2. Dataflow Style in VHDL

2.1 What is Dataflow Style

The Dataflow style is built on concurrency because the **central idea is to model the system as a set of concurrent operations on signals**, which directly reflects the physical reality of hardware. In any integrated circuit, all components are active and processing data in parallel; they don't execute in a step-by-step sequence. To accurately describe this, VHDL uses concurrent statements as the foundation. This approach ensures that the model's behavior matches the hardware's true, simultaneous nature, where multiple data transformations happen at the same time.

This is why **the Dataflow style describes a circuit by focusing on the flow of data and the transformations applied to it, rather than specifying the exact gate-level structure.** Each concurrent statement you write whether it's a simple logical operation or a conditional assignment defines one of these parallel transformations. This method allows you to build a functional description of the circuit by defining the paths and changes that signals undergo. By focusing on this "flow," you let the synthesis tool determine the best gate-level implementation, making it an intuitive and powerful way to translate a high-level circuit diagram into code.

2.2 Concurrent Statement

In general, digital circuits are concurrent in nature (working in parallel or simultaneously). A change in the output of a concurrent circuit is directly influenced by a change in an input. Therefore, VHDL also adopts the concept of concurrency when processing the inputs and outputs of a circuit. A program in VHDL cannot be equated to software programming where instructions are executed sequentially. Instructions in VHDL are executed directly and simultaneously. However, with certain techniques, VHDL can also describe sequential circuits whose processes are executed in order.

A concurrent statement is a method used to describe the parallel operation of hardware in VHDL. There are four ways to describe a concurrent statement:

- **Concurrent signal assignment :** This is the most common method used to describe a concurrent statement. The output of the circuit can change at any time when one of its inputs changes. Example description of a 3-input NAND and AND gate using concurrent signal assignment:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Entity defines the inputs and outputs (the "black box")
entity Logic_Gates is
    port (
        a, b, c : in  STD_LOGIC;          -- Three inputs
        y_and   : out STD_LOGIC;          -- Output for the AND gate
        y_nand  : out STD_LOGIC           -- Output for the NAND gate
    );
end Logic_Gates;

-- Architecture describes what happens inside the box
architecture Behavioral of Logic_Gates is
begin
    -- These two statements are CONCURRENT. They happen at the same time.
    y_and  <= a and b and c;
    y_nand <= a nand b and c;

end Behavioral;
```

- **Conditional signal assignment :** This method is used to describe a statement that has a single target signal but has more than one condition to evaluate. Example description of a 2-to-1 Mux using conditional signal assignment:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Entity for a 2-to-1 Multiplexer
entity Mux_2_to_1_Conditional is
    port (
        i0, i1 : in  STD_LOGIC;  -- The two data inputs
        sel    : in  STD_LOGIC;  -- The select line
        y      : out STD_LOGIC   -- The single output
    );
end Mux_2_to_1_Conditional;

-- Architecture using the "when/else" concurrent statement
architecture Behavioral of Mux_2_to_1_Conditional is
begin
```

```
    -- The output 'y' gets the value of 'i0' WHEN 'sel' is '0',
    -- ELSE it gets the value of 'i1'.
    y <= i0 when sel = '0' else i1;

end Behavioral;
```

- **Selected signal assignment :** This method differs from the conditional signal assignment method. In a selected signal assignment, the target is based on the evaluation of an expression. Example description of a 2-to-1 Mux with selected signal assignment:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Entity for a 2-to-1 Multiplexer
entity Mux_2_to_1_Selected is
    port (
        i0, i1 : in  STD_LOGIC;  -- The two data inputs
        sel    : in  STD_LOGIC;  -- The select line
        y      : out STD_LOGIC   -- The single output
    );
end Mux_2_to_1_Selected;

-- Architecture using the "with/select" concurrent statement
architecture Behavioral of Mux_2_to_1_Selected is
begin
    -- WITH the value of 'sel', SELECT the output 'y'
    with sel select
        y <= i0 when '0',       -- When sel is '0', y gets i0
             i1 when '1',       -- When sel is '1', y gets i1
             'X' when others;  -- For any other value (like 'U' or 'Z'), output 'X' (unknown)

end Behavioral;
```

- **Process Statement :** A method that can be used to execute many instructions sequentially. This section will be discussed in more detail in module 3 regarding Behavorial Style.

Revision #6
Created 2025-09-05 15:52:16 UTC by AX
Updated 2025-09-05 16:17:28 UTC by AX