Module 7 - Procedure, Function, and Impure Function

Procedure

In VHDL, a *procedure* is a type of language construct used to group several statements and specific tasks into a single block of code. Procedures help in organizing and simplifying the understanding of complex VHDL designs.

A procedure in VHDL is similar to a void function in languages like C. It performs a specific task but does not return a value. Instead, it may modify signals or variables passed to it as parameters, allowing designers to reuse code and improve readability.

Procedure Declaration

A procedure is defined using a procedure declaration. This declaration specifies the procedure name, any required parameters (if any), and the data type that is returned (if applicable). Below is an example of a procedure declaration in VHDL:

```
procedure Nama_Procedure(parameter1: tipe_data; parameter2: tipe_data) return tipe_data
  is
  begin
  -- Blok kode procedure
end Nama_Procedure;
```

Parameters in Procedure
A procedure can accept parameters as arguments. These parameters are used to pass data into the procedure so that it can be processed. The required parameters are defined in the procedure declaration and can be of different modes such as in, out, or inout, depending on whether the data is being read, written, or both.

Procedure Body (Code Block)
The body of a procedure is the section where the tasks to be performed by the procedure are written. Within this block, you can write statements to perform various operations such as calculations, condition checks, data manipulation, and more. This is where the logic of the procedure is implemented, similar to the body of a function in other programming languages.

Procedure Call

To use a procedure, it can be called from the main part of the design or from within another procedure. A procedure is invoked by providing arguments that match the parameters defined in its declaration. Below is an example of how a procedure is used in VHDL.

```
variable1 := Nama_Procedure(nilai_parameter1, nilai_parameter2);
```

Example Code

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

entity Procedure_Example is
  port(
    clk : in std_logic;         -- Clock input
    result_out : out integer    -- Output to observe the result
  );
end Procedure_Example;

architecture Behavioral of Procedure_Example is

  -- Signal declarations
  signal sigA : integer := 5;
  signal sigB : integer := 7;
  signal adder_result : integer;

  procedure adder(
    A, B : in integer;       -- Input parameters
    Hasil : out integer      -- Output parameter (will write to adder_result)
  ) is
  begin
    Hasil := A + B;
  end procedure;

begin

  process(clk)
  begin
    if rising_edge(clk) then
      adder(sigA, sigB, adder_result);  -- Call the procedure
    end if;
  end process;
```

```
  -- Output assignment
  result_out <= adder_result;

end Behavioral;
```

Function

In VHDL, a **function** is a subprogram used to perform calculations or data processing that **returns a single value as a result**. Functions in VHDL are similar to functions in traditional programming languages such as C or Java, where the main purpose is to compute a value based on the input arguments. Unlike procedures, functions **must return exactly one value** and **cannot modify signals or variables outside the function directly**. They are typically used for operations like arithmetic, logical computations, or data conversion.

Functions improve code readability, reusability, and modularity in complex VHDL designs.

Function Declaration

A function is defined using a function declaration. This declaration specifies:

- The function name
- Input parameters (if any)
- The return data type

Below is an example of a function declaration in VHDL:

```
function Function_Name(parameter1: data_type; parameter2: data_type) return return_type is
begin
  -- Function code block
  return value;
end Function_Name;
```

Parameters in Function

A function can accept input parameters only. These parameters are used to pass data into the function to be processed. Unlike procedures, functions cannot have out or inout parameters. All data returned from a function must be provided through the return statement.

Function Body (Code Block)

The body of a function contains the logic used to compute and return a value. This may include arithmetic operations, conditions, or other expressions. The function must include a return statement that provides the final result.

Function Call

A function is called by using its name and passing the required arguments. The returned value can be directly assigned to a signal or variable.

```
variable1 := Function_Name(input_value1, input_value2);
```

Example Code

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

entity Function_Example is
  port(
    clk : in std_logic;         -- Clock input
    result_out : out integer    -- Output to observe the result
  );
end Function_Example;

architecture Behavioral of Function_Example is

  -- Signal declarations
  signal sigA : integer := 10;
  signal sigB : integer := 20;
  signal function_result : integer;

  -- Function Declaration
  function adder(
    A: integer;
    B: integer
  ) return integer is
  begin
    return A + B;
  end function;

begin

  process(clk)
  begin
    if rising_edge(clk) then
      function_result <= adder(sigA, sigB);  -- Call the function
```

```
    end if;
  end process;

  -- Output assignment
  result_out <= function_result;

end Behavioral;
```

Impure Function

In VHDL, an **impure function** is a special type of function that is allowed to **read or modify signals, variables, or states outside its local scope**. Unlike a pure function, which always produces the same output for the same input (no side effects), an impure function **can interact with external data** and may produce different results each time it is called.

Impure functions are useful when you need to:

- Access or modify global variables or signals
- Read the current value of a signal that may change over time
- Implement functions with memory or state (like random number generators, counters, etc.)

To define an impure function, you must use the keyword `impure`.

Impure Function Declaration

```
impure function Function_Name(parameter1: data_type) return return_type is
begin
  -- Function code block with external side effects
  return value;
end Function_Name
```

Code Example

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

entity Impure_Function_Example is
  port(
    clk : in std_logic;
    result_out : out integer
  );
end Impure_Function_Example;
```

```
architecture Behavioral of Impure_Function_Example is

  -- Signal declaration
  signal counter : integer := 0;
  signal function_result : integer;

  -- Impure Function Declaration
  impure function read_and_increment return integer is
  begin
    counter <= counter + 1;          -- Modifies an external signal
    return counter;                  -- Returns updated value
  end function;

begin

  process(clk)
  begin
    if rising_edge(clk) then
      function_result <= read_and_increment;  -- Call the impure function
    end if;
  end process;

  -- Output assignment
  result_out <= function_result;

end Behavioral;
```

When to Use Impure or Pure Functions

Code 1

```
function multiply(a, b: integer) return integer is
begin
  return a * b;
end function;
```

This is **pure** because it depends only on the inputs a and b.

Code 2

```
signal counter : integer := 0;

impure function increment_counter return integer is
begin
  counter <= counter + 1;  -- Modifies external signal
  return counter;
end function;
```

This function must be **impure** because it changes an external signal (counter).

Code 3

```
signal total_sum : integer := 0;

impure function add_and_accumulate(a, b : integer) return integer is
begin
  total_sum <= total_sum + (a + b); -- Modify external signal
  return total_sum;                 -- Return updated accumulated value
end function;
```

This function must be **impure** because it changes an external signal (total_sum).

Code 4

```
signal current_status : integer := 3;

impure function read_status return integer is
begin
  return current_status; -- Reads external signal, so must be impure
end function;
```

This function mus be **impure** because it reads external signal that may change over time.

Code 5

```
impure function random_generator return integer is
  variable seed : integer := 1; -- Static variable retains value between calls
begin
  seed := (seed * 1103515245 + 12345) mod 256; -- Simple random algorithm
  return seed;
end function;
```

This function must be impure because it maintains internal state (seed) that changes on each call.

Procedure, Function and Impure Function Synthesis

In VHDL, both "function" and "procedure" can be used in hardware descriptions, but it should be understood that hardware synthesis is usually more suitable for implementations based on deterministic and synchronous behavior. Therefore, there are several limitations on the use of functions and procedures in the context of synthesis:

- **Procedure**: Procedures in VHDL are used to perform tasks without returning a value. They can be used in hardware descriptions to manage operations and organize code. Hardware synthesis will usually replace procedure calls with the corresponding physical actions in the target hardware. Therefore, deterministic procedures can be synthesized. However, there are some limitations on the use of procedures that depend on time flow or behavior that is difficult to predict. Some VHDL compilers may not support the synthesis of such procedures.
- **Function**: VHDL functions that do not have impure properties (e.g., they produce a deterministic value based on input arguments only) can usually be synthesized well.
- **Impure Function**: Impure functions, which produce unpredictable results or depend on external factors, are usually not suitable for deterministic hardware synthesis. Impure functions that depend on random or non-deterministic behavior will not be synthesizable, as the resulting hardware must be deterministic and predictable.

So, while functions and procedures can be used in hardware descriptions and can be synthesized if they meet certain requirements, impure functions are usually not suitable for VHDL synthesis.

Difference between Procedure, Function and Impure Function

Criteria	Procedure	Function	Impure Function
Purpose	Perform tasks without returning a value.	Return a value from a calculation.	Produce an unpredictable value or one that depends on external factors.
Arguments	Can have input and/or output arguments.	Can have input arguments only.	Can have input arguments only.
Return Value	Does not return a value (void).	Returns a value from a calculation.	Returns a value from a calculation.
Usage	Used to organize tasks or operations.	Used for calculations or data processing.	Used when the function's result depends on external factors.
Example	`vhdl procedure SetFlag(flag: out boolean);`	`vhdl function Add(a, b: integer) return integer;`	`vhdl function RandomNumber return integer;`
Synthesis	Can be synthesized if it is deterministic and synchronous.	Can be synthesized if it is deterministic and synchronous.	Not suitable for synthesis because the result is unpredictable or depends on external factors.