Lecture 2

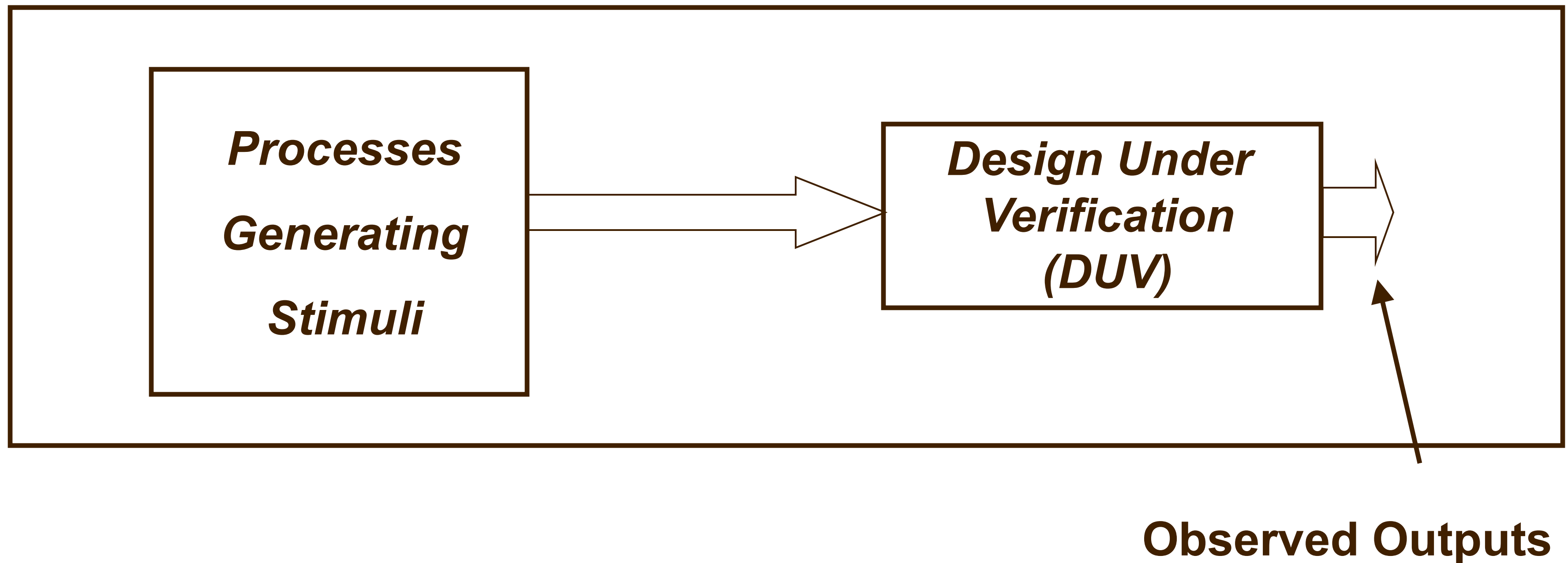
Introduction to Testbenches

Reading

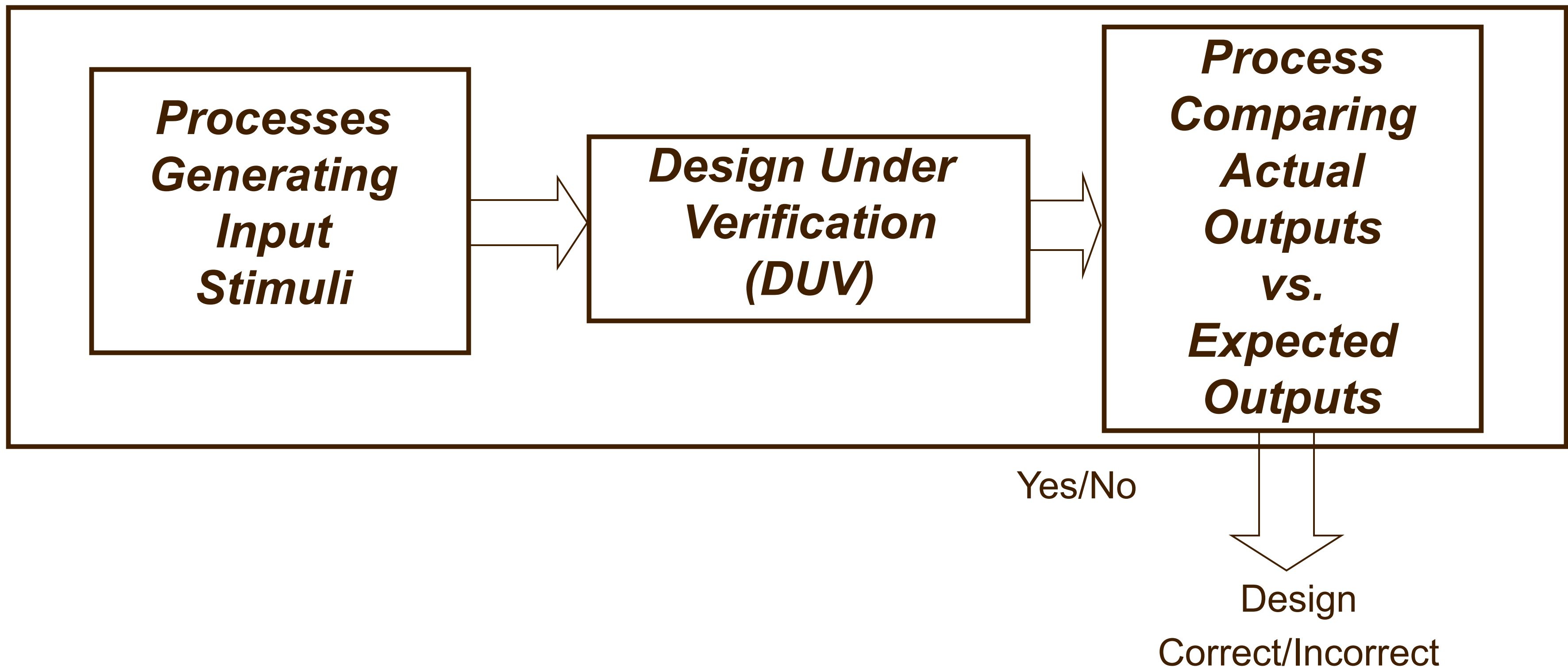
- P. Chu, *FPGA Prototyping by VHDL Examples*

  Chapter 1.5 Testbench
  Chapter 4.4 Testbench for sequential circuits

Simple Testbench



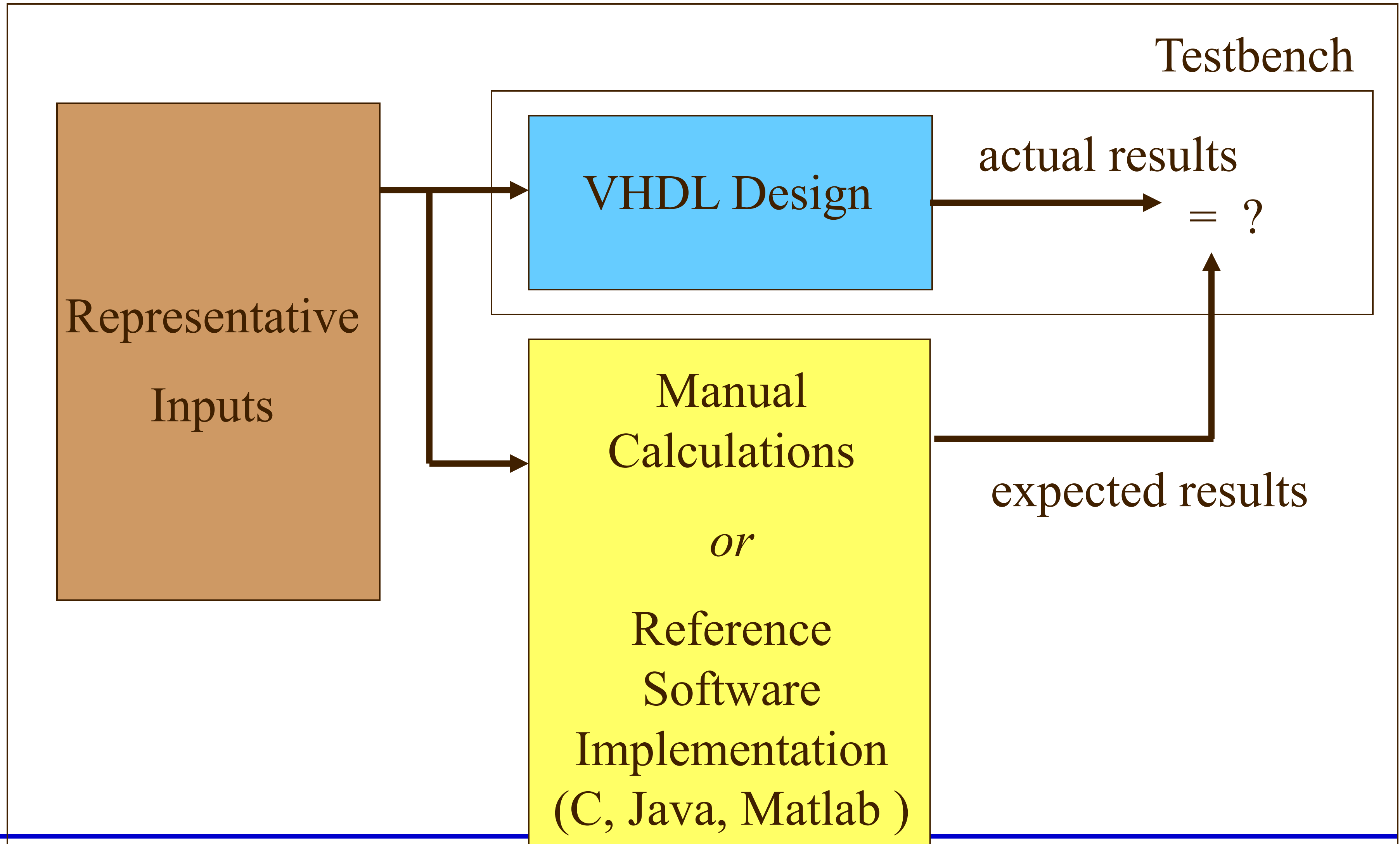
Advanced Testbench



Testbench Defined

- *Testbench* = VHDL entity that applies stimuli (drives the inputs) to the Design Under Verification (DUV) 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.

Possible sources of expected results used for comparison



Test vectors

Set of pairs: {Input Values i, Expected Outputs Values i}

Input Values 1, Expected Output Values 1

Input Values 2, Expected Output Values 2

……………………………

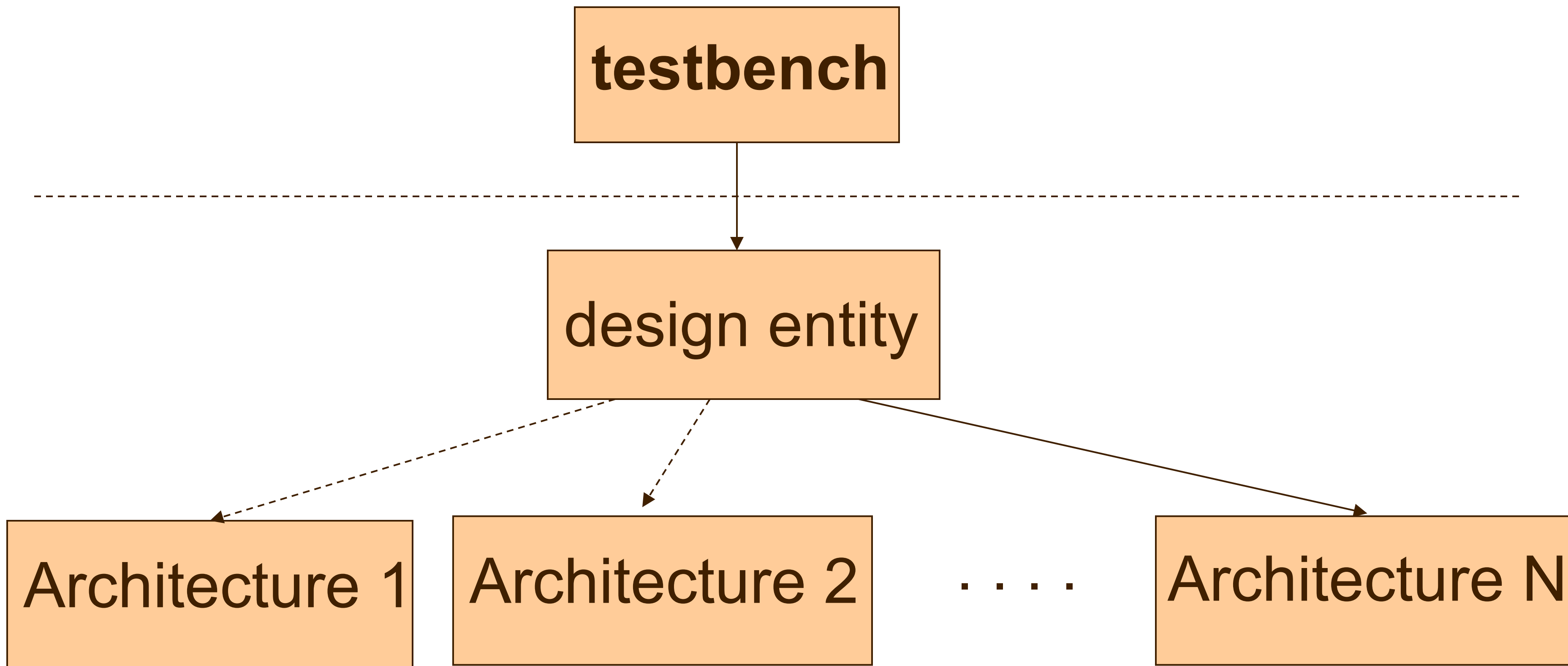
Input Values N, Expected Output Values N

Test vectors can cover either:

- all combinations of inputs (for very simple circuits only)
- selected representative combinations of inputs (most realistic circuits)

Testbench

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



Anatomy of a Simple Testbench

```
ENTITY my_entity_tb IS
              --TB entity has no ports
END my_entity_tb;

ARCHITECTURE behavioral OF my_entity_tb IS

     --Local signals and constants
-----------------------------------------------------
BEGIN
  DUV: entity work.TestComp(dataflow) PORT MAP(      -- Instantiations of DUVs
                      );

  testSequence: PROCESS
                            }   -- Input stimuli
  END PROCESS;


END behavioral;
```

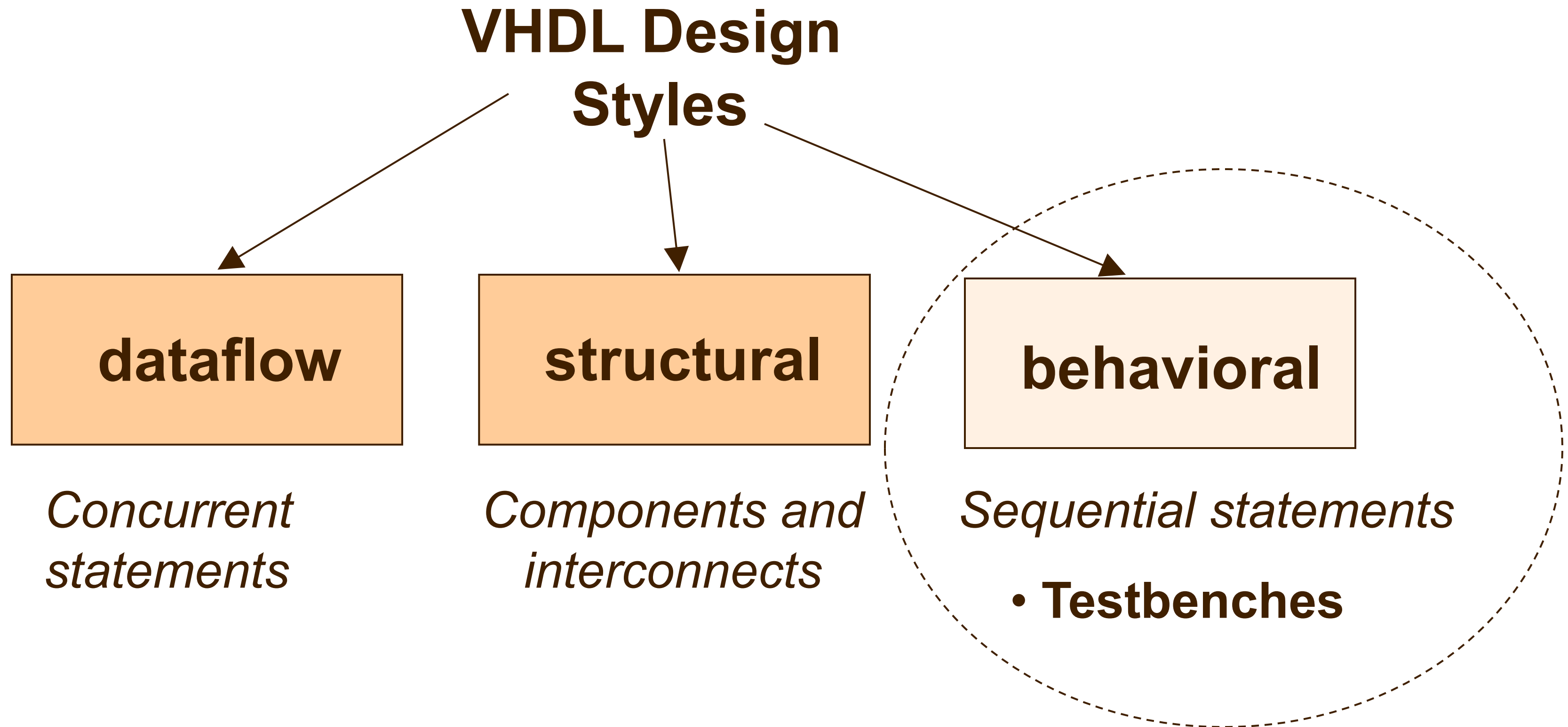
Testbench for XOR3 (1)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY xor3_tb IS
END xor3_tb;

ARCHITECTURE behavioral OF xor3_tb IS

-- 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;
```

Testbench for XOR3 (2)

```
BEGIN
DUV : entity work.xor3(dataflow)
 PORT MAP (
                A => test_vector(2),
                B => test_vector(1),
                C => test_vector(0),
                Result => test_result);
                                      );
 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;
    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



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.

The keyword PROCESS

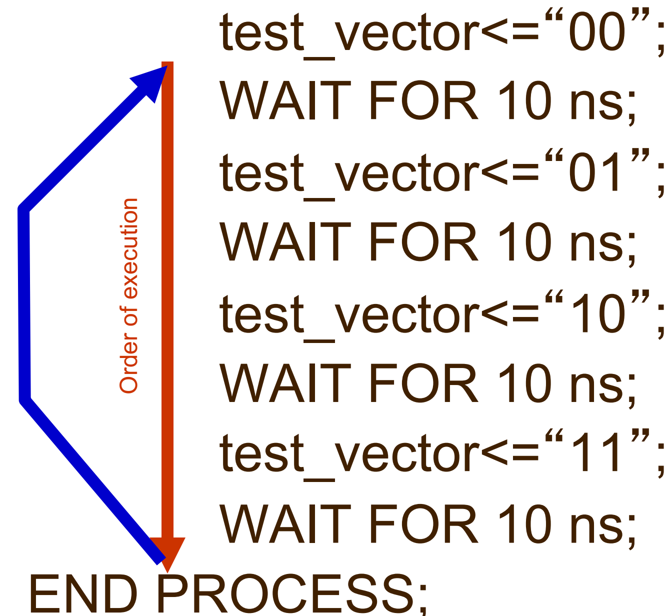
- A process can be given a unique name using an optional LABEL
- This is followed by the keyword PROCESS
- The keyword BEGIN is used to indicate the start of the process
- All statements within the process are executed SEQUENTIALLY. Hence, order of statements is important.
- A process must end with the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

```
Testing: PROCESS
BEGIN
        test_vector<="00";
        WAIT FOR 10 ns;
        test_vector<="01";
        WAIT FOR 10 ns;
        test_vector<="10";
        WAIT FOR 10 ns;
        test_vector<="11";
        WAIT FOR 10 ns;
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.

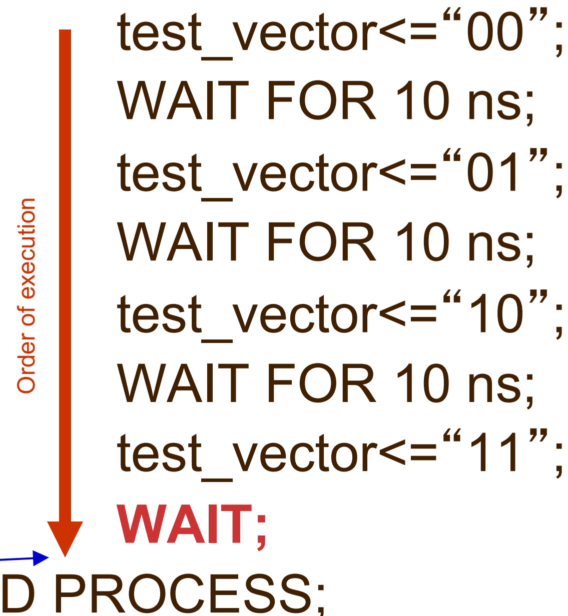
Testing: PROCESS
BEGIN



Program control is passed to the first statement after BEGIN

PROCESS with a WAIT Statement

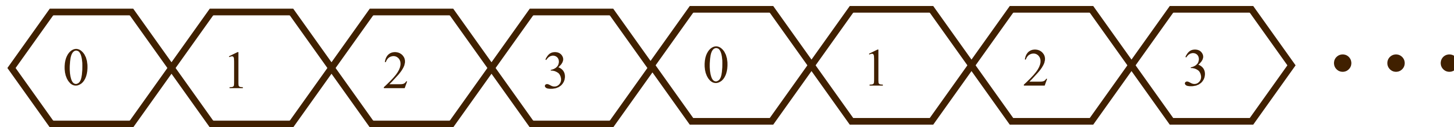
- The last statement in the PROCESS is a WAIT instead of WAIT FOR 10 ns.
- This will cause the PROCESS to suspend indefinitely when the WAIT statement is executed.
- This form of WAIT can be used in a process included in a testbench when all possible combinations of inputs have been tested or a non-periodical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



Program execution stops here

WAIT FOR vs. WAIT

WAIT FOR: waveform will keep repeating itself forever



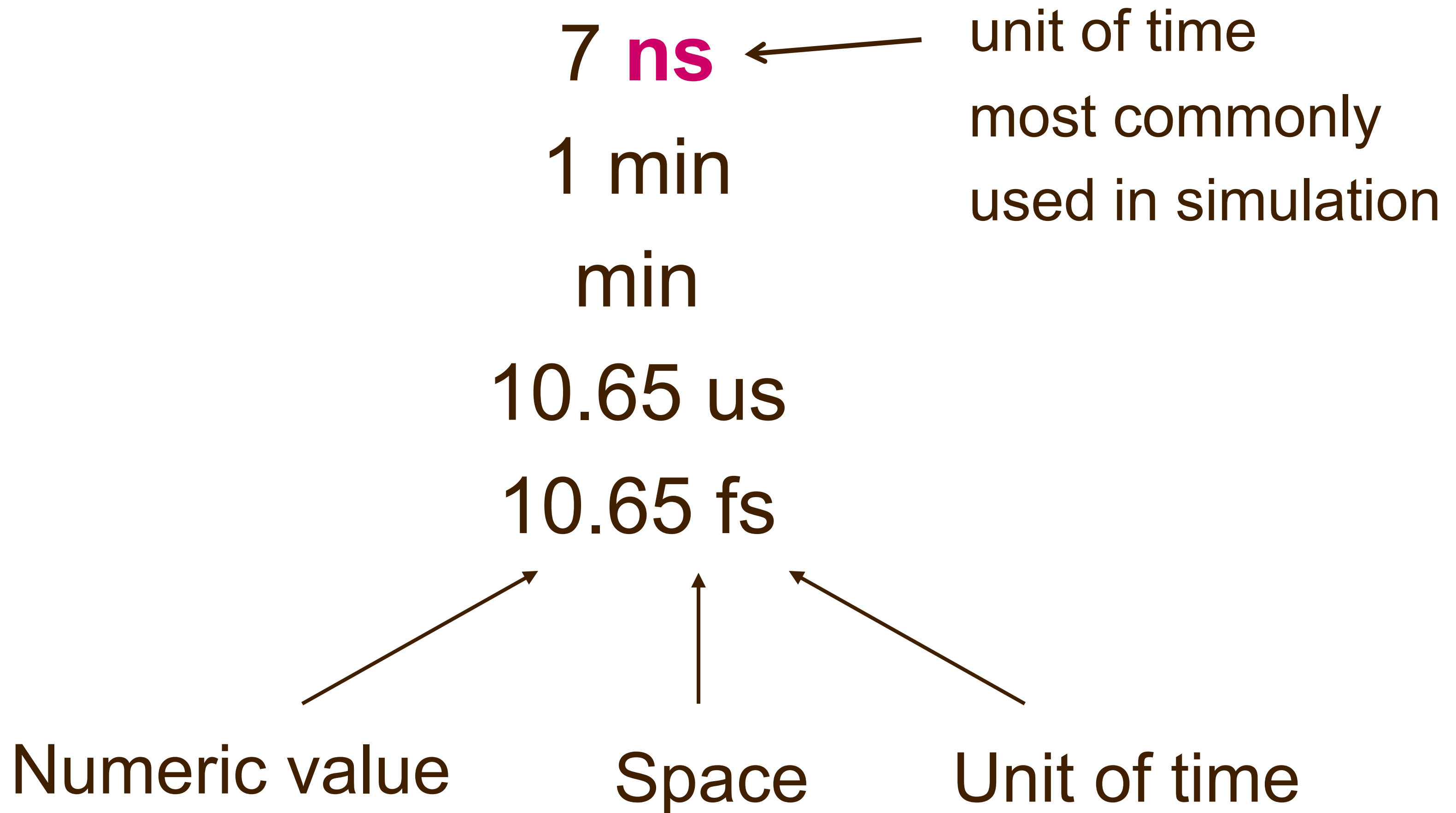
WAIT : waveform will keep its state after the last wait instruction.

WAIT UNTIL

```
Testing: PROCESS
BEGIN
        …..
        WAIT UNTIL done= '1';
        …..

        …..
        WAIT UNTIL falling_edge(clk);
        d <= ”0001”;

END PROCESS;
```

Specifying time in VHDL

Time values (physical literals) - Examples



Units of time

Unit	Definition
Base Unit	
fs	femtoseconds (10^{-15} seconds)
Derived Units	
ps	picoseconds (10^{-12} seconds)
ns	**nanoseconds (10^{-9} seconds)**
us	microseconds (10^{-6} seconds)
ms	miliseconds (10^{-3} seconds)
sec	seconds
min	minutes (60 seconds)
hr	hours (3600 seconds)

Simple Testbenches

Generating selected values of one input

```
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

```
use IEEE.NUMERIC_STD.ALL;

SIGNAL test_vector : STD_LOGIC_VECTOR(3 downto 0):="0000";

BEGIN
    .......

    testing: PROCESS
    BEGIN
            WAIT FOR 10 ns;
            test_vector <= std_logic_vector(unsigned(test_vector) + 1);
    end process TESTING;

    ........
END behavioral;
```

Different ways of performing the same operation

signal count: unsigned(7 downto 0);

You can use:

count <= count + “00000001”;

or

count <= count + 1;

Different declarations for the same operator

Declarations in the package numeric_std:

```
function "+" ( L: unsigned; R: unsigned)
                return unsigned;

function "+" ( L: unsigned; R: natural)
                return unsigned;

function "+" ( L: natural; R: unsigned)
                return unsigned;
```

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

```
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 <= std_logic_vector(unsigned(test_ab) + 1);
                          end loop;
                          test_sel <= std_logic_vector(unsigned(test_sel) + 1);
                        end loop;
                END PROCESS;
        ........
        END behavioral;
```

Generating periodical signals, such as clocks

```
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;

    clk2 <= not clk2  after  clk2_period/2;
  .......
END behavioral;
```

Generating one-time signals, such as resets

```
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;
```

Generating one-time signals, such as resets

```
CONSTANT reset3_width : TIME := 200 ns;
SIGNAL reset3 : STD_LOGIC;

BEGIN
    .......
            reset3 <= '1', '0' after reset3_width;
    .......
END behavioral;
```

Typical error

```
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;
```

Records

Records

```
TYPE test_vector IS RECORD
  operation : STD_LOGIC_VECTOR(1 DOWNTO 0);
  a : STD_LOGIC;
  b: STD_LOGIC;
  y : STD_LOGIC;
END RECORD;

CONSTANT num_vectors : INTEGER := 16;

TYPE test_vectors IS ARRAY (0 TO num_vectors-1) OF test_vector;

CONSTANT and_op   : STD_LOGIC_VECTOR(1 DOWNTO 0) := "00";
CONSTANT or_op    : STD_LOGIC_VECTOR(1 DOWNTO 0) := "01";
CONSTANT xor_op   : STD_LOGIC_VECTOR(1 DOWNTO 0) := "10";
CONSTANT xnor_op  : STD_LOGIC_VECTOR(1 DOWNTO 0) := "11";
```

Records

```
CONSTANT test_vector_table: test_vectors :=(
 (operation => AND_OP, a=>'0', b=>'0', y=>'0'),
 (operation => AND_OP, a=>'0', b=>'1', y=>'0'),
 (operation => AND_OP, a=>'1', b=>'0', y=>'0'),
 (operation => AND_OP, a=>'1', b=>'1', y=>'1'),
 (operation => OR_OP, a=>'0', b=>'0', y=>'0'),
 (operation => OR_OP, a=>'0', b=>'1', y=>'1'),
 (operation => OR_OP, a=>'1', b=>'0', y=>'1'),
 (operation => OR_OP, a=>'1', b=>'1', y=>'1'),
 (operation => XOR_OP, a=>'0', b=>'0', y=>'0'),
 (operation => XOR_OP, a=>'0', b=>'1', y=>'1'),
 (operation => XOR_OP, a=>'1', b=>'0', y=>'1'),
 (operation => XOR_OP, a=>'1', b=>'1', y=>'0'),
 (operation => XNOR_OP, a=>'0', b=>'0', y=>'1'),
 (operation => XNOR_OP, a=>'0', b=>'1', y=>'0'),
 (operation => XNOR_OP, a=>'1', b=>'0', y=>'0'),
 (operation => XNOR_OP, a=>'1', b=>'1', y=>'1')
);
```

Variables

Variables - features

- Can only be declared within processes and subprograms (functions & procedures)
- Initial value can be explicitly specified in the declaration
- When assigned take an assigned value immediately
- Variable assignments represent the desired behavior, not the structure of the circuit
- Can be used freely in testbenches
- Should be avoided, or at least used with caution in a synthesizable code

Variables - Example

```
testing: PROCESS

VARIABLE error_cnt: INTEGER := 0;

BEGIN
   FOR i IN 0 to num_vectors-1 LOOP
         test_operation <= test_vector_table(i).operation;
         test_a <= test_vector_table(i).a;
         test_b <= test_vector_table(i).b;
         WAIT FOR 10 ns;
         IF test_y /= test_vector_table(i).y THEN
                  error_cnt := error_cnt + 1;
         END IF;
   END LOOP;
END PROCESS testing;
```

Variables - Example

In the declarative portion of the architecture:

```
SIGNAL test_operation : STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL test_a : STD_LOGIC;
SIGNAL test_b : STD_LOGIC;
SIGNAL test_y : STD_LOGIC;
```

Between BEGIN and END

```
  DUT: entity work.ALU(dataflow)
          PORT MAP (operation => test_operation,
                    a => test_a,
                    b => test_b,
                    y  => test_y);
```

Asserts & Reports

Assert

Assert is a **non-synthesizable** statement which purpose is to write out messages on the screen when problems are found during simulation.

Depending on the **severity of the problem**, The simulator is instructed to continue simulation or halt.

Assert - syntax

ASSERT condition
[REPORT "message"]
[SEVERITY severity_level];

The message is written when the condition is FALSE.

Severity_level can be:
Note, Warning, Error (default), or Failure.

Assert – Examples (1)

```
assert initial_value <= max_value
  report "initial value too large"
  severity error;

assert packet_length /= 0
  report "empty network packet received"
  severity warning;

assert false
  report "Initialization complete"
  severity note;
```

Assert – Examples (2)

```
stim_proc: process
  begin
        wait for 20 ns
        assert false
           report "PASSED CHECKPOINT 1"
           severity note;
        wait for 10 ns;
        assert false
           report "PASSED CHECKPOINT 2"
           severity warning;
```

Assert – Examples (3)

```
            wait for 10 ns;
            assert false
               report "PASSED CHECKPOINT 3"
               severity error;
            wait for 10 ns;
            assert false
               report "PASSED CHECKPOINT 4"
            severity failure;
            wait;
end process;
```

Format of messages in Vivado Simulator

File Edit Flow Tools Window Layout View Run Help Quick Access Ready

Flow Navigator — SIMULATION - Behavioral Simulation - Functional - sim_1 - Assert_Examples

- PROJECT MANAGER
  - Settings
  - Add Sources
  - Language Templates
  - IP Catalog
- IP INTEGRATOR
  - Create Block Design
  - Open Block Design
  - Generate Block Design
- SIMULATION
  - Run Simulation
- RTL ANALYSIS
- SYNTHESIS
  - Run Synthesis
  - Open Synthesized Design
- IMPLEMENTATION
  - Run Implementation
  - Open Implemented Design
- PROGRAM AND DEBUG
  - Generate Bitstream
  - Open Hardware Manager

Assert_Examples.vhd — /nhome/ffarahma/projects/vivado20172/545_advancedtestbench/545_advancedtestbench.srcs/sim_1/new/Assert_Examples.vhd

```
11
12  stim_proc: process
13      begin
14          wait for 20 ns;
15          assert false
16              report "PASSED CHECKPOINT 1"
17          severity note;
18          wait for 10 ns;
19          assert false
20              report "PASSED CHECKPOINT 2"
21          severity warning;
22          wait for 10 ns;
23          assert false
24              report "PASSED CHECKPOINT 3"
25          severity error;
26          wait for 10 ns;
27          assert false
28              report "PASSED CHECKPOINT 4"
29          severity failure;
30          wait;
31      end process;
32
33  end Behavioral;
34
```

Tcl Console

```
#     set_property needs_save false [current_wave_config]
#   } else {
#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a wave window. If you want to open a wave window go to 'File->New Waveform Configuration' or type 'create_wave_config' in the TC
#   }
# }
WARNING: [Simtcl 6-168] No object found for the given pattern.
WARNING: [Add_Wave-1] No top level signals found. Simulator will start without a wave window. If you want to open a wave window go to 'File->New Waveform Configuration' or type 'create_wave_config' in the TCL console.
# run 1000ns
Note: PASSED CHECKPOINT 1
Time: 20 ns  Iteration: 0  Process: /Assert_Examples/stim_proc  File: /nhome/ffarahma/projects/vivado20172/545_advancedtestbench/545_advancedtestbench.srcs/sim_1/new/Assert_Examples.vhd
Warning: PASSED CHECKPOINT 2
Time: 30 ns  Iteration: 0  Process: /Assert_Examples/stim_proc  File: /nhome/ffarahma/projects/vivado20172/545_advancedtestbench/545_advancedtestbench.srcs/sim_1/new/Assert_Examples.vhd
Error: PASSED CHECKPOINT 3
Time: 40 ns  Iteration: 0  Process: /Assert_Examples/stim_proc  File: /nhome/ffarahma/projects/vivado20172/545_advancedtestbench/545_advancedtestbench.srcs/sim_1/new/Assert_Examples.vhd
Failure: PASSED CHECKPOINT 4
Time: 50 ns  Iteration: 0  Process: /Assert_Examples/stim_proc  File: /nhome/ffarahma/projects/vivado20172/545_advancedtestbench/545_advancedtestbench.srcs/sim_1/new/Assert_Examples.vhd
$finish called at time : 50 ns : File "/nhome/ffarahma/projects/vivado20172/545_advancedtestbench/545_advancedtestbench.srcs/sim_1/new/Assert_Examples.vhd" Line 29
INFO: [USF-XSim-96] XSim completed. Design snapshot 'Assert_Examples_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

Report - syntax

REPORT "message"
  [SEVERITY severity_level];

The message is always written.

Severity_level can be:

Note (default), Warning, Error, or Failure.

Report - Examples

```
report "Initialization complete";

report  "Incorrect branch"  severity error;

report  "Current time = " & time'image(now);
```

The concatenation operator

Attribute of the type time converting value of the type time into a string

Function returning the current simulation time as a value of the type time

Report - Examples

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity example_1_tb is
end example_1_tb;

architecture behavioral of example_1_tb is
    signal clk : std_logic := '0';
begin
    clk <= not clk after 100 ns;
    process
    begin
                wait for 1000 ns;
                report "Initialization complete";
                report  "Current time = " & time'image(now);
                wait for 1000 ns;
                report "SIMULATION COMPLETED"  severity failure;
    end process;
end behavioral;
```