Module 3 - Behavioural Style

A behavioral style in VHDL describes a digital system by specifying its functionality using high-level algorithms and sequential statements without detailing the underlying hardware structure.

Understanding Behavioral Style

One of the three architecture models is the behavioral style. Unlike the data-flow style, a VHDL program written in behavioral style does not need to describe how the circuit will be implemented when synthesized. Instead, the behavioral style describes how the circuit's output will react to the inputs given to the circuit. The core of writing a program using the behavioral style is the **process statement**.

Using **behavioral style** in VHDL is like writing a recipe. You don't explain how the kitchen is built or how the oven is wired (the hardware implementation); instead, you describe the *steps* the cook should follow and how the dish will *turn out* depending on the ingredients (the inputs). The **process statement** is like the main set of instructions in the recipe that guides the entire cooking process.

Process Statement

A **process statement** is a concurrent command that consists of a label, sensitivity list, declaration area, begin–end (body) area, and sequential statements. An example of a process statement description in VHDL is:

```
process (<Sensitivity List>)
  -- Variable declaration area
begin
  -- VHDL statement block here
end process;
```

The difference between a **concurrent signal assignment statement** and a **process statement** lies in the sequential statements. The syntax or statements inside the begin–end (body) section are executed sequentially, line by line, just like in general programming. The **process label** itself is simply a self-descriptive naming form to help us recognize which process is being executed in that section, so the naming can be changed or even omitted.

In a **concurrent statement**, every time a change occurs in the input, the output is re-evaluated. In a **behavioral style** model using a process statement, whenever a change occurs in a signal listed in the sensitivity list of the process, all the sequential statements within the process body are re-evaluated.

Since a **process statement** is itself a concurrent statement, if there are two processes in the architecture body, the execution of both processes will be carried out concurrently.

Sequential Statement

In a process, the execution of sequential statements will be initiated when there is a change in the signals listed in the **sensitivity list**. In general, the execution of statements in the process body will be carried out **continuously** until the end of the process **sequentially**. There are **two types** of sequential statements that will be discussed in this module:

● If statement

The if statement is used to create a branch in the execution flow of sequential statements. In VHDL, the if statement can only be used inside the process body. Example of a NAND gate with if statement:

```
library ieee;
use ieee.std_logic_1164.all;

entity nand_gate is
    port(
        A, B : IN STD_LOGIC;
        Y    : OUT STD_LOGIC
    );
end nand_gate;

architecture behavioral of nand_gate is
begin
    nand_proc : process (A, B) is
    begin
        if (A = '1' AND B = '1') then
            Y <= '0';
        else
            Y <= '1';
        end if;
    end process nand_proc;
end behavioral;
```

● Case statement

The case statement works in a way similar to the previous if statement. The difference is that the case statement will be more efficient to use when there are many variations of values. Example of a NAND gate using case statement:

```
library ieee;
use ieee.std_logic_1164.all;

entity nand_gate is
    port(
        A, B : IN STD_LOGIC;
        Y    : OUT STD_LOGIC
    );
end nand_gate;

architecture behavioral of nand_gate is
begin
    AB <= A & B;  -- combining signals A and B

    nand_proc : process (A, B) is
    begin
        case (AB) is
            when "11"    => Y <= '0';
            when others  => Y <= '1';
        end case;
    end process nand_proc;
end behavioral;
```

When using behavioral style, nested sequential statements are common and often used. This is also what makes behavioral style more **powerful** than data-flow style. However, even though behavioral style statements are used like programming in general, it should be noted that VHDL is a hardware description language, not a programming language. Also, try to always keep the process statement in the description you create as **simple** as possible to make hardware design easier when the circuit becomes more complex.

Wait Statements

Wait Statements
Wait statements are used to make a process wait for a certain condition, signal/variable, or a specific time interval. The following wait statements are used:

● **Wait until [condition]** and **wait on [signal]**
`wait until [condition]` will block the process while checking whether a condition is true or false. The process will remain blocked until the condition being checked becomes true. Meanwhile, `wait on [signal]` will wait until there is a change in the specified signal. The syntax of `wait until` and `wait on` can be synthesized.

```
process is
begin
    signal1 <= '1';
    signal2 <= '0';

    wait until rst <= '1';

    signal1 <= '0';
    signal2 <= '1';

    wait on clk;
end process;
```

● **Wait for [time period]**
`wait for [time period]` will block the process for the specified time period. The syntax of `wait for` cannot be synthesized but can be simulated in the waveform using ModelSim. Therefore, `wait for` is commonly used for a testbench, which will be studied in the next module.

```
process is
begin
    signal1 <= '1';
    signal2 <= '0';

    wait for 50 ps;

    signal1 <= '0';
    signal2 <= '1';
```

```
    wait for 100 ps;
end process;
```

Report Statements

In VHDL, the **report statement** is used to generate text messages during simulation. This statement is useful for providing information about the status or certain values during simulation. The generated report will appear in the transcript to help with debugging.

‘Image Attribute
In VHDL, the `'image` attribute is used to convert a certain data type into a string data type. This attribute is useful when you want to combine or display values of different data types in the form of a string. In a report statement, this attribute is used so that variables/signals can be printed to the transcript, which only accepts strings.

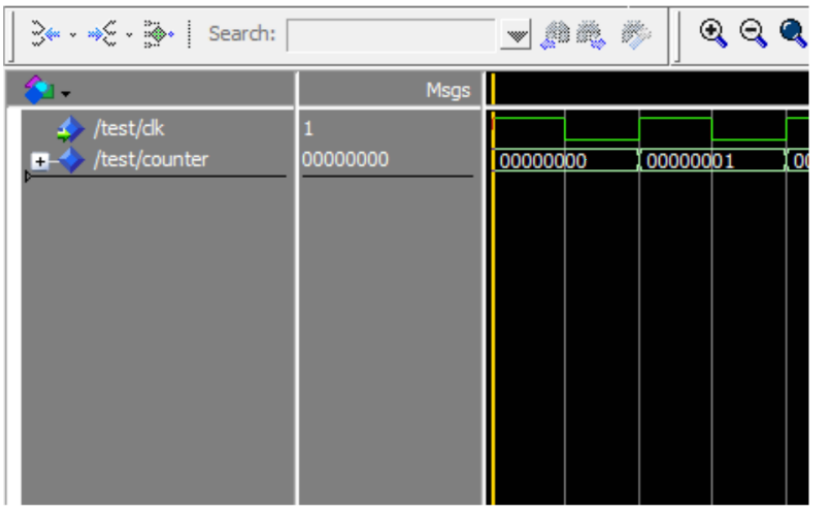
Here is an example of a report statement that also uses the `'image` attribute:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

ENTITY test IS
PORT (
    clk : IN STD_LOGIC
);
END test;

ARCHITECTURE Behavioral OF test IS
    SIGNAL counter : UNSIGNED(7 DOWNTO 0);
BEGIN
    PROCESS
    BEGIN
        WAIT FOR 50 ps;
        LOOP
            WAIT UNTIL rising_edge(clk);
            REPORT "Counter = " & INTEGER'image(conv_integer(counter));
            counter <= counter + 1;
        END LOOP;
    END PROCESS;
END Behavioral;
```

Example simulation:



Library | Project | sim

Transcript

```
# Loading ieee.std_logic_arith(body)
# Loading work.test(behavioral)
add wave -position insertpoint sim:/test/*
force -freeze sim:/test/clk 1 0, 0 {50 ps} -r 100
VSIM 9> run
# ** Note: Counter = 0
#    Time: 100 ps  Iteration: 0  Instance: /test
# ** Note: Counter = 1
#    Time: 200 ps  Iteration: 0  Instance: /test
run
# ** Note: Counter = 2
#    Time: 300 ps  Iteration: 0  Instance: /test
# ** Note: Counter = 3
#    Time: 400 ps  Iteration: 0  Instance: /test
run
# ** Note: Counter = 4
#    Time: 500 ps  Iteration: 0  Instance: /test
# ** Note: Counter = 5
#    Time: 600 ps  Iteration: 0  Instance: /test
```