**Simple Testbenches**

**Required reading**

- P. Chu, *RTL Hardware Design using VHDL*
  
  *Chapter 2.2.4, Testbenches*

---

**Testbench Defined**

- **Testbench** = VHDL entity that applies stimuli (drives the inputs) to the Design Under Test (DUT) and (optionally) verifies expected outputs.
- The results can be viewed in a waveform window or written to a file.
- Since **Testbench** is written in VHDL, it is not restricted to a single simulation tool (portability).
- The same **Testbench** can be easily adapted to test different implementations (i.e. different architectures) of the same design.

---

**Simple Testbench**

- Processes Generating Stimuli
- Design Under Test (DUT)
- Observed Outputs

---

**Possible sources of expected results used for comparison**

- VHDL Design
- Manual Calculations
- Reference Software Implementation (C, Java, Matlab)
- Tested actual results
- Expected results

---
Testbench
The same testbench can be used to test multiple implementations of the same circuit (multiple architectures)

testbench

---

Testbench Anatomy

ENTITY my_entity_tb IS
  -- Entity has no ports
END my_entity_tb;

ARCHITECTURE behavioral OF tb IS
  -- Local signals and constants
  COMPONENT TestComp -- All Design Under Test component declarations
  PORT ( );
END COMPONENT;

BEGIN
  DUT: TestComp PORT MAP ( );
  testSequence: PROCESS
    -- Input stimuli
  END PROCESS;
END behavioral;

Testbench Anatomy

ENTITY my_entity_tb IS
  -- Entity has no ports
END my_entity_tb;

ARCHITECTURE behavioral OF tb IS
  -- Local signals and constants
  COMPONENT TestComp -- All Design Under Test component declarations
  PORT ( );
END COMPONENT;

BEGIN
  DUT: TestComp PORT MAP ( );
  testSequence: PROCESS
    -- Input stimuli
  END PROCESS;
END behavioral;

Testbench for XOR3 (1)

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY xor3_tb IS END xor3_tb;

ARCHITECTURE behavioral OF xor3_tb IS
  -- Component declaration of the tested unit
  COMPONENT xor3 PORT
    A : IN STD_LOGIC;
    B : IN STD_LOGIC;
    C : IN STD_LOGIC;
    Result : OUT STD_LOGIC
  );
END COMPONENT;

-- Stimulus signals - signals mapped to the input and inout ports of tested entity
SIGNAL test_vector : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL test_result : STD_LOGIC;

BEGIN
  UUT : xor3 PORT MAP
    ( A => test_vector(2), B => test_vector(1), C => test_vector(0), Result => test_result );
  Testing: PROCESS
    test_vector <= "000";
    WAIT FOR 10 ns;
    test_vector <= "001";
    WAIT FOR 10 ns;
    test_vector <= "010";
    WAIT FOR 10 ns;
    test_vector <= "011";
    WAIT FOR 10 ns;
    test_vector <= "100";
    WAIT FOR 10 ns;
    test_vector <= "101";
    WAIT FOR 10 ns;
    test_vector <= "110";
    WAIT FOR 10 ns;
    test_vector <= "111";
    WAIT FOR 10 ns;
  END PROCESS;
END behavioral;

Testbench for XOR3 (2)

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY xor3_tb IS END xor3_tb;

ARCHITECTURE behavioral OF xor3_tb IS
  -- Component declaration of the tested unit
  COMPONENT xor3 PORT
    A : IN STD_LOGIC;
    B : IN STD_LOGIC;
    C : IN STD_LOGIC;
    Result : OUT STD_LOGIC
  );
END COMPONENT;

-- Stimulus signals - signals mapped to the input and inout ports of tested entity
SIGNAL test_vector : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL test_result : STD_LOGIC;

BEGIN
  UUT : xor3 PORT MAP
    ( A => test_vector(2), B => test_vector(1), C => test_vector(0), Result => test_result );
  Testing: PROCESS
    test_vector <= "000";
    WAIT FOR 10 ns;
    test_vector <= "001";
    WAIT FOR 10 ns;
    test_vector <= "010";
    WAIT FOR 10 ns;
    test_vector <= "011";
    WAIT FOR 10 ns;
    test_vector <= "100";
    WAIT FOR 10 ns;
    test_vector <= "101";
    WAIT FOR 10 ns;
    test_vector <= "110";
    WAIT FOR 10 ns;
    test_vector <= "111";
    WAIT FOR 10 ns;
  END PROCESS;
END behavioral;

VHDL Design Styles

dataflow
structural
behavioral

Concurrent statements
Components and interconnects
Sequential statements
• Testbenches

Process without Sensitivity List and its use in Testbenches
What is a PROCESS?

- A process is a sequence of instructions referred to as sequential statements.
- A process can be given a unique name using an optional LABEL.
- This is followed by the keyword PROCESS.
- The keyword BEGIN is used to indicate the start of the process.
- All statements within the process are executed SEQUENTIALLY. Hence, order of statements is important.
- A process must end with the keywords END PROCESS.

Execution of statements in a PROCESS

- The execution of statements continues sequentially till the last statement in the process.
- After execution of the last statement, the control is again passed to the beginning of the process.

PROCESS with a WAIT Statement

- The last statement in the PROCESS is a WAIT instead of WAIT FOR 10 ns.
- This will cause the PROCESS to suspend indefinitely when the WAIT statement is executed.
- This form of WAIT can be used in a process included in a testbench when all possible combinations of inputs have been tested or a non-periodical signal has to be generated.

WAIT FOR vs. WAIT

- WAIT FOR: waveform will keep repeating itself forever.
- WAIT: waveform will keep its state after the last wait instruction.

Time values (physical literals) - Examples

- 7 ns → unit of time most commonly used in simulation.
### Units of time

<table>
<thead>
<tr>
<th>Unit</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Unit</td>
<td></td>
</tr>
<tr>
<td>fs</td>
<td>femtoseconds ((10^{-15} \text{ seconds}))</td>
</tr>
<tr>
<td>Derived Units</td>
<td></td>
</tr>
<tr>
<td>ps</td>
<td>picoseconds ((10^{-12} \text{ seconds}))</td>
</tr>
<tr>
<td>ns</td>
<td>nanoseconds ((10^{-9} \text{ seconds}))</td>
</tr>
<tr>
<td>us</td>
<td>microseconds ((10^{-6} \text{ seconds}))</td>
</tr>
<tr>
<td>ms</td>
<td>milliseconds ((10^{-3} \text{ seconds}))</td>
</tr>
<tr>
<td>sec</td>
<td>seconds</td>
</tr>
<tr>
<td>min</td>
<td>minutes ((60 \text{ seconds}))</td>
</tr>
<tr>
<td>hr</td>
<td>hours ((3600 \text{ seconds}))</td>
</tr>
</tbody>
</table>

---

### Simple Testbenches

**Generating selected values of one input**

```vhdl
SIGNAL test_vector : STD_LOGIC_VECTOR(2 downto 0);
BEGIN
  testing: PROCESS
  BEGIN
    test_vector <= "000";
    WAIT FOR 10 ns;
    test_vector <= "001";
    WAIT FOR 10 ns;
    test_vector <= "010";
    WAIT FOR 10 ns;
    test_vector <= "011";
    WAIT FOR 10 ns;
    test_vector <= "100";
    WAIT FOR 10 ns;
  END PROCESS;
END behavioral;
```

---

**Generating all values of one input**

```vhdl
SIGNAL test_vector : STD_LOGIC_VECTOR(3 downto 0) := "0000";
BEGIN
  testing: PROCESS
  BEGIN
    WAIT FOR 10 ns;
    test_vector <= test_vector + 1;
  end process TESTING;
END behavioral;
```

---

### Arithmetic Operators in VHDL (1)

To use basic arithmetic operations involving `std_logic_vectors` you need to include the following library packages:

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
or
USE ieee.std_logic_signed.all;
```

---

### Arithmetic Operators in VHDL (2)

You can use standard `+`, `-` operators to perform addition and subtraction:

```vhdl
signal A : STD_LOGIC_VECTOR(3 downto 0);
signal B : STD_LOGIC_VECTOR(3 downto 0);
signal C : STD_LOGIC_VECTOR(3 downto 0);

C <= A + B;
```
Different ways of performing the same operation

signal count: std_logic_vector(7 downto 0);

You can use:

- `count <= count + "00000001";`
- `count <= count + 1;`
- `count <= count + '1';`

Different declarations for the same operator

Declarations in the package ieee.std_logic_unsigned:

- Function `+` (L: std_logic_vector; R: std_logic_vector) return std_logic_vector;
- Function `+` (L: std_logic_vector; R: integer) return std_logic_vector;
- Function `+` (L: std_logic_vector; R: std_logic) return std_logic_vector;

Operator overloading

- Operator overloading allows different argument types for a given operation (function)
- The VHDL tools resolve which of these functions to select based on the types of the inputs
- This selection is transparent to the user as long as the function has been defined for the given argument types.

Generating all possible values of two inputs

```vhdl
SIGNAL test_ab : STD_LOGIC_VECTOR(1 downto 0);
SIGNAL test_sel : STD_LOGIC_VECTOR(1 downto 0);
BEGIN
  ....
  double_loop: PROCESS
  BEGIN
    test_ab <= "00";
    test_sel <= "00";
    for I in 0 to 3 loop
      for J in 0 to 3 loop
        wait for 10 ns;
        test_ab <= test_ab + 1;
        test_sel <= test_sel + 1;
      end loop;
    end loop;
  END PROCESS;
  ....
END behavioral;
```

Generating periodical signals, such as clocks

```vhdl
CONSTANT clk1_period : TIME := 20 ns;
CONSTANT clk2_period : TIME := 200 ns;
SIGNAL clk1 : STD_LOGIC;
SIGNAL clk2 : STD_LOGIC := '0';
BEGIN
  ....
  clk1_generator: PROCESS
  clk1 <= '0';
  WAIT FOR clk1_period/2;
  clk1 <= '1';
  WAIT FOR clk1_period/2;
  END PROCESS;
  END behavioral;
```

Generating one-time signals, such as resets

```vhdl
CONSTANT reset1_width : TIME := 100 ns;
CONSTANT reset2_width : TIME := 150 ns;
SIGNAL reset1 : STD_LOGIC;
SIGNAL reset2 : STD_LOGIC := '1';
BEGIN
  ....
  reset1_generator: PROCESS
  reset1 <= '1';
  WAIT FOR reset1_width;
  reset1 <= '0';
  WAIT;
  END PROCESS;
  reset2_generator: PROCESS
  WAIT FOR reset2_width;
  reset2 <= '0';
  WAIT;
  END PROCESS;
  END behavioral;
```
Typical error

```vhdl
SIGNAL test_vector : STD_LOGIC_VECTOR(2 downto 0);
SIGNAL reset : STD_LOGIC;
BEGIN
  generator1: PROCESS
  reset <= '1';
  WAIT FOR 100 ns
  reset <= '0';
  test_vector <= "000";
  WAIT;
  END PROCESS;
  generator2: PROCESS
  WAIT FOR 200 ns
  test_vector <= "001";
  WAIT FOR 600 ns
  test_vector <= "011";
  END PROCESS;
END behavioral;
```