

# Laboratório de Sistemas Digitais

## Aula Teórica-Prática 4

Ano Letivo 2024/25

Modelação em VHDL de circuitos sequenciais  
elementares, contadores, divisores de  
frequência e temporizadores

Parametrização de componentes sequenciais

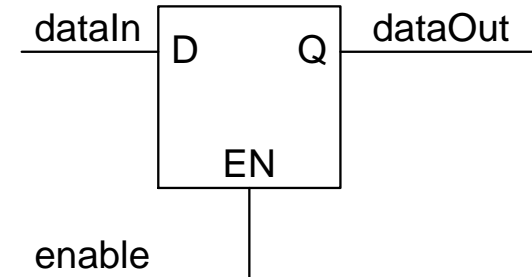
# Conteúdo

- Modelação em VHDL de circuitos sequenciais
  - Latch D
  - Flip-flop tipo D
  - Registos
    - Parametrização
  - Contadores
    - Divisores de frequência
    - Temporizadores

# Módulo Sequencial Trivial – Latch D

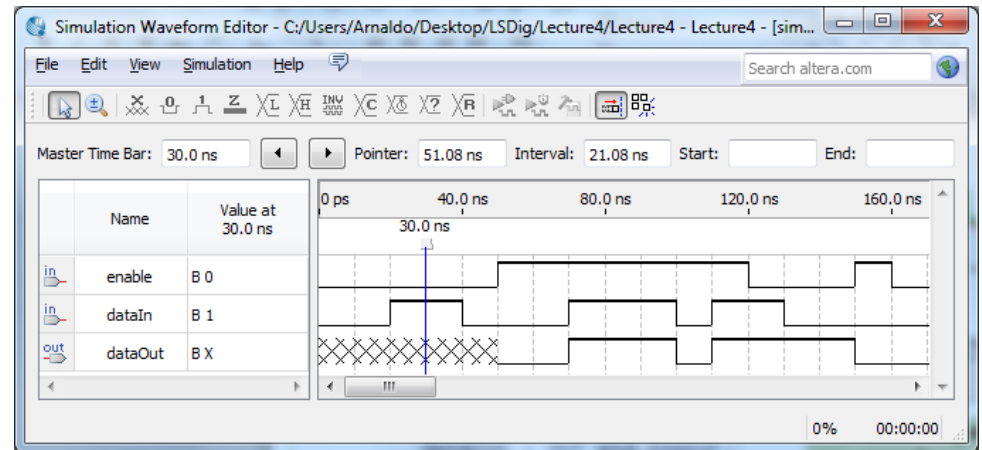
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity LatchD is
    port(enable : in std_logic;
          dataIn : in std_logic;
          dataOut : out std_logic);
end LatchD;
```



A saída “segue” a entrada  
quando enable= '1'

```
architecture Behav of LatchD is
begin
    process(enable, dataIn)
    begin
        if (enable = '1') then
            dataOut <= dataIn;
        end if;
    end process;
end Behav;
```



Porque razão o sinal dataOut surge inicialmente como “XXXXXXXX”? É relevante? Como resolver?

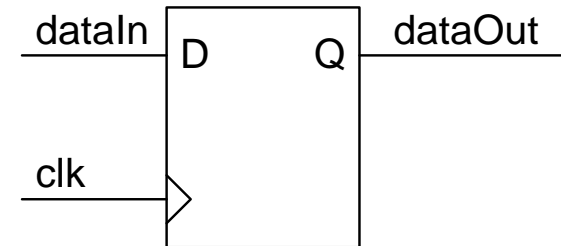
# Módulo Sequencial Simples com Clock

## Flip-flop tipo D

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity FFD is
    port (clk      : in  std_logic;
          dataIn   : in  std_logic;
          dataOut  : out std_logic);
end FFD;

architecture Behav of FFD is
begin
    process (clk)
    begin
        if (clk'event and clk = '1') then
            dataOut <= dataIn;
        end if;
    end process;
end Behav;
```

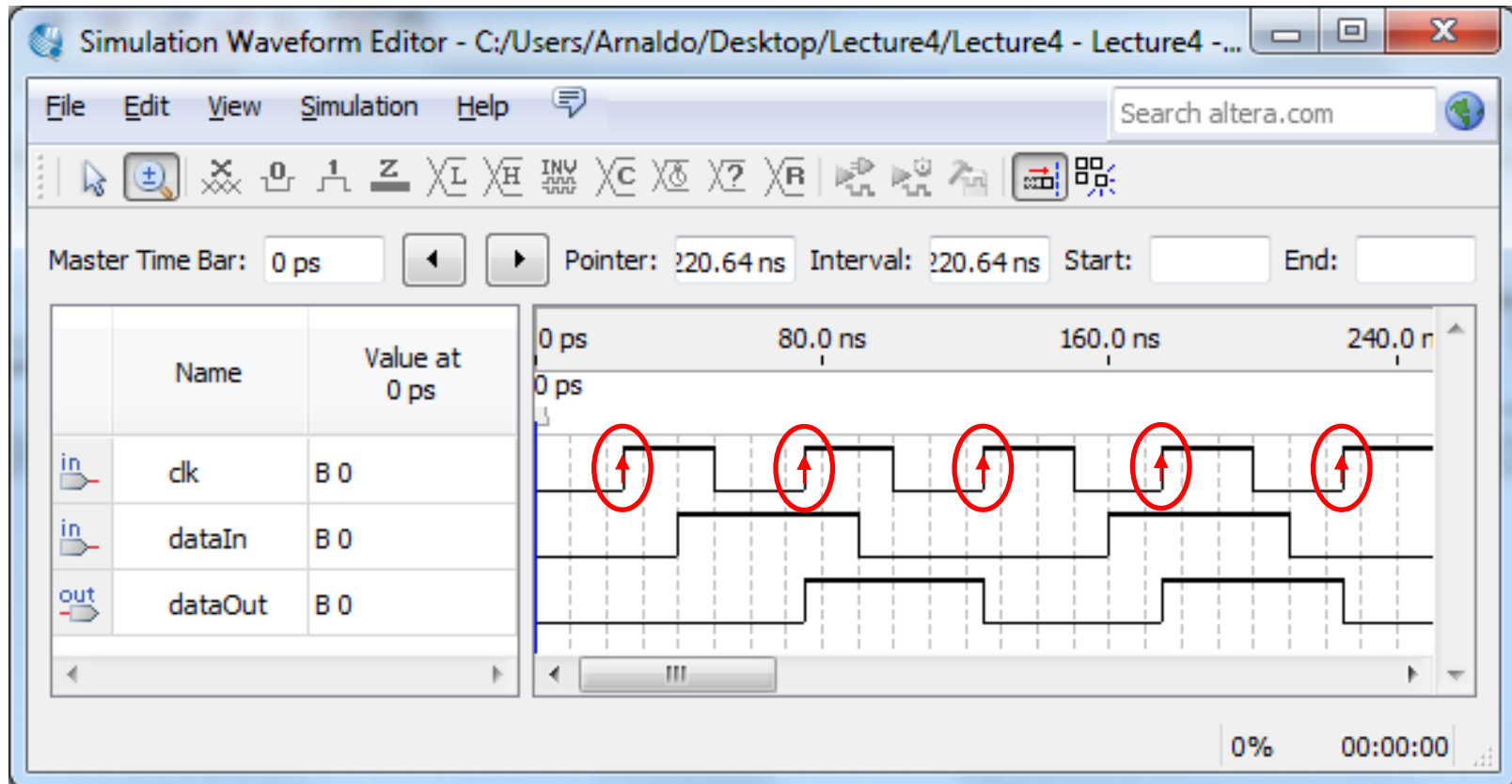


Porque razão apenas o `clk` é incluído na lista de sensibilidade?

`clk'event and clk = '1'` é equivalente a `rising_edge(clk)`

`clk'event and clk = '0'` é equivalente a `falling_edge(clk)`

# Simulação do Flip-Flop D

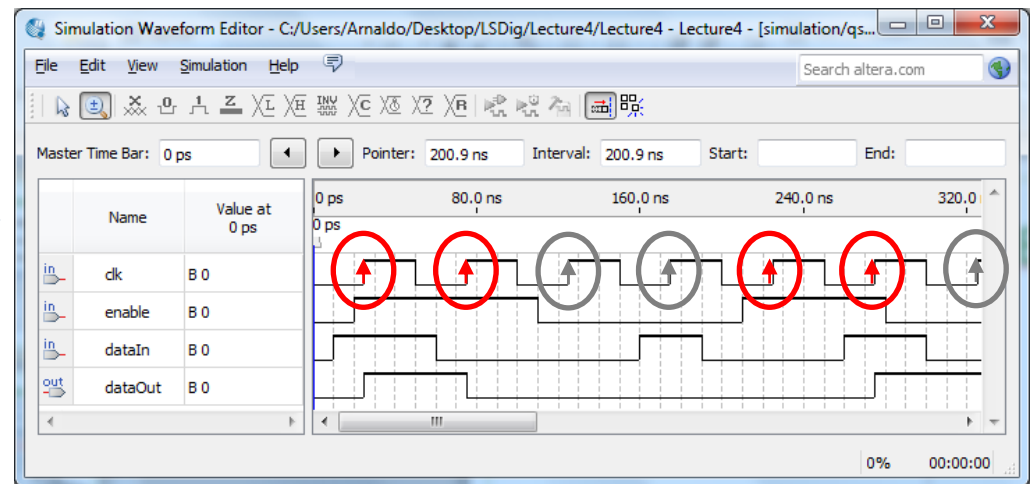
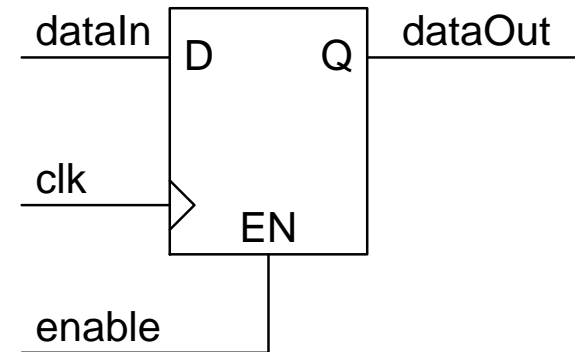


# FF tipo D com Enable

```
entity FFDEn is
    port(clk      : in  std_logic;
          enable   : in  std_logic;
          dataIn   : in  std_logic;
          dataOut  : out std_logic);
end FFDEn;
```

```
architecture Behav of FFDEn is
begin
```

```
    process(clk)
    begin
        if (rising_edge(clk)) then
            if (enable = '1') then
                dataOut <= dataIn;
            end if;
        end if;
    end process;
end Behav;
```



# FF tipo D com Enable e Reset

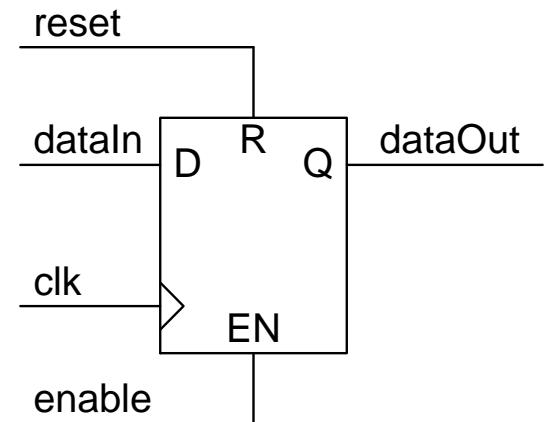
```
entity FFDEnRst is
  port(reset    : in  std_logic;
        clk     : in  std_logic;
        enable   : in  std_logic;
        dataIn   : in  std_logic;
        dataOut  : out std_logic);
end FFDEnRst;
```

```
architecture BehavRASyn of FFDEnRst is
begin
  process(reset, clk)
  begin
    if (reset = '1') then
      dataOut <= '0';
    elsif (rising_edge(clk)) then
      if (enable = '1') then
        dataOut <= dataIn;
      end if;
    end if;
  end process;
end BehavRASyn;
```

**Variante com  
Reset Assíncrono**

```
architecture BehavRSync of FFDEnRst is
begin
  process(clk)
  begin
    if (rising_edge(clk)) then
      if (reset = '1') then
        dataOut <= '0';
      elsif (enable = '1') then
        dataOut <= dataIn;
      end if;
    end if;
  end process;
end BehavRSync;
```

**Variante com  
Reset Síncrono**



# Registo de 8 bits

```
entity FFD8EnRst is
  port(reset    : in  std_logic;
        clk     : in  std_logic;
        enable  : in  std_logic;
        dataIn  : in  std_logic_vector(7 downto 0);
        dataOut : out std_logic_vector(7 downto 0));
end FFD8EnRst;
```

```
architecture Behav of FFD8EnRst is
begin
```

```
  process(clk)
```

```
  begin
```

```
    if (rising_edge(clk)) then
```

```
      if (reset = '1') then
```

```
        dataOut <= (others => '0');
```

```
      elsif (enable = '1') then
```

```
        dataOut <= dataIn;
```

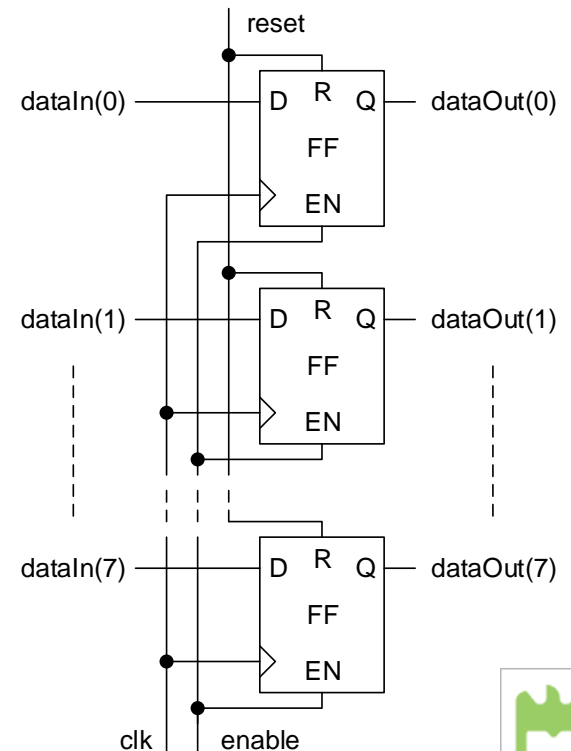
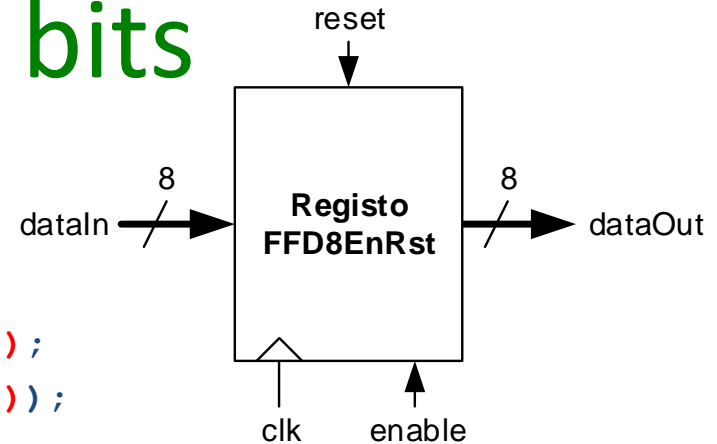
```
      end if;
```

```
    end if;
```

```
  end process;
```

```
end Behav;
```

**Exemplo com  
Reset Síncrono**





# Registo de Tamanho Parametrizável

```
entity FFDNEnRst is
```

```
  generic(N      : positive := 8);
```

```
  port(reset     : in  std_logic;
```

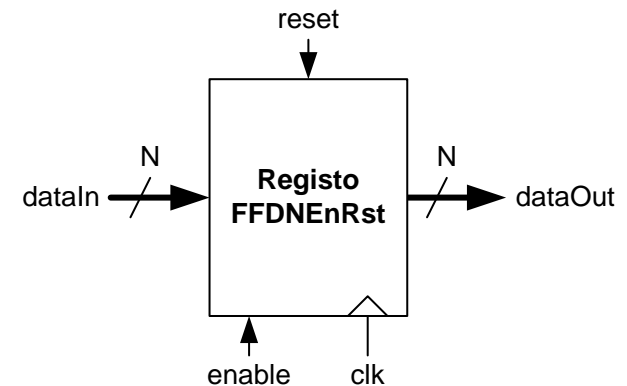
```
        clk      : in  std_logic;
```

```
        enable   : in  std_logic;
```

```
        dataIn   : in  std_logic_vector(N-1 downto 0);
```

```
        dataOut  : out std_logic_vector(N-1 downto 0));
```

```
end FFDNEnRst;
```



```
architecture Behav of FFDNEnRst is
begin
```

```
  process(clk)
```

```
  begin
```

```
    if (rising_edge(clk)) then
```

```
      if (reset = '1') then
```

```
        dataOut <= (others => '0');
```

```
      elsif (enable = '1') then
```

```
        dataOut <= dataIn;
```

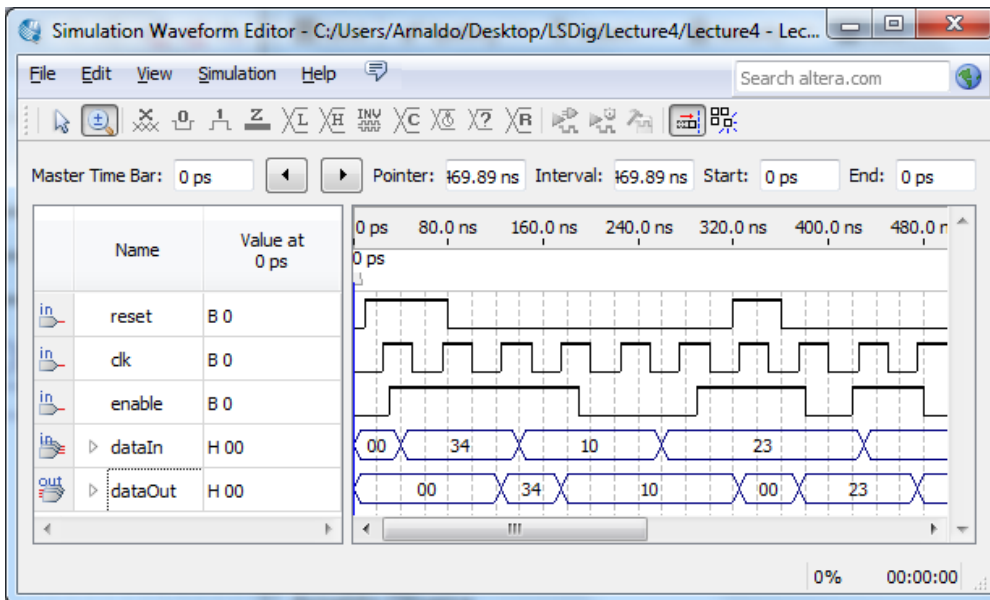
```
      end if;
```

```
    end if;
```

```
  end process;
```

```
end Behav;
```

**Exemplo com  
Reset Síncrono**



# Contador Binário Crescente de 8 bits com Enable e Reset Síncrono

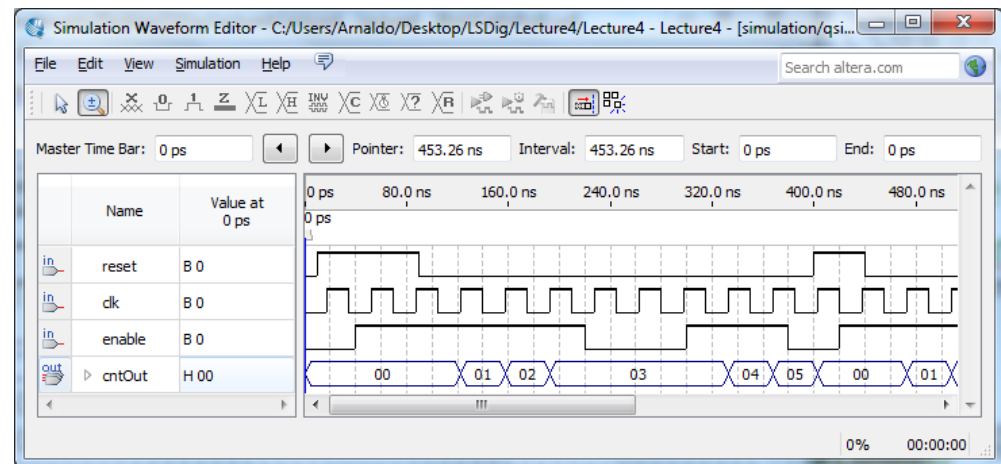
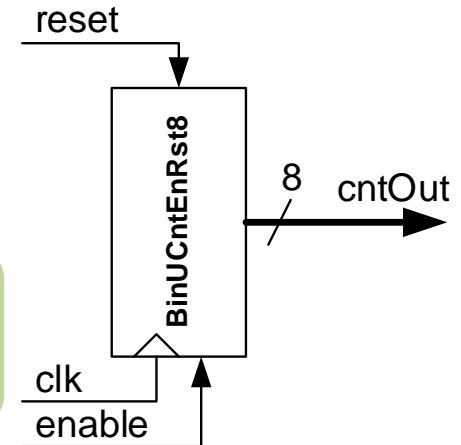
```
entity BinUCntEnRst8 is
  port(reset  : in  std_logic;
        clk   : in  std_logic;
        enable : in  std_logic;
        cntOut : out std_logic_vector(7 downto 0));
end BinUCntEnRst8;
```

```
architecture Behav of BinUCntEnRst8 is
  signal s_cntValue : unsigned(7 downto 0);
begin
```

```
  process (clk)
  begin
    if (rising_edge(clk)) then
      if (reset = '1') then
        s_cntValue <= (others => '0');
      elsif (enable = '1') then
        s_cntValue <= s_cntValue + 1;
      end if;
    end if;
  end process;
```

```
  cntOut <= std_logic_vector(s_cntValue);
end Behav;
```

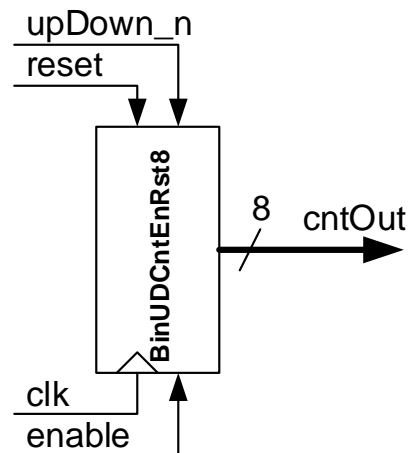
**Porque razão é  
necessário declarar  
o sinal s\_cntValue?**



# Contador Binário Crescente/Decrescente

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity BinUDCntEnRst8 is
  port(reset      : in  std_logic;
        clk       : in  std_logic;
        enable    : in  std_logic;
        upDown_n  : in  std_logic;
        cntOut    : out std_logic_vector(7 downto 0));
end BinUDCntEnRst8;
```



```
architecture Behav of BinUDCntEnRst8 is
  signal s_cntValue : unsigned(7 downto 0);
begin
  process(clk)
  begin
    if (rising_edge(clk)) then
      if (reset = '1') then
        s_cntValue <= (others => '0');
      elsif (enable = '1') then
        if (upDown_n = '1') then
          s_cntValue <= s_cntValue + 1;
        else
          s_cntValue <= s_cntValue - 1;
        end if;
      end if;
    end if;
  end process;

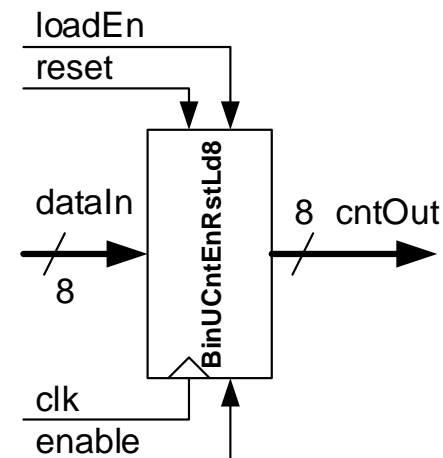
  cntOut <= std_logic_vector(s_cntValue);
end Behav ;
```

# Contador Binário com Entrada de Carregamento Paralelo

```
entity BinUCntEnRstLd8 is
  port(reset  : in  std_logic;
        clk   : in  std_logic;
        enable : in  std_logic;
        loadEn : in  std_logic;
        dataIn : in  std_logic_vector(7 downto 0);
        cntOut : out std_logic_vector(7 downto 0));
end BinUCntEnRstLd8;
```

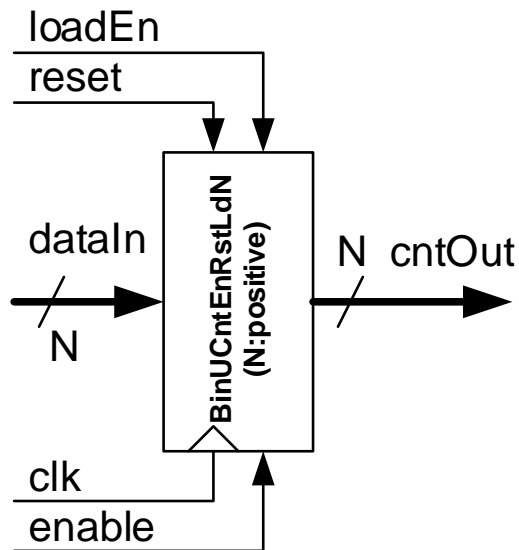
```
architecture Behav of BinUCntEnRstLd8 is
  signal s_cntValue : unsigned(7 downto 0);
begin
  process(clk)
  begin
    if (rising_edge(clk)) then
      if (reset = '1') then
        s_cntValue <= (others => '0');
      elsif (enable = '1') then
        if (loadEn = '1') then
          s_cntValue <= unsigned(dataIn);
        else
          s_cntValue <= s_cntValue + 1;
        end if;
      end if;
    end if;
  end process;
  cntOut <= std_logic_vector(s_cntValue);
end Behav ;
```

Sinais de controlo loadEn, upDown\_n e/ou outros podem ser combinados no mesmo contador de acordo com a prioridade relativa pretendida



# Parametrização do Número de Bits do Contador

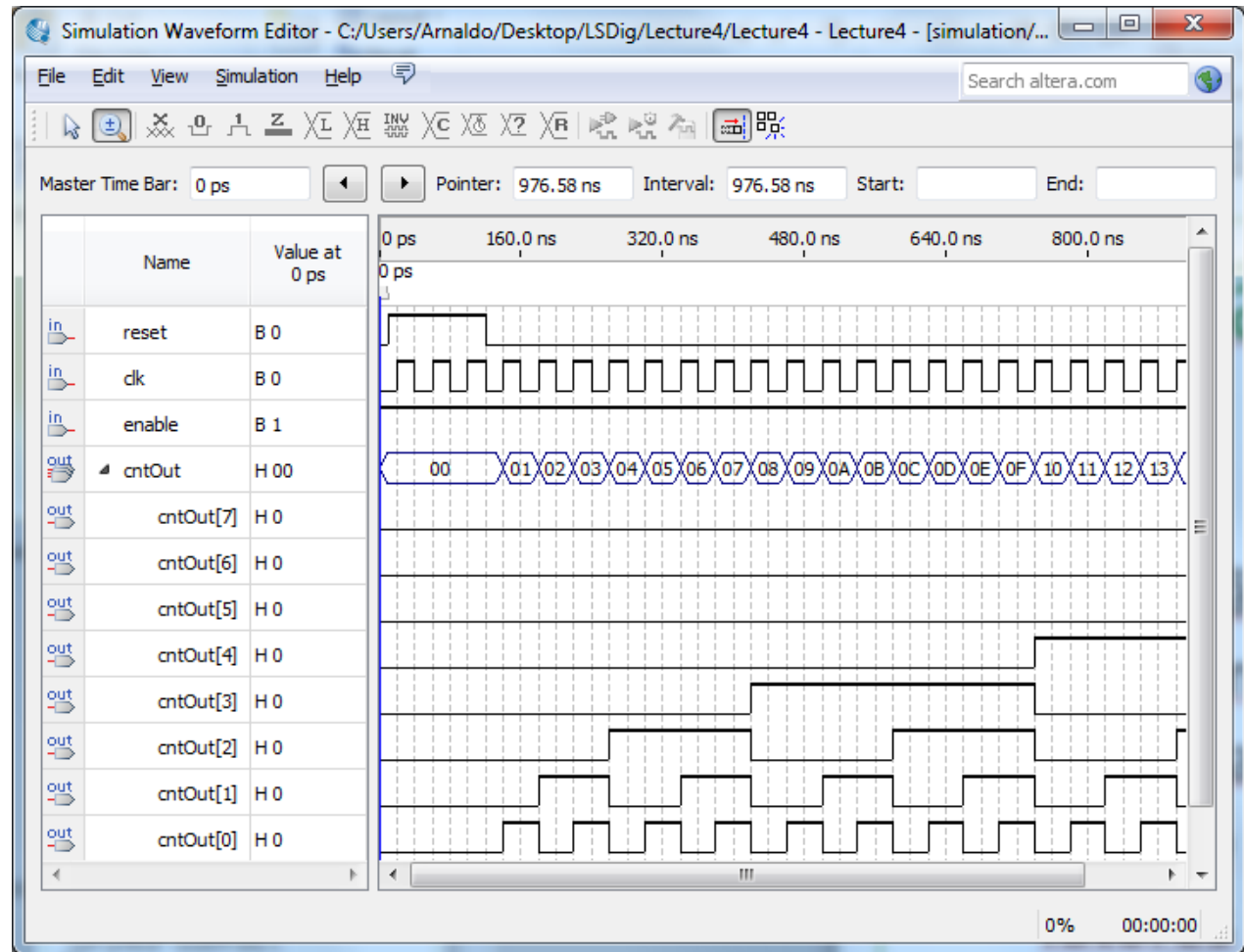
```
entity BinUCntEnRstLdN is
  generic(N      : positive := 8);
  port(reset    : in  std_logic;
        clk     : in  std_logic;
        enable   : in  std_logic;
        loadEn   : in  std_logic;
        dataIn   : in  std_logic_vector((N-1) downto 0);
        cntOut   : out std_logic_vector((N-1) downto 0));
end BinUCntEnRstLdN;
```



```
architecture Behav of BinUCntEnRstLdN is
  signal s_cntValue : unsigned((N-1) downto 0);
begin
  process(clk)
  begin
    if (rising_edge(clk)) then
      if (reset = '1') then
        s_cntValue <= (others => '0');
      elsif (enable = '1') then
        if (loadEn = '1') then
          s_cntValue <= unsigned(dataIn);
        else
          s_cntValue <= s_cntValue + 1;
        end if;
      end if;
    end if;
  end process;
  cntOut <= std_logic_vector(s_cntValue);
end Behav ;
```

# Divisão da Frequência de um Sinal de Relógio (*clock*) por $2^N$

- A divisão da frequência de um sinal de relógio por fatores inteiros “potência de base 2” pode ser efetuada por um contador



# Divisor (simples) de Frequência

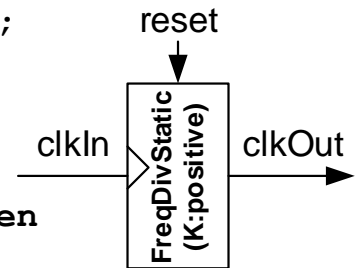
- A divisão da frequência de um sinal de relógio por **fatores inteiros arbitrários (K)** requer hardware “mais elaborado” (baseado num contador de modulo **K**)
- Exemplo de um módulo divisor de frequência **configurável estaticamente** (**K** fixado em *compile time*, aquando da instanciação com **generic map**)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity FreqDivStatic is
    generic(K      : positive := 4);
    port(reset     : in  std_logic;
          clkIn    : in  std_logic;
          clkOut   : out std_logic);
end FreqDivStatic;
```

architecture Behavioral of FreqDivStatic is

```
    signal s_counter : natural;
begin
    process(clkIn)
    begin
        if rising_edge(clkIn) then
            if ((reset = '1') or
                (s_counter = K - 1)) then
                clkOut    <= '0';
                s_counter <= 0;
            else
                if (s_counter = K/2 - 1) then
                    clkOut    <= '1';
                end if;
                s_counter <= s_counter + 1;
            end if;
        end if;
    end process;
end Behavioral;
```



**Contador free running de módulo K**  
clkOut <= '1' a “meio” da contagem  
clkOut <= '0' no final da contagem

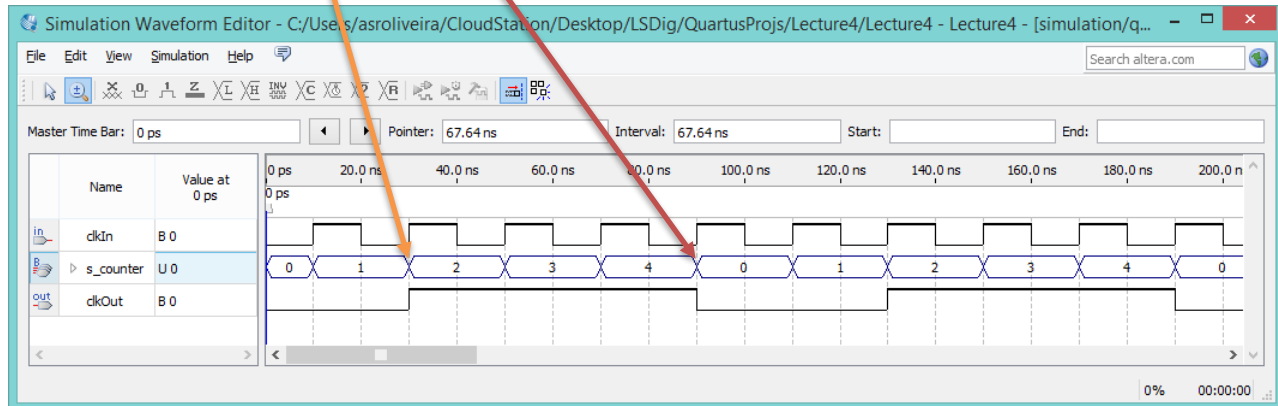
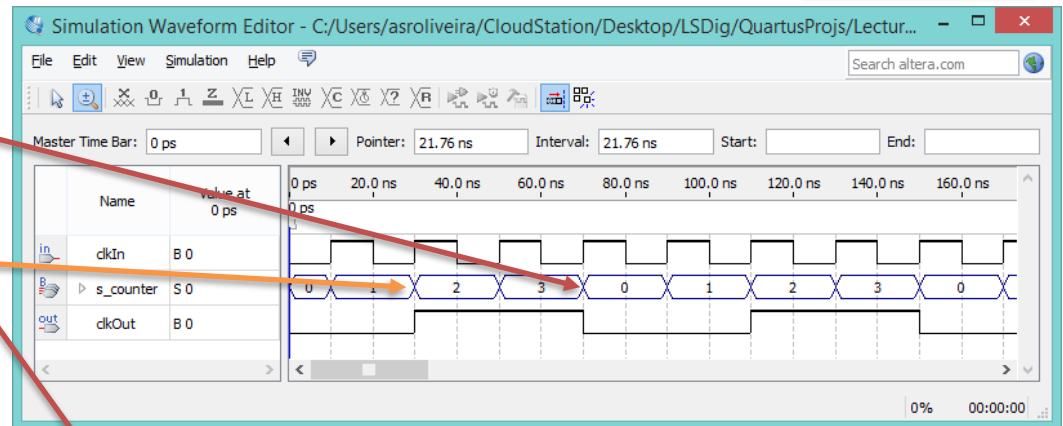
# Simulação do Div. (Simples) de Freq.

```
process (clkIn)
begin
    if rising_edge(clkIn) then
        if ((reset = '1') or
            (s_counter = K - 1)) then
            clkOut    <= '0';
            s_counter <= 0;
        else
            if (s_counter = K/2 - 1) then
                clkOut    <= '1';
            end if;
            s_counter <= s_counter + 1;
        end if;
    end if;
end process;
```

Vamos assumir  
que  $f_{clkIn} = 50 \text{ MHz}$

**K=4**

$f_{\text{clkOut}}$  ?  
Duty cycle?



**K=5**  
f<sub>clkOut</sub> ?  
*Duty cycle?*

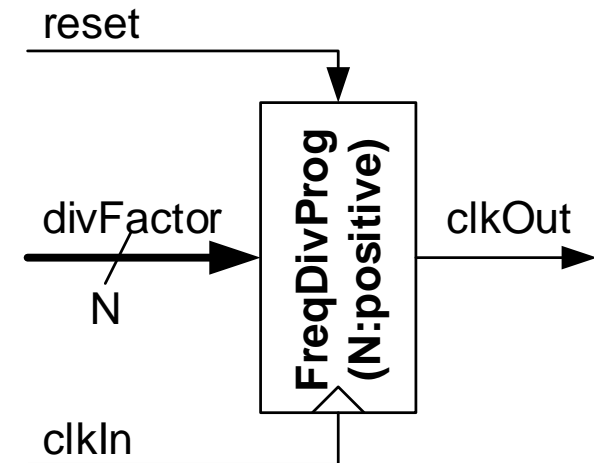


# Divisor de Frequência Programável Dinamicamente (Entidade)

- Exemplo de um módulo divisor de frequência **programável / configurável dinamicamente** (em *runtime* através do porto **divFactor**)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity FreqDivProg is
    generic(N : positive := 3);
    port(reset      : in  std_logic;
          divFactor  : in  std_logic_vector(N-1 downto 0);
          clkIn      : in  std_logic;
          clkOut     : out std_logic);
end FreqDivProg;
```



O número de bits “N” do fator de divisão é **fixado estaticamente**

O valor do **fator de divisão** é definido **dinamicamente**

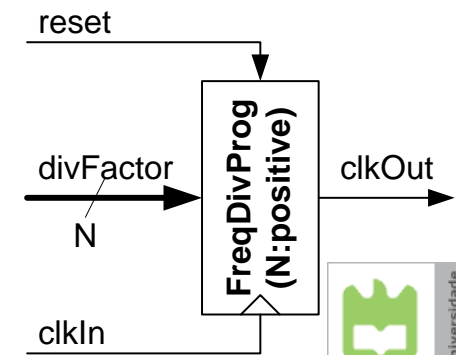
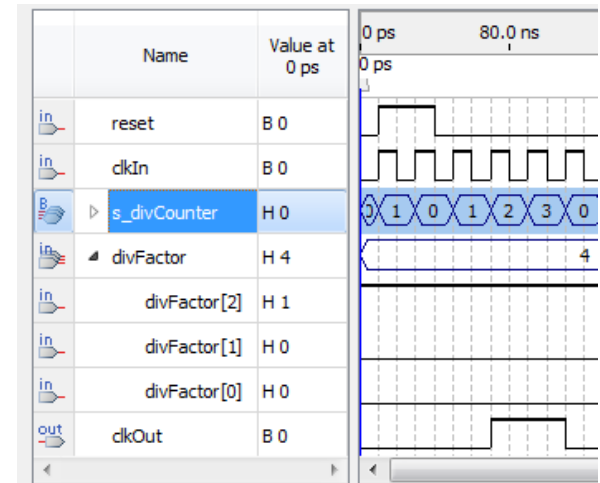
# Div. de Freq. Programável Dinamicamente (Arquitetura)

```

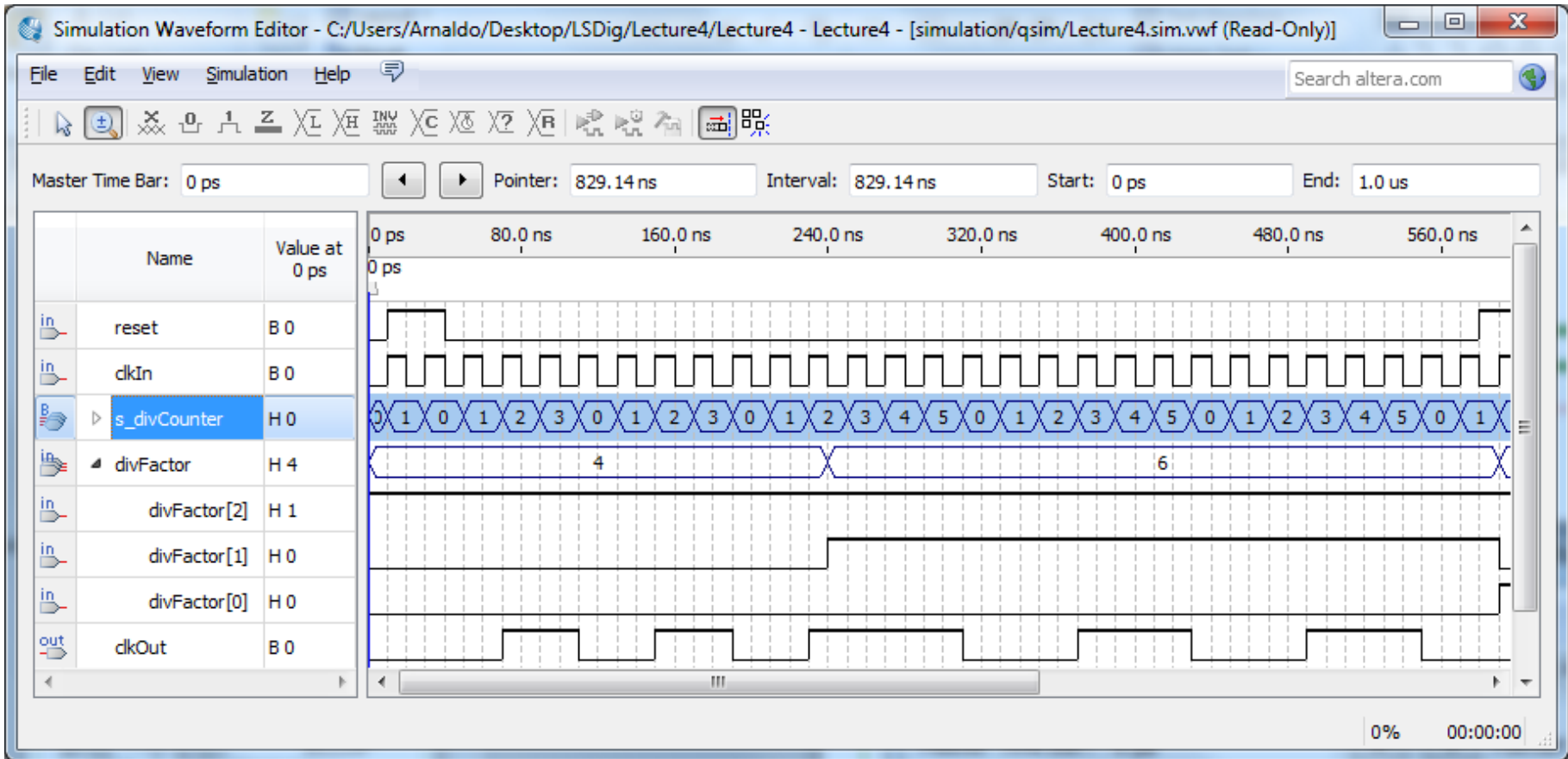
architecture Behavioral of FreqDivProg is
    signal s_divCounter : unsigned(N-1 downto 0);
begin
    process(clkIn)
    begin
        if (rising_edge(clkIn)) then
            if ((reset = '1') or
                (s_divCounter >= unsigned(divFactor) - 1)) then
                clkOut      <= '0';
                s_divCounter <= (others => '0');
            else
                if (s_divCounter = (unsigned(divFactor)/2 - 1)) then
                    clkOut <= '1';
                end if;
                s_divCounter <= s_divCounter + 1;
            end if;
        end if;
    end process;
end Behavioral;

```

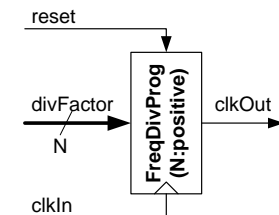
Descrição semelhante ao **FreqDivStatic**, mas em que o fator de divisão é programável em *run-time*



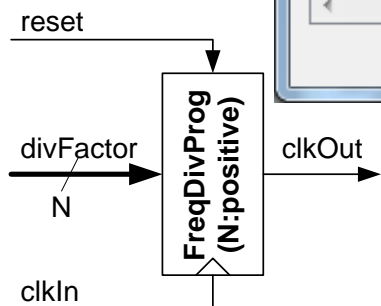
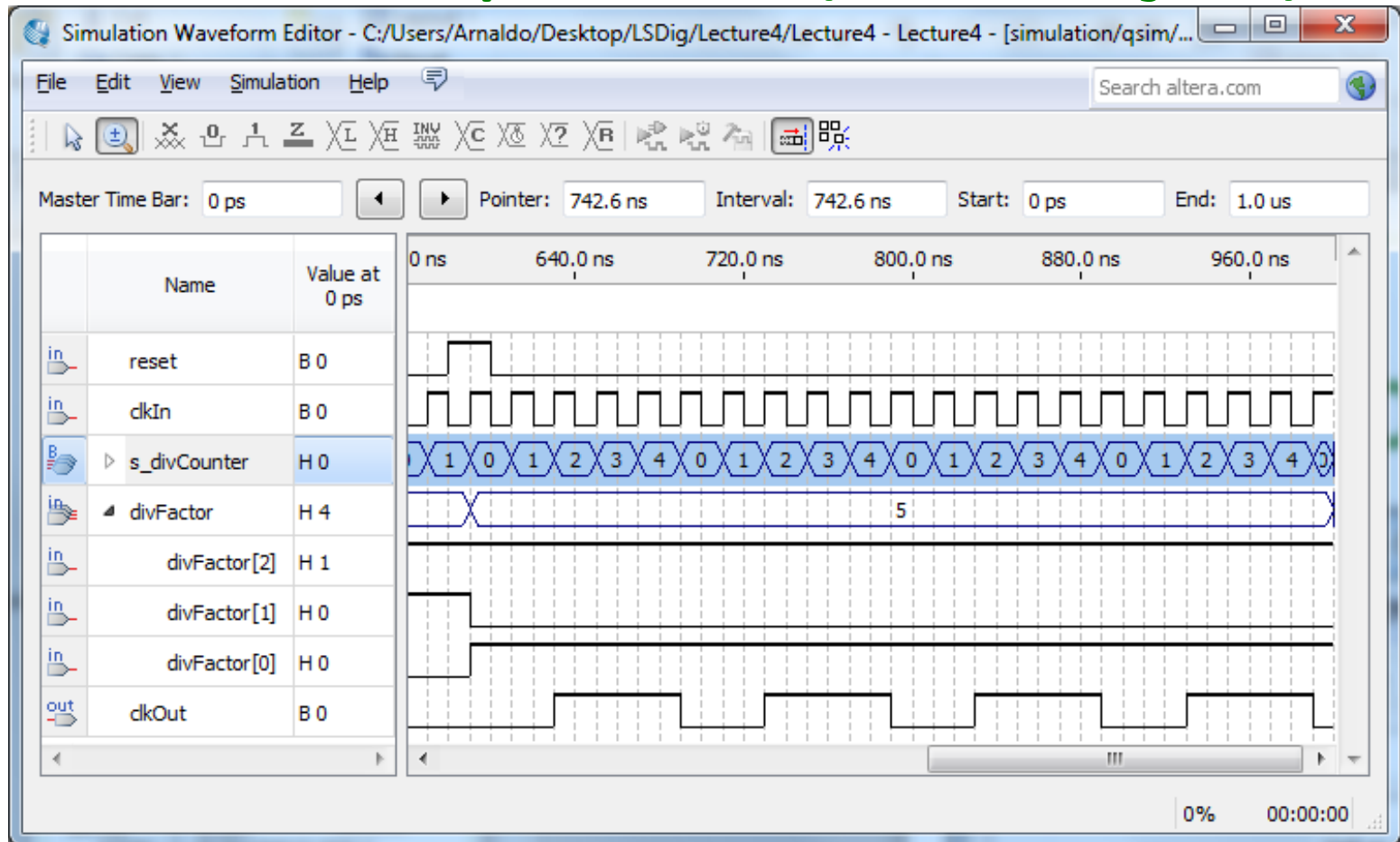
# Divisor de Frequência (Simulação)



divFactor "par" → Duty cycle = 50%



# Divisor de Frequência (Simulação)

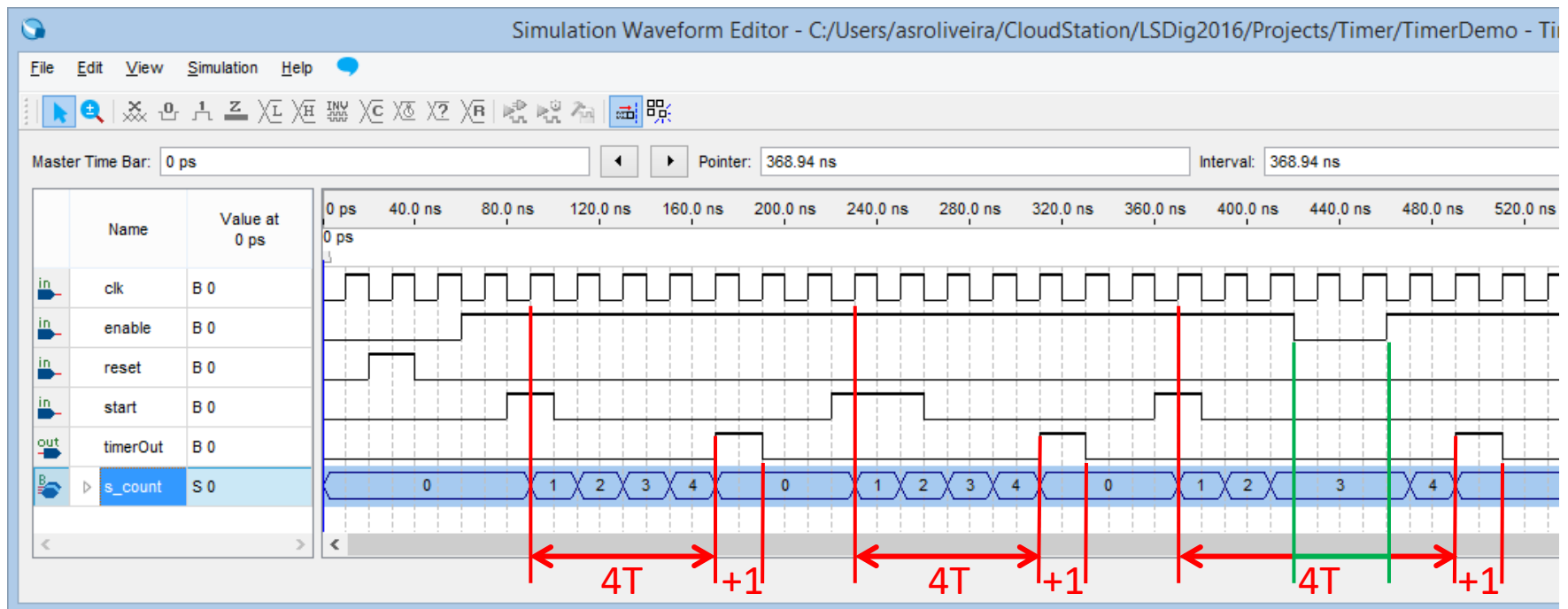


divFactor “ímpar” → Duty cycle  $\neq$  50%

# Temporizador

## (Exemplo de Comportamento e Simulação)

- Um temporizador é um módulo usado para medir tempo, ou para gerar um evento após ter decorrido um dado intervalo de tempo (depois do temporizador ter sido iniciado/disparado)
- Exemplo em que a saída (“timerOut”) é ativada durante  $1T$ , após um intervalo de tempo predefinido ( $4T$ ) depois do disparo da entrada (“start”)



$T$  = período do sinal de relógio

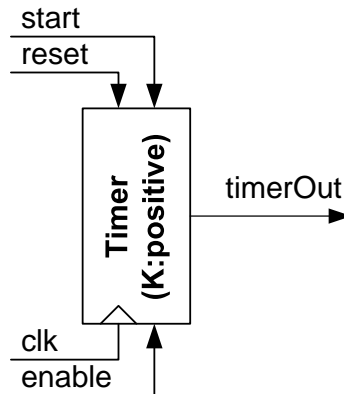


# Temporizador

## (Exemplo de Implementação em VHDL)

```
library ieee;
use ieee.std_logic_1164.all;

entity TimerOnDelay is
    generic(K : positive := 5);
    port(clk      : in  std_logic;
         reset    : in  std_logic;
         enable   : in  std_logic;
         start    : in  std_logic;
         timerOut : out std_logic);
end TimerOnDelay;
```



**TPC:** Desenvolver um novo temporizador em que a saída é ativada após o disparo do temporizador e desativada após ter decorrido o intervalo de tempo KT (com K programável dinamicamente)

```
architecture Behavioral of TimerOnDelay is
```

```
    signal s_count : integer := 0;
```

```
begin
```

```
    assert(K >= 2);
```

**K deve ser ≥ 2. Porquê?**

```
    process(clk)
```

```
    begin
```

```
        if (rising_edge(clk)) then
```

```
            if (reset = '1') then
```

```
                timerOut <= '0';
```

```
                s_count <= 0;
```

**Inicialização**

```
            elsif (enable = '1') then
```

**Teste do sinal "enable"**

**Se contador parado**

```
                if (s_count = 0) then
```

```
                    if (start = '1') then
```

```
                        s_count <= s_count + 1;
```

```
                    end if;
```

**Deteção de um novo disparo**

**Desativação da saída**

```
                        timerOut <= '0';
```

```
                else
```

**Deteção do final de contagem e ativação da saída**

```
                    if (s_count = (K - 1)) then
```

```
                        timerOut <= '1';
```

```
                        s_count <= 0;
```

```
                    else
```

**Incremento do contador "a meio" da contagem**

```
                        timerOut <= '0';
```

```
                        s_count <= s_count + 1;
```

```
                    end if;
```

```
                end if;
```

```
            end if;
```

```
        end if;
```

```
    end process;
```

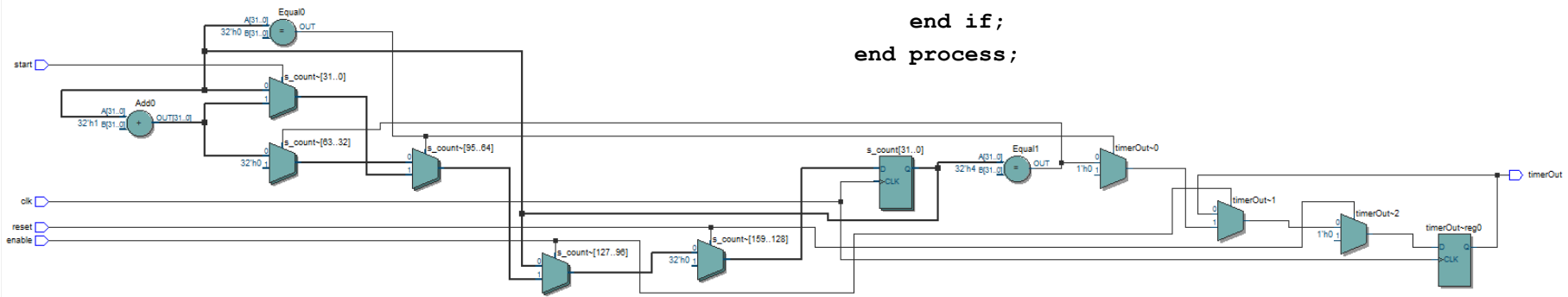
```
end Behavioral;
```

# Temporizador Sintetizado

- TPC

– Análise do circuito resultante da síntese do modelo em VHDL

```
process (clk)
begin
    if (rising_edge(clk)) then
        if (reset = '1') then
            timerOut <= '0';
            s_count <= 0;
        elsif (enable = '1') then
            if (s_count = 0) then
                if (start = '1') then
                    s_count <= s_count + 1;
                end if;
                timerOut <= '0';
            else
                if (s_count = (K - 1)) then
                    timerOut <= '1';
                    s_count <= 0;
                else
                    timerOut <= '0';
                    s_count <= s_count + 1;
                end if;
            end if;
        end if;
    end if;
end process;
```



# Estrutura Geral de um Processo VHDL de um Temporizador

Tipos de temporizadores:

- **Atraso “à operação”**
  - Saída ativada após decorrido um tempo predefinido
- **Atraso “à desoperação”**
  - Saída desativada após decorrido um tempo predefinido

```
process (clk)
```

```
begin
```

Em cada flanco ativo do sinal de relógio

Se o sinal de reset estiver ativo

Desativa saída

Coloca contador a 0

Senão, se o sinal de enable estiver ativo

Se o temporizador estiver parado (contador = 0)

Se a entrada start estiver ativa

Desativa a saída / Ativa a saída

Incrementa o contador

Senão

Desativa a saída

Senão (contador /= 0)

Se o contador tiver atingido o valor máximo (K-1)

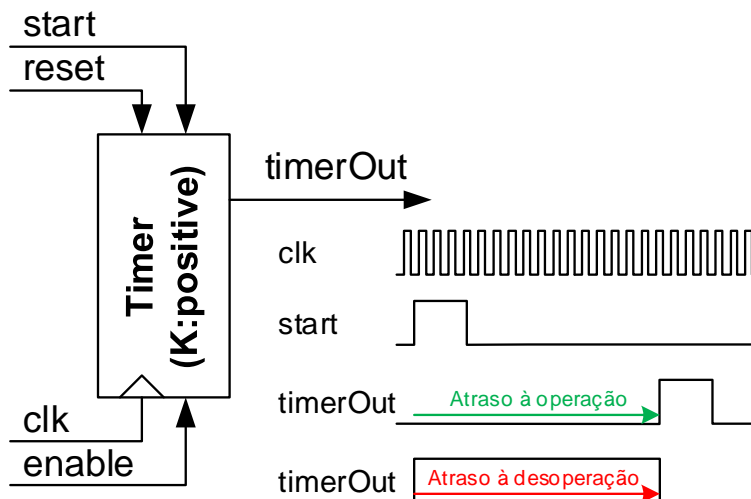
Ativa a saída / Desativa a saída

Coloca contador a 0

Senão

Desativa a saída / Ativa a saída

Incrementa o contador

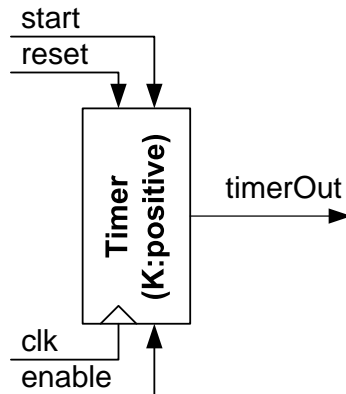




# Temporizador (Atraso à desoperação)

```
library ieee;
use ieee.std_logic_1164.all;

entity TimerOnDelay is
    generic(K : positive := 5);
    port(clk      : in  std_logic;
         reset    : in  std_logic;
         enable   : in  std_logic;
         start    : in  std_logic;
         timerOut : out std_logic);
end TimerOnDelay;
```



**TPC:** Desenvolver um novo temporizador em que a saída é ativada após o disparo do temporizador e desativada após ter decorrido o intervalo de tempo KT (com K programável dinamicamente)

```
architecture Behavioral of TimerOnDelay is
    signal s_count : integer := 0;
begin
```

**assert(K >= 2);** **K deve ser ≥ 2. Porquê?**

```
process(clk)
```

```
begin
```

```
    if (rising_edge(clk)) then
```

**Inicialização**

```
        if (reset = '1') then
```

```
            timerOut <= '0';
```

```
            s_count <= 0;
```

**Teste do sinal “enable”**

```
        elsif (enable = '1') then
```

**Se contador parado**

```
            if (s_count = 0) then
```

```
                if (start = '1') then
```

```
                    s_count <= s_count + 1;
```

```
                    timerOut <= '1';
```

**Deteção de um novo disparo**

**Ativação da saída**

```
                end if;
```

```
            else
```

**Deteção do final de contagem  
e desativação da saída**

```
                if (s_count = (K - 1)) then
```

```
                    timerOut <= '0';
```

```
                    s_count <= 0;
```

```
                else
```

**Incremento do contador  
“a meio” da contagem**

```
                    timerOut <= '1';
```

```
                    s_count <= s_count + 1;
```

```
                end if;
```

```
            end if;
```

```
        end if;
```

```
    end if;
```

```
end process;
```

```
end Behavioral;
```

# Comentários Finais

- Todos os modelos apresentados podem ser usados da forma fornecida (módulo autónomo reutilizável com *Entity + Architecture*), ou, alternativamente, o processo que descreve a funcionalidade pode também ser integrado em módulos (arquiteturas) mais complexos com outros processos, instanciações de componentes e/ou atribuições concorrentes
- No final desta aula e do trabalho prático 4 de LSD, deverá ser capaz de:
  - Modelar componentes sequenciais fundamentais em VHDL
    - Registos
    - Contadores
    - Divisores de frequência
    - Temporizadores
- No final do trabalho prático 5 de LSD, deverá ser capaz de:
  - Modelar componentes parametrizáveis combinatórios e sequenciais em VHDL
- ... bons trabalhos práticos 4 e 5, disponíveis no site da UC ☺
  - [elearning.ua.pt](http://elearning.ua.pt)