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.

Revision #1
Created 2025-10-23 05:55:22 UTC by CH
Updated 2025-10-23 06:04:25 UTC by CH