Continued part of Data type:-

4. Composite types: Array and Record :-

Composite data objects consists of a collection of related data elements in the form of an array or record.

(i) Array type :

type array_name is array (indexing scheme) of element type;

eg:-

type my_word is array (15 downto 0) of std_logic;

type VAR is array (0 to 7) of integer;

type your_word is array (0 to 15) of std_logic;

In the 1st example we have defined a one-dimensional array of elements of type std_logic indexed from 15 down to 0.

And, in 3rd example, array is indexed from 0 to 15.

→ we can now declare objects of these data types:-

eg:- signal mem_addr : my_word;

signal data_word : your_word := "1101101000101100";

∴ in 1st example, signal mem_addr is an array of 16 bits, initialized to all '0's. To access individual elements of an array we specify the index.

For eg:-

- mem_addr(15) accesses left most bit of the array,
- data_word(15) accesses right most bit of the array

(ii) Record type:

- A record consists of multiple elements that may be of different types.

```
type name is
    record
        identifier : subtype_indication;
        :
        identifier : subtype_indication;
    end record;
```

eg:-

```
type my_module is
    record
        rise_time : time;
        fall_time : time;
        size      : integer range 0 to 200;
        data      : bit_vector (15 downto 0);
    end record

signal A, B : my_module;
```

→ To access values or assign values to records, one can use one of the following methods:-

```
A.rise_time <= 5 ns;
A.size <= 120;
B <= A;
```

Type Conversions :

- Since VHDL is a strongly typed language one cannot assign a value of one data type to a signal of a different data type.

To allow assigning data between objects of different types, one needs to convert one type to another type.

Syntax of type conversion:-

```
type_name (expression);
```

In order for the conversion to be legal, the expression must return a type that can be converted into the type type_name.

Following conditions must be fulfilled for the conversion to be possible:

1. type conversions between integer types or between similar array types are possible.
2. Conversion between array types is possible if they have the same lengths and if they have identical element types or convertible element types.
3. Enumerated types cannot be converted.

Operators:

- VHDL supports different classes of operators that operate on signals, variables and constants.

Class						
1. Logical Operator	and	or	nand	nor	xor	xnor
2. Relational Operator	=	/=	<	<=	>	>=
3. Shift Operator	sll	srl	sla	sra	rol	ror
4. Addition operator	+	=	&			
5. ~~Unitary~~ Unary operator	+	—				
6. Multiplying Operator	*	/	mod	rem		
7. Miscellaneous Operator	**	abs	not			

→ the order of precedence is the highest for the operators of class 7, followed by class 6 with the lowest precedence for class 1. Unless parenthesis are used, operators with highest precedence are applied first.

→ Operators of same class have the same precedence and are applied from left to right in an expression.

1. Logical Operators :

- These operators are defined for the "bit", "boolean", "std_logic", and "std_ulogic" types and their vectors.
- they are used to define boolean logic expressions or to perform bit-per-bit operations
- they give result of same type as the operand (bit or Boolean)

Note:- nand and nor are not associative. one should use parenthesis in a sequence of nand or nor operators to prevent a syntax error.

→ X nand Y nand Z will give a syntax error.
∴ is (X nand Y) nand Z.

2. Relational operator :

- These operators test the relative values of two scalar types and give the result a Boolean output of "TRUE" or "FALSE".

- Note that symbol of operator "<=" (smaller or equal to) is same as assignment operator used to assign a value to a signal or variable.

Eg:-

```
variable STS : Boolean;
constant A   : integer := 24;
constant B   : integer := 32;
constant C   : integer := 14
STS <= (A < B);  -- will assign value "TRUE" to STS
STS <= ((A >= B) or (A > C)); -- will result in "TRUE".
```

→ Here, 1st "<=" is assignment operator.

Aanta Officer

3. Shift operators :-

- These operators perform a bit-wise shift or rotate operation on a one-dimensional array of elements of the type bit (or std_logic) or Boolean.

Operator	Description	Operand type
sll	shift left logical (fill right vacated bits with 0).	left : Any one-dimensional array type with elements of type bit or Boolean. Right : integer
srl	Shift right logical (fill left vacated bits with 0)	"
sla	Shift left arithmetic (fill right vacated bits with right most bit)	"
sra	Shift right arithmetic (fill left vacated bits with leftmost bit)	"
rol	Rotate left (Circular)	"
ror	Rotate right (circular)	"

eg:-

```
variable num1 : bit_vector := "10010110";
num1 srl 2;
```

Here, num1 = operand
2 = no. of shifts

Result :- ~~1 0 0 1 0 1 1 0~~

```
1 0 0 1 0 1 1 0
0 1 0 0 1 0 1 1   1st shift
0 0 1 0 0 1 0 1   2nd shift
```

∴ num1 will become 00100101.

$\Rightarrow$ num1 srl -2 would be equivalent to num1 sll 2 and result will be :-

```
  10010110
  00101100   1st
{ 01011000   2nd
↓ result.
```

Now,

eg:- bit_vector A = "101001"

```
variable A : bit_vector := "101001"
{ A sll 2;
  A srl 2;
  A sla 2;
  A sra 2;
  A rol 2;
  A ror 2;
↓ Assignment
```

This is an assignment to determine the result for the above shift operators.