For Loop

For Loop

The `for` **loop** is the most common type of sequential loop in VHDL. It is used to repeat a block of code a specific, pre-determined number of times. This makes it ideal for tasks where you know exactly how many iterations you need, such as processing the bits of a vector.

Syntax

The basic structure of a `for` loop is:

```
loop_label: for <index_variable> in <range> loop
    -- code code code...
end loop loop_label;
```

Example:

```
testloop: for i in 0 to 2 loop
    -- code cedo deco ... .. . .
end loop
```

- **`loop_label`** is the name of the loop (**recommended!**). It's not mandatory, but helps a lot in coding.
- **`<index_variable>`** is a temporary var that holds the value of the current iteration.
- **`<range>`**, the sequence of values the index will take. This is defined with `to` for an ascending range (`0 to 7`) or `downto` for a descending range (`7 downto 0`).

Notes

When using a `for` loop, there are a few important rules to remember:

- The index variable (`i`) is created automatically by the loop and does **not** need to be declared in the process.
- You can read the value of `i` within the loop, but you **cannot** manually assign a new value to it.
- The loop index will always increment or decrement by one in each iteration. You cannot specify a different step value.

Example: Init Memory

The code below shows the `for` loop iterating through every address of the memory to write a '0' value to it, like setting a RAM's content to 0.

```
type t_memory is array(0 to 63) of std_logic_vector(7 downto 0);


-- Inside your architecture...
p_Memory_Reset: process (i_Clock, i_Reset)
    variable v_ram : t_memory;
begin
    if (i_Reset = '1') then
        -- Initialize all 64 locations of the RAM to zero.
        -- We use a label "RAM_INIT_LOOP" for clarity.
        RAM_INIT_LOOP: for i in v_ram'range loop
        -- RAM_INIT_LOOP: for i in 0 to 63 loop <- would work too. variable`range is more
flexible
            v_ram(i) := (others => '0');
        end loop RAM_INIT_LOOP;

    elsif rising_edge(i_Clock) then
        -- ... normal memory operation code ...
    end if;
end process p_Memory_Reset;
```

This loop is much cleaner than writing 64 individual lines of code. It uses `v_ram'range` (variable'range) to automatically loop through the entire size of the array.

Revision #1
Created 2025-10-03 16:43:34 UTC by JD
Updated 2025-10-03 16:58:58 UTC by JD