VHDL - Data Types, User-defined Types

Topics

- Pre-Defined Data Types
- User-Defined Data Types
- Subtypes
- Arrays
- Port Array
- Records
- Signed and Unsigned Data Types
- Data Conversion

In order to write VHDL code efficiently it is essential to know

- What data types are allowed.
- How to specify and use them.
- What data types are synthesizable.
- Data compatibility and data conversion
- Remember: operations between data of different types are not allowed (unless using data conversion).

Architecture declarative **objects**

- In ARCHITECTURE declarative part, we can declare the two next objects:

  CONSTANT and **SIGNAL**

- **CONSTANT** object:
  - ✓ Serves to establish default values.
  - ✓ Can be declared in a **PACKAGE**, **ENTITY**, or **ARCHITECTURE**.
  - ✓ When declared in a **package**, it is truly global (for the entities that using the package).
  - ✓ When declared in an **entity** (after PORT), it is global to all architectures that follow that top entity.
  - ✓ when declared in an **architecture** (in its declarative part), it is local to that architecture's code.
- Syntax:

CONSTANT name : data_type := value ;

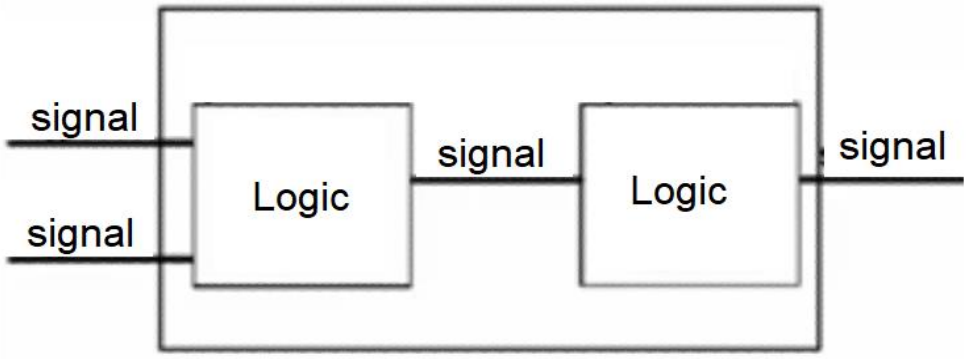
```
CONSTANT bus_width : INTEGER := 16 ;
```

Architecture declarative **objects**

- **SIGNAL** object:
  - ✓ SIGNAL serves to pass values in and out the circuit, as well as between its internal units (a signal represents circuit interconnects).
  - ✓ All PORTS of an ENTITY are signals by default (seen by its **architecture** only).
  - ✓ When declared in an **architecture** (in its declarative part), it uses only as interconnections between its internal units (seen by **architecture** only).
  - ✓ Direct **SIGNAL**s of different **ENTITY**s must not be wired (their internal and can be seen by their **architecture** only), it's possible using intermediate **SIGNAL**s.
  - ✓ Syntax:

SIGNAL signal_name : type [:= initial_value];

```
sig_name <= sig_value;
```



Pre-Defined Data Types

BIT data type: two-level values logic '0', '1'.

```
-- x is declared as a one-digit signal of type BIT.
SIGNAL x: BIT;

-- y is a 4-bit vector, with the leftmost bit being the MSB.
SIGNAL y: BIT_VECTOR (3 DOWNTO 0);

-- w is an 8-bit vector, with the rightmost bit being the MSB.
SIGNAL w: BIT_VECTOR (0 TO 7);

    x <= '1'; -- single quotes (' ') are used for a single bit.

    y <= "0111";  -- double quotes (" ") are used for vectors.
    -- y is a 4-bit signal, whose value is "0111"(MSB='0').

    w <= "01110001";
    -- w is an 8-bit signal,whose value is "01110001" (MSB='1').
```

Pre-Defined Data Types

BOOLEAN data type: True, False.

- The Boolean type is used for conditional operations.
- The default value of any object of the Boolean type is false.
- Boolean values (false and true) are NOT identical to logical 0 and 1, respectively.

```
SIGNAL a,b: BOOLEAN;
    a <= true;
    b <= false;
```

Pre-Defined Data Types

INTEGER data type: 32-bit integers (from -2,147,483,647 to +2,147,483,647).

```
SIGNAL e: INTEGER RANGE 0 TO 255;
    e <= 120 ; -- integer
```

Integer Sub Types:

- **NATURAL** data type: Non-negative integers (from 0 to +2,147,483,647).
- **POSITIVE** data type: Non-negative integers (from 1 to +2,147,483,647).

Data Types - STD_LOGIC_1164 IEEE package

STD_LOGIC , STD_LOGIC_VECTOR data types:

- *A 9-value resolved logic type Std_logic is not a part of the VHDL Standard. It is defined in IEEE Std 1164.*

  'U‘ - Uninitialized
  ‘X’ - Forcing Unknown (**synthesizable** unknown)
  ‘0’ - Forcing Low (**synthesizable** logic ‘1’)
  ‘1’ - Forcing High (**synthesizable** logic ‘0’)
  ‘Z’ - High impedance (**synthesizable** tri-state buffer)
  ‘W’ - Weak unknown
  ‘L’ - Weak low
  ‘H’ - Weak high
  ‘–’ - Don’t care

STD_LOGIC , STD_LOGIC_VECTOR - examples

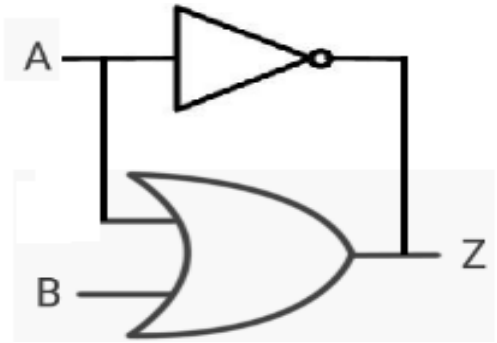
```
-- y is declared as a one-digit signal of type STD_LOGIC.
SIGNAL y: STD_LOGIC;
-- w is an 8-bit vector, with the rightmost bit being the MSB.
SIGNAL w: STD_LOGIC_VECTOR (0 TO 7);
-- temp is a 8-bit vector, with the leftmost bit being the MSB.
SIGNAL temp: STD_LOGIC_VECTOR (7 DOWNTO 0);

y <= '1' ;
w <= "10101010" ;       -- 8-bit, Binary representation assignment(msb=0)
temp <= "10101010" ;    -- 8-bit, Binary representation assignment(msb=1)
temp <= B"1010_1010" ; -- 8-bit, Binary representation assignment
temp <= x"AA" ;         -- 8-bit, Hex representation assignment
temp <= O"252" ;        -- 8-bit, Octal representation assignment
temp(7) <= '1' ;                    -- single bit assignment
temp(7 DOWNTO 4) <= "1010" ;    -- Partial bits assignment
temp <= ('1','0','1','0','1','0','1','0');
temp <= (1=>'1', 3=>'1', 5=>'1', 7=>'1',OTHERS=>'0');
temp <= (OTHERS=>'0', 3=>'1', 1=>'1', 5=>'1', 7=>'1');
temp <= "1010"&"1010";
```

STD_LOGIC - Resolved logic system

If multiple drivers are driving different values onto a signal of the **std_logic** type, the signal value will be decided by the next *resolution* table (The type has a built-in mechanism to determine what the signal value should be):

	X	0	1	Z	W	L	H	-
X	X	X	X	X	X	X	X	X
0	X	0	X	0	0	0	0	X
1	X	X	1	1	1	1	1	X
Z	X	0	1	Z	W	L	H	X
W	X	0	1	W	W	W	W	X
L	X	0	1	L	W	L	W	X
H	X	0	1	H	W	W	H	X
-	X	X	X	X	X	X	X	X



std_ulogic data type – is the unresolved type

```
signal Sig1 : std_logic;
    Sig1 <= '0';
```

User-Defined Data Types

- VHDL also allows the user to define his own data types of two categories **integer** and **enumerated**.
- User defined data types are usually defined in the declarative part of the architecture unit, or it is defined in the packet unit.
- Syntax:

```
type user_define_data is range_specification;
```

User-defined **integer** types

- Definition: The **integer** type is a scalar whose set of values includes integer numbers of the specified range.
- Syntax:

```
TYPE type_name IS RANGE left_limit TO right_limit;
```

- Examples:

```
TYPE integer IS RANGE -2147483647 TO +2147483647;
-- This is indeed the pre-defined type INTEGER.

TYPE natural IS RANGE 0 TO +2147483647;
-- This is indeed the pre-defined type NATURAL.

TYPE my_integer IS RANGE -32 TO 32;
-- A user-defined subset of integers.

TYPE student_grade IS RANGE 0 TO 100;
-- A user-defined subset of integers or naturals.
```

User-defined **enumerate** types

- Definition: The **enumeration** type is a type whose values are defined by listing (enumerating) them explicitly.
- Note: The encoding of enumerated types is done sequentially and automatically unless specified otherwise by a user-defined.
- Syntax:

```
TYPE  type_name IS (values_separated_by_commas) ;
```

- Examples:

```
TYPE bit IS ('0', '1');
-- This is indeed the pre-defined type BIT

TYPE my_logic IS ('0', '1', 'Z');
-- A user-defined subset of std_logic.

TYPE state IS (idle, forward, backward, stop);
-- An enumerated data type, typical of finite state machines.

TYPE color IS (red, green, blue, white);
-- default encoding: red="00",green="01",blue="10",white="11".
```

Subtypes

- A **SUBTYPE** is a **TYPE** with a constraint.
- The main reason for using a subtype rather than specifying a new type is that, though operations between data of different types are not allowed, they are allowed between a subtype and its corresponding base type.
- Syntax:

```
subtype subtype_name is base_type range range_constraint;
```

- Examples:

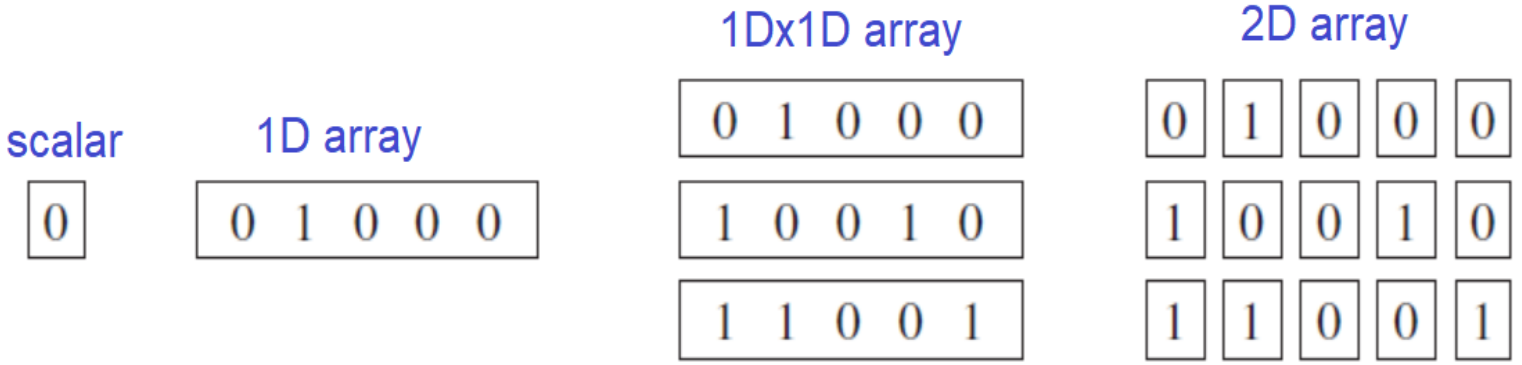
```
SUBTYPE my_logic IS STD_LOGIC RANGE '0' TO 'Z';
-- Recall that STD_LOGIC=('X','0','1','Z','W','L','H','-').
-- Therefore, my_logic=('0','1','Z').

SUBTYPE small_integer IS INTEGER RANGE -32 TO 32;
-- A subtype of INTEGER

type BIT_VECTOR is array(0 to 31) of BIT;
subtype MY_VECTOR is BIT_VECTOR(0 to 15);
-- A subtype of BIT_VECTOR
```

Arrays

- Arrays are collections of objects of the same type.
- The allowed data types are: **CONSTANT, SIGNAL, VARIABLE**.
- They can be **1D**, **2D**, or **1Dx1D** (higher dimensions, are not synthesizable).



- Syntax:

```
TYPE array_type_name IS ARRAY array_size OF data_type;
```

Arrays - Examples

```
TYPE byte IS ARRAY (7 DOWNTO 0) OF STD_LOGIC; -- 1D array
TYPE mem1 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC; -- 2D array
TYPE mem2 IS ARRAY (0 TO 3) OF byte; -- 1Dx1D array
TYPE mem3 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(0 TO 7); -- 1Dx1D array

SIGNAL x: byte; -- 1D signal
SIGNAL w1: mem1; -- 2D signal
SIGNAL w2: mem2; -- 1Dx1D signal
SIGNAL w3: mem3; -- 1Dx1D signal

    x <= "11111110"; -- same types (STD_LOGIC)
    w1(2,5) <= x(3); -- same types (STD_LOGIC)
    w2(0)(0) <= x(2); -- same types (STD_LOGIC)
    w2(2)(5) <= x(7); -- same types (STD_LOGIC)
    w1(2,5) <= w2(3)(7); -- same types (STD_LOGIC)
    w2(0)(7 DOWNTO 0) <= "11110000";
    w3(1) <= "00000000";
    w3(1) <= (OTHERS => '0');
    w2 <= ((OTHERS=>'0'),(OTHERS=>'0'),(OTHERS=>'0'),(OTHERS=>'0'));
    w3 <= ("11111100", ('0','0','0','0','Z','Z','Z','Z',),
          (OTHERS=>'0'), (OTHERS=>'0'));
    w1 <= ((OTHERS=>'Z'), "11110000" ,"11110000", (OTHERS=>'0'));
```

Records

- Definition: Records are similar to arrays, with the only difference that they contain objects of different types.
- Example:

```
constant LEN:  INTEGER := 8;
subtype BYTE_VEC is BIT_VECTOR(LEN-1 downto 0);
type BYTE_AND_IX is record
    BYTE: BYTE_VEC; -- field 1
    IX:   INTEGER range 0 to LEN; -- field 2
end record;

    signal X, Y, Z: BYTE_AND_IX;  --  record variables
    signal DATA: BYTE_VEC;
    signal NUM:  INTEGER;
                . . .
    X.BYTE <= "11110000";
    X.IX   <= 2;
    DATA <= Y.BYTE;
    NUM  <= Y.IX;
```

Signed and Unsigned Data Types

- **Signed** and **Unsigned** Data Types defined in the *std_logic_arith* package of the ieee library, their syntax is similar to that of **STD_LOGIC_VECTOR**.
- An **UNSIGNED** value is a number never lower than zero.
- An **SIGNED** value is a number in two's complement format.
- Their intended mainly for arithmetic operations, that is, contrary to STD_LOGIC_VECTOR (On the other hand, logical operations are not allowed).
- Examples:

```
SIGNAL a: SIGNED (7 DOWNTO 0);
SIGNAL b: SIGNED (7 DOWNTO 0);
SIGNAL y: UNSIGNED (0 TO 3);
          ...
v <= a + b;   -- legal (arithmetic operation OK)
w <= a AND b; -- illegal (logical operation not OK)
```

packages **std_logic_signed** and **std_logic_unsigned**

- There is a simple way of allowing data of type **STD_LOGIC_VECTOR** to participate directly in arithmetic operations.
- For that, the ieee library provides two packages, *std_logic_signed* and *std_logic_unsigned*.
- Examples:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all; -- extra package included

SIGNAL a: STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL x: STD_LOGIC_VECTOR (7 DOWNTO 0);
                ...
v <= a + b; -- legal (arithmetic operation OK), unsigned
w <= a AND b; -- legal (logical operation OK)
```

Data Conversion

- VHDL does not allow direct operations (arithmetic, logical, etc.) between data of different types.
- It is often necessary to convert data from one type to another. for that, we invoke a FUNCTION from a pre-defined PACKAGE which is capable of doing it for us.
- Conversion functions:

```
function CONV_UNSIGNED(ARG: INTEGER; SIZE: INTEGER) return UNSIGNED;
function CONV_UNSIGNED(ARG: UNSIGNED; SIZE: INTEGER) return UNSIGNED;
function CONV_INTEGER(ARG: UNSIGNED) return INTEGER;
function CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER) return STD_LOGIC_VECTOR;
```

- Overloading operators:

```
function "+"(L: UNSIGNED; R: UNSIGNED) return UNSIGNED;
function "-"(L: UNSIGNED; R: UNSIGNED) return UNSIGNED;
```

Data Conversion

- Example 1:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

TYPE long IS INTEGER RANGE -100 TO 100;
TYPE short IS INTEGER RANGE -10 TO 10;
SIGNAL x : short;
SIGNAL y : long;
          ...
y <= 2*x + 5;          -- error, type mismatch
y <= long(2*x + 5); -- OK, result converted into type long
```

Data Conversion

- Example 2:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
          ...
SIGNAL a: UNSIGNED (7 DOWNTO 0);
SIGNAL b: UNSIGNED (7 DOWNTO 0);
SIGNAL y: STD_LOGIC_VECTOR (7 DOWNTO 0);
          ...
y <= CONV_STD_LOGIC_VECTOR ((a+b), 8);
```

```
function CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER) return STD_LOGIC_VECTOR;
```

Synthesizable data types

Data types	Synthesizable values
BIT, BIT_VECTOR	'0', '1'
STD_LOGIC, STD_LOGIC_VECTOR	'X', '0', '1', 'Z' (resolved)
STD_ULOGIC, STD_ULOGIC_VECTOR	'X', '0', '1', 'Z' (unresolved)
BOOLEAN	True, False
NATURAL	From 0 to +2, 147, 483, 647
INTEGER	From −2,147,483,647 to +2,147,483,647
SIGNED	From −2,147,483,647 to +2,147,483,647
UNSIGNED	From 0 to +2,147,483,647
User-defined integer type	Subset of INTEGER
User-defined enumerated type	Collection enumerated by user
SUBTYPE	Subset of any type (pre- or user-defined)
ARRAY	Single-type collection of any type above
RECORD	Multiple-type collection of any types above