Assertion Statement

- Assertion statements are useful in modeling constraints of an entity. For example, you may want to check if a signal value lies within a specified range, or check the setup and hold times for signals arriving at the inputs of an entity. If the check fails, an error is reported.
- The syntax of an assertion statement is

assert *boolean-expression*

[**report *string-expression*]**

[**severity *expression*]:**

Assertion Statement cont..

- If the value of the boolean expression is false, the report message is printed along with the severity level. The expression in the severity clause must generate a value of type SEVERTTY_LEVEL (a predefined enumerated type in the language with values NOTE, WARNING, ERROR, and FAILURE).

- The severity level is typically used by a simulator to initiate appropriate actions depending on its value. For example, if the severity level is ERROR, the simulator may stop the simulation process and provide relevant diagnostic information. At the very least, the severity level is

Other Sequential Statements

- There are two other forms of sequential statements
- 1. Procedure call statement,
- 2. Return statement.

Return statement

- A return statement, which is also a sequential statement, is a special statement that is allowed only within subprograms. The format of a return statement is
- **return *[Expression];***
- The return statement causes the subprogram to terminate and control is returned back to the calling object. All functions must have a return statement and the value of the expression in the return statement is returned to the calling program. For procedures, objects of mode out and inout return their values to the calling program.

Procedure call statement

- Procedures allow decomposition of large behaviors into modular sections.
- Procedures are invoked by using procedure calls. A procedure call can either be a sequential statement or a concurrent statement; this is based on where the actual procedure call statement is present. If the call is inside a process statement or inside another subprogram, then it is a sequential procedure call statement, else it is a concurrent procedure call statement. The syntax of a procedure call statement is

- *procedure-name (list-of-actual-parameters);*

Procedure call statement

- The actual parameters specify the expressions that are to be passed into the procedure and the names of objects that are to receive the computed values from the procedure. Actual parameters may be specified using positional association or named association. For example,

```
ARITH_UNIT (D1, D2, ADD, SUM, COMP); -- Positional association.

ARITH_UNIT (Z=>SUM, B=>D2, A=>D1,
OP=>ADD, ZCOMP=>COMP);
-- Named association.
```

Procedure call statement cont..

- A sequential procedure call statement is executed sequentially with respect to the sequential statements surrounding it inside a process or a subprogram. A concurrent procedure call statement is executed whenever an event occurs on one of the parameters which is a signal of mode in or inout.

procedure INT_2_VEC (signal D: out BIT_VECTOR;

START_BIT, STOP_BIT: **in INTEGER;**

signal VALUE: in INTEGER);

Concurrent Signal Assignment Statement

- One of the primary mechanisms for modeling the dataflow behavior of an entity is by using the concurrent signal assignment statement. An example of a dataflow model for a 2-input or gate.

```
entity OR2 is
port (signal A, B: in BIT; signal Z: out BIT);
end OR2;
architecture OR2 of OR2 is
begin
Z <= A or B after 9 ns;
end OR2;
```

Concurrent versus Sequential Signal Assignment

- Signal assignment statements can also appear within the body of a process statement. Such statements are called *sequential signal assignment statements, while signal assignment statements that appear outside of a process are called concurrent signal assignment statements.*
- *Concurrent signal assignment statements are event triggered, that is, they are executed whenever there is an event on a signal that appears in its expression, while sequential signal assignment statements are not event triggered and are executed in sequence in relation to the other sequential statements that appear within the process.*

Example

```
architecture SEQ_SIG_ASG of FRAGMENT1 is
- A, B and Z are signals.
begin
process (B)
begin -- Following are sequential signal assignment statements:
    A<=B;
    Z<=A;
end process;
end;
```

```
architecture CON_SIG_ASG of FRAGMENT2 is
begin -- Following are concurrent signal assignment
   statements:
A<=B;
Z<=A;
end;
```