Operators and Attributes
Logical, arithmetic, relational, etc

Operators and attributes are very important to develop a VHDL code, which allows us to work with data.

VHDL
VHSIC HARDWARE
DESCRIPTION LANGUAGE

Operators and Attributes

Operators

- This is a very simple, straightforward, but **necessary** chapter. It defines the set of tools that you must work with data. We will start with operators.
- Operators are the key for managing data and making all the logic within or code.

- VHDL provide the following pre-defined operators:
  1. **Assignment** operators
  2. **Logical** Operators
  3. **Arithmetic** Operators
  4. **Relational** Operators
  5. **Shift** Operators
  6. **Concatenation** Operators

Operators

- **Assignment operators** are used to assign values to signals, variables and constants

<= 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.

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

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

Operators and Attributes

Operators

- **Logical operators** are used to perform logical operations. Data must be **BIT** or **STD_LOGIC** or **STD_ULOGIC** (and its vector forms), these are:

  - NOT (it has precedence over the others)
  - OR
  - NOR
  - XOR
  - XNOR
  - AND
  - NAND

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

Operators and Attributes

Operators

- **Arithmetic Operators** are used to perform arithmetic operations.

Data must be **INTEGER, UNSIGNED or REAL** (this is not synthesized directly).

If *std_logic_signed* or *std_logic_unsigned* is used, STD_LOGIC_VECTOR can be also used directly for addition and substraction.

+	Addition	
-	Subtraction	
*	Multiplication	
/	Division	-> Only power of two dividers are allows (shift operation).
**	Exponentiation	-> Only static values of base and exponent are allowed.
MOD	Modulus	-> a MOD b returns the remainder of a/b with the sign of b
REM	Remainder	-> a REM b returns the remainder of a/b with the sign of a
ABS	Absolute value.	-> Return the absolute value

Note that MOD , REM and ABS are usually not synthesized.

Operators

- **Comparison Operators** are used to perform comparisons. Data can be **INTEGER, UNSIGNED or REAL** (this is not synthesized directly), **STD_LOGIC** and its vector versions.
  - = Equal to
  - /= Not Equal to
  - < Less than
  - > Greater than
  - <= Less than or equal to
  - >= Greater than or equal to

- **Shift Operators** uses the syntax: <left operand><shift operation><right operand>.

Left operand must be BIT_VECTOR
Right operand must be INTEGER (+ or – in front of it can be used)

- sll Shift Left Logic -> positions on the right are filled with 0s
- srl Shift Right Logic -> positions on the left are filled with 0s

Examples:
"1100" sll 1 yields "1000"
"1100" srl 2 yields "0011"

Operators and Attributes

Signal Attributes

- **Signal Attributes** helps us to check the state of signals. Suppose a signal **x**, then (full list):
  - x'EVENT — Returns true when an event (a change) occurs in x
  - x'STABLE — Returns true when no event occurs in x
  - x'ACTIVE — Returns true if x='1'
  - x'QUIET<time> — Returns true if no event has occurred during "time"
  - x'LAST_EVENT — Return the time since the last event
  - x'LAST_ACTIVE — Return the time since the last event x='1'
  - x'LAST_VALUE — Return the value of x before the event.
  - Others see full list here

```
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
```

All these conditions are true when a rising clk occurs. Note that the latter instructions call a function. (we see that later)

Operators and Attributes

USER Defined Attributes

- **Signal Attributes** can be also defined by the user. This can be declared anywhere, except in a PACKAGE BODY (here it will be omitted by the synthesis). Attributes of an entity, architecture, configuration or package must be specified inside that region. The syntax is:

ATTRIBUTE attribute_name: attribute_type;	Attribute Declaration
ATTRIBUTE attribute_name OF target_name: class IS value;	Attribute Specification

attribute_type: any data type (BIT, INTEGER, STD_LOGIC_VECTOR, etc.)

class: TYPE, SIGNAL, FUNCTION, etc.

value: '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
```

Operators and Attributes

USER Defined Operations

- Like attributes, operations can be also user defined. To define a new operation, only operators symbols or combination of those can be used. See IEEE1076Std (you must be connected to UC network to be able to download this file) to check all available operator symbols. Creating a new operation, based on available symbols is known as **"operator overloading".**
- Suppose you want to use the symbol **+** to add an integer to a binary 1-bit number.

```
----------------------------------------
FUNCTION "+" (a: INTEGER, b: BIT) RETURN INTEGER IS
BEGIN
   IF (b='1') THEN RETURN a+1;
   ELSE RETURN a;
   END IF;
END "+";
----------------------------------------
```

Calling the function

```
------------------------------
SIGNAL inp1, outp: INTEGER RANGE 0 TO 15;
SIGNAL inp2: BIT;
   (...)
outp <= 3 + inp1 + inp2;
   (...)
------------------------------
```

Here the first "+" adds two integers, using pre-defined addition operator. The second "+" adds an integer with a BIT using our user-defined operator.

Operators and Attributes

GENERIC

- This is a form to define generic parameters that can used several times and changed easily. The purpose is to design a more portable and reusable code. The syntax is:
  - GENERIC (parameter_name : parameter_type := parameter_value);

```
ENTITY my_entity IS
   GENERIC (n : INTEGER := 8);
   PORT (...);
END my_entity;
ARCHITECTURE my_architecture OF my_entity IS
   ...
END my_architecture:
```

"n" has been defined as an INTEGER with value equal to 8. The parameter n Will be replaced by 8 within the entity.

Operators and Attributes

Summary

Operators.

Operator type	Operators	Data types
Assignment	<=, :=, =>	Any
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.

Application	Attributes	Return value
For regular DATA	d'LOW	Lower array index
	d'HIGH	Upper array index
	d'LEFT	Leftmost array index
	d'RIGHT	Rightmost array index
	d'LENGTH	Vector size
	d'RANGE	Vector range
	d'REVERSE_RANGE	Reverse vector range
For enumerated DATA	d'VAL(pos)◆	Value in the position specified
	d'POS(value)◆	Position of the value specified
	d'LEFTOF(value)◆	Value in the position to the left of the value specified
	d'VAL(row, column)◆	Value in the position specified
For a SIGNAL	s'EVENT	True when an event occurs on s
	s'STABLE	True if no event has occurred on s
	s'ACTIVE◆	True if s is high

The black diamond indicates that the construct is not synthesizable (or it has little support).

Remember that a **non-synthesizable statement is one that can not create hardware**. It is useful for simulation, but not after synthesis.

PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

END-Operators and Attributes
Logical, arithmetic, relational, etc

Operators and attributes are very important to develop a VHDL code, which allows us to work with data.

VHDL
VHSIC HARDWARE
DESCRIPTION LANGUAGE