Data Types
Vector, bits, etc

This chapter discusses different data types availables in VHDL

VHDL
VHSIC HARDWARE
DESCRIPTION LANGUAGE

Data Types

VHDL Pre-Defined Data Types

- Package standard of library std, defines:

  BIT, **BOOLEAN**, **INTEGER**, and **REAL** data types.

- Package std_logic_1164 of library ieee, defines:

  STD_LOGIC and **STD_ULOGIC** data types.

- Package numeric (formal std_logic_arith by synopsis) of library ieee, defines:

  SIGNED and **UNSIGNED** data types, plus several data conversion functions,
  like conv_integer(p), conv_unsigned(p, b), conv_signed(p, b), and conv_std_logic_vector(p, b).

- Packages std_logic_signed and std_logic_unsigned of library ieee: Contain functions that allow operations with **STD_LOGIC_VECTOR** data to be performed as if the data were of type **SIGNED** or **UNSIGNED**, respectively

VHDL Pre-Defined Data Types

- **BIT** : It can only have the value 0 or 1. When assigning a value of 0 or 1 to a BIT in VHDL code, the 0 or 1 must be enclosed in single quotes: '0' or ‘1’.

Example:

```
SIGNAL x: BIT;          -- x is declared as a one-digit signal of type BIT.
x <= '1’;               -- x is a single-bit signal (as specified above), whose value is
```

- **BIT_VECTOR:** The BIT_VECTOR data type is the vector version of the BIT type consisting of two or more bits. Each bit in a BIT_VECTOR can only have the value 0 or 1. When assigning a value to a BIT_VECTOR, the value must be enclosed in double quotes, e.g. "1011" and the number of bits in the value must match the size of the BIT_VECTOR.

Example:

```
SIGNAL y: BIT_VECTOR (3 DOWNTO 0); -- y is a 4-bit vector, with the leftmost bit being the MSB.
y <= "0111";                       -- y is a 4-bit signal (as specified above), whose value is "0111"
                                   -- (MSB='0'). Notice that double quotes (" ") are used for vectors.
```

Data Types

VHDL Pre-Defined Data Types

- **STD_LOGIC** : Data type can have 8 logic values. When assigning the value, it must be in enclosed in single quotes.

  - ‘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

Only four logic values are synthesizable, i.e. usable for programming FPGA or CPLD. The rest can be used for simulation.

- **STD_LOGIC_VECTOR:** Vector version of STD_LOGIC. Each bit can also take same 8 logic values. Value must be enclosed in double quotes

```
SIGNAL x: STD_LOGIC;                          -- x is declared as a one-digit (scalar) signal of type STD_LOGIC.
SIGNAL y: STD_LOGIC_VECTOR (3 DOWNTO 0) := "0001";   -- y is declared as a 4-bit vector, with the leftmost bit being MSB.
                                              -- The initial value (optional) of y is "0001“ (use ":=“)
```

VHDL Pre-Defined Data Types

- If in a real implementation two signals connected to the same node with conflict of logic value. It is solved based on the following table. For instance, if one signal is 1 and the other is weak low, then it is forced to 1.

'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

	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

Resolved Logic System.
Note: W is forced to X.

STD_ULOGIC and STD_ULOGIC_VECTOR: Is same as studied but with an extra "Unresolved" (U) logic state. In this case, conflicting logic levels are not automatically resolved. Therefore, output wires are never supposed to be connected. Otherwise, we will face an error. (unless we are trying to look for that error)

Data Types

VHDL Pre-Defined Data Types Examples

- x0 <= '0’; -- bit, std_logic, or std_ulogic value '0'
- x1 <= "00011111"; -- bit_vector, std_logic_vector,
  -- std_ulogic_vector, signed, or unsigned
- x2 <= "0001_1111"; -- underscore allowed to ease visualization
- x3 <= "101111" -- binary representation of decimal 47
- x4 <= B"101111" -- binary representation of decimal 47
- x5 <= O"57“ -- octal representation of decimal 47
- x6 <= X"2F" -- hexadecimal representation of decimal 47
- n <= 1200; -- integer
- m <= 1_200; -- integer, underscore allowed
- IF ready THEN... -- Boolean, executed if ready=TRUE
- y <= 1.2E-5; -- real, not synthesizable
- q <= d after 10 ns; -- physical, not synthesizable

LEGAL and ILEGAL Assignment

```
SIGNAL a: BIT;
SIGNAL b: BIT_VECTOR(7 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;
...
a <= b(5); -- legal (same scalar type: BIT)
b(0) <= a; -- legal (same scalar type: BIT)
c <= d(5); -- legal (same scalar type: STD_LOGIC)
d(0) <= c; -- legal (same scalar type: STD_LOGIC)
a <= c; -- illegal (type mismatch: BIT x STD_LOGIC)
b <= d; -- illegal (type mismatch: BIT_VECTOR x STD_LOGIC_VECTOR)
e <= b; -- illegal (type mismatch: INTEGER x BIT_VECTOR)
e <= d; -- illegal (type mismatch: INTEGER x STD_LOGIC_VECTOR)
```

Data Types

VHDL Pre-Defined Data Types

- **BOOLEAN**: True, False.
- **INTEGER**: 32-bit integers (from -2,147,483,647 to +2,147,483,647).
- **NATURAL**: Non-negative integers (from 0 to +2,147,483,647).
- **REAL**: Real numbers ranging from 1.0E38 to þ1.0E38. Not synthesizable.
- **Physical literals**: Used to inform physical quantities, like time, voltage, etc. Useful in simulations. Not synthesizable.
- **Character literals**: Single ASCII character or a string of such characters. Notsynthesizable.
- **SIGNED** and **UNSIGNED**: data types defined in the std_logic_arith package of the ieee library. They have the appearance of STD_LOGIC_VECTOR, but accept arithmetic operations, which are typical of INTEGER data types

VHDL Pre-Defined Data Types

- **Exercise**

Write the following program in VIVADO and check if the assignment is legal or illegal and identify why.

```
SIGNAL a: BIT;
SIGNAL b: BIT_VECTOR(7 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;
... (define your own entity and architecture)
a <= b(5);
b(0) <= a;
c <= d(5);
d(0) <= c;
a <= c;
b <= d;
e <= b;
e <= d;
```

```
a <= b(5); -- legal (same scalar type: BIT)
b(0) <= a; -- legal (same scalar type: BIT)
c <= d(5); -- legal (same scalar type: STD_LOGIC)
d(0) <= c; -- legal (same scalar type: STD_LOGIC)
a <= c; -- illegal (type mismatch: BIT x STD_LOGIC)
b <= d; -- illegal (type mismatch: BIT_VECTOR x
-- STD_LOGIC_VECTOR)
e <= b; -- illegal (type mismatch: INTEGER x BIT_VECTOR)
e <= d; -- illegal (type mismatch: INTEGER x
-- STD_LOGIC_VECTOR)
```

Data Types

VHDL User Define Data Types

- VHDL Allows user to define their own type of variables.

User-defined *integer* types:

```
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 *enumerated* types (very useful for state machines):

- 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); -- Another enumerated data type.
- Generally, encoding is assigned sequentially and automatically, i.e. *00 for red, 01 for green, 10 for blue and 011 for white*.
- TYPE bit_vector IS ARRAY (NATURAL RANGE <>) OF BIT;
  -- This is indeed the pre-defined type BIT_VECTOR.
  -- RANGE <> is used to indicate that the range is unconstrained.
  -- NATURAL RANGE <>, on the other hand, indicates that the only
  -- restriction is that the range must fall within the NATURAL range.

Data Types

VHDL Sub-Types Data

- A SUBTYPE of data is a TYPE with constrain.
- 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.

Example: Legal and illegal operations between types and subtypes.

```
SUBTYPE my_logic IS STD_LOGIC RANGE '0' TO '1';
SIGNAL a: BIT;
SIGNAL b: STD_LOGIC;
SIGNAL c: my_logic;
...
b <= a; -- illegal (type mismatch: BIT versus STD_LOGIC)
b <= c; -- legal (same "base" type: STD_LOGIC)
```

Data Types

VHDL ARRAYS

- Arrays are collection of objects of the same type.
- Arrays can be one dimensional or two dimensional to be synthesizable. Higher dimensions are not synthesizable.

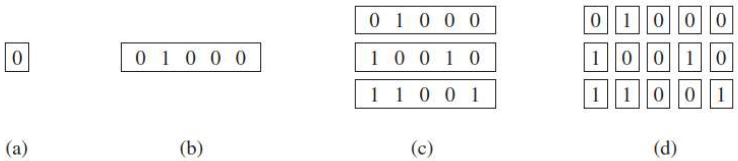


Figure 3.1
Illustration of (a) scalar, (b) 1D, (c) 1Dx1D, and (d) 2D data arrays.

To specify a new array type:
TYPE type_name IS ARRAY (specification) OF data_type;

To make use of the new array type:
SIGNAL signal_name: type_name [:= initial_value];

Example:
TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC; -- 1D array
TYPE matrix IS ARRAY (0 TO 3) OF row; -- 1Dx1D array
SIGNAL x: matrix; -- 1Dx1D signal

Similarly:
TYPE matrix IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);

For 2D Array:
TYPE matrix2D IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;
Initialization:

```
.. :="0001";                                  -- for 1D array
... :=('0','0','0','1')                       -- for 1D array
... :=(('0','1','1','1'), ('1','1','1','0')); -- for 1Dx1D or
                                              -- 2D array
```

Data Types

Array Example

```
TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC; -- 1D array
TYPE array1 IS ARRAY (0 TO 3) OF row; -- 1Dx1D array
TYPE array2 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);-- 1Dx1D
TYPE array3 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC; -- 2D array
SIGNAL x: row;
SIGNAL y: array1;
SIGNAL v: array2;
SIGNAL w: array3;
--------- Legal scalar assignments: ----------
-- The scalar (single bit) assignments below are all legal,
-- because the "base" (scalar) type is STD_LOGIC for all signals
-- (x,y,v,w).
x(0) <= y(1)(2); -- notice two pairs of parenthesis
                 -- (y is 1Dx1D)
x(1) <= v(2)(3); -- two pairs of parenthesis (v is 1Dx1D)
x(2) <= w(2,1); -- a single pair of parenthesis (w is 2D)
y(1)(1) <= x(6);
y(2)(0) <= v(0)(0);
y(0)(0) <= w(3,3);
w(1,1) <= x(7);
w(3,0) <= v(0)(3);
--------- Vector assignments: ----------
x <= y(0); -- legal (same data types: ROW)
x <= v(1); -- illegal (type mismatch: ROW x STD_LOGIC_VECTOR)
x <= w(2); -- illegal (w must have 2D index)
x <= w(2, 2 DOWNTO 0); -- illegal (type mismatch: ROW x STD_LOGIC)
v(0) <= w(2, 2 DOWNTO 0); -- illegal (mismatch: STD_LOGIC_VECTOR x STD_LOGIC)
v(0) <= w(2); -- illegal (w must have 2D index)
y(1) <= v(3); -- illegal (type mismatch: ROW x STD_LOGIC_VECTOR)
y(1)(7 DOWNTO 3) <= x(4 DOWNTO 0); -- legal (same type, same size)
v(1)(7 DOWNTO 3) <= v(2)(4 DOWNTO 0); -- legal (same type, same size)
w(1, 5 DOWNTO 1) <= v(2)(4 DOWNTO 0); -- illegal (type mismatch)
```

Data Types

Port Array

- As studied no pre-defined data types of more than one dimension exist.
- We would like to specify input and output ports of an entity as arrays, but type declarations are not allowed in ENTITY!
- The solution is to declare user-defined data types in PACKAGE- This is visible for the all design, including ENTITY.

The user-defined data type named *vector_array* contains an indefinite number of vectors of 8 bits. It is saved in a package named *my_data_types.* This is used to specify the port named *inp* with 4 vectors of 8 bits.

```
------- Package: -------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
----------------------------
PACKAGE my_data_types IS
TYPE vector_array IS ARRAY (NATURAL RANGE <>) OF
STD_LOGIC_VECTOR(7 DOWNTO 0);
END my_data_types;
-----------------------------------------
------- Main code: -------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_data_types.all; -- user-defined package
---------------------------
ENTITY mux IS
PORT (inp: IN VECTOR_ARRAY (0 TO 3);
... );
END mux;
... ;
-----------------------------------------
```

Data Types

Signed and Unsigned Data Types

- These data type are defined in *std_logic_arith* of the package ieee.
- *Syntax is like STD_LOCIG_VECTOR (not like integer):*
  - *SIGNAL x: SIGNED (7 DOWNTO 0);*
  - *SIGNAL y: UNSIGNED (0 TO 3);*
- *Unsigned uses all bits for number representation. Signed type uses "two complement" format.*
- *SIGNED and UNSIGNED are intended for arithmetic operations. (contrary to STD_LOGIC, which do not accept arithmetic operations). However,* ***logic operations are not accepted.*** *There are no restrictions regarding relational (comparison) operations.*

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;    -- extra package necessary
...
SIGNAL a: IN SIGNED (7 DOWNTO 0);
SIGNAL b: IN SIGNED (7 DOWNTO 0);
SIGNAL x: OUT SIGNED (7 DOWNTO 0);
...
v <= a + b;      -- legal (arithmetic operation OK)
w <= a AND b;    -- illegal (logical operation not OK)
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;    -- no extra package required
...
SIGNAL a: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
...
v <= a + b;      -- illegal (arithmetic operation not OK)
w <= a AND b;    -- legal (logical operation OK)
```

Data Types

Data Conversion

- *VHDL does not allow direct operations between data of different type.*
- *To convert data, it can be done manually based on a FUNCTION from a pre-defined PACKAGE.*
- *OR use the **std_logic_1164** from ieee library which provides **straightforward conversion when data are closely related.***

```
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
```

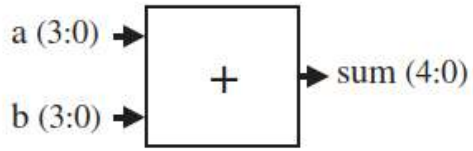
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
...
SIGNAL a: IN UNSIGNED (7 DOWNTO 0);
SIGNAL b: IN UNSIGNED (7 DOWNTO 0);
SIGNAL y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
...
y <= CONV_STD_LOGIC_VECTOR ((a+b), 8);
-- Legal operation: a+b is converted from UNSIGNED to an
-- 8-bit STD_LOGIC_VECTOR value, then assigned to y.
```

There are several data conversión functions in the library.

Data Types

VHDL Pre-Defined Data Types

- **Exercise:** You need to write a code to make a 4-bit adder as the figure. Considering:
- i) All signals are SIGNED type
- ii) Same as before but change the output as INTEGER.




```
1  ----- Solution 1: in/out=SIGNED ----------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE ieee.std_logic_arith.all;
5  ----------------------------------------
6  ENTITY adder1 IS
7     PORT ( a, b : IN SIGNED (3 DOWNTO 0);
8            sum : OUT SIGNED (4 DOWNTO 0));
9  END adder1;
10 ----------------------------------------
11 ARCHITECTURE adder1 OF adder1 IS
12 BEGIN
13    sum <= a + b;
14 END adder1;
15 ----------------------------------------
```

```
1  ------ Solution 2: out=INTEGER -----------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE ieee.std_logic_arith.all;
5  ----------------------------------------
6  ENTITY adder2 IS
7     PORT ( a, b : IN SIGNED (3 DOWNTO 0);
8            sum : OUT INTEGER RANGE -16 TO 15);
9  END adder2;
10 ----------------------------------------
11 ARCHITECTURE adder2 OF adder2 IS
12 BEGIN
13    sum <= CONV_INTEGER(a + b);
14 END adder2;
15 ----------------------------------------
```

END-Data Types
Vector, bits, etc

This chapter discusses different data types availables in VHDL

VHDL
VHSIC HARDWARE
DESCRIPTION LANGUAGE