Comments in VHDL begin with double dashes (no space between them) and continue to the end of the current line. Example:

```
-- this is a comment
```

**Identifier** (naming) rules:
1. Can consist of alphabet characters (a-z), numbers (0-9), and underscore (_)
2. First character must be a letter (a-z)
3. Last character cannot be an underscore
4. Consecutive underscores are not allowed
5. Upper and lower case are equivalent (case insensitive)
6. VHDL keywords cannot be used as identifiers

**VHDL models** consist of two major parts:
1. Entity declaration – defines the I/O of the model
2. Architectural body – describes the operation of the model

**Format of Entity**:
```
entity entity_name is
    generic(generic_name: type := default_value;
          
          generic_name: mode signal_type);
    port(signal_name: mode signal_type;
         
         signal_name: mode signal_type);
end entity entity_name;
```

**Note**: signals of the same mode and signal_type can be grouped on 1 line

**MODE** describes direction of data transfer through port
- in – data flows into the port
- out – data flows out of port only
- buffer – data flows out of port as well as internal feedback
- inout – bi-directional data flow into and out of port

**Note**: ‘buffer’ can be used for any output regardless of feedback.

**SIGNAL_TYPE** defines the data type for the signal(s)
- bit – single signals that can have logic values 0 and 1
- bit_vector – bus signals that can have logic values 0 and 1
- std_logic – same as bit but for standard simulation and synthesis (IEEE standard 1164)
- std_logic_vector – same as bit_vector but IEEE standard for simulation and synthesis

**Note**: All vectors must have a range specified. Example:
```
bit_vector (3 downto 0) or std_logic_vector (3 downto 0)
```

**Note**: For simulation and synthesis, is it best to use std_logic over bit. You must include the library and package declarations in the VHDL model before the entity. Example:
```
library IEEE;
use IEEE.std_logic_1164.all;
```

**Values for std-logic**:
```
U     un-initialized (undefined logic value) Z  high impedance (tri-state)
X     forced unknown logic value W  weak unknown
0     logic 0 L  weak 0
1     logic 1 H  weak 1
-  don’t care value (for synthesis minimization)
```

**Note**: U is the default value for all signals at start of simulation.
REVIEW OF VHDL

Format for Architecture body:
architecture architecture_name of entity_name is
-- data type definitions (ie, states, arrays, etc.)
-- internal signal declarations
signal signal_name: signal_type;
;
signal signal_name: signal_type;
-- component declarations – see format below
-- function and procedure declarations
begin
-- behavior of the model is described here and consists of concurrent interconnecting:
-- component instantiations
-- processes
-- concurrent statements including:
Signal Assignment statements
When-Else statements
With-Select-When statements
end architecture architecture_name;
Note: entity and architecture in the end statement is optional.

Format for component declaration:
component component_name is
generic (generic_name(s): type := initial_value;
;
generic_name(s): type := initial_value);
port (signal_name(s): mode signal_type;
;
signal_name(s): mode signal_type);
end component component_name;
Note: This is the same format as an entity statement but the order of generics and signals do not have to be the same at that of the entity (you can adjust for positional notation below).

Format for component instantiation (keyword notation):
instatiation_label: component_name
   generic map (generic_name => value, -- note , at end & not ;
   :
   generic_name => value) -- note no ; at end
   port map (port_name => signal_name, -- note , at end & not ;
   :
   port_name => signal_name);
Note: There are 2 types of component instantiations: keyword notation and positional notation.
Keyword notation: port_name is the signal_name from the component decalaration (same as original entity). The signal_name given here is the internal signal in the hierarchical design being connected to that particular port_name. The order of the generic values and signals can be in any order in keyword notation since each is associated to a unique generic or port by the => operator.
Positional notation: generic_values and signal_names must appear in the order given in the component declaration in order to connect to the correct generic_name or port_name, respectively.
**Format for process statement:**

```vhdl
process_label: process (sensitivity_list_signal, ..., sensitivity_list_signal)
variable variable_name: type;
variable variable_name: type;
begin
    -- sequential statements describing behavior of process including:
    If-Then-Else statements
    Case-When statements
    For-Loop statements
    While-Loop statements
    Wait statements
end process process_label;
```

**Notes:** The `process_label` is optional. The sensitivity list a list of signals that cause the process to execute when an event occurs on any of these signals. Within the process each statement is executed sequentially and only sequential statements can be used in a process.

**VHDL Modeling Guidelines (for synthesis)** used with great success in industry for past 20 years:

**Two process model:**

1) Synchronous process – single-clock, single-edge (or single active value with modeling latches)
   a. Minimize asynchronous operations of associated with flip-flops
      i. Reset/clear
      ii. Set/preset
   b. Focus on synchronous operation of flip-flops
      i. Reset/clear
      ii. Set/preset
      iii. Clock enable, load
      iv. Simple counting and/or shifting operations (ie, count enable, shift/load) - keep it simple, remember you can always partition out complicated combinational logic as in the Mealy and Moore models
   c. Assign only those signals representing flip-flops

2) Combinational logic process – include all dependencies in sensitivity list
   a. Partition logic functions and focus on one at a time
      i. Use one type of conditional construct for that logic (if-then-else or case-when)
      ii. Completely specify for all conditions
      iii. Assign don’t cares whenever possible (and legitimate)
   b. Assign only those signals representing the outputs of combinational logic functions (do not assign any signals representing flip-flops or latches)
   c. For complicated logic functions use multiple combinational processes
      i. Assign any given signal in one and only one process

**Exceptions to the rule:**

1) When the use of multiple clock edges is required (ie, rising-edge for most flip-flops then falling-edge for a few input or output flip-flops), use two synchronous processes – one for each clock edge. Assign any given signal representing a flip-flop in one and only one of the synchronous processes.

2) You can make very simple signal assignments (with no logic or conditions) using concurrent signal assignments (ie, Z <= A;). This is particularly good for primary outputs to avoid the need for buffer signal types. It is also good for ensuring that primary inputs and outputs meet I/O naming conventions and specifications while using desired internal signals (ie, bit_vectors) for more efficient modeling.
REVIEW OF VHDL

SEQUENTIAL STATEMENTS:
If-Then-Else  general format:
   if (condition) then
      do stuff
   elsif (condition) then
      do more stuff
   else
      do other stuff
   end if;

Note: ‘elsif’ and ‘else’ clauses are optional, BUT an incompletely specified ‘if’ statement (no else) implies a memory element (latch) since all signals retain their value if not specified.

Case-When  general format:
   case expression is
      when value =>
         do stuff
      when value =>
         do more stuff
      when others =>
         do other stuff
   end case;

For-Loop  general format:
   label: for identifier in range loop
      do a bunch of junk
   end loop label;

Note: The label: is optional and the variable k implied in for-loop and does not need to be declared.

While-Loop  general format:
   label: while condition loop
      do silly stuff
   end loop label;

Note: The label: is optional and the variable k must be declared as variable in process (before begin).

CONCURRENT STATEMENTS:
logical operators with signal assignment <=
When-Else  general format:
   expression when condition else
   expression when condition else
   expression when others;

Note: “when others” maybe redundant and incompatible with some tools

With-Select-When  general format:
   with selection select
   expression when condition,
   expression when condition,
   expression when others;

example: Z <= A and B;
example: Z <= A when S = “00” else B when S = “11” else C;
example: Z <= A when “00”, B when “11”, C when others;
OPERATORS:

Logic Operators are the heart of logic equations and conditional statements.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td><code>AND</code></td>
</tr>
<tr>
<td>OR</td>
<td><code>OR</code></td>
</tr>
<tr>
<td>NOT</td>
<td><code>NOT</code></td>
</tr>
<tr>
<td>NAND</td>
<td><code>NAND</code></td>
</tr>
<tr>
<td>NOR</td>
<td><code>NOR</code></td>
</tr>
<tr>
<td>XOR</td>
<td><code>XOR</code></td>
</tr>
<tr>
<td>XNOR</td>
<td><code>XNOR</code></td>
</tr>
</tbody>
</table>

*Note*: there is NO order of precedence so use lots of parentheses.

Relational Operators are primarily used in conditional statements.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>equal to</td>
<td><code>=</code></td>
</tr>
<tr>
<td>not equal to</td>
<td><code>/=</code></td>
</tr>
<tr>
<td>less than</td>
<td><code>&lt;</code></td>
</tr>
<tr>
<td>less then or equal to</td>
<td><code>&lt;=</code></td>
</tr>
<tr>
<td>greater than</td>
<td><code>&gt;</code></td>
</tr>
<tr>
<td>greater than or equal to</td>
<td><code>&gt;=</code></td>
</tr>
</tbody>
</table>

Adding Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>addition</td>
<td><code>+</code></td>
</tr>
<tr>
<td>subtraction</td>
<td><code>-</code></td>
</tr>
<tr>
<td>concatenation</td>
<td><code>&amp;</code></td>
</tr>
</tbody>
</table>

*Example:*

```vhdl
signal A: bit_vector(5 downto 0);
signal B,C: bit_vector(2 downto 0);
B <= '0' & '1' & '0';
C <= '1' & '1' & '0';
A <= B & C; -- A now has “010110”
```

*Note*: For bit vector arithmetic use std_logic_vector and ‘unsigned’ and/or ‘arith’ packages as follows:

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all; or
use IEEE.std_logic_arith.all;
```

Multiplying Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiplication</td>
<td><code>*</code></td>
</tr>
<tr>
<td>division</td>
<td><code>/</code></td>
</tr>
<tr>
<td>modulus</td>
<td><code>mod</code></td>
</tr>
<tr>
<td>rem remainder</td>
<td><code>rem</code></td>
</tr>
</tbody>
</table>

Misc. Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponentiation</td>
<td><code>**</code></td>
</tr>
<tr>
<td>absolute value</td>
<td><code>abs</code></td>
</tr>
<tr>
<td>inversion</td>
<td><code>not</code></td>
</tr>
</tbody>
</table>

Shift Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>shift left logical</td>
<td><code>sll</code></td>
</tr>
<tr>
<td>shift right logical</td>
<td><code>srl</code></td>
</tr>
<tr>
<td>shift left arithmetic</td>
<td><code>sla</code></td>
</tr>
<tr>
<td>shift right arithmetic</td>
<td><code>sra</code></td>
</tr>
<tr>
<td>rotate left</td>
<td><code>rol</code></td>
</tr>
<tr>
<td>rotate right</td>
<td><code>ror</code></td>
</tr>
</tbody>
</table>

*Note*: All shift operators have two operands:

- left operand is bit_vector to shift/rotate
- right operand is integer for # shifts/rotates
- integer same as opposite operator with + integer

Order of Precedence:

<table>
<thead>
<tr>
<th>Highest</th>
<th>Lowest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misc.</td>
<td>Multiplying</td>
</tr>
</tbody>
</table>

Evaluation Rules:

1. Operators evaluated in order of precedence highest are evaluated first
2. Operators of equal precedence are evaluated from left to right
3. Deepest nested parentheses are evaluated first

*Note*: Because of #2 you should use lots of parentheses.
Predefined data types:

- **bit**: ‘0’ or ‘1’ (note that std_logic is not predefined)  
  (it is defined in IEEE library and std_logic_1164 package)
- **boolean**: FALSE or TRUE
- **integer**: 
  
- **time**: 
  
- **real**: -1.0E38 to +1.0E38

Specifying values:

- **Binary bit**: ‘0’ or ‘1’
- **Binary string**: “1010” or B”1010”  
  note: no space between B and “
- **Hex string**: H”0a5c”
- **Octal string**: O”71”
- **Decimal number**: 255