**VARIABLE**
A variable can have a single value, as with a constant, but a variable can be updated using a variable assignment statement. The variable is **updated without any delay** as soon as the statement is executed. Variables must be declared **inside** a process (and are local to the process). The variable declaration is as follows:

```plaintext
variable list_of_variable_names: type [ := initial value] ;
```

A few examples follow:

- `variable CNTR_BIT: bit :=0;
- variable VAR1: boolean :=FALSE;
- variable SUM: integer range 0 to 256 :=16;
- variable STS_BIT: bit_vector (7 downto 0);

The variable SUM, in the example above, is an integer that has a range from 0 to 256 with initial value of 16 at the start of the simulation. The fourth example defines a bit vector or 8 elements: `STS_BIT(7), STS_BIT(6),..., STS_BIT(0)`.

A variable can be updated using a variable assignment statement such as

```plaintext
Variable_name := expression;
```

As soon as the expression is executed, the variable is updated **without any delay**.

---

### Example of a process using Variables

<table>
<thead>
<tr>
<th>architecture VAR of EXAMPLE is</th>
</tr>
</thead>
<tbody>
<tr>
<td>signal TRIGGER, RESULT: integer := 0;</td>
</tr>
<tr>
<td>begin</td>
</tr>
<tr>
<td>process</td>
</tr>
<tr>
<td>variable variable1: integer :=1;</td>
</tr>
<tr>
<td>variable variable2: integer :=2;</td>
</tr>
<tr>
<td>variable variable3: integer :=3;</td>
</tr>
<tr>
<td>begin</td>
</tr>
<tr>
<td>wait on TRIGGER;</td>
</tr>
<tr>
<td>variable1 := variable2;</td>
</tr>
<tr>
<td>variable2 := variable1 + variable3;</td>
</tr>
<tr>
<td>variable3 := variable2;</td>
</tr>
<tr>
<td>RESULT &lt;= variable1 + variable2 + variable3;</td>
</tr>
<tr>
<td>end process</td>
</tr>
<tr>
<td>end VAR</td>
</tr>
</tbody>
</table>

---


---

**FOR LOOP**

Just like any loops in an procedural language, FOR LOOP structure allows the execution of a block of codes iteratively. The LOOP structure should have an index. The syntax is:
FOR i IN 0 To 7 LOOP
    -- statements
END LOOP;

In this example the LOOP will execute the statements 8 times. In VHDL, there is no need to declare the loop variable "i". It is implicitly declared! Most of the synthesis tools support the FOR LOOP construct when the range it iterate is fixed. Usually the LOOP body is unroled (unfolded) and logic described by the statements of the LOOP body is duplicated once for each iteration it specified. Therefore in the previous example the logic is duplicated 8 times. Of course once the logic is duplicated, optimization on these newly generated logic is performed. Here is an example which uses the LOOP statement. It loops for 16 times and assignment a value when it suspends. Remember all we have talked about so far on PROCESS and LOOP. Please read the code and think about the behavior it describes.

ENTITY ex_loop IS
    PORT (a: IN STD_LOGIC_VECTOR(0 TO 15);
        sel: INTEGER RANGE 0 TO 15;
        z: OUT STD_LOGIC);
END ex_loop;

ARCHITECTURE rtl OF ex_loop IS
BEGIN
    ex3 : PROCESS (a, sel)
    BEGIN
        FOR i IN 0 TO 15 LOOP
            IF sel = i then
                z <= a(i);
            END IF;
        END LOOP;
    END PROCESS ex3;
END rtl;

Source: [http://web.engr.oregonstate.edu/~sllu/vhdl/lec2f.html](http://web.engr.oregonstate.edu/~sllu/vhdl/lec2f.html)