



UNIVERSITÀ
DEGLI STUDI DI TRIESTE



Scalar data types and their operations

A.Carini – Progettazione di sistemi elettronici

VHDL objects

- VHDL objects are items that have a value of a specified type.
- There are four classes of objects:
 - *Constants*
 - *Variables*
 - *Signals*
 - *Files*
- Constants cannot change, while variables change with assignments.
- In VHDL, variables are used to implement memory elements.

Constants' declaration

```
constant_declaration ←  
    constant identifier { , ... } : subtype_indication [ := expression ] ;
```

- Example:


```
constant number_of_bytes : integer := 4;  
constant number_of_bits : integer := 8 * number_of_bytes;  
constant e : real := 2.718281828;  
constant prop_delay : time := 3 ns;  
constant size_limit, count_limit : integer := 255;
```

Variables' declaration

```
variable_declaration ←  
  variable identifier { , ... } : subtype_indication [ := expression ] ;
```

- Example:

```
variable index : integer := 0;  
variable sum, average, largest : real;  
variable start, finish : time := 0 ns;
```



```
variable start : time := 0 ns;  
variable finish : time := 0 ns;
```

Example of declarations

```
entity ent is
end entity ent;

architecture sample of ent is
    constant pi : real := 3.14159;
begin

    process is
        variable counter : integer;
    begin
        -- ...          -- statements using pi and counter
    end process;
end architecture sample;
```

Assignment Statements

```
variable_assignment_statement  $\Leftarrow$  [ label : ] name := expression ;
```

- Example:

```
program_counter := 0;  
index := index + 1;
```

Difference between variables and signals

- In variables, the assignment modifies immediately the value of the variable.
- In signals, the assignment plans a new value that will be applied in some future moment.
- For this reason different symbols are used:
 - := for variables.
 - <= for signals.

Scalar types

- The VHDL is strongly typed
 - Each object can assume only values of its declared type!
- Declaration of a new type:

```
type_declaration ← type identifier is type_definition ;
```

- Types with the same declaration but different names are distinct and incompatible.
- Example:

```
type apples is range 0 to 100;  
type oranges is range 0 to 100;
```

Example of use in entity ports

```
package int_types is
    type small_int is range 0 to 255;
end package int_types;
```

```
use work.int_types.all;
entity small_adder is
    port ( a, b : in small_int; s : out small_int );
end entity small_adder;
```

Integer types

- Assume only integer values.
- Predefined type : integer
- Contains at least the interval $(-2^{31}+1, 2^{31}-1)$
- Syntax for defining a new integer type:

```
integer_type_definition ←  
    range simple_expression ( to | downto ) simple_expression
```

- **TO** used for ascending ranges
- **DOWNTO** used for descending ranges

Examples of integer types

- Example:

```
type day_of_month is range 0 to 31;  
type year is range 0 to 2100;  
  
variable today : day_of_month := 9;  
variable start_year : year := 1987;
```

- Illegal assignment:

```
start_year := today;
```

Ranges and expressions

- The expressions that provide the range limits must be locally static
 - They can depend from constant but not from variables!

```
constant number_of_bits : integer := 32;  
type bit_index is range 0 to number_of_bits - 1;
```

Operations on integer types

+	addition, or identity
-	subtraction, or negation
*	multiplication
/	division
mod	modulo
rem	remainder
abs	absolute value
**	exponentiation

`maximum(3, 20) = 20` `minimum(3, 20) = 3`

REM

- It is the remainder of the division and it is defined such that:

$$A = (A / B) * B + (A \text{ rem } B)$$

- A **rem** B has the same sign of A and absolute value lower than that of B.
- Examples:

$$5 \text{ rem } 3 = 2, \quad (-5) \text{ rem } 3 = -2 \quad 5 \text{ rem } (-3) = 2, \quad (-5) \text{ rem } (-3) = -2$$

MOD

- It is the module operation and it is defined such that:

$$A = B * N + (A \bmod B) \quad \text{--- for some integer } N$$

- $A \bmod B$ has the same sign of B and absolute value lower than that of B.
- Examples:

$$5 \bmod 3 = 2, \quad (-5) \bmod 3 = 1, \quad 5 \bmod (-3) = -1, \quad (-5) \bmod (-3) = -2$$

Floating point types

- Used for approximating the real numbers with an mantissa-exponent representation.
- Predefined type: *real*
- Contains at least the interval $(-1.0E+38, +1.0E+38)$ with at least 6 precision digits.
- Syntax for defining a new floating point type:

```
floating_type_definition ←  
    range simple_expression ( to | downto ) simple_expression
```

```
type input_level is range -10.0 to +10.0;  
type probability is range 0.0 to 1.0;
```

Operations on floating point types

+	addition, or identity
-	subtraction, or negation
*	multiplication
/	division

abs	absolute value
**	exponentiation

- The exponent must be an integer.

Physical types

- Used for representing physical quantities (like the length, the current, the voltage, etc.)
- We will always have a primary unit and we could have also some secondary units.

```
physical_type_definition ←  
    range simple_expression ( to | downto ) simple_expression  
    units  
        identifier ;  
        { identifier = physical_literal ; }  
    end units [ identifier ]  
physical_literal ← [ decimal_literal | based_literal ] unit_name
```

Example: a resistance

```
type resistance is range 0 to 1E9
  units
    ohm;
end units resistance;
```

- Its values are written as:

```
5 ohm  22 ohm  471_000 ohm
```

```
ohm  1 ohm
```

Example: a resistance

- By introducing also some secondary units:

```
type resistance is range 0 to 1E9
  units
    ohm;
    kohm = 1000 ohm;
    Mohm = 1000 kohm;
end units resistance;
```

Example: a length

- Secondary units does not need to be a power of 10:

type length is range 0 to 1E9

units

um;

-- *primary unit: micron*

mm = 1000 um;

-- *metric units*

m = 1000 mm;

inch = 25400 um;

-- *English units*

foot = 12 inch;

end units length;

Operations on physical types

+	addition, or identity
-	subtraction, or negation
*	multiplication
/	division
abs	absolute value

- Note:

$$5 \text{ mm} * 6 = 30 \text{ mm}$$

$$18 \text{ kohm} / 2.0 = 9 \text{ kohm}, \quad 33 \text{ kohm} / 22 \text{ ohm} = 1500$$

Time

- Predefined physical type, used for delays

```
type time is range implementation defined  
units  
    fs;  
    ps = 1000 fs;  
    ns = 1000 ps;  
    us = 1000 ns;  
    ms = 1000 us;  
    sec = 1000 ms;  
    min = 60 sec;  
    hr = 60 min;  
end units;
```

Enumeration types

- Syntax:

```
enumeration_type_definition  $\leftarrow$  ( ( identifier  $\mid$  character_literal ) { , ... } )
```

- Examples:

```
type alu_function is (disable, pass, add, subtract, multiply, divide);
```

```
type octal_digit is ('0', '1', '2', '3', '4', '5', '6', '7');
```

```
variable alu_op : alu_function;
```

```
variable last_digit : octal_digit := '0';
```

```
alu_op := subtract;
```

```
last_digit := '7';
```

Overloading of identifiers

- In case of overloading of identifiers, the context determines the kind of identifier.

```
type logic_level is (unknown, low, undriven, high);  
variable control : logic_level;  
type water_level is (dangerously_low, low, ok);  
variable water_sensor : water_level;
```

```
control := low;  
water_sensor := low;
```

Predefined enumeration types

```
type severity_level is (note, warning, error, failure);  
type file_open_status is (open_ok, status_error, name_error, mode_error);  
type file_open_kind is (read_mode, write_mode, append_mode);
```

- **Character**
- **Boolean**
- **Bit**

Predefined type Boolean

```
type boolean is (false, true);
```

- Operators:
 - Relational
 - =, /=
 - <, >, <=, >=
 - Logical `and`, `or`, `nand`, `nor`, `xor`, `xnor` and `not`
- `and`, `or`, `nand`, `nor` are “short-circuit” operators
- They evaluate the right term only if the left term is insufficient to compute the result

```
(b /= 0) and (a/b > 1)
```

Predefined type Bit

`type bit is ('0', '1');`

- We can apply also the logical operators

`and, or, nand, nor, xor, xnor and not`

- '0' corresponds to false
- '1' corresponds to true
- But we can not mix bits and Booleans
 - It is illegal:

`'0' and true`

IEEE standard logic 1164

```
type std_ulogic is ( 'U', -- Uninitialized
                    'X',  -- Forcing Unknown
                    '0',  -- Forcing zero
                    '1',  -- Forcing one
                    'Z',  -- High Impedance
                    'W',  -- Weak Unknown
                    'L',  -- Weak zero
                    'H',  -- Weak one
                    '-' ); -- Don't care
```

```
library ieee; use ieee.std_logic_1164.all;
```

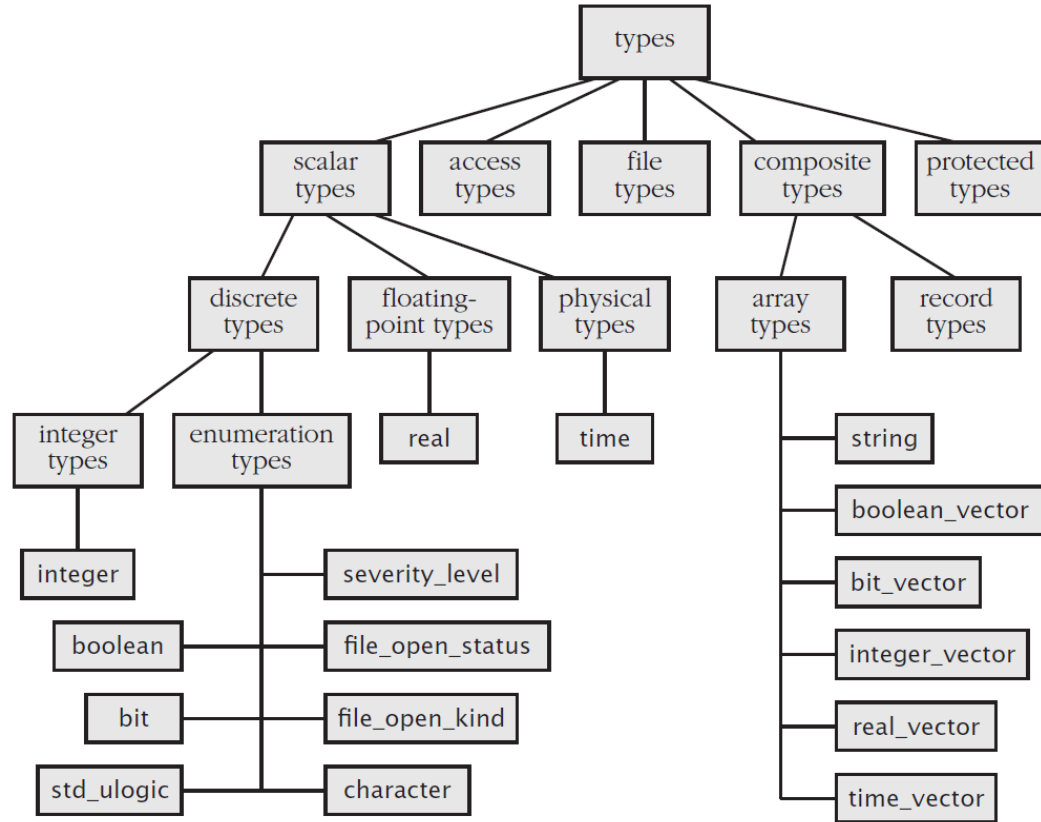
IEEE standard logic 1164

- We can apply the logic operators, which are optimistic, i.e., '0' and 'Z' = '0'
- From VHDL 2008 there are relational operators that neglect the force:

'1' ?= 'H' = '1' '1' ?/= 'H' = '0'
'0' ?= 'H' = '0' '0' ?/= 'H' = '1'

'Z' ?= 'H' = 'X' 'W' ?/= 'H' = 'X'
'0' ?= 'U' = 'U' '0' ?/= 'U' = 'U'
'1' ?= '-' = '1' '-' ?/= '-' = '0'

Type classification



Subtypes

```
subtype_declaration ← subtype identifier is subtype_indication ;  
subtype_indication ←  
    type_mark [ range simple_expression ( to | downto ) simple_expression ]
```

Example:

```
subtype small_int is integer range -128 to 127;
```

```
variable deviation : small_int;
```

```
variable adjustment : integer;
```

we can use them in calculations:

```
deviation := deviation + adjustment;
```

Predefined subtypes

```
subtype natural is integer range 0 to highest_integer;  
subtype positive is integer range 1 to highest_integer;  
  
subtype delay_length is time range 0 fs to highest_time;
```

Type qualification

`Type_name' (value)`

Example:

```
type logic_level is (unknown, low, undriven, high);  
type system_state is (unknown, ready, busy);
```

```
logic_level'(unknown)    system_state'(unknown)
```

Type conversion

Type_name (value)

Example:

```
real(123), integer(3.6)
```

Attributes of scalar types

Type_name' attribute

For all scalar types:

T'left	first (leftmost) value in T
T'right	last (rightmost) value in T
T'low	least value in T
T'high	greatest value in T
T'ascending	true if T is an ascending range, false otherwise
T'image(x)	a string representing the value of x
T'value(s)	the value in T that is represented by s

Example of attributes

```
type resistance is range 0 to 1E9
  units
    ohm;
    kohm = 1000 ohm;
    Mohm = 1000 kohm;
  end units resistance;

type set_index_range is range 21 downto 11;
type logic_level is (unknown, low, undriven, high);
```

Example of attributes

```
resistance'left = 0 ohm
resistance'right = 1E9 ohm
resistance'low = 0 ohm
resistance'high = 1E9 ohm
resistance'ascending = true
resistance'image(2 kohm) = "2000 ohm"
resistance'value("5 Mohm") = 5_000_000 ohm

set_index_range'left = 21
set_index_range'right = 11
set_index_range'low = 11
set_index_range'high = 21
set_index_range'ascending = false
set_index_range'image(14) = "14"
set_index_range'value("20") = 20

logic_level'left = unknown
logic_level'right = high
logic_level'low = unknown
logic_level'high = high
logic_level'ascending = true
```

Attributes for discrete types and physical types

T'pos(x)	position number of x in T
T'val(n)	value in T at position n
T'succ(x)	value in T at position one greater than that of x
T'pred(x)	value in T at position one less than that of x
T'leftof(x)	value in T at position one to the left of x
T'rightof(x)	value in T at position one to the right of x

Example:

```
logic_level'pos(unknown) = 0  
logic_level'val(3) = high  
logic_level'succ(unknown) = low  
logic_level'pred(undriven) = low
```

Example of area computation

```
type length is range integer'low to integer'high
  units
    mm;
  end units length;

type area is range integer'low to integer'high
  units
    square_mm;
  end units area;
```

```
variable L1, L2 : length;
variable A : area;
A := L1 * L2;    -- this is incorrect
A := area'val( length'pos(L1) * length'pos(L2) );
```

See:

- Peter Ashenden, «The designers' guide to VHDL» Morgan Kaufmann,
 - Chapter 2