1- Introduction

2- Basic Structures of VHDL

3- Combinational Circuits

4- Sequential Circuits

5- Memory

6- Writing Testbenches

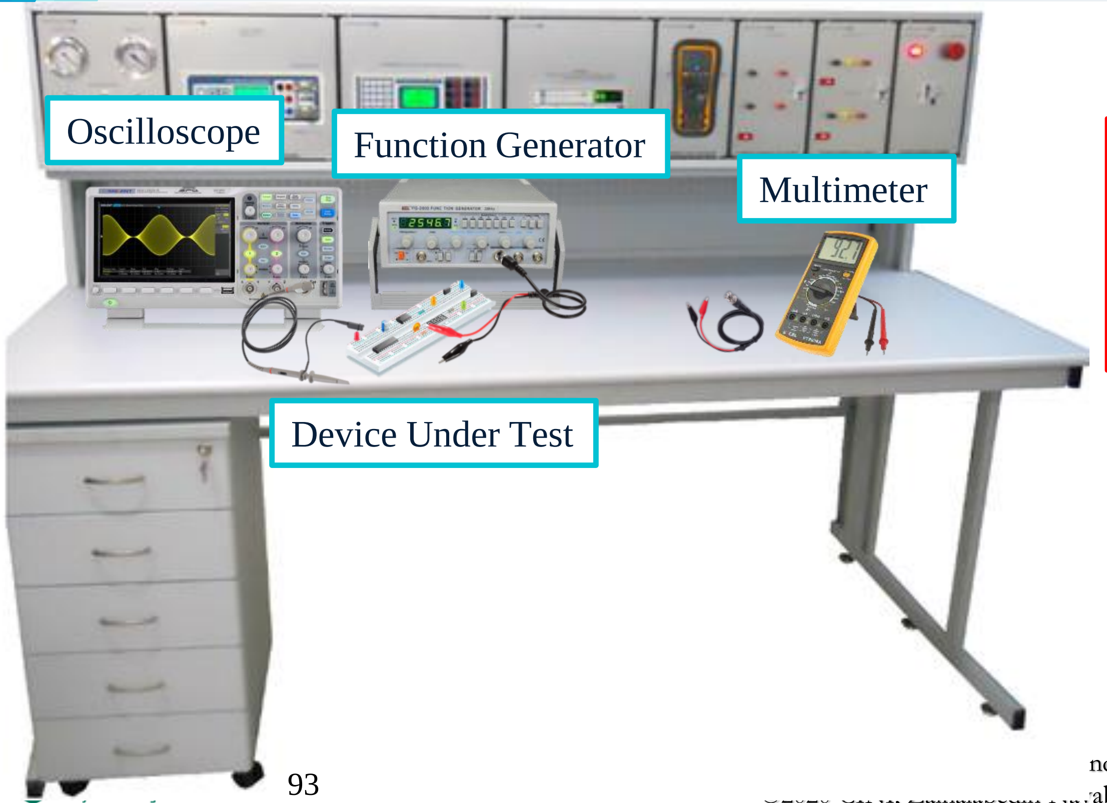
7- Synthesis Issues

8- RTL Cores

6- Writing Testbenches

- Testbench
- Simulation environment
- Writing a testbench
- Simulation waveform

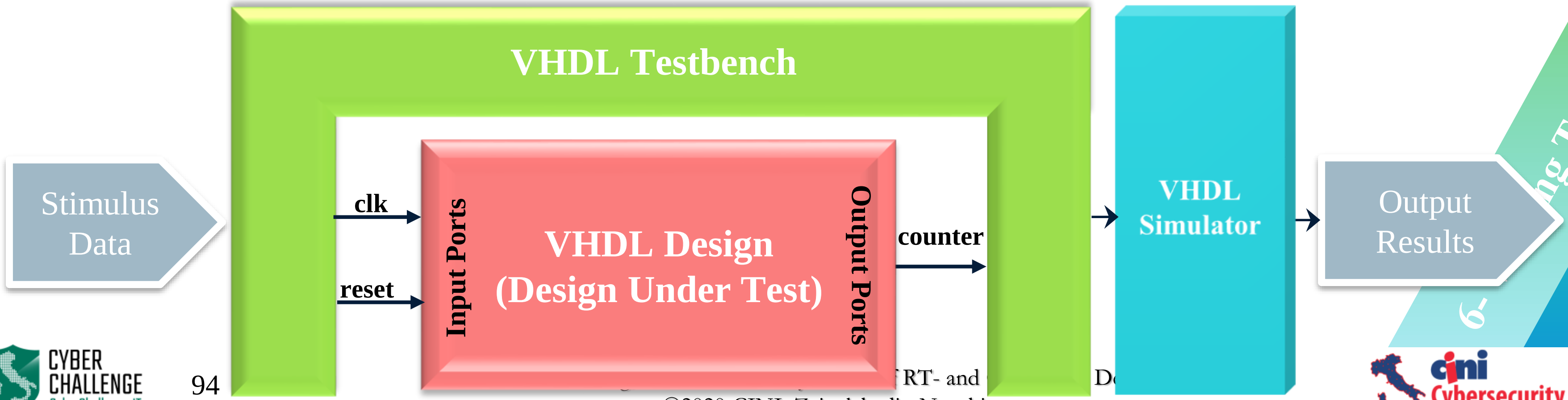
Testbench

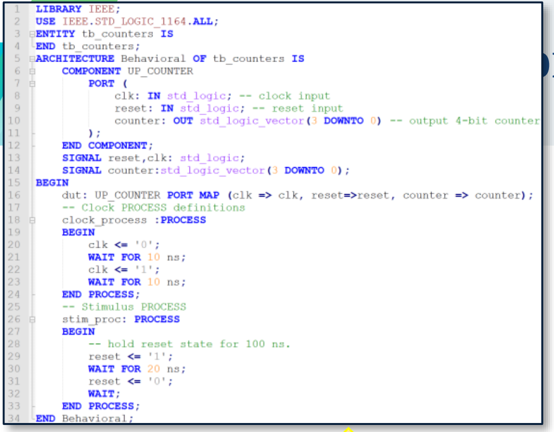


- A **testbench** is an environment used to verify the correctness or soundness of a design or model.

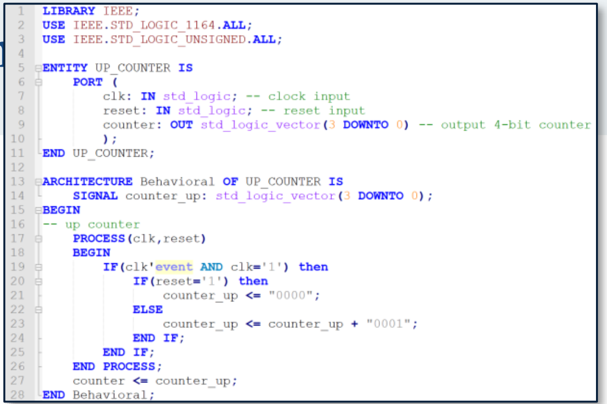
Simulation Environment

- Testbench writing is a involved process
- We suggest several methods of test data generation



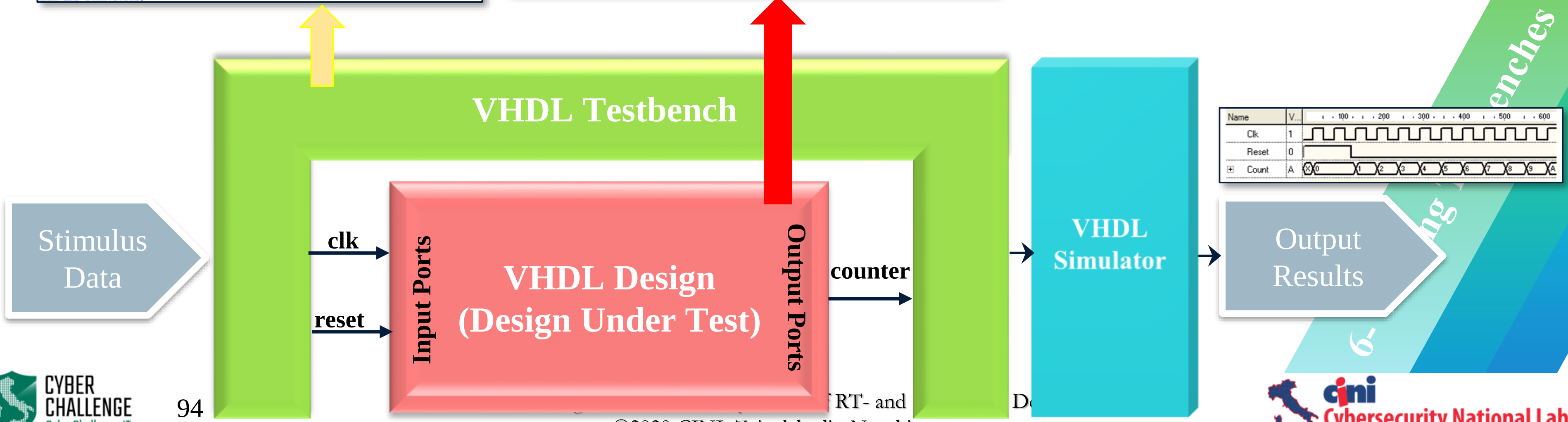


```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY tb_counters IS
END tb_counters;
ARCHITECTURE Behavioral OF tb_counters IS
    COMPONENT UP_COUNTER
        PORT (
            clk: IN std_logic; -- clock input
            reset: IN std_logic; -- reset input
            counter: OUT std_logic_vector(3 DOWNTO 0) -- output 4-bit counter
        );
    END COMPONENT;
    SIGNAL reset,clk: std_logic;
    SIGNAL counter:std_logic_vector(3 DOWNTO 0);
BEGIN
    dut: UP_COUNTER PORT MAP (clk => clk, reset=>reset, counter => counter);
    -- Clock PROCESS definitions
    clock_process :PROCESS
    BEGIN
        clk <= '0';
        WAIT FOR 10 ns;
        clk <= '1';
        WAIT FOR 10 ns;
    END PROCESS;
    -- Stimulus PROCESS
    stim_proc: PROCESS
    BEGIN
        -- hold reset state for 100 ns.
        reset <= '1';
        WAIT FOR 20 ns;
        reset <= '0';
        WAIT;
    END PROCESS;
END Behavioral;
```



```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY UP_COUNTER IS
    PORT (
        clk: IN std_logic; -- clock input
        reset: IN std_logic; -- reset input
        counter: OUT std_logic_vector(3 DOWNTO 0) -- output 4-bit counter
        );
END UP_COUNTER;

ARCHITECTURE Behavioral OF UP_COUNTER IS
    SIGNAL counter_up: std_logic_vector(3 DOWNTO 0);
BEGIN
-- up counter
    PROCESS(clk,reset)
    BEGIN
        IF(clk'event AND clk='1') then
            IF(reset='1') then
                counter_up <= "0000";
            ELSE
                counter_up <= counter_up + "0001";
            END IF;
        END IF;
    END PROCESS;
    counter <= counter_up;
END Behavioral;
```

- Testbench writing is a involved process
- We suggest several methods of test data generation



A Testbench consists of

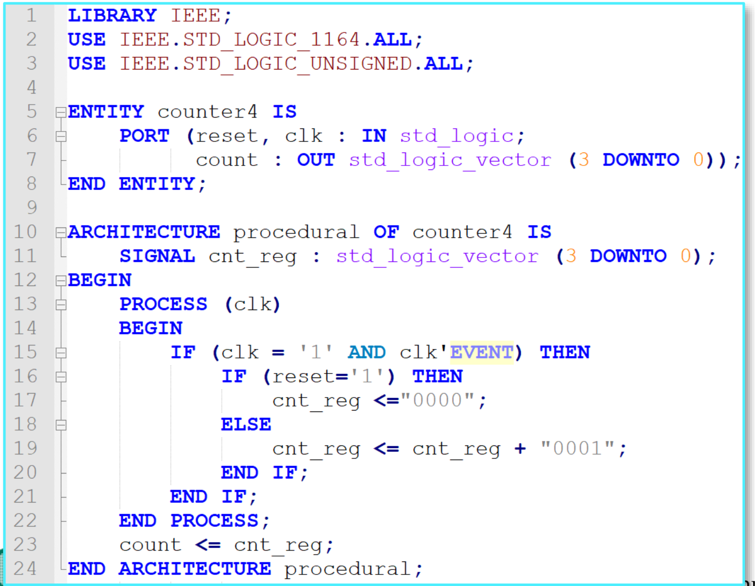
- Consider UUT as your hierarchical design
- Testbench provides data to UUT from various sources
- A testbench can also be responsible for verifying the results

A. Unit Under Test (UUT): instantiate one or more UUT's

B. Stimulus of UUT inputs: from arrays, from files, …

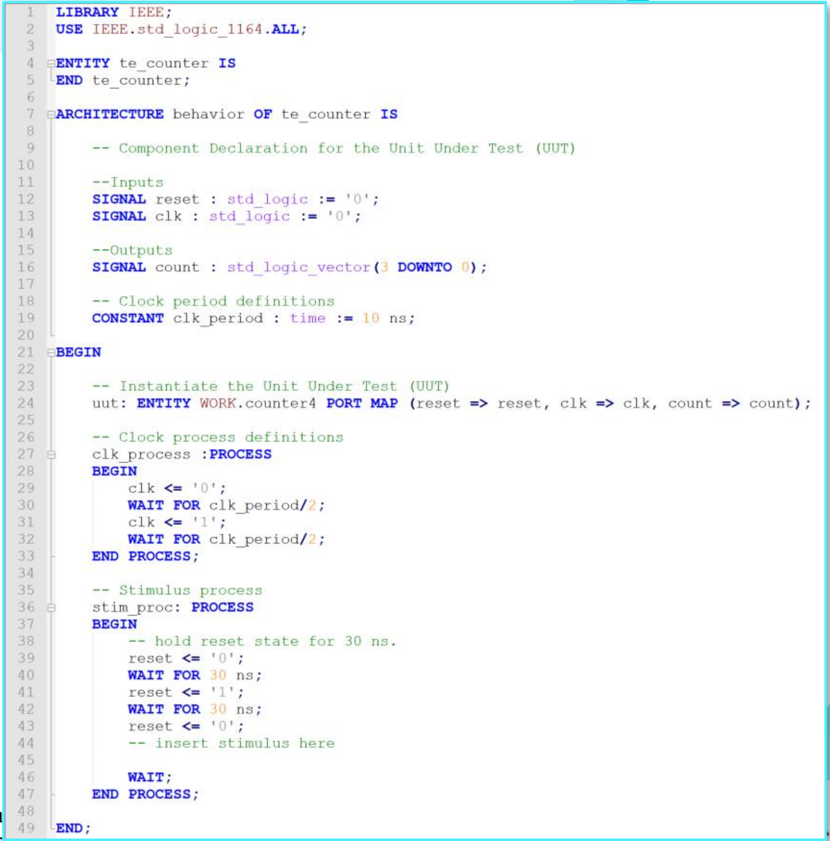
- As an example let’s test our 4-bit counter
- The top level *te_counter* entity is the testbench

Design Under Test



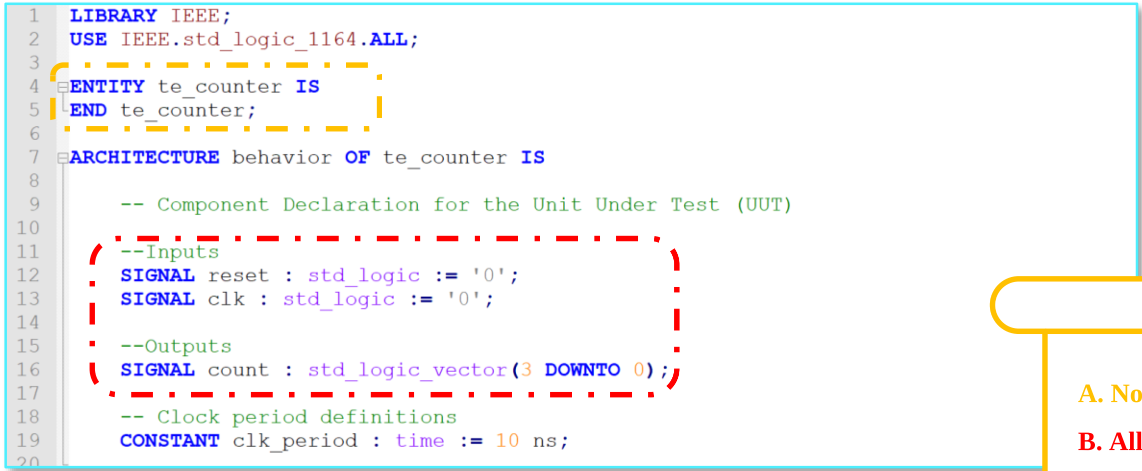
```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY counter4 IS
    PORT (reset, clk : IN std_logic;
          count : OUT std_logic_vector (3 DOWNTO 0));
END ENTITY;

ARCHITECTURE procedural OF counter4 IS
    SIGNAL cnt_reg : std_logic_vector (3 DOWNTO 0);
BEGIN
    PROCESS (clk)
    BEGIN
        IF (clk = '1' AND clk'EVENT) THEN
            IF (reset='1') THEN
                cnt_reg <="0000";
            ELSE
                cnt_reg <= cnt_reg + "0001";
            END IF;
        END IF;
    END PROCESS;
    count <= cnt_reg;
END ARCHITECTURE procedural;
```

Testbench



```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY te_counter IS
END te_counter;

ARCHITECTURE behavior OF te_counter IS

    -- Component Declaration for the Unit Under Test (UUT)

    --Inputs
    SIGNAL reset : std_logic := '0';
    SIGNAL clk : std_logic := '0';

    --Outputs
    SIGNAL count : std_logic_vector(3 DOWNTO 0);

    -- Clock period definitions
    CONSTANT clk_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: ENTITY WORK.counter4 PORT MAP (reset => reset, clk => clk, count => count);

    -- Clock process definitions
    clk_process :PROCESS
    BEGIN
        clk <= '0';
        WAIT FOR clk_period/2;
        clk <= '1';
        WAIT FOR clk_period/2;
    END PROCESS;

    -- Stimulus process
    stim_proc: PROCESS
    BEGIN
        -- hold reset state for 30 ns.
        reset <= '0';
        WAIT FOR 30 ns;
        reset <= '1';
        WAIT FOR 30 ns;
        reset <= '0';
        -- insert stimulus here

        WAIT;
    END PROCESS;

END;
```

Writing a Testbench



```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY te_counter IS
END te_counter;

ARCHITECTURE behavior OF te_counter IS

    -- Component Declaration for the Unit Under Test (UUT)

    --Inputs
    SIGNAL reset : std_logic := '0';
    SIGNAL clk : std_logic := '0';

    --Outputs
    SIGNAL count : std_logic_vector(3 DOWNTO 0);

    -- Clock period definitions
    CONSTANT clk_period : time := 10 ns;
```

- Follow template presented here for doing your testbenches
- It is easiest to use sequential coding, thus a process statement for test data
- `WAIT FOR` waits for a given time
- `WAIT;` waits forever

A. No external inputs/outputs for the testbench entity

B. All test signals are generated within the testbench

C. Instantiate the Design Under Test (DUT) in the testbench

D. Generate and apply stimuli to the DUT

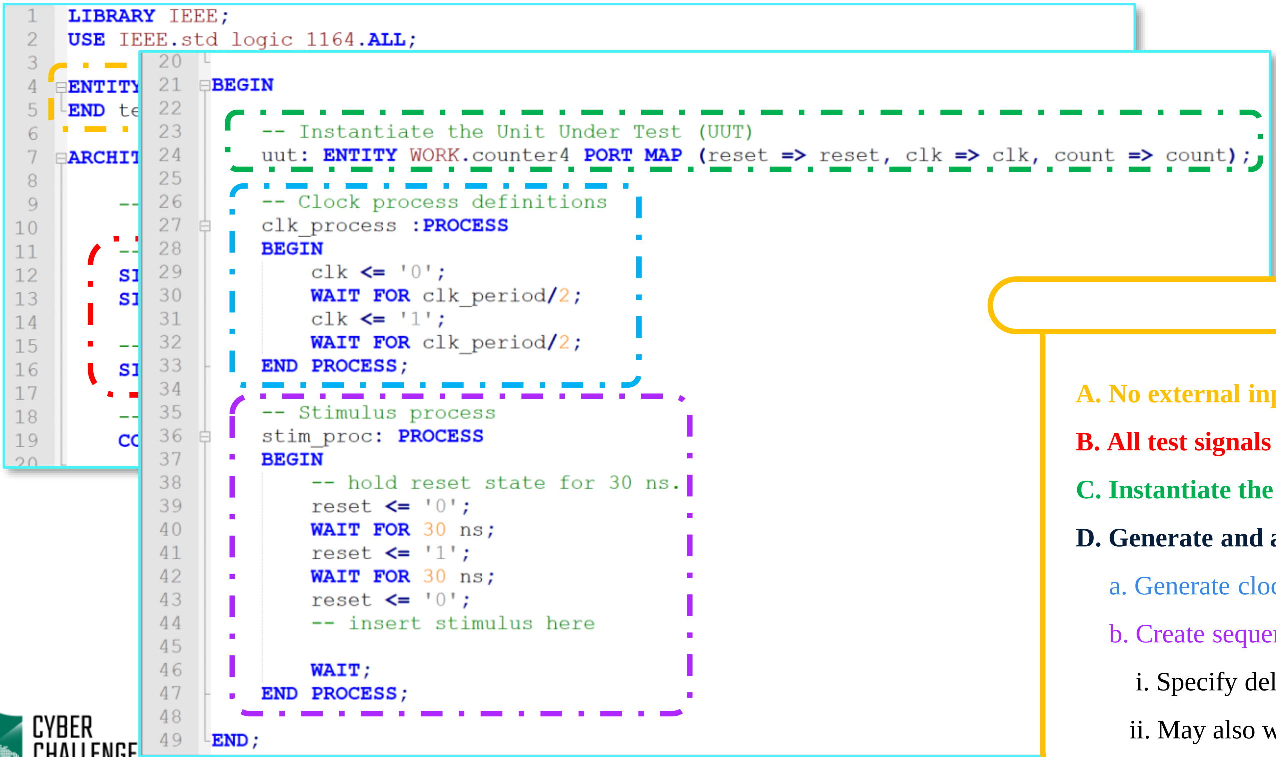
a. Generate clocks (process)

b. Create sequence of signal changes (process)

i. Specify delays between signal changes

ii. May also wait for designated signal events

Writing a Testbench



```
LIBRARY IEEE;
USE IEEE.std logic 1164.ALL;

ENTITY
END te

ARCHIT
```

```
BEGIN
   -- Instantiate the Unit Under Test (UUT)
   uut: ENTITY WORK.counter4 PORT MAP (reset => reset, clk => clk, count => count);

   -- Clock process definitions
   clk_process :PROCESS
   BEGIN
        clk <= '0';
        WAIT FOR clk_period/2;
        clk <= '1';
        WAIT FOR clk_period/2;
   END PROCESS;

   -- Stimulus process
   stim_proc: PROCESS
   BEGIN
        -- hold reset state for 30 ns.
        reset <= '0';
        WAIT FOR 30 ns;
        reset <= '1';
        WAIT FOR 30 ns;
        reset <= '0';
        -- insert stimulus here

        WAIT;
   END PROCESS;

END;
```

- Follow template presented here for doing your testbenches
- It is easiest to use sequential coding, thus a process statement for test data
- **WAIT FOR** waits for a given time
- **WAIT;** waits forever

A. No external inputs/outputs for the testbench entity

B. All test signals are generated within the testbench

C. Instantiate the Design Under Test (DUT) in the testbench

D. Generate and apply stimuli to the DUT

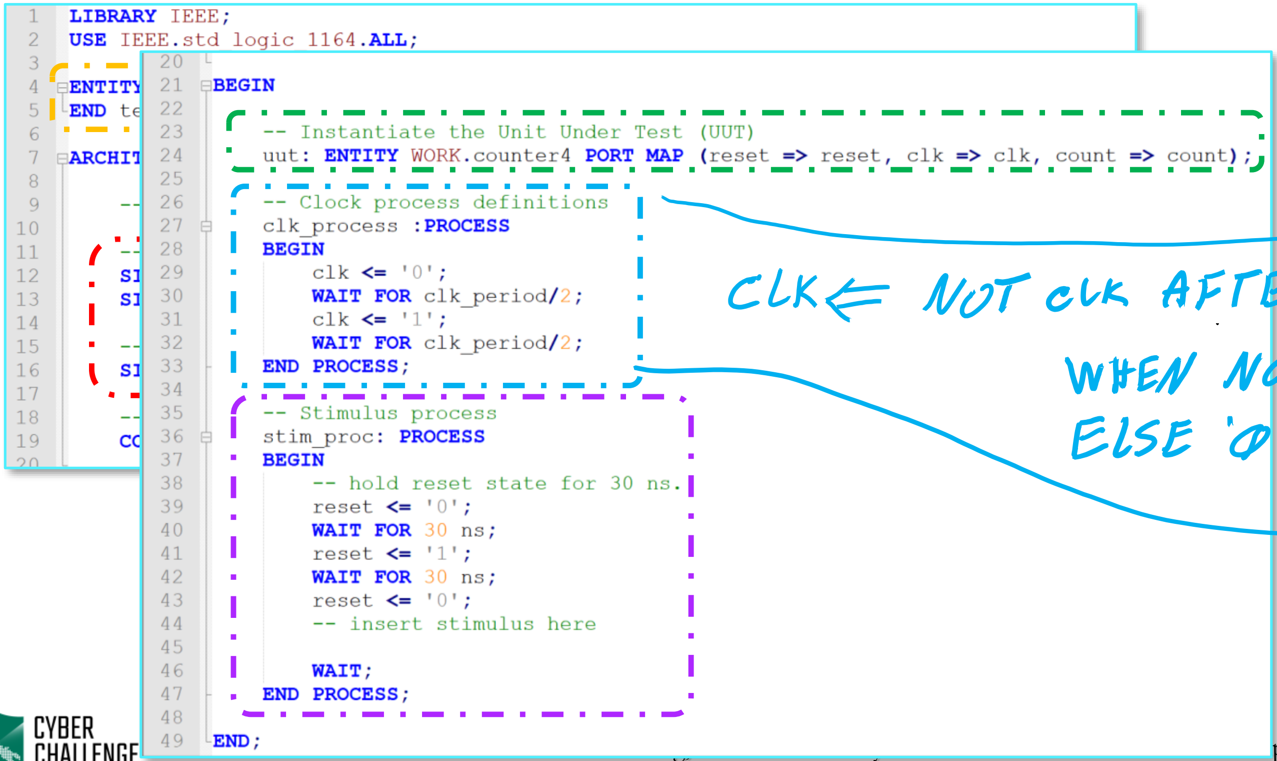
a. Generate clocks (process)

b. Create sequence of signal changes (process)

i. Specify delays between signal changes

ii. May also wait for designated signal events

Writing a Testbench



```
LIBRARY IEEE;
USE IEEE.std logic 1164.ALL;

ENTITY
END te

ARCHIT

    --

    --
    SI
    SI

    --
    SI

    --
    CO
```

```

BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: ENTITY WORK.counter4 PORT MAP (reset => reset, clk => clk, count => count);

   -- Clock process definitions
   clk_process :PROCESS
   BEGIN
        clk <= '0';
        WAIT FOR clk_period/2;
        clk <= '1';
        WAIT FOR clk_period/2;
   END PROCESS;

   -- Stimulus process
   stim_proc: PROCESS
   BEGIN
        -- hold reset state for 30 ns.
        reset <= '0';
        WAIT FOR 30 ns;
        reset <= '1';
        WAIT FOR 30 ns;
        reset <= '0';
        -- insert stimulus here

        WAIT;
   END PROCESS;

END;
```

CLK <= NOT clk AFTER clk_period/2
WHEN NOW <= 456 NS
ELSE '0';

- Use a conditional signal assignment
- Use NOW to check against the end
- Make sure signal has an initial value

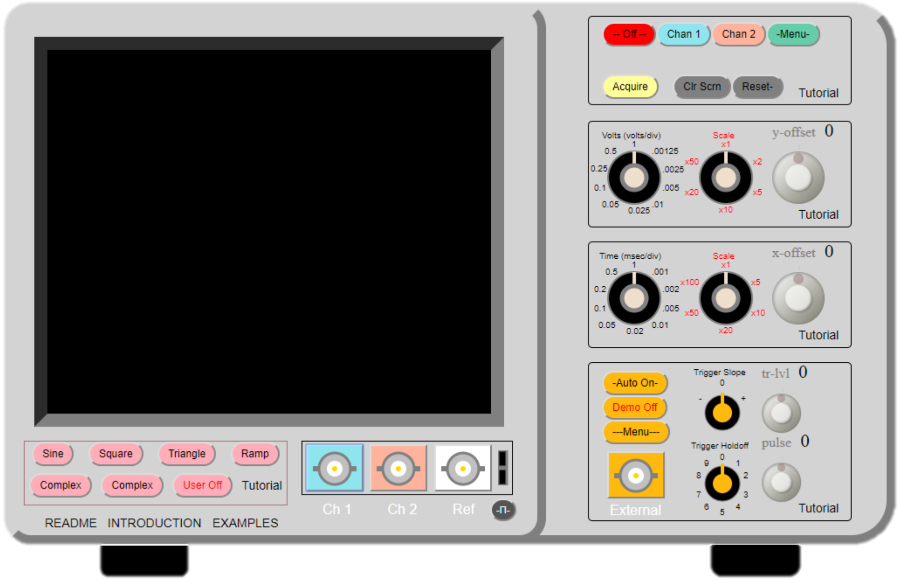
6- Writing Testbenches

CYBER CHALLENGE CyberChallenge.IT

cini Cybersecurity National Lab

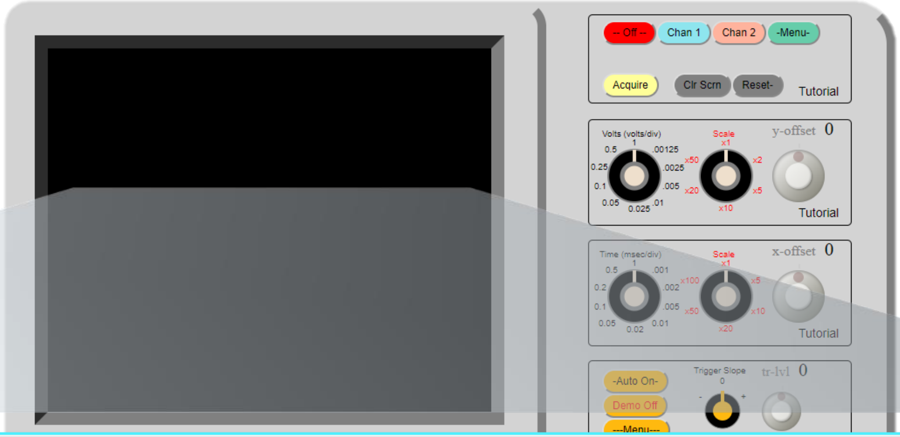
Simulation Waveform

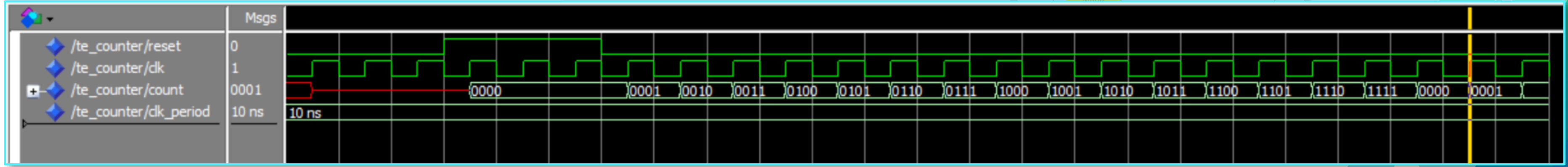
- What shows on a logic analyzer or scope is shown in a wave window



Simulation Waveform

- What shows on a logic analyzer or scope is shown in a wave window





	Msgs
/te_counter/reset	0
/te_counter/clk	1
/te_counter/count	0001
/te_counter/clk_period	10 ns

```
count:      0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111 0000 0001
clk_period: 10 ns
```