



IC-UNICAMP

# MC 613

IC/Unicamp

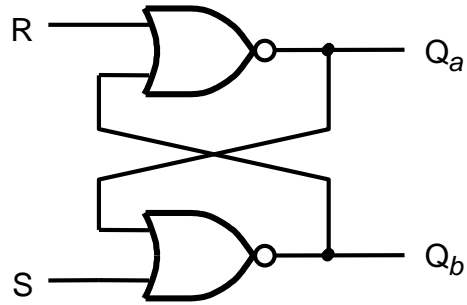
Prof Guido Araújo

Prof Mario Côrtes

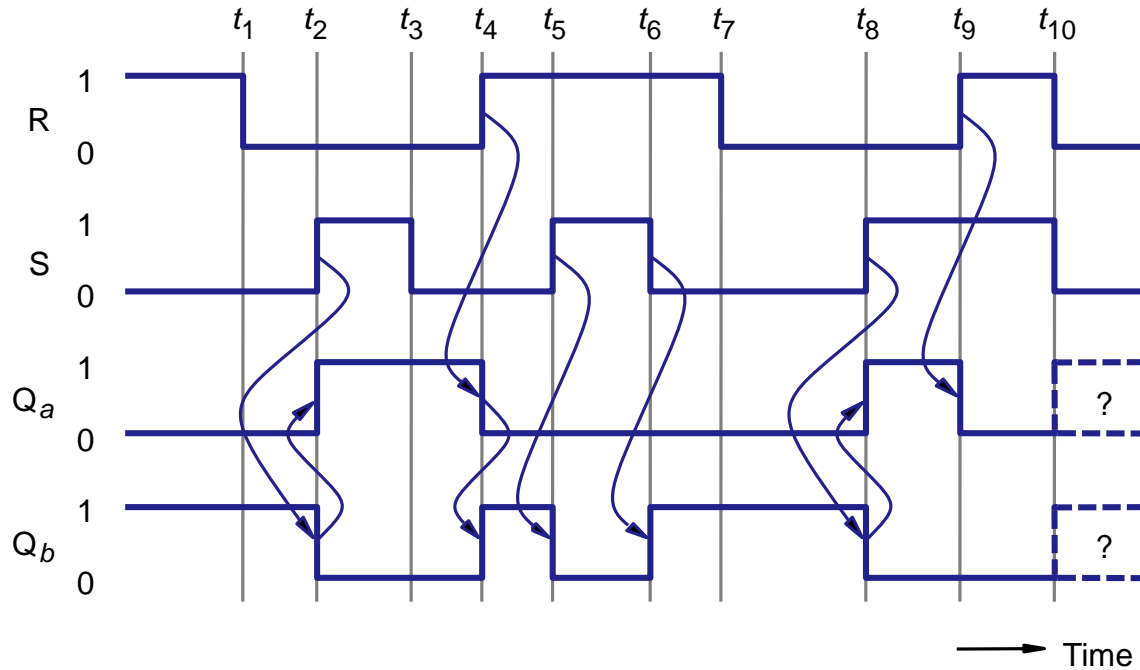
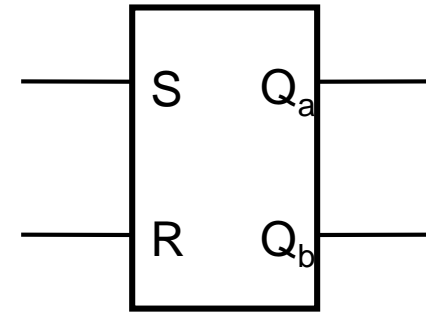
## Elementos de armazenamento: Latches e Flip-flops

- Latches
  - SR e SR chaveado
  - Tipo D
- Flip-flops
  - Mestre-Escravo
  - Tipo D
  - Tipo JK
  - Tipo T
- Comportamento transparente e sensível à borda
- Preset e Clear síncronos e assíncronos

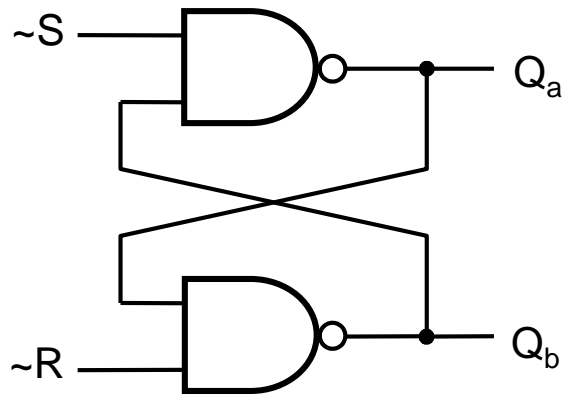
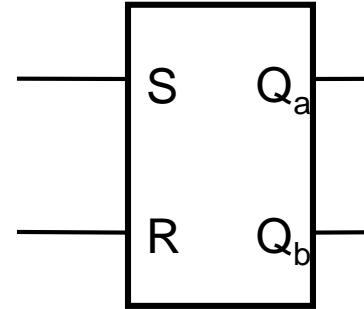
# Latch SR com NORs



S	R	Q <sub>a</sub>	Q <sub>b</sub>	
0	0	0/1	1/0	(no change)
0	1	0	1	
1	0	1	0	
1	1	0	0	

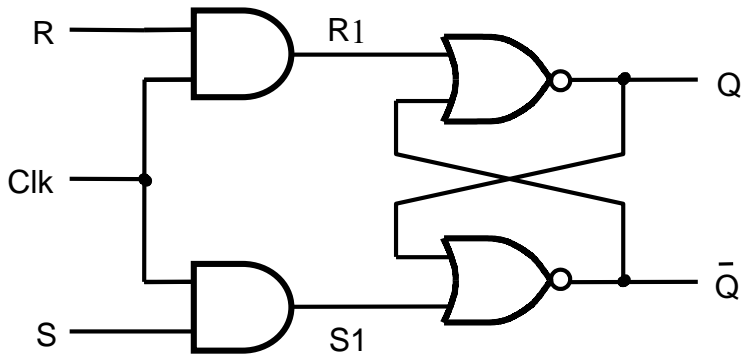


# Latch SR com NANDs



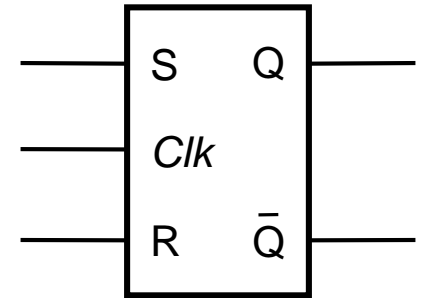
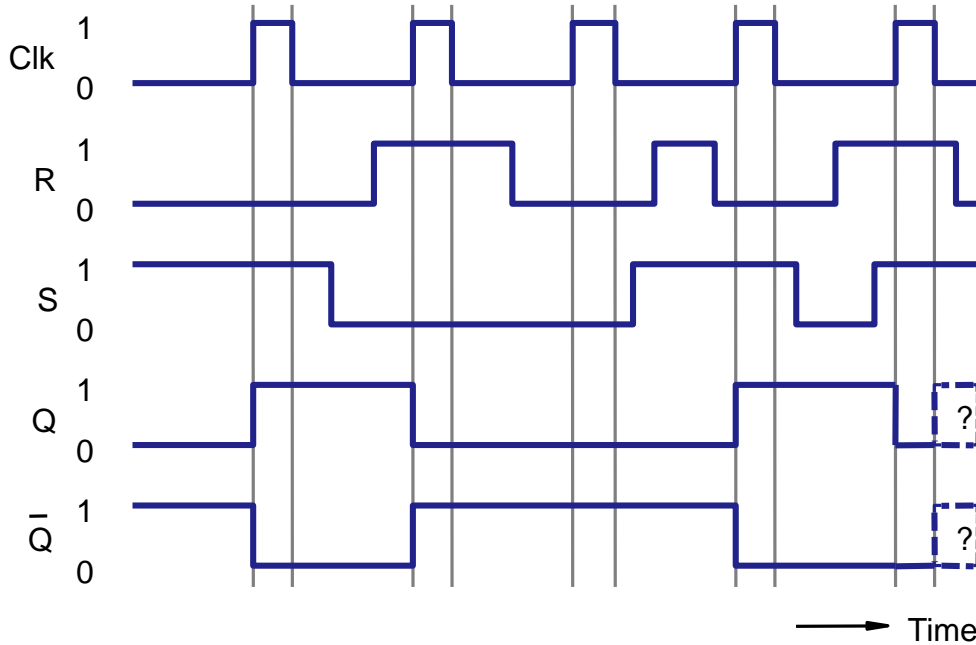
$\sim S$	$\sim R$	$Q_a$	$Q_b$	
1	1	0/1	1/0	(no change)
0	1	1	0	
1	0	0	1	
0	0	1	1	

# Latch SR chaveado

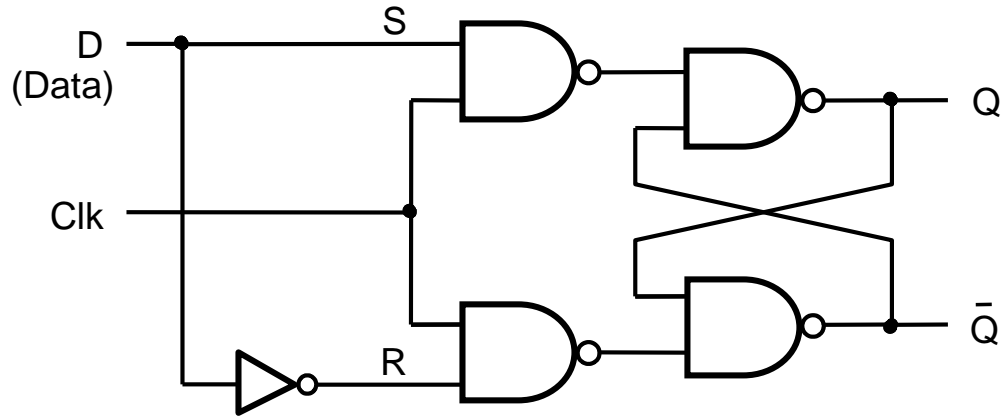


Clk	S	R	$Q(t+1)$
0	x	x	$Q(t)$ (no change)
1	0	0	$Q(t)$ (no change)
1	0	1	0
1	1	0	1
1	1	1	x

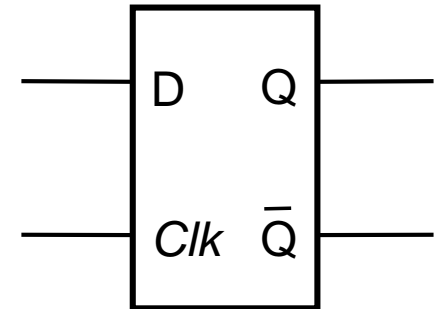
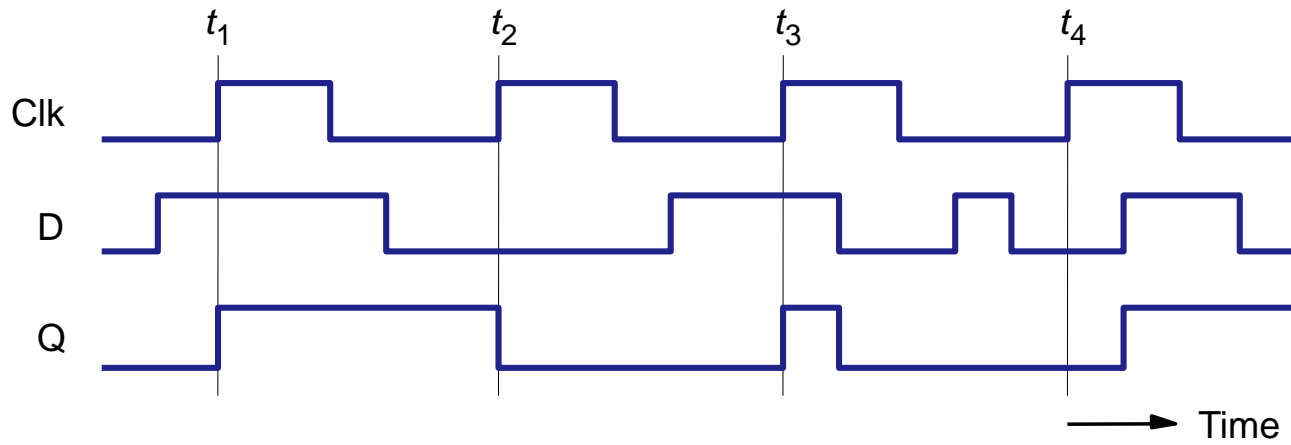
Por que?



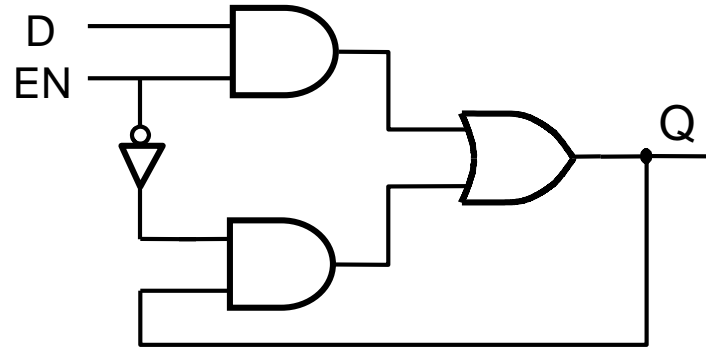
# Latch tipo D chaveado



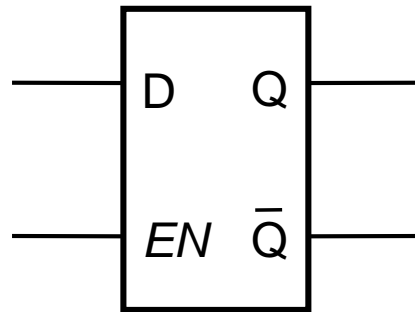
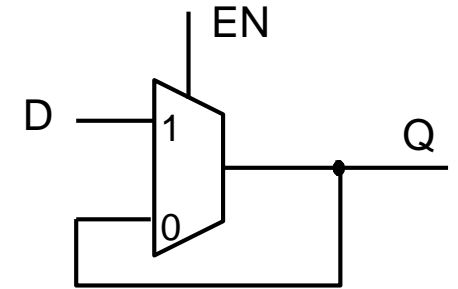
Clk	D	$Q(t+1)$
0	x	$Q(t)$
1	0	0
1	1	1



# Latch tipo D (alternativa)



Equivalente a



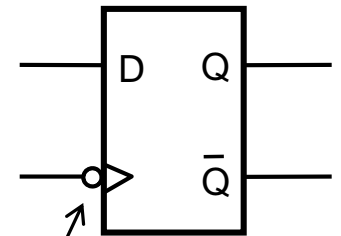
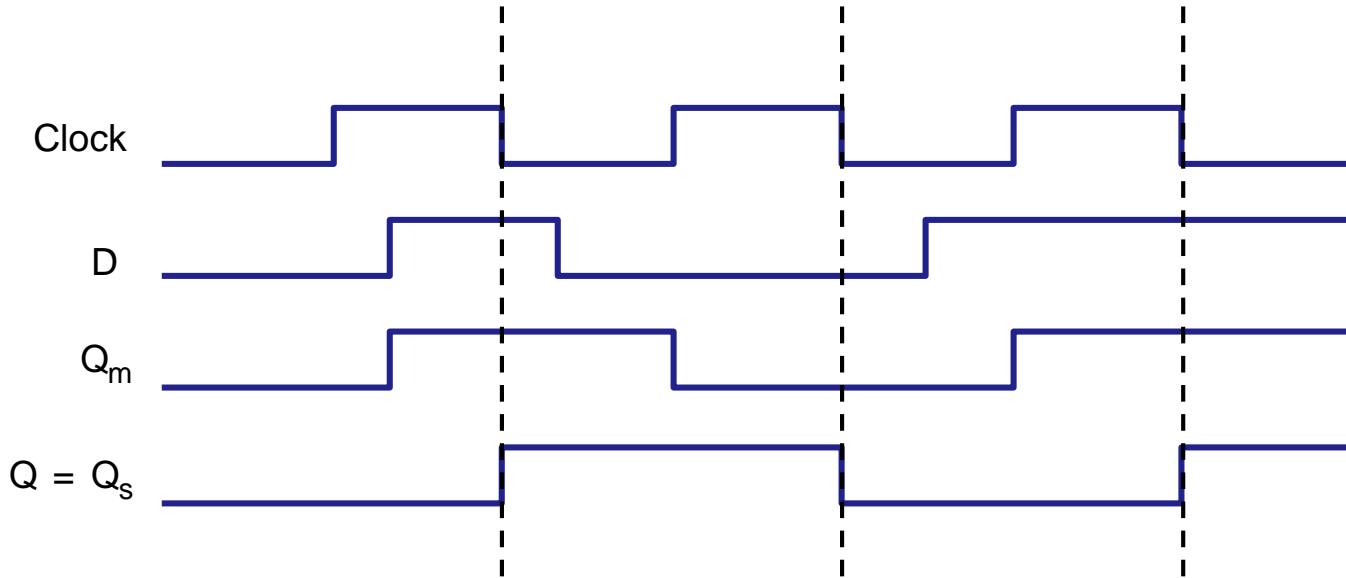
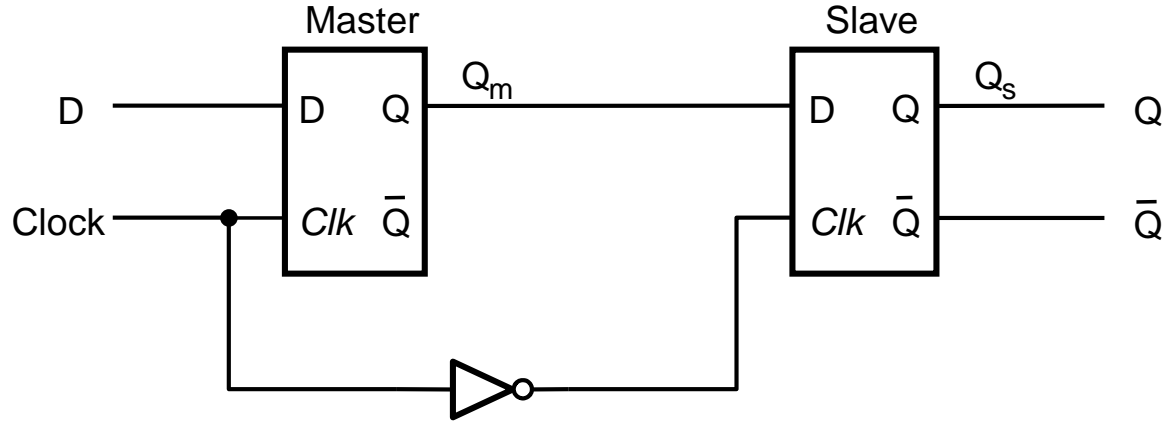
# Latches e Flip-Flops: diferenças



- Manifestação da saída Q em função de variações na entrada D:
  - Latch: transparente durante EN (ou Ck) ativos, ou seja, entrada D passa diretamente para a saída Q
  - Flip-Flop: na borda do Clock, o valor presente na entrada D é transferido para Q
- Instante em que o valor da entrada D é armazenado
  - Latch: valor armazenado é o presente na entrada D no instante em que EN (ou Ck) é desativado (operação de latch ou travamento)
  - Flip-Flop: na borda do Clock, o valor presente na entrada D é armazenado

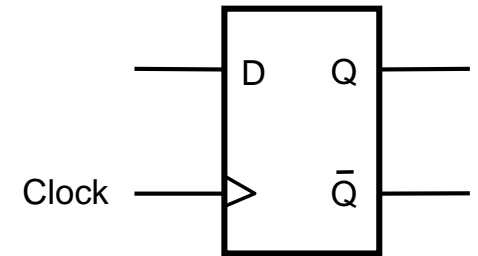
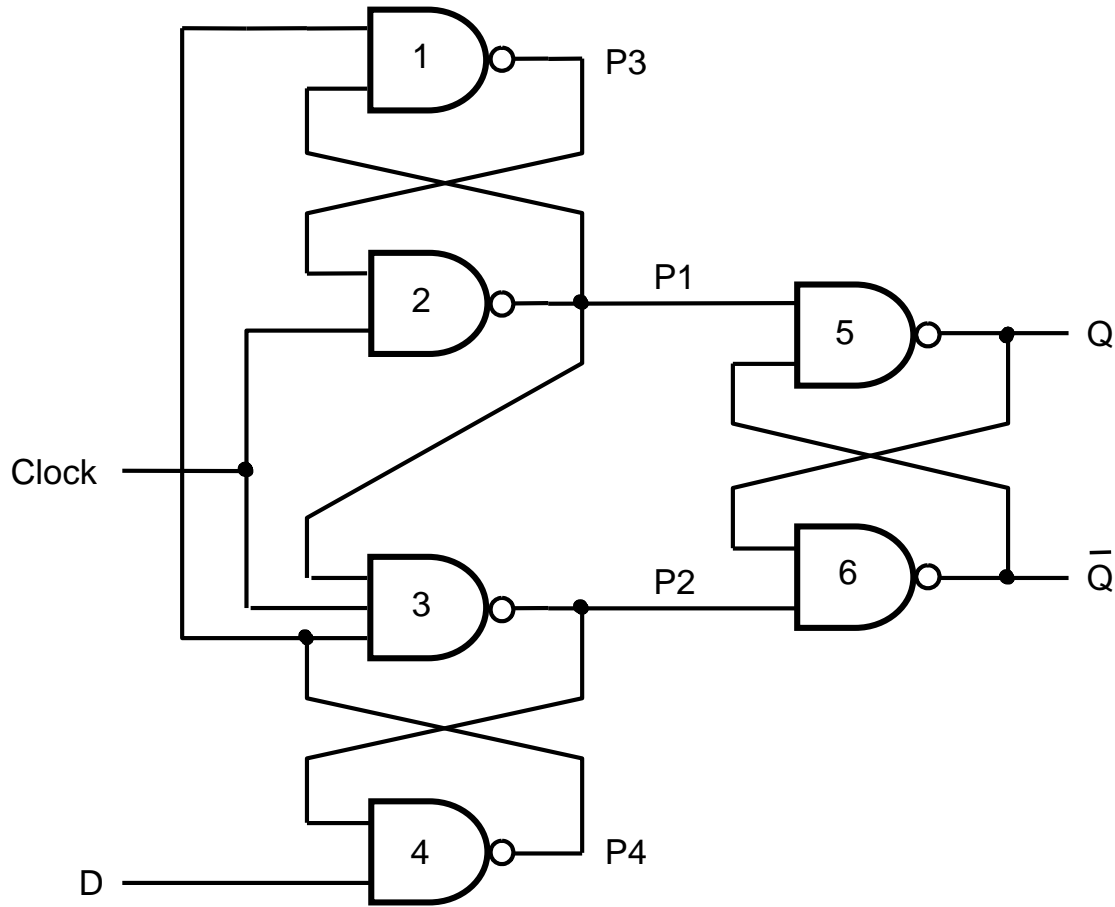


# Flip-Flop Mestre Escravo

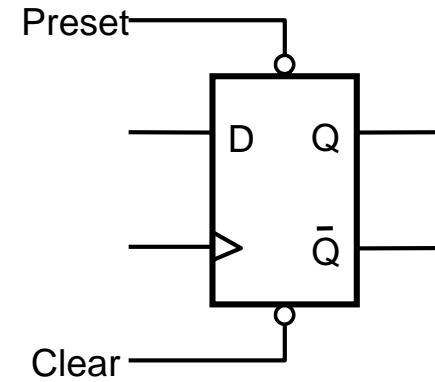
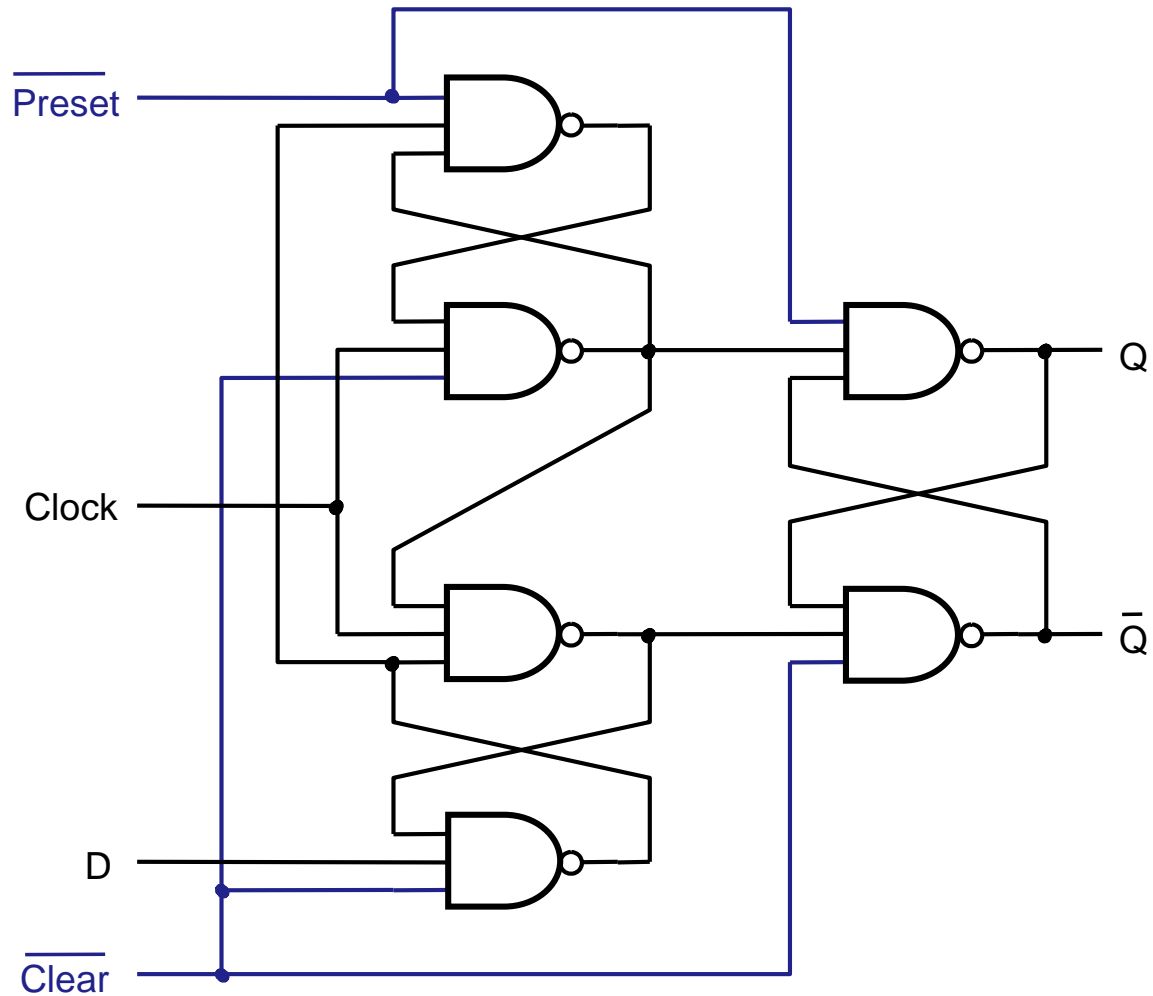


Sensível à borda de DESCIDA

# Um Flip-Flop tipo D clássico

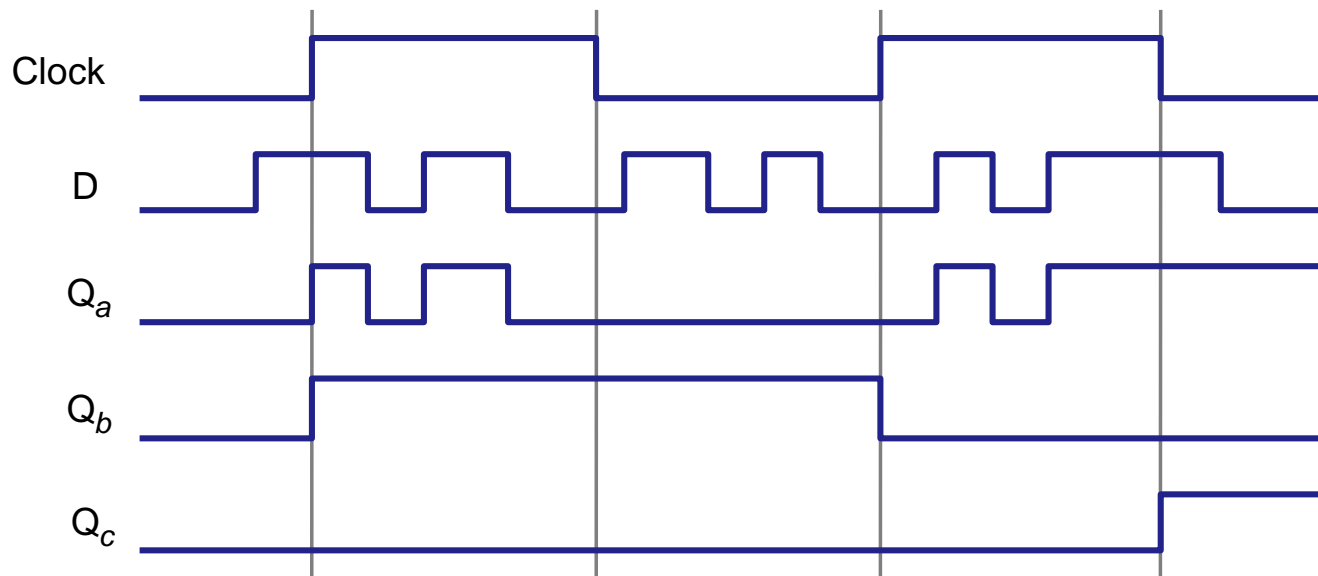
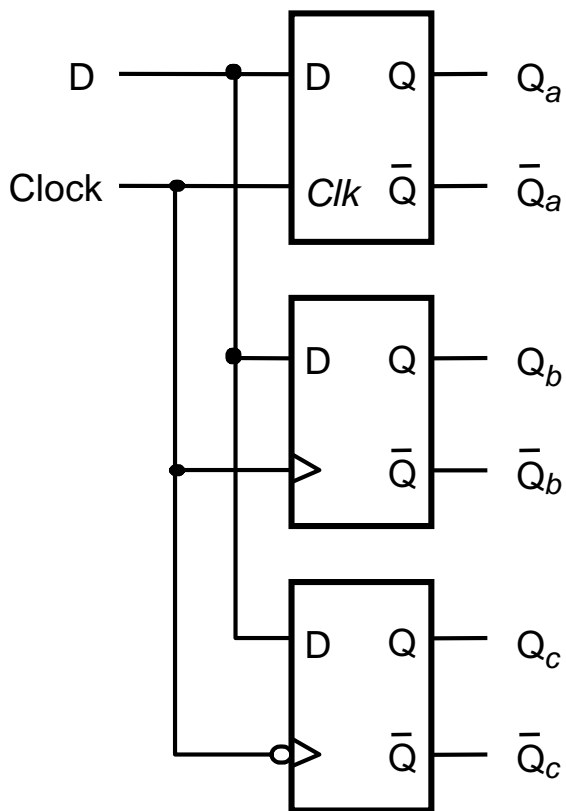


# FF D: borda de subida, com Preset e Clear assíncronos

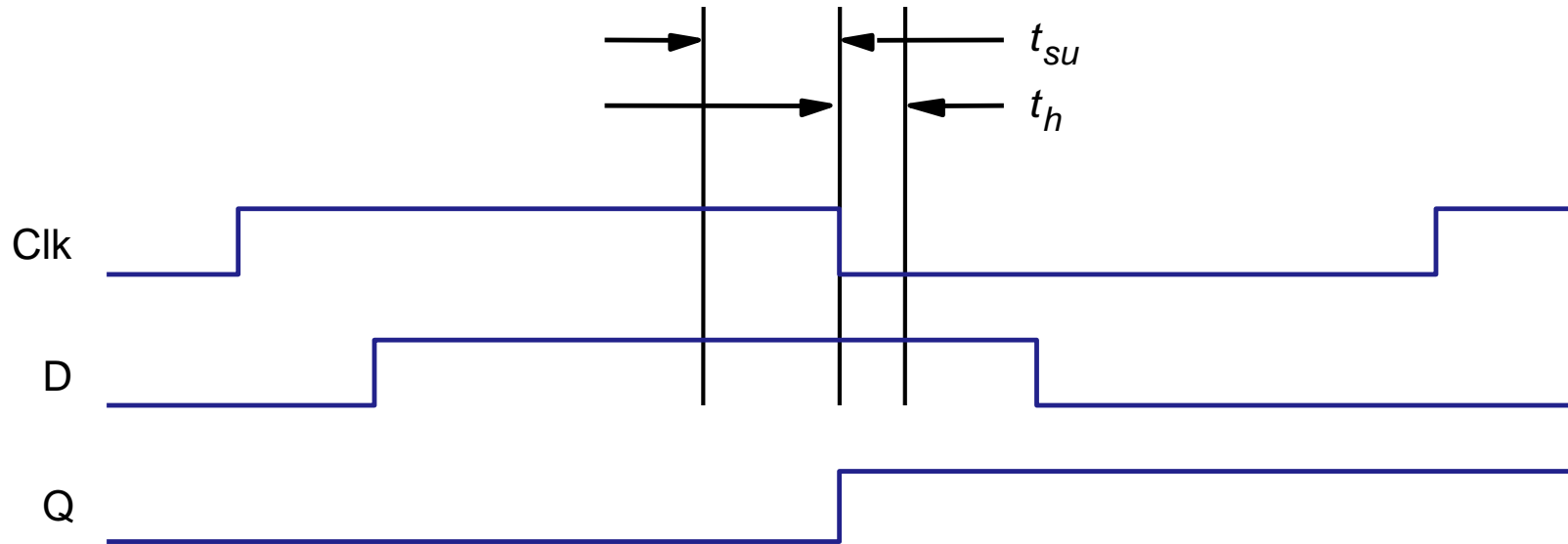




# Latch e FF: comportamento comparado

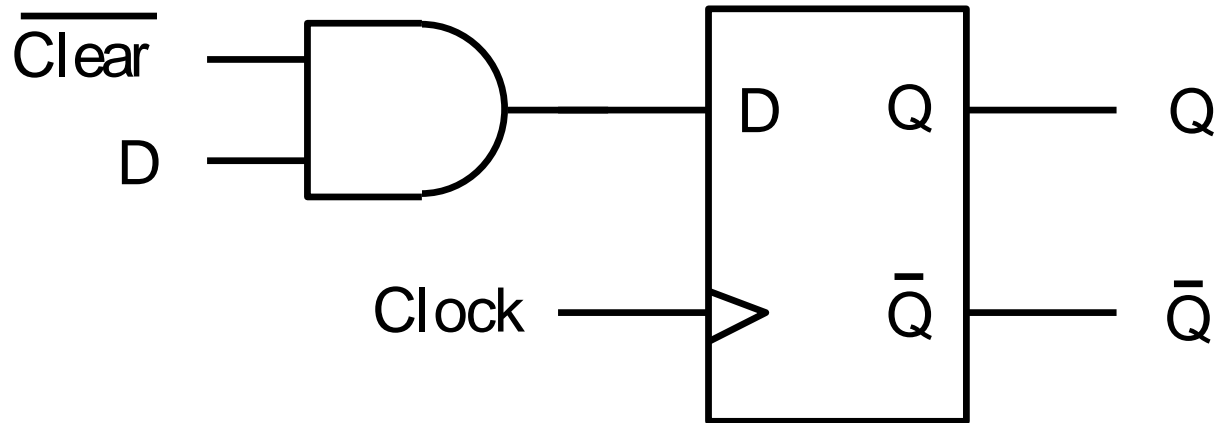


# Tempos de setup e hold

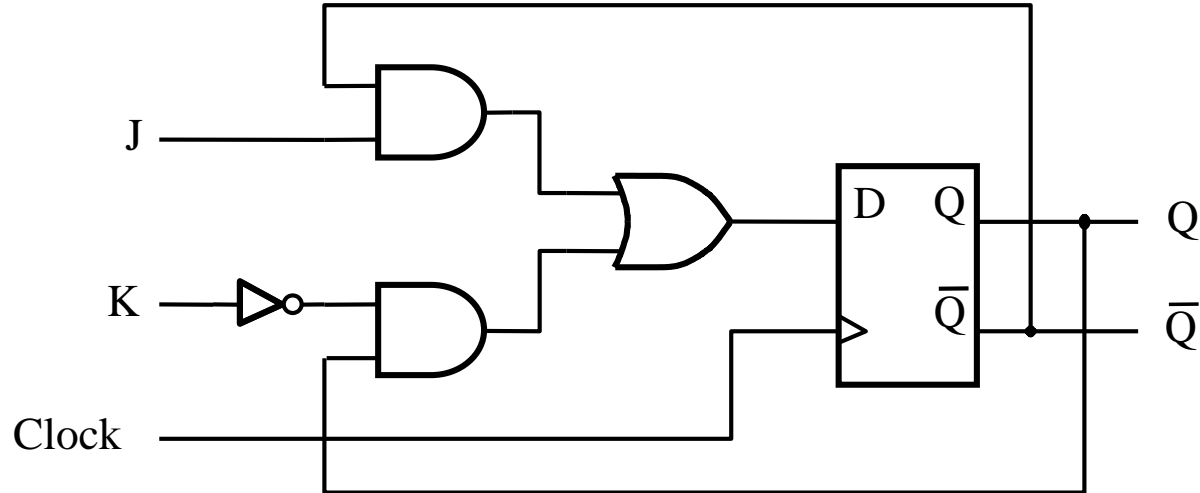


- $T_{su}$ : tempo de guarda antes da borda do clock (de descida, no exemplo) durante o qual a entrada D não deve mudar
- $T_h$ : idem, para depois da borda do clock

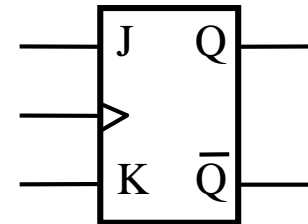
# FF D com Clear síncrono



# Flip-Flop JK

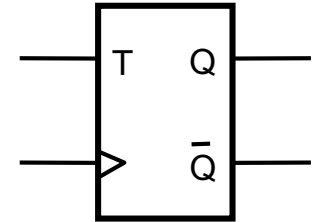
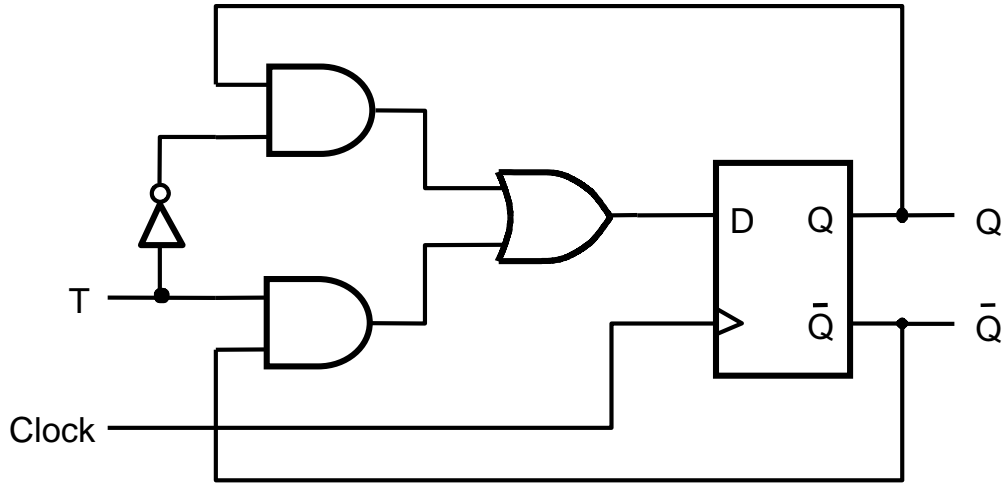


J	K	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	$\bar{Q}(t)$

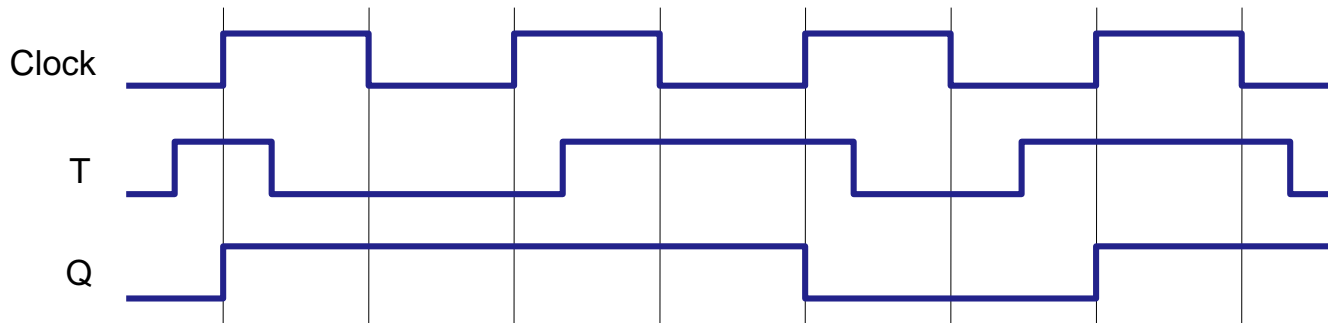
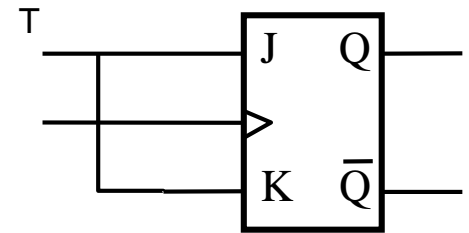


Resolve problema do SR

# Flip-Flop tipo T



Equivalente a



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



# Descrições em VHDL

- Conceito importante: process

```
PROCESS ( A, B )  
  BEGIN  
    .....  -- corpo do processo  
  END PROCESS
```

- Trecho entre Begin e End é executado sequencialmente (a ordem importa)
- O processo é executado concorrentemente como as demais declarações
- O processo é invocado quando muda algum sinal/variável na lista de sensibilidade



# Instanciação de FFD de um pacote

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

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

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

# Memória implícita



IC-UNICAMP

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

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

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



# Latch tipo D chaveado

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

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

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

# Flip-Flop tipo D



IC-UNICAMP

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

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

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



# FFD com Wait Until

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

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

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



# FFD com Reset assíncrono

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

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

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



# FFD com Reset síncrono

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

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

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





# FF-JK c reset assíncrono estrutural

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

ENTITY ffjk is
    port( J,K, Reset, Clock: in std_logic; Q: out std_logic);
END ffjk;

ARCHITECTURE Estrutural OF ffjk IS
BEGIN
    PROCESS (Clock, Reset)
        VARIABLE temp: std_logic;
    BEGIN
        IF Reset='1' THEN
            temp := '0';
        ELSIF (Clock'event and Clock='1') THEN
            temp := (J AND NOT(temp)) OR (NOT(K) and temp);
        END if;
        Q <= temp;
    END PROCESS;
END Estrutural;
```



# FF-JK comportamental

ARCHITECTURE Behavioral of ffjk is

BEGIN

```
PROCESS (Clock, Reset)
```

```
    VARIABLE temp: std_logic;
```

```
    VARIABLE jk: std_logic_vector (2 downto 1);
```

```
BEGIN
```

```
    jk := J & K;
```

```
    IF Reset='1' THEN temp := '0';
```

```
    ELSIF (Clock'event and Clock='1') then
```

```
        CASE (jk) is
```

```
            WHEN "11" => temp := not (temp);
```

```
            WHEN "10" => temp := '1';
```

```
            WHEN "01" => temp := '0';
```

```
            WHEN others => temp := temp;
```

```
        END CASE;
```

```
    END if;
```

```
    Q <= temp;
```

```
END PROCESS;
```

```
END Behavioral;
```