VHDL - Operators and Attributes

©Hanan Ribo

VHDL Operators

- In order to write any code efficiently it’s important to be familiar with the predefined Operators and Attributes.
- Pre-defined operators:
  - ✓ Assignment operators
  - ✓ Logical operators
  - ✓ Arithmetic operators
  - ✓ Relational (comparison) operators
  - ✓ Shift operators
  - ✓ Concatenation operators

Assignment Operators

- Operator **<=** Used to assign a value to a **SIGNAL**
- Operator **:=** Used to assign a value to a **VARIABLE**, **CONSTANT**, or **GENERIC** and also for establishing **initial values**.
- Operator **=>** Used to assign values to individual vector elements or with OTHERS.
- Examples:

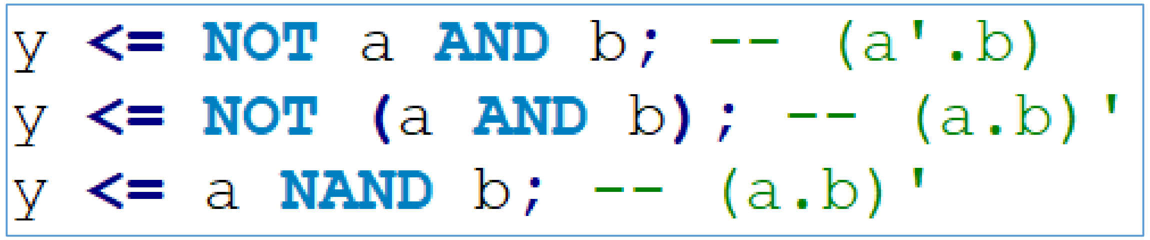
```
SIGNAL x : STD_LOGIC;
SIGNAL w: STD_LOGIC_VECTOR(0 TO 7); -- Rightmost bit is MSB
VARIABLE y : STD_LOGIC_VECTOR(3 DOWNTO 0); -- Leftmost bit is MSB

x <= '1'; -- '1' is assigned to SIGNAL x using "<="
w <= "10000000"; -- LSB is '1', the others are '0'
w <= (0 =>'1', OTHERS =>'0'); -- LSB is '1', the others are '0'
y := "0000"; -- "0000" is assigned to VARIABLE y using ":="
```

Logical Operators

- Used to perform logical operations.
- The data must be of type **BIT**, **STD_LOGIC**, or **STD_ULOGIC** (or their extensions, **BIT_VECTOR**, **STD_LOGIC_VECTOR**, or **STD_ULOGIC_VECTOR**).
- The logical operators are:

  NOT, AND, OR, NAND, NOR, XOR, XNOR
- Example:



```
y <= NOT a AND b; -- (a'.b)
y <= NOT (a AND b); -- (a.b)'
y <= a NAND b; -- (a.b)'
```

Arithmetic Operators

- Used to perform arithmetic operations.
- The data can be of type **INTEGER**, **SIGNED**, **UNSIGNED**, or REAL (for simulation only).
- If the *std_logic_signed* or the *std_logic_unsigned* package of the ieee library is used, then **STD_LOGIC_VECTOR** can also be employed directly in addition and subtraction operations

Arithmetic Operators	**synthesis restriction**
+ Addition, **–** Subtraction , ***** Multiplication	no synthesis restrictions
/ Division	only power of two dividers (shift operation) are allowed
****** Exponentiation	only static values of base and exponent are accepted
MOD Modulus, **REM** Remainder, **ABS** Absolute value	no synthesis support

Relational (comparison) operators

- Used for making comparisons.
- The data can be **any of the types listed before**.

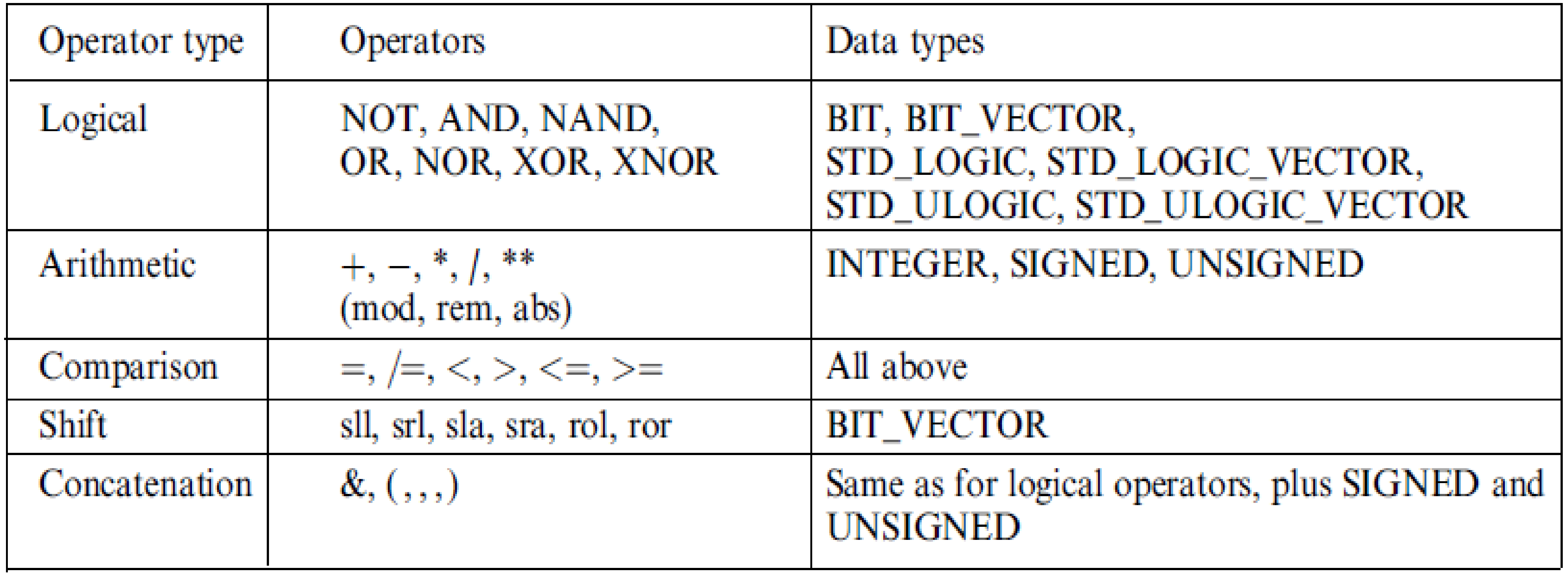
Comparison operation	**Explanation**
=	Equal to
/=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Shift Operators

- Used for shifting data left or right.
- Syntax:
  - ✓ <left operand> <shift operation> <right operand>
  - ✓ the left operand must be of type **BIT_VECTOR**
  - ✓ the right operand must be an **INTEGER**
- The shift operators are:

shift operation	**Explanation**
sll	Shift left logic = positions on the right are filled with ‘0’s
srl	Shift right logic = positions on the left are filled with ‘0’s

Operator summary table



Operator type	Operators	Data types
Logical	NOT, AND, NAND, OR, NOR, XOR, XNOR	BIT, BIT_VECTOR, STD_LOGIC, STD_LOGIC_VECTOR, STD_ULOGIC, STD_ULOGIC_VECTOR
Arithmetic	+, −, *, /, ** (mod, rem, abs)	INTEGER, SIGNED, UNSIGNED
Comparison	=, /=, <, >, <=, >=	All above
Shift	sll, srl, sla, sra, rol, ror	BIT_VECTOR
Concatenation	&, (, , ,)	Same as for logical operators, plus SIGNED and UNSIGNED

Attributes

- **Definition**: A value, function, type, range, signal, or constant that may be associated with one or more types, objects, subprograms, etc.
- **Description:** An attribute gives extra information about a specific part of a VHDL description *(pre-defined)*. Additionally, users can define new attributes *(user-defined)*.
- Improving code maintenance and generality.
- Each type or subtype T has a basic attribute called T'Base, which indicates the base type for type T. It should be noted that this attribute could be used only as a prefix for other attributes.
- **Syntax**: object_name ‘ attribute_name[(expression)];

Data Attributes (pre-defined)

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

- **d’LOW** - Returns lower array index
- **d’HIGH** - Returns upper array index
- **d’LEFT** - Returns leftmost array index
- **d’RIGHT** - Returns rightmost array index
- **d’LENGTH** - Returns vector size
- **d’RANGE** - Returns vector range
- **d’REVERSE_RANGE** - Returns vector range in reverse order

Example:

```
SIGNAL d : STD_LOGIC_VECTOR (7 DOWNTO 0);
```

So the equivalents are:

```
d'LOW=0, d'HIGH=7, d'LEFT=7, d'RIGHT=0, d'LENGTH=8,

d'RANGE=(7 downto 0), d'REVERSE_RANGE=(0 to 7).
```

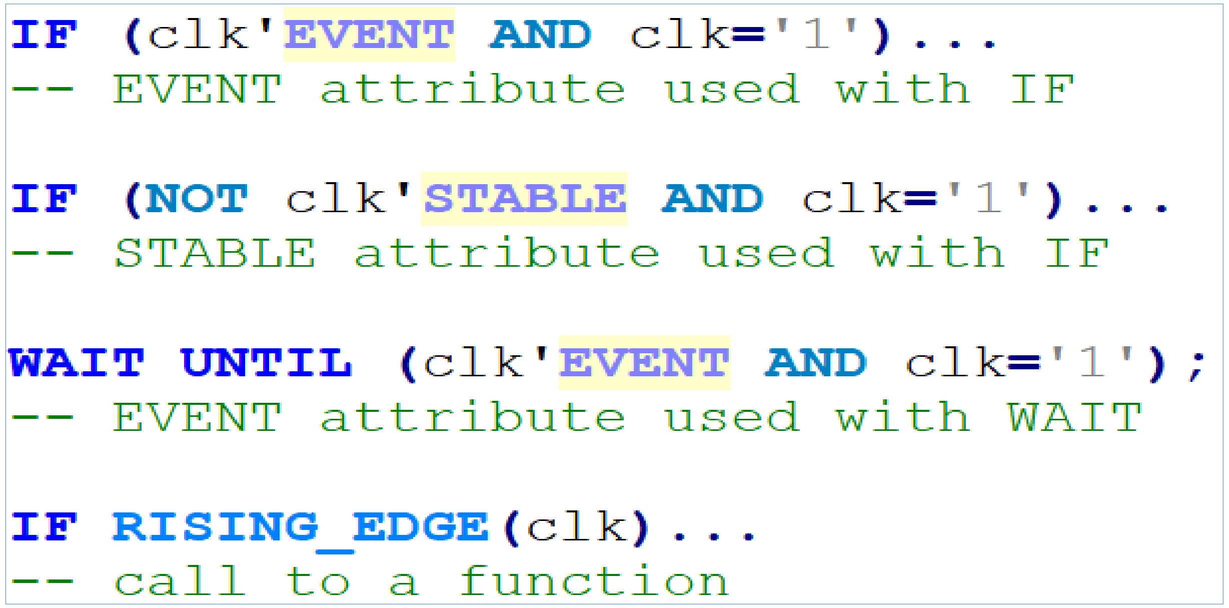
SIGNAL Attributes (pre-defined)

The pre-defined, **synthesizable** Signal attributes are the following:

- **s'EVENT** - Returns true when an event occurs on s
- **s'STABLE** - Returns true if no event has occurred on s

Example: `SIGNAL clk: STD_LOGIC;`

All the next four lines are equal and synthesizable. The condition return TRUE **on rising clk event.**



```
IF (clk'EVENT AND clk='1')...
-- EVENT attribute used with IF

IF (NOT clk'STABLE AND clk='1')...
-- STABLE attribute used with IF

WAIT UNTIL (clk'EVENT AND clk='1');
-- EVENT attribute used with WAIT

IF RISING_EDGE(clk)...
-- call to a function
```

Simulation purpose Attributes

For simulation only, there are many more attributes.

- *Scalar type attributes*
- *Attributes of discrete or physical types and subtypes*
- *Attributes of the array type or objects of the array type*
- *Signals attributes*
- *Attributes of named entities*

For details see the next link:

Extended predefined attributes

User-Defined Attributes

- VHDL also allows the construction of user-defined attributes.
- A user-defined attribute can be declared anywhere, except in a **PACKAGE BODY** (not recognized by the synthesis tool).
- Attribute declaration:

```
ATTRIBUTE attribute_name: attribute_type;
```

- Attribute specification:

```
ATTRIBUTE attribute_name OF target_name: class IS value;
```

  - ✓ **attribute_type** is any data type (**BIT, INTEGER, STD_LOGIC_VECTOR**, etc.)
  - ✓ **class** is any of **TYPE, SIGNAL, FUNCTION**, etc.
  - ✓ **value** is any of ‘0’, 27, ‘‘00 11 10 01’’, etc.

- Example:

```
ATTRIBUTE number_of_inputs: INTEGER; -- declaration
ATTRIBUTE number_of_inputs OF nand3: SIGNAL IS 3; -- specification

inputs <= nand3'number_of_inputs; -- attribute call, returns 3
```

Operator Overloading (user-defined operators)

- The arithmetic operations are specified between data of certain types.
- For instance, the pre-defined "+" operator does not allow addition between data of type BIT. We can use "+" to indicate a new kind of addition, between values of type **BIT_VECTOR**. This technique is called operator overloading.
- Example:

```
FUNCTION "+" (a: INTEGER, b: BIT) RETURN INTEGER IS
BEGIN
    IF (b='1') THEN RETURN a+1;
    ELSE RETURN a;
    END IF;
END "+";
SIGNAL inp1, outp: INTEGER RANGE 0 TO 15;
SIGNAL inp2: BIT;

  outp <= 3 + inp1 + inp2;
```