Chapter 5
Operators and Attributes

VHDL

Outline

- Logical Operators
- Relational Operators
- Shift Operators
- Adding Operators
- Multiplying Operators
- Miscellaneous Operators
- Attributes
- Numeric Standard

Logical Operators

- **The seven logical operator are:**

  AND, OR, NAND, NOR , XOR, XNOR, NOT

- **Logical operators used in boolean expression.**
- **For types: bit, boolean, std_logic, std_ulogic, and their 1D arrays.**
- **Bit value '0' and '1' are treated as FALSE and TRUE respectively.**
- **NOT has higher precedence; others have equal, lower precedence.**
- **Parentheses usually required for multilevel equations.**
  - Examples:

```
z <= a AND b AND c OR d NAND e OR NOT f;
-- Equivalent:
z <= ((((a AND b) AND c) OR d) NAND e) OR (NOT f);
-- Not equivalent, but usual algebraic meaning:
z <= (a AND b AND c) OR (d NAND e) OR (NOT f);
```

- **The result of logical operation has the same type as its operand.**

Relational Operators

- **The six relational operator are:**

=	/=	<	<=	>	>=

- **The result type for all relational operators is always boolean.**
- **The = and /= operators are predefined on any type (except file).**
- **The remaining four relational operators are predefined on any scalar type (e.g., integer, enumerated, real) or discrete array type (i.e., arrays in which element values belong to a discrete type).**
- **When operands are discrete array types, comparison is performed one element at a time from left to right.**
  - Examples:

```
"011"  < "101"       -- true
"VHDL" < "VHDL92"    -- true no char is null
```


Shift Operators

- **The six shift operator are (1076-1993 only):**

sll	srl	sla	sra	rol	ror

- **Each of the operators takes an array of bit or boolean as the left operand and an integer value as the right operand and performs the specified operation.**
- **If the integer value is a negative number, the opposite action is performed, that is, a left shift or rotate becomes a right shift or rotate, respectively, and vice versa.**
- **The sll operator (shift left logical) and the srl operator (shift right logical) fill the vacated bits with *left-operand-type*'left.**
- **The sla operator (shift left arithmetic) fills the vacated bits with the rightmost bit of the left operand, while the sra operator (shift right arithmetic) fills the vacated bits with the leftmost bit of the left operand.**

Shift Operators (continue)

- **The rotate operators cause the vacated bits to be filled with the displaced bits in a circular fashion.**
- **When operands are discrete array types, comparison is performed one element at a time from left to right.**
  - Examples:

```
"1001010" sll 2  ⇒  "0101000"   -- filled with '0'
"1001010" srl 3  ⇒  "0001001"   -- filled with '0'
"1001010" sla 2  ⇒  "0101000"   -- filled with rmb
"1001010" sra 3  ⇒  "1111001"   -- filled with lmb
"1001010" rol 2  ⇒  "0101010"   -- rotate left
"1001010" ror 3  ⇒  "0101001"   -- rotate right
"1001010" sla -2 ⇒  "1110010"   -- sra 2
"1001010" rol -1 ⇒  "0100101"   -- ror 1
```

Arithmetic Operators

- **Standard Arithmetic Operators:**

Add	+	
Subtract	-	
Multiply	*	
Divide	/	
Mod	**MOD**	integers only
Remainder	**REM**	integers only
Exponentiation	**	right operand integer-only
Absolute Value	**ABS**	
Concatenate	**&**	
Sign	+ or -	

- **Most of standard operators only useful for simulation, not synthesis.**
- **Most operators are overloaded by packages:**
  - **numeric_std** (1076.3) for signed, unsigned, and integer types
  - **std_arith** (Warp) for std_logic_vector types (as unsigned values)

Adding Operators

- **The adding operator are:**

  + (add) **-** (sub) **&** (concat)

- **The operands for the + (addition) and - (subtraction) operators must be of the same numeric type, with the result being of the same type.**
- **The + and - operators may also be used as unary operators, in which case the operand and the result type are the same.**
- **The operands for the & (concatenation) operator can be either a one-dimensional array type or an element type. The result is always an array type.**
  - Examples:

```
'0' & '1'          ⇒  "01"     --
'C' & 'A' & 'T'    ⇒  "CAT"    --
"VH" & "DL"        ⇒  "VHDL"   --
 X <= "01"; Y <= "11"; Z <= '0';
 X & Y & Z         ⇒  "01110"
```


Multiplying Operators

- **The multiplying operator are:**

  * (mul) / (div) **MOD** (modulus) **REM** (remainder)

- **The * and / operators are predefined for both operands being of the same integer or real type. The result is also of the same type.**
- **The multiplication operator is also defined for the case when one of the operands is of physical type and the second operand is of integer or real type. The result is of physical type.**
- **For the division operator, division of a value of physical type by either an integer or a real value is allowed, and the result type is the physical type. Division of a value of physical type by another object of the same physical type is also defined, and it yields an integer value as a result.**
- **The REM and MOD operators operate on operands of integer types, and the result is also of the same type.**

Multiplying Operators (continue)

- **The result of a REM operation has the sign of its first operand and is defined as: A REM B ≡ A - (A / B) * B (as % at C)**
- **The result of a MOD operation has the sign of its second operand and is defined as: A MOD B ≡ A - B * N**
- Examples:

```
  7  MOD 4     ⇒   3    --
  7  REM 4     ⇒   3    --
(-7) MOD 4     ⇒   1    --
(-7) REM 4     ⇒  -3    --
  7  MOD (-4)  ⇒  -1    --
(-7) REM (-4)  ⇒  -3    --
```

- **Synthesis tools vary in their support for multiplying operators.**

Miscellaneous Operators

- **The Miscellaneous operator are:**

ABS (absolute)	****** (exponentiation)

- **The ABS (absolute) operator is defined for any numeric type.**
- **The ** (exponentiation) operator is defined for the left operand to be of integer or real type, and for the right operand (i.e., the exponent) to be of integer type only.**

Attributes

- **Provide information or characteristic about an item: a signal, a variable, a type, a function, etc.**
- **Format: *ItemName'attribute***
- **Used like a constant (read-only value) in places like conditional expressions and the right-hand side of assignments.**
- **Some useful predefined attributes:**

Attribute	Meaning
t'left	leftmost value of a type
t'right	rightmost value of a type
t'low	lowest, smallest value of a type
t'high	highest, greatest value of a type
a'length	length, number of elements, of an array
a'range	range of an array (x TO y, x DOWNTO y)
s'event	TRUE if a transition just occurred on a signal

Attributes

TYPE bitcount IS integer RANGE 5 DOWNTO -3;
TYPE opcode IS (Add, Sub, Jump, Call, Nop);
TYPE byte IS ARRAY (7 DOWNTO 0) OF std_logic;
SIGNAL clk: std_logic;

	bitcount	opcode	byte	clk
'left				
'right				
'low				
'high				
'length				
'range				
'event				

Attributes

TYPE bitcount IS integer RANGE 5 DOWNTO -3;
TYPE opcode IS (Add, Sub, Jump, Call, Nop);
TYPE byte IS ARRAY (7 DOWNTO 0) OF std_logic;
SIGNAL clk: std_logic;

	bitcount	opcode	byte	clk
'left	**5**	**Add**	**7**	
'right	**-3**	**Nop**	**0**	
'low	**-3**	**Add**	**0**	
'high	+5	**Nop**	**7**	
'length			**8**	
'range			◄	**7 DOWNTO 0**
'event				◄ **TRUE or FALSE**

Numeric Standard Data Types

- **IEEE 1076.3** synthesis standard defines two packages, **numeric_std** and **numeric_bit**, which
  - Define new types **signed** and **unsigned** for binary integers.
  - Overload operators for these types - **arithmetic**, **relational** , **logical**.
  - Define new functions for these types - **type conversions**.
  - Are incompatible: choose one, not both.
  - All require **library** / **use** statements before entity declaration:

```
LIBRARY ieee;
USE ieee.numeric_std.all;
```

```
--numeric_std
TYPE unsigned IS ARRAY (natural RANGE <>) of std_logic;
TYPE signed IS ARRAY (natural RANGE <>) of std_logic;

--numeric_bit
TYPE unsigned IS ARRAY (natural RANGE <>) of bit;
TYPE signed IS ARRAY (natural RANGE <>) of bit;
```

Numeric Types Conversion Functions

- The numeric type conversion functions are used to convert between **integer** data type and **signed** and **unsigned** data types.

```
FUNCTION To_Integer(arg: unsigned) RETURN natural;
FUNCTION To_Integer(arg: signed) RETURN integer;
FUNCTION To_Unsigned(arg: natural, size: natural) RETURN
 unsigned;
FUNCTION To_Signed(arg: integer, size: natural) RETURN
 signed;

FUNCTION Resize(arg: signed, new_size: natural) RETURN
 signed;
FUNCTION Resize(arg: unsigned, new_size: natural) RETURN
 unsigned;
```


Std_Arith Package

- Operators may be overloaded for operation on a *different* types.
- **Numeric_std** package (**IEEE 1076.3**) allows
  - comparing **signed** to **integer**, and **unsigned** to **natural**
- **Std_arith** package (**IEEE or Warp**) allows
  - Operate **std_logic_vector** (as unsigned) with **integer.**
  - Need:

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
-- or USE work.std_arith.ALL;
```