VietNam National University
University of Engineering and Technology

Functional Verification

TS. Nguyễn Kiêm Hùng
Email: kiemhung@vnu.edu.vn

Laboratory for Smart Integrated Systems

Objectives

In this lecture you will be introduced to:

- Key concepts of functional verification
  - **Verification by Simulation**
- Putting VHDL to service for hardware simulation
  - Writing a Test Bench by using VHDL
  - Modular testbench design for reusing throughout a design cycle
- Using EDA tools for circuit and synthesis such as ModelSim, Xilinx Vivado Design Suite, etc.

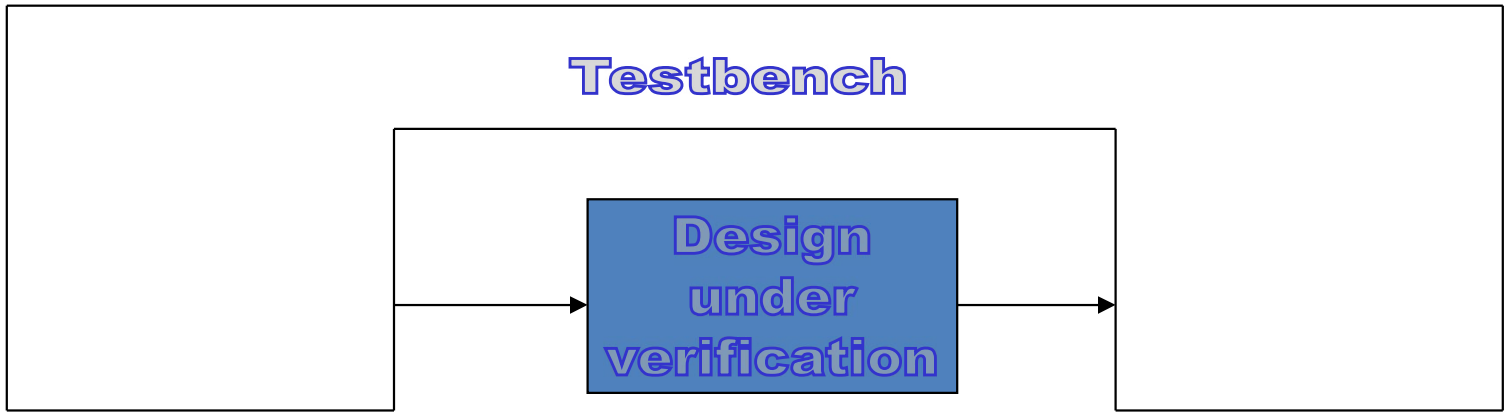
Verification

What is verification?

- Verification is the process of checking whether a design meets the specification and performance goals.
- The ultimate goal of design verification is to avoid the manufacturing and deployment of flawed designs.
- **Notice:**
  - Verification **is not** a testbench or a series of testbenches.
  - Verification **does not** insure that the specification is correct. It is the act of ensuring that the logic design conforms to the specifications.

Verification testbenches

- In verification the testbench provides the inputs and monitors the outputs.



- The challenge of verification is to determine what input patterns to supply and what is the expected output of a properly working design

Functional vs. Parametric Verification

- **Functionality**
  - describes what response a circuit produces at the output ports when presented with some stimuli at the input ports (aka logic behavior, input-to-output mapping)…
  - Expressed in terms of algorithms, equations, state graphs, truth tables and the like.
- **Parametric characteristics** relate to physical quantities measured in units such as Mbit/s, ns, V, A, mW, pF, etc.

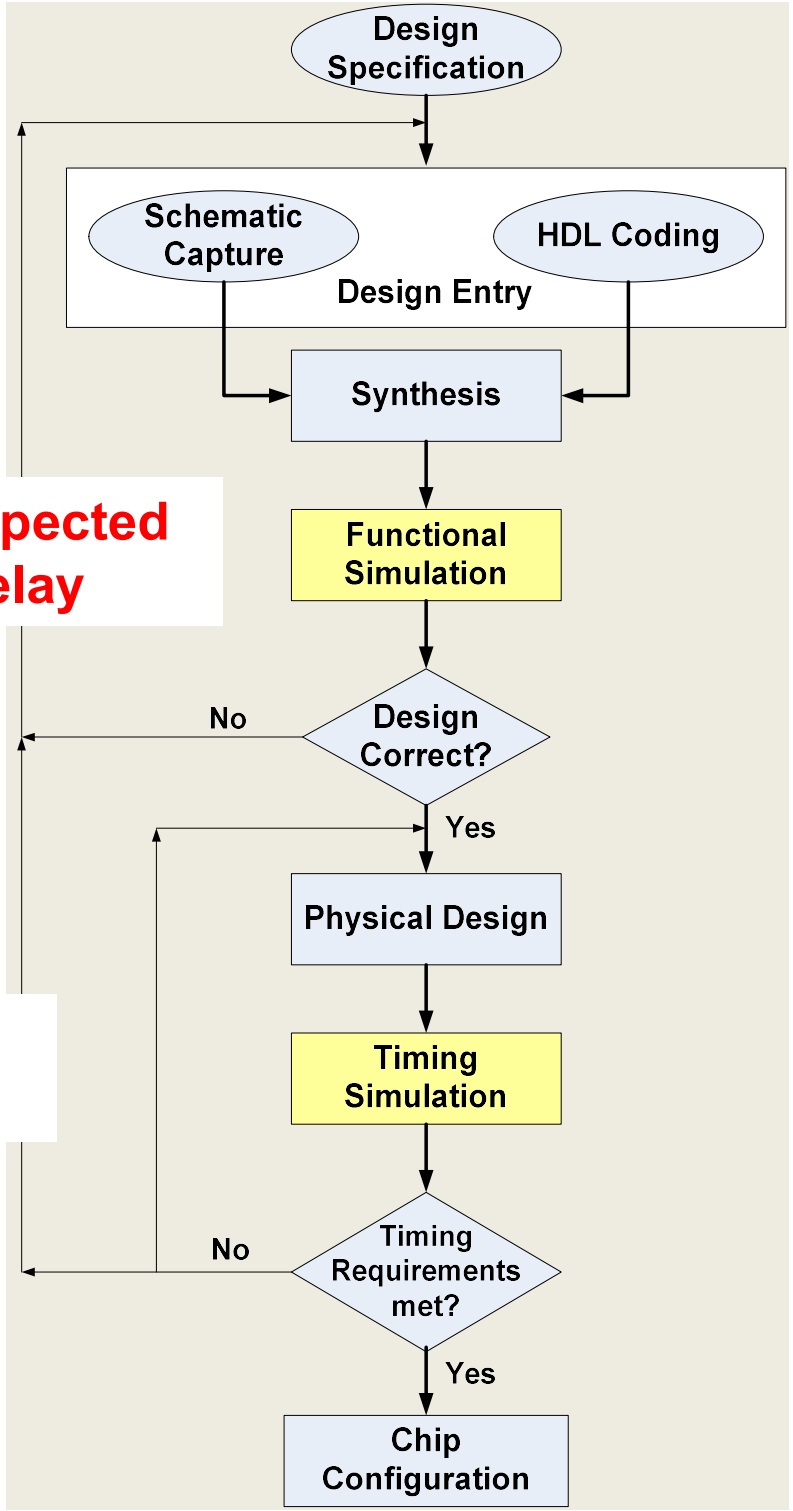
A design's functionality and its parametric characteristics are best checked separately since goals, methods, and tools are quite different.

Simulation

Review:

to verify that design will function as expected without considering propagation delay

to determine if the generated circuit meets the timing requirements

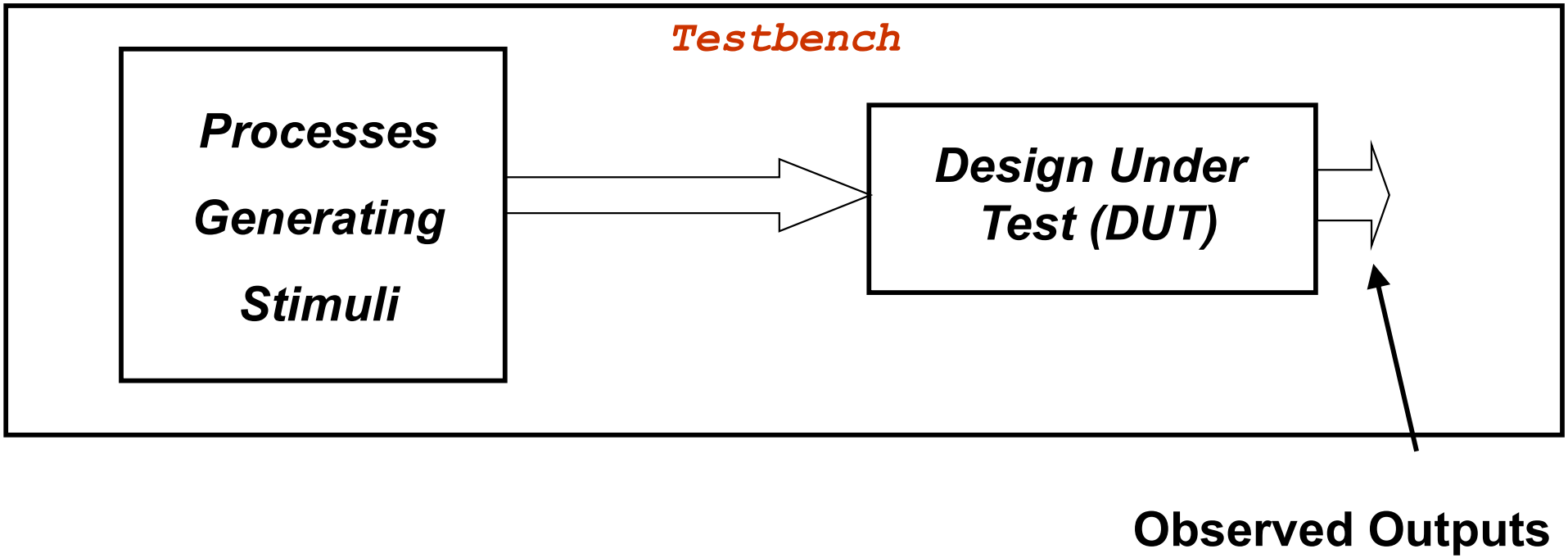


SIMULATION

- Process of verifying the functional, timing characteristics of imitative models at any level of abstraction.
- Use **Simulators** (e.g. ModelSim) to simulate the **Hardware models** (i.e. user design in VHDL) with a **Testbench** (which generates clk, reset and the required test vectors etc.)

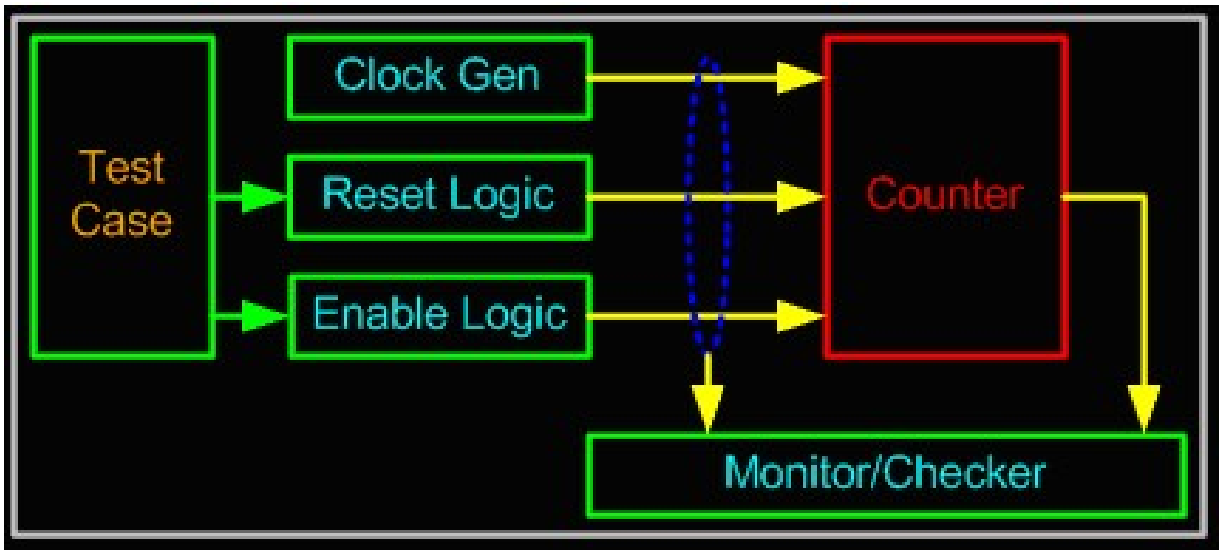
TESTBENCHES

Testbench Block Diagram

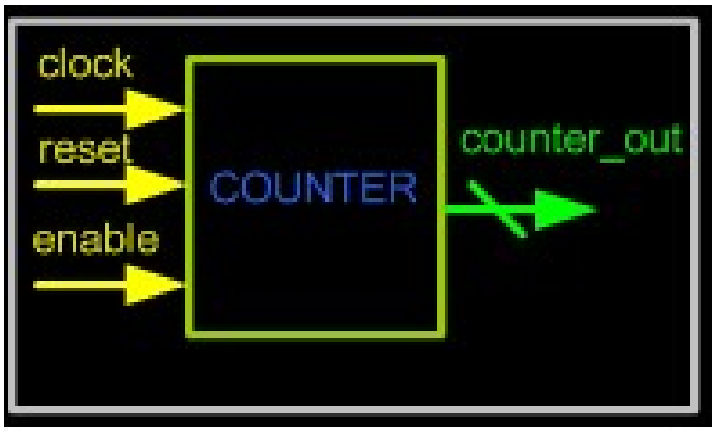


Testbench Block Diagram

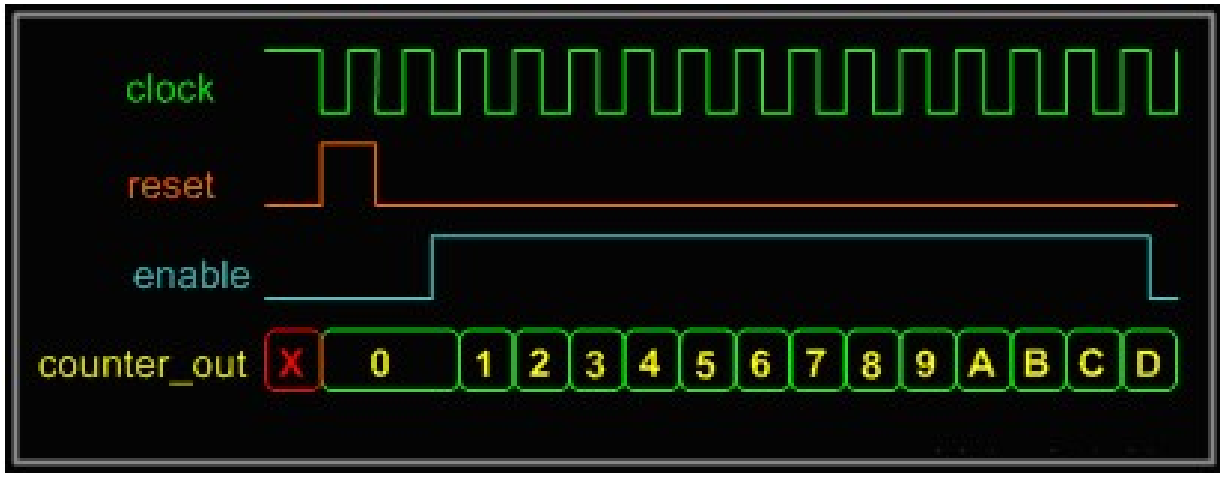
Counter Test Bench



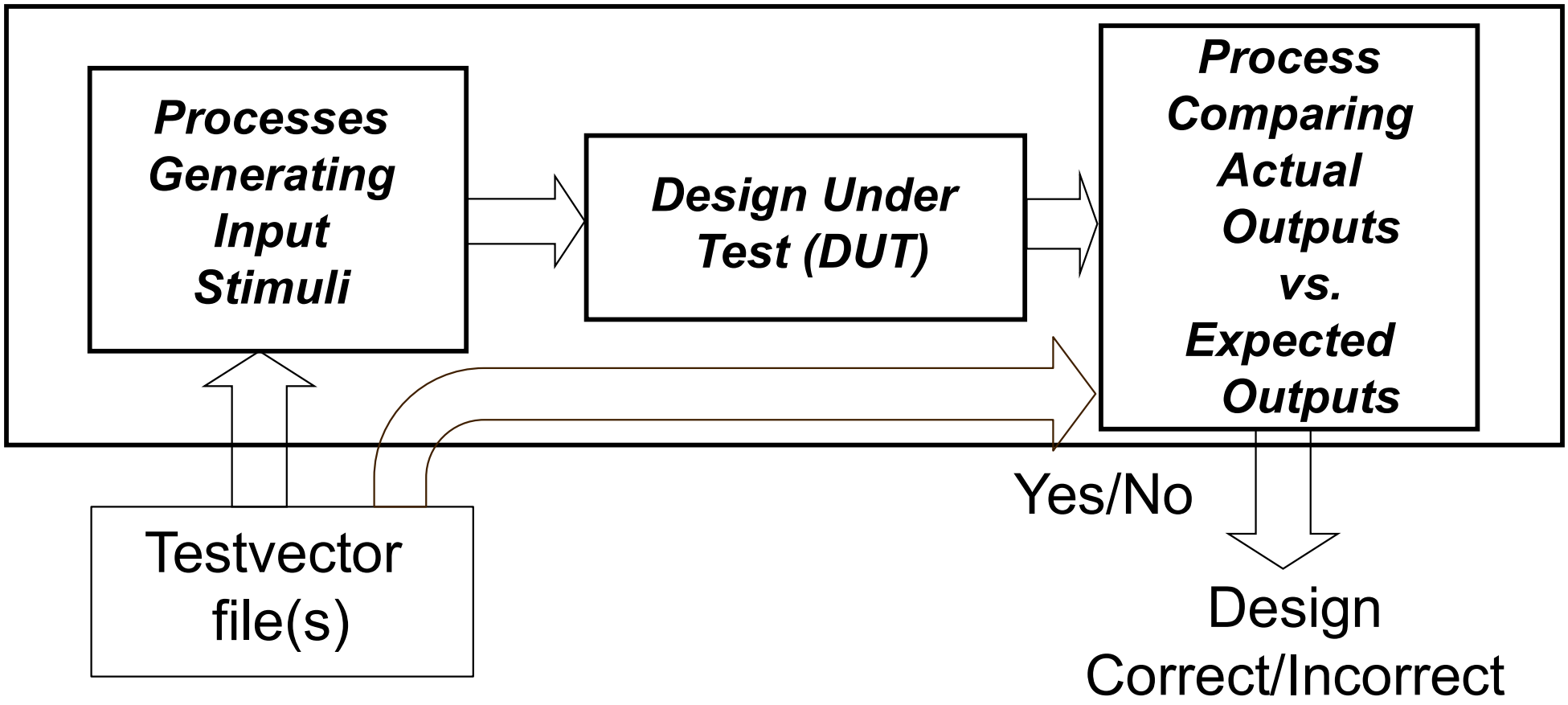
Counter Design



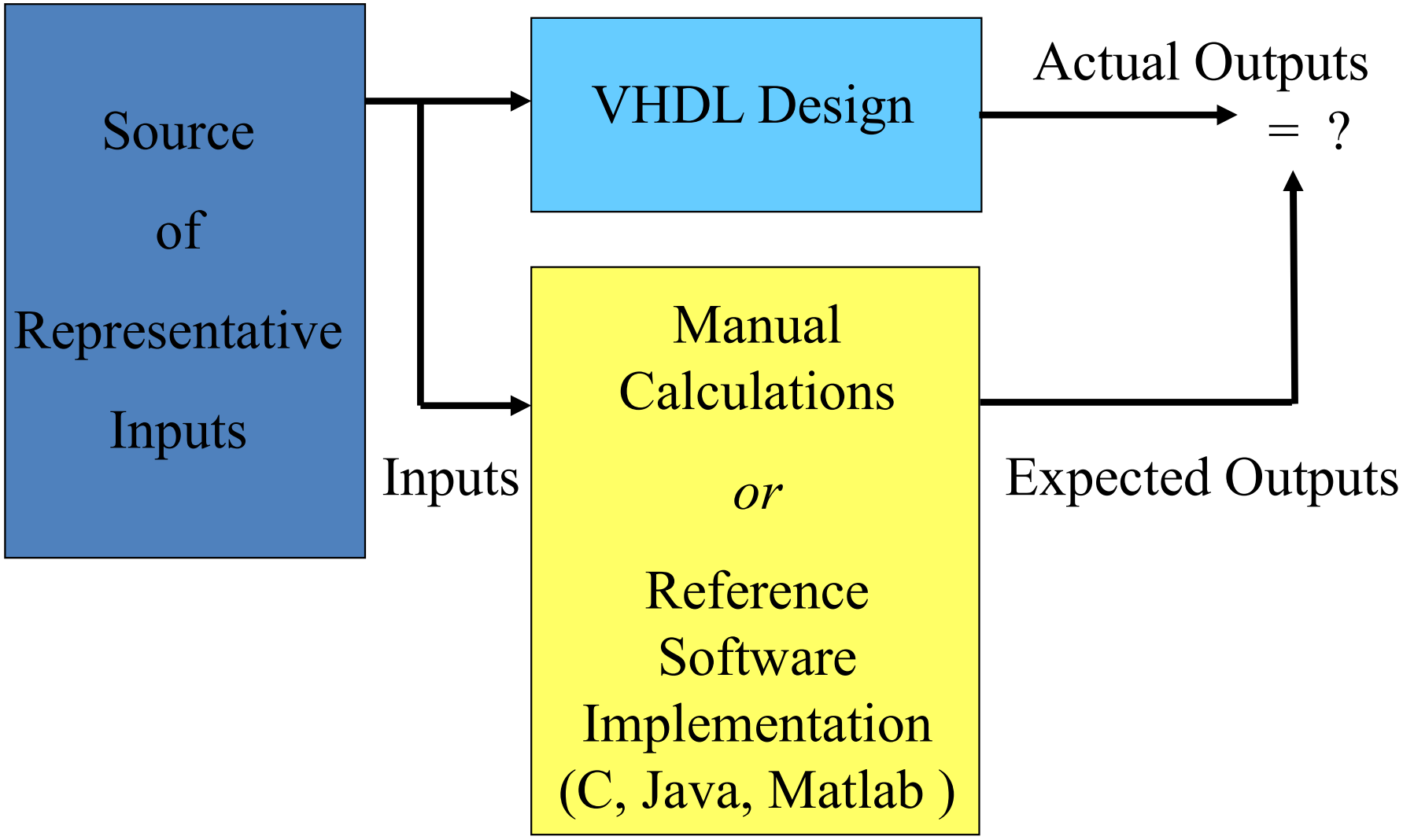
Counter Waveform



Advanced Testbench

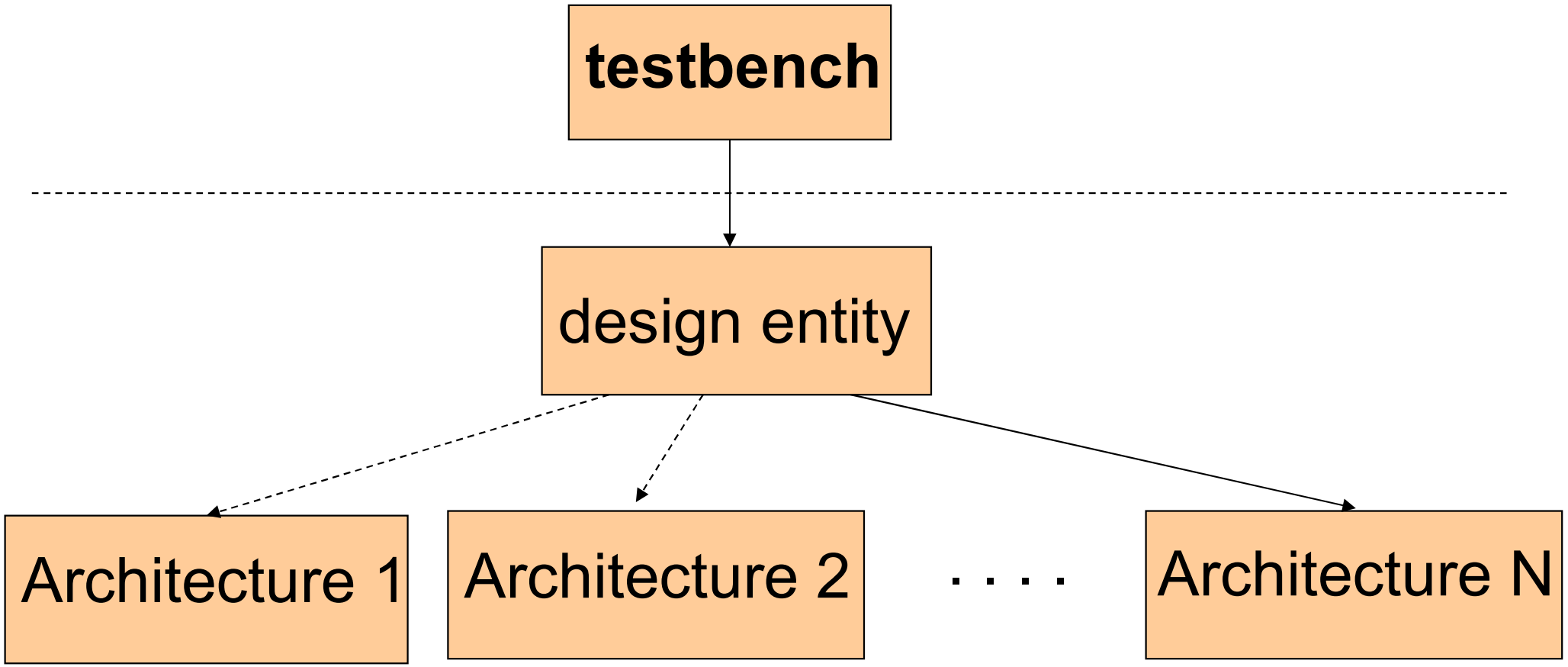


Possible Sources of Expected Outputs



Testbench

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



Test vectors

Set of pairs: {Input i, Expected Output i}

Input 1, Expected Output 1
Input 2, Expected Output 2
……
Input N, Expected Output N

Test vectors can be:
- defined in the testbench source file
- stored in a data file

Testbench Definition

- *Testbench* 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.

Testbench Anatomy

```
ENTITY tb IS
              --TB entity has no ports; sometime including Generic
END tb;

ARCHITECTURE arch_tb OF tb IS

              --Local signals and constants

  COMPONENT TestComp --All Design Under Test component declarations
            PORT (  );
  END COMPONENT;
-------------------------------------------------
BEGIN
  testSequence: PROCESS                    -- Generating the input stimuli
  END PROCESS;

  DUT:TestComp PORT MAP(                   -- Instantiations of DUTs
                       );
END arch_tb;
```

Quiz: Testbench for Full-Adder

Testbench for Full-Adder (1)

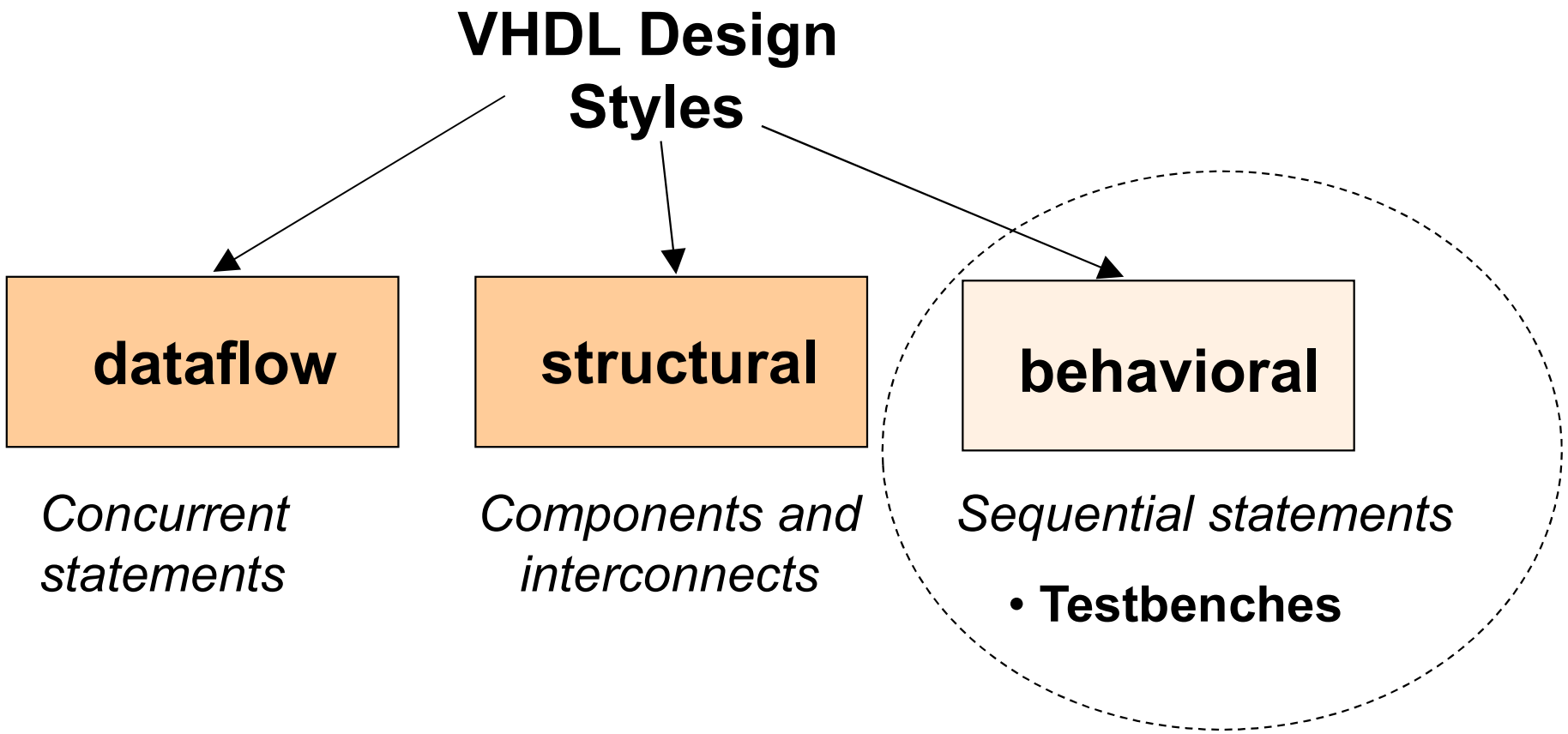
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY FullAdder_tb IS
END FullAdder_tb;

ARCHITECTURE FullAdder_tb_architecture OF FullAdder_tb IS
-- Component declaration of the tested unit
COMPONENT Fulladder_bev
PORT(
      a,b :  IN  std_logic;
      cin :  IN  std_logic;
      sum   :  OUT std_logic;
      cout : 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 Sum, Cout : STD_LOGIC;
```

Testbench for Full-Adder (2)

```
BEGIN
UUT : Fulladder_bev
 PORT MAP (
                A => test_vector(0),
                B => test_vector(1),
                Cin => test_vector(2),
                Sum => Sum,
                Cout => Cout);
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 FullAdder_tb_architecture;
```

VHDL Design Styles

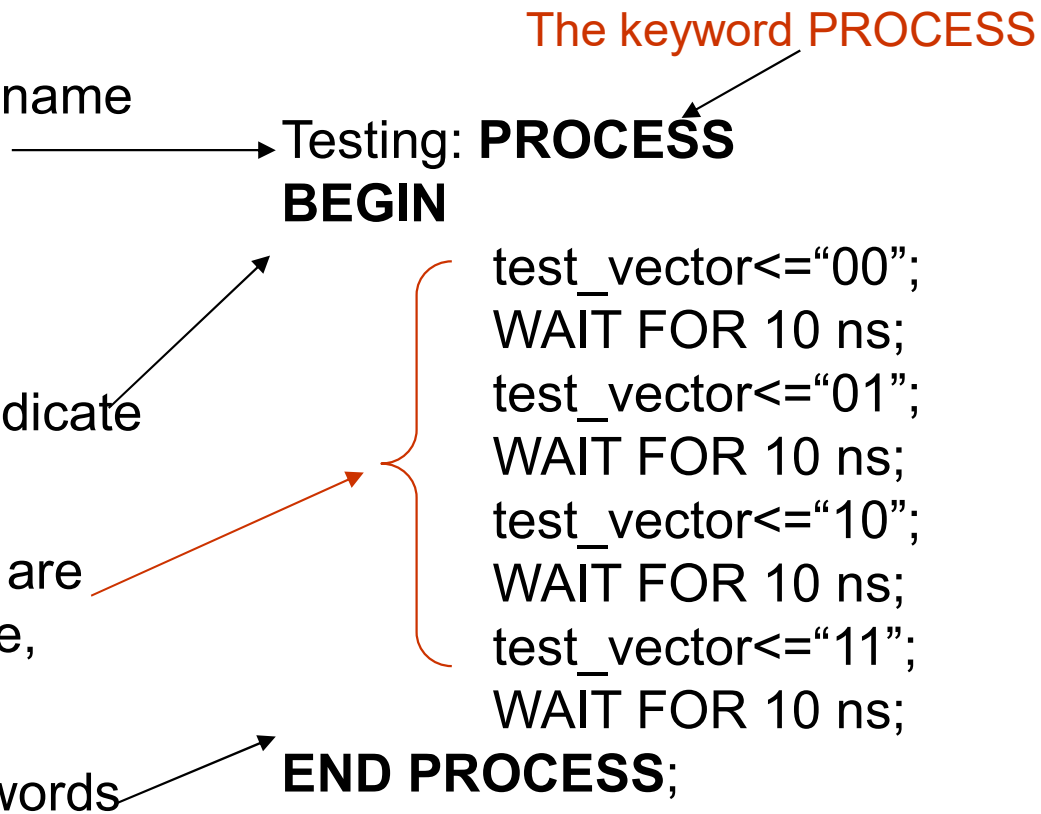


Process without Sensitivity List and its use in Testbenches

What is a PROCESS?

– A **process** is a **concurrent statement**. However, the **statements inside a process are executed sequentially**.

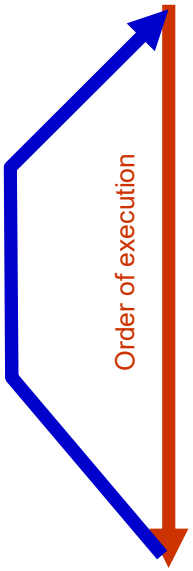
- A process can be given a unique name using an optional LABEL
- This is followed by the keyword PROCESS
- The keyword BEGIN is used to indicate the start of the process
- All statements within the process are executed SEQUENTIALLY. Hence, order of statements is important.
- A process must end with the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

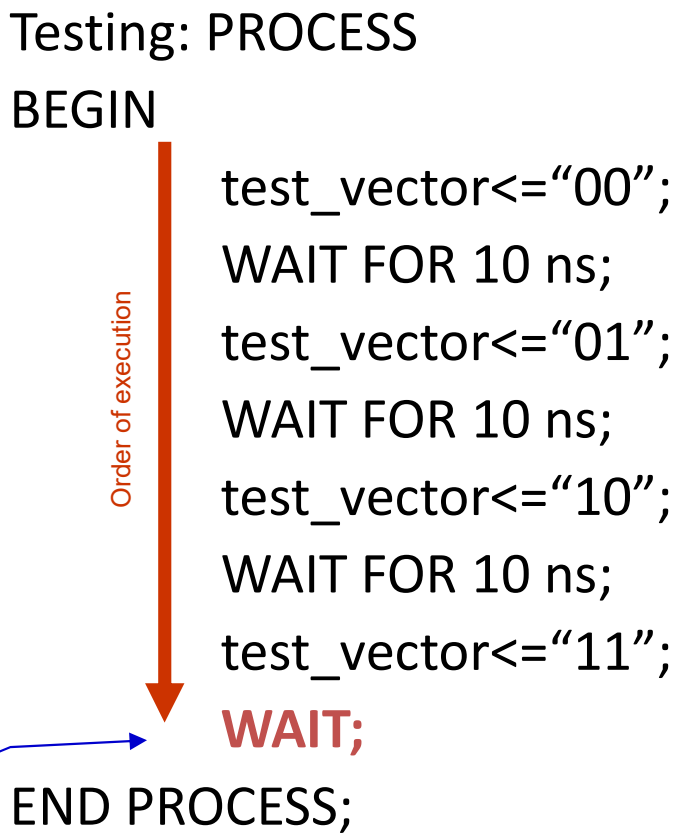
```
Testing: PROCESS
BEGIN
    test_vector<="00";
    WAIT FOR 10 ns;
    test_vector<="01";
    WAIT FOR 10 ns;
    test_vector<="10";
    WAIT FOR 10 ns;
    test_vector<="11";
    WAIT FOR 10 ns;
END PROCESS;
```



Program control is passed to the first statement after BEGIN

PROCESS with a WAIT Statement

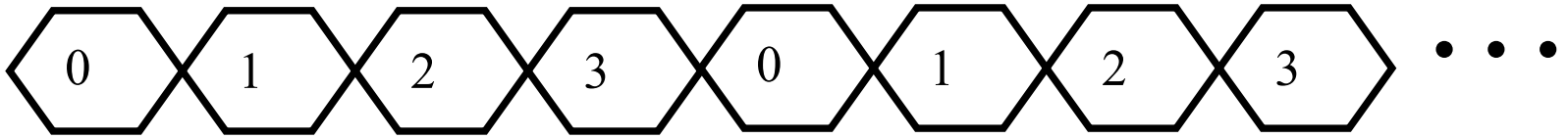
- The last statement in the PROCESS is a WAIT instead of WAIT FOR 10 ns.
- This will cause the PROCESS to suspend indefinitely when the WAIT statement is executed.
- This form of WAIT can be used in a process included in a testbench when all possible combinations of inputs have been tested or a non-periodical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

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 periodical signals, such as clocks

```
    CONSTANT clk1_period : TIME := 20 ns;
    CONSTANT clk2_period : TIME := 200 ns;
    SIGNAL clk1 : STD_LOGIC:='0';
    SIGNAL clk2 : STD_LOGIC := '0';

BEGIN
   .......
    clk1_generator: PROCESS
              WAIT FOR clk1_period/2;
              clk1  <= not clk1;
    END PROCESS;

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

Generating one-time signals, such as resets

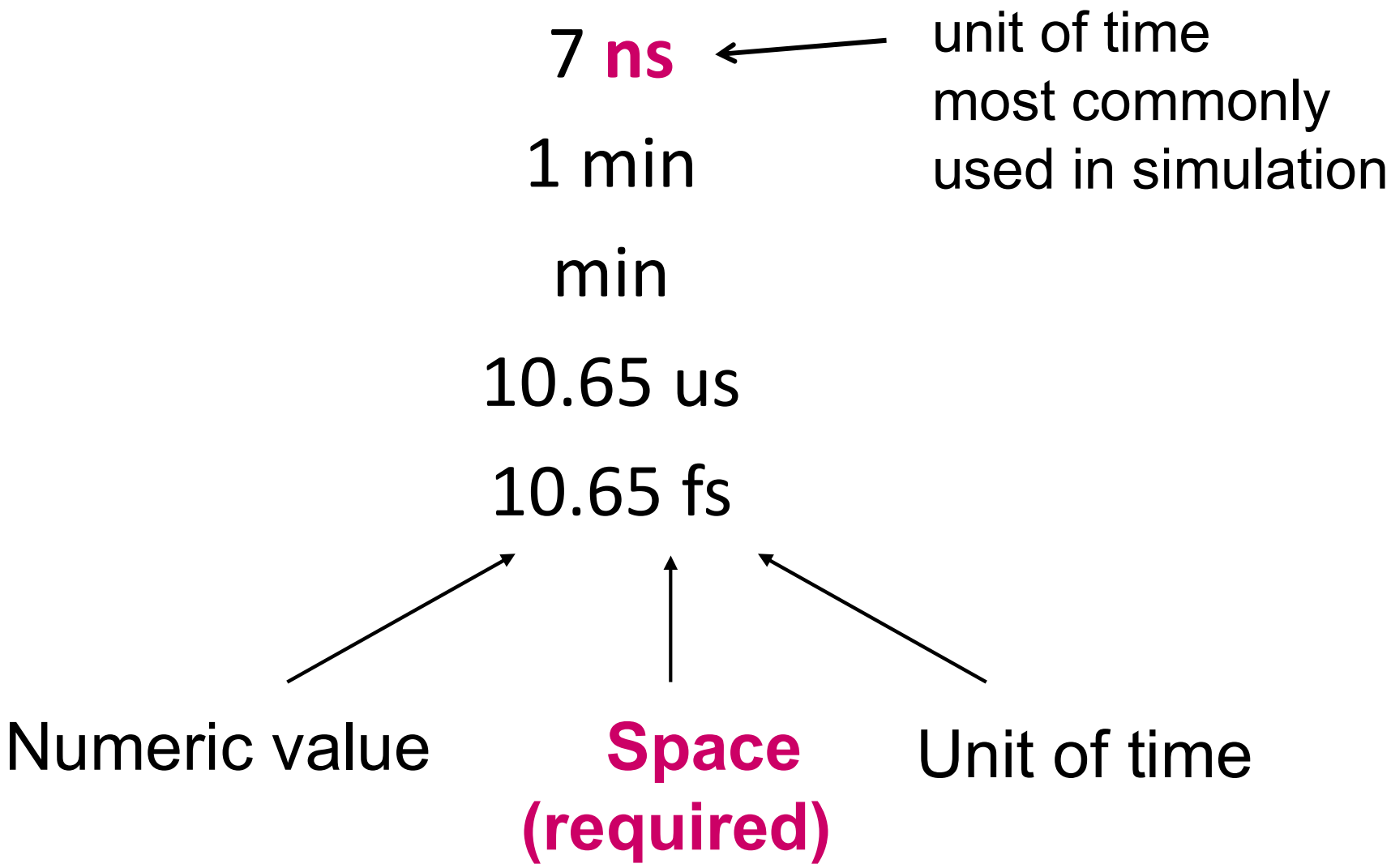
```
    CONSTANT reset1_width : TIME := 100 ns;
    CONSTANT reset2_width : TIME := 150 ns;
    SIGNAL reset1 : STD_LOGIC;
    SIGNAL reset2 : STD_LOGIC := '1';

BEGIN
    .......
    reset1_generator: PROCESS
            reset1 <= '1';
            WAIT FOR reset_width;
            reset1 <= '0';
            WAIT;
    END PROCESS;

    reset2_generator: PROCESS
            WAIT FOR reset_width;
            reset2 <= '0';
            WAIT;
    END PROCESS;
    .......
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;
```

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)

LOOP Statement
and its use in Testbenches

Loop Statement

- Loop Statement

```
FOR i IN range LOOP
    statements
END LOOP;
```

- Repeats a Section of VHDL Code
  - Example: processing every element in an array in the same way

Loop Statement – Example (1)
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;
```

Loop Statement – Example (1)
Generating all values of one input

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

BEGIN
    .......

    Testing: PROCESS
    BEGIN
            -- test_vector<="000";
            FOR i IN 0 TO 7 LOOP
                WAIT FOR 10 ns;
                test_vector<=test_vector+”001";
            END LOOP;
    END PROCESS;

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

Loop Statement – Example (2)
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
        .....
        Testing: 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+"01";
                        END LOOP;
                        test_sel<=test_sel+"01";
                END LOOP;
        END PROCESS;
        ........
END behavioral;
```

Design of Counter

Reset	Enable	Clock	$Count(4)_n$
1	-	-	"0000"
0	0	-	$Count_{n-1}$
0	1	↑	$Count_{n-1} + 1$

Design of Counter

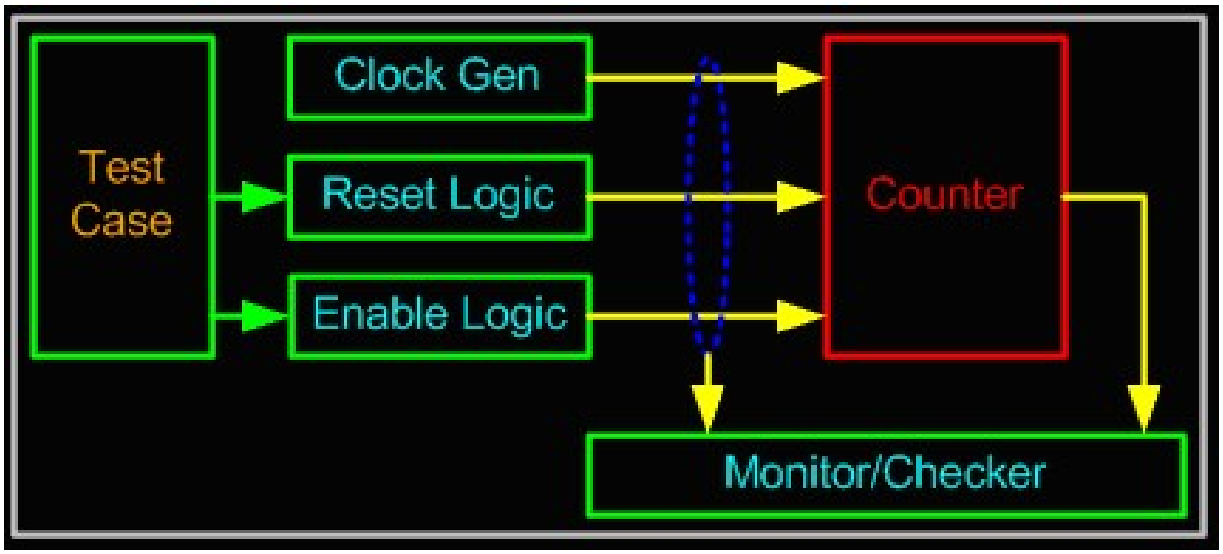
```
--counter.vhd
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
ENTITY counter IS
 PORT (clk:  IN std_logic;
     reset: IN std_logic;
     enable: IN std_logic;
     count:  OUT std_logic_vector(3 downto 0));
END counter;
```

Design of Counter

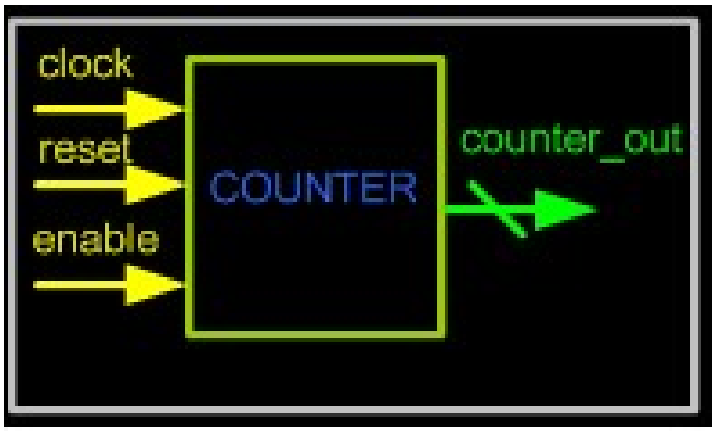
```
ARCHITECTURE behav OF counter IS
  SIGNAL pre_count: std_logic_vector(3 downto 0);
  BEGIN
    PROCESS(clk, enable, reset)
    BEGIN
      IF reset = '1' THEN
        pre_count <= "0000";
      ELSIF (clk='1' and clk'event) THEN
        IF enable = '1' THEN
            pre_count <= pre_count + "1";
          END IF;
    END IF;
    END PROCESS;
    count <= pre_count;
END behav;
```

Testbench for Counter

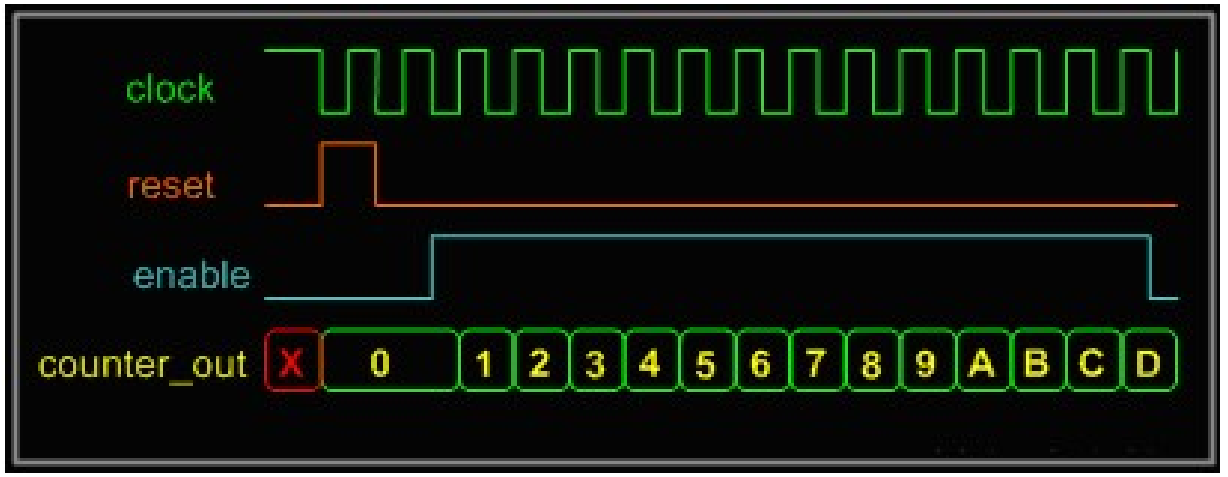
Counter Test Bench



Counter Design



Counter Waveform



Testbench for Counter (1)

```
--counter_tb.vhd
library ieee ;use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_textio.all;
ENTITY counter_tb IS
END;
ARCHITECTURE counter_tb of counter_tb IS
COMPONENT counter
   PORT (
 count : OUT std_logic_vector(3 downto 0);
 clk   : IN std_logic;
 enable: IN std_logic;
 reset : IN std_logic);
END COMPONENT ;
```

Testbench for Counter (2)

```
SIGNAL  clk : std_logic := '0';
SIGNAL  reset : std_logic := '0';
SIGNAL enable : std_logic := '0';
SIGNAL count : std_logic_vector ( 3
          downto 0);
```

```
Begin
dut : counter
   PORT MAP (  count => count,
 clk => clk,
enable=> enable,
reset => reset );
clock : PROCESS
begin
    wait for 1 ns; clk  <= not clk;
END PROCESS clock;
stimulus : PROCESS
  begin
     wait for 5 ns; reset  <= '1';
     wait for 4 ns; reset  <= '0';
     wait for 4 ns; enable <= '1';
     wait;
END PROCESS stimulus;
END counter_tb;
```