While Loop and For Loop

What is looping in VHDL?

A looping construct (looping statement) in VHDL is an instruction that allows a program to repeat the same block of code iteratively. In VHDL, there are two types of looping constructs: the while-loop and the for-loop.

While Loop

```
label_name: while (condition) loop
    -- Statements
end loop label_name;
```

For Loop

```
for index in range loop
    -- Statements
end loop;
```

The following are things that must be considered when using the looping construct:

- While-loop and for-loop can only be used inside a process statement.
- Unlike for-loop, for-generate can be used outside the process statement.
- Like process statements and structural assignments, looping constructs can be labeled. However, labeling is optional and only serves to help understand the code, especially in the nested-loop section.

While Loop

While loops operate by repeatedly checking the result of a condition. As long as this condition returns "true," the code within it will continue to execute. The looping process will end when the condition returns a "false" value. While loops are very useful when we don't have exact information

about how many times the code needs to be repeated.

However, it's important to remember that there must be a method in the program to change the result of the condition from "true" to "false" so that the loop can stop. Without such a method, the program will be stuck in an infinite loop.

Here is an example of VHDL code for a Shift Register that can be modified by implementing looping.

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

entity Shift_Register is
    port (
        clk, reset: in std_logic;
        data_in: in std_logic;
        data_out: out std_logic
    );
end entity;

architecture RTL of Shift_Register is
    signal reg: std_logic_vector(7 downto 0) := (others => '0');
begin
    process (clk, reset)
    begin
        if reset = '1' then
            reg <= (others => '0');
        elsif rising_edge(clk) then
            reg(7) <= reg(6);
            reg(6) <= reg(5);
            reg(5) <= reg(4);
            reg(4) <= reg(3);
            reg(3) <= reg(2);
            reg(2) <= reg(1);
            reg(1) <= reg(0);
            reg(0) <= data_in;
        end if;
    end process;

    data_out <= reg(7);
```

```
end architecture;
```

The code above still uses the regular assignment method. To implement while loop, the process in the code above can be replaced with the following code.

```
process (clk, reset)
    variable i: integer := 0;
begin
    if reset = '1' then
        reg <= (others => '0');
    elsif rising_edge(clk) then
        while i < 8 loop
            reg(i) <= reg(i+1);
            i := i + 1;
        end loop;
    end if;
end process;
```

For Loop

A for loop repeats the code within a certain range using an index variable. The code inside is executed once on each iteration. This type of looping is suitable when the number of iterations required is known in advance.

The following are things to keep in mind when using for-loops: The range to be iterated over can be defined directly or taken from another previously declared variable in the form of a vector, array, or list.

- Index variables do not need to be declared beforehand; they will be created automatically.
- Index variables can be used in calculations inside the loop but cannot be manually changed.
- Index variables can only increase or decrease by 1 at each iteration.
- Index variables can be incremented or decremented by changing the range from “i0 to in” to “in downto i0”.

```
process (clk, reset)
begin
    if reset = '1' then
        reg <= (others => '0');
    elsif rising_edge(clk) then
```

```
            for i in 7 downto 1 loop
                reg(i) <= reg(i-1);
            end loop;
            reg(0) <= data_in;
        end if;
    end process;
```

Revision #1
Created 1 March 2025 15:07:21 by GI
Updated 1 March 2025 15:07:50 by GI