

ECE-238 Logic Design – Lab

Syntax (Data type and operation)

* Words written using lower-case letters and possibly one or many hyphens, are used to denote a syntactical category, for example: entity-declaration. Reserved words are written with bold characters, for example: **entity**, **constant**.

* Every replacement rule contains a left hand side expression and a right hand side expression separated by the sign \rightarrow , which means "looks as" or "may be replaced with".

* |, a vertical line (the *pipe* sign) is used to separate many mutually exclusive alternatives. [], square brackets surround optional things that may occur once or not at all. {}, braces surround optional things that may occur once, many times or not at all. (), parenthesis are used to clarify how and in which order a rule is evaluated.

Reserved words and characters surrounded by apostrophes, ' ', are included "as is" in the source code.

Type	Possible values	Operation(by priority)	Example
INTEGER	At least -2147483647~2147483647	ABS ** * / MOD REM + - (sign) + - = /= < <= > >=	architecture sample1 of consts is signal A, B, Z : integer range 0 to 3; constant A1: real :=0.1231; constant PERIOD : time := 50 ns; signal A2 : bit := '1';
REAL	At least: -1.0E38 ~ 1.0E38	ABS ** * / + - (sign) + - = /= < <= > >=	signal A3, B3, Z3: bit_vector (1 downto 0); constant A4: Boolean :=FALSE; constant A5: charater :='c'; constant A6: string := "This is a string"; begin
TIME	At least: -2147483647 ~2147483647 (fs, ps, ns, us, ms, sec, min, hr)	ABS ** * / + - (sign) + - = /= < <= > >=	A <= 2; A2 <= not A2 after PERIOD *2; A3 <= "10"; B3<=" 11" process(Z3,A3,B3)
BIT	'0','1'	NOT = /= < <= > >= AND NAND OR NOR XOR XNOR	begin if Z3 = '1' then Z 3<= A3; else Z3 <= B3; end if; end process;
BIT_VECT OR	Unconstrained array of BIT	NOT & SLL* SRL* SLA* SRA* ROL* ROR* = /= < <= > >= AND NAND OR NOR XOR XNOR*	end smaple1;
BOOLEAN	FALSE, TRUE	NOT = /= < <= > >= AND NAND OR NOR XOR XNOR*	
CHARACT ER	128 characters in VHDL'87 [ISO 646-1983] 256 characters in VHDL'93 [ISO 8859-1 : 1987(E)]		
STRING	Unconstrained array of CHARACTER		

Logical operators

The logical operators and, or, nand, nor, xor and xnor are used to describe Boolean logic operations, or perform bit-wise operations, on bits or arrays of bits.

<i>Operator</i>	<i>Description</i>	<i>Operand Types</i>	<i>Result Types</i>
and	And	Any Bit or Boolean type	Same Type
or	Or	Any Bit or Boolean type	Same Type
nand	Not And	Any Bit or Boolean type	Same Type
nor	Not Or	Any Bit or Boolean type	Same Type
xor	Exclusive OR	Any Bit or Boolean type	Same Type
xnor	Exclusive NOR	Any Bit or Boolean type	Same Type

Relational operators

Relational operators are used to test the relative values of two scalar types. The result of a relational operation is always a Boolean true or false value.

<i>Operator</i>	<i>Description</i>	<i>Operand Types</i>	<i>Result Type</i>
=	Equality	Any type	Boolean
/=	Inequality	Any type	Boolean
<	Less than	Any scalar type or discrete array	Boolean
<=	Less than or equal	Any scalar type or discrete array	Boolean
>	Greater than	Any scalar type or discrete array	Boolean
>=	Greater than or equal	Any scalar type or discrete array	Boolean

Adding operators

The adding operators can be used to describe arithmetic functions or, in the case of array types, concatenation operations.

<i>Operator</i>	<i>Description</i>	<i>Operand Types</i>	<i>Result Type</i>
+	Addition	Any numeric type	Same type
-	Subtraction	Any numeric type	Same type
&	Concatenation	Any numeric type	Same type
&	Concatenation	Any array or element type	Same array type

Multiplying operators

The multiplying operators can be used to describe mathematical functions on numeric types.

Note: Synthesis tools vary in their support for multiplying operators.

<i>Operator</i>	<i>Description</i>	<i>Operand Types</i>	<i>Result Type</i>
*	Multiplication	Left: any integer or floating point type. Right: same type	Same as left
*	Multiplication	Left: any physical type. Right: integer or real type.	Same as left
*	Multiplication	Left: integer or real type. Right: any physical type.	Same as right

/	Division	Left: any integer or floating point type. Right: same type	Same as left
/	Division	Left: any integer or floating point type. Right: same type	Same as left
/	Division	Left: integer or real type. Right: any physical type.	Same as right
mod	Modulus	Any integer type	Same type
rem	Remainder	Any integer type	Same type

Sign operators

Sign operators can be used to specify the sign (either positive or negative) of a numeric object or literal.

<i>Operator</i>	<i>Description</i>	<i>Operand Types</i>	<i>Result Type</i>
+	Identity	Any numeric type	Same type
-	Negation	Any numeric type	Same type

Miscellaneous operators

The exponentiation and absolute value operators can be applied to numeric types, in which case they result in the same numeric type. The logical negation operator results in the same type (bit or Boolean), but with the reverse logical polarity. The shift operators provide bit-wise shift and rotate operators for arrays of type bit or Boolean.

<i>Operator</i>	<i>Description</i>	<i>Operand Types</i>	<i>Result Type</i>
**	Exponentiation	Left: any integer type Right: integer type	Same as left
**	Exponentiation	Left: any floating point type Right: integer type	Same as left
abs	Absolute value	Any numeric type	Same as left type
not	Logical negation	Any Bit or Boolean type	Same as left type
sll	Shift left logical	Left: Any one-dimensional array of Bit or Boolean Right: integer type	Same as left type
srl	Shift right logical	Left: Any one-dimensional array of Bit or Boolean Right: integer type	Same as left type
sla	Shift left arithmetic	Left: Any one-dimensional array of Bit or Boolean Right: integer type	Same as left type
sra	Shift right arithmetic	Left: Any one-dimensional array of Bit or Boolean Right: integer type	Same as left type
rol	Rotate left	Left: Any one-dimensional array of Bit or Boolean Right: integer type	Same as left type
ror	Rotate right	Left: Any one-dimensional array of Bit or Boolean Right: integer type	Same as left type