# 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:**

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

**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 and 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)

# VHDL IDENTIFIERS, SIGNALS, & ATTRIBUTES

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;
```