Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Dipartimento di Elettronica e Informazione
Corso di Laurea in Ingegneria Informatica

Test Benching

Nov. 2018

Mar. 2007

Politecnico di Milano

Summary

- Introduction
- A test-benching
- The structure of a testbench
- Regression testing

© 2005 - Fabio Salice

Introduction

- A relevant aspect of the implementation of a project is to check that the description matches with the requirements
- Test a project consists in to generate a sequence of input patterns and to check the correctness of the corresponding output sequence
  - Generate the test vectors
    - It can be performed manually, one input at time
      - It is often impractical
    - Through one or several testbenches
  - Checking the outputs
    - The verification process can be a visual inspection of the generated output patterns (graphic analisys) …
    - or can be an automated way that verifies the correctness of the operations (comparing values against the expected results)

© 2005 - Fabio Salice

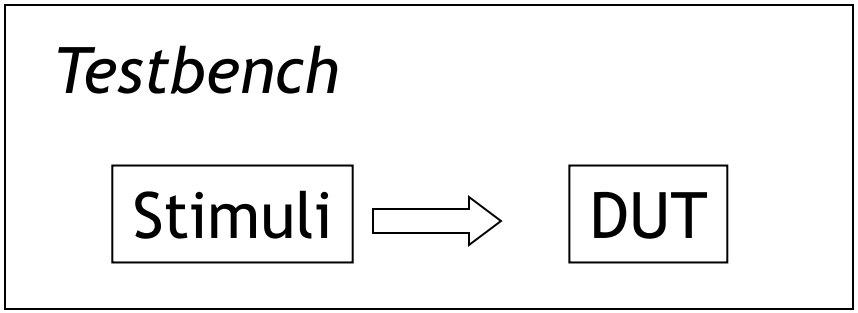
Introduction

- A project has been entirely verified when test vectors cover all the functionalities
  - The specification must be fully verified (100% coverage)
    - “If anything can go wrong, it will” –Murphy’s Law- (i.e. : any unverified aspect will fail)
- There are three aspects to consider:
  - How to build the testbench
  - How to perform the verification
  - How to ensure that the coverage of the implemented testbench is complete

© 2005 - Fabio Salice

A testbenching primer

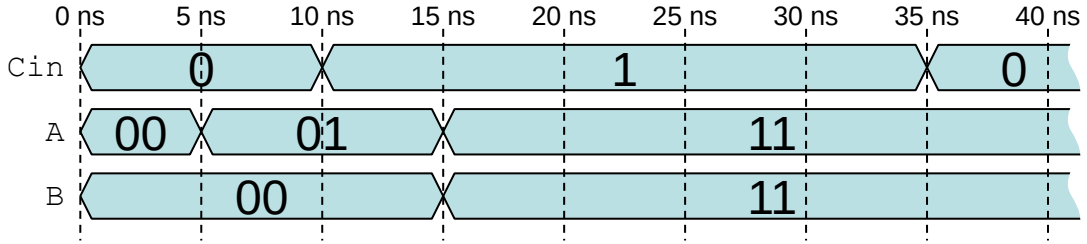
- A testbench is a VHDL module (not necessarily synthesizable) composed by:
  - An entity with no ports
    - The testbench **represents** the environment
  - The component under test
  - The constructs or processes that generate the stimuli
- DUT: design under test



© 2005 - Fabio Salice

A testbenching

- Example

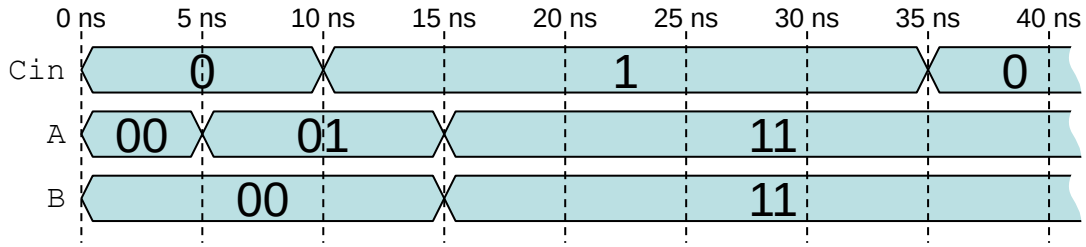



```
entity TestAdder is
end entity TestAdder;

architecture TB of TestAdder is
          Component adder is
                    port(
                         A, B: in std_logic_vector(1 downto 0);
                         Sum: out std_logic_vector(1 downto 0);
                         Cin: in std_logic;
                         Cout: out std_logic);
          End Component;
          signal Alfa, Beta, Somma: std_logic_vector(1 downto 0);
          signal CarryIN, CarryOUT: std_logic;
begin
          DUT: adder port map(Alfa, Beta, Somma, CarryIN, CarryOUT);

          CarryIN <= '0', '1' after 10 ns,  '0' after 25 ns;
          Alfa <= "00", "01" after  5 ns, "11" after 10 ns;
          Beta <= "00", "11" after 15 ns;
end;
```


© 2005 - Fabio Salice

A testbenching

Example




```
entity TestAdder is
end entity TestAdder;

architecture TB of TestAdder is
          Component adder is
                    port(
                         A, B: in std_logic_vector(1 downto 0);
                ... );
       End Component;
          signal Alfa, Beta, Somma: std_logic_vector(1 downto 0);
          signal CarryIN, CarryOUT: std_logic;

begin
          DUT: adder port map(Alfa, Beta, Somma, CarryIN, CarryOUT);
          process is
            begin
                    CarryIN <= '0'; Alfa <= "00"; Beta <="00";        wait for 5  ns;
                    Alfa   <= "01";                                   wait for 5  ns;
                    CarryIN <= '1';                                   wait for 5  ns;
                    Alfa   <= "11"; Beta<= "11";                      wait for 20 ns;
                    CarryIN <= '0';                                   wait;
          end process;
end;
```


© 2005 - Fabio Salice

A testbenching

- To enhance the readability of the testbench any type of variable have to be use. For example, it is possible to generate integer values instead of single bits or bit vectors
  - Wrappers can be built around the DUTs to take care of the type conversion
- VHDL is strongly typed; non-omogeneous data types must be explicitly casted
  - Example of explicit casting:
    - std_logic_vector(*unsigned*): from unsigned to std_logic_vector
    - to_unsigned(*integer, val*): from integer to unsigned
    - unsigned(*std_logic_vector*): from std_logic_vector to unsigned
    - to_integer(*unsigned*): from unsigned to integer

A testbenching

- Example

```
entity TestAdder is
end entity TestAdder;

architecture TB of TestAdder is
        Component adder is
                port(
                … );
         End Component;
        signal AInt, BInt, SumInt: natural;
begin
        DUT: adder port map(A, B, Cin, Sum, Cout);
        A <= std_logic_vector(to_unsigned(Aint, 2));
        B <= std_logic_vector(to_unsigned(Bint, 2));
        SumInt <= to_integer(unsigned(Cout&Sum));
        process is
        begin
                Cin <= '0'; AInt <= 0; BInt <= 0;
                wait for 5 ns;
                AInt <= 1;
                …
```

Data casting wrapper

Integer values

© 2005 - Fabio Salice

A testbenching

- To generate signals with a given periodicity (e.g. modulo 2^n), it is convenient to use a *process*
  - For example: input signals for combinational logic

```
signal A: std_logic_vector(n-1 downto 0);
…
        SignA: process is
        begin
                wait for 10 ns;
                A <= A + 1;
        end process;
```


© 2005 - Fabio Salice

A testbenching

- In the same way, it is also possible to generate periodic scalar signals
  - For example: clock

```
constant PERIOD : time := 10 ns;
signal clock : std_logic := '0';

begin
        clk: process is
        begin
                clock <= not clock after 10 ns;
        end process;
end;
```

Or (another way)

```
clock <= not clock after PERIOD/2;
```


© 2005 - Fabio Salice

A testbenching

- To generate signals with a duty cycle different from 50%, it is necessary to declare all the timing intervals

```
clk: process is
begin
        clock <= '0'; wait for 15 ns;
        clock <= '1'; wait for 5 ns;
end process;
```

Or (another way)

```
clock <= '0' after 3*PERIOD/4 when clock='1' else
         '1' after PERIOD/4   when clock='0' else
         '0';
```

The clock must be initialized

© 2005 - Fabio Salice

A testbenching

- To generate non periodic signals with a process, the `process` must end with the a *`wait`*
  - The `wait` suspend the process indefinitely
- For example: reset, clear

```
rst: process is
begin
        reset <= '1'; wait for 20 ns;
        reset <= '0'; wait for 20 ns; -- active low
        reset <= '1';
        wait;
end process;
```

```
reset <= '1',
         '0' after 20 ns,
         '1' after 40 ns;
```


© 2005 - Fabio Salice

A testbenching

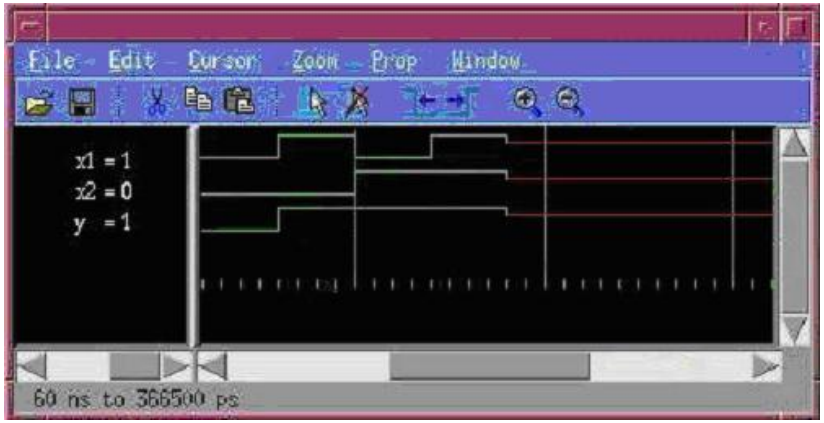
- Finally, to generate stimuli synchronized with other signals, they must be generated within a process with the triggering signal in the sensitivity list

```
SignA_sinc: process (clock) is
Begin
     if (clock=1 and clock'event)
          A <= A+1 after 10 ns;
     end if;
end process;
```


© 2005 - Fabio Salice

A testbenching

- Once the testbench is ready, the simulation produces a waveform that has to be analyzed by the designer to check the correctness of the project



- If waveforms are particularly long and complex, the manual verification could be impracticable and an error-prone process

© 2005 - Fabio Salice

A testbenching primer

- A way to bypass the burden-related issue in the verification process is to analyze some specific conditions involved in the behaviour of the signals
- For this purpose, the `assert` costruct is available
  - `Assert` *condition*
    `Report` *messagge*
    `Severity` *error gravity*
    - The `assert` costruct continuously checks the validity of the condition. If the condition becomes true, it generates the corresponding message together with the chosen level of gravity for the error

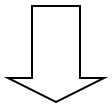
© 2005 - Fabio Salice

A testbenching

- Example:

```
wait for 10 ns;
 x1 <= '1'; x2 <= '1';
 assert y = (x1 xor x2)
      report "E@TB: failure at:"&
          "x1="  & str(x1)&
          " x2=" & str(x2)
      severity Error;
wait for 10 ns;
```

`str()` from the package `txt_util.vhd`



```
# ** Error: E@TB: failure at: x1=1 x2=1
# Time: 40 ns  Iteration: 0 Instance: /tb1
```


© 2005 - Fabio Salice

A testbenching

- It is recommended to use a “standard” format for the error message to ease the identification of the source of the error
- The suggested standard is:
  - A letter to identify the severity:
    - `I`=Information, `W`=Warning, `E`=Error, `F`=Failure
  - Followed by the symbol `@` (or `at`)
  - Finally followed by the name of the entity generating the message

```
# ** Error: E@TB: failure at: x1=1 x2=1
# Time: 40 ns  Iteration: 0 Instance: /tb1
```


© 2005 - Fabio Salice

A testbenching

- To generate dynamic reactions based on the behavior of the DUT (Design Under Test), the following procedure is available:
  - It is suitable only for simple tests

```
process(Data_Bus)
begin
   case Data_Bus is
      when 3      => response <= '1' after 10 ns;
      when others => response <= '0' after 10 ns;
   end case;
end process;
```


© 2005 - Fabio Salice

The structure of a testbench

- Usually, also the testbench code is based on a modular structure
  - The complexity of a testbench is comparable with the complexity of the module to verify
  - The probability of to reuse (at least of a partial reuse) a testbench is reasonably high
- A testbench can contain three different kind of components
  - Models
  - Transactors
  - Bus Functional Models (*BFMs*)

© 2005 - Fabio Salice

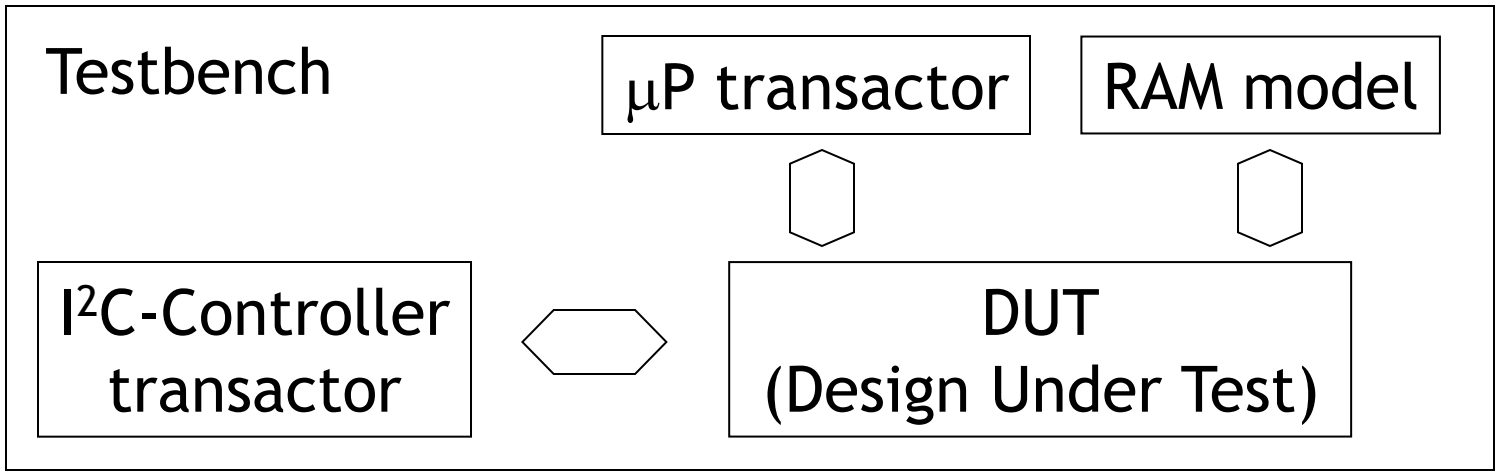
The structure of a testbench

- Model
  - It's the description of a device and it behaves in the same way the device does; its behaviour is controlled only by the stimuli it is feeded with.
    - E.g.: a RAM
- Transactor
  - It's a special type of model with some added control mechanisms such as, for example, the interaction with custom built files containing control data.
    - E.g.: a pre-programmed ROM, or an external device responsible for a protocol
- Bus Functional Model (*BFM*)
  - It's the model of a devices that behaves like a BUS.
    - E.g.: a processor containing only read/write processes, but with no ALU, registers, …

© 2005 - Fabio Salice

The structure of a testbench

- Typical structure of a testbench
  - Example
    - The top-entity (the testbench) instantiates four components:
      - the processor and the I²C-Controller (both transactors),
      - the RAM (a model)
      - and the system that is being verified (DUT)
    - Each component in the testbench produces the stimuli for the DUT and verifies the replies coming from it



© 2005 - Fabio Salice

Transactor and BFM

- To enable the creation of a Transactor or a BFM there is the need to access files in order to output information, and, more frequently, to read data and commands.
- To be able to manage files (and their strings), it is necessary to include this package

```
Use STD.textio.all
```

- As with many other languages, the files must be declared and opened
  - Declaration of a file:

```
file FP : text is in "nome";
```

The access mode can be either `in` or `out`

© 2005 - Fabio Salice

Transactor and BFM

- Process for the generation of file-driven stimuli

```
constant PERIOD  : time := 10 ns;
begin
...
STIMULI : process
  variable L_IN : line;
  variable DATA : std_logic_vector(7 downto 0);
  file STIM_IN : text is in "stim_in.txt";
begin
  W_VALUE <= (others => '0');
  wait for PERIOD;
  while not endfile (STIM_IN) loop
     readline (STIM_IN, L_IN);// reads an entire line from the file
     hread (L_IN, DATA);// reads an hexadecimal value from L_IN and
                        // translates it into a std_logic_vector
     W_VALUE <= DATA;
     wait for PERIOD;
  end loop;
  wait;
end process STIMULI;
```


© 2005 - Fabio Salice

Transactor and BFM

- Another example

```
STIMULI : process
  variable L_IN : line;
  variable DATA1 : std_logic_vector(7 downto 0);
  variable DATA2 : std_logic_vector(7 downto 0);
  variable CHAR : character;
  file STIM_IN : text is in "stim_in.txt";
begin
  while not endfile (STIM_IN) loop
     readline(STIM_IN, L_IN);
     hread(L_IN, DATA1);
     read(L_IN, CHAR);
     hread(L_IN, DATA2);
     W_VALUE <= DATA1&DATA2;
     wait for PERIOD;
  end loop;
  wait;
end process STIMULI;
```

```
File's content
FF FF
FF A1
A0 FF
…
```


© 2005 - Fabio Salice

Transactor and BFM

- Process for the output of the simulation results

```
RESPONSE : process(W_RESULT)
    variable L_OUT : line;
    variable CHAR_SPACE : character := ' ';
    file STIM_OUT : text is out "stim_out.txt";
  begin
    write (L_OUT, now);
    write (L_OUT, CHAR_SPACE);
    write (L_OUT, W_RESULT);
    write (L_OUT, CHAR_SPACE);
    hwrite (L_OUT, W_RESULT);   // hexadecimal
    write (L_OUT, CHAR_SPACE);
    write (L_OUT, W_OVERFLOW);
    writeline (STIM_OUT, L_OUT);// writes to the file
end process RESPONSE;
```


© 2005 - Fabio Salice

Regression Testing

- Tests to verify that a refinement of a design is correct
  - For example that a structural model performs the same as a behavioral one
  - "..*The intent of regression testing is to ensure that changes have not introduced new faults*."
- The test bench includes two instances of the Design Under Test
  - E.g.: behavioral and structural
    - Where *structural* is considered the evolution of the *behavioral*
  - It stimulates both with same inputs
  - It compares the outputs for equality
- The timing differences need to be taken into account

© 2005 - Fabio Salice

Regression Test Example

```
architecture regression of test_bench is
    signal d0, d1, d2, d3, en, clk : bit;
    signal q0a, q1a, q2a, q3a, q0b, q1b, q2b, q3b : bit;
begin
    dut_a : entity reg4_struct
        port map ( d0, d1, d2, d3, en, clk, q0a, q1a, q2a, q3a );
    dut_b : entity reg4_behav
        port map ( d0, d1, d2, d3, en, clk, q0b, q1b, q2b, q3b );
    stimulus: process is
    begin
        d0 <= '1';  d1 <= '1';  d2 <= '1';  d3 <= '1';  wait for 20 ns;
        en <= '0';  clk <= '0';  wait for 20 ns;
        en <= '1';  wait for 20 ns;
        clk <= '1';  wait for 20 ns;
        …
        wait;
    end process stimulus;
    ...
```


© 2005 - Fabio Salice

Regression Test Example

```
    …
    verify : process is
    begin
        wait for 10 ns;
        assert q0a = q0b and q1a = q1b and q2a = q2b and q3a = q3b
            report ”implementations have different outputs”
            severity error;
        wait on d0, d1, d2, d3, en, clk;
    end process verify;
end architecture regression;
```


© 2005 - Fabio Salice

Appendix

- **`str()`**: transforms a std_logic into a string (in `txt_util.vhd`)
- **`hstr()`**: trasforms a std_logic_vector into an hexadecimal string (in `txt_util.vhd`)

© 2005 - Fabio Salice

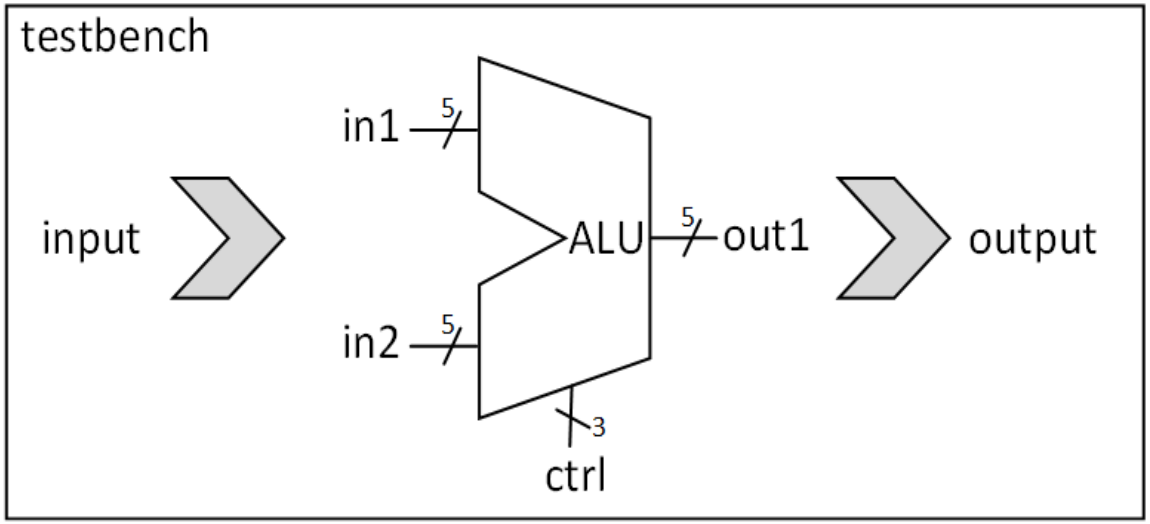
Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Dipartimento di Elettronica e Informazione
Corso di Laurea in Ingegneria Informatica

Estratto da introduzione al VHDL

Prof. Antonio Miele

Esempio di circuito 16

- Vogliamo specificare in VHDL un circuito di test (testbench) per l'esempio 8 (utilizziamo il valore di default per il parametro N)
- Il testbench è un banco di prova da usare durante una simulazione come ambiente che genera gli stimoli per il circuito e raccoglie/analizza gli output



Entity del testbench

- La entity del circuito di esempio 16:

```
library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY esempio16 IS
END esempio16;
```

Entity del testbench

- La entity del circuito di esempio 16:

```
library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY esempio16 IS
END esempio16;
```

Nel testbench possiamo utilizzare altri tipi di dato non sintetizzabili (`file`, `string`, ...)

La entity non contiene alcuna porta

Architecture del testbench

- La architecture del circuito di esempio 16:

```
ARCHITECTURE testbench_arch OF esempio16 IS
  COMPONENT esempio8
    generic (
      N : integer := 5
    );
    port(
      in1, in2: in std_logic_vector(N-1 downto 0);
      ctrl: in std_logic_vector(2 downto 0);
      out1: out std_logic_vector(N-1 downto 0)
    );
  END COMPONENT;

  SIGNAL in1 : std_logic_vector (4 DownTo 0) := "00000";
  SIGNAL in2 : std_logic_vector (4 DownTo 0) := "00000";
  SIGNAL ctrl : std_logic_vector (2 DownTo 0) := "000";
  SIGNAL out1 : std_logic_vector (4 DownTo 0) := "00000";
--...
```

Architecture del testbench

- La architecture del circuito di esempio 16:

```
ARCHITECTURE testbench_arch OF esempio16 IS
  COMPONENT esempio8
    generic (
      N : integer := 5
    );
    port(
      in1, in2: in std_logic_vector(N-1 downto 0);
      ctrl: in std_logic_vector(2 downto 0);
      out1: out std_logic_vector(N-1 downto 0)
    );
  END COMPONENT;

  SIGNAL in1 : std_logic_vector (4 DownTo 0) := "00000";
  SIGNAL in2 : std_logic_vector (4 DownTo 0) := "00000";
  SIGNAL ctrl : std_logic_vector (2 DownTo 0) := "000";
  SIGNAL out1 : std_logic_vector (4 DownTo 0) := "00000";
--...
```

- Componente da testare
- Segnali da connettere alle porte dell'istanza

Architecture del testbench

- La *architecture* del circuito di esempio 16 (seconda parte):

```
--...
BEGIN

  UUT : esempio8
  PORT MAP (
      in1 => in1,
      in2 => in2,
      ctrl => ctrl,
      out1 => out1
  );

--...
```

Architecture del testbench

- La architecture del circuito di esempio 16 (seconda parte):

```
--...
BEGIN

UUT : esempio8
PORT MAP (
    in1 => in1,
    in2 => in2,
    ctrl => ctrl,
    out1 => out1
);

--...
```

- Istanziazione del componente da testare
- Connessione dei segnali che verranno stimolati e letti

Architecture del testbench

- La architecture del circuito di esempio 16 (terza parte):

```
--...
  PROCESS
  BEGIN
    -- -------------  Current Time:  100ns
    WAIT FOR 100 ns;
    in1 <= "00001";
    in2 <= "00100";
    ctrl <= "001";
    -- -------------  Current Time:  300ns
    WAIT FOR 200 ns;
    ctrl <= "010";
    -- -------------  Current Time:  500ns
    WAIT FOR 200 ns;
    ctrl <= "011";
    -- ------------------------------------
    WAIT FOR 1500 ns;
    ASSERT(FALSE) REPORT "Simulation OK." SEVERITY FAILURE;
  END PROCESS;

END testbench_arch;
```

Architecture del testbench

- La architecture del circuito di esempio 16 (terza parte):

```
-- ...
 PROCESS
 BEGIN
   -- -------------  Current Time:  100ns
   WAIT FOR 100 ns;
   in1 <= "00001";
   in2 <= "00100";
   ctrl <= "001";
   -- -------------  Current Time:  300ns
   WAIT FOR 200 ns;
   ctrl <= "010";
   -- -------------  Current Time:  500ns
   WAIT FOR 200 ns;
   ctrl <= "011";
   -- ------------------------------------
   WAIT FOR 1500 ns;
   ASSERT(FALSE) REPORT "Simulation OK." SEVERITY FAILURE;
 END PROCESS;

END testbench_arch;
```

- Nessuna lista di sensibilità
- Il processo è avviato una sola volta a tempo 0

Architecture del testbench

- La architecture del circuito di esempio 16 (terza parte):

```
--...
  PROCESS
  BEGIN
    -- -------------  Current Time:  100ns
    WAIT FOR 100 ns;
    in1 <= "00001";
    in2 <= "00100";
    ctrl <= "001";
    -- -------------  Current Time:  300ns
    WAIT FOR 200 ns;
    ctrl <= "010";
    -- -------------  Current Time:  500ns
    WAIT FOR 200 ns;
    ctrl <= "011";
    -- ------------------------------------
    WAIT FOR 1500 ns;
    ASSERT(FALSE) REPORT "Simulation OK." SEVERITY FAILURE;
  END PROCESS;

END testbench_arch;
```

- I segnali di ingresso vengono forzato ad assumere un dato valore
- L'istruzione `wait` forza l'aggiornamento dei segnali scritti e sospende il processo per un intervallo di tempo specificato

Architecture del testbench

- La architecture del circuito di esempio 16 (terza parte):

```
--...
  PROCESS
  BEGIN
    -- -------------  Current Time:  100ns
    WAIT FOR 100 ns;
    in1 <= "00001";
    in2 <= "00100";
    ctrl <= "001";
    -- -------------  Current Time:  300ns
    WAIT FOR 200 ns;
    ctrl <= "010";
    -- -------------  Current Time:  500ns
    WAIT FOR 200 ns;
    ctrl <= "011";
    -- ---------------------------------
    WAIT FOR 1500 ns;
    ASSERT(FALSE) REPORT "Simulation OK." SEVERITY FAILURE;
  END PROCESS;

END testbench_arch;
```

- In questo testbench non vengono collezionati gli output
- Si usa direttamente il simulatore per disegnare le forme d'onda

Architecture del testbench

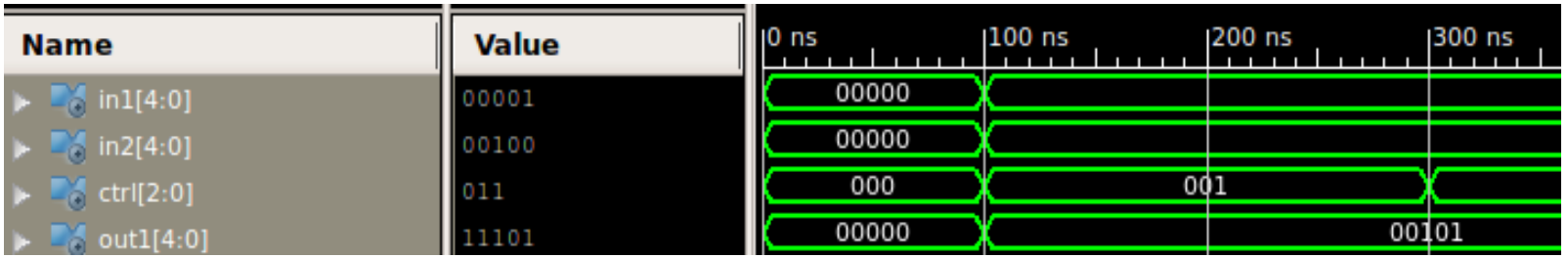
- La architecture del circuito di esempio 16 (terza parte):

```
--...
  PROCESS
  BEGIN
    -- -------------  Current Time:  100ns
    WAIT FOR 100 ns;
    in1 <= "00001";
    in2 <= "00100";
    ctrl <= "001";
    -- -------------  Current Time:  300ns
    WAIT FOR 200 ns;
    ctrl <= "010";
    -- -------------  Current Time:  500ns
    WAIT FOR 200 ns;
    ctrl <= "011";
    -- ------------------------------------
    WAIT FOR 1500 ns;
    ASSERT(FALSE) REPORT "Simulation OK." SEVERITY FAILURE;
  END PROCESS;

END testbench_arch;
```

Ferma la simulazione

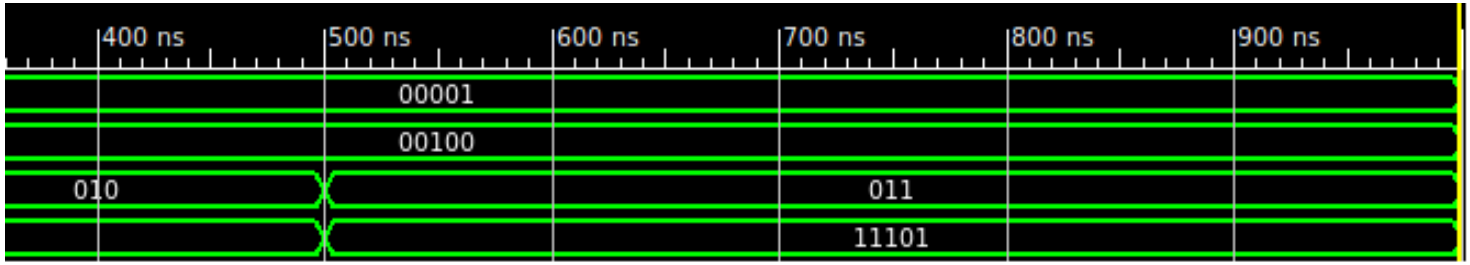
Esecuzione del testbench

- Output del simulatore:



...

...



Architecture del testbench

- La architecture del circuito di esempio 16 (terza parte):

```
--...
  PROCESS
  BEGIN
    -- -------------  Current Time:  100ns
    WAIT FOR 100 ns;
    in1 <= "00001";
    in2 <= "00100";
    ctrl <= "001";
    -- -------------  Current Time:  300ns
    WAIT FOR 200 ns;
    ctrl <= "010";
    -- -------------  Current Time:  500ns
    WAIT FOR 200 ns;
    ctrl <= "011";
    -- ------------------------------------
    WAIT FOR 1500 ns;
    ASSERT(FALSE) REPORT "Simulation OK." SEVERITY FAILURE;
  END PROCESS;

END testbench_arch;
```

ATTENZIONE: questa descrizione non può essere sintetizzata ma solo simulata!

Architecture del testbench

- Implementazione alternativa con lettura ed analisi automatizzata dei risultati nella architecture del circuito di esempio 16:

```
--...
  PROCESS
  BEGIN
    -- -------------  Current Time:  100ns
    WAIT FOR 100 ns;
    in1 <= "00001";
    in2 <= "00100";
    ctrl <= "001";
    WAIT FOR 0 ns;
    ASSERT (out1="00101") REPORT "Simulation Failure." SEVERITY FAILURE;
    WAIT FOR 200 ns;
    --...
  END PROCESS;
```

Architecture del testbench

- Implementazione alternativa con lettura ed analisi automatizzata dei risultati nella architecture del circuito di esempio 16:

```
--...
  PROCESS
  BEGIN
    -- -------------  Current Time:  100ns
    WAIT FOR 100 ns;
    in1 <= "00001";
    in2 <= "00100";
    ctrl <= "001";
    WAIT FOR 0 ns;
    ASSERT (out1="00101") REPORT "Simulation Failure." SEVERITY FAILURE;
    WAIT FOR 200 ns;
    --...
  END PROCESS;
```

- Il processo è sospeso per permettere l'aggiornamento dei segnali
  - Sospendiamo la simulazione per 0 secondi perché il componente testato non porta ritardi

Architecture del testbench

- Implementazione alternativa con lettura ed analisi automatizzata dei risultati nella architecture del circuito di esempio 16:

```
--...
  PROCESS
  BEGIN
    -- -------------  Current Time:  100ns
    WAIT FOR 100 ns;
    in1 <= "00001";
    in2 <= "00100";
    ctrl <= "001";
    WAIT FOR 0 ns;
    ASSERT (out1="00101") REPORT "Simulation Failure." SEVERITY FAILURE;
    WAIT FOR 200 ns;
    --...
  END PROCESS;
```

- Lettura ed analisi dei risultati
- La `assert` bloccherà l'esecuzione nel caso di valore differente da quello atteso

Architecture del testbench

- Implementazione alternativa con lettura ed analisi automatizzata dei risultati nella architecture del circuito di esempio 16:

```
--...
PROCESS
BEGIN
  -- -------------  Current Time:  100ns
  WAIT FOR 100 ns;
  in1 <= "00001";
  in2 <= "00100";
  ctrl <= "001";
  WAIT FOR 0 ns;
  ASSERT (out1="00101") REPORT "Simulation Failure." SEVERITY FAILURE;
  WAIT FOR 200 ns;
  --...
END PROCESS;
```

- I dati di input possono anche essere letti da file ed i risultati scritti su file