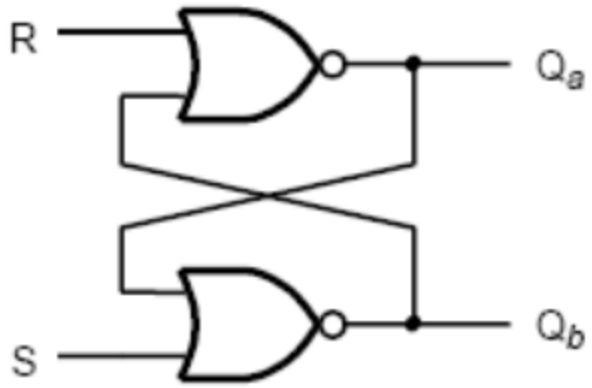
# Circuitos Sequenciais

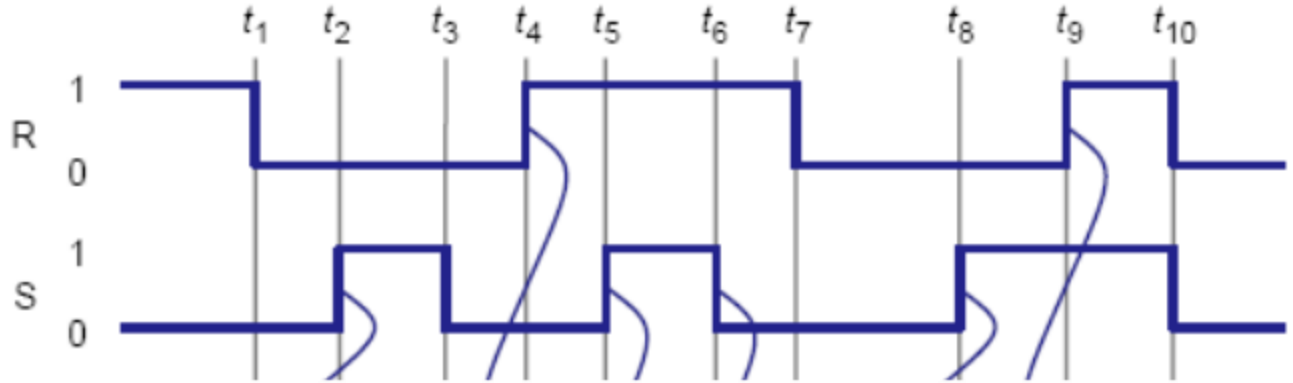
- Como o estado dos Latches e Flip-Flops é função dos valores anteriores das entradas, diz-se que as saídas de um circuito sequencial dependem dos valores das entradas e do histórico do próprio circuito.
- O comportamento de um circuito sequencial é especificado pela sequência temporal das entradas e de seus estados internos.




Lógica sequencial

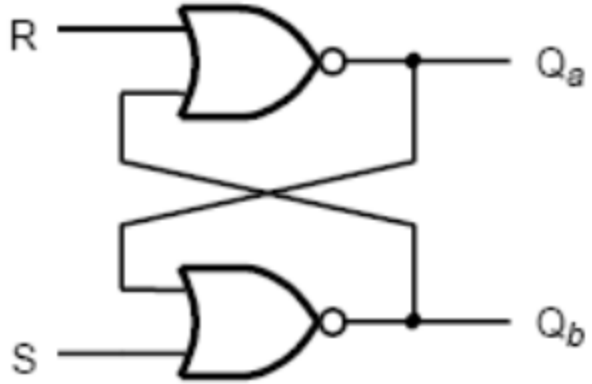
# Latch SR com portas NOR

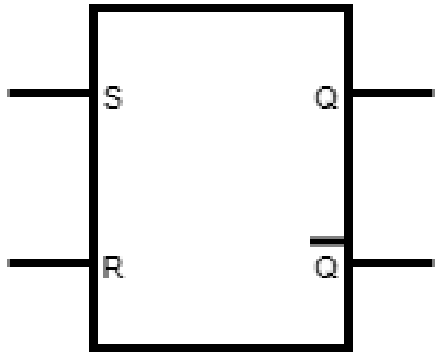


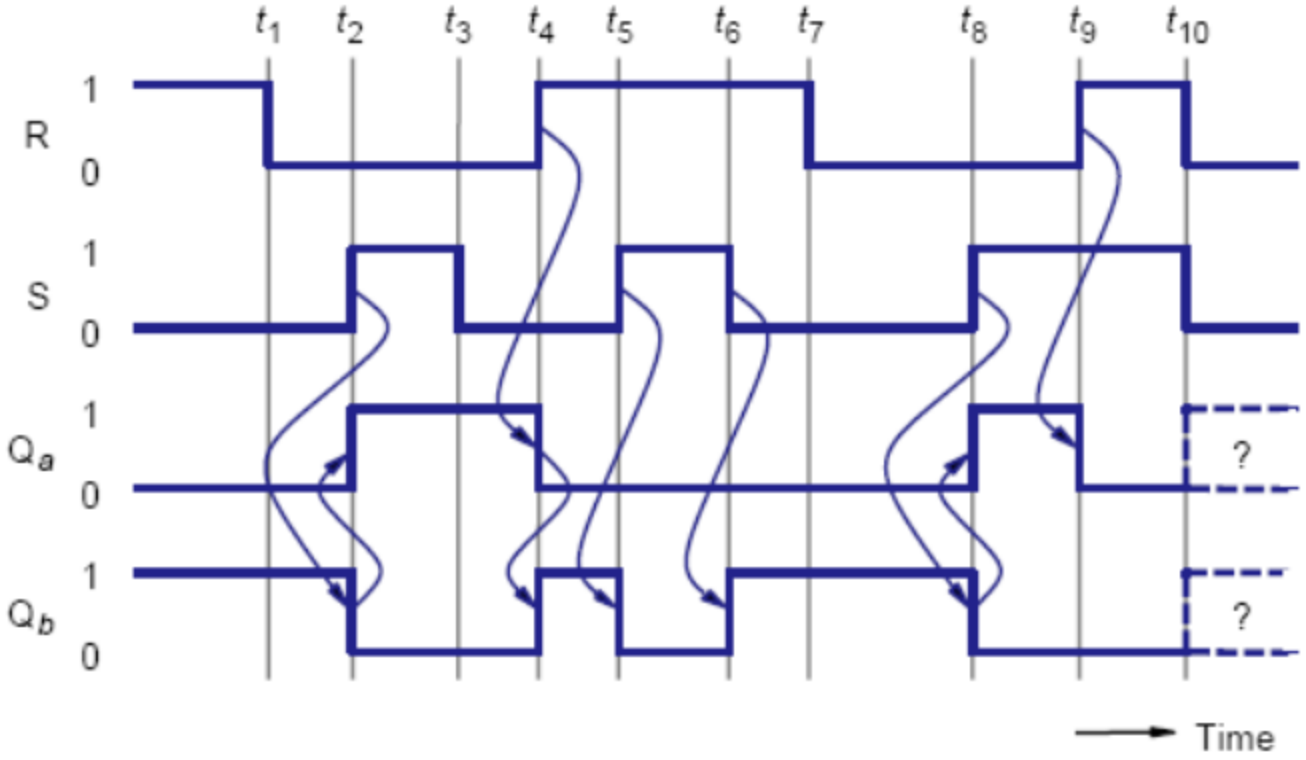




- Observação: um Latch é um circuito sequencial com um único laço de realimentação.
- Dica: para analisar circuitos com realimentação, adote valores iniciais nos laços de realimentação.
- Exercício: analisar este circuito, assumindo que em t=0, os valores da saída são Qa= ‘0’ e Qb = ‘1’.

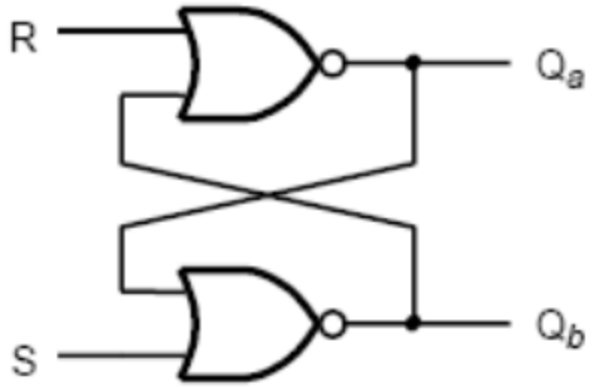
# Latch SR com portas NOR



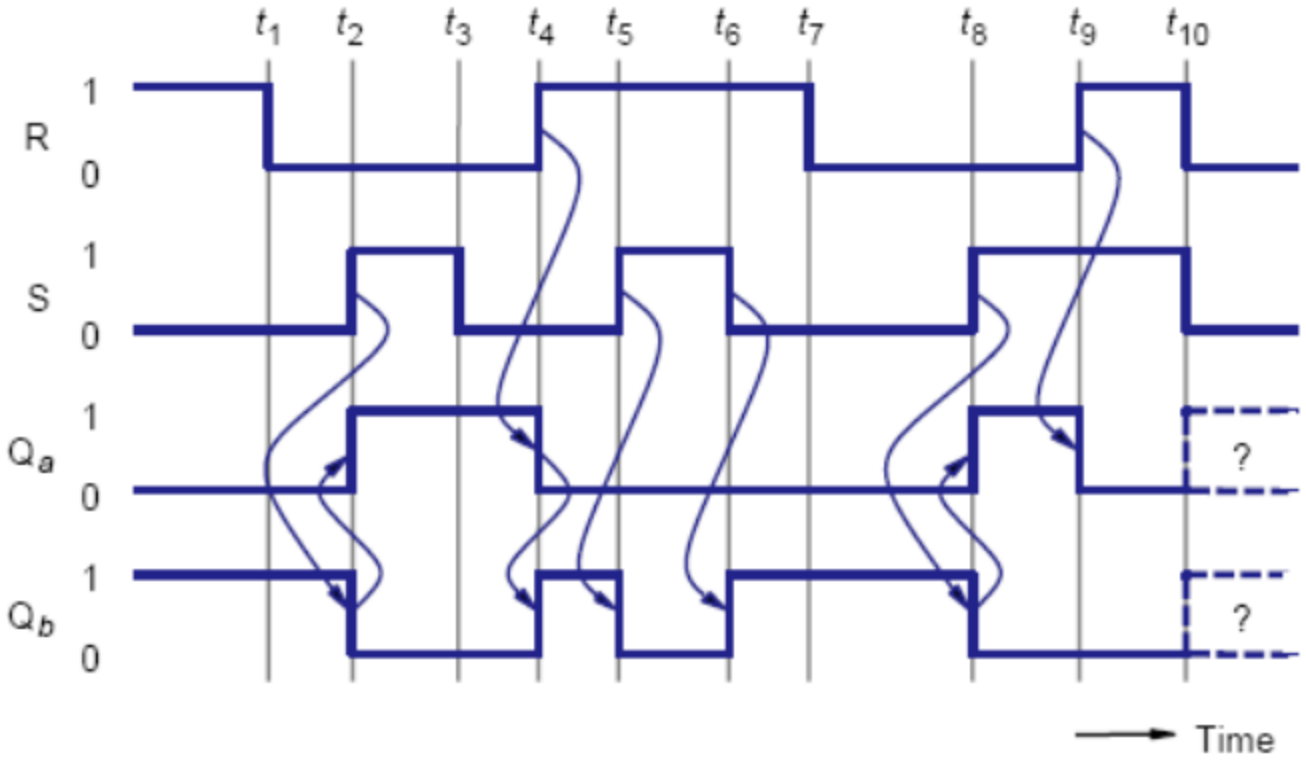



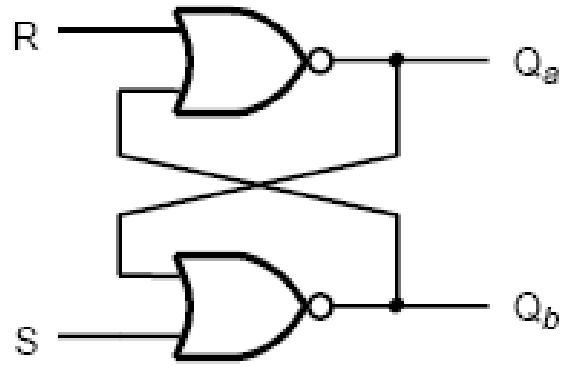
# Latch SR com portas NOR




| S | R | $Q_a$ | $Q_b$ | |
|---|---|---|---|---|
| 0 | 0 | 0/1 | 1/0 | (no change) |
| 0 | 1 | 0 | 1 | |
| 1 | 0 | 1 | 0 | |
| 1 | 1 | 0 | 0 | |



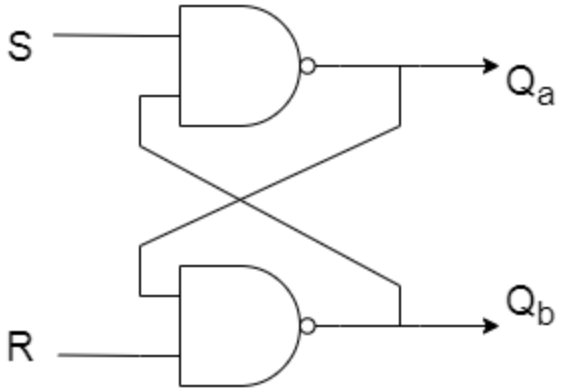
# Latch SR com portas NOR

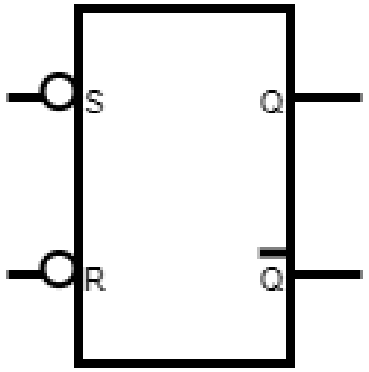



| S | R | $Q_a$ | $Q_b$ | |
|---|---|---|---|---|
| 0 | 0 | 0/1 | 1/0 | (no change) |
| 0 | 1 | 0 | 1 | |
| 1 | 0 | 1 | 0 | |
| 1 | 1 | 0 | 0 | |

- Um comportamento oscilante se apresenta quando S e R mudam de '1' para '0' exatamente ao mesmo tempo. Se as duas portas lógicas possuem o mesmo delay então aparecerá um '0' nas duas saídas exatamente ao mesmo tempo.
- Este estado será realimentado produzindo um '1' nas saídas exatamente ao mesmo tempo e posteriormente um '0', depois '1', etc...
- Este comportamento oscilante continuará indefinidamente até parar em um dos estados possíveis ('0' ou '1'). Comportamento aleatório!

# Latch SR com portas NAND

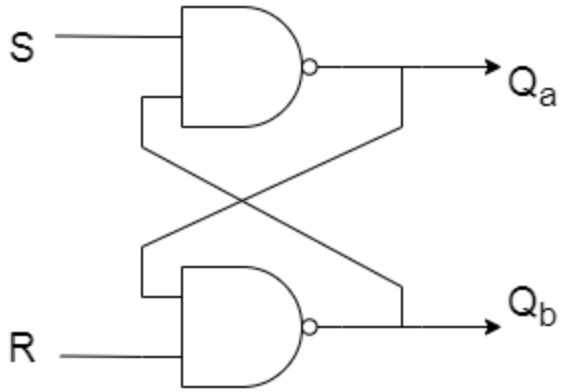


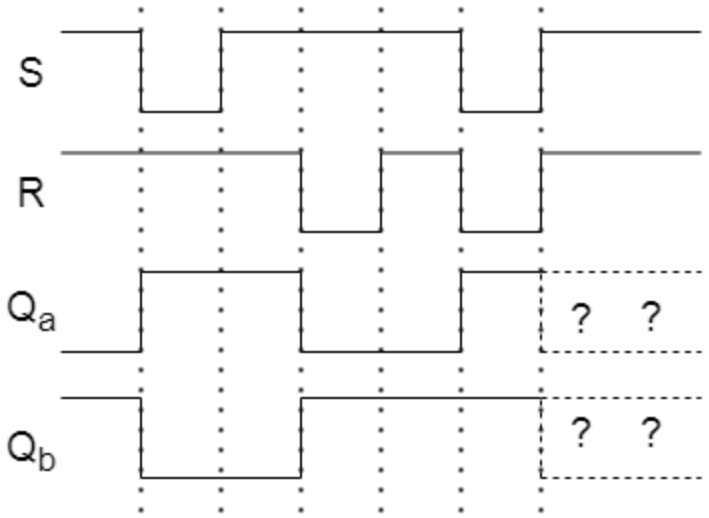



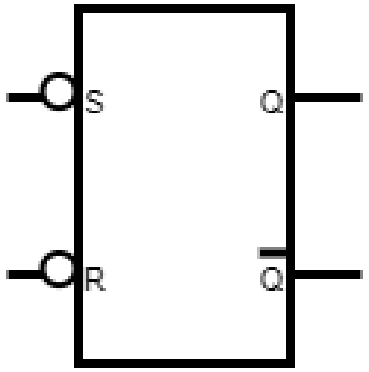

- Exercício 1: analisar o circuito para S=R=‘1’, assumindo que os valores iniciais

  Qa=‘0’, Qb= ‘1’

# Latch SR com portas NAND

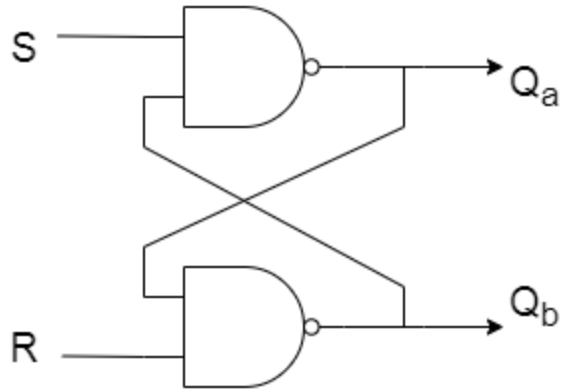


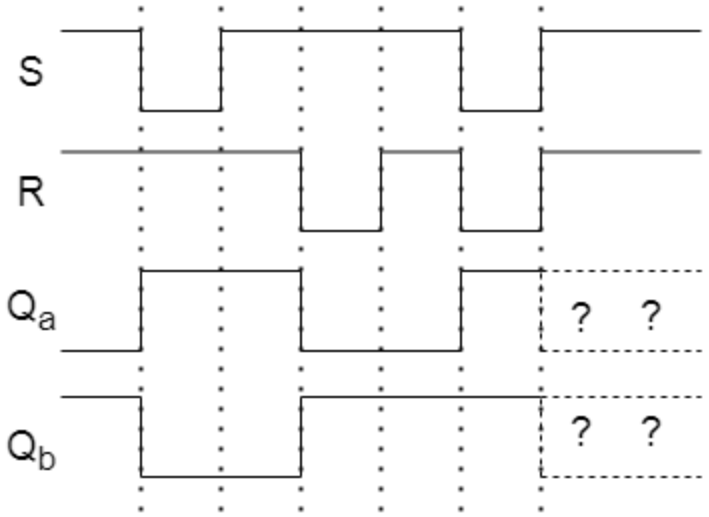




- Comportamento oscilante se apresenta quando S e R mudam de '0' para '1' exatamente ao mesmo tempo.
- Se o delay das portas e o comprimento dos fios forem os mesmos, não se sabe quando o circuito vai estabilizar, nem em qual estado.

# Latch SR com portas NAND

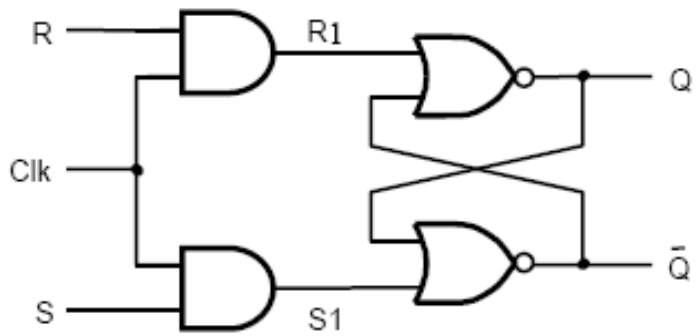



| S | R | $Q_a$ | $Q_b$ |
|---|---|---|---|
| 1 | 1 | No change | |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 0 | Invalid/not used | |

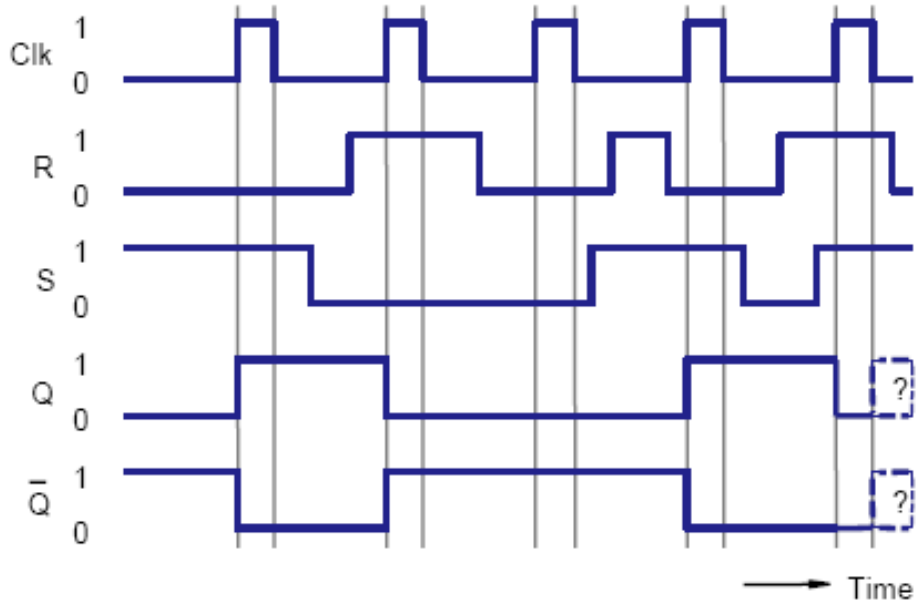
- Comportamento oscilante se apresenta quando S e R mudam de '0' para '1' exatamente ao mesmo tempo.
- Se o delay das portas e o comprimento dos fios forem os mesmos, não se sabe quando o circuito vai estabilizar, nem em qual estado.

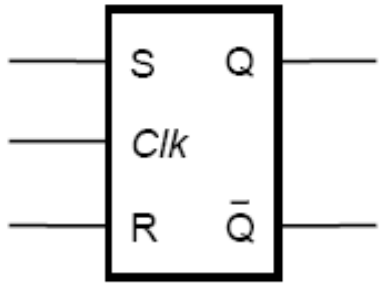
# Latch SR chaveado



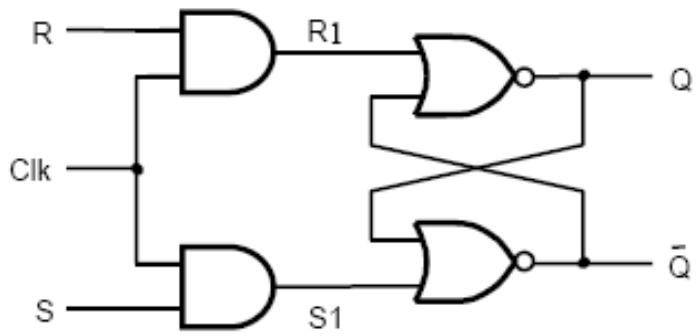

| Clk | S | R | Q(t + 1) |
|---|---|---|---|
| 0 | x | x | Q(t) (no change) |
| 1 | 0 | 0 | Q(t) (no change) |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | x |

Por que?



# Latch SR chaveado

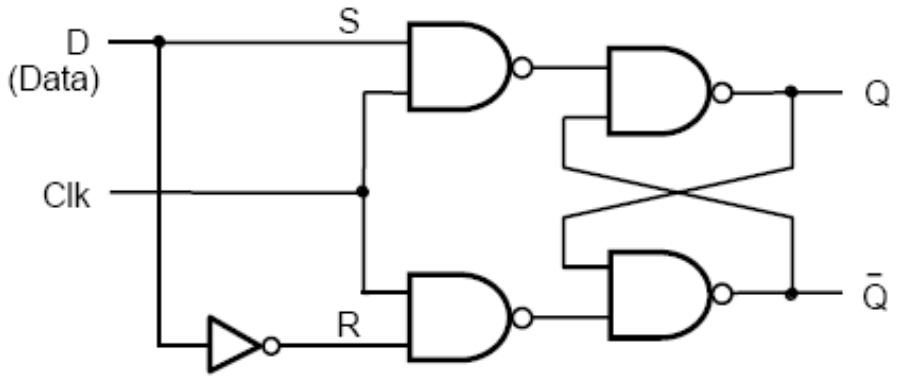



| Clk | S | R | Q(t + 1) |
|---|---|---|---|
| 0 | x | x | Q(t) (no change) |
| 1 | 0 | 0 | Q(t) (no change) |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | x |

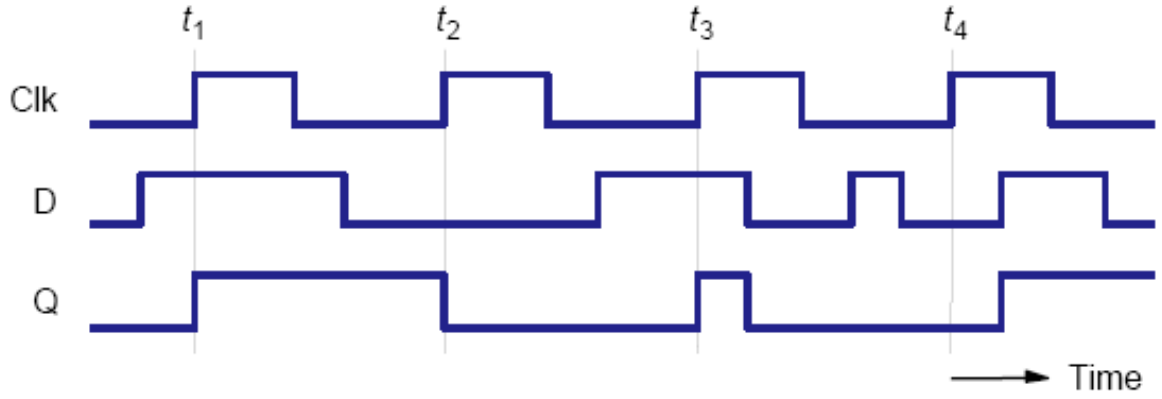
Por que?

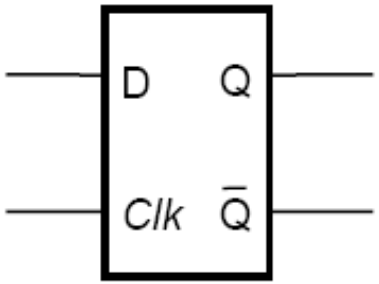
- Observe-se que quando clk=S=R='1' as saídas assumem os valores Q=Q'='0', o que não é permitido, pois Q' é a negação de Q.
- Ainda se apresenta o comportamento oscilante quando S e R mudam de '1' para '0' ao mesmo tempo. Se o delay das portas for o mesmo, o comportamento oscilante permanece. Na prática não sabemos qual qual porta é mais rápida do que a outra. Portanto, não se sabe qual será o estado do Latch. Assim, é dito que o estado é *indefinido*.

# Latch D chaveado

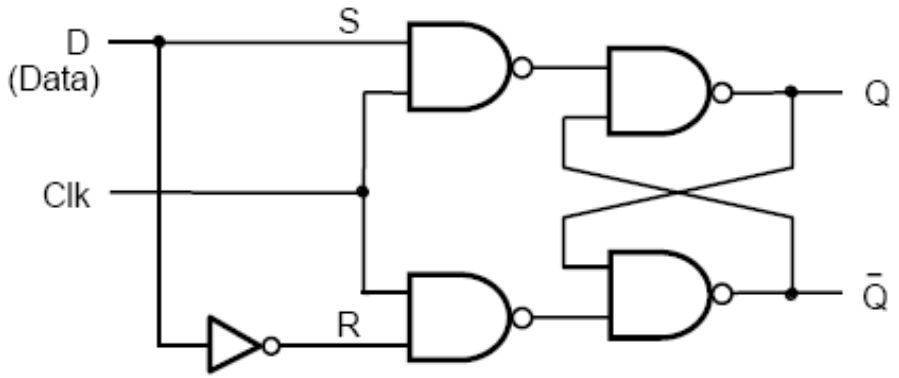



| Clk | D | $Q(t+1)$ |
|---|---|---|
| 0 | x | $Q(t)$ |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

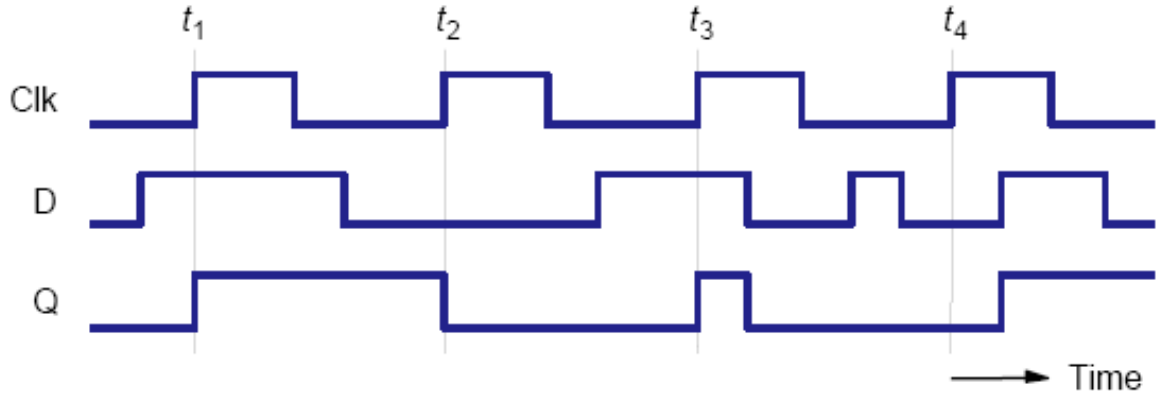


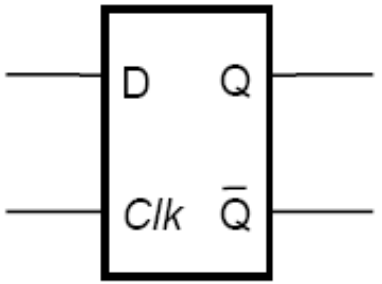



# Latch D chaveado



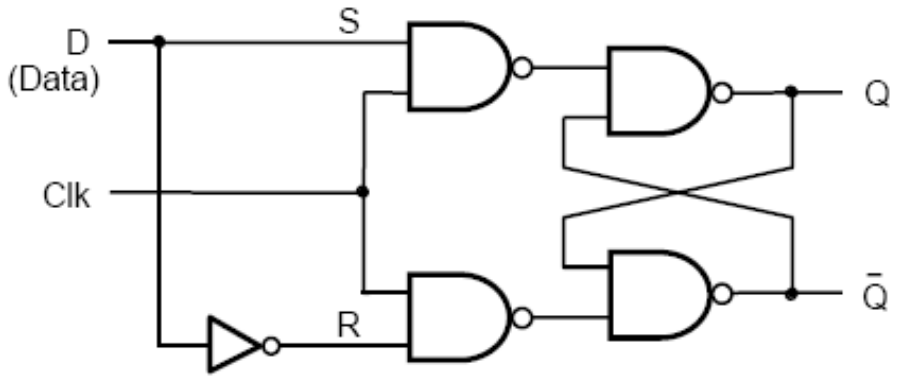

| Clk | D | Q(t + 1) |
|---|---|---|
| 0 | x | Q(t) |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



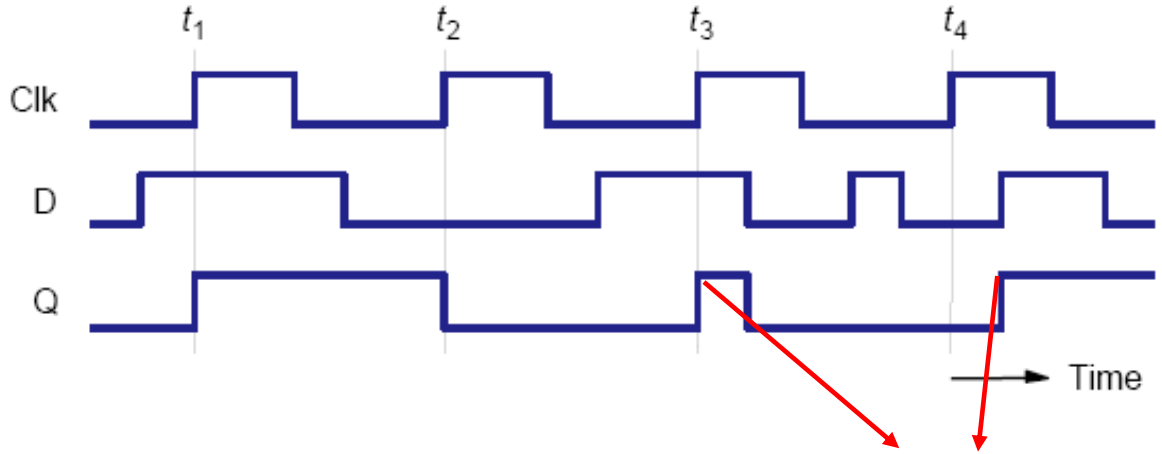



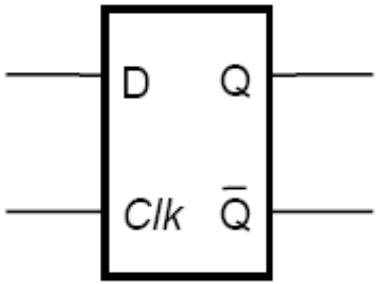

A porta negadora impede que S e R tenham o mesmo valor, estabilizando o Latch. Assim, elimina-se a condição proibida e o comportamento oscilante.

# Latch D chaveado




| Clk | D | Q(t + 1) |
|---|---|---|
| 0 | x | Q(t) |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

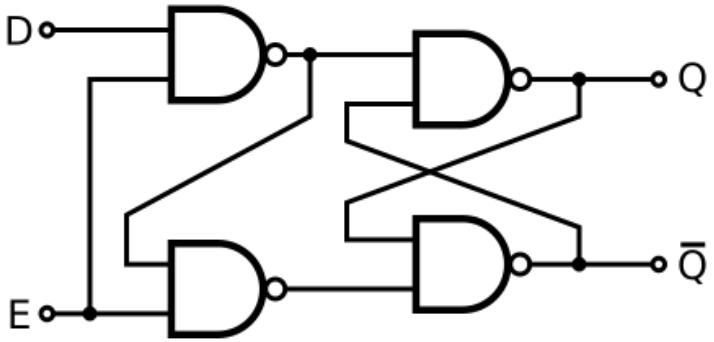


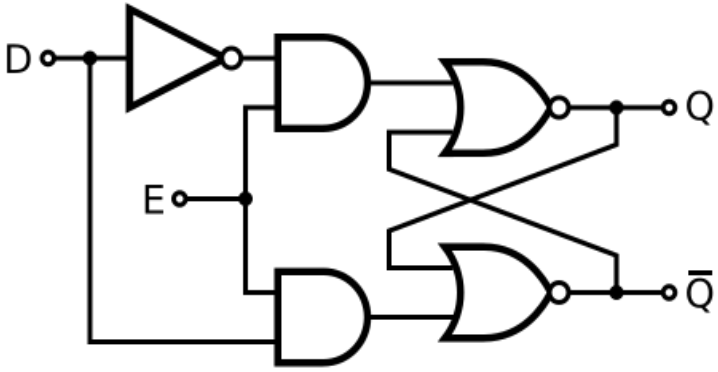




Observe-se que a saída é sensível ao **nível** do clock. Se clk='1' o latch está aberto e variações na entrada podem apresentar-se na saída.

# Latch D chaveado

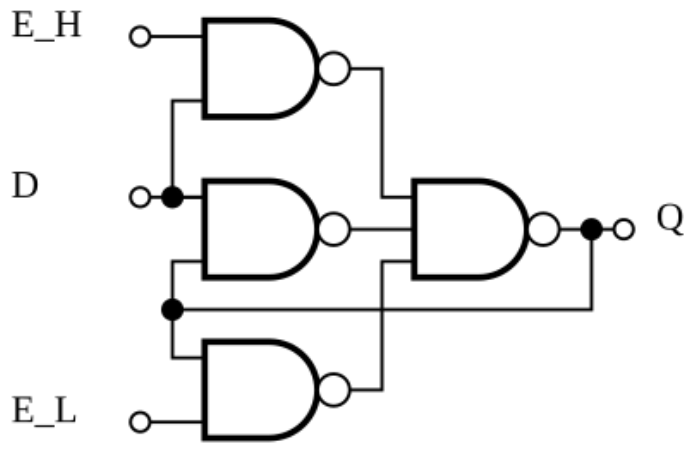
- O Latch D resolve o problema da condição proibida, porém apresenta algumas características indesejáveis.
- Observe-se que o tempo da entrada para a saída não é constante, uma saída requer dois atrasos de portas lógicas enquanto a outra saída requer três atrasos. Essa diferença do tempo de propagação na saída produz hazards (azares) que se manifestam como glitches (mudanças rápidas nas saídas), as quais são indesejadas em algumas aplicações (exemplo: eletrônica de alta frequência).



# Latch Earle

- O Latch Earle requer apenas dois níveis de lógica.
- Possui apenas uma saída e tempo de propagação de dois gate delays.
- Usa enable (ou clock) complementário: enable ativo em alto (E_H) e enable ativo em baixo (E_L)
- Latch Earle é livre de hazards por ter apenas dois níveis de lógica.



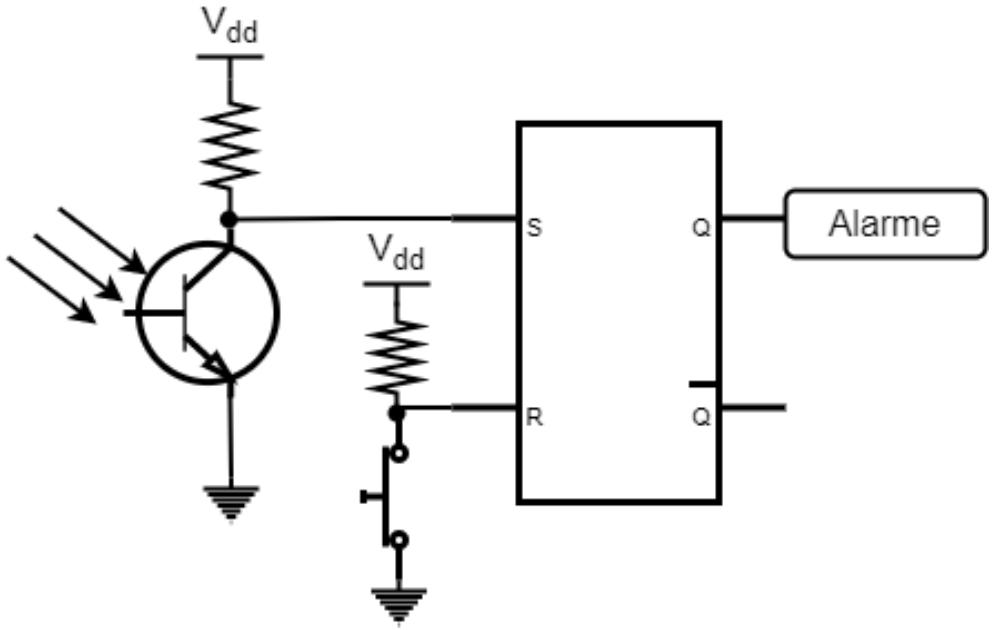
# Exercícios Aula 1 – Latches

1) Qual é o estado de repouso (efeito memória) do Latch SR com NOR?
2) Qual é o estado de repouso (efeito memória) do Latch SR com NAND?
3) Qual é o estado ativo das entradas no Latch NOR? e no Latch NAND?
4) Verdeiro ou falso: no Latch SR com NOR, se S=R='1' Qa='1' e Qb='0'
5) Qual é a condição oscilante do Latch SR com NOR e com NAND?
6) Verdadeiro ou falso: no Latch SR chaveado não existe mais o comportamento oscilante.
7) Implemente um Latch SR chaveado com portas NAND.
8) Porque o Latch D não tem comportamento oscilante (condição proibida)?

# Exercícios Aula 1 – Latches

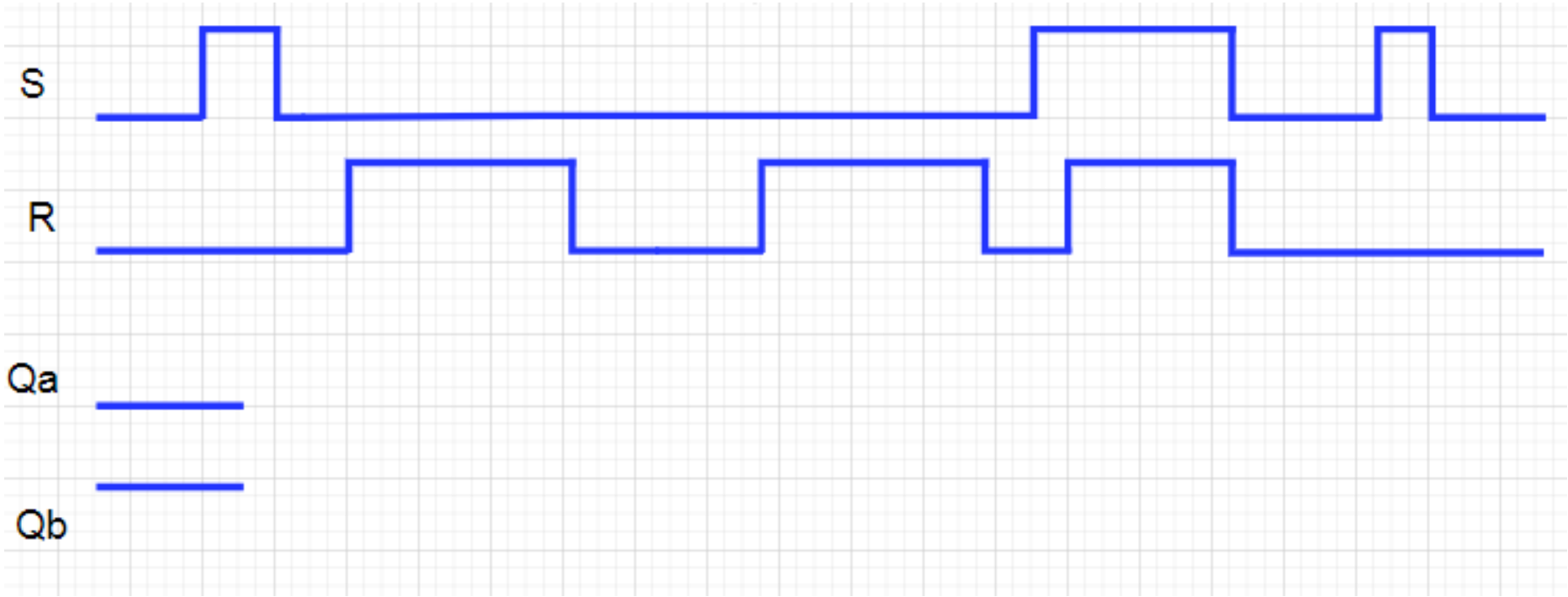
9) Para o circuito abaixo, assuma um latch SR com NOR e que o valor inicial Q='0' (chave pressionada). Inicialmente o fototransistor recebe um feixe de luz, analise o comportamento do circuito antes e depois de soltar a chave. Em seguida, analise o circuito se o feixe de luz for interrompido.

10) Implemente o circuito usando um Latch SR com NAND.



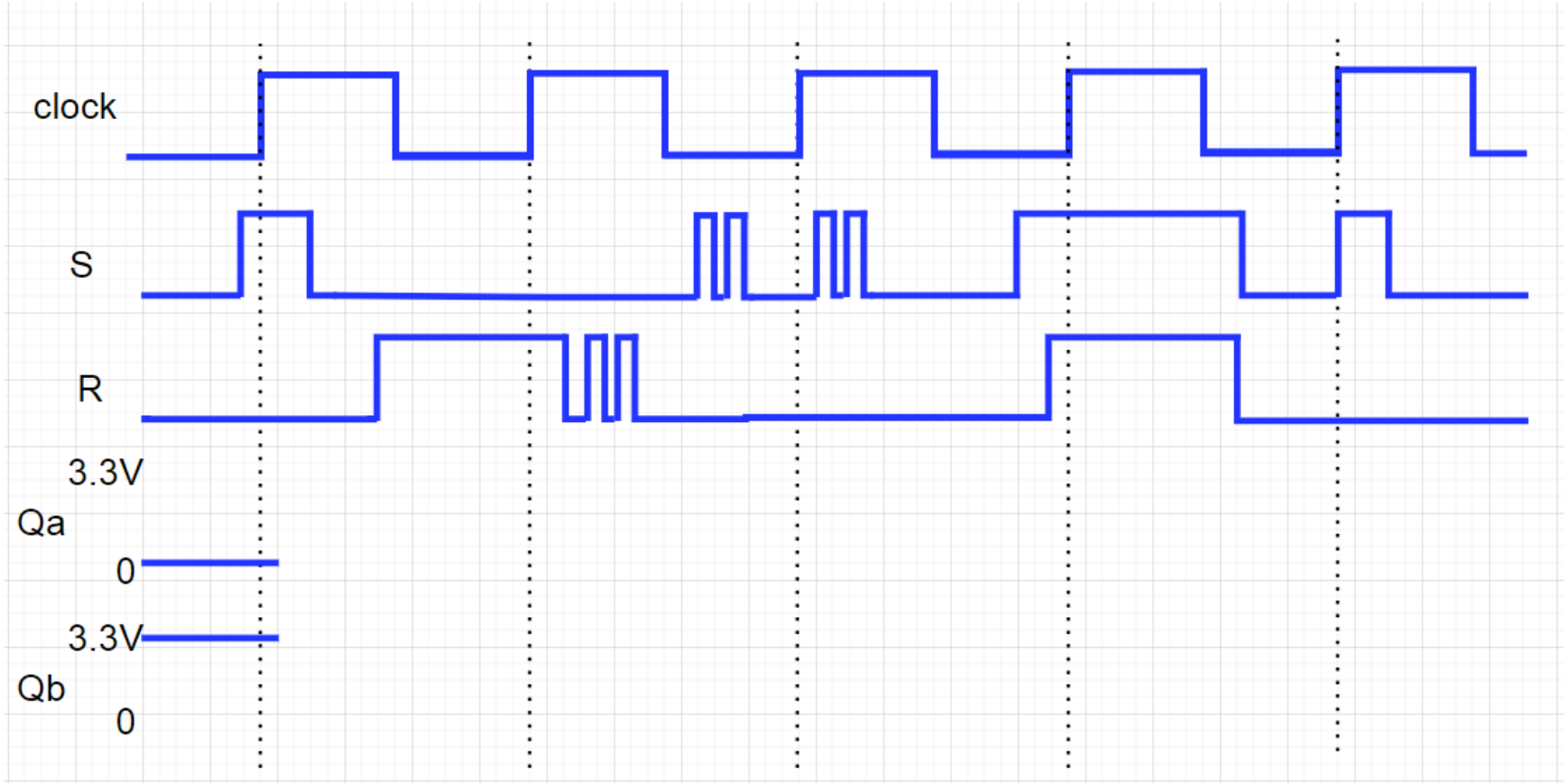
# Exercícios Aula 1 – Latches

12) Assuma que o valor inicial de Qa='0' e Qb='1', complete a forma de onda para um Latch SR com NOR. Repita o exercício com para o Latch SR com NAND.



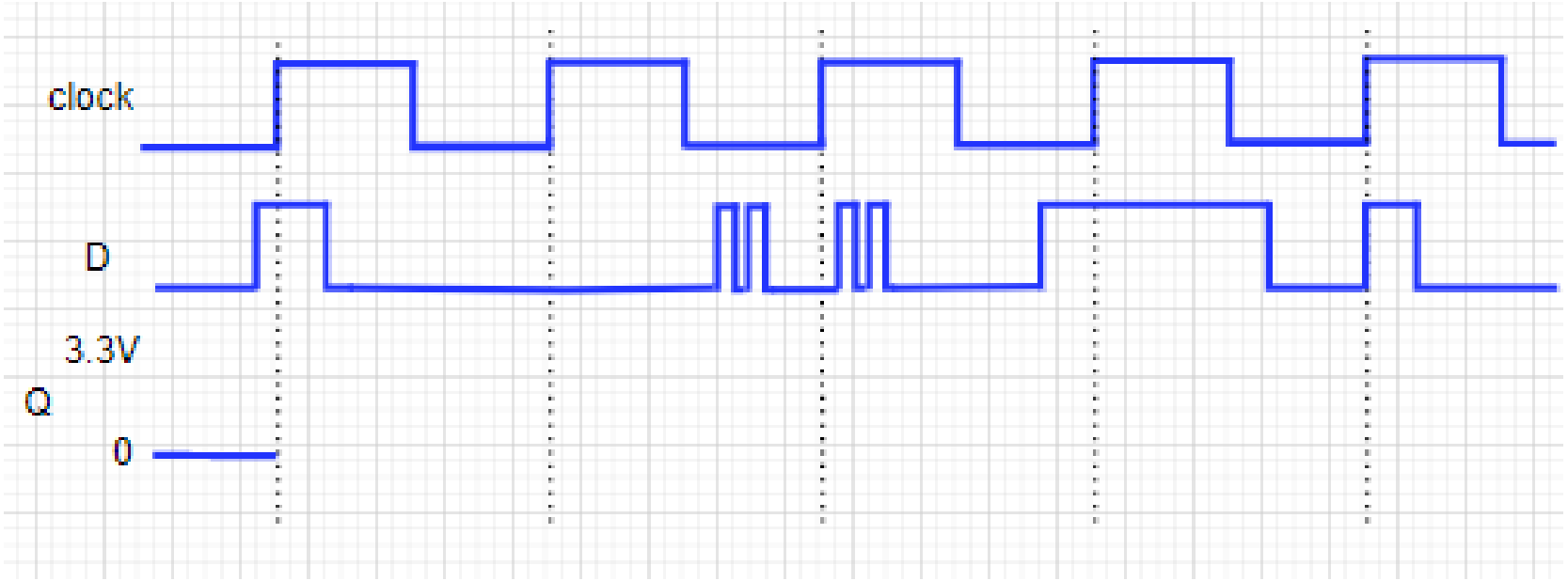
# Exercícios Aula 1 – Latches

13) Para o Latch SR chaveado com portas NOR, complete o diagrama abaixo.



# Exercícios Aula 1 – Latches

13) Para o Latch D com portas NAND, complete o diagrama abaixo.



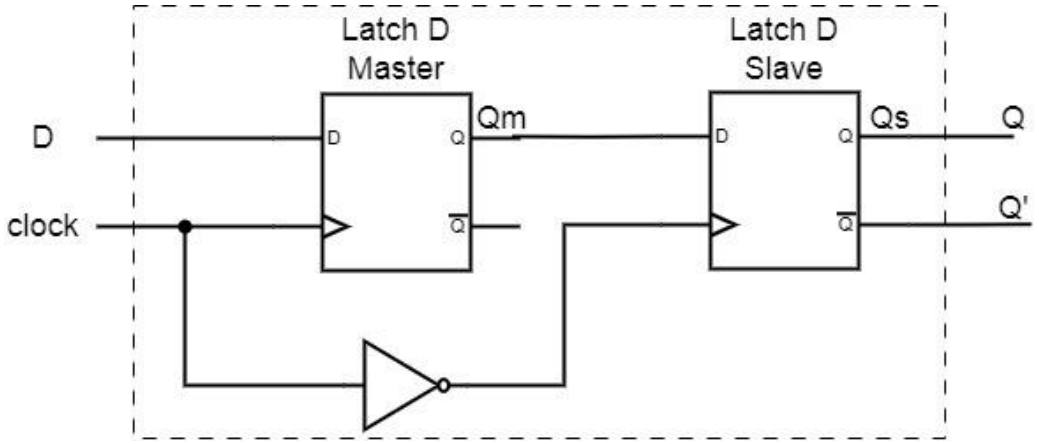
Página em branco.

# Aula 2
# Flip-Flops

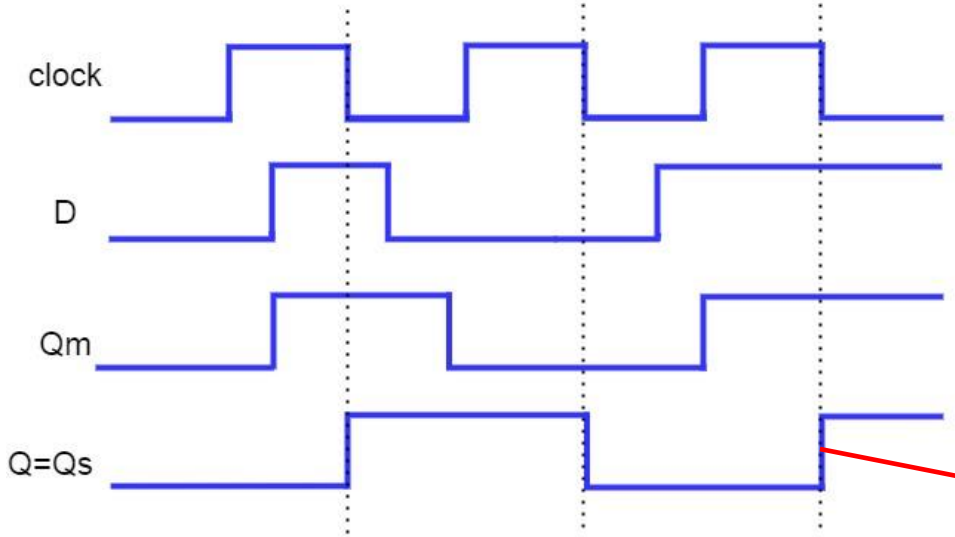
# Flip-Flops

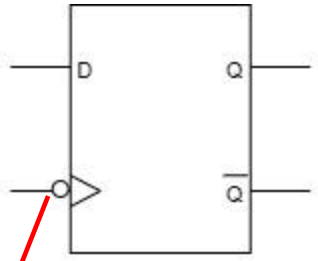
- O Flip-Flop é um circuito digital sequencial que armazena um (1) bit de informação.
- Diferente do Latch (sensível ao nível do clock), o Flip-flop é sensível à borda do clock.
- As aplicações mais comuns dos Flip-Flops são:
  - Registradores.
  - Registradores de deslocamento (amplamente usados em operações aritméticas e comunicações digitais).
  - Contadores.
  - Memórias.
  - Circuitos sequenciais.

# Flip-Flop tipo D sensível a borda de descida



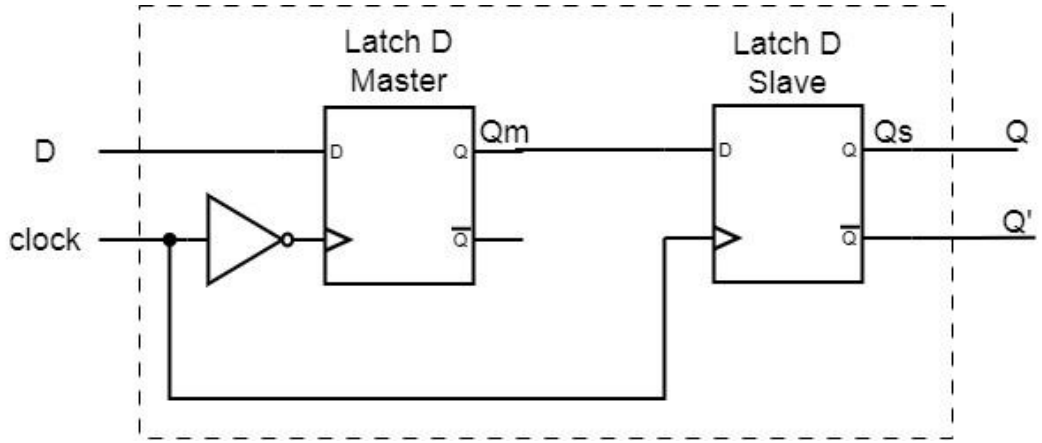

| D | CLK | Modo | Saída |
|---|---|---|---|
| 0 | ↓ | Armazena '0' | Q(n+1)=0 |
| 1 | ↓ | Armazena '1' | Q(n+1)=1 |
| X | | Hold (memória) | Q(n+1)=Q(n) |



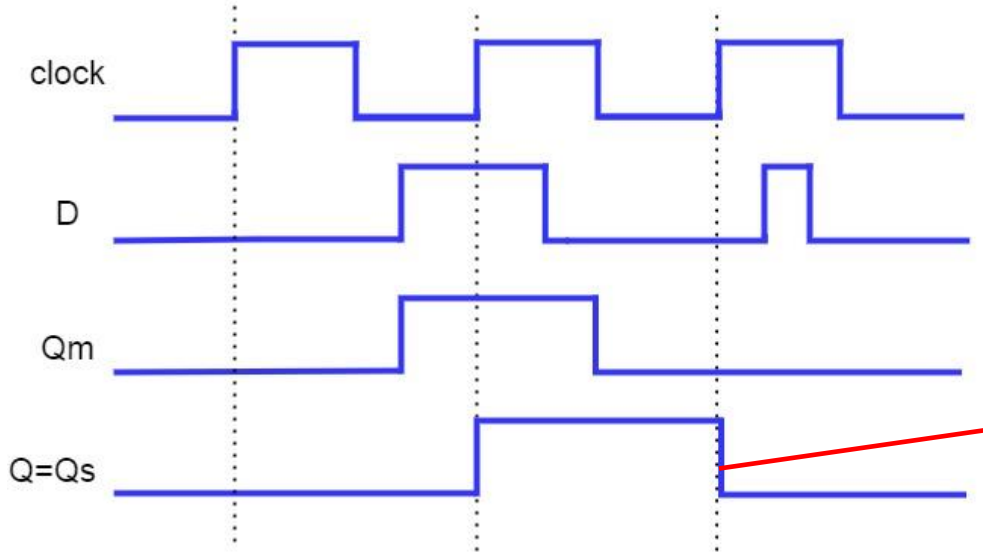



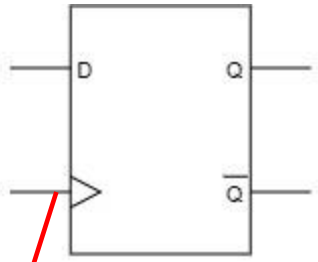

Flip-Flop D
sensível à borda de descida

# Flip-Flop tipo D sensível a borda de subida



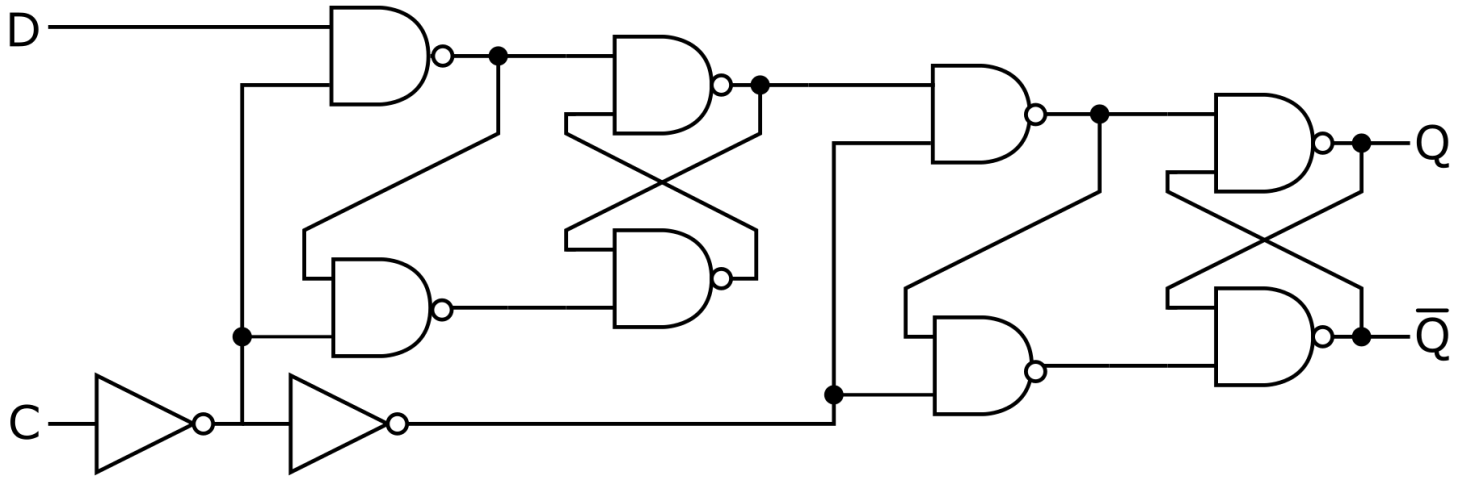

| D | CLK | Modo | Saída |
|---|---|---|---|
| 0 | ↑ | Armazena '0' | Q(n+1)=0 |
| 1 | ↑ | Armazena '1' | Q(n+1)=1 |
| X | | Hold (memória) | Q(n+1)=Q(n) |

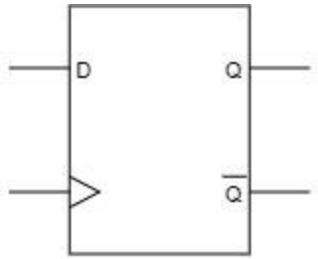



Flip-Flop D
sensível à borda de subida

# FFD sensível a borda de subida com portas NAND




Flip-Flop D
sensível à borda de subida

# Flip-flop D com entradas Set e Reset

- Alguns Flip-Flops D em circuitos integrados possuem a capacidade de forçar os estados de set e reset, i.e. ignorando as entradas D e clock.
- Para tal são usadas as entradas assíncronas de Set e Reset.
- A entrada Reset do Flip-Flop apaga o conteúdo (clear). Esta entrada pode ser sensível em nível alto ou baixo.
- A entrada Set do Flip-Flop força o conteúdo em nível lógico alto. Esta entrada pode ser sensível em nível alto ou baixo.
- Existem Flip-Flops com entradas Set e Reset que podem ser sensíveis aos níveis lógicos alto e/ou baixo.
- Vejamos alguns exemplos:

# Flip-flop D com entradas Set e Reset

Símbolo esquemático Flip-Flops D com Set e Reset

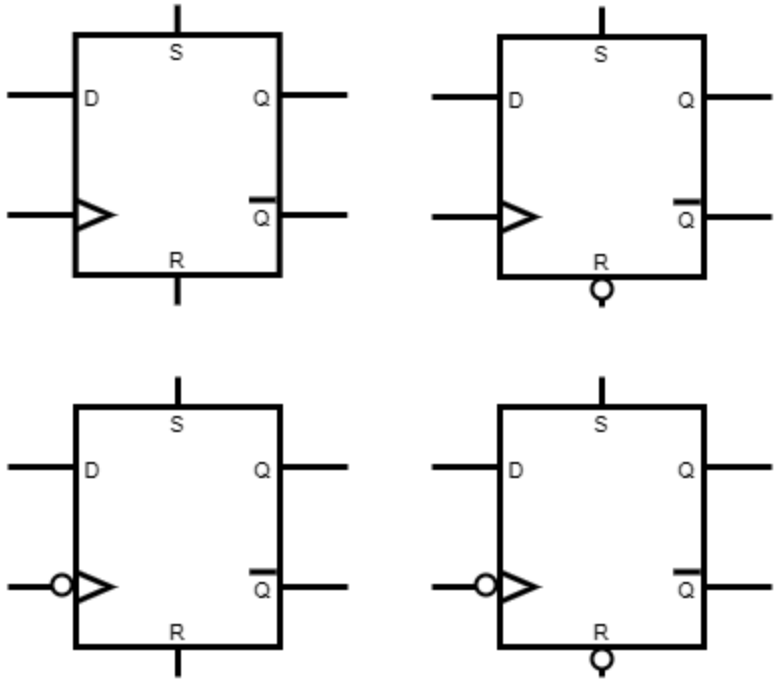


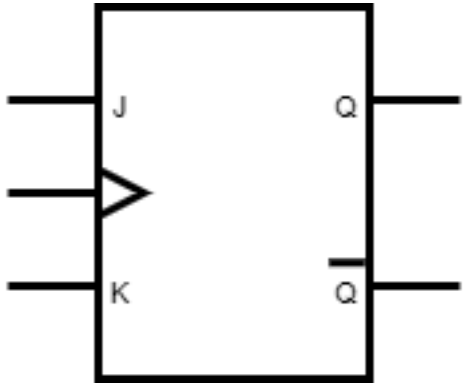

Tabela característica FF D sensível à borda de subida com Set e Reset em nível alto:

**D Flip Flop Truth Table**

| CL (Note 1) | D | R | S | Q | $\overline{Q}$ |
|---|---|---|---|---|---|
| ⌐ | 0 | 0 | 0 | 0 | 1 |
| ⌐ | 1 | 0 | 0 | 1 | 0 |
| ⌐ | x | 0 | 0 | Q | $\overline{Q}$ |
| x | x | 1 | 0 | 0 | 1 |
| x | x | 0 | 1 | 1 | 0 |
| x | x | 1 | 1 | 1 | 1 |

No Change
x = Don't Care Case
**Note 1:** Level Change

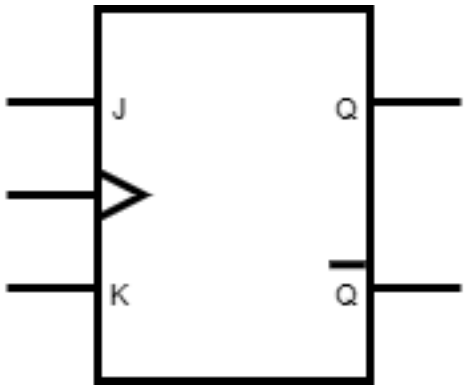
# Flip-Flop JK

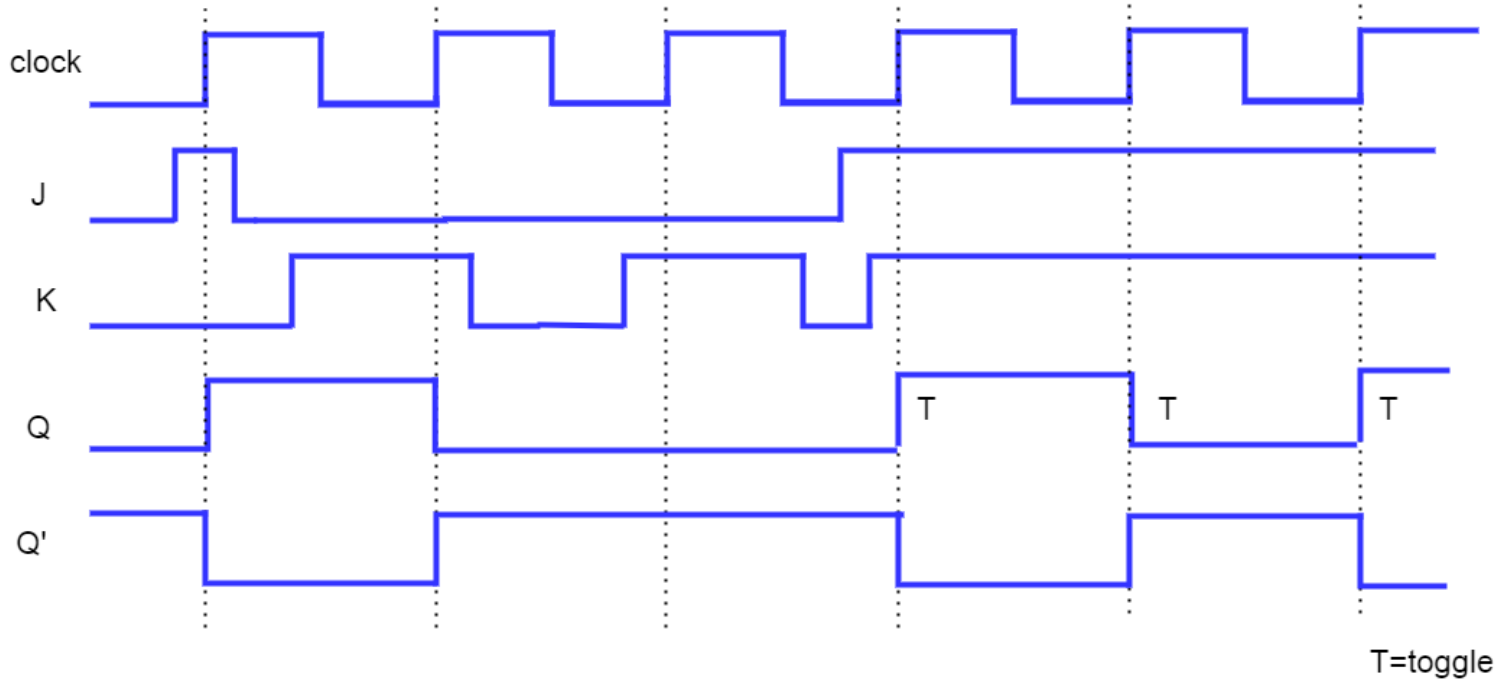



| J | K | CLK | Saída - Próximo estado Q(n+1) |
|---|---|---|---|
| 0 | 0 | ↑ | Q(n) (saída não muda) |
| 1 | 0 | ↑ | 1 |
| 0 | 1 | ↑ | 0 |
| 1 | 1 | ↑ | Q'(n) (comuta=toggle) |

J: entrada de set.
K: entrada de reset.
Clk: clock
Q (Saída): estado atual do flip-flop.
Q' (Saída complementar): estado oposto ao de Q.

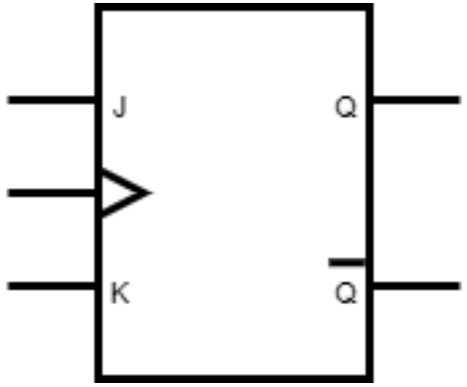
# Flip-Flop JK




J: entrada de set.
K: entrada de reset.
Clk: clock
Q (Saída): estado atual do flip-flop.
Q' (Saída complementar): estado oposto ao de Q.

# Flip-Flop JK

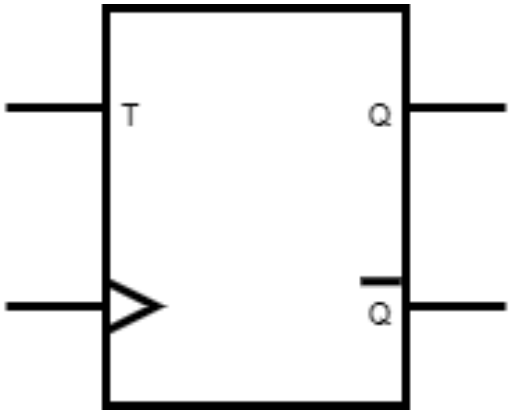



| J | K | CLK | Saída - Próximo estado Q(n+1) |
|---|---|---|---|
| 0 | 0 | ↑ | Q(n) (saída não muda) |
| 1 | 0 | ↑ | 1 |
| 0 | 1 | ↑ | 0 |
| 1 | 1 | ↑ | Q'(n) (comuta=toggle) |

Equação característica do Flip-flop JK:
Q(n+1)=JQ'(n)+K'Q(n)

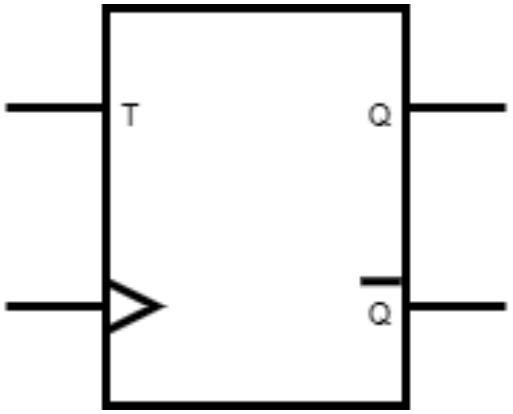
Q(n) é o estado atual.
Q(n+1) é o próximo estado.
Toggle: alterna entre 0 e 1.

# Flip-Flop T (toogle)




- Se T = '1', a saída muda de estado sempre que ocorrer uma borda no clock.
- Se T= '0', a saída mantem o valor anterior.
- Pode ser implementado com um Flip-flop JK com as entradas J=K='1'

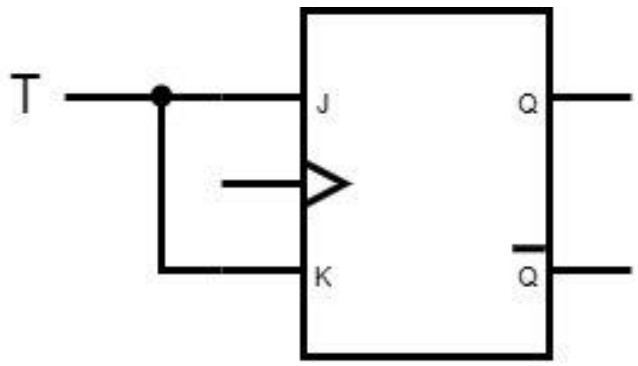
# Flip-Flop T (toogle)

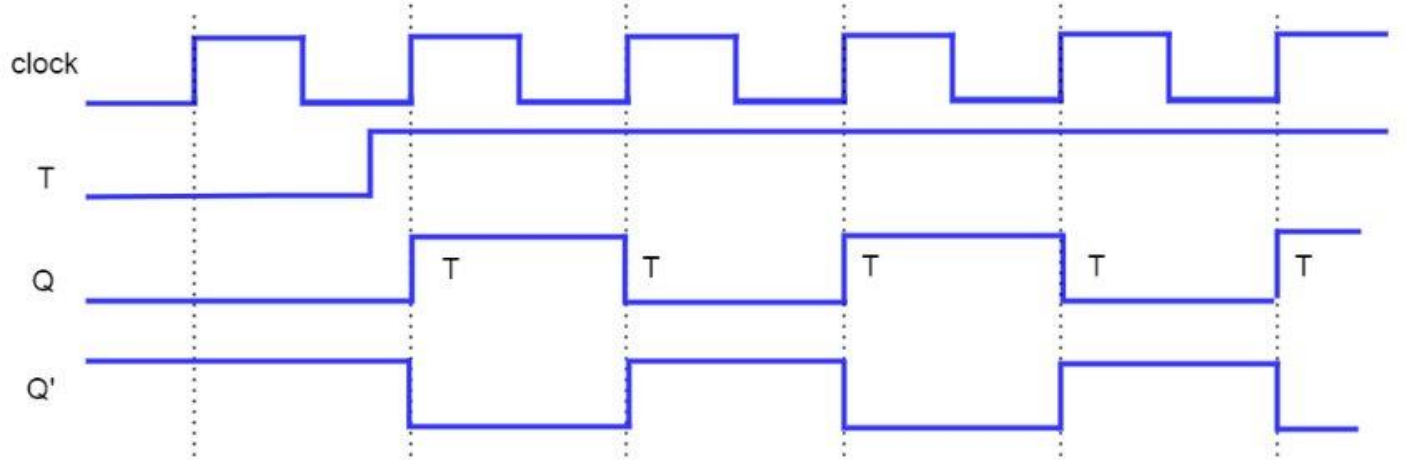



- Se T = '1', a saída muda de estado sempre que ocorrer uma borda no clock.
- Se T= '0', a saída mantem o valor anterior.
- Pode ser implementado com um Flip-flop JK com as entradas J=K='1'

| T | CLK | Modo | Q(n) | Q(n+1) |
|---|---|---|---|---|
| X | 0 ou 1 | Memória | 0/1 | 0/1 |
| 0 | ↑ | Memória | 0 | 0 |
| 0 | ↑ | Memória | 1 | 1 |
| 1 | ↑ | Toggle | 0 | 1 |
| 1 | ↑ | Toggle | 1 | 0 |

# Flip-Flop T (toogle)

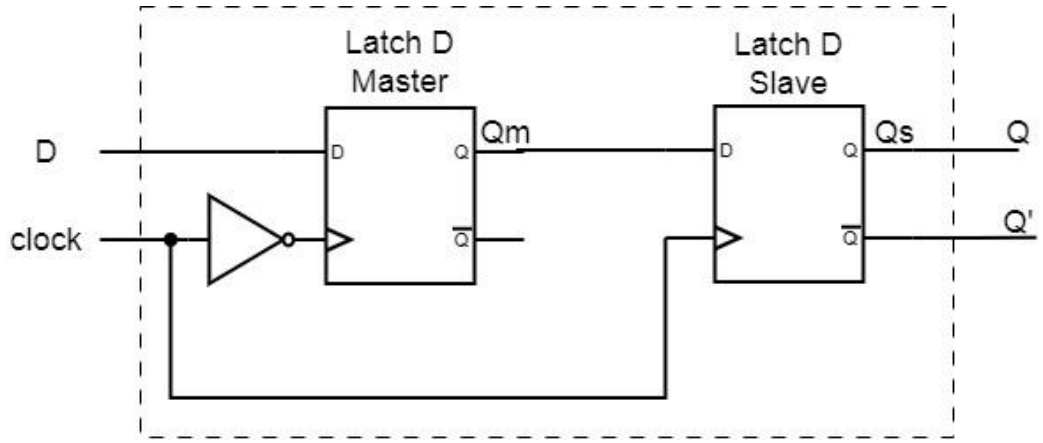


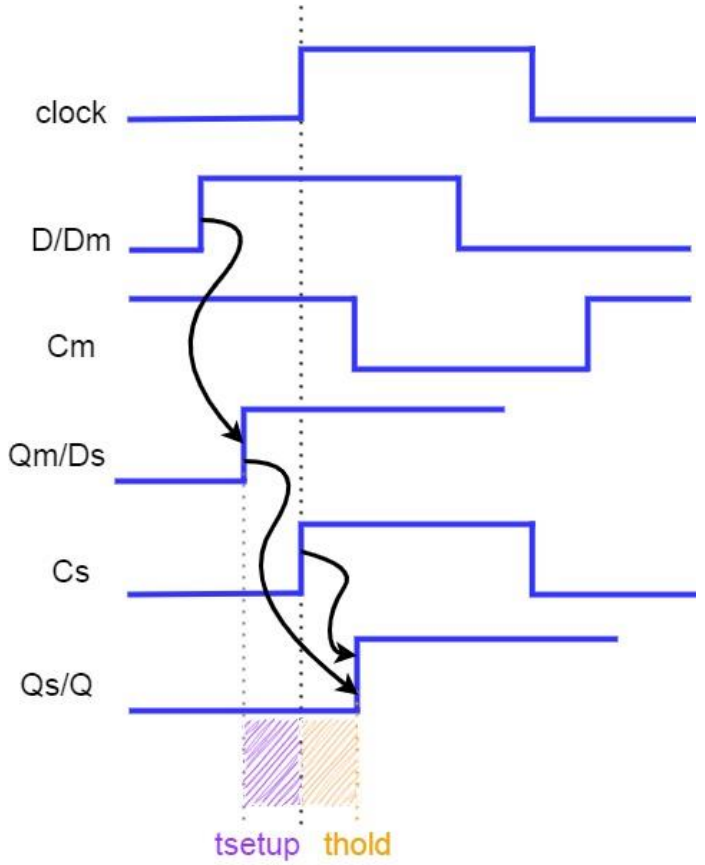




T=toggle

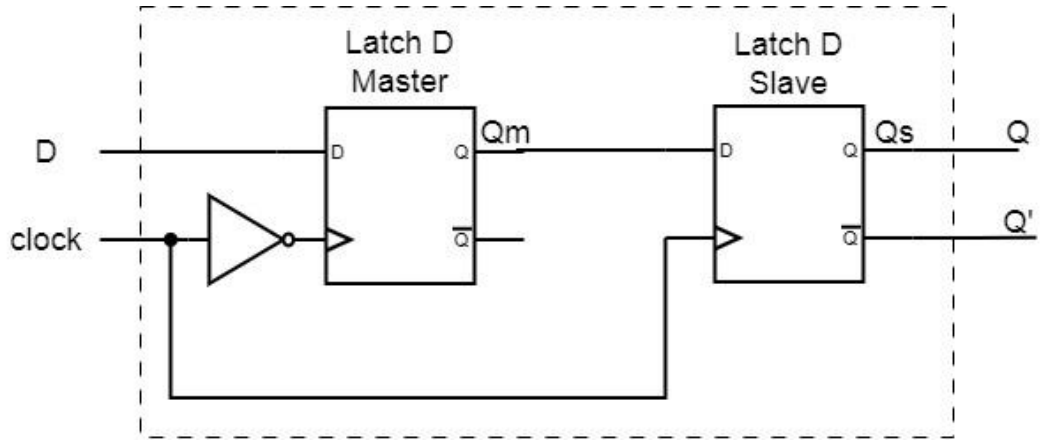
| T | CLK | Modo | Q(n) | Q(n+1) |
|---|---|---|---|---|
| X | 0 ou 1 | Memória | 0/1 | 0/1 |
| 0 | ↑ | Memória | 0 | 0 |
| 0 | ↑ | Memória | 1 | 1 |
| 1 | ↑ | Toggle | 0 | 1 |
| 1 | ↑ | Toggle | 1 | 0 |

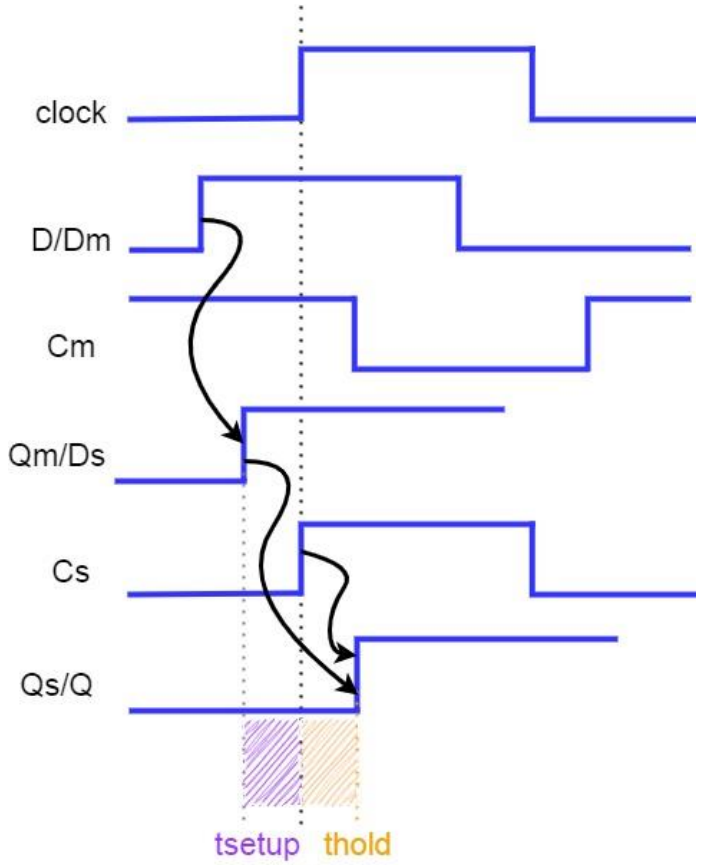
# Parâmetros de temporização de um Flip-Flop



# Parâmetros de temporização de um Flip-Flop

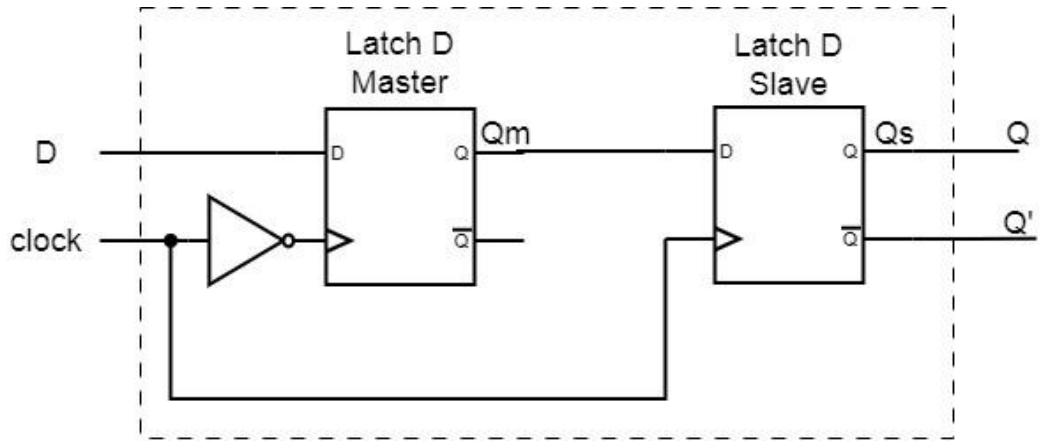


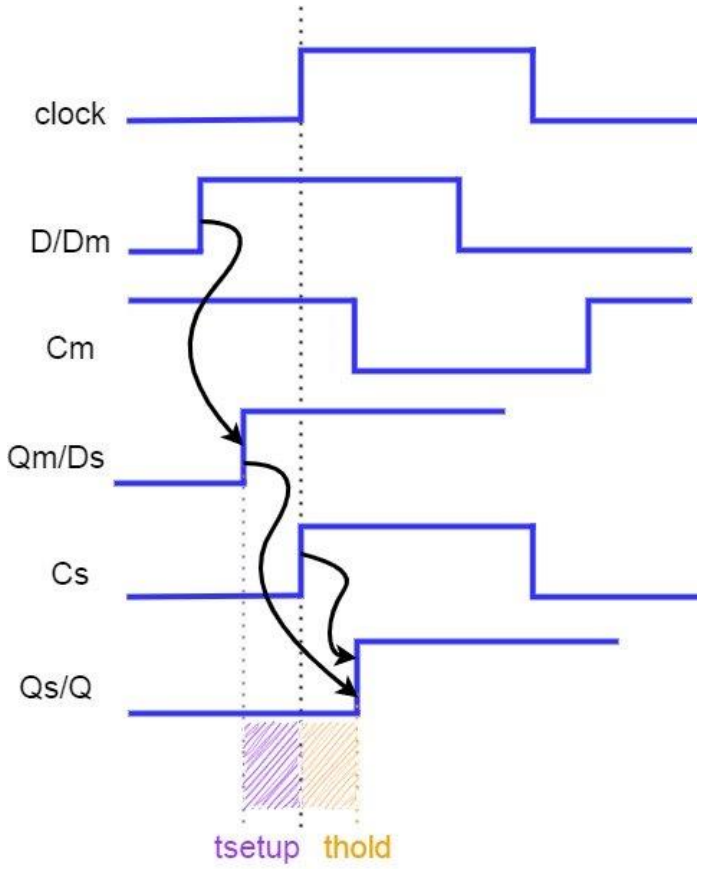

**Tempo de setup** (ts): tempo que o dado de entrada D do flip-flop deve permanecer estável **antes** da borda de clock.

**Tempo de hold** (th): tempo que o dado de entrada D do flip-flop deve permanecer estável **depois** da borda de clock.

# Parâmetros de temporização de um Flip-Flop

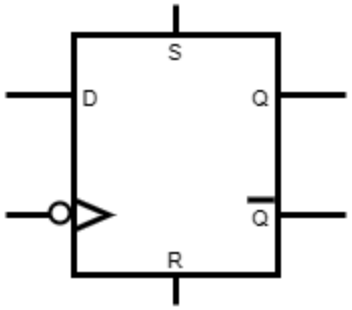


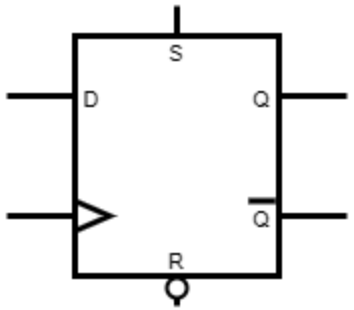

**Nota:** caso o dado de entrada D mude dentro da faixa do tempo de setup ou do tempo de hold, pode haver perda de informação, isto é, não há garantia que o dado será registrado no Flip-Flop.

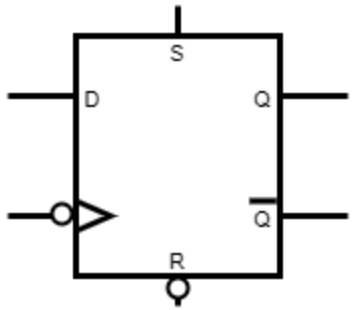


# Exercícios Aula 2 – Flip-flops

Exercício 1: realize a tabela verdade (tabela característica) dos seguintes Flip-Flops

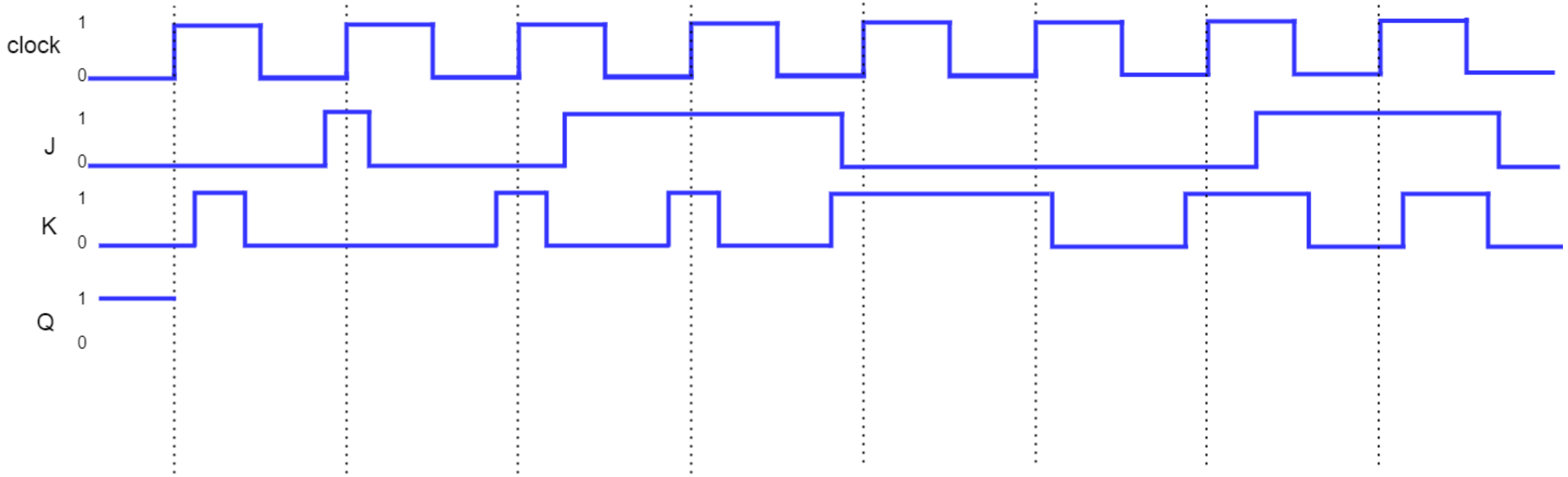


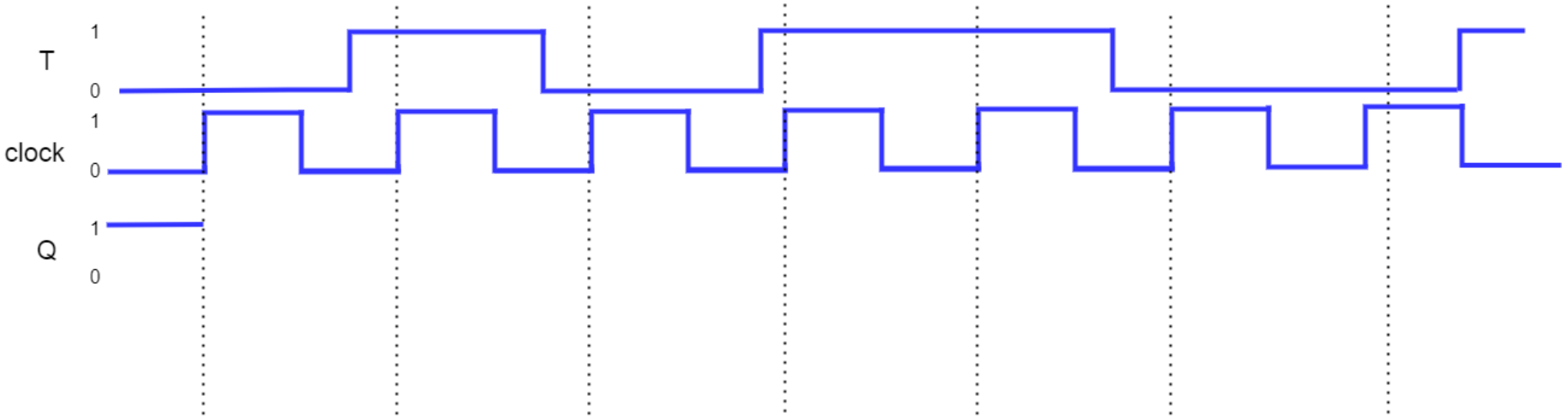
# Exercícios Aula 2 – Flip-flops

Exercício 2: Considere um Flip-Flop JK e complete o seguinte diagrama de tempo



# Exercícios Aula 2 – Flip-flops

Exercício 3: Considere um Flip-FLop tipo T, complete o seguinte diagrama de tempo



# Exercícios Aula 2 – Flip-flops

Exercício 4: Como esse Flip-Flop D pode ser convertido em um Flip-Flop T?

Exercício 5: Construa um Flip-Flop tipo D um usando Flip-Flop JK e lógica combinacional adicional.

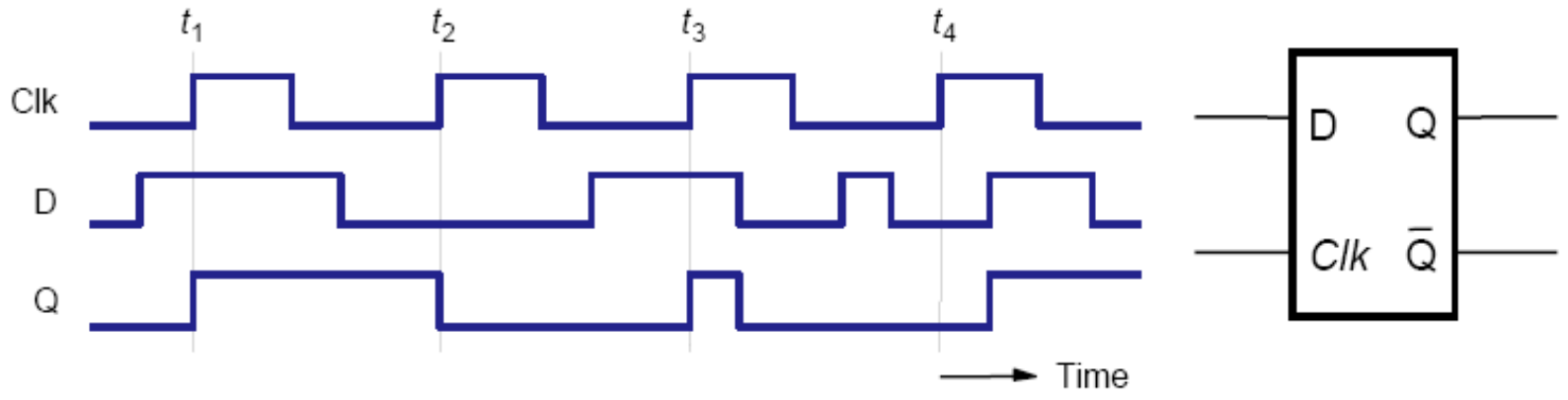
Exercício 6: Construa um Flip-Flop JK usando um Flip-Flop D e lógica combinacional adicional.

Página em branco.

# Aula 3
# Implementação de Latches em HDLs

# Revisão de Latches

- Latches tipo D são sensíveis ao nível de clock, ou seja, variações na entrada de dados passam para a saída enquanto o clock esteja ativo. Isto inclui variações de alta frequência na entrada de dados.
- Quando o clock é desativado (travamento do Latch D), variações na entrada não se manifestam na saída do Latch.



# Inferência de Latches em HDLs

- Em lógica sequencial, Latches são inferidos quando:
  1. Sentenças IF – ELSE IF ou CASE – WHEN são usadas por nível lógico.
  2. Todas as possibilidades de condicionais IF – ELSE IF ou CASE – WHEN não são explícitas na descrição de hardware.
- Nestes casos, a ferramenta de síntese e de simulação precisam manter a saída prévia (criar uma memória) sob certas condições se as sentenças ELSE ou WHEN OTHERS não são incluídas.

# Inferência de Latches em HDLs

- Exemplo1: inferência de Latch em Verilog com sentenças IF:

```
14  module latch_ex1_vlog (clk,D,Q);
15      input clk;
16      input D;
17      output reg Q;
18
19      always @(clk,D) begin
20          if (clk) begin
21              Q = D;
22          end
23      end
24
25  endmodule
```

# Inferência de Latches em HDLs

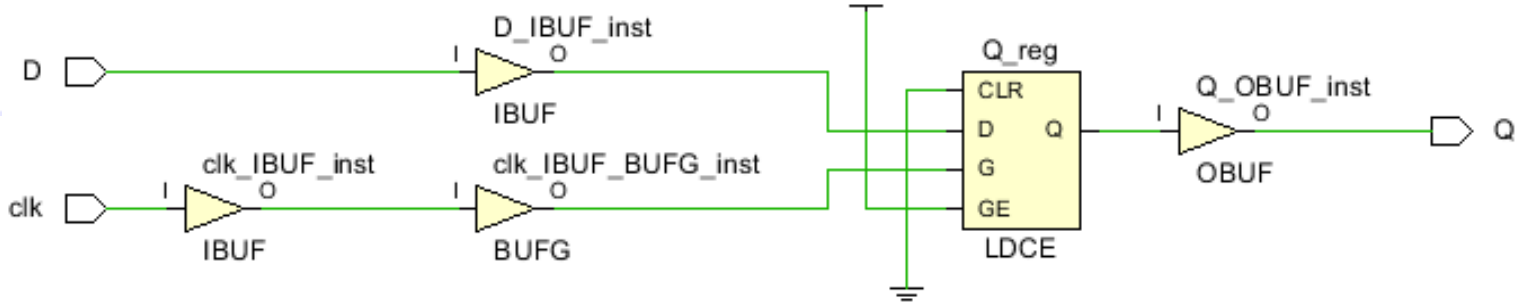
- Exemplo1: inferência de Latch em Verilog com sentenças IF:

```
14  module latch_ex1_vlog (clk,D,Q);
15      input clk;
16      input D;
17      output reg Q;
18
19      always @(clk,D) begin
20          if (clk) begin
21              Q = D;
22          end
23      end
24
25  endmodule
```

Synthesis (2 warnings, 14 infos)

- [Common 17-349] Got license for feature 'Synthesis' and/or device 'xc7a35t'
- [Synth 8-6157] synthesizing module 'latch_ex1_vlog' [latch_ex1_vlog.v:14]
- [Synth 8-6155] done synthesizing module 'latch_ex1_vlog' (1#1) [latch_ex1_vlog.v:14]
- [Device 21-403] Loading part xc7a35tcpg236-1
- [Synth 8-327] inferring latch for variable 'Q_reg' [latch_ex1_vlog.v:21]
- [Project 1-571] Translating synthesized netlist
- [Netlist 29-17] Analyzing 1 Unisim elements for replacement

Observe a inferência do lacth tipo D com clear e clock enable (LDCE)



# Inferência de Latches em HDLs

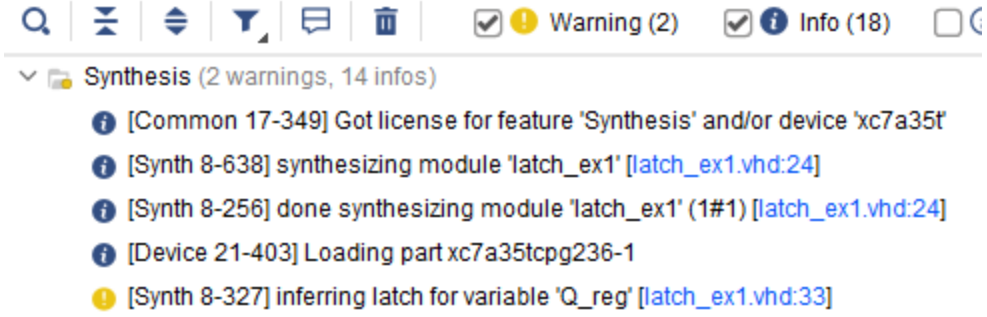
- Exemplo1: inferência de Latch em VHDL com sentenças IF:

```
18  entity latch_ex1 is
19      Port ( clk : in STD_LOGIC;
20             D : in STD_LOGIC;
21             Q : out STD_LOGIC);
22  end latch_ex1;
23
24  architecture Behavioral of latch_ex1 is
25
26
27  begin
28
29      -- processo sequencial inferencia de latch
30      process(clk, D)
31      begin
32          if clk='1' then
33              Q <= D;
34          end if;
35      end process;
36
37  end Behavioral;
```

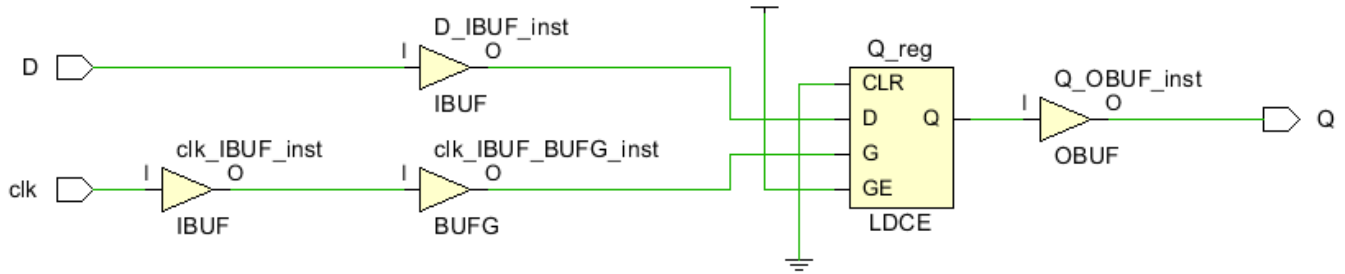
# Inferência de Latches em HDLs

- Exemplo1: inferência de Latch em VHDL. Resultado da síntese lógica.

```
18  entity latch_ex1 is
19      Port ( clk : in STD_LOGIC;
20             D : in STD_LOGIC;
21             Q : out STD_LOGIC);
22  end latch_ex1;
23
24  architecture Behavioral of latch_ex1 is
25
26
27  begin
28
29      -- processo sequencial inferencia de latch
30      process(clk, D)
31      begin
32          if clk='1' then
33              Q <= D;
34          end if;
35      end process;
36
37  end Behavioral;
```




Observe a inferência do lacth tipo D com clear e clock enable (LDCE)



# Inferência de Latches em HDLs

- Corrigindo a descrição para evitar a inferência de Latch:

```
18  entity latch_ex1 is
19      Port ( clk : in STD_LOGIC;
20             D : in STD_LOGIC;
21             Q : out STD_LOGIC);
22  end latch_ex1;
23
24  architecture Behavioral of latch_ex1 is
25
26
27  begin
28
29      -- processo sequencial inferencia de latch
30      process(clk, D)
31      begin
32          if clk='1' then
33              Q <= D;
34          end if;
35      end process;
36
37  end Behavioral;
```

```
14  module latch_ex1_vlog (clk,D,Q);
15      input clk;
16      input D;
17      output reg Q;
18
19      always @(clk,D) begin
20          if (clk) begin
21              Q = D;
22          end
23      end
24
25  endmodule
```

para evitar a inferência de Latch coloque a sentença ELSE e atribua um valor padrão.

# Inferência de Latches em HDLs

- Corrigindo a descrição para evitar a inferência de Latch:

```vhdl
entity latch_ex1_corrigido is
    Port ( clk : in STD_LOGIC;
           D : in STD_LOGIC;
           Q : out STD_LOGIC);
end latch_ex1_corrigido;

architecture Behavioral of latch_ex1_corrigido is


begin

    -- Adicionando setenca ELSE para
    -- evitar a inferencia de latch
    process(clk, D)
    begin
        if clk='1' then
            Q <= D;
        else
            Q <= '0';
        end if;
    end process;

end Behavioral;
```

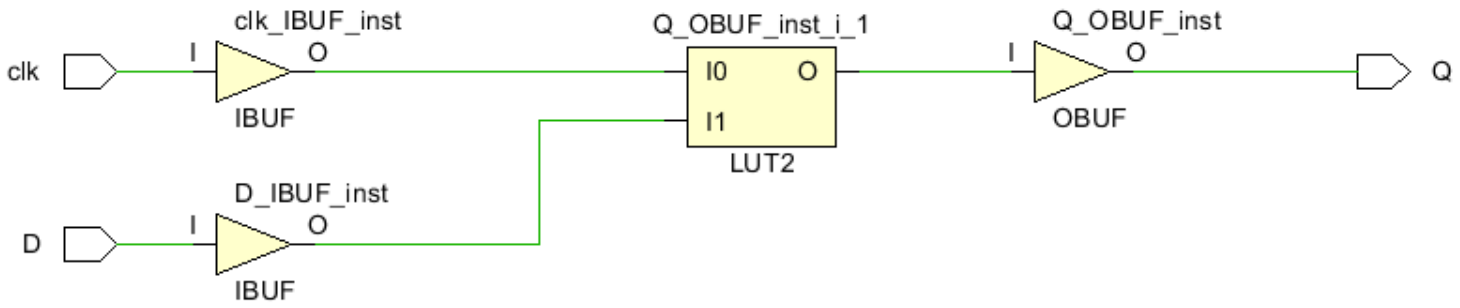
```verilog
module latch_ex1_corrigido_vlog (clk,D,Q);
    input clk;
    input D;
    output reg Q;

    always @(clk,D) begin
        if (clk) begin
            Q = D;
        end else begin
            Q = 1'b0;
        end
    end

endmodule
```

# Inferência de Latches em HDLs

- Corrigindo a inferência de Latch. Resultado da síntese lógica.

```
18  entity latch_ex1_corrigido is
19      Port ( clk : in STD_LOGIC;
20             D : in STD_LOGIC;
21             Q : out STD_LOGIC);
22  end latch_ex1_corrigido;
23
24  architecture Behavioral of latch_ex1_corrigido is
25
26
27  begin
28
29      -- Adicionando setenca ELSE para
30      -- evitar a inferencia de latch
31      process(clk, D)
32      begin
33          if clk='1' then
34              Q <= D;
35          else
36              Q <= '0';
37          end if;
38      end process;
39
40  end Behavioral;
```

Observe a não há inferência de Latch, em lugar foi inferido uma LUT de duas entradas que implementa a expressão O=I0 and I1, ou seja,
Q = clk AND D



# Inferência de Latches em HDLs

- Exemplo2: inferência de Latch em Verilog com sentenças CASE:

```
14  module latch_ex2_vlog (
15      input [1:0] sel,
16      input A,
17      input B,
18      output reg Y
19  );
20
21      always @(sel,A,B) begin
22          case (sel)
23              2'b00: Y = A;
24              2'b01: Y = B;
25              default: ; // Nao faz nada
26          endcase
27      end
28
29  endmodule
```

# Inferência de Latches em HDLs

- Exemplo2: inferência de Latch em Verilog com sentenças CASE:

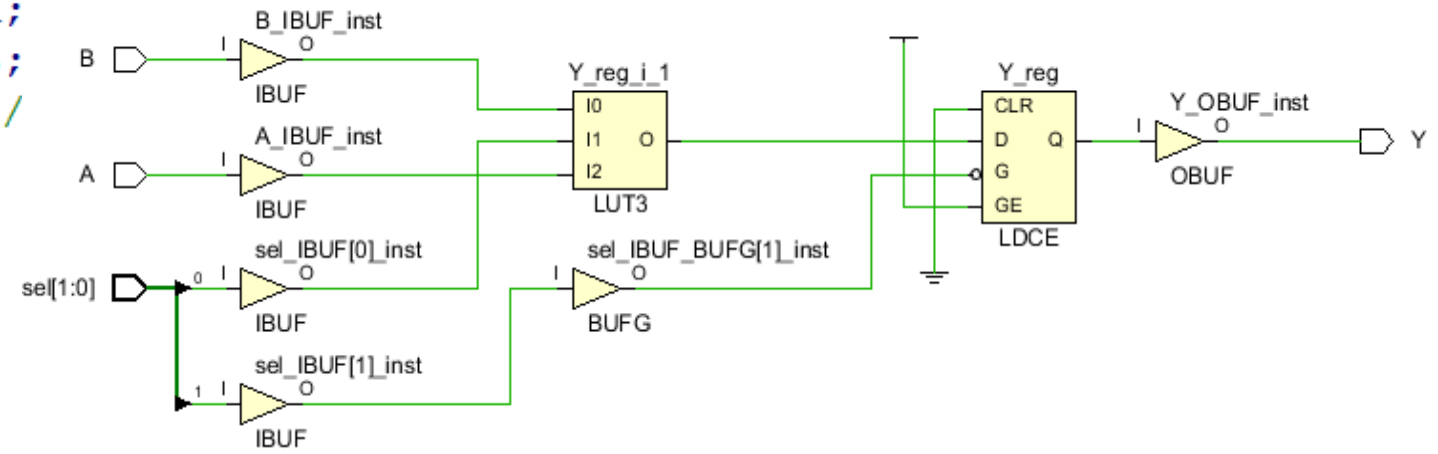
```
14  module latch_ex2_vlog (
15      input [1:0] sel,
16      input A,
17      input B,
18      output reg Y
19  );
20
21      always @(sel,A,B) begin
22          case (sel)
23              2'b00: Y = A;
24              2'b01: Y = B;
25              default: ; //
26          endcase
27      end
28
29  endmodule
```

Synthesis (2 warnings)

[Synth 8-327] inferring latch for variable 'Y_reg' [latch_ex2_vlog.v:23]

[Constraints 18-5210] No constraints selected for write.
Resolution: This message can indicate that there are no constraints for the
used when running synth_design to not write synthesis constraints to the

Observe a inferência do lacth tipo D com clear e clock enable (LDCE)



# Inferência de Latches em HDLs

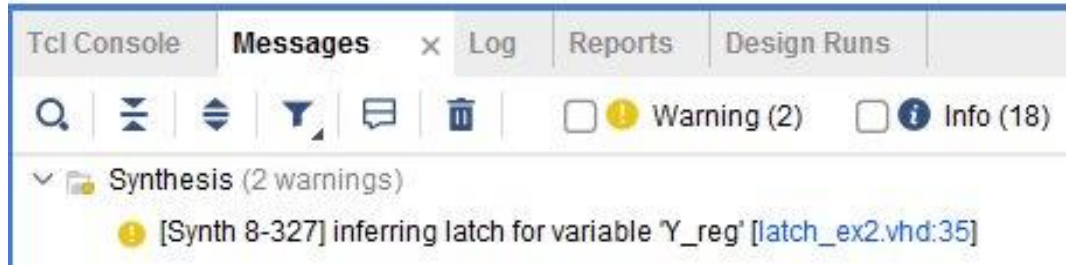
- Exemplo2: inferência de Latch em VHDL com sentenças CASE:

```
18  entity latch_ex2 is
19      Port ( sel : in STD_LOGIC_VECTOR(1 downto 0);
20             A : in STD_LOGIC;
21             B : in STD_LOGIC;
22             Y : out STD_LOGIC);
23  end latch_ex2;
24
25  architecture Behavioral of latch_ex2 is
26
27
28  begin
29
30      -- processo sequencial inferencia de latch
31      -- sentenca case-when usa when others
32      -- mas nao faz nada nele.
33      process(sel, A, B)
34      begin
35          case sel is
36              when "00" => Y <= A;
37              when "01" => Y <= B;
38              when others => null;
39          end case;
40      end process;
41
42  end Behavioral;
```

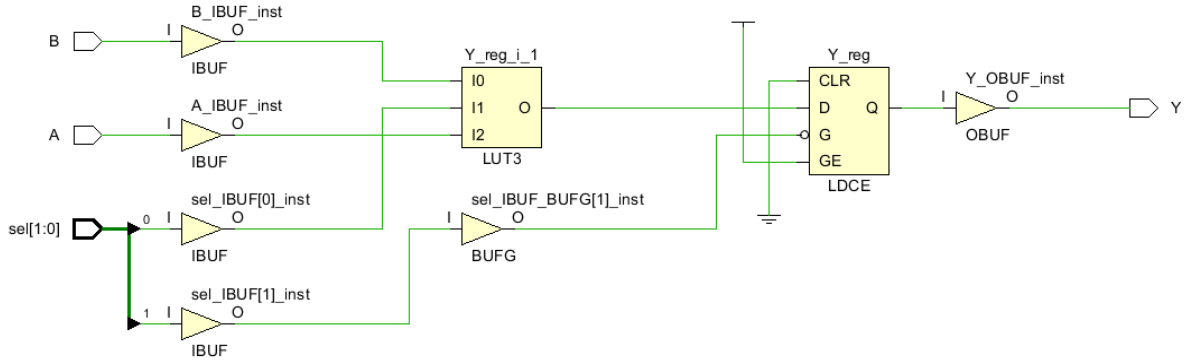
# Inferência de Latches em HDLs

- Exemplo2: inferência de Latch em VHDL. Resultado da síntese lógica

```
18  entity latch_ex2 is
19      Port ( sel : in STD_LOGIC_VECTOR(1 downto 0);
20             A : in STD_LOGIC;
21             B : in STD_LOGIC;
22             Y : out STD_LOGIC);
23  end latch_ex2;
24
25  architecture Behavioral of latch_ex2 is
26
27
28  begin
29
30      -- processo sequencial inferencia de latch
31      -- sentenca case-when usa when others
32      -- mas nao faz nada nele.
33      process(sel, A, B)
34      begin
35          case sel is
36              when "00" => Y <= A;
37              when "01" => Y <= B;
38              when others => null;
39          end case;
40      end process;
41
42  end Behavioral;
```




Observe a inferência do lacth tipo D com clear e clock enable (LDCE)



# Inferência de Latches em HDLs

- Corrigindo a descrição para evitar a inferência de Latch:

```
18  entity latch_ex2 is
19      Port ( sel : in STD_LOGIC_VECTOR(1 downto 0);
20             A : in STD_LOGIC;
21             B : in STD_LOGIC;
22             Y : out STD_LOGIC);
23  end latch_ex2;
24
25  architecture Behavioral of latch_ex2 is
26
27
28  begin
29
30      -- processo sequencial inferencia de latch
31      -- sentenca case-when usa when others
32      -- mas nao faz nada nele.
33      process(sel, A, B)
34      begin
35          case sel is
36              when "00" => Y <= A;
37              when "01" => Y <= B;
38              when others => null;
39          end case;
40      end process;
41
42  end Behavioral;
```

```
14  module latch_ex2_vlog (
15      input [1:0] sel,
16      input A,
17      input B,
18      output reg Y
19  );
20
21      always @(sel,A,B) begin
22          case (sel)
23              2'b00: Y = A;
24              2'b01: Y = B;
25              default: ; // Nao faz nada
26          endcase
27      end
28
29  endmodule
```

para evitar a inferência de Latch atribua um valor padrão na sentença when others.

# Inferência de Latches em HDLs

- Corrigindo a descrição para evitar a inferência de Latch:

```vhdl
entity latch_ex2_corrigido is
    Port ( sel : in STD_LOGIC_VECTOR(1 downto 0);
           A : in STD_LOGIC;
           B : in STD_LOGIC;
           Y : out STD_LOGIC);
end latch_ex2_corrigido;

architecture Behavioral of latch_ex2_corrigido is


begin

    -- Adicionando valor padrao da saida no
    -- when others para evitar inferencia de latch
    process(sel, A, B)
    begin
        case sel is
            when "00" => Y <= A;
            when "01" => Y <= B;
            when others => Y <= '0';
        end case;
    end process;

end Behavioral;
```

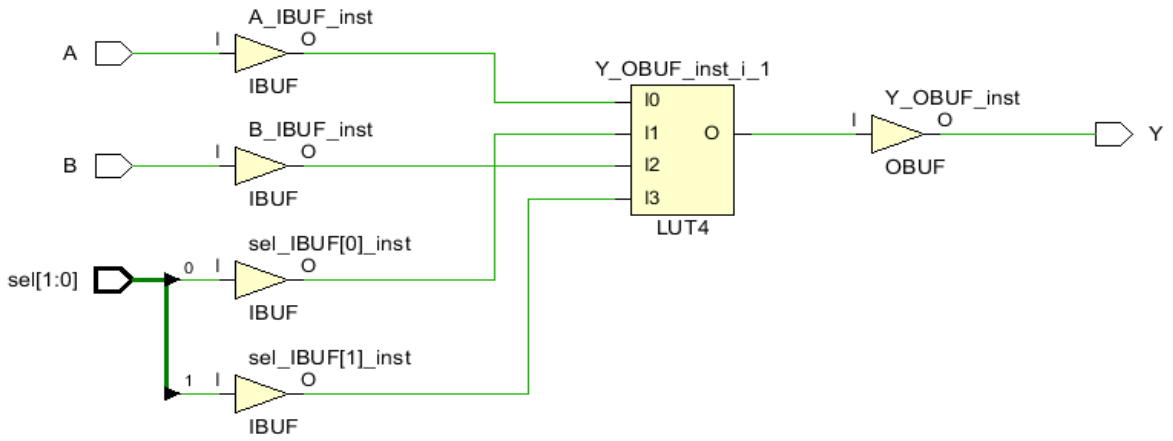
```verilog
module latch_ex2_corrigido_vlog (
    input [1:0] sel,
    input A,
    input B,
    output reg Y
);
    // adicionando valor padrao
    // na saida para evitar latch
    always @(sel,A,B) begin
        case (sel)
            2'b00: Y = A;
            2'b01: Y = B;
            default: Y = 1'b0;
        endcase
    end

endmodule
```

# Inferência de Latches em HDLs

- Corrigindo a inferência de Latch. Resultado da síntese lógica.

```
18  entity latch_ex2_corrigido is
19      Port ( sel : in STD_LOGIC_VECTOR(1 downto 0);
20             A : in STD_LOGIC;
21             B : in STD_LOGIC;
22             Y : out STD_LOGIC);
23  end latch_ex2_corrigido;
24
25  architecture Behavioral of latch_ex2_corrigido is
26
27
28  begin
29
30      -- Adicionando valor padrao da saida no
31      -- when others para evitar inferencia de latch
32      process(sel, A, B)
33      begin
34          case sel is
35              when "00" => Y <= A;
36              when "01" => Y <= B;
37              when others => Y <= '0';
38          end case;
39      end process;
40
41  end Behavioral;
```

Observe que em lugar do Latch foi inferida uma LUT que deve implementar um multiplexador de duas entradas e uma saída (Mux 2to1).



# Inferência de Latches em HDLs

- O projeto de circuitos sequenciais assíncronos deve atender requisitos adicionais se comparado com a lógica sequencial síncrona. Por exemplo, evitar condições de corrida, evitar corridas críticas, evitar hazards (azares), os quais produzem glitches (variações momentâneas nas saídas), entre outros.
- Se o projetista não tomar os devidos cuidados, a ferramenta de síntese pode inferir Latches para implementar lógica sequencial assíncrona. Vejamos um exemplo em VHDL:

**Observação:** O projeto de circuitos sequenciais assíncronos está fora do escopo deste curso. Sempre evite a inferência de Latches!

# Inferência de Latches em HDLs

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24  use IEEE.STD_LOGIC_UNSIGNED.ALL;
25  use IEEE.NUMERIC_STD.ALL;
26
27  entity cnt_latch is
28      Port ( reset : in STD_LOGIC;
29             clk : in STD_LOGIC;
30             load : in STD_LOGIC;
31             preset : in STD_LOGIC_VECTOR (15 downto 0);
32             led : out STD_LOGIC_VECTOR (15 downto 0));
33  end cnt_latch;
34
35  architecture Behavioral of cnt_latch is
36
37  signal cnt : unsigned(15 downto 0) := (others=>'0');
```

```
37  signal cnt : unsigned(15 downto 0) := (others=>'0');
38
39  begin
40
41      -- processo sequencial inferencia de latch
42      process(reset,clk,preset)
43      begin
44          if reset='1' then
45              cnt <= (others=>'0');
46          elsif clk='1' then
47              if load='1' then
48                  cnt <= unsigned(preset);
49              elsif cnt = "0000000000000000" then
50                  cnt <= unsigned(preset);
51              else
52                  cnt <= cnt -1;
53              end if;
54          end if;
55      end process;
56
57      led <= std_logic_vector(cnt);
58
59  end Behavioral;
```

O que faz este circuito?

Qual o problema deste circuito?

Como corrigir o(s) problema(s)?

# Inferência de Latches em HDLs

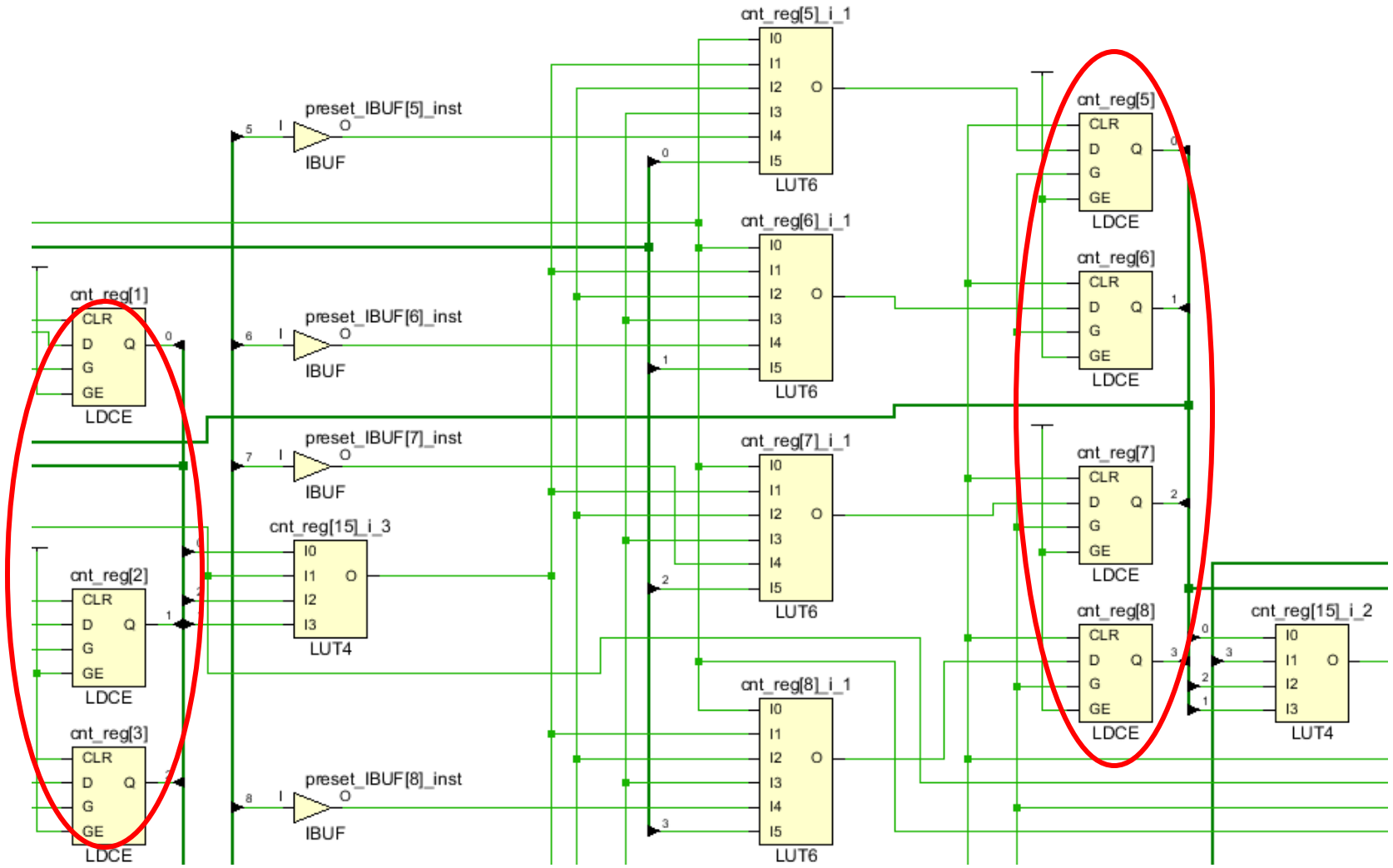
```
37  signal cnt : unsigned(15 downto 0) := (others=>'0');
38
39  begin
40
41      -- processo sequencial inferencia de latch
42      process(reset,clk,preset)
43      begin
44          if reset='1' then
45              cnt <= (others=>'0');
46          elsif clk='1' then
47              if load='1' then
48                  cnt <= unsigned(preset);
49              elsif cnt = "0000000000000000" then
50                  cnt <= unsigned(preset);
51              else
52                  cnt <= cnt -1;
53              end if;
54          end if;
55      end process;
56
57      led <= std_logic_vector(cnt);
58
59  end Behavioral;
```

O circuito implementa um contador descrescente (aulas futuras vão detalhar este tipo de circuito).

O clk é usado por nível, portanto haverá inferência de latches.

O registrador cnt de 16 bits vai ser implementado usando 16 Latches.

# Inferência de Latches em HDLs




Resultado de síntese lógica:

Observe a inferência de Latches para implementar o registrador cnt.

LUTs implementam equações booleanas para soma e comparação.

# Exercícios Aula 3 – Latches em HDLs

Exercício 1: Quais as vantagens dos Latches? Quais são as suas desvantages?

Exercício 2: Realize uma pesquisa e descreva em quais situações (ou aplicações) podem/devem ser usados Latches.

Exercício 3: Construa em Verilog ou VHDL um multiplexador de 4 entradas e uma saída com largura de bits parametrizável. Use sentenças IF – ELSIF para produzir a presença de Latches. Realize a síntese lógica e analise o esquemático RTL. A ferramenta inferiu Latches? Quantos Latches foram inferidos? Realize um testbench e verifique o funcionamento através de simulação.

Exercício 4: Repita o exercício anterior usando diretivas CASE.

Página em branco.

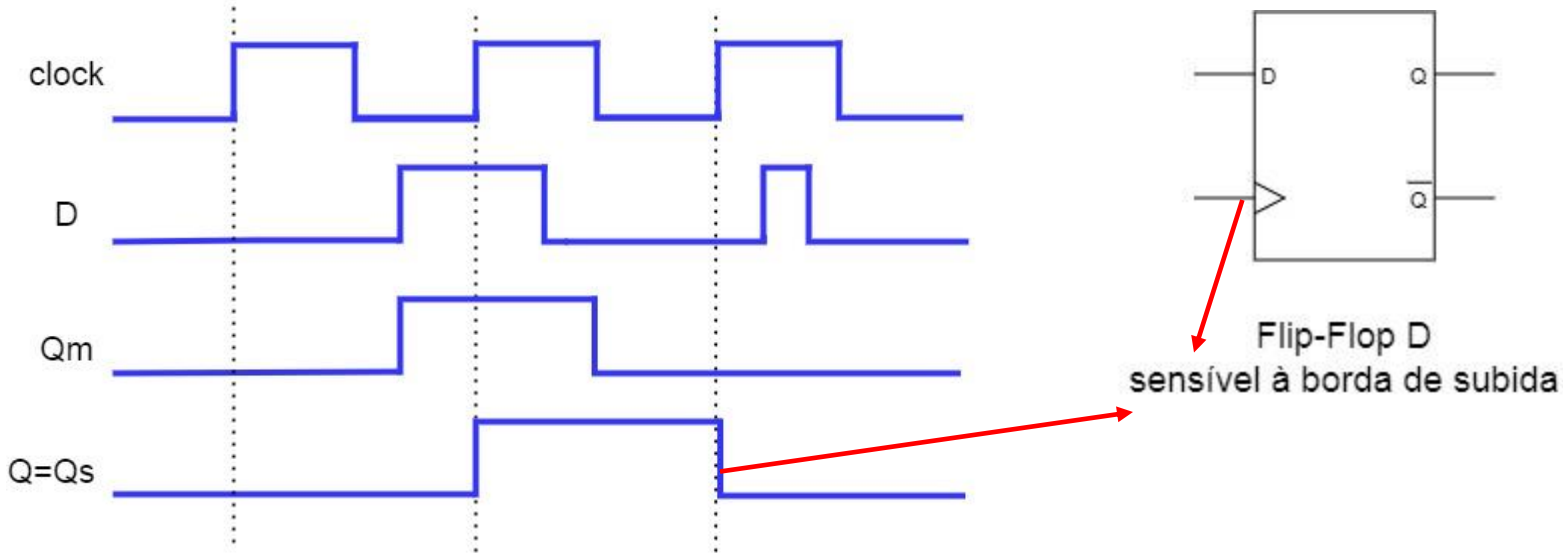
# Aula 4
# Implementação de Flip-flops em VHDL

# Revisão sobre Flip-Flops

- Flip-flops são sensíveis à borda de clock. São circuitos mais estáveis do que os Latches. A borda de subida é chamada de borda positiva e a borda de descida é chamada de borda negativa.
- Quando o clock não apresenta uma borda, as variações na entrada não se manifestam na saída.
- O dado de entrada de cada Flip-flop deve atender o tempo de setup (tempo que o dado deve estar estável antes da borda do clock) e o tempo de hold (tempo que o dado deve estar estável após a borda de clock).
- Alguns Flip-flops tem entradas de Reset (clear) ou de Set as quais podem ser usadas de forma assíncrona.

# Revisão de Flip-Flops

- Flip-flops são sensíveis à borda de clock. Quando o clock não apresenta uma borda, as variações na entrada não se manifestam na saída.



# Inferência de Flip-Flops em HDLs

- Flip-flops são inferidos se sentenças IF – ELSIF – ELSE ou sentenças CASE – WHEN são condicionadas à borda de clock.

# Inferência de Flip-Flops

- Flip-flops são inferidos se sentenças IF – ELSIF – ELSE ou sentenças CASE – WHEN são condicionadas à borda de clock.

- Em Verilog a condição de borda positiva (subida) de clock pode ser declarada com a sentença

  always @(posedge clk)

- Em Verilog a condição de borda negativa (descida) de clock pode ser declarada com a sentença

  always @(negedge clk)

# Inferência de Flip-Flops

- Flip-flops são inferidos se sentenças IF – ELSIF – ELSE ou sentenças CASE – WHEN são condicionadas à borda de clock.

- Em VHDL a condição de borda positiva (subida) de clock pode ser declarada com as sentenças

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

- Em VHDL a condição de borda negativa (descida) de clock pode ser declarada com as sentenças

  falling_edge(clk) ou clk'event and clk='0'

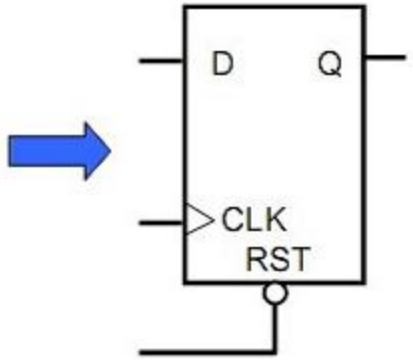
# Inferência de Flip-Flops

- Flip-flop sensível a borda positiva com clear (reset) assíncrono.

reset assíncrono

```
architecture FLOP of DFCLR is
begin
  INFER: process (CLK, RST)
  begin
    if (RST ='0') then
        Q <=  '0';
    elsif (CLK'event and CLK ='1') then
        Q  <= D;
   end if ;
  end process INFER;
end FLOP;
```

D flip-flop with asynchronous low reset and active high clock edge



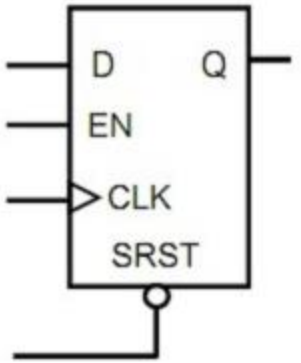
# Inferência de Flip-Flops

- Flip-flop sensível a borda positiva com enable e clear (reset) síncrono em lógica invertida (srst negado também chamado de srstn).

D flip-flop with synchronous low reset, active high enable and rising edge clock

reset síncrono

```
architecture FLOP of DFSLRHE is
begin
  INFER: process (CLK)
  begin
    if (CLK'event and CLK ='1') then
      if (SRST = '0') then
        Q <= '0';
      elsif (EN = '1') then
        Q  <= D;
      end if ;
    end if ;
  end process INFER;
end FLOP;
```



# Inferência de Flip-Flops

- Em VHDL outra forma de indicar uma borda de clock é com as diretivas rising_edge() e falling_edge(). Por exemplo:

```
8  architecture comportamental of flip-flop is
9  begin
10
11     process(clk)
12     begin
13         if rising_edge(clk) then
14             if reset='1' then
15                 q <= '0';
16             else
17                 q <= d;
18             end if;
19         end if
20     end process;
21
22 end comportamental;
```

Flip-flop sensível a borda de subida com reset síncrono.

# Inferência de Flip-Flops

- Exemplo 1: Flip-flop tipo D sensível à borda de subida com reset assíncrono (clear).

# Inferência de Flip-Flops

- Exemplo 1 em Verilog: Flip-flop tipo D sensível à borda de subida com reset assíncrono (clear).

```
14  module FFD_ex1_vlog (
15      input logic reset,
16      input logic clk,
17      input logic D,
18      output logic Q
19  );
20
21      always @(posedge clk or posedge reset) begin
22          if (reset) begin
23              Q <= 1'b0;
24          end else begin
25              Q <= D;
26          end
27      end
28
29  endmodule
```

# Inferência de Flip-Flops

- Exemplo 1 em VHDL: Flip-flop tipo D sensível à borda de subida com reset assíncrono (clear).

```
15  library IEEE;
16  use IEEE.STD_LOGIC_1164.ALL;
17
18  entity FFD_ex1 is
19      Port ( reset : in STD_LOGIC;
20             clk : in STD_LOGIC;
21             D : in STD_LOGIC;
22             Q : out STD_LOGIC);
23  end FFD_ex1;
24
25  architecture Behavioral of FFD_ex1 is
26
```

```
25  architecture Behavioral of FFD_ex1 is
26
27
28  begin
29
30      -- processo sequencial inferencia de FF
31      process(reset, clk, D)
32      begin
33          if reset='1' then
34              Q <= '0';
35          elsif rising_edge(clk) then
36              Q <= D;
37          end if;
38      end process;
39
40  end Behavioral;
```

Vejamos o resultado de simulação e síntese lógica:

# Inferência de Flip-Flops

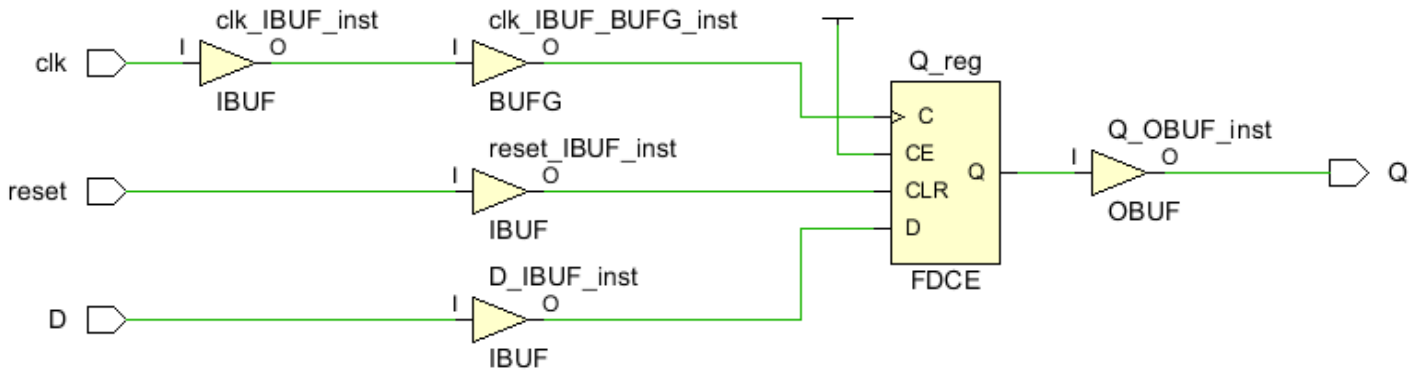
- Exemplo 1: Flip-flop tipo D sensível à borda de subida com reset assíncrono (clear). Simulação:

# Inferência de Flip-Flops

- Exemplo 1: Flip-flop tipo D sensível à borda de subida com reset assíncrono (clear). Síntese lógica:

```
25  architecture Behavioral of FFD_ex1 is
26
27
28  begin
29
30      -- processo sequencial inferencia de FF
31      process(reset, clk, D)
32      begin
33          if reset='1' then
34              Q <= '0';
35          elsif rising_edge(clk) then
36              Q <= D;
37          end if;
38      end process;
39
40  end Behavioral;
```

Resultado de síntese lógica. Observe a inferência de um FDCE, ou seja Flip-flop tipo D com clear e enable.



# Inferência de Flip-Flops

- Exemplo 2 em Verilog: Flip-flop tipo D sensível à borda de subida com preset assíncrono.

# Inferência de Flip-Flops

- Exemplo 2: Flip-flop tipo D sensível à borda de subida com preset assíncrono em Verilog.

```
14  module FFD_ex2_vlog (
15      input preset,
16      input clk,
17      input D,
18      output reg Q
19  );
20
21      always @(posedge clk or posedge preset) begin
22          if (preset) begin
23              Q <= 1'b1;
24          end else begin
25              Q <= D;
26          end
27      end
28
29  endmodule
```

# Inferência de Flip-Flops

- Exemplo 2 em VHDL  Flip-flop tipo D sensível à borda de subida com preset assíncrono.

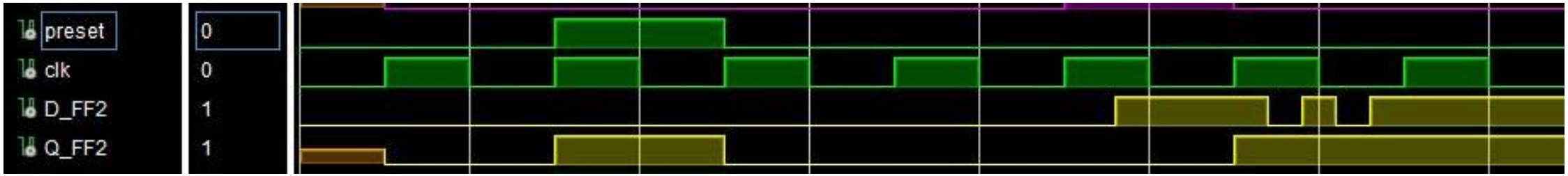
```
15  library IEEE;
16  use IEEE.STD_LOGIC_1164.ALL;
17
18  entity FFD_ex2 is
19      Port ( preset : in STD_LOGIC;
20             clk : in STD_LOGIC;
21             D : in STD_LOGIC;
22             Q : out STD_LOGIC);
23  end FFD_ex2;
24
25  architecture Behavioral of FFD_ex2 is
```

```
25  architecture Behavioral of FFD_ex2 is
26
27
28  begin
29
30      -- processo sequencial inferencia de FF
31      process(preset,clk,D)
32      begin
33          if preset='1' then
34              Q <= '1';
35          elsif rising_edge(clk) then
36              Q <= D;
37          end if;
38      end process;
39
40  end Behavioral;
41
```

Vejamos o resultado de simulação e síntese lógica:

# Inferência de Flip-Flops

- Exemplo 2: Flip-flop tipo D sensível à borda de subida com preset assíncrono. Simulação:

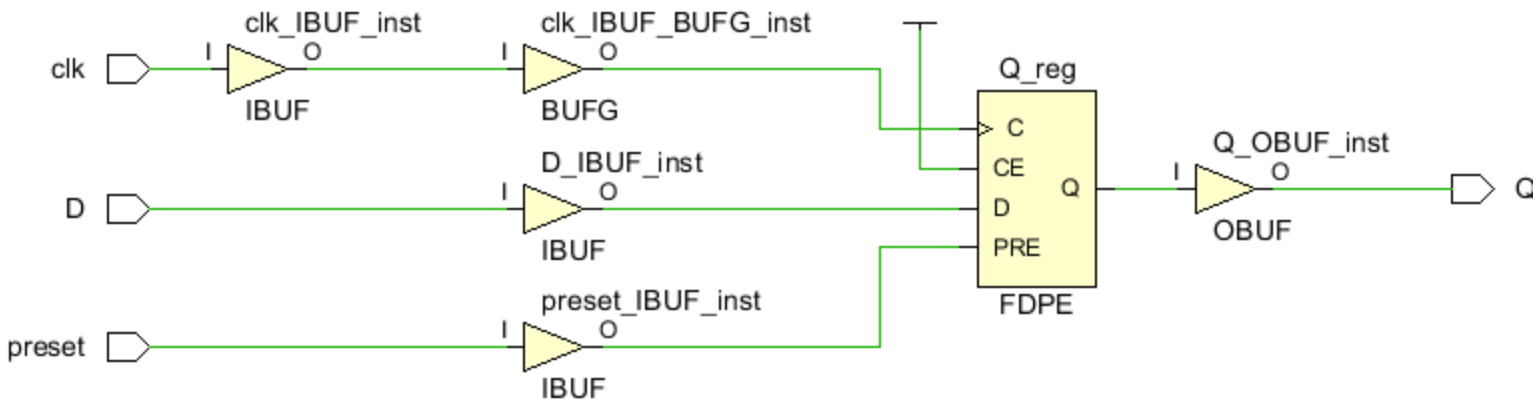


# Inferência de Flip-Flops

- Exemplo 2: Flip-flop tipo D sensível à borda de subida com preset assíncrono. Síntese lógica:

```
28  begin
29
30      -- processo sequencial inferencia de FF
31      process(preset,clk,D)
32      begin
33          if preset='1' then
34              Q <= '1';
35          elsif rising_edge(clk) then
36              Q <= D;
37          end if;
38      end process;
39
40  end Behavioral;
41
```

Resultado de síntese lógica. Observe a inferência de um FDPE, ou seja Flip-flop tipo D com preset e enable.



# Inferência de Flip-Flops

- Exemplo 3 em Verilog: Flip-flop tipo D sensível à borda de descida com enable e reset assíncrono.

# Inferência de Flip-Flops

- Exemplo 3 em Verilog: Flip-flop tipo D sensível à borda de descida com enable e reset assíncrono.

```
14  module FFD_ex3_vlog (
15      input reset,
16      input clk,
17      input load,
18      input D,
19      output reg Q
20  );
21
22      always @(negedge clk or posedge reset) begin
23          if (reset) begin
24              Q <= 1'b0;
25          end else if (load) begin
26              Q <= D;
27          end
28      end
29
30  endmodule
```

# Inferência de Flip-Flops

- Exemplo 3 em VHDL: Flip-flop tipo D sensível à borda de descida com enable e reset assíncrono.

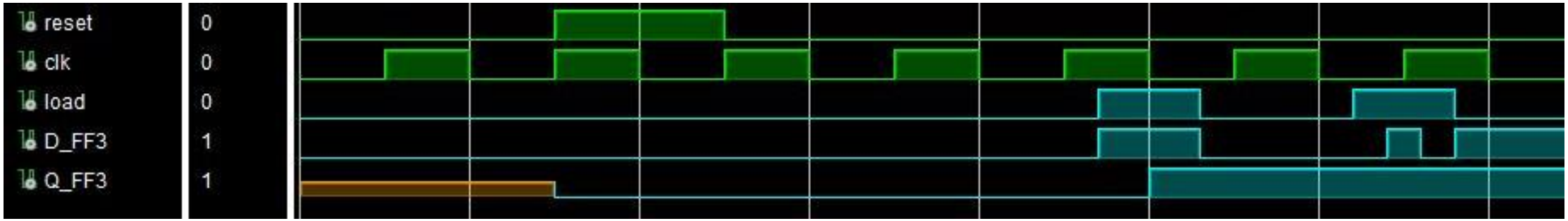
```
15  library IEEE;
16  use IEEE.STD_LOGIC_1164.ALL;
17
18  entity FFD_ex3 is
19      Port ( reset : in STD_LOGIC;
20             clk : in STD_LOGIC;
21             load : in STD_LOGIC;
22             D : in STD_LOGIC;
23             Q : out STD_LOGIC);
24  end FFD_ex3;
25
26  architecture Behavioral of FFD_ex3 is
```

```
26  architecture Behavioral of FFD_ex3 is
27
28  begin
29
30      -- processo sequencial inferencia de FF
31      process(reset,clk,load,D)
32      begin
33          if reset='1' then
34              Q <= '1';
35          elsif falling_edge(clk) then
36              if load='1' then
37                  Q <= D;
38              end if;
39          end if;
40      end process;
41
42  end Behavioral;
```

Vejamos o resultado de simulação e síntese lógica:

# Inferência de Flip-Flops

- Exemplo 3: Flip-flop tipo D sensível à borda de descida com enable e reset assíncrono. Simulação:

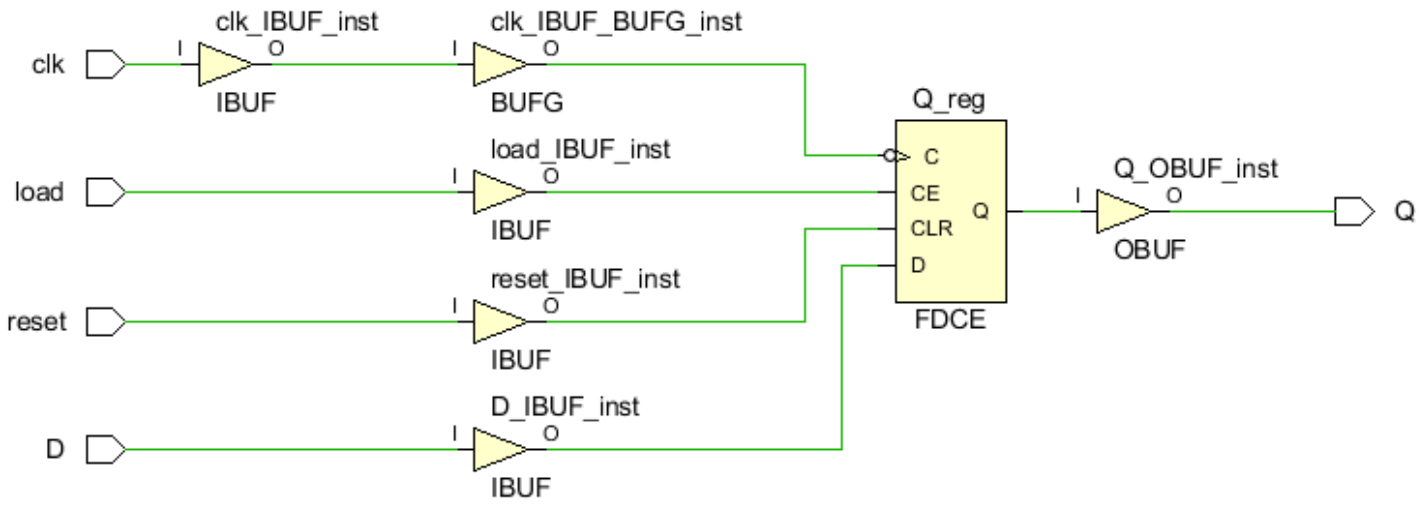


# Inferência de Flip-Flops

- Exemplo 3: Flip-flop tipo D sensível à borda de descida com enable e reset assíncrono. Síntese lógica:

```
30      -- processo sequencial inferencia
31      process(reset,clk,load,D)
32      begin
33          if reset='1' then
34              Q <= '1';
35          elsif falling_edge(clk) then
36              if load='1' then
37                  Q <= D;
38              end if;
39          end if;
40      end process;
```

Resultado de síntese lógica. Observe a inferência de um FDCE, ou seja Flip-flop tipo D com reset e clock enable (CE). Observe a borda negativa do clk.



# Exercícios Aula 4 – Flip-Flops em HDLs

- Exercício 1: implemente em Verilog e em VHDL um registrador de 8 bits sensível à borda de subida do clock com reset assíncrono em lógica invertida. Realize uma simulação comportamental e sintetize o circuito. Analise o esquemático RTL.
- Exercício 2: implemente em Verilog e em VHDL um registrador de 16 bits sensível à borda de descida do clock com reset síncrono. Realize uma simulação comportamental e sintetize o circuito. Analise o esquemático RTL.
- Exercício 3: implemente em Verilog e em VHDL um registrador de 4 bits com clear e preset assíncronos, sensível à borda de subida do clock. Na condição de preset o valor "1010" deve ser carregado no registrador. Realize uma simulação comportamental e sintetize o circuito. Analise o esquemático RTL.

Página em branco.