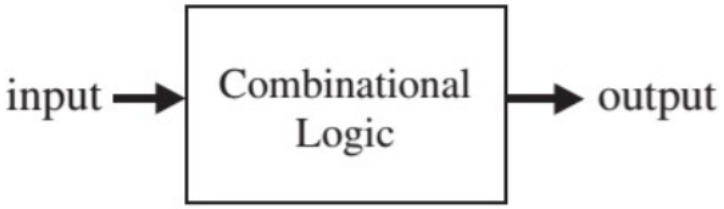
Aula 6 – Projeto Lógico Sequencial

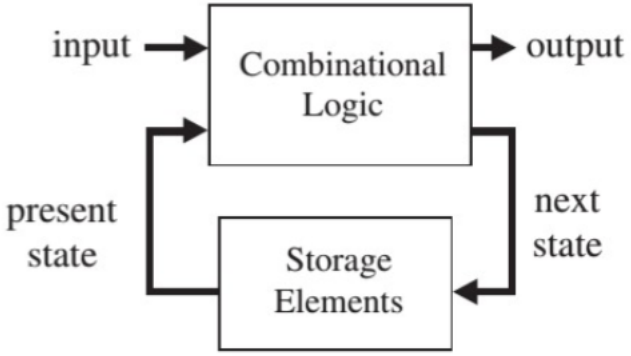
Latches, Flip-Flops, Divisor de *Clock*

Combinacional x Sequencial

- **Combinacional**: saída depende das combinações das entradas (tabela verdade).
- **Sequencial**: possui estado, ou seja, depende de valores passados das entradas (memória)



Lógica combinacional

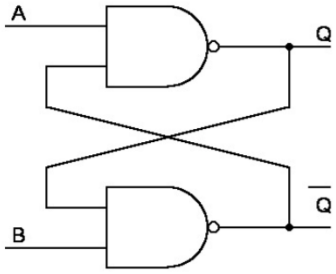


Lógica sequencial

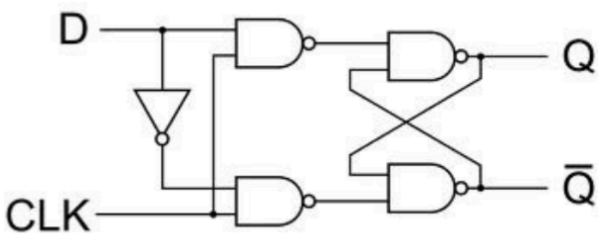
Circuito Sequencial

- Além de portas lógicas, emprega latches e flip-flops como elementos de armazenamento

Latch SR



Flip-flop D



- Comportamento é especificado pela sequência temporal das entradas e de seus estados internos

Código Sequencial

- Em VHDL, o código é executado de forma **concorrente/paralela**
- Entretanto, trechos de códigos em **processos**, **procedimentos** e **funções** são executados de forma **sequencial**
- **Variáveis** são válidas apenas dentro de código sequencial. Diferentemente do **sinal**, uma variável não pode ser global

Processos, Sinais e Variáveis

Processos

- Segmentos sequenciais de código VHDL
- Diretivas ***if***, ***loop***, ***case*** e ***wait*** são válidas apenas dentro de processos, procedimentos e funções
- Possuem uma **lista de sensibilidade** (exceto quando ***wait*** é usado)
- São executados quando um dos sinais da lista muda de valor

Processos

- Sintaxe:

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

- **Variáveis**: declaradas antes do ***begin.*** Valor inicial não é sintetizável
- **Sinais**: podem ser declarados em pacotes, entidades ou na arquitetura

Variáveis x Sinais

- Variáveis são locais e sinais são globais
- O valor da variável é atualizado imediatamente
- O valor do sinal, quando usado em processo, só é atualizado no final da execução do mesmo

Variáveis x Sinais

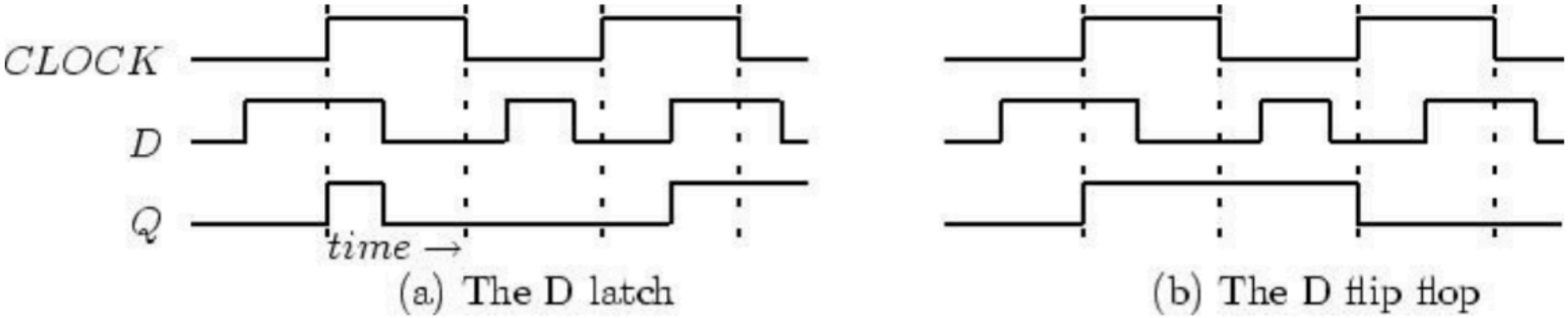
	SIGNAL	VARIABLE
Assignment	<=	:=
Utility	Represents circuit interconnects (wires)	Represents local information
Scope	Can be global (seen by entire code)	Local (visible only inside the corresponding PROCESS, FUNCTION, or PROCEDURE)
Behavior	Update is not immediate in sequential code (new value generally 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

Latches e Flip-Flops

Latch D x Flip-Flop D

- Manifestação da saída Q em função da entrada D:
  - **Latch**: transparente quando EN (ou CK) está ativo
  - **Flip-Flop**: D é transferida para Q na borda de subida do clock
- Instante em que D é armazenada:
  - **Latch**: no instante em que EN (ou CK) é desativado
  - **Flip-Flop**: na borda do clock

Latch D x Flip-Flop D



(a) The D latch

(b) The D flip flop

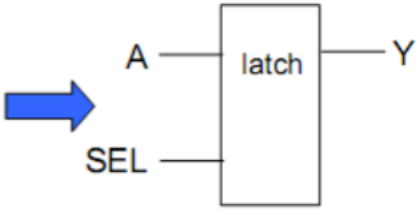
Em VHDL

- Latches ou flip-flops são inferidos se todas as possibilidades de um ***if*** não estão explícitas
- **Latch** é inferido quando ***if*** é usado por **nível lógico**
- **Flip-flop** é inferido quando ***if*** é usado por **borda de clock**
- Ferramentas de simulação precisam manter a saída prévia (memória) sob certas condições se ***else*** não é incluído

Latches em VHDL

- **Latch** é inferido quando ***if*** é usado por **nível lógico** e nem todas as possibilidades são incluídas

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

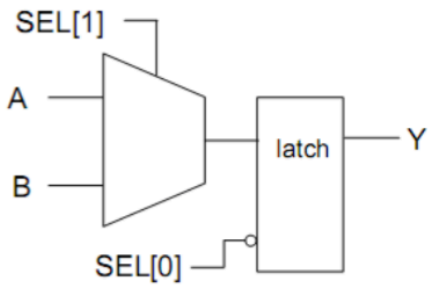


Para evitar latches, coloque as sentenças ELSE

Latches em VHDL

- **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;
```

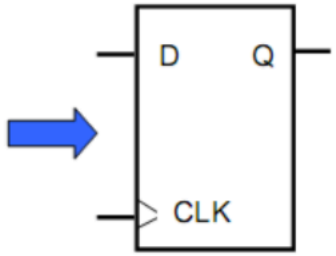


Para evitar latches, defina a saída para as outras condições, por exemplo,
when others => Y <= '0' ;

Flip-Flops em VHDL

- Um **flip-flop** é inferido se a condicional ***if*** detecta borda de clock
- **Dica**: usar processos e a condicional ***if*** para descrever lógica sequencial

```
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;
```



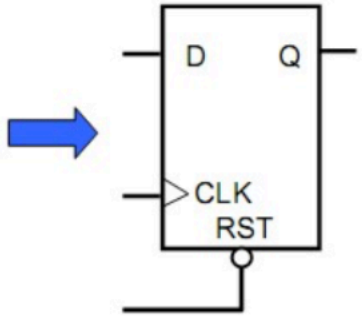
Flip-Flops em VHDL

- Exemplo 1: 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



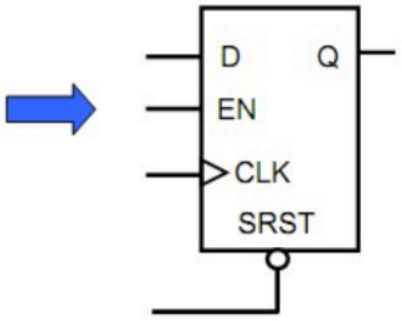
Flip-Flops em VHDL

- Exemplo 2: reset síncrono (sempre usar em lógica sequencial!)

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

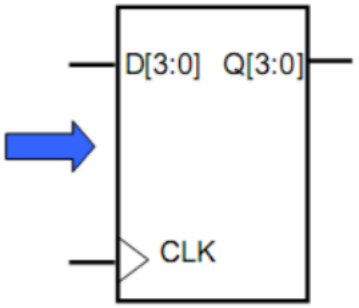


Flip-Flops em VHDL

- Exemplo 3: registrador de 4 bits

```
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



Recapitulando:

- Flip-flop ativo em borda de subida ou descida?
- Enable síncrono ou assíncrono? Ativo em nível alto ou baixo?

```
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;
```

rising_edge(clk)

- ***rising_edge(clk):*** borda de subida
- ***falling_edge(clk):*** borda de descida

```
 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;
```

rising_edge(clk) x clk'event

- ***rising_edge(clk)*** é mais descritivo, pois considera também outras bordas positivas que são ignoradas por ***clk'event and clk = '1'*** (e.g., transição de '0' para 'H')
- **Dica 1**: usar ***rising_edge(clk)*** ou ***falling_edge(clk)*** ao invés de ***clk'event***
- **Dica 2**: usar sempre a mesma borda (subida ou descida) em todo o circuito

Divisor de *Clock*

Divisor de *Clock*

- **Objetivo:** reduzir a frequência de entrada
- Basta utilizar um fator de escala, que é a relação entre as frequências de entrada e a desejada na saída

$$Scale = \frac{f_{in}}{f_{out}}$$

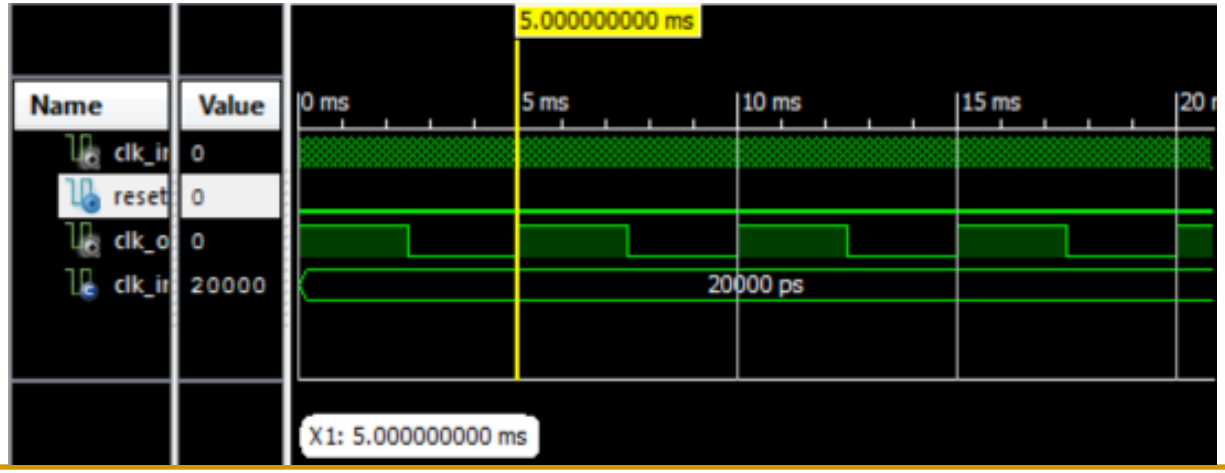
- **Exemplo**: assumindo f_{in} = 50 MHz, para obter f_{out} = 200 Hz, o fator de escala é 250.000

Divisor de *Clock*

- Considerando-se o *duty cycle* de 50%, o sinal de saída deve ficar ativo e inativo durante o mesmo período (**250.000/2 = 125.000**)
- Cria-se um contador (**counter**),incrementado a cada borda de subida do clock, para contar de **0** a **124.99,** quando muda-se o nível do valor de saída (**temp <= not temp**) e reinicia-se a contagem (**counter <= 0**);

Divisor de *Clock*

- Como a contagem começa em 0, quando a mesma atingir (**250.000/2)-1 = 124.999**, muda-se o nível do valor de saída (**temp <= not temp**) e reinicia-se a contagem (**counter <= 0**);



Divisor de *Clock*

```
architecture Behavioral of clock_divider_example is
   signal counter: integer:=0;
   signal temp: std_logic:='1';
begin
   process(clk_in,reset)
   begin
      if reset = '1' then
         counter <= 0;
         temp <= '1';
      elsif rising_edge(clk_in) then
         if (counter = 124999) then
            temp <= not temp;
            counter <= 0;
         else
            counter <= counter + 1;
         end if;
      end if;
   end process;
   clk_out <= temp;
end Behavioral;
```