- Objects
- Constants
- Variables
- Types and Type Declarations
- Numbers
- Physical Types
- Enumeration Types
- Subtypes
- Operators

Objects, Types, and Operations

Outline

- Objects
- Object Classes
- Class Types
- Operations on Types of Classes

Objects

- Object: Anything That Has a Name and Is of a Specified Type

- Four Classes of Objects
  - Constants
  - Variables
  - Signals (discussion deferred to later)
  - Files (discussion deferred to later)

Objects

- Classes of Objects Can Be of Different Types

Object Declaration

- Before an Object Can Be Used, It Must Be Declared

- Declarations
  - Specify a unique identifier
  - Define the type
  - May specify initial (default) value

Constants

- Constant initialized to a Value That Cannot Change
  - If not initialized, called a deferred constant
  - May only appear in package declaration
- Constant declaration insures that a Value has a Type

Constant Syntax

`constant` *identifier_list* `:` *subtype_indication* `[ :=` *expression* `] ;`

where

identifier_list **<=** *identifier* `{ , ... }`

Constant Declaration, e.g.,

```
constant PI : real := 3.1415926535897 ;
constant BUS_WIDTH : integer := 32 ;
constant
    INTENSITY_DYNAMIC_RANGE :
    real := 16 # FF . F ;
constant START_TIME_MINUTES :
    integer := 00 ;
```

Variables

- **Variable:** an Object Whose Value May be Changed After Creation

- Initialization Value is Optional.
- if not Initialized the Default for Scalar Types is:
  - The first in the list of an enumeration type
  - The lowest in an ascending range
  - The highest in a descending range

Variables Syntax

- Only Declare where it can be Accessed by One Process

variable *identifier_list* **:** *subtype_indication* [:= *expression*] ;

Variable Declaration, e.g.,

```
variable ControlValue : real := 3 . 68 ;

variable MinTemp, MaxTemp, MeanTemp : real
   := 0 . 0 ;
```

Variable Declaration, e.g.,

```
variable ImageWidth, ImageHeight :
         integer := 256 ;

variable DiskSize, MemUsed, MemLeft :
     integer ;

variable MBus : bit_vector
     ( 31 downto 0 ) ;
```

Variable Assignment Syntax

- Immediately Overwrites Variable with New Value
- *Unlike the way a Signal Does*

:= Replacement Operator for Variables

<= Replacement Operator for Signals

[*label* :] *identifier* := *expression* ;

Variable Assignment, e.g.,

```
MinTemp  :=  0 . 0 ;

ImageWidth := 128 ;

MainBus : = 16 # ffff_ffff ;

MainBus : = x " FFFF_FFFF " ;
```

Types

- The Type of a Data Object
  - Defines the set of values an object can take on
  - Defines operations which can be performed on object

- Scalar Type
  - Consists of a set of single, indivisible values

Types

- Composite Type

- Many Predefined Types

Type Syntax

- Type Qualification Is Used to Avoid Type Ambiguity in Overloaded Enumeration Literals

type_name **'** **(** *expression* **)**

  – Only states type of value

Type Syntax

- Type Conversion Can Be Used to Perform Mixed Arithmetic

```
New_Type ( Value_of_Old_Type )
```

- *e.g.*,

```
real ( 238 )
positive ( My_Integer_Value )
```

  - Rounds to nearest integer
  - Changes type of value

Type Declaration Syntax

```
type identifier is type_definition ;
```

type_definition **<=**

scalar_type_definition

| *composite_type_definition*

| *access_type_definition*

| *file_type_definition*

Type Declaration, *e.g.*

- **Identical Type Declarations Are Distinct**

```
type MidTermGrades is range 0 to 100 ;

type FinalGrades is range 0 to 100 ;
```

Scalar Type Declaration

- Scalar Type
  - Number types
  - Enumerated list
  - Physical quantities

Scalar Type Declaration Syntax

scalar_type_definition <=

enumeration_type_definition

| *integer_type_definition*

| *floating_type_definition*

| *physical_type_definition*

Predefined Integer Type

- Integer Type
  - A range of integer values within a specified range including the endpoints

- Integer Type Range
  - minimum range $(- 2^{31} + 1)$ to $(+ 2^{31} - 1)$

Operations on Integer Types

Highest precedence:	**	**abs**	**not**			
	*	/	**mod**	**rem**		
	+ (sign)	– (sign)				
	+	–	&			
	=	/=	<	<=	>	>=
Lowest precedence:	**and**	**or**	**nand**	**nor**	**xor**	

Table 7-1. Operators and precedence.

***Ashenden, VHDL cookbook**

Integer Type Definition Syntax

range *simple_expression* (**to** | **downto**)
simple_expression

to : left to right from smallest value to largest

downto : left to right from largest value to smallest

Integer Type Definition , e.g.,

```
type StreetNumbers is range 10107 to
  12568 ;

type ImagingSensors is range 0 to 5 ;

type Celsius is range 100 downto 0 ;

type PointSpread is range 14 downto 0 ;
```

Pre-defined Floating-Point Type Definition

- Floating-Point Type
  - A range of real values within a specified range including the endpoints
- Real
  - Minimum range (-1.0E+38) to (+1.0E+38)
  - 6-digits minimum precision
  - Corresponds to IEEE 32-bit representation
  - Floating-point type

Operations on Floating-Point Types

- Binary Operators

+	Add
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Operations on Floating-Point Types

- Unary Operators

-	Negation
+	Identity
abs	Absolute value

Floating-Point Type Syntax

range *simple_expression* (**to** | **downto**) *simple_expression*

to : left to right from smallest value to largest

downto : left to right from largest value to smallest

Floating-Point Type, *e.g.*,

```
type StreetPosition is range
     101 . 07 to 125 . 68 ;

type ImagingSensorSensitivity is range
     0 . 0 to 5 . 0 ;
```

Floating-Point Type, *e.g.*,

```
type Celsius is range 100.0 downto 0 . 0 ;

type PointSpread is range 15.0 downto 0 . 0 ;
```

Physical Type Definition

- *identifier* Is the Primary Unit With the Smallest Unit Represented

- *identifier-n* Secondary Units Defined in Terms of Primary Unit

Operations on Physical Types

- Binary Operators
  - * Multiplication by an integer or float
  - / Division by an integer or float
    - » Division by objects of same physical type yields an integer

Operations on Physical Types

- Unary Operators

-	negation
+	identity

Physical Type Definition Syntax

range *simple_expression* (**to** | **downto**)
simple_expression

units

identifier ;

{ *identifier-n* = *physical_literal* ; }

end units [*identifier*] ;

Operations on Physical Types

- Multiplication or Division of Different Physical Types Not Allowed

- If Required,
  - Convert to integers
  - Perform operation
  - Convert result to correct type

Predefined Physical Type, e.g.,

```
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 ; [ time ]
```

identifier

Identifier-n

Simulation Time Resolution Limit

- The Resolution Limit Determines the Precision to Which Time Values Are Represented.
- Values of Time Smaller Than the Resolution Limit Round Down to Zero.
- **fs** Is the *Normal Resolution Limit* During Model Simulation. **FEMTOSECOND**

Simulation Time Resolution Limit

- Larger Values of Time Can Be Used As a *Secondary Time Resolution Limit*
  - Units of all physical literals involving time must not be smaller than the secondary resolution limit

Physical Type Definition, e.g.,

```
type capacitance is range 0 to 1e12
  units
    picofarad ;
    nanofarad          = 1000 picofarad ;
    microfarad    = 1000 nanofarad ;
    farad         = 1e6 microfarad ;
  end units capacitance ;
```

Physical Type Resolution

- **`47 picofarad`**
- **`10.6 nanofarad`**
- **`4.7 picofarad`**
  - **rounds DOWN to 4 picofarads since pf is smallest unit**
  - **can only have integer value of base unit**

Enumeration Type Definition

- Enumeration Type
  - – An ordered set of identifiers or characters
  - – The identifiers and characters within a single enumeration type must be unique.
  - – Identifiers and characters may be reused in different enumeration types.

((*identifier* | *character_literal* **) { , ... })**

Enumeration Type, e.g.,

```
type Buffer_Direction is ( in , out , tri_state ) ;

type FF_Type is
      ( Toggle , Set_Reset , Data , JK ) ;
```

Enumeration Type, *e.g.,*

```
type MemoryType is ( Read_Only ,
                          Write_Only ,
                          RW ) ;


type GateType is ( AND , OR , INVERT ) ;
```

Predefined Enumeration Types

```
type severity_level is ( note , warning ,
                         error , failure ) ;
```

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

– Used to model abstract conditions

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

– Used to model hardware logic levels

Predefined Enumeration Types

```
type file_open_status is
 ( open_ok , status_error , name_error
  , mode_error ) ;

type character is ( NUL , SOH , ... ) ;
```

- All characters in ISO 8-bit character set

- IEEE std_logic_1164 Accounts for Electrical Properties

Subtypes

- Subtype
  - Values which may be taken on by an object and
  - are a subset of some base type, and,
  - may include all values.

Subtypes

- Subtypes Mixed in Expressions
  - Computations done in base type
  - Assignment fails if result is not within range of result variable (sub)type

Subtype Syntax

subtype *identifier* **is** ***subtype_indication*** **;**

subtype_indication <=

identifier **[range** ***simple_expression*** **(**
to | downto) ***simple_expression*** **]**

Subtype Cases

- A Subtype May Constrain Values From a Scalar Type to Be Within a Specified Range

```
subtype Pin_Count is integer range 0 to 400;

subtype Octal_Digits is character
     range ' 0 ' to ' 7 ' ;
```

Subtype Cases

- A Subtype May Constrain an Otherwise Unconstrained Array Type by Specifying Bounds for the Indices

```
subtype id is string ( 1 to 20 ) ;

subtype MyBus is bit_vector ( 8 downto 0 ) ;
```

Predefined Numeric 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 ;
```

Scalar Type Attributes

- Predefined Attributes Associated With Each Type

Type_Name ' *Attribute_Name*

All Scalar Type Attributes

T'left	leftmost value in T
T'right	rightmost value in T
T'low	least value in T
T'high	greatest value in T
T'ascending	True if ascending range, else false
T'image(x)	a string representing x
T'value(s)	the value in T that is represented by s

Discrete and Physical Scalar Type Attributes

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

Operators

- "Short-Circuit" Operators
  - Behavior with binary operators
    - Evaluate left operand
    - If value of operand determines the *value of expression*, set result
    - Else evaluate right operand

Operators

- Left operand can be used to prevent right operand from causing arithmetic error such as divide by zero
- Reduces computation time by eliminating redundant calculations

■ Logic Operators

AND , **OR** , **NAND** , **NOR**

Operators

- Relational Operators

  = , /= , < , <= , > , >=

  – Operands must be of the same type

  – Yield Boolean results

- Equality, Inequality Operators

  = , /=

  – Operands of any type

Operators

- Concatenation Operator

  &

  – Operates on one-dimensional arrays to form a new array
- Arithmetic

  ***** , **/**

  – Operate on integer, floating point and physical types types.

Operators

- Modulo, Remainder

  mod , **rem**

  – Operate only on integer types.
- Absolute Value

  abs

  – Operates on any numeric type

Operators

- Exponentiation

  **

  – Integer or floating point left operand
  – Integer right operand required
  – Negative right operand requires floating point left operand

Sources

Max Salinas - VI Workshop Revision

Prof. K. J. Hintz

Department of Electrical and Computer Engineering

George Mason University

End of Leture