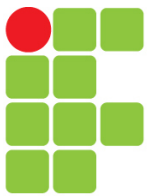
Simulação e TestBenchs

Engenharia Eletrônica

Prof. Renan Augusto Starke

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

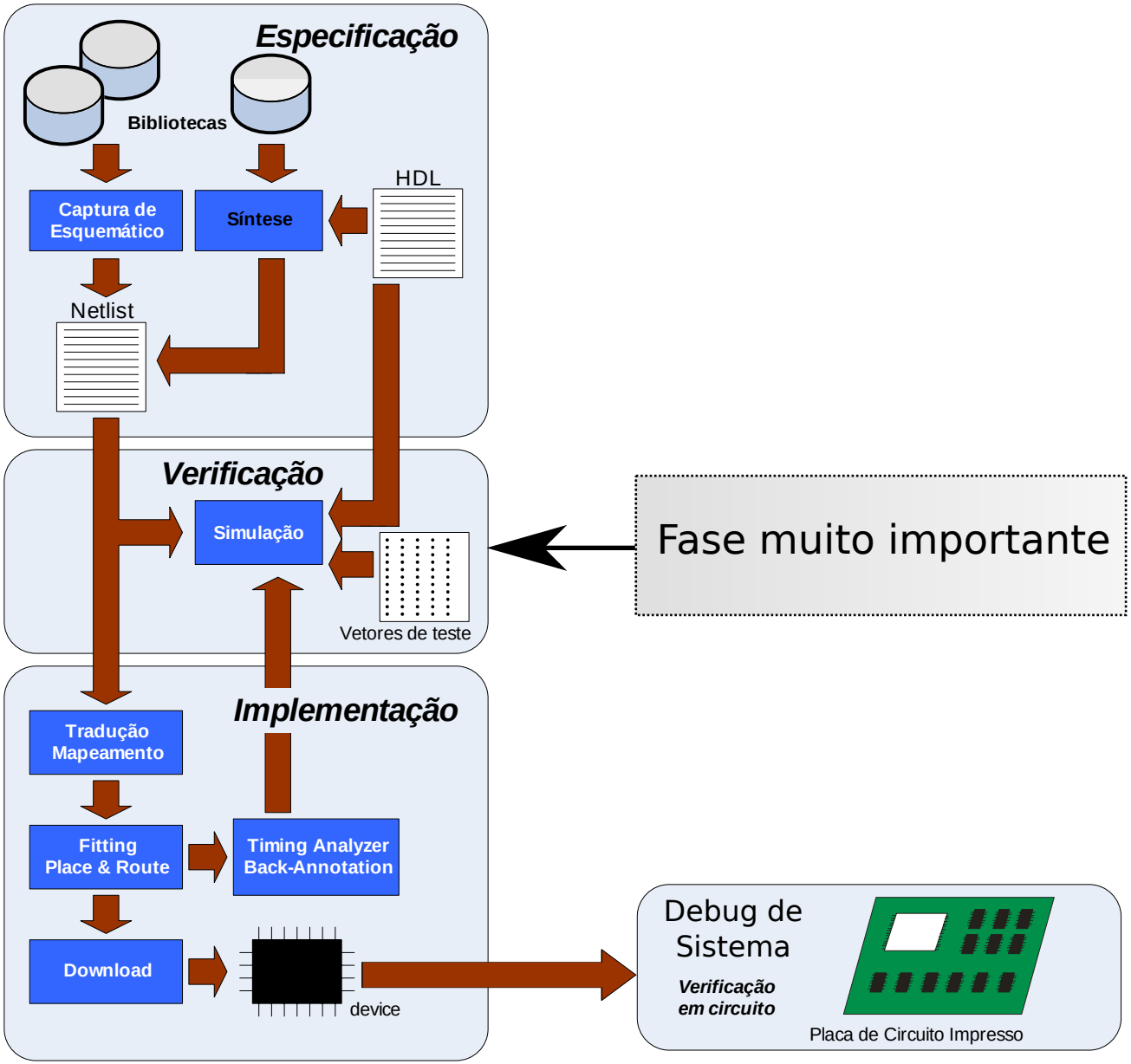
19 de julho de 2022



INSTITUTO FEDERAL
SANTA CATARINA

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

Fluxo de projeto



Síntese e simulação

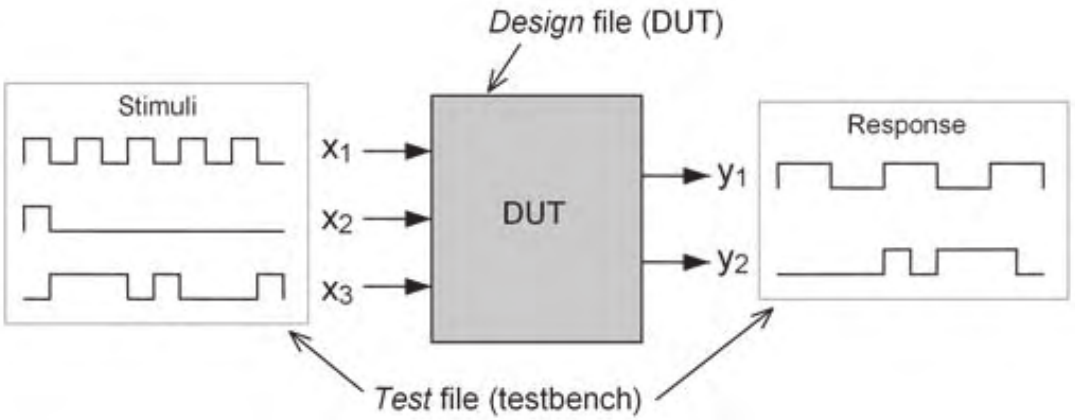
Síntese

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)

Procedimento geral

Testbench

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

- Testbenchs 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.)

Tipos de Testbenchs

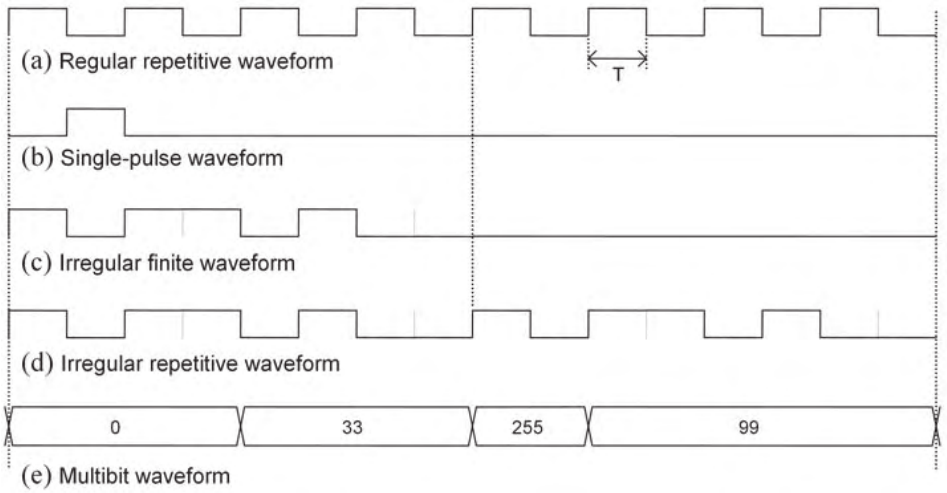
- Tipo I:
  - Não considera tempos de propagação
  - Inspeção manual da saída
  - Tipo mais simples
- Tipo II:
  - Tempos de propagação do DUT são considerados
  - Inspeção manual da saída
  - Conhecido como: *simulação temporal manual*

Tipos de Testbenchs

- 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*

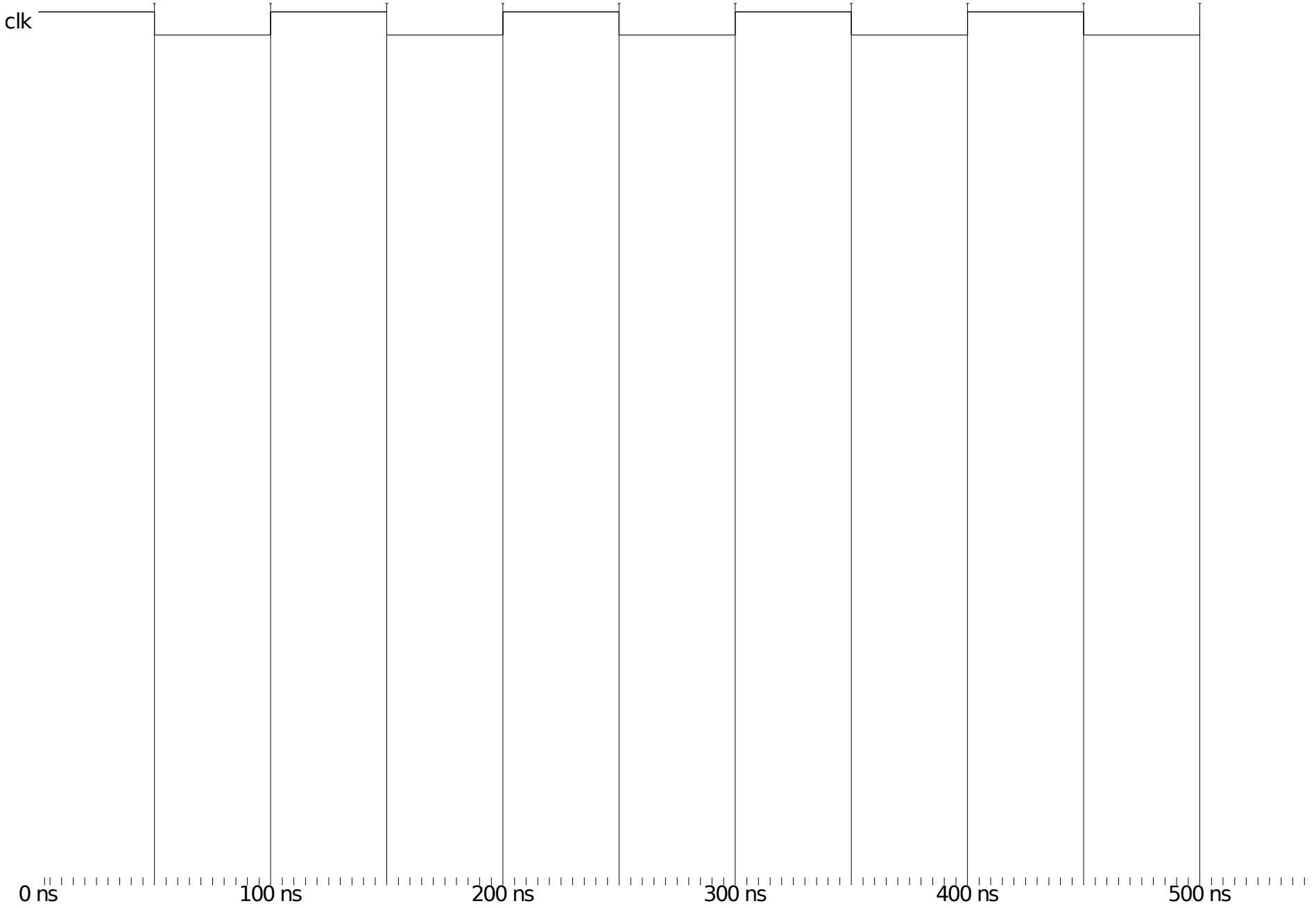
c) Forma de onda irregular finita

d) Forma de onda irregular repetitiva

e) Forma de onda multibit

Geração de estímulos: clocks

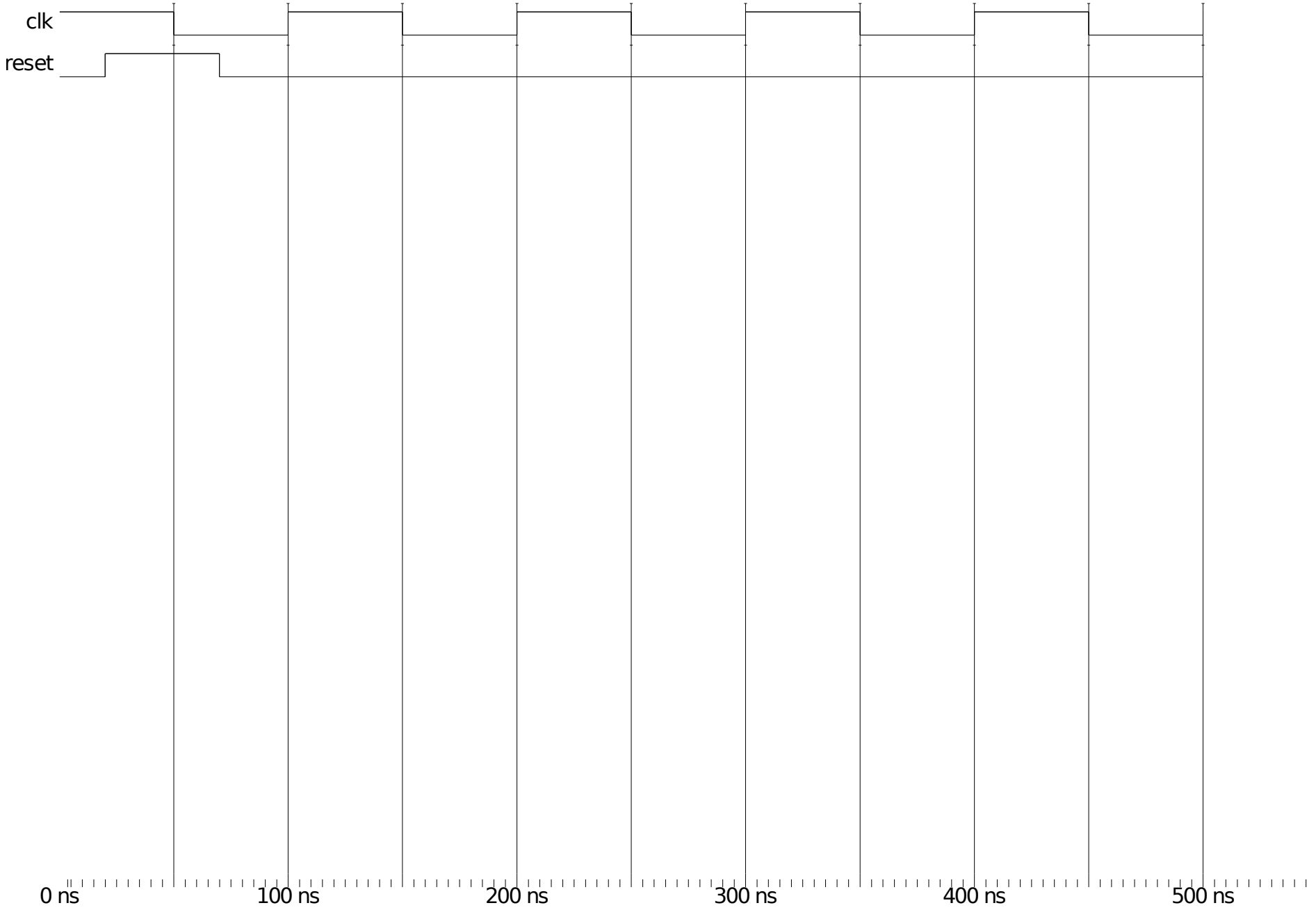
```
signal clk : std_logic;

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



Entity:testbench Architecture:stimulus Date: Sun Jul 10 18:45:27 BRT 2016 Row: 1 Page: 1

Geração de estímulos: reset

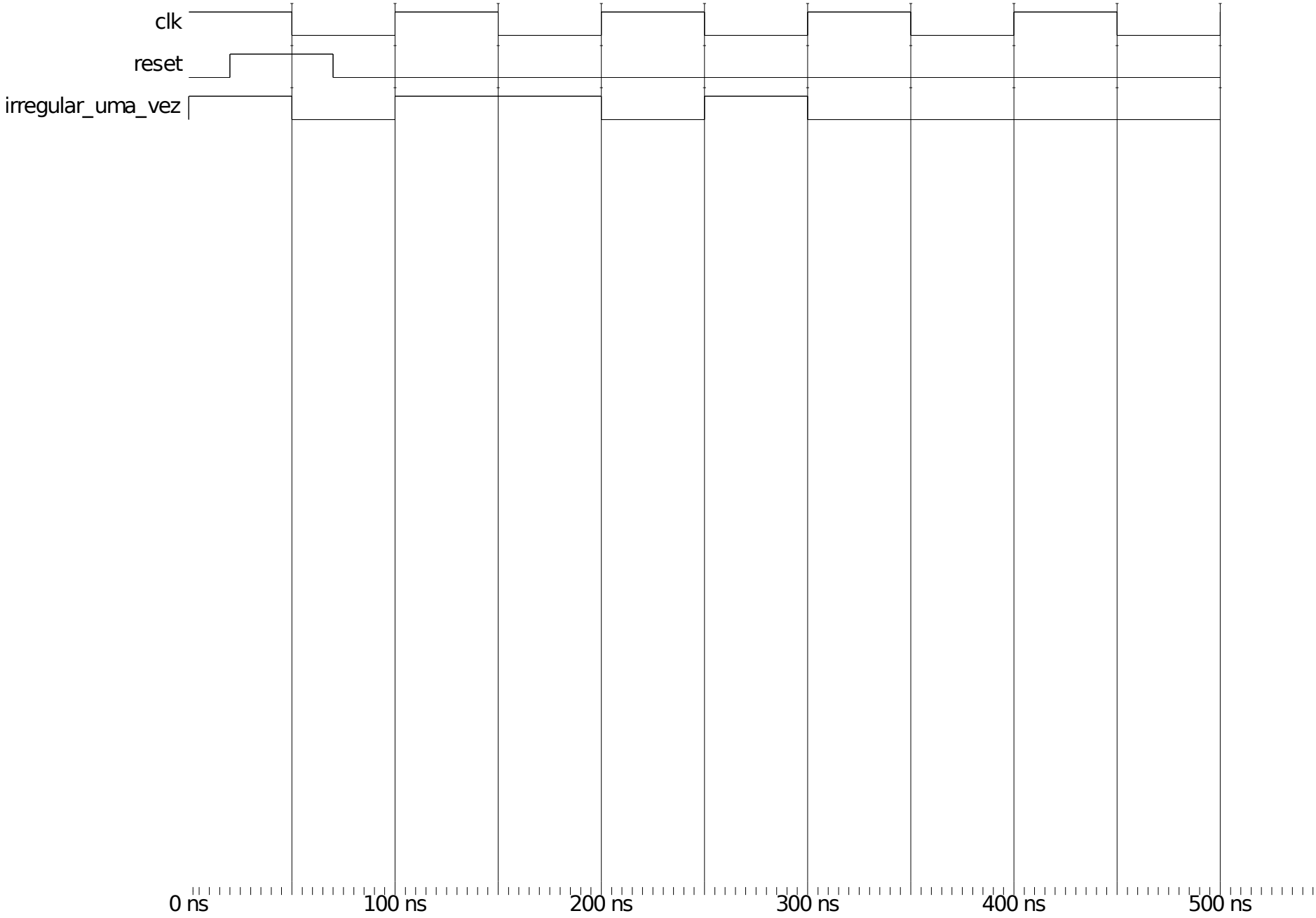
```
signal rst : std_logic;

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



Entity:testbench Architecture:stimulus Date: Sun Jul 10 19:11:44 BRT 2016 Row: 1 Page: 1

Geração de estímulos: não repetitiva irregular

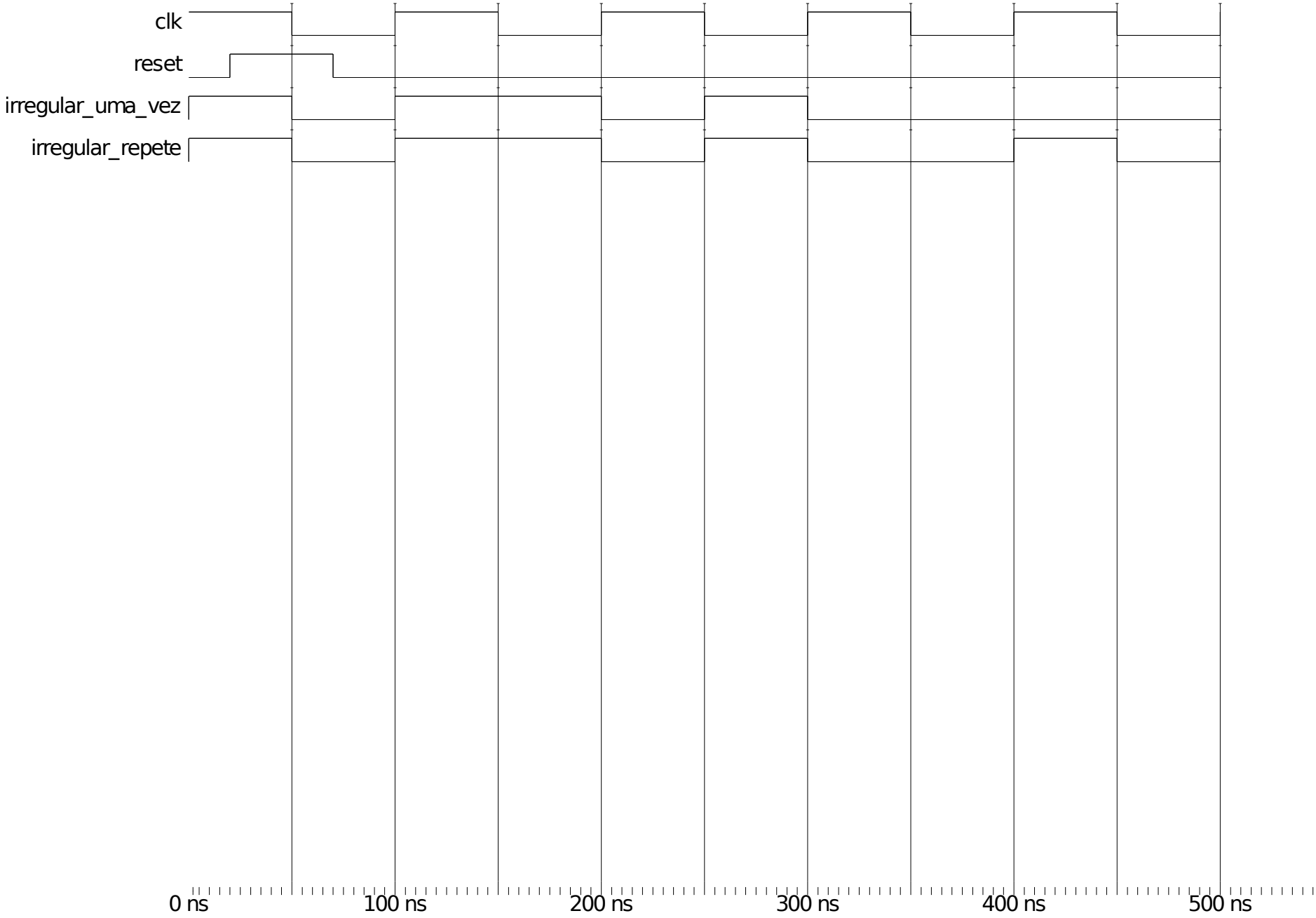
```
signal x : std_logic;

-- 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;
```



Entity:testbench Architecture:stimulus Date: Sun Jul 10 19:16:17 BRT 2016 Row: 1 Page: 1

Geração de estímulos: repetitiva irregular

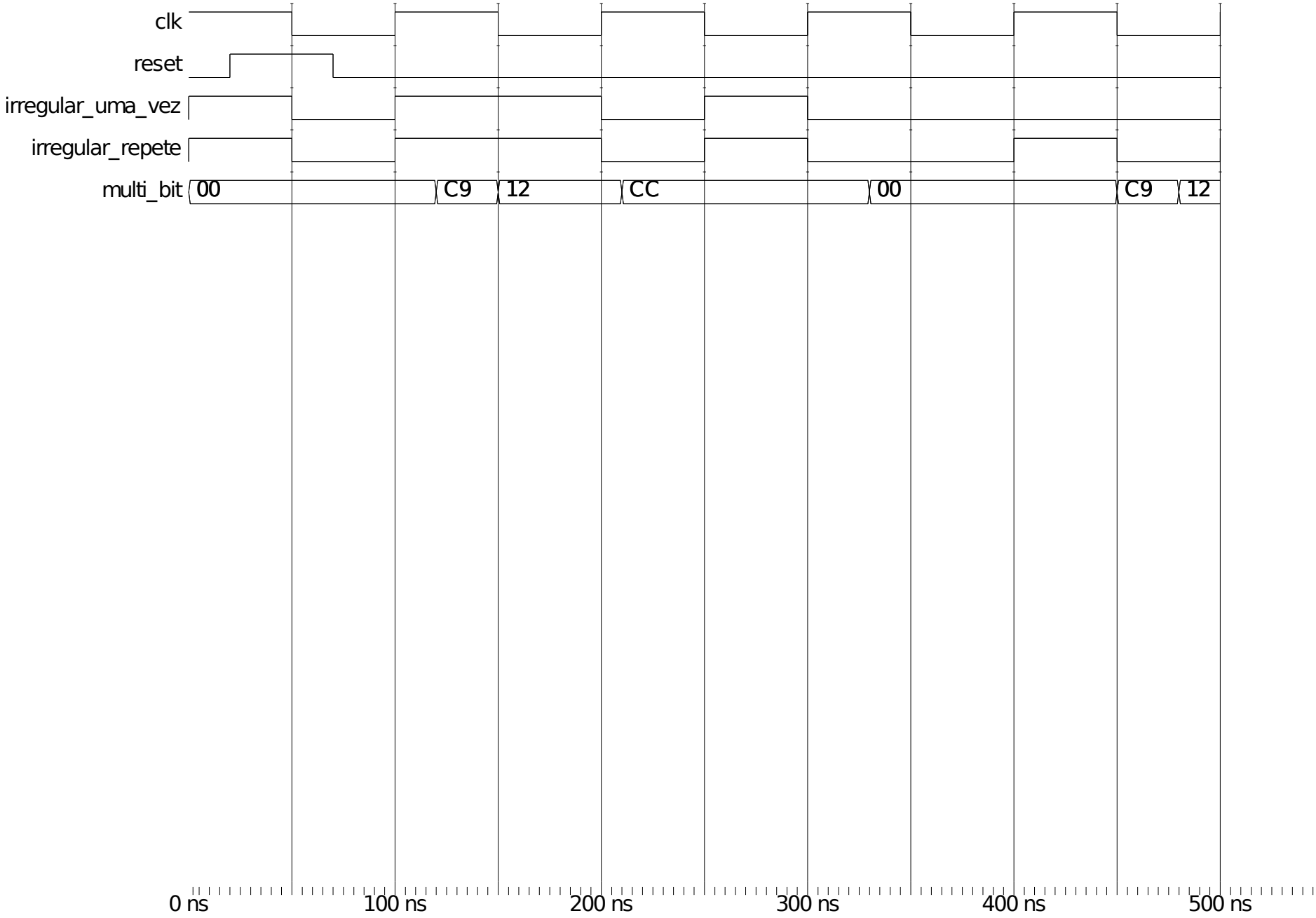
```
signal y : std_logic;

-- 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;
```



Entity:testbench Architecture:stimulus Date: Sun Jul 10 19:19:49 BRT 2016 Row: 1 Page: 1

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;
```



Entity:testbench Architecture:stimulus Date: Sun Jul 10 19:21:26 BRT 2016 Row: 1 Page: 1

Geração de estímulos

```
-- Bibliotecas e clásulas
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;
    signal rst : std_logic;
    signal x   : std_logic;
    signal y   : std_logic;
    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
        clk <= '1';
        wait for 50 ns;
        clk <= '0';
        wait for 50 ns;
    end process;
```

Geração de estímulos

```vhdl
-- Bibliotecas e clásulas
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;
    signal rst : std_logic;
    signal x   : std_logic;
    signal y   : std_logic;
    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
        clk <= '1';
        wait for 50 ns;
        clk <= '0';
        wait for 50 ns;
    end process;
```

Geração de estímulos

```
-- Bibliotecas e clásulas
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;
    signal rst : std_logic;
    signal x   : std_logic;
    signal y   : std_logic;
    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
        clk <= '1';
        wait for 50 ns;
        clk <= '0';
        wait for 50 ns;
    end process;
```

Geração de estímulos

```
-- Bibliotecas e clásulas
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;
    signal rst : std_logic;
    signal x   : std_logic;
    signal y   : std_logic;
    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
        clk <= '1';
        wait for 50 ns;
        clk <= '0';
        wait for 50 ns;
    end process;
```

Geração de estímulos

```
--gera uma forma de onda de um pulso
process
begin
    rst <= '0';
    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

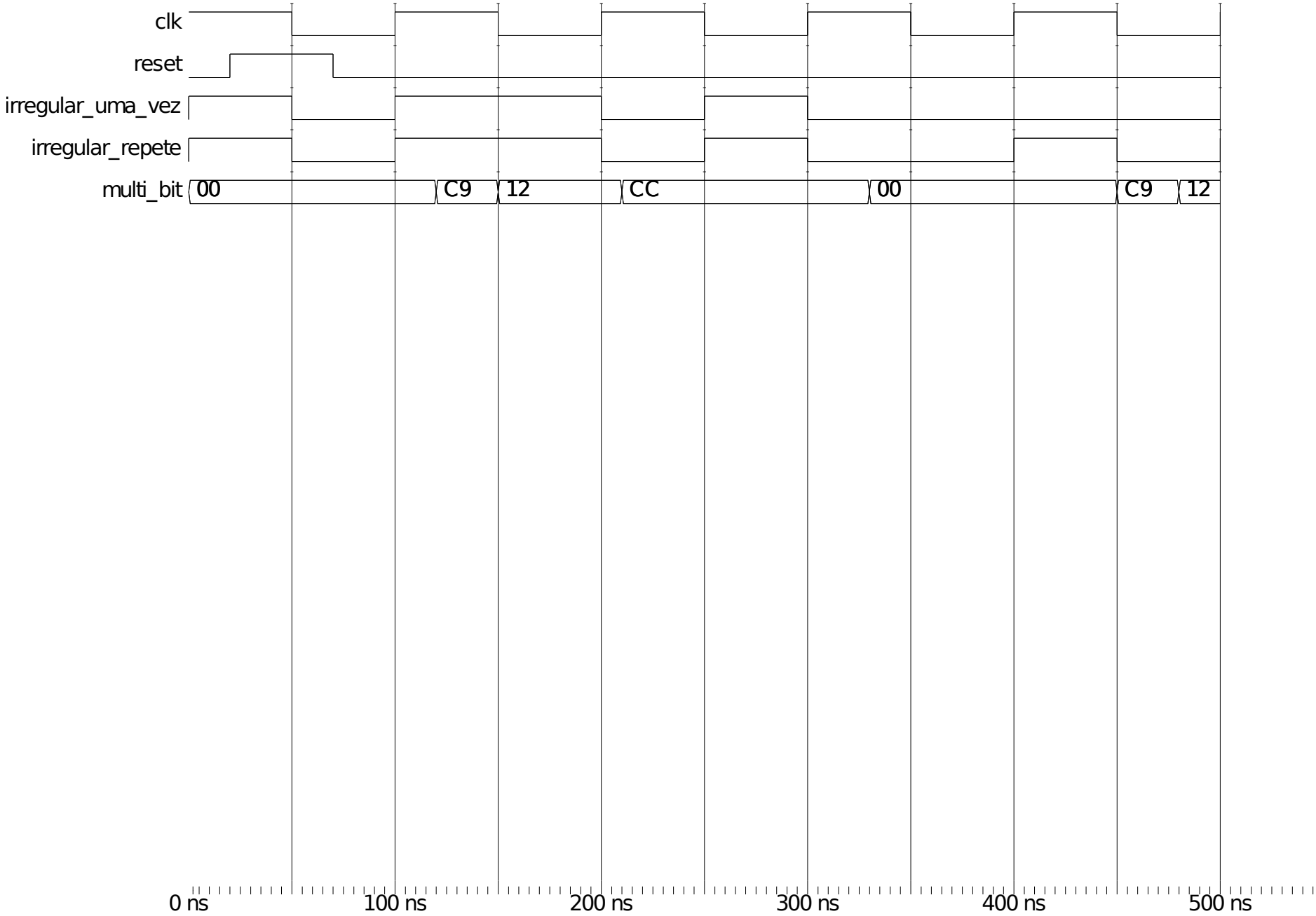
```
--gera uma forma de onda de um pulso
process
begin
    rst <= '0';
    wait for 20 ns;
    rst <= '1';
    wait for 50 ns;
    rst <= '0';
    wait;
end process;

-- 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
    rst <= '0';
    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;
```

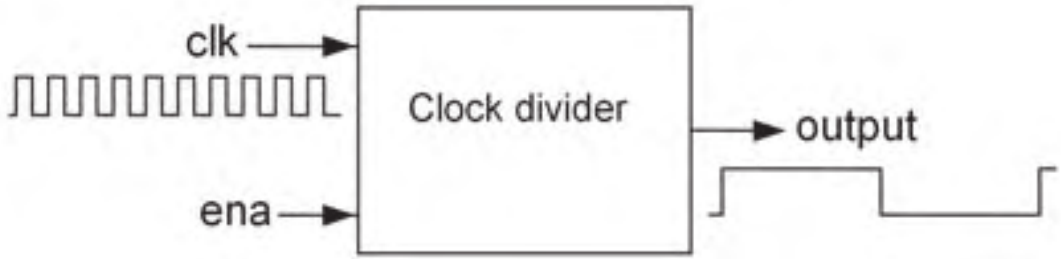
Geração de estímulos

```
    -- 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;
```



Entity:testbench Architecture:stimulus Date: Sun Jul 10 19:21:26 BRT 2016 Row: 1 Page: 1

Exemplo: divisor de clock



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

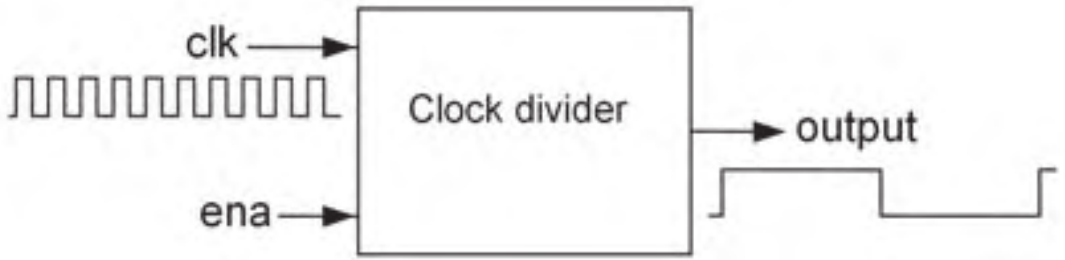
Exemplo: divisor de clock

```vhdl
-- Bibliotecas e clásulas
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

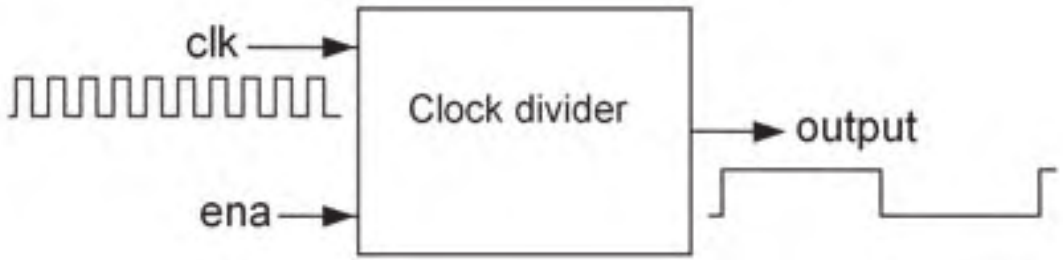
```vhdl
-- Bibliotecas e clásulas
LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.ALL;

-- 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

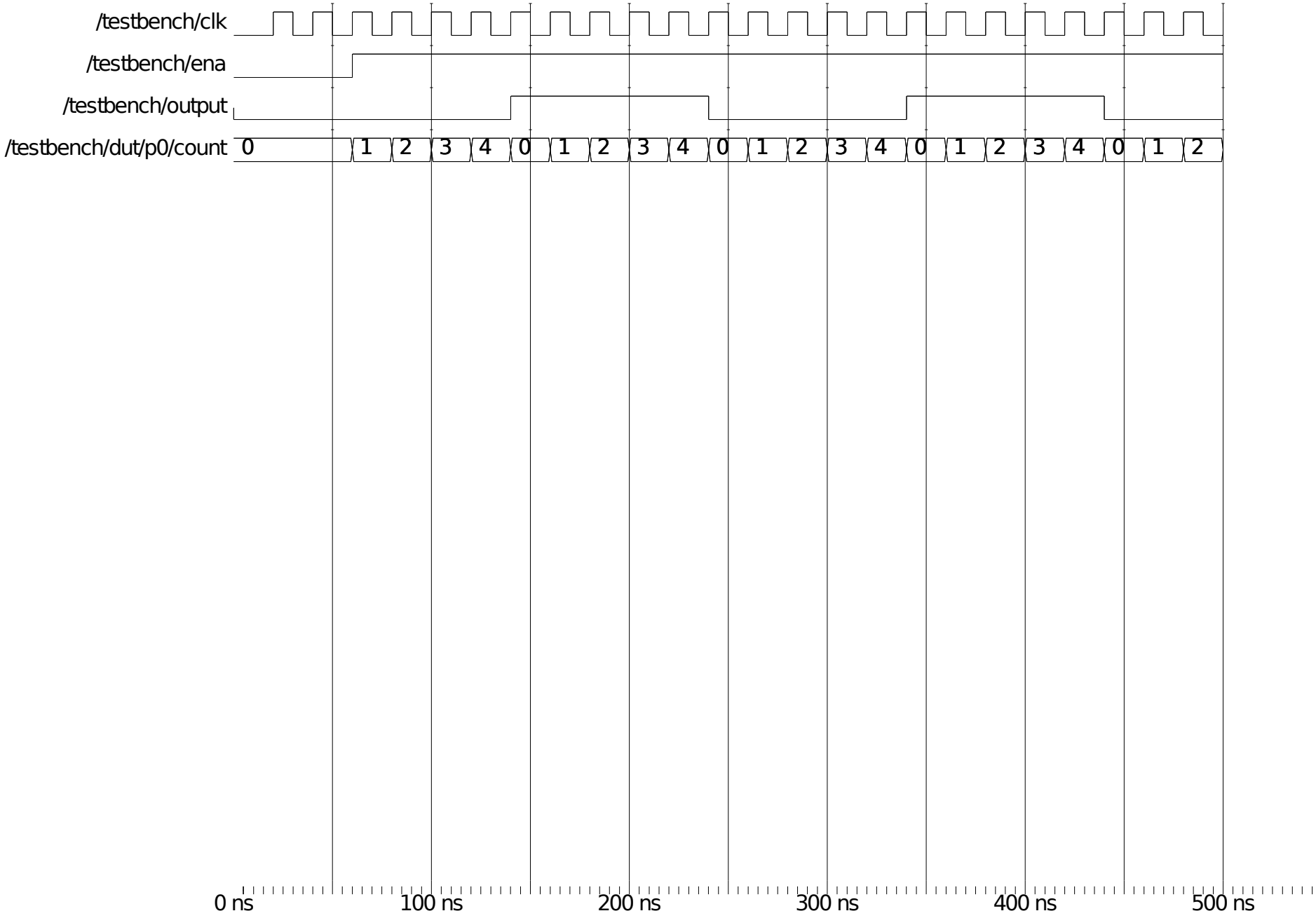
```
ARCHITECTURE stimulus OF testbench IS

    -- declaração de sinais
    signal clk_tb : std_logic;
    signal ena_tb : std_logic;
    signal output_tb : std_logic;

BEGIN  -- início do corpo da arquitetura

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

Exemplo: divisor de clock

```
ARCHITECTURE stimulus OF testbench IS

    -- declaração de sinais
    signal clk_tb : std_logic;
    signal ena_tb : std_logic;
    signal output_tb : std_logic;

BEGIN  -- início do corpo da arquitetura

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

Exemplo: divisor de clock

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

    --gera enable
    process
    begin
        ena_tb <= '0';
        wait for 60 ns;
        ena_tb <= '1';
        wait;
    end process;

END ARCHITECTURE stimulus;
```



Entity:testbench Architecture:stimulus Date: Mon Jul 11 18:54:22 BRT 2016 Row: 1 Page: 1

Utilizado ModelSim

- 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 aquivo. Ordem é importante
vcom divisor_clock.vhd tb_divisor.vhd

#Simula (work é o diretório, testbench é o nome da entity)
vsim  -voptargs="+acc" -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 -label clk -radix binary /clk_tb
add wave -label en -radix binary  /ena_tb
add wave -label output -radix binary  /output_tb

#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
```