Sistemas Digitais 2

Projeto lógico sequencial
Latches e Flip-flops

Prof. Daniel M. Muñoz Arboleda

FGA - UnB

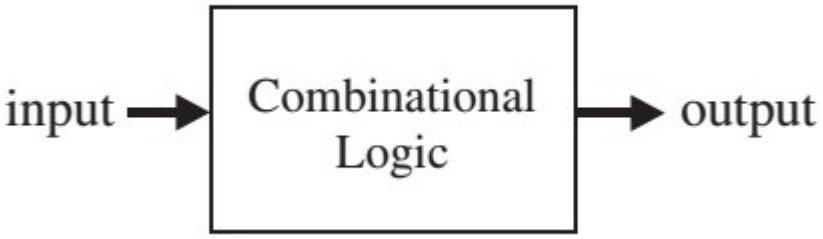
Agenda

- Projeto lógico sequencial em VHDL
  - Processos
  - Variáveis e sinais
- Latch tipo SR
- Latch SR chaveado
- Latch tipo D chaveado
- Flip-flop tipo D
- Flip-flop versus Latches
- Descrição de latches e flip-flops em VHDL
  - Inferência de latch
  - Inferência de flip-flop
- Diferenças entre rising_edge(clk) e clk'event

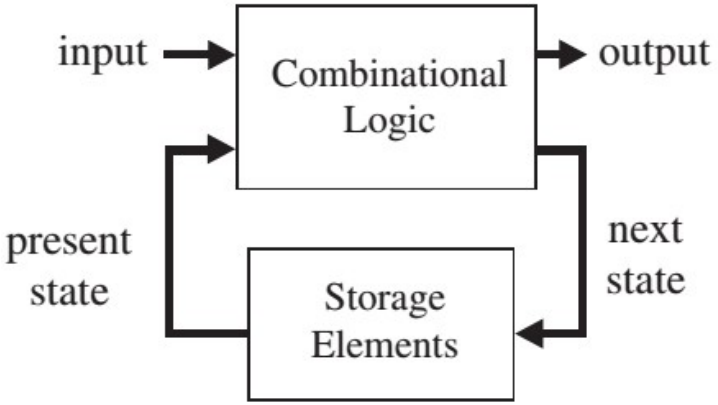
Sistemas combinacionais e sequenciais

Um *sistema digital combinacional* é qualquer sistema digital onde o comportamento de cada saída pode ser descrito como uma função que depende exclusivamente das **combinações** de valores instantâneos das entradas do sistema. Um sistema digital combinacional pode ser totalmente descrito por uma *tabela verdade*.

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

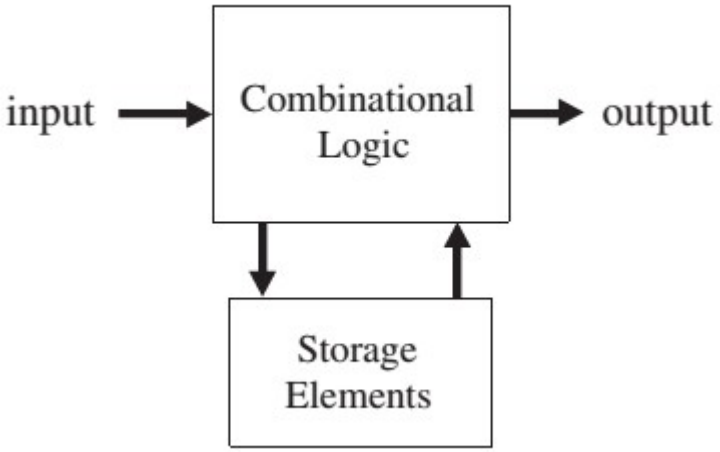


Lógica sequencial

Sistemas combinacionais e sequenciais

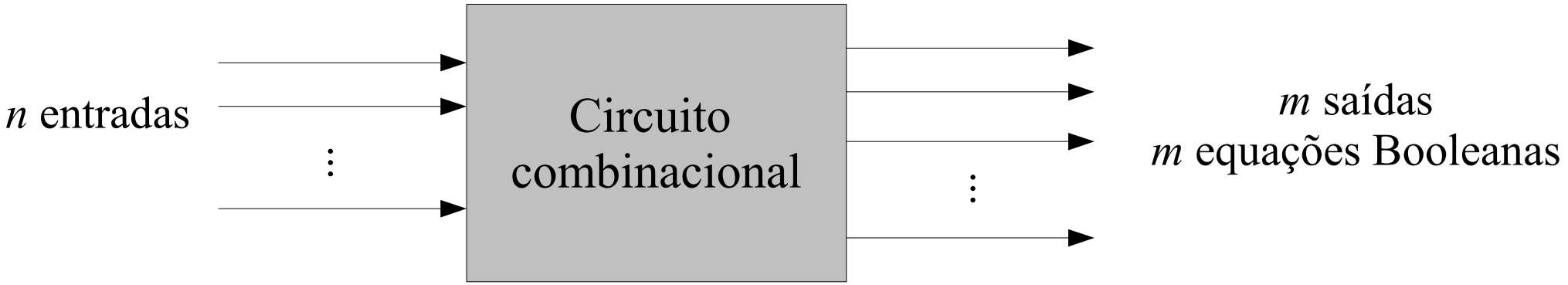
Um sistema combinacional é definido como um caso especial de um sistema sequencial.

Um erro comum é pensar que qualquer circuito que possui elementos de armazenamento (flip-flops ou latches) é sequencial. Um exemplo disto é uma memória RAM, na qual não existem loops de realimentação. Observe que o funcionamento da memória depende exclusivamente do valor atual presente no vetor de endereços. O valor na saída não depende dos acessos anteriores à memória.

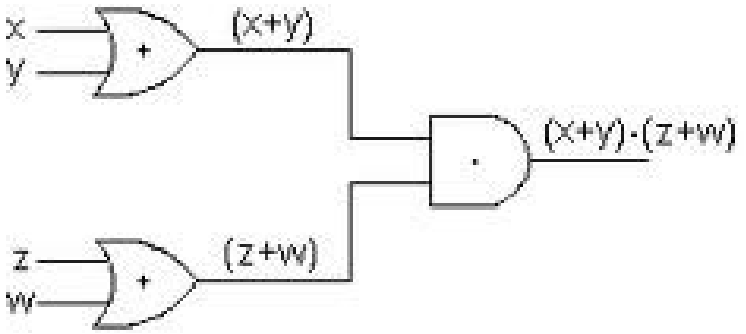


Modelo de uma memória RAM
(circuito combinacional)

Circuito combinacional



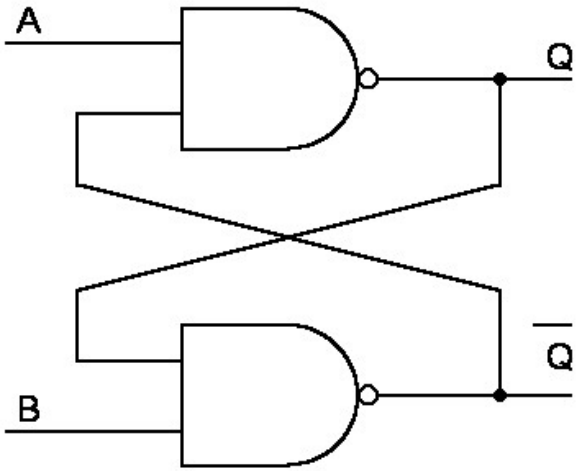
Pode-se dizer que um circuito combinacional realiza uma operação de processamento de informação a qual pode ser especificada por meio de um conjunto de equações Booleanas.



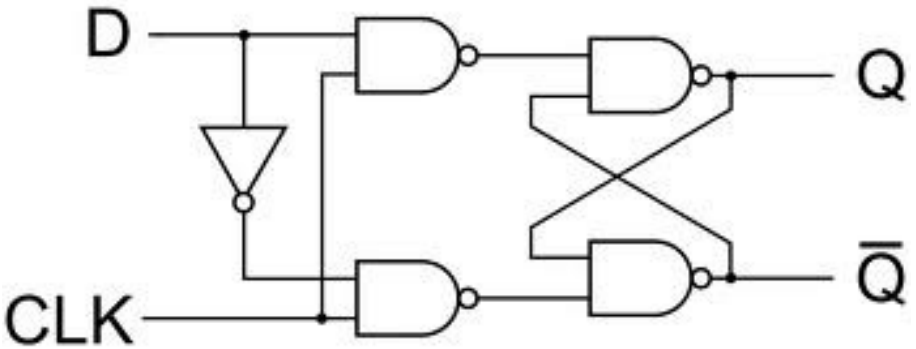
Circuito sequencial

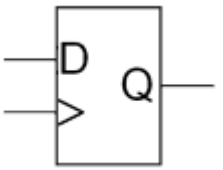
Um circuito sequencial, por sua vez, emprega elementos de armazenamento denominados latches e flip-flops, além de portas lógicas. Os valores das saídas do circuito dependem dos valores das entradas e dos estados dos latches ou flip-flops utilizados.

Latch SR



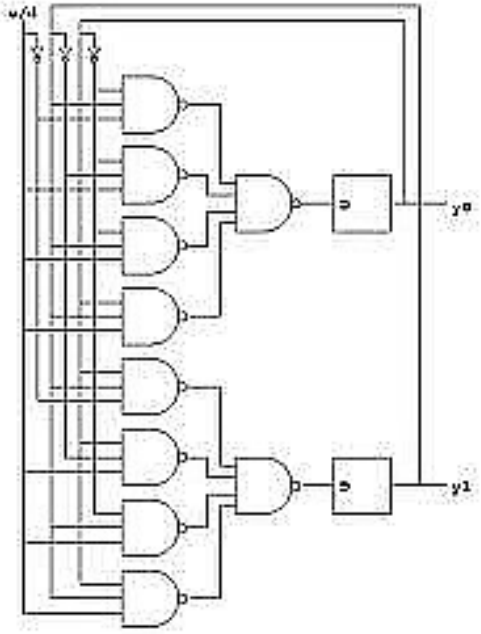
Flip-flop D

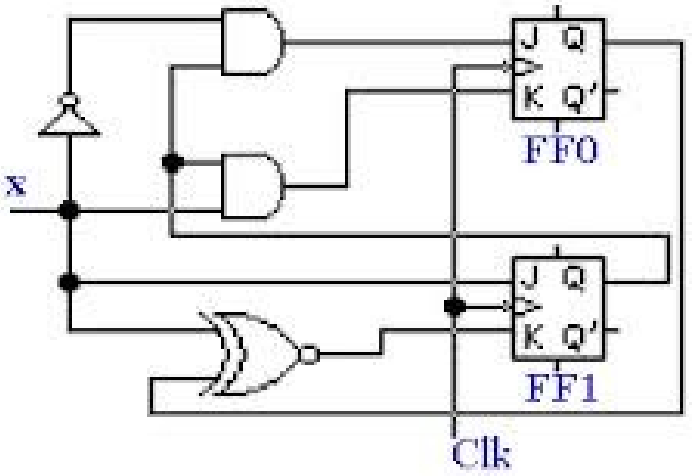




Circuito sequencial

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. Logo, o comportamento de um circuito sequencial é especificado pela sequência temporal das entradas e de seus estados internos.





Código sequencial

O código VHDL é inerentemente **concorrente** (paralelo). Em VHDL os ***Processos***, *Procedures e Funções* são os únicos trechos de código que são executados de forma sequencial.

É importante lembrar que o código sequencial não está limitado à lógica sequencial. Isto quer dizer que o código sequencial pode ser construído com circuitos sequenciais e combinacionais.

Diretivas **IF, LOOP, CASE, e WAIT** são válidas apenas dentro de ***Processos***, *Procedures e Funções.*

Variables são apenas válidas dentro de código sequencial. Diferentemente ao ***signal***, uma variável não pode ser global.

Processos

PROCESS é um segmento sequencial de código VHDL. O **PROCESS** se caracteriza pela presença de diretivas **IF**, **LOOP**, **CASE**, **WAIT** e uma **LISTA DE SENSIBILIDADE** (exceto quando o WAIT é usado).

O **PROCESS** é executado cada vez que um dos sinais da **LISTA SENSÍVEL** muda de valor.

O sintaxe é a seguinte:

```
[label:] PROCESS (sensitivity list)
  [VARIABLE name type [range] [:= initial_value;]]
BEGIN
  (sequential code)
END PROCESS [label];
```

Processos: Diferença entre sinal e variável

Em VHDL existem duas formas de passar valores não-estáticos, **variáveis** (**VARIABLES**) e **sinais** (**SIGNALS**)

VARIÁVEIS são opcionais e são apenas válidas dentro de processos. Se usadas devem ser declaradas antes do BEGIN do PROCESS. O valor inicial da variável não é sintetizável, sendo considerado apenas em simulação.

SINAIS podem ser declaradas em *packages*, *entidades* ou na *arquitetura.*

O valor de uma **variável** não pode ser usado fora do processo diretamente. Se necessário, deve-se atribuir primeiro a um **sinal**. **Variáveis são locais e sinais são globais.**

Atualização de variáveis e sinais

O valor da **variável** é atualizado imediatamente (seu valor pode ser usado na próxima linha de código). **Em contraste, o valor do sinal (quando usado em um processo) só é atualizado no final da execução atual do processo. Em outras palavras o valor do sinal só está disponível após a execução do processo.**

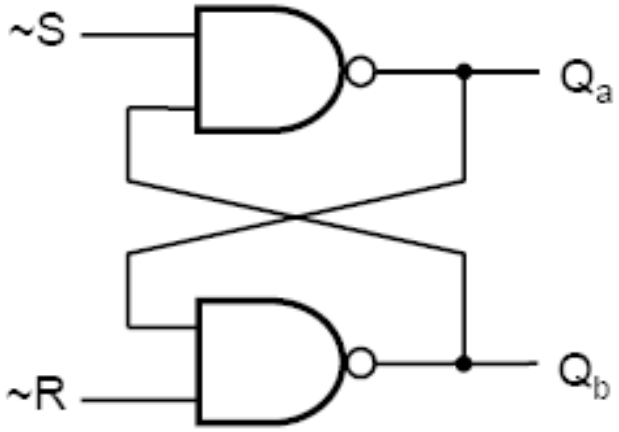
Processos: Diferença entre sinal e variável

	SIGNAL	VARIABLE
Assignment	<=	:=
Utility	Represents circuit interconnects (wires)	Represents local information
Scope	Can be global (seen by entire code)	Local (visible only inside the corresponding PROCESS, FUNCTION, or PROCEDURE)
Behavior	Update is not immediate in sequential code (new value generally only available at the conclusion of the PROCESS, FUNCTION, or PROCEDURE)	Updated immediately (new value can be used in the next line of code)
Usage	In a PACKAGE, ENTITY, or ARCHITECTURE. In an ENTITY, all PORTS are SIGNALS by default	Only in sequential code, that is, in a PROCESS, FUNCTION, or PROCEDURE

Agenda

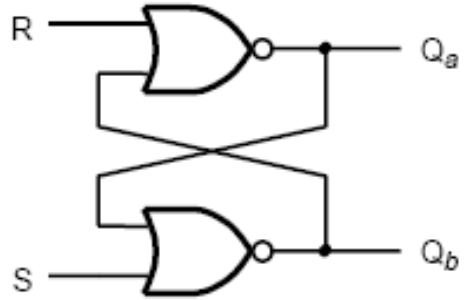
- Projeto lógico sequencial em VHDL
  - Processos, procedures e funções
- **Latch tipo SR**
- **Latch SR chaveado**
- **Latch tipo D chaveado**
- **Flip-flop tipo D**
- Flip-flop versus Latches
- Descrição de latches e flip-flops em VHDL
  - Inferência de latch
  - Inferência de flip-flop
- Diferenças entre rising_edge(clk) e clk'event

Latch SR com NAND

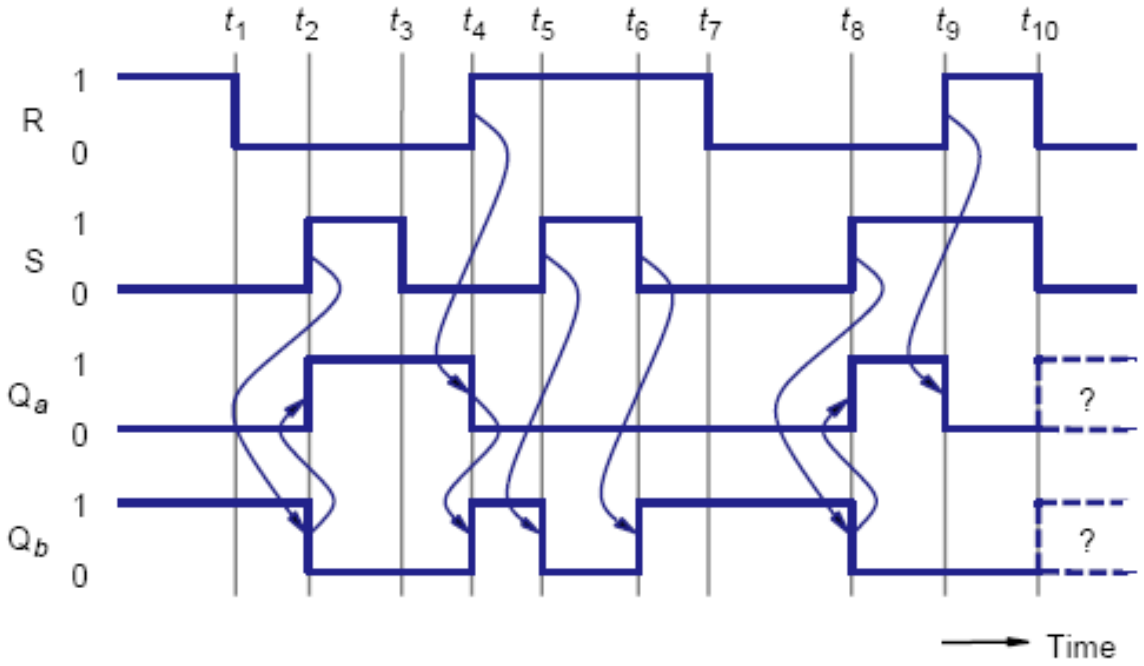


~S	~R	Q_a	Q_b	
1	1	0/1	1/0	(no change)
0	1	1	0	
1	0	0	1	
0	0	1	1	

Latch SR com NORs



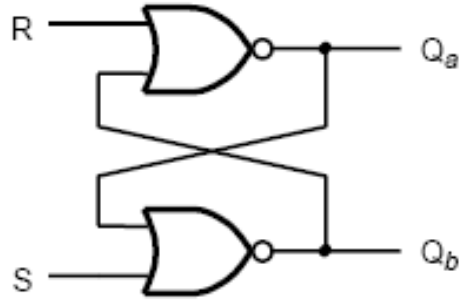
S	R	Q_a	Q_b	
0	0	0/1	1/0	(no change)
0	1	0	1	
1	0	1	0	
1	1	0	0	



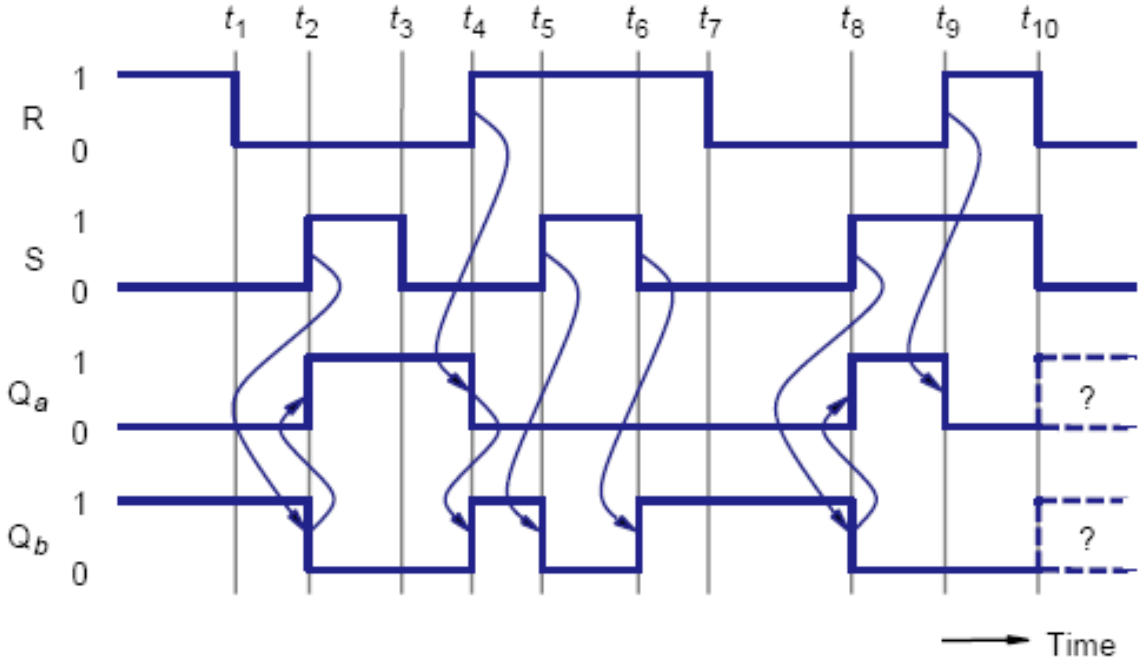
Um comportamento *oscilante* se apresenta quando S e R mudam de '1' para '0' exatamente ao mesmo tempo.

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 para sempre.

Latch SR com NORs



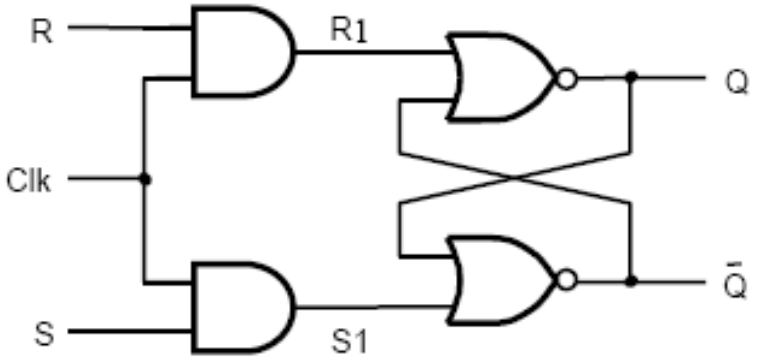
S	R	Q_a	Q_b	
0	0	0/1	1/0	(no change)
0	1	0	1	
1	0	1	0	
1	1	0	0	



Um comportamento *oscilante* se apresenta quando S e R mudam de '1' para '0' exatamente ao mesmo tempo.

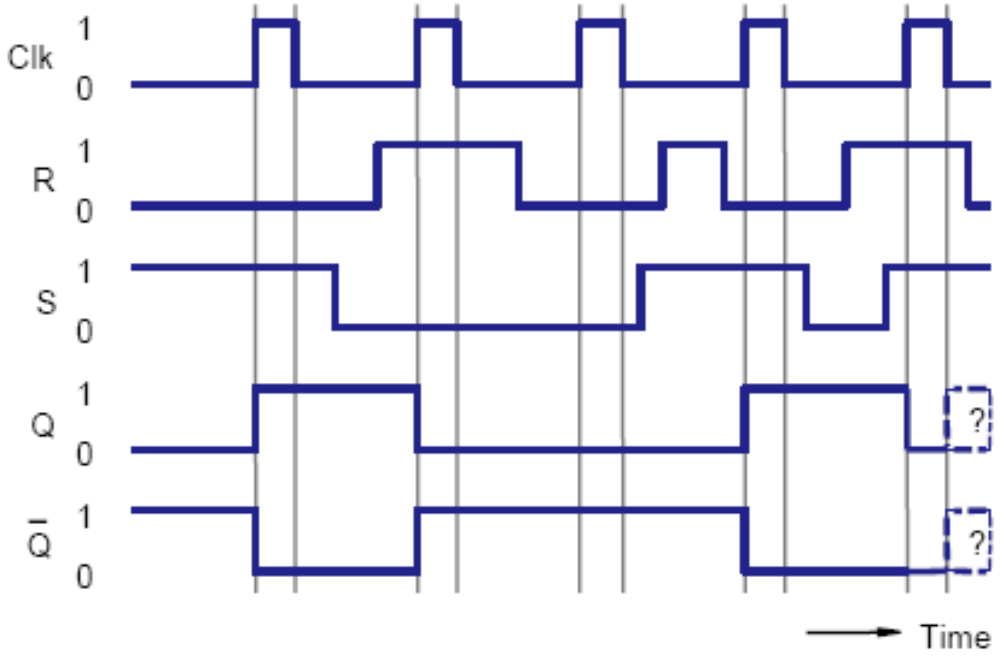
Se as portas lógicas não tem o mesmo delay, então o comportamento oscilante desaparece, porém, na prática não sabemos qual porta é mais rápida do que a outra. Portanto, não se sabe qual será o estado do Latch. Assim, é dito que o estado é *indefinido*.

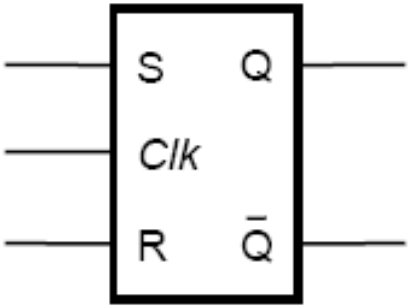
Latch SR chaveado



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

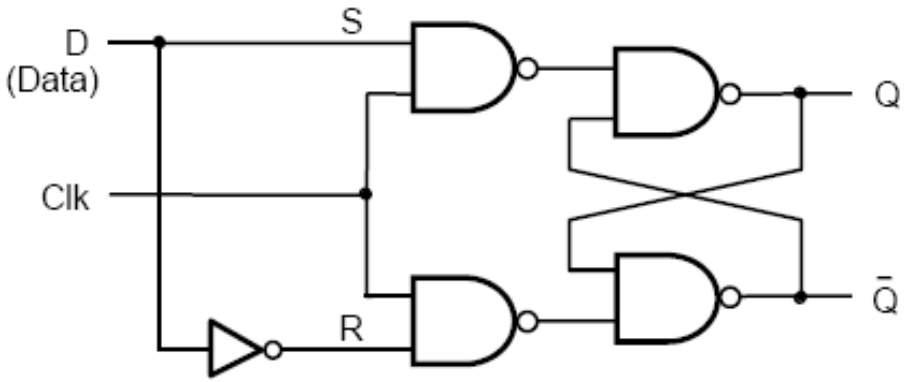
Por que?



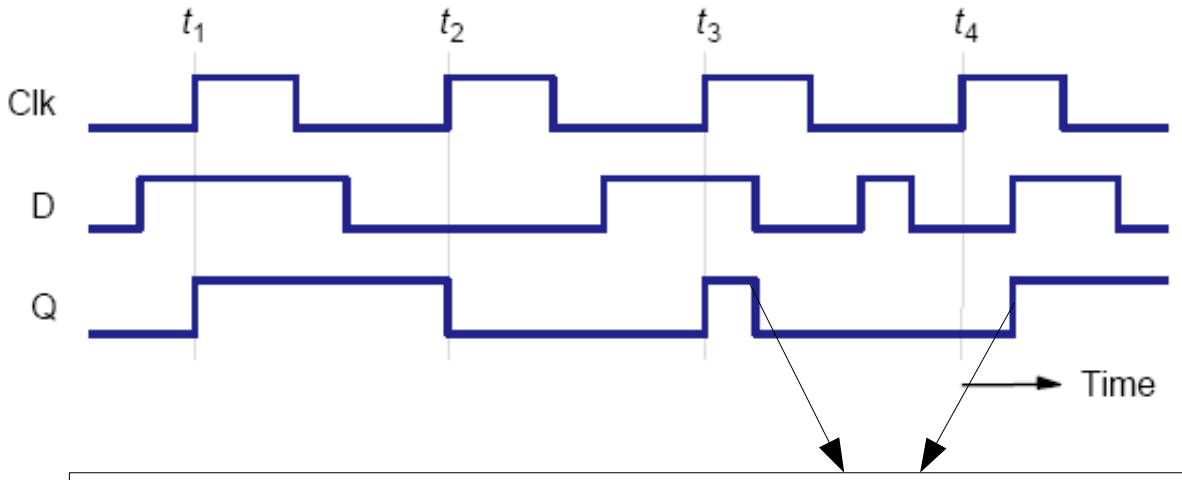


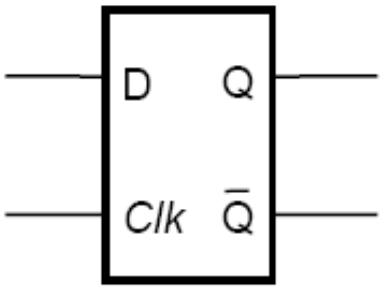
Comportamento oscilante. quando *clk* vai para '0' e S=R='1'.

Latch tipo D chaveado



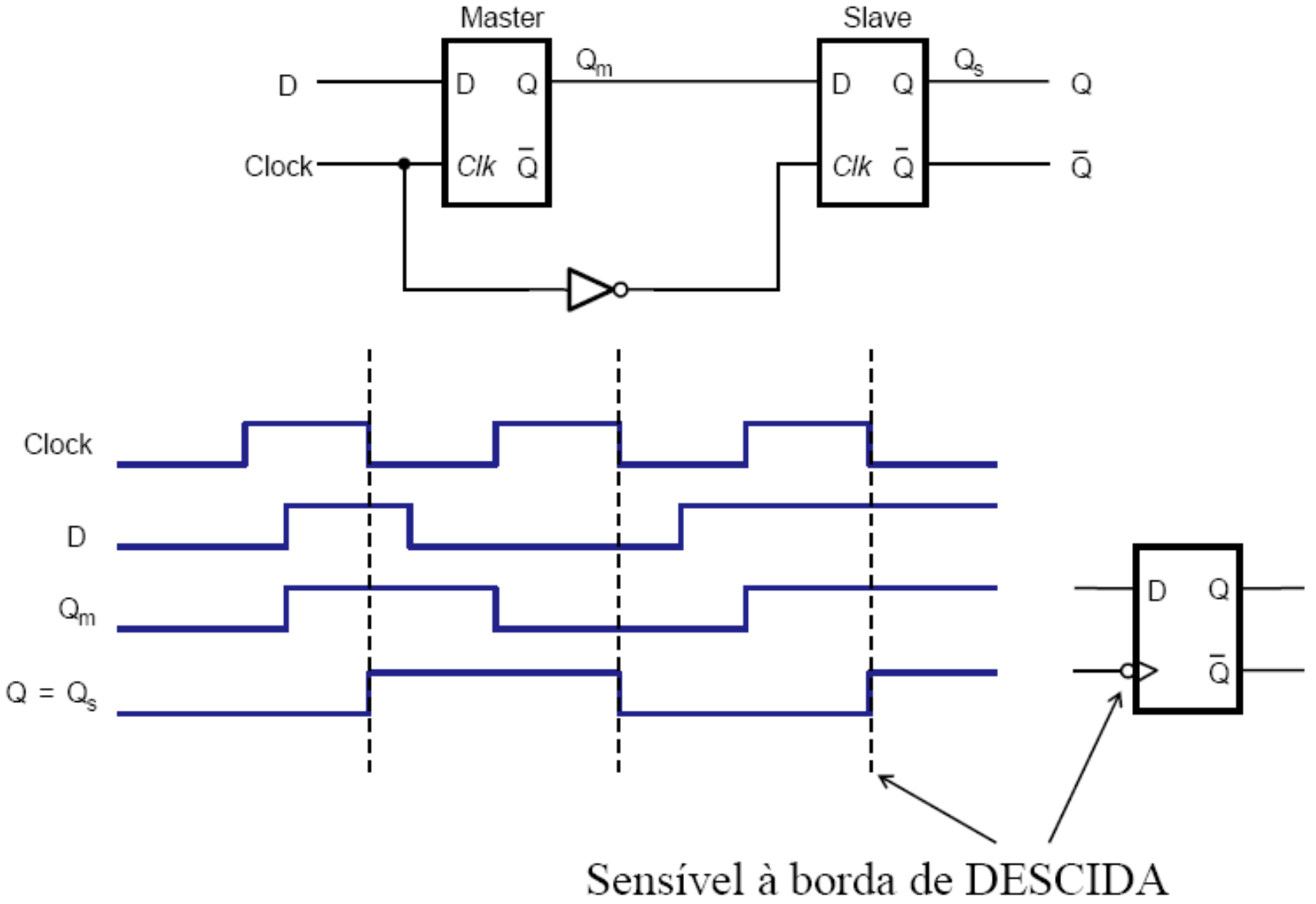
Clk	D	Q(t + 1)
0	x	Q(t)
1	0	0
1	1	1



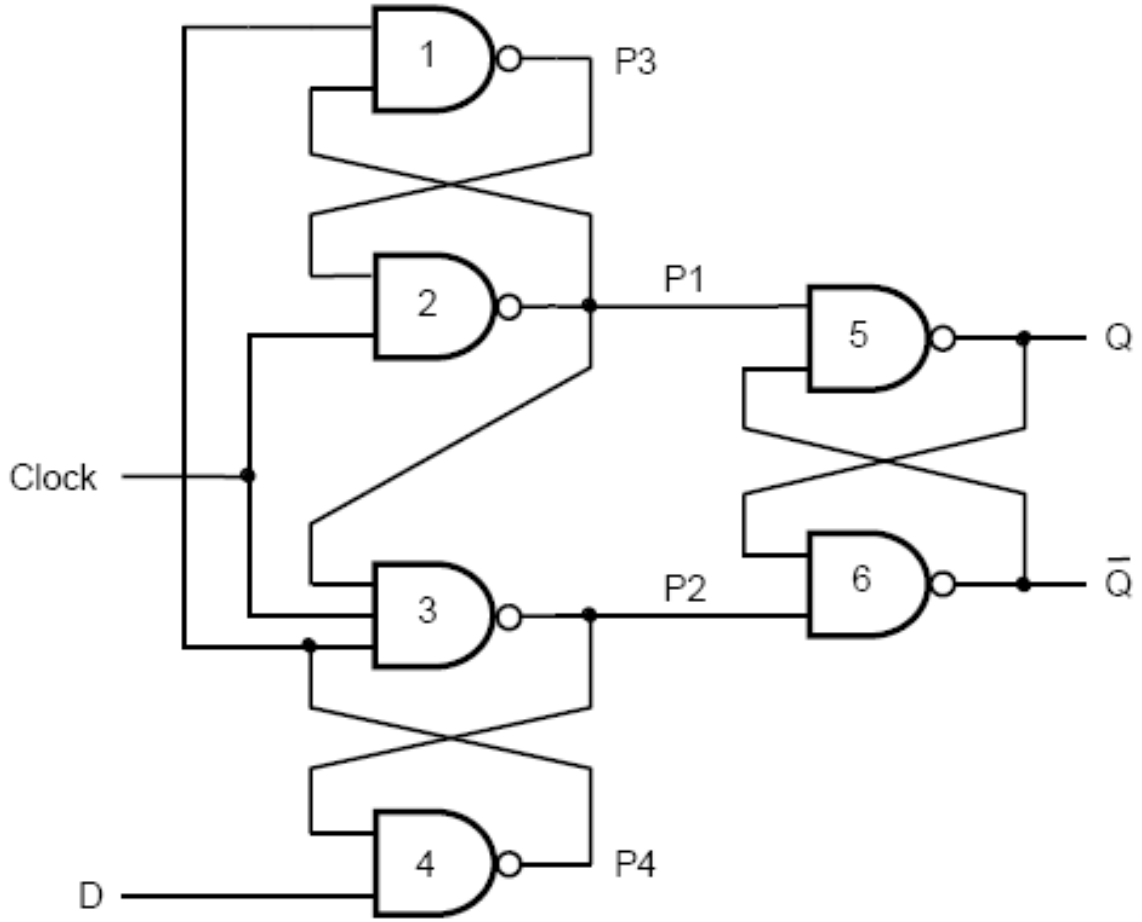


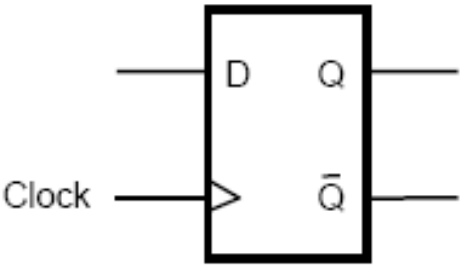
Observe-se que a saída é sensível ao nível do clock. Uma solução mais estável é sincronizar a saída com a borda do clock.

Flip-flop mestre escravo



Flip-flop D clássico





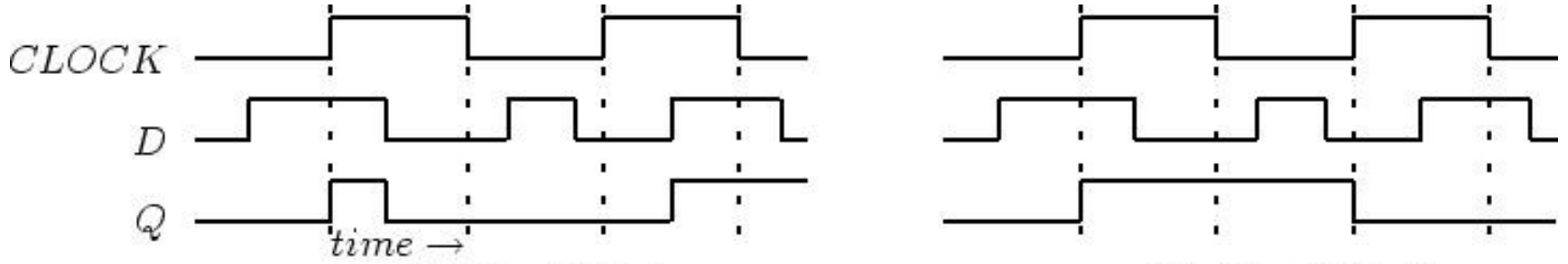
Agenda

- Projeto lógico sequencial em VHDL
  - Processos
  - Variáveis e sinais
- Latch tipo SR
- Latch SR chaveado
- Latch tipo D chaveado
- Flip-flop tipo D
- **Flip-flop versus Latches**
- Descrição de latches e flip-flops em VHDL
  - Inferência de latch
  - Inferência de flip-flop
- Diferenças entre rising_edge(clk) e clk'event

Resumo: Flip-flop D vs Latch D

- 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



(a) The D latch (b) The D flip flop

Agenda

- Projeto lógico sequencial em VHDL
  - Processos
  - Variáveis e sinais
- Latch tipo SR
- Latch SR chaveado
- Latch tipo D chaveado
- Flip-flop tipo D
- Flip-flop versus Latches
- **Descrição de latches e flip-flops em VHDL**
  - **Inferência de latch**
  - **Inferência de flip-flop**
- Diferenças entre rising_edge(clk) e clk'event

Descrições em VHDL de Flip-flop D e Latch D.

- Latches ou Flip-flops são inferidos se todas as possibilidades de um condicional IF não estão explícitas na descrição de *hardware.*

- Latch é inferido quando a sentença IF é usada por *nível lógico*

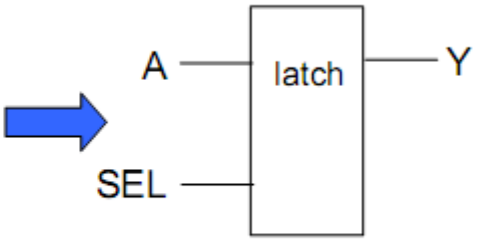
- Flip-Flop é inferido quando a sentença IF é usada por borda de clock

- As ferramentas de simulação precisam manter a saída prévia (criar uma memória) sob certas condições se as sentenças ELSE não são incluídas

Latches em VHDL

Um *latch* é inferido quando um condicional IF é usado por nível lógico e não todas as possibilidades foram incluídas

```
process (SEL, A)
begin
  if (SEL = '1') then Y <= A;
  end if ;
end process;
```

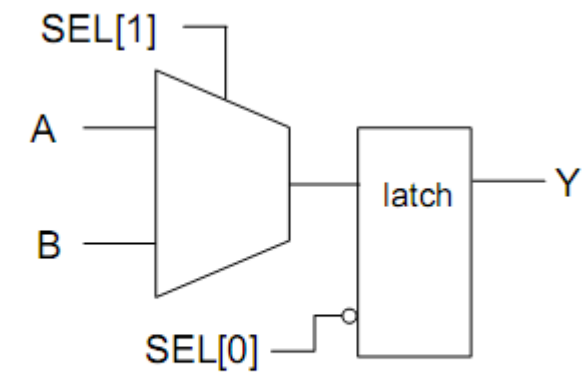


Para evitar latches, coloque as sentenças ELSE

Latches em VHDL

Um *latch* pode ser inferido quando as sentenças CASE usam
when others => null e o tipo de dado é *std_logic* ou *std_logic_vector*

```
-- sel, A, B are std_logic
process (SEL, A, B)
begin
case SEL is
  when "00" => Y <= A;
  when "10" => Y <= B;
  when others => null;
end case;
end process;
```



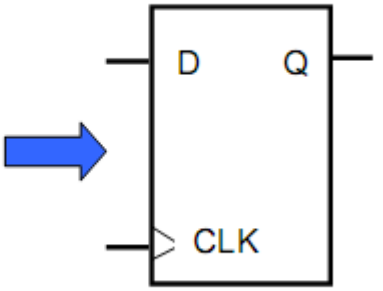
Para evitar latches, defina a saída para as outras condições, por exemplo,
when others => Y <= '0' ;

Flip-flops em VHDL

Dica: use Processos e condicionais IF para descrever lógica sequencial.

Um flip-flip é inferido se os condicionais IF detectam bordas de clock.

```
architecture BEHAVE of DF is
begin
  INFER: process (CLK) begin
    if (CLK'event and CLK ='1') then
         Q <= D;
    end if ;
  end process INFER;
end BEHAVE;
```



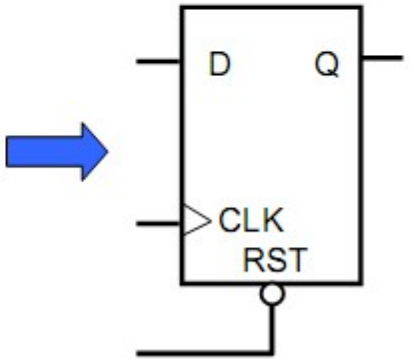
Flip-flops em VHDL (exemplo 1)

Um flip-flip é inferido se os condicionais IF detectam bordas de clock.

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



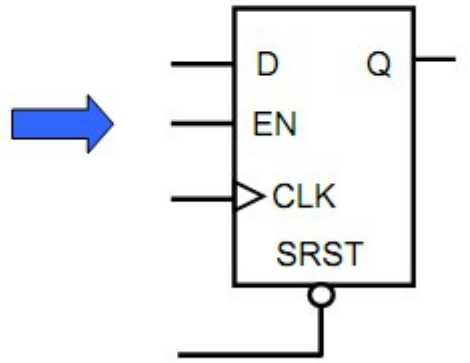
Flip-flops em VHDL (exemplo 2)

Um flip-flop é inferido se os condicionais IF detectam bordas de clock.
Dica: descreva lógica sequencial usando *reset* síncrono !

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;
```

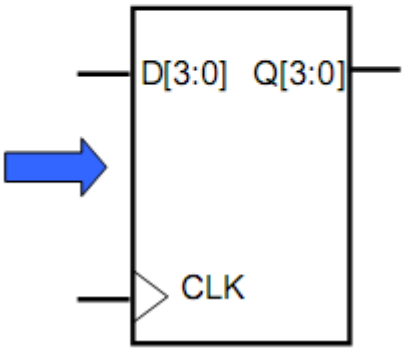
D flip-flop with synchronous low reset, active high enable and rising edge clock



Flip-flops em VHDL (exemplo 3)

```
library ieee;
use ieee.std_logic_1164.all;
entity DF_4 is
port (D: in std_logic_vector(3 downto 0);
      CLK: in std_logic;
      Q: out std_logic_vector(3 downto 0));
end DF_4 ;
architecture FLOP of DF_4 is
begin
  INFER: process ←— Where's the sensitivity list?
  begin
    wait until (CLK'event and CLK ='1');
         Q <= D;
  end process INFER;
end FLOP;
```

4-bit register using WAIT statement



Flip-flops em VHDL (recapitulando)

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

1. flip-flop ativo em borda de subida ou descida ?
2. O *enable* é síncrono ou assíncrono ?
3. O *enable* é ativo em nível lógico alto ou baixo ?

Agenda

- Projeto lógico sequencial em VHDL
  - Processos
  - Variáveis e sinais
- Latch tipo SR
- Latch SR chaveado
- Latch tipo D chaveado
- Flip-flop tipo D
- Flip-flop versus Latches
- Descrição de latches e flip-flops em VHDL
  - Inferência de latch
  - Inferência de flip-flop
- **Diferenças entre rising_edge(clk) e clk'event**

Flip-flops em VHDL

Outra forma de indicar a borda de subida e descida do clock é usando a diretiva **rising_edge(clk)** e **falling_edge(clk)**, respectivamente.

```
 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-flops em VHDL

Dica1: use **rising_edge(clk)** ou **falling_edge(clk)** em lugar de **clock'event**
Dica2: use sempre a mesma borda (subida ou descida) em todo o circuito.

```
FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN BOOLEAN IS
BEGIN
    RETURN (s'EVENT AND (To_X01(s) = '1') AND
                        (To_X01(s'LAST_VALUE) = '0'));
END;

FUNCTION To_X01  ( s : std_ulogic ) RETURN  X01 IS
BEGIN
    RETURN (cvt_to_x01(s));
END;

CONSTANT cvt_to_x01 : logic_x01_table := (
                    'X',  -- 'U'
                    'X',  -- 'X'
                    '0',  -- '0'
                    '1',  -- '1'
                    'X',  -- 'Z'
                    'X',  -- 'W'
                    '0',  -- 'L'
                    '1',  -- 'H'
                    'X'   -- '-'
                   );
```

rising_edge(clk) é mais descritivo, pois também considera outras bordas positivas.

Ex: transição de '0' a 'H'. rising_edge(clk) dispara, (clk'event and clk = '1') não dispara.