



Basic Syntax and Structure

Entity Declaration

The `entity` declaration defines the interface of a design.

```
entity entity_name is
    port (
        port_name : mode data_type;
        ...
    );
end entity entity_name;
```

Example:

```
entity AND2 is
    port (
        A : in std_logic;
        B : in std_logic;
        Y : out std_logic
    );
end entity AND2;
```

Architecture Body

The `architecture` body implements the behavior of the entity.

```
architecture architecture_name of
entity_name is
    -- Declarations (signals, components,
    etc.)
begin
    -- Concurrent statements
end architecture architecture_name;
```

Example:

```
architecture Behavioral of AND2 is
begin
    Y <= A and B;
end architecture Behavioral;
```

Libraries and Packages

VHDL uses libraries and packages for predefined types and functions.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

- `ieee.std_logic_1164`: Standard logic types (`std_logic`, `std_logic_vector`).
- `ieee.numeric_std`: Arithmetic operations on `std_logic_vector`.

Data Types and Operators

Standard Data Types

<code>std_logic</code>	Represents a single bit with nine possible values ('U', 'X', 'O', '1', 'Z', 'W', 'L', 'H', '-').
<code>std_logic_vector</code>	An array of <code>std_logic</code> elements. <code>std_logic_vector(7 downto 0)</code> .
<code>integer</code>	Represents signed integer values. Range is implementation-dependent.
<code>boolean</code>	Represents boolean values (<code>TRUE</code> or <code>FALSE</code>).
<code>real</code>	Represents floating-point values.
<code>time</code>	Represents time values with a specified unit (e.g., <code>10 ns</code>).

Operators

Logical Operators	<code>and</code> , <code>or</code> , <code>nand</code> , <code>nor</code> , <code>xor</code> , <code>xnor</code> , <code>not</code>
Relational Operators	<code>=</code> , <code>/=</code> , <code><</code> , <code><=</code> , <code>></code> , <code>>=</code>
Arithmetic Operators	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>mod</code> , <code>rem</code> , <code>abs</code> , <code>**</code>
Shift Operators	<code>sll</code> , <code>srl</code> , <code>sla</code> , <code>sra</code> , <code>rol</code> , <code>ror</code>
Concatenation Operator	<code>&</code> (e.g., <code>A & B</code>)

Concurrent and Sequential Statements

Concurrent Statements

Concurrent statements execute in parallel.

- **Signal Assignment:**

```
signal_name <= expression;
```

- **Conditional Signal Assignment:**

```
signal_name <= expression1 when condition else expression2;
```

- **Selected Signal Assignment:**

```
with selector select
  signal_name <= expression1 when choice1,
                           expression2 when choice2,
                           ...
                           expressionN when others;
```

- **Process:**

```
process (sensitivity_list)
begin
  -- Sequential statements
end process;
```

Sequential Statements

Sequential statements execute in order within a process.

- **if-then-else:**

```
if condition then
  -- statements
elsif condition then
  -- statements
else
```

```
-- statements
end if;
```

- **case:**

```
case expression is
  when choice1 =>
    -- statements
  when choice2 =>
    -- statements
  when others =>
    -- statements
end case;
```

- **for loop:**

```
for loop_variable in range loop
  -- statements
end loop;
```

- **while loop:**

```
while condition loop
  -- statements
end loop;
```

- **variable assignment:**

```
variable_name := expression;
```

Advanced Concepts

Components

Components are instances of entities used within an architecture.

- **Component Declaration:**

```
component component_name is
  port (
    port_name : mode data_type;
    ...
  );
end component;
```

- **Component Instantiation:**

```
instance_name : component_name
  port map (
    port_name => signal_name,
    ...
  );
```

Example:

```
entity Adder is
  generic (
    WIDTH : integer := 8
  );
  port (
    A : in std_logic_vector(WIDTH-1
downto 0);
    B : in std_logic_vector(WIDTH-1
downto 0);
    Y : out std_logic_vector(WIDTH-1
downto 0)
  );
end entity Adder;
```

Functions and Procedures

Function A function returns a value and cannot have side effects.

```
function function_name
  (parameter_list) return
  return_type is
    -- Declarations
begin
  -- Statements
  return return_value;
end function;
```

Procedure A procedure does not return a value directly and can have side effects.

```
procedure procedure_name
  (parameter_list) is
    -- Declarations
begin
  -- Statements
end procedure;
```

Generics

Generics provide a way to parameterize entities and components.

```
entity entity_name is
  generic (
    generic_name : data_type :=
default_value;
    ...
  );
  port (
    ...
  );
end entity;
```