Course Name: Digital Hardware Design
Course Code: 17B1NEC741

VHDL-1

Dr. Arti Noor
Dean, Academic Affairs
Electronics and Communication Engineering,
Jaypee Institute of Information Technology, Noida

General Introduction

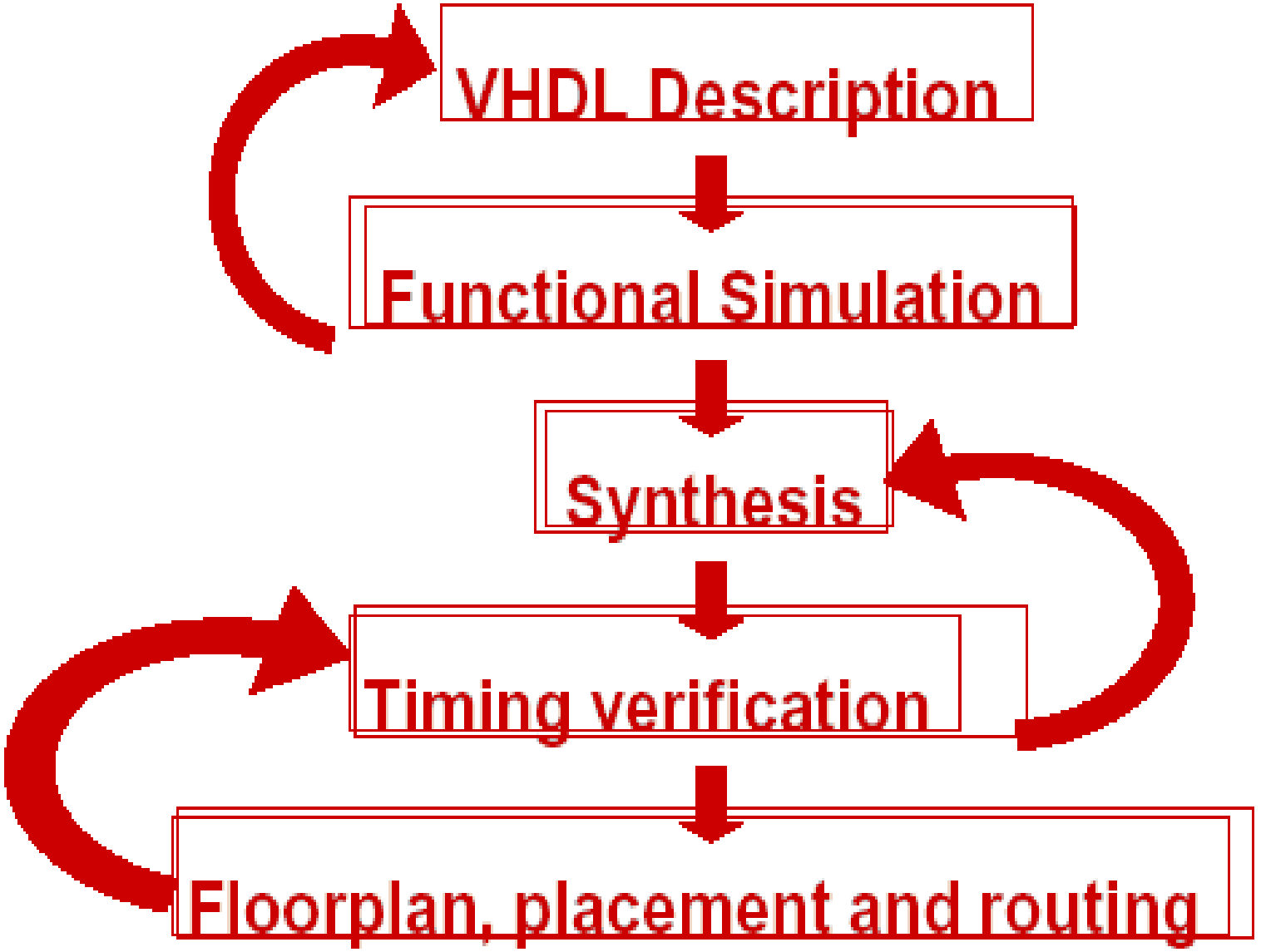
VHDL: VHSIC Hardware Description Language.
VHSIC: Very High Speed Integrated Circuit

Advantages of VHDL

- Supports various design methodologies (Top-down and Bottom-up approach).
- Supports a multi-level abstraction and tight coupling to lower levels of design.
- Supports all CAD tools.
- IEEE Standard with simulation and synthesis .

Flow Chart for the HDL based design

Top Down Approach



General Introduction

Four Basic Elements:

1. Entity
   -Used to define external view of a model.
2. Architecture
   -Used to define function of a model.
3. Configuration
   -Used to associate an Architecture with Entity
4. Package
   -Collection of information referenced by VHDL model(library)

ENTITY

An ENTITY is a list with specifications of all input and output pins (PORTS) of the circuit. It may also include a set of generic values, used to declare circuit properties.

Entity Declaration syntax

```
entity entity_name is
    Generic declarations
    Port declarations
end entity_name;
```

entity_name : any alpha/numeric name except VHDL reserved words.

Port declarations : describe inputs & outputs i.e. pins

Generic declarations: used to pass information into the model

Example

```
1. entity orgate is
2.    port (
3.                a : in   bit;
4.                b : in   bit;
5.                c : out  bit
6.             );
7. end orgate;
```

- The mode of the signal can be IN, OUT, INOUT, or BUFFER.
- IN and OUT are truly unidirectional pins, while INOUT is bidirectional.
- BUFFER, on the other hand, is employed when the output signal must be used (read) internally.
- The type of the signal can be BIT, STD_LOGIC, INTEGER, etc.

Example: for Generic

```
1.   entity Logic_Gates is
2.  generic (Delay : Time := 10ns);
3.  port (
4.        Input1 : in bit;
5.        Input2 : in bit;
6.        Output : out bit
7.        );
8.  end Logic_Gates;
```

Identifiers

Upper and lower case characters are considered to be identical when used as a basic identifier.

ARCHITECTURE

- Architecture describes the functionality and its timing model.
- Must be associated with an entity. Entity can have multiple architectures.
- It can contain both concurrent and sequential statements.

Architecture syntax:

1. architecture architecture_name of entity_name is
2. declaration
3. begin
4. (concurrent statements)
5. end architecture_name;

Example:

```
ENTITY nand_gate IS
    PORT (a, b : IN BIT;
          x : OUT BIT);
END nand_gate;
```

```
ARCHITECTURE myarch OF nand_gate IS
BEGIN
    x <= a NAND b;
END myarch;
```

Data types, Objects and Class

- **Objects mean the name of ports or intermediate signals used for connection.**
- **An object contains a value of a specified type and a class.**
- **Type indicates what type of data the object contains.**
- **Class indicates what can be done with the object**

Example:
signal a : bit;

a is **object**, signal is **class** and bit is **data type**.

- **VHDL is strongly typed language. It means different data types can't be mixed i.e. data of different type can't be assigned to each other.**
- **If we want to assign two objects of different data type then first of all it should be converted to compatible form by using type conversion concept.**

VHDL Class

Four Types:

1. Constants

A constant is an object that holds a single value. Value cannot be changed during the whole code.

Example: constant number_of_bytes integer:=8;

2. Variables

A variable holds a single value of a given type but its value may change during simulation. The assignment operator assigns variables ":=".

Example:

variable index: integer :=0;

VHDL Class

Four Types:

3. Signals

Signals can be declared in architecture and used anywhere within the architecture. Signals are assigned by the assignment operator "**<=**".

Example:

Signal sig1: bit;

Sig1 <= ‘1’

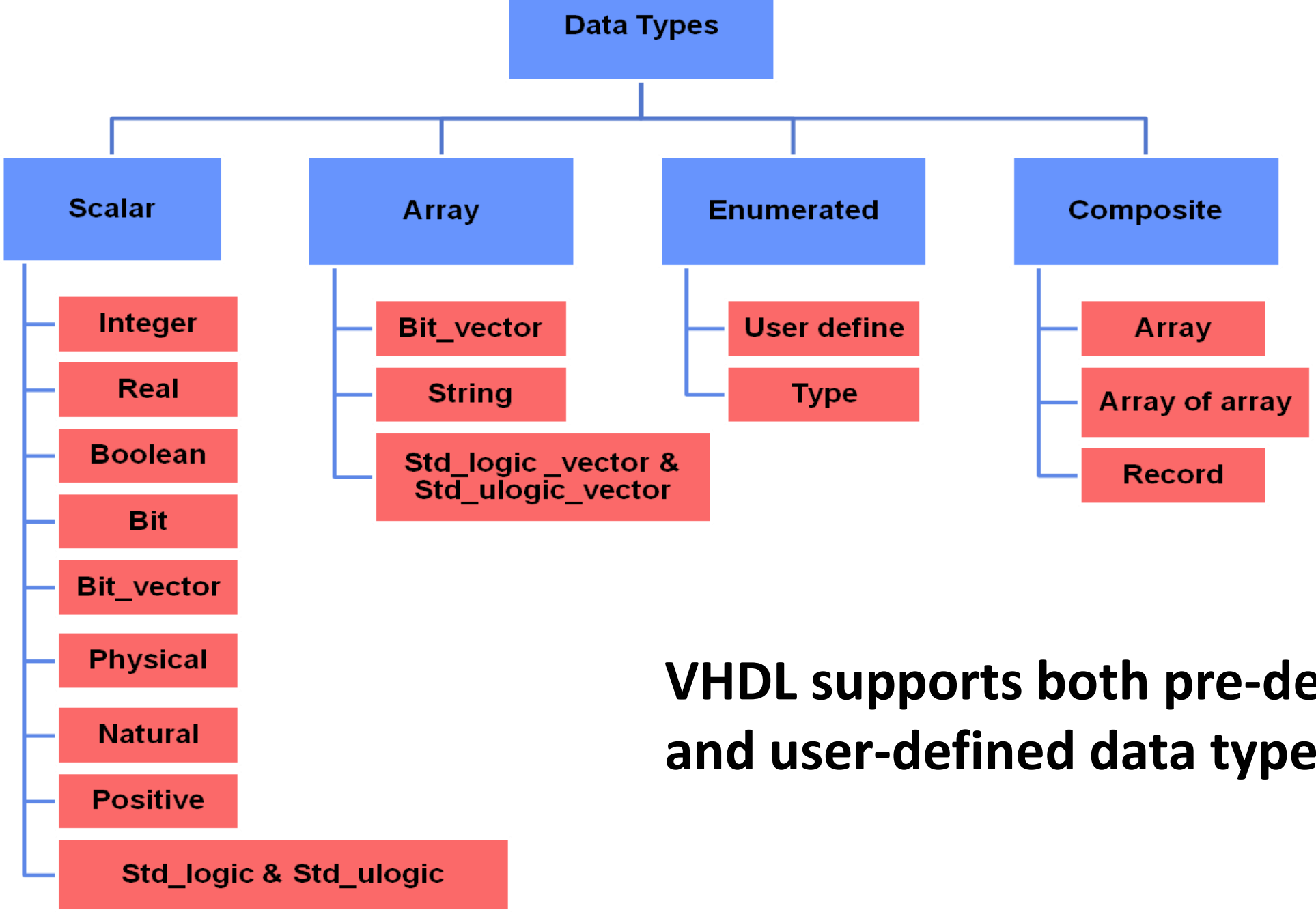
4. File: It consists of sequence of values. Values can be read or written to the file using read procedures and write procedures respectively.

Data Types

VHDL supports both pre-defined and user defined data type

Pre-defined: these data types are part of

- Package standard of library std
- Package std_logic_1164 of library ieee:
- Package std_logic_arith of library ieee:
- Packages std_logic_signed and std_logic_unsigned of library ieee.



VHDL supports both pre-defined and user-defined data types.

Data Types(pre-defined)

Package standard of library std (Included by default):

bit type (0, 1)

bit vectors (group of multi-bit signal → bus)

Example

- SIGNAL x: BIT;
- SIGNAL y: BIT_VECTOR (3 DOWNTO 0);
- SIGNAL w: BIT_VECTOR (0 TO 7);

Signal assignment operator **<=**

- x <= '1';
- y <= "0111";
- w <= "01110001";

Data Types(pre-defined)

Package standard of library std (Included by default):

BOOLEAN (TRUE, FALSE)

- Example
  - variable VAR1: boolean := FALSE;

INTEGER (32 bit, -2,147,483,647 to +2,147,483,647

- Example
  - SIGNAL SUM: integer range 0 to 256 :=16;

REAL (from -1.0E38 to +1.0E38)

- Example
  - constant Pi : real := 3.14159;

BIT (and BIT_VECTOR) leads to several restrictions in the design
For: tri-state logic, unknown and don't care signals can not be assigned.

IEEE defined new data types as std_logic and std_ulogic

STD_LOGIC (and STD_LOGIC_VECTOR)

8-valued logic system introduced in the IEEE 1164 standard.

'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

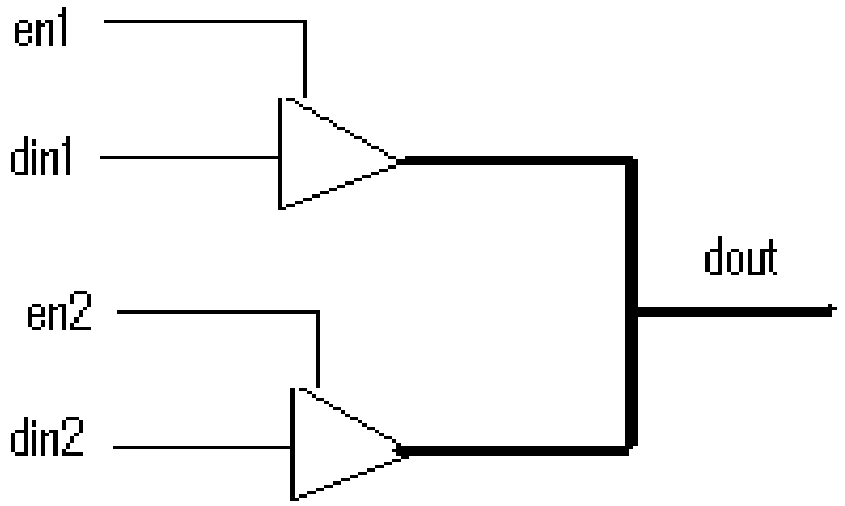
- **STD_ULOGIC type:**
  'U' -- for uninitialized
  and all cases of STD_LOGIC

Resolved Data Type

- Std_logic is resolved means that, if a signal is driven by several drivers, a predefined resolution function is called up, which then resolves the conflict and decides which value the signal will be given.

dout <= din1 when en1=’1’ else ‘z’;

dout <= din2 when en2=’1’ else ‘z’;



If a **std_ulogic** signal is driven by more than one driver, it will result in an error, as VHDL does not permit an unresolved signal to be driven by more than one driver.

std_logic is preferred.

Data Types (Pre-defined)

- **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 std_logic_arith 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.**

Examples:

```
SIGNAL x: STD_LOGIC;
SIGNAL y: STD_LOGIC_VECTOR (3 DOWNTO 0) := "0001“;
```

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 +1.0E38.

Character literals

Single ASCII character or a string of such characters.

SIGNED and UNSIGNED data types

It is defined in the std_logic_arith package of the ieee library.

Examples

```
x0 <= '0';          -- bit, std_logic, or std_ulogic value '0'
x1 <= "00011111";   -- bit_vector, std_logic_vector
x2 <= "0001_1111";  -- underscore allowed to ease visualization
x3 <= "101111"      -- binary representation of decimal 47
```

Data Types (User-Defined)

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

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

Subtypes

A SUBTYPE is a TYPE with a constraint.

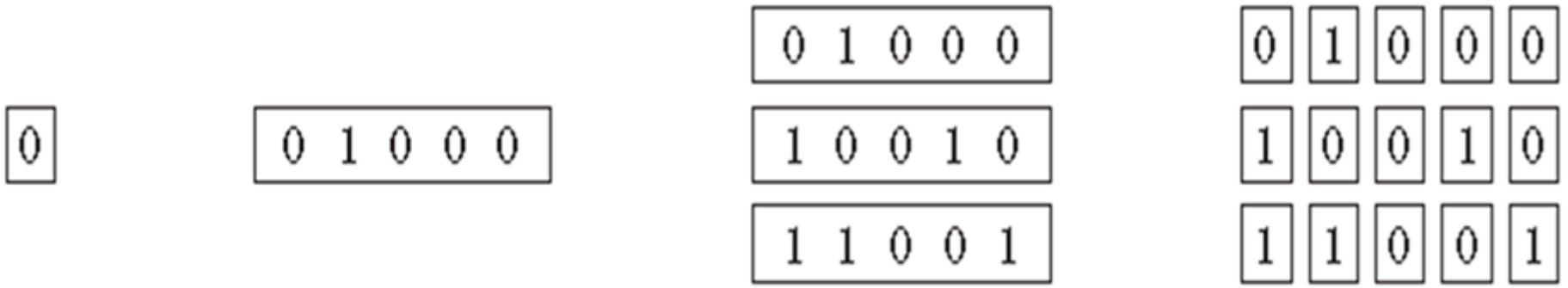
Example

```
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 my_color IS color RANGE red TO blue;
-- Since color=(red, green, blue, white), then
-- my_color=(red, green, blue).
```

Arrays

- Arrays are collections of objects of the same type.
- They can be one-dimensional (1D), two-dimensional (2D), or one-dimensional-by-one-dimensional (1Dx1D).



(a) Salar (b) 1D (c) 1D X 1D (d) 2D

Syntax

TYPE type_name IS ARRAY (specification) of data_type;

Example

```
TYPE row is ARRAY (7 downto 0) of STD_LOGIC;      --1D array
TYPE matrix is ARRAY (0 to 3) of row;             --1D x 1D array
SIGNAL  X: matrix;                                -- 1D x 1D signal
TYPE  T_2D is array (3 downto 0, 1 downto 0) of integer;
signal X_2D : T_2D;
X_2D <= ((0,0), (1,1), (2,2), (3,3));
X_2D(3,1) <= 4;
```

Records

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

Example:

```
TYPE birthday IS RECORD
    day: INTEGER RANGE 1 TO 31;
    month: month_name;
END RECORD;
```

Data Attributes

The pre-defined, synthesizable data attributes are the following:

'High: Returns upper array index
'Low: Returns lower array index
'Left: Returns leftmost array index
'Right: Returns rightmost array index
'Length—returns the length (number of elements) of an array
'RANGE: Returns vector range
'REVERSE_RANGE: Returns vector range in reverse order

Example:
SIGNAL d : STD_LOGIC_VECTOR(7 downto 0);
Then
d'LOW=0, d'HIGH=7, d'LEFT=7, d'RIGHT=0, d'LENGTH=8,
d'RANGE=(7 downto 0), d'REVERSE_RANGE=() to 7).

Signed and Unsigned Data Types

These types are defined in the std_logic_arith package of the ieee library.

Examples:

```
SIGNAL x: SIGNED (7 DOWNTO 0);
SIGNAL y: UNSIGNED (0 TO 3);
```

- **An UNSIGNED value is a number never lower than zero.**
- **For example, “0101” represents the decimal 5, while “1101” signifies 13.**
- **If type SIGNED is used instead, the value can be positive or negative**

Operators

VHDL provides several kinds of pre-defined operators:

- Assignment operators
- Logical operators
- Arithmetic operators
- Relational operators
- Shift operators
- Concatenation operators

Assignment Operators

Are used to assign values to signals, variables, and constants.
They are:
<= Used to assign a value to a SIGNAL.
:= Used to assign a value to a VARIABLE, CONSTANT, or GENERIC. Used also for establishing initial values.
=> Used to assign values to individual vector elements or with OTHERS.

Example:

```
X <= '1';                -- '1' is assigned to signal x
Y := "0000";             --  "0000" is assigned to variable Y
```

Logical Operators

- Used to perform logical operations.

- The logical operators are:
  - NOT
  - AND
  - OR
  - NAND
  - NOR
  - XOR
  - XNOR

Examples:

```
y <= NOT a AND b;
y <= NOT (a AND b);
```

Arithmetic operator:

There are predefined arithmetic operators in VHDL.

Symbol	Operator
+	Addition
-	Subtraction
*	Multiplication
/	Division
rem	Remainder
mod	Modulus
**	Exponentiation
abs	Absolute value

- **These operators are predefined for data types integers, real and time. To use these operations on std_logic or std_logic_vector, then the following library has to be included in VHDL code.**

```
Use ieee.std_logic_unsigned.all;
```

Comparison Operators

Used for making comparisons.

=	test for equality, result is Boolean
/=	test for inequality, result is Boolean
<	test for less than, result is Boolean
<=	test for less than or equal, result is Boolean
>	test for greater than, result is Boolean
>=	test for greater than or equal, result is Boolean

Shift Operators

Used for shifting data.

sll	Shift left logical
srl	Shift right logical
sla	Shift left arithmetic
sra	Shift right arithmetic
rol	Rotate left logical
ror	Rotate right logical

Example

variable Zm5 : BIT_VECTOR(3 **downto** 0) := ('1','0','1','1');

Zm5 **sll** 1 -- ('0', '1', '1', '0')

Zm5 **sll** 3 -- ('1', '0', '0', '0')

Zm5 **srl** 1 -- ('0', '1', '0', '1')

Zm5 **srl** 3 -- ('0', '0', '0', '1')

Zm5 **sla** 1 -- ('0', '1', '1', '1')

Zm5 **sla** 3 -- ('1', '1', '1', '1')

Zm5 **sra** 1 -- ('1', '1', '0', '1')

Zm5 **sra** 3 -- ('1', '1', '1', '1')

Zm5 **rol** 1 -- ('0', '1', '1', '1')

Zm5 **rol** 3 -- ('1', '1', '0', '1')

Zm5 **ror** 1 -- ('1', '1', '0', '1')

Zm5 **ror** 3 -- ('0', '1', '1', '1')

Concatenation

Ampersand (&) symbol is used for concatenation.

```
Signal a: std_logic_vector (0 to 2);
Signal b: std_logic_vector (0 to 4);
Signal c: std_logic_vector(0 to 7);
Begin
c<= a &b;
end;
```

- **It is not permissible to use concatenation on the left of the signal assignment symbol.**

```
Signal a: std_logic_vector (0 to 2);
Signal b: std_logic_vector (0 to 3);
Begin
‘0’ & a <= b;
End;
```