VHDL IDENTIFIERS, SIGNALS, & ATTRIBUTES

**Identifier** (naming) rules:
- Can consist of alphabet characters, numbers, and underscore
- First character must be a letter (alphabet)
- Last character cannot be an underscore
- Consecutive underscores are not allowed
- Upper and lower case are equivalent (case insensitive)
- VHDL keywords cannot be used as identifiers

**Reserved Keywords:**

<table>
<thead>
<tr>
<th></th>
<th>downto</th>
<th>library</th>
<th>postponed</th>
<th>srl</th>
<th>access</th>
<th>else</th>
<th>linkage</th>
<th>procedure</th>
<th>subtype</th>
<th>after</th>
<th>elsif</th>
<th>literal</th>
<th>process</th>
<th>then</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>alias</td>
<td>end</td>
<td></td>
<td></td>
<td></td>
<td>loop</td>
<td>pure</td>
<td></td>
<td>to</td>
<td></td>
</tr>
<tr>
<td>all</td>
<td>entity</td>
<td>map</td>
<td>range</td>
<td></td>
<td>and</td>
<td>exit</td>
<td></td>
<td></td>
<td></td>
<td>mod</td>
<td>record</td>
<td></td>
<td>type</td>
<td></td>
</tr>
<tr>
<td>architecture</td>
<td>file</td>
<td>nand</td>
<td>register</td>
<td></td>
<td>architecture</td>
<td>file</td>
<td>nand</td>
<td>register</td>
<td>unaffected</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>for</td>
<td>new</td>
<td>reject</td>
<td></td>
<td>array</td>
<td>for</td>
<td>new</td>
<td>reject</td>
<td>units</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>assert</td>
<td>function</td>
<td>next</td>
<td>rem</td>
<td></td>
<td>assert</td>
<td>function</td>
<td>next</td>
<td>rem</td>
<td>until</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>attribute</td>
<td>generate</td>
<td>nor</td>
<td>report</td>
<td></td>
<td>attribute</td>
<td>generate</td>
<td>nor</td>
<td>report</td>
<td>use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>begin</td>
<td>generic</td>
<td>not</td>
<td>return</td>
<td></td>
<td>begin</td>
<td>generic</td>
<td>not</td>
<td>return</td>
<td>variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>block</td>
<td>group</td>
<td>null</td>
<td>rol</td>
<td></td>
<td>block</td>
<td>group</td>
<td>null</td>
<td>rol</td>
<td>wait</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>body</td>
<td>guarded</td>
<td>of</td>
<td>ror</td>
<td></td>
<td>body</td>
<td>guarded</td>
<td>of</td>
<td>ror</td>
<td>when</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>buffer</td>
<td>if</td>
<td>on</td>
<td>select</td>
<td></td>
<td>buffer</td>
<td>if</td>
<td>on</td>
<td>select</td>
<td>while</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bus</td>
<td>impure</td>
<td>open</td>
<td>severity</td>
<td></td>
<td>bus</td>
<td>impure</td>
<td>open</td>
<td>severity</td>
<td>with</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>case</td>
<td>in</td>
<td>or</td>
<td>signal</td>
<td></td>
<td>case</td>
<td>in</td>
<td>or</td>
<td>signal</td>
<td>xnor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>component</td>
<td>inertial</td>
<td>others</td>
<td>shared</td>
<td></td>
<td>component</td>
<td>inertial</td>
<td>others</td>
<td>shared</td>
<td>xor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>configuration</td>
<td>inout</td>
<td>out</td>
<td>sla</td>
<td></td>
<td>configuration</td>
<td>inout</td>
<td>out</td>
<td>sla</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>constant</td>
<td>is</td>
<td>package</td>
<td>sll</td>
<td></td>
<td>constant</td>
<td>is</td>
<td>package</td>
<td>sll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disconnect</td>
<td>label</td>
<td>port</td>
<td>sra</td>
<td></td>
<td>disconnect</td>
<td>label</td>
<td>port</td>
<td>sra</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Data Types** for ports and signals

**BIT and BIT VECTOR:**
- BIT VECTOR is an array of BITs
- Can have values: 0 1 Note: 0 is initialization value (first value is initial value)

**STD_LOGIC and STD_LOGIC VECTOR:**
- STD_LOGIC VECTOR is an array of STD_LOGICS
- Can have values: U undefined logic value     
  X forcing unknown (not don’t care) 
  0 
  1 
  Z high impedance (tri-state) 
  W weak unknown 
  L weak 0 
  H weak 1 
  - don’t care 

Note: U is initialization value (first value is initial value)
To use STD_LOGIC and STD_LOGIC_VECTOR, include the following at beginning model:

library IEEE;
use IEEE.std_logic_1164.all;

Some other commonly used IEEE library packages include:
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

These packages allow bit vector arithmetic – very useful for counters, etc.
example: COUNT <= COUNT + 1;  -- increments COUNT value

**Signals** represent wires or outputs of gates, FFs, etc. Ports (ins, outs, inouts) in the entity are signals. Internal signals are often needed in complex models and are declared in the architecture description as follows:

architecture **architecture_name** of **entity_name** is

signal **signal_name**: type;
:

signal **signal_name**: type;

begin
:
end architecture **architecture_name**;

The signal **type** can be bit, bit_vector, std_logic, or std_logic_vector
Signals can be initialized to a beginning value at the declaration **BUT** this is meaningless to synthesis tools since no hardware mechanism exists to produce this “power-up” init value

**Example**: signal COUNT: bit_vector(3 downto 0) := “0101”;
here := means immediate assignment and used to indicate an initial value
but the normal assignment operator is <=

**Example**: COUNT <= “0101”;
this does correspond to synthesizable logic and will be recognized by tools

**Attributes** provide information about items such as signals. The most important signal attribute is ‘event which yields a Boolean value of true if an event has just occurred on the signal to which the attribute is applied an event on a signal means a change in value
signal’event allows us to condition on a transition for FFs

**Example**: entity DFF is

port (CK, D: in bit;
       Q: out bit);
end entity DFF;

architecture AUFB of DFF is
begin
process (CK) begin
  if (CK’event and CK=’1’) then Q <= D;  -- occurs only on the rising edge of CK
end if;
end process;
end architecture AUFB;