Verificação

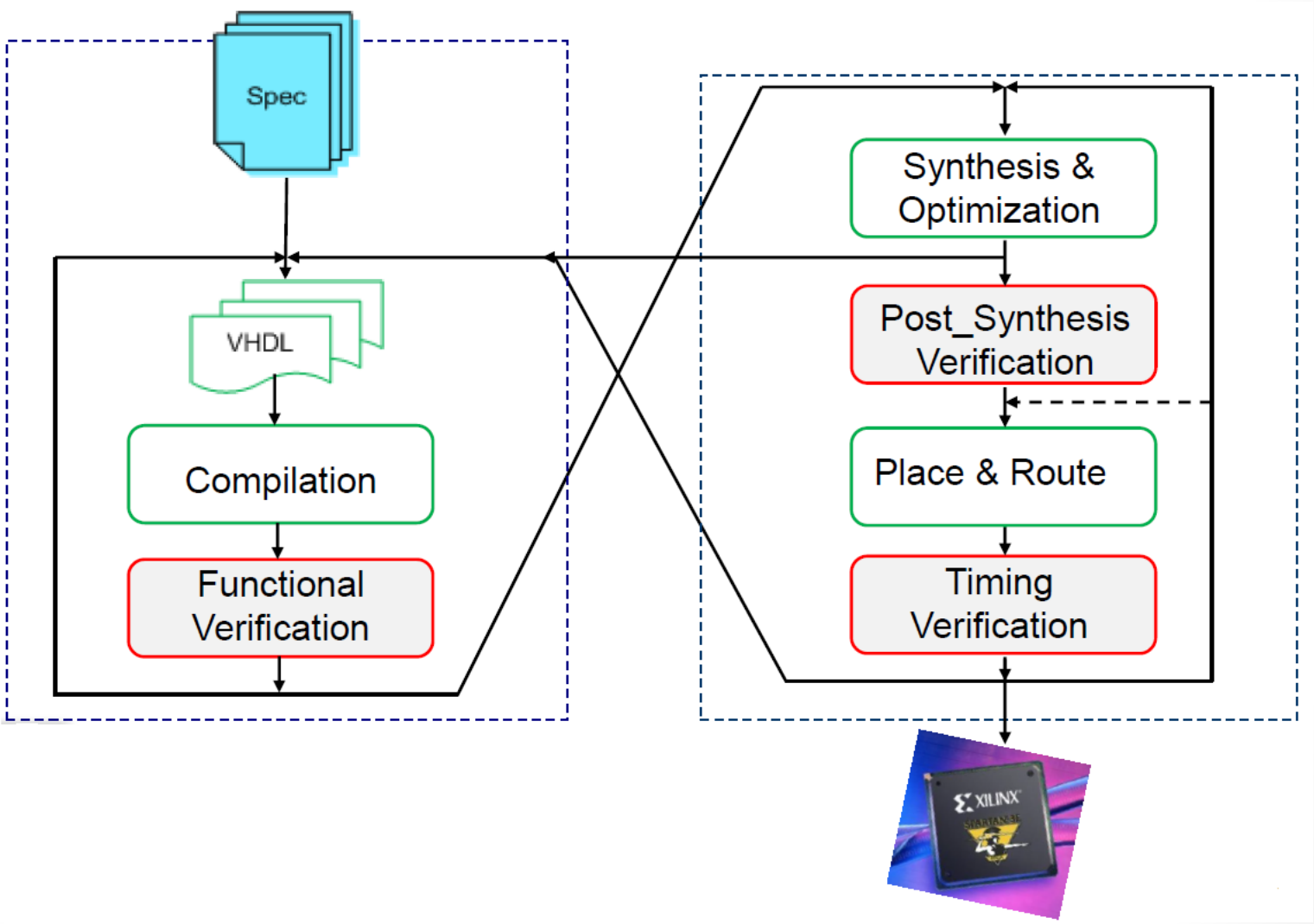
Verificação

É uma parte importante do processo de design: Como saber que o nosso projeto funciona como esperado?

É importante/necessario verificar a funcionalidade do projeto em pelo menos uma das seguientes etapas:

- Functional Verification / Pre-synthesis Verification
- Post-synthesis
- Timing Verification / Post-Place & Route Verification

Verificação



Verificação-Test Bench

Test bench

è um modelo em HDL que gera estímulos (formas de onda) para avaliar o circuito digital descrito em HDL.

- É usado para verificar o correto funcionamento do projeto de hardware.
- Além da síntese, o VHDL pode ser usado como uma linguagem de verificação.
- Muito importante para obter uma boa compreensão e verificação do seu projeto.
- Para simular o seu projeto é preciso criar uma entidade e arquitetura adicional.

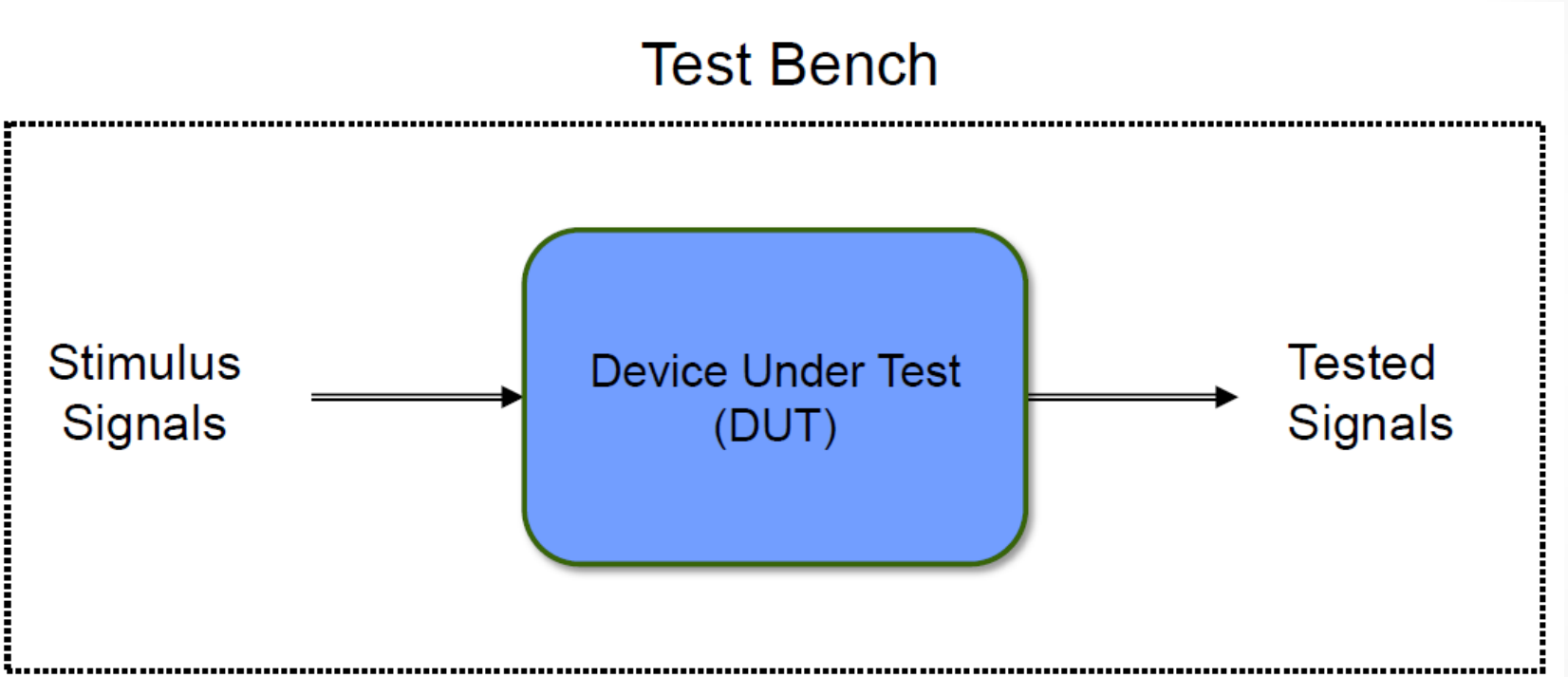
Verificação - Test Bench

O Test Bench tem especificamente 3 propósitos

1. Gerar estímulos para simulação.
2. Aplicar os estímulos na entidade sobre verificação e coletar os dados das saídas.
3. Comparar as respostas das saídas com os valores esperados.

Test bench deveriam ser criados por engenheiros diferentes aos projetistas do modulo sobre verificação.

Verificação - Test Bench



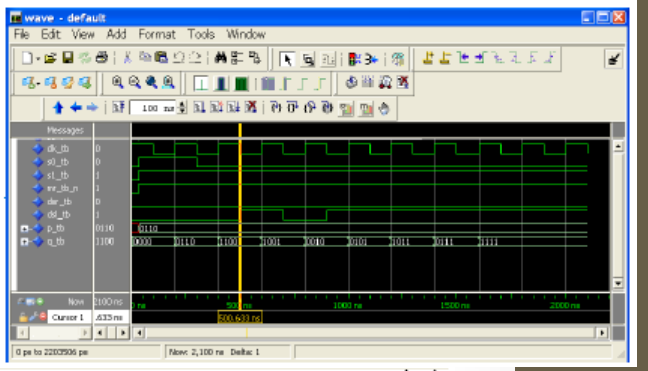
Verificação - Test Bench

Componentes

1. TB entidade
   - Sem Declaração (Vazia)
2. TB arquitetura
   - Declaração de componentes
   - Declaração de sinais locais
   - Instanciar componentes
   - Sinais de estimulus
   - Verificação de valores esperados

Verificação - Test Bench

Simulator



Test Bench Processes

Data Generation

Clock Generation

Reset Generation

Data Verification

DUT

Comb. Logic

Comb. Logic

Comb. Logic

Verificação-Test Bench

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

entity test_my_design is
end test_my_design;

architecture testbench of test_my_design is
  component my_design is
     port (a,b : in std_logic;
           x,y : out std_logic);
  end component;

  signal as,bs : std_logic:='1';
  signal xs,ys : std_logic;

begin
  uut : my_design port map
        (a=>as, b=>bs, x=>xs, y=>ys);

    as <= not(as) after 50 ns;

  process begin
    bs <= '1'; wait for 75 ns;
    bs <= '0'; wait for 25 ns;
  end process;
end testbench;
```

Define library, same as in VHDL source code

VHDL model without entity interface

Component declaration of the device to test

Define signal names

Instantiated UUT in test bench

Define the stimulus of the test

Verificação-Test Bench

Use of *wait*

The wait statement can be located anywhere between begin and end process

- Basic Usages:

```
wait for time;

wait until condition;

wait on signal_list;

wait;
```

Verificação-Test Bench

Use of *wait*

```
process
   . . .
  J <= '1';
  wait for 50 ns;-- process is suspended for 50 ns after J is
 . . .                -- assigned to '1'
end process;
```

```
process
   . . .
  wait until CLK = '1';-- sync with CLK rising edge before
                       -- continuing of simulation
   . . .
end process;
```

Verificação-Test Bench

Use of *wait*

```
process
  . . .
  wait on A until CLK = '1';-- the process is resumed
                            -- after a change on A signal,
                            -- but only when the value of
                            -- the signal CLK is equal to '
  . . .
end process;
```

```
process
    rst <= '1';
    wait for 444 ns;
    rst <= '0';
    wait;                   -- used without any condition,
                            -- the process will be suspended
                            -- forever
end process;
```

Verificação-Test Bench

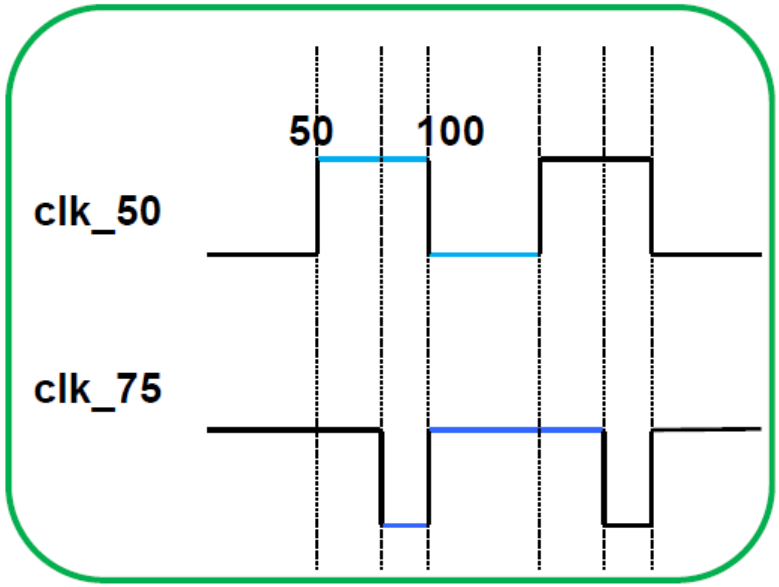
Clock Generation

```
architecture testbench of test_my_design is
  signal clk_50 : std_logic := '1';
  signal clk_75 : std_logic := '0';
  constant clk_period  : time := 100 ns;
  constant h_clk_period: time := 50 ns;

begin

-- case 1: concurrent statement
clk_50 <= not(clk_50) after h_clk_period;-- 50% duty

-- case 2: sequential statement
clk_75_proc: process
begin
    clk_75 <= '1';
    wait for 75 ns; -- 75% duty
    clk_75 <= '0';
    wait for 25 ns;
end process clk_75_proc;
. . .
. . .
end testbench;
```



Verificação-Test Bench

Clock Generation

```
architecture testbench of test_my_design is

signal clk: std_logic := '0';
constant period: time := 40 ns;

begin

diff_duty_clk_cycle: process
begin
   clk <= '0';
    wait for period * 0.60;
   clk <= '1';
    wait for period * 0.40; -- 60% duty cycle

end process diff_duty_clk_cycle;

. . .
end architecture testbench;
```

Verificação-Test Bench

Clock Generation

```
architecture testbench of test_my_design is

   signal clk: std_logic:= '0';
   constant half_period: time := 20 ns;

begin

clk_cycle: process
begin
   clk <= '0';
    wait for half_period; -- the clock will toggle
   clk <= '1';
    wait for half_period; -- as long as the simulation
                          -- is running
end process clk_cycle;
. . .

end architecture testbench;
```

Verificação-Test Bench

Data Generation

- Avoid race conditions between data and clock
  - Applying the data and the active edge of the clock simultaneously might cause a race condition

- To keep data synchronized with the clock while avoiding race condition, apply the data at a different point in the clock period that at the active edge of clock

Verificação-Test Bench

Data Generation

Example of Data generation on inactive clock edge

```
clk_gen_proc: process
begin
Clk <= '0';
wait for 25 ns;
clk<= '1';
wait for 25 ns;
end process clk_gen_proc;
```

Clock generation process

```
data_gen_proc: process
while not (data_done) loop
    DATA1 <= X1;
    DATA2 <= X2;
    ...
    wait until falling_edge(clk);
 end loop;
end process data_gen_proc;
```

Data generation process

Verificação-Test Bench

Data Generation

Relative time: signal waveforms that are specified to change at simulation times relative to the previous time, in a time accumulated manner

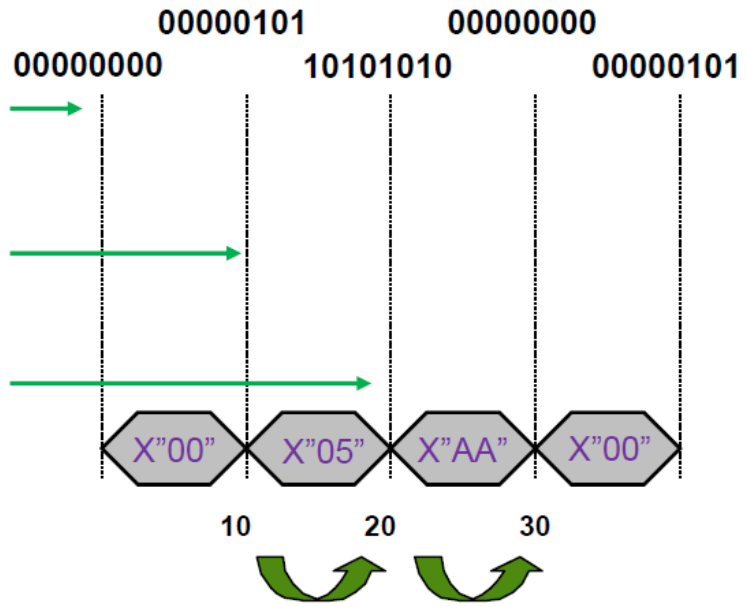
```
architecture relative_timing of myTest is

     signal Add_Bus : std_logic_vector(7 downto 0);

begin
   patt_gen_proc: process
     begin
       Add_Bus <= "00000000";
       wait for 10 ns;

       Add_Bus <= "00000101";
       wait for 10 ns;

       Add_Bus <= "10101010";
       wait for 10 ns;
   end process patt_gen_proc;
. . .
end relative_timing;
```



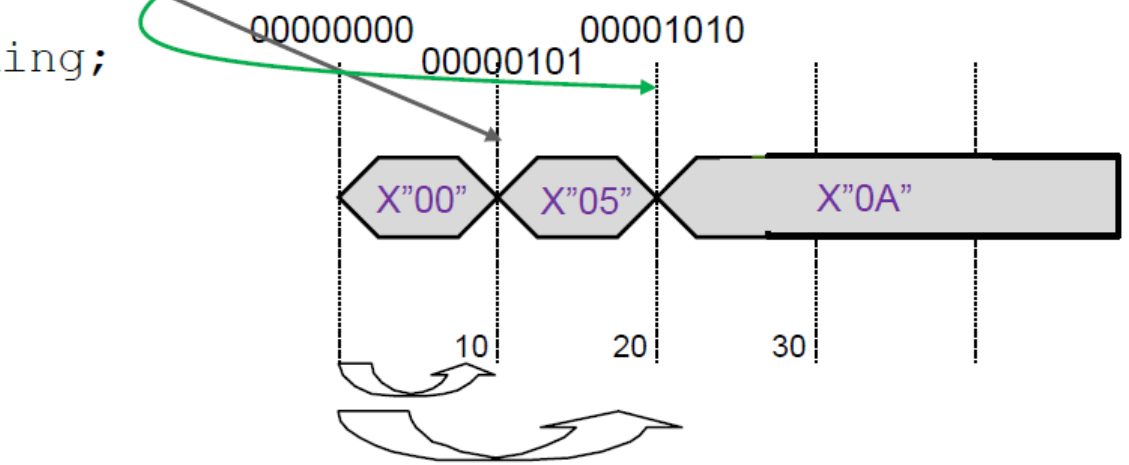
Verificação-Test Bench

Data Generation

Absolute time: signal waveforms that are specified to change at simulation times absolute since the moment that the simulation begin

```
architecture absolute_timing of testbench is
     signal A_BUS : std_logic_vector(7 downto 0);

begin
       A_BUS   <=    "00000000",
                     "00000101"  after 10 ns,
                     "00001010"  after 20 ns;
                                 -- etc.
. . .
end absolute_timing;
```

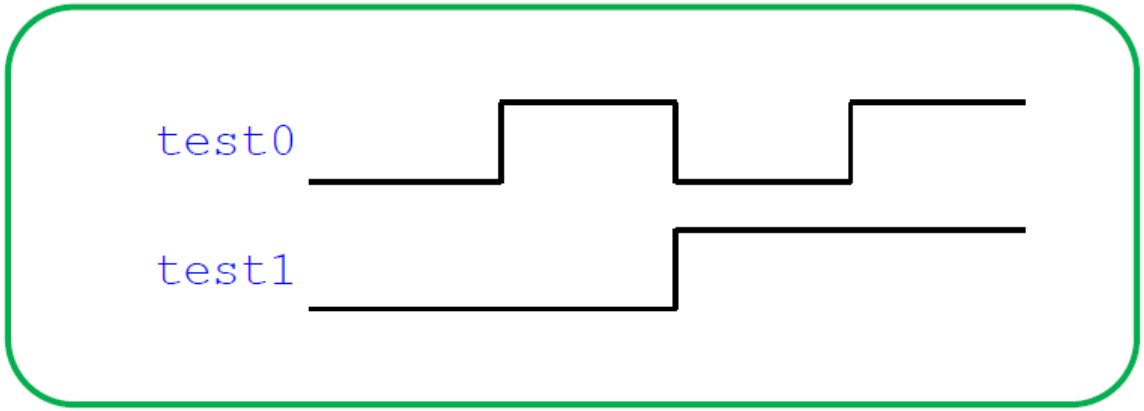


Verificação-Test Bench

Data Generation

Absolute time: signal waveforms that are specified to change at simulation times absolute since the moment that the simulation begin

```
-- 2 bits test pattern
begin
        test0 <= '0', '1' after 10 ns, '0' after 20 ns,
                 '1' after 30 ns;
        test1 <= '0',
                 '1' after 20 ns;
. . .
```



Verificação-Test Bench

Data Generation

```
architecture array_usage of in_test_benches is
   signal Add_Bus : std_logic_vector(7 downto 0);
   -- type & signal declarations: 5 data for 8 bits
   type stimulus is array (0 to 4) of
                              std_logic_vector (7 downto 0);

   constant DATA : stimulus :=
              ("00000000",          -- declare the stimulus
               "00000001",          -- as an array.
               "00000010",          -- these values will be
               "00000011",          -- used to stimulate the
               "00000100");         -- inputs
begin
     stim_proc: process
     begin
       for i in 0 to 4 loop      -- for loop that assign
           Add_BUS <= DATA(i); -- to Add_Bus a new value
           wait for 10 ns;       -- from stimulus every 10ns
       end loop;
     end process stim_proc;
. . .
end array_usage;
```

Verificação-Test Bench

Reset Generation

```
-- asynchronous desassert reset
reset: process
begin
   rst <= '1';
   wait for 23 ns;
   rst <= '0';
   wait for 1933 ns;
   rst <= '1';
   wait for 250 ns;
   rst <= '1';
   wait;
end process;
```

Verificação-Test Bench

Reset Generation

```
-- synchronous desassert reset
sreset: process
begin
   rst <= '1';
   for i in 1 to 5 loop
     wait until clk = '1';
   end loop;
   rst <= '0';
end process;
```

Verificação-Test Bench

Ex. Decoder 2:4

```
entity dcd_2_4 is
  port (in1  : in  std_logic_vector (1 downto 0);
        out1 : out std_logic_vector (3 downto 0));
end dcd_2_4;
--
architecture dataflow of dcd_2_4 is
begin
  with in1 select
    out1 <= "0001" when "00",
            "0010" when "01",
            "0100" when "10",
            "1000" when "11",
            "0000" when others;
end dataflow;
```

Verificação-Test Bench

Ex. Decoder 2:4

```
-- Test Bench to exercise and verify
-- correctness of DECODE entity
entity tb2_decode is
end tb2_decode;

architecture test_bench of tb2_decode is

type input_array is array(0 to 3) of
                          std_logic_vector(1 downto 0);
constant input_vectors: input_array :=
                          ("00", "01", "10", "11");

signal in1  : std_logic_vector (1 downto 0);
signal out1 : std_logic_vector (3 downto 0);

component decode
      port (
            in1 : in  std_logic_vector(1 downto 0);
            out1: out std_logic_vector(3 downto 0));
end component;
```

Verificação-Test Bench

Ex. Decoder 2:4

Stimulus to the Inputs and Component Instantiation:

```
begin
decode_1: decode port map(
                    in1  => in1,
                    out1 => out1);

apply_inputs: process
begin
  for j in input_vectors'range loop
      in1 <= input_vectors(j);
      wait for 50 ns;
  end loop;
end process apply_inputs;
```

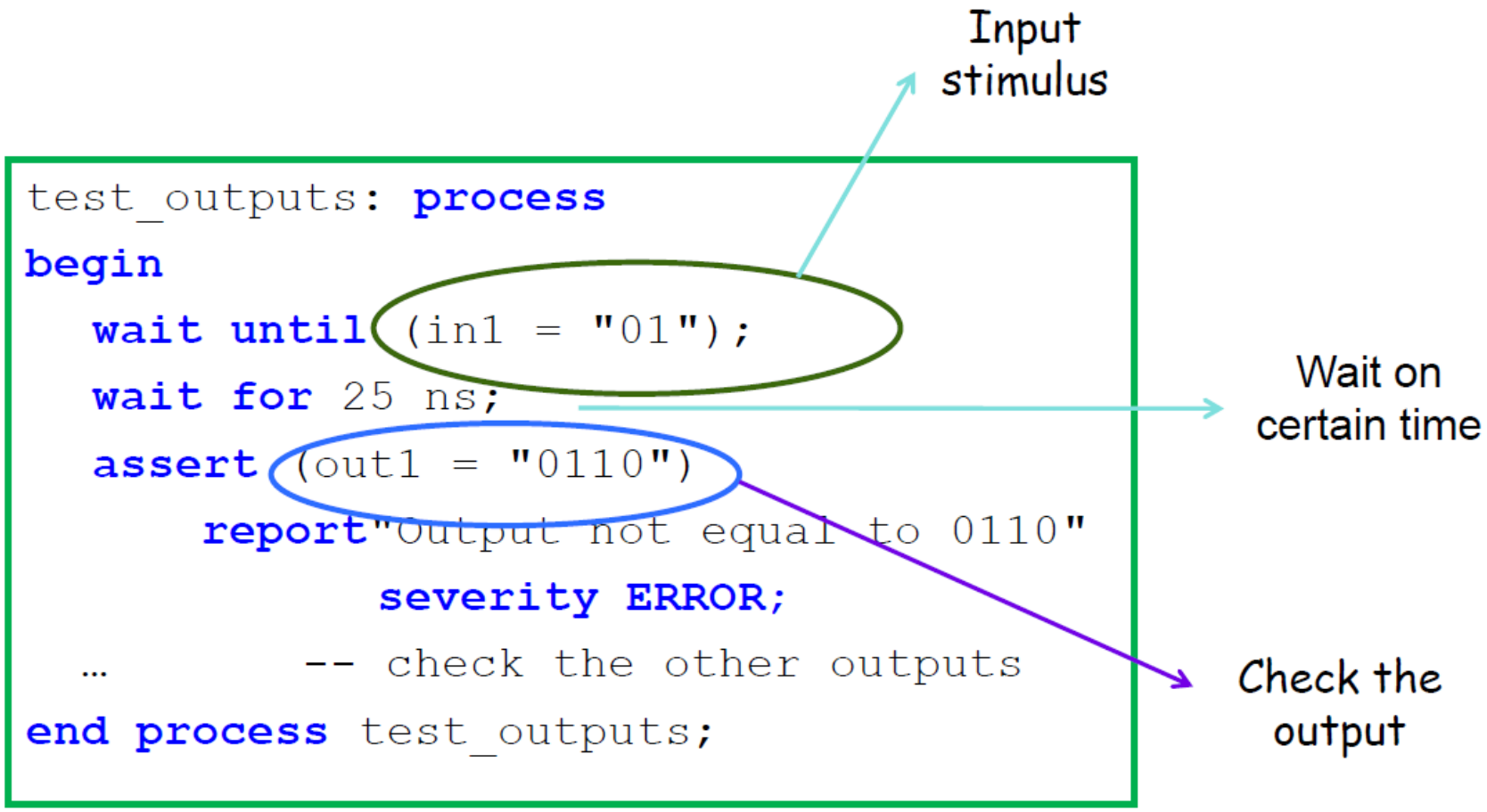
Component Instantiation Inputs Stimulus and Outputs port map

Data generation

Verificação-Test Bench

Ex. Decoder 2:4

Data verification:



Verificação - Test Bench

Ex. Decoder 2:4

```
-- Test Bench to exercise and verify
-- correctness of DECODE entity
entity tb3_decode is
end tb3_decode;

architecture test_bench of tb3_decode is

type decoder_test is record
  in1: std_logic_vector(1 downto 0);
  out: std_logic_vector(3 downto 0);
end record;

type test_array is array(natural range <>) of decoder_test;

constant test_data: test_array :=
   ("00", "0001",
    "01", "0010",
    "10", "0100",
    "11", "1000");
-- same declaration as before
```

Verificação-Test Bench

Ex. Decoder 2:4

```
begin
decode_1: decode port map(
                    in1  => in1,
                    out1 => out1);
apply_in_check_outs: process
begin
   for j in test_data'range loop
        in1 <= test_data(j).in1);
        wait for 50 ns;
        assert (out1 = test_data(j).out1)
            report "Output not equal to the expected value"
               severity ERROR;
   end loop;
end process apply_in_check_outs;
. . .
```

Verificação-Test Bench

Ex. Decoder 2:4

```
begin
…
apply_inputs: process
begin
   for j in test_data'range loop
        in1 <= test_data(j).in1);
        wait for 50 ns;
   end loop;
end process apply_inputs;
data_verif: process
begin
  wait for 25 ns;
  assert (out1 = test_data(j).out1)
            report "Output not equal to the expected
   value"
               severity ERROR;
  wait for 50 ns;
end process data_verif;
```

Verificação-Test Bench

File

```
file input_file_0 : text open read_mode  is "inputs0.txt";
```

```
process
    variable lin   : line;
    variable input : std_logic_vector(31 downto 0);
begin
    ff_in0_empty <= '0';
    wait until clk = '0';
    while not endfile(input_file_0) loop
        if ff_in0_re = '1' then
            readline(input_file_0, lin);
            read(lin, input);
            ff_in0_data <= input;
        end if;
        wait for clock_period;
    end loop;
    ff_in0_empty <= '1';
    wait until rst_n = '0';
end process;
```

Verificação-Test Bench

File

```
file output_file  : text open write_mode is "outputs.txt";

    process
        variable lout   : line;
        variable output : std_logic_vector(31 downto 0);
    begin
        ff_out_full <= '0';
        wait until clk = '0';
        while true loop
            if ff_out_we = '1' then
                output := ff_out_data;
                write(lout, output);
                writeline(output_file, lout);
            end if;
            wait for clock_period;
        end loop;
    end process;
```

Verificação-Test Bench

Concluçãões

- TB is done in VHDL code
- Emulate the Hardware
- Use all the power of VHLD
- There is no size limit
- TB is the top-level unit
- Usually is more complex than the design itself