

Simulação e TestBenchs

Engenharia Eletrônica

Prof. Renan Augusto Starke

Instituto Federal de Santa Catarina – IFSC
Campus Florianópolis
renan.starke@ifsc.edu.br

13 de julho de 2016



INSTITUTO FEDERAL
SANTA CATARINA

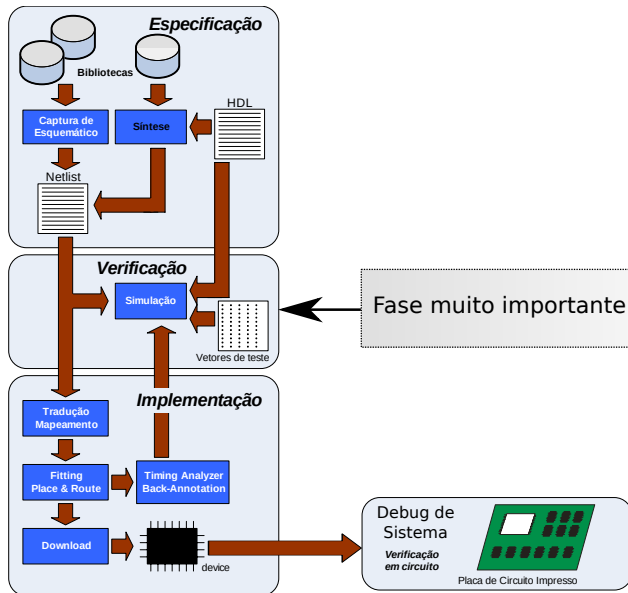
Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
INSTITUTO FEDERAL DE SANTA CATARINA

- 1 Introdução
- 2 Simulação de circuitos: testbenchs
- 3 Exemplo: divisor de clock
- 4 ModelSim
- 5 Exercícios

► Tópicos da aula de hoje:

- Continuação do estudo de VHDL
- Simulação: ModelSim
- Criação de *TestBenches*

Fluxo de projeto



- 1 Introdução
- 2 Simulação de circuitos: testbenchs**
- 3 Exemplo: divisor de clock
- 4 ModelSim
- 5 Exercícios

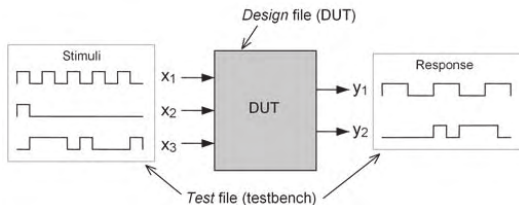
Síntese

Síntese é o processo de tradução de um código VHDL em um conjunto de estruturas de hardware que implementam as funcionalidades descritas.

Simulação

Simulação é o procedimento de teste usado para verificar se o circuito sintetizado de fato implementa o comportamento pretendido.

Procedimento geral



- ▶ Projeto sobre teste: DUT – *design under test*
- ▶ Estímulos aplicado ao DUT
- ▶ Resposta do circuito (forma de onda, texto de saída, erros, etc)

Testbench

Testbench é composto pela geração de estímulos de entrada e verificação da saída do circuito.

- ▶ Testbenches corretos e eficazes aumentam a produtividade
- ▶ Garante-se que o circuito está funcionando dentro do esperado antes da síntese
- ▶ Algumas ferramentas de simulação consideram comportamento real do hardware (atrasos, frequência máxima de operação, etc.)

► Tipo I:

- Não considera tempos de propagação
- Inspeção manual da saída
- Tipo mais simples

► Tipo II:

- Tempos de propagação do DUT são considerados
- Inspeção manual da saída
- Conhecido como: *simulação temporal manual*

► Tipo III:

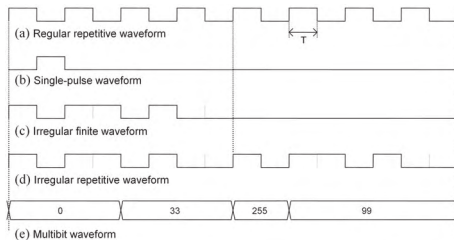
- Não considera tempos de propagação
- Saída é automaticamente verificada pelo simulador
- Conhecido como: *Simulação funcional automática*

► Tipo IV:

- Tempos de propagação do DUT são considerados
- Saída é automaticamente verificada pelo simulador
- Tipo mais complexo de código de simulação
- Conhecido como: *Simulação temporal automática ou full bench*

Geração de estímulos

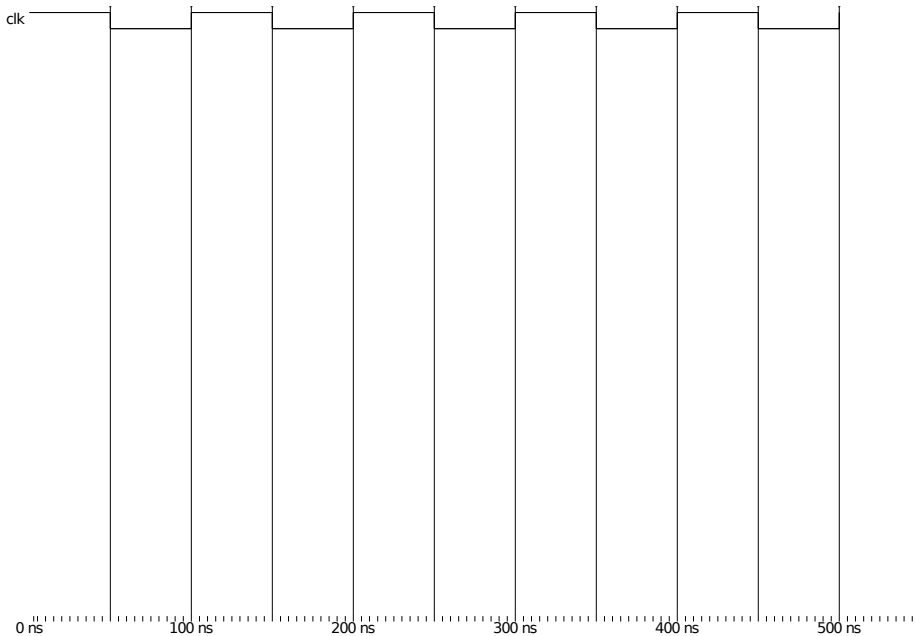
Exemplos de geração de estímulos:



- a) Forma de onda regular repetitiva: *clocks*
- b) Forma de onda com pulso único: *resets*
- c) Forma de onda irregular finita
- d) Forma de onda irregular repetitiva
- e) Forma de onda multibit

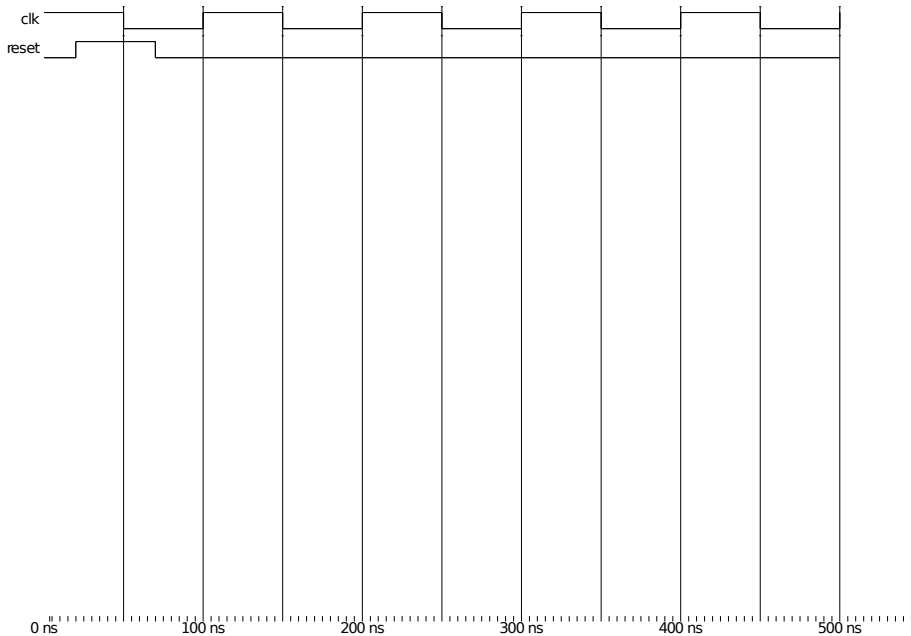
Geração de estímulos: clocks

```
signal clk : std_logic := '1';  
  
-- gera uma forma de onda repetitiva e regular: clocks  
process  
begin  
    wait for 50 ns;  
    clk <= '0';  
    wait for 50 ns;  
    clk <= '1';  
end process;
```



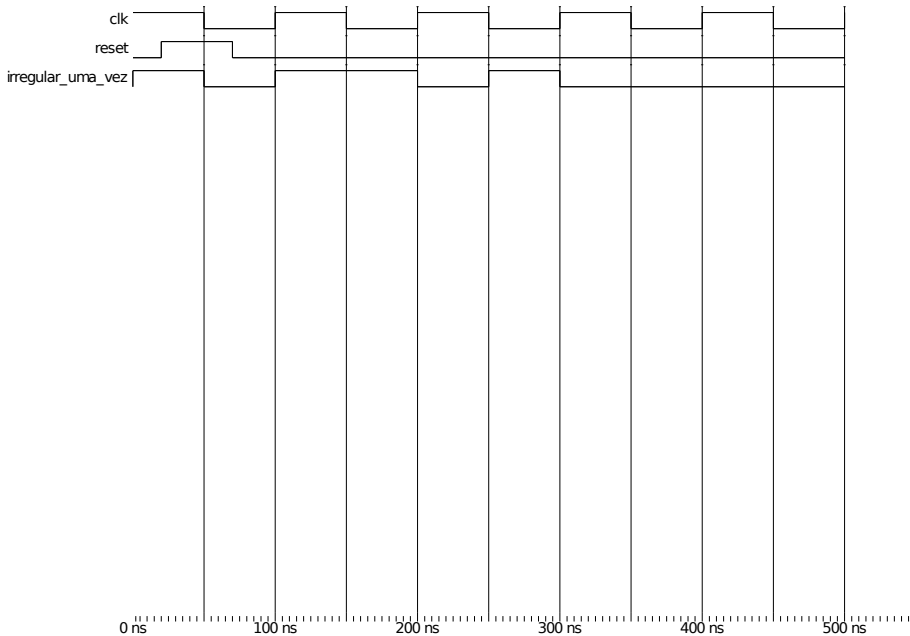
Geração de estímulos: reset

```
signal rst : std_logic := '0';  
  
--gera uma forma de onda de um pulso  
process  
begin  
    wait for 20 ns;  
    rst <= '1';  
    wait for 50 ns;  
    rst <= '0';  
    wait;  
end process;
```



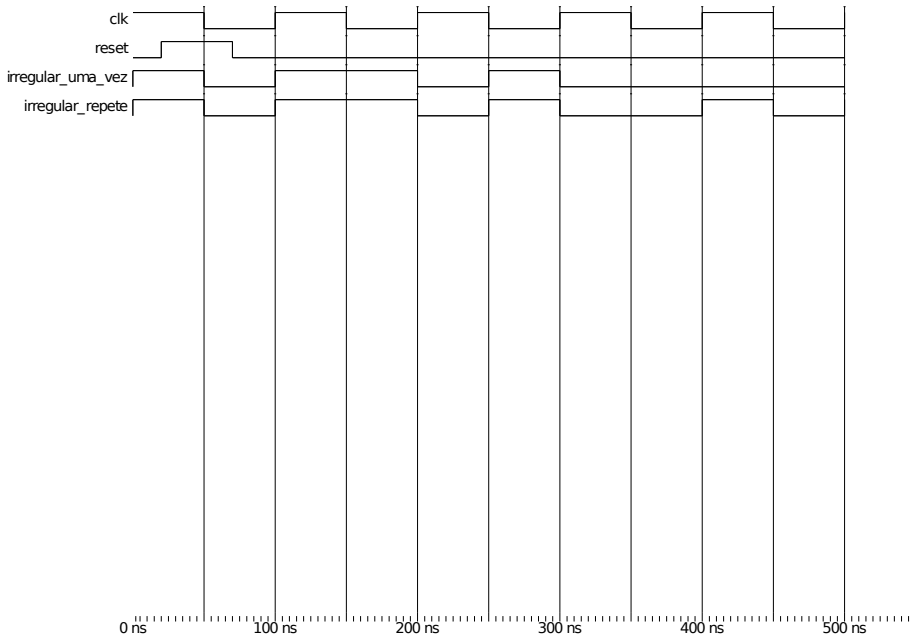
Geração de estímulos: não repetitiva irregular

```
signal x : std_logic := '0';  
  
-- gera uma forma de onda não repetitiva irregular  
process  
  constant wave: std_logic_vector(1 to 8) := "10110100";  
begin  
  for i in wave'range loop  
    x <= wave(i);  
    wait for 50 ns;  
  end loop;  
  wait;  
end process;
```

Geração de estímulos: repetitiva irregular

```
signal y : std_logic := '0';  
  
-- gera uma forma de onda repetitiva irregular  
process  
  constant wave: std_logic_vector(1 to 8) := "10110100";  
begin  
  for i in wave'range loop  
    y <= wave(i);  
    wait for 50 ns;  
  end loop;  
end process;
```



Geração de estímulos: multibit

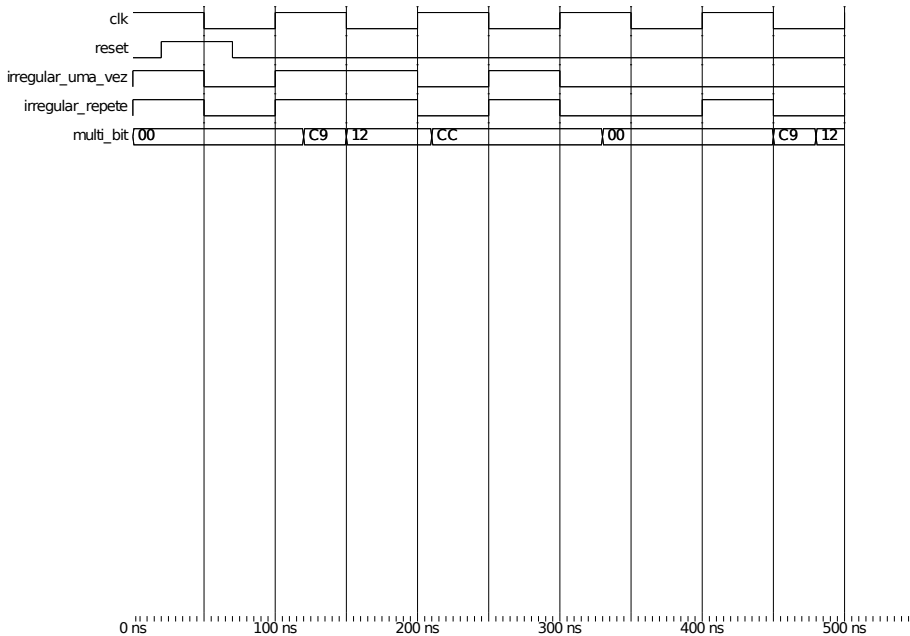
```
signal z : std_logic_vector(7 downto 0);

-- gera uma forma de onda multibit
process
begin
    z <= (z'range => '0');
    wait for 120 ns;

    z <= "11001001";
    wait for 30 ns;

    z <= x"12";
    wait for 60 ns;

    z <= x"CC";
    wait for 120 ns;
end process;
```



Geração de estímulos

```
-- Bibliotecas e cláusulas
LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.ALL;

-----

ENTITY testbench IS
END ENTITY testbench;

-----

ARCHITECTURE stimulus OF testbench IS

    -- declaração de sinais
    signal clk : std_logic := '1';
    signal rst : std_logic := '0';
    signal x   : std_logic := '0';
    signal y   : std_logic := '0';
    signal z   : std_logic_vector(7 downto 0);

BEGIN -- início do corpo da arquitetura

    -- gera uma forma de onda repetitiva e regular: clocks
    process
    begin
        wait for 50 ns;
        clk <= '0';
        wait for 50 ns;
        clk <= '1';
    end process;
```

Geração de estímulos

```
-- Bibliotecas e cláusulas
LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.ALL;

-----
ENTITY testbench IS
END ENTITY testbench;
-----

ARCHITECTURE stimulus OF testbench IS

    -- declaração de sinais
    signal clk : std_logic := '1';
    signal rst : std_logic := '0';
    signal x   : std_logic := '0';
    signal y   : std_logic := '0';
    signal z   : std_logic_vector(7 downto 0);

BEGIN -- início do corpo da arquitetura

    -- gera uma forma de onda repetitiva e regular: clocks
    process
    begin
        wait for 50 ns;
        clk <= '0';
        wait for 50 ns;
        clk <= '1';
    end process;
```

Geração de estímulos

```
-- Bibliotecas e cláusulas
LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.ALL;

-----
ENTITY testbench IS
END ENTITY testbench;
-----

ARCHITECTURE stimulus OF testbench IS

    -- declaração de sinais
    signal clk : std_logic := '1';
    signal rst : std_logic := '0';
    signal x   : std_logic := '0';
    signal y   : std_logic := '0';
    signal z   : std_logic_vector(7 downto 0);

BEGIN    -- início do corpo da arquitetura

    -- gera uma forma de onda repetitiva e regular: clocks
    process
    begin
        wait for 50 ns;
        clk <= '0';
        wait for 50 ns;
        clk <= '1';
    end process;
```


Geração de estímulos

```
-- Bibliotecas e cláusulas
LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.ALL;
-----

ENTITY testbench IS
END ENTITY testbench;
-----

ARCHITECTURE stimulus OF testbench IS

    -- declaração de sinais
    signal clk : std_logic := '1';
    signal rst : std_logic := '0';
    signal x   : std_logic := '0';
    signal y   : std_logic := '0';
    signal z   : std_logic_vector(7 downto 0);

BEGIN -- início do corpo da arquitetura

    -- gera uma forma de onda repetitiva e regular: clocks
    process
    begin
        wait for 50 ns;
        clk <= '0';
        wait for 50 ns;
        clk <= '1';
    end process;
```

Geração de estímulos

```
--gera uma forma de onda de um pulso
process
begin
    wait for 20 ns;
    rst <= '1';
    wait for 50 ns;
    rst <= '0';
    wait;
end process;

-- gera uma forma de onda não repetitiva irregular
process
    constant wave: std_logic_vector(1 to 8) := "10110100";
begin
    for i in wave'range loop
        x <= wave(i);
        wait for 50 ns;
    end loop;
    wait;
end process;

-- gera uma forma de onda repetitiva irregular
process
    constant wave: std_logic_vector(1 to 8) := "10110100";
begin
    for i in wave'range loop
        y <= wave(i);
        wait for 50 ns;
    end loop;
end process;
```

Geração de estímulos

```
--gera uma forma de onda de um pulso
process
begin
    wait for 20 ns;
    rst <= '1';
    wait for 50 ns;
    rst <= '0';
    wait;
end process;

-- gera uma forma de onda não repetitiva irregular
process
    constant wave: std_logic_vector(1 to 8) := "10110100";
begin
    for i in wave'range loop
        x <= wave(i);
        wait for 50 ns;
    end loop;
    wait;
end process;

-- gera uma forma de onda repetitiva irregular
process
    constant wave: std_logic_vector(1 to 8) := "10110100";
begin
    for i in wave'range loop
        y <= wave(i);
        wait for 50 ns;
    end loop;
end process;
```

Geração de estímulos

```
--gera uma forma de onda de um pulso
process
begin
    wait for 20 ns;
    rst <= '1';
    wait for 50 ns;
    rst <= '0';
    wait;
end process;

-- gera uma forma de onda não repetitiva irregular
process
    constant wave: std_logic_vector(1 to 8) := "10110100";
begin
    for i in wave'range loop
        x <= wave(i);
        wait for 50 ns;
    end loop;
    wait;
end process;

-- gera uma forma de onda repetitiva irregular
process
    constant wave: std_logic_vector(1 to 8) := "10110100";
begin
    for i in wave'range loop
        y <= wave(i);
        wait for 50 ns;
    end loop;
end process;
```

Geração de estímulos

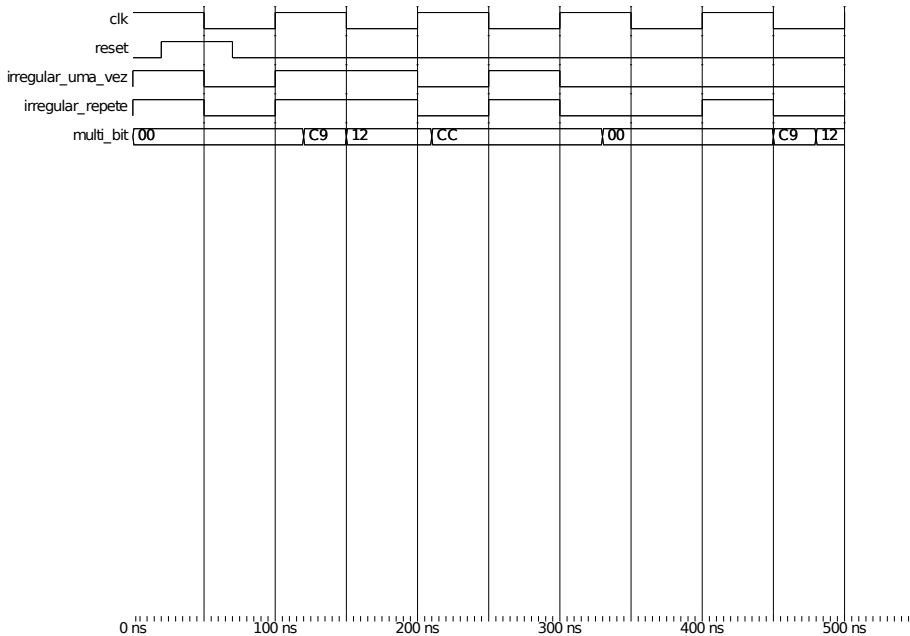
```
-- gera uma forma de onda multibit
process
begin
    z <= (z'range => '0');
    wait for 120 ns;

    z <= "11001001";
    wait for 30 ns;

    z <= x"12";
    wait for 60 ns;

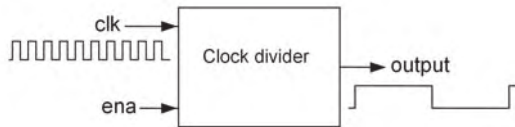
    z <= x"CC";
    wait for 120 ns;
end process;

END ARCHITECTURE stimulus;
```



- 1 Introdução
- 2 Simulação de circuitos: testbenchs
- 3 Exemplo: divisor de clock**
- 4 ModelSim
- 5 Exercícios

Exemplo: divisor de clock



- ▶ Recebe um sinal de clock em *clk*
- ▶ Divide esta frequência por 10
- ▶ Ativado quando *ena* estiver em nível lógico alto

Exemplo: divisor de clock

```
-- Bibliotecas e cláusulas
LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.ALL;

-- Entidade e portas
ENTITY divisor_clock IS
    PORT (clk, ena: IN std_logic;
          output: OUT std_logic);
END ENTITY;

-- Arquitetura
ARCHITECTURE rtl OF divisor_clock IS
    constant max: NATURAL := 5;
BEGIN
    p0: process(clk)
        variable count: natural range 0 to max := 0;
        variable temp: std_logic := '0';
    begin
        if (rising_edge(clk)) then
            if ena = '1' then
                count := count + 1;
                if (count = max) then
                    temp := not temp;
                    count := 0;
                end if;
            end if;
        end if;
        output <= temp;
    end process;
END ARCHITECTURE;
```

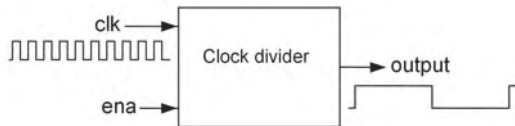
Exemplo: divisor de clock

```
-- Bibliotecas e cláusulas
LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.ALL;

-- Entidade e portas
ENTITY divisor_clock IS
    PORT (clk, ena: IN std_logic;
          output: OUT std_logic);
END ENTITY;

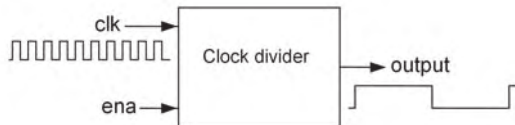
-- Arquitetura
ARCHITECTURE rtl OF divisor_clock IS
    constant max: NATURAL := 5;
BEGIN
    p0: process(clk)
        variable count: natural range 0 to max := 0;
        variable temp: std_logic := '0';
    begin
        if (rising_edge(clk)) then
            if ena = '1' then
                count := count + 1;
                if (count = max) then
                    temp := not temp;
                    count := 0;
                end if;
            end if;
        end if;
        output <= temp;
    end process;
END ARCHITECTURE;
```

Exemplo: divisor de clock



- Quais sinais são necessários para simular este componente?

Exemplo: divisor de clock



- ▶ Quais sinais são necessários para simular este componente?
 - *clk*
 - *ena*
 - Saída é verificada pela inspeção de *output*

Exemplo: divisor de clock

```
ARCHITECTURE stimulus OF testbench IS

  -- Decaclarção do componente
  component divisor_clock is
    port (clk, ena: IN std_logic;
          output: OUT std_logic);
  end component divisor_clock;

  -- declaração de sinais
  signal clk : std_logic := '0';
  signal ena : std_logic := '0';
  signal output : std_logic;

BEGIN  -- início do corpo da arquitetura

  -- Instância de divisor_clock com nome dut, pode haver
  -- mais do que uma
  dut: divisor_clock port map (
    clk => clk,
    ena => ena,
    output => output);
```

Exemplo: divisor de clock

```
ARCHITECTURE stimulus OF testbench IS

  -- Declaração do componente
  component divisor_clock is
    port (clk, ena: IN std_logic;
          output: OUT std_logic);
  end component divisor_clock;

  -- declaração de sinais
  signal clk : std_logic := '0';
  signal ena : std_logic := '0';
  signal output : std_logic;

BEGIN  -- início do corpo da arquitetura

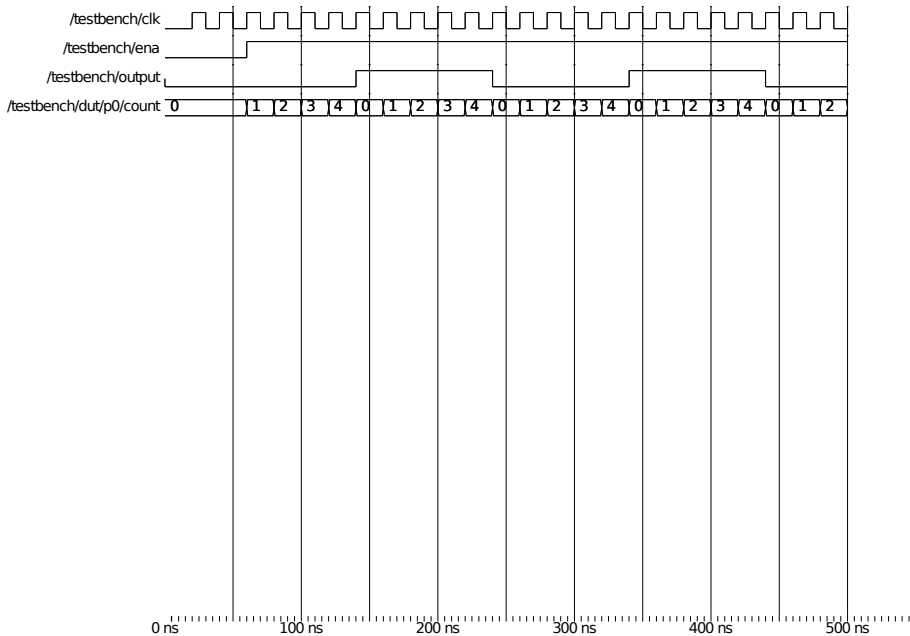
  -- Instância de divisor_clock com nome dut, pode haver
  -- mais do que uma
  dut: divisor_clock port map (
    clk => clk,
    ena => ena,
    output => output);
```

Exemplo: divisor de clock

```
-- gera um clock
process
begin
    wait for 10 ns;
    clk <= '0';
    wait for 10 ns;
    clk <= '1';
end process;

--gera enable
process
begin
    wait for 60 ns;
    ena <= '1';
    wait;
end process;

END ARCHITECTURE stimulus;
```



- 1 Introdução
- 2 Simulação de circuitos: testbenchs
- 3 Exemplo: divisor de clock
- 4 ModelSim**
- 5 Exercícios

- ▶ ModelSim é a ferramenta de simulação muito poderosa
- ▶ Para *Testbenchs* Tipo I (funcionamento sem propagação) ele é independente da ferramenta de síntese.
- ▶ Para simular:
 - Mude para o diretório do Projeto: **File** → **Change Directory**
 - Na janela **Transcript** execute: *do <script de simulacao.do>*

Utilizado ModelSim

```
#Cria biblioteca do projeto
vlib work

#compila projeto: todos os arquivo. Ordem é importante
vcom divisor_clock.vhd tb_divisor.vhd

#Simula (work é o diretório, testbench é o nome da entity)
vsim -t ns work.testbench

#Mosta forma de onda
view wave

#Adiciona ondas específicas
# -radix: binary, hex, dec
# -label: nome da forma de onda
add wave -radix binary /clk
add wave -radix binary /ena
add wave -radix binary /output
#Como mostrar sinais internos do processo
add wave -radix dec /dut/p0/count

#Simula até um 500ns
run 500ns

wave zoomfull
write wave wave.ps
```

- 1 Introdução
- 2 Simulação de circuitos: testbenchs
- 3 Exemplo: divisor de clock
- 4 ModelSim
- 5 Exercícios**

- ▶ Baixe do Moodle os arquivos exemplos
- ▶ Execute cada uma das simulações no ModelSim