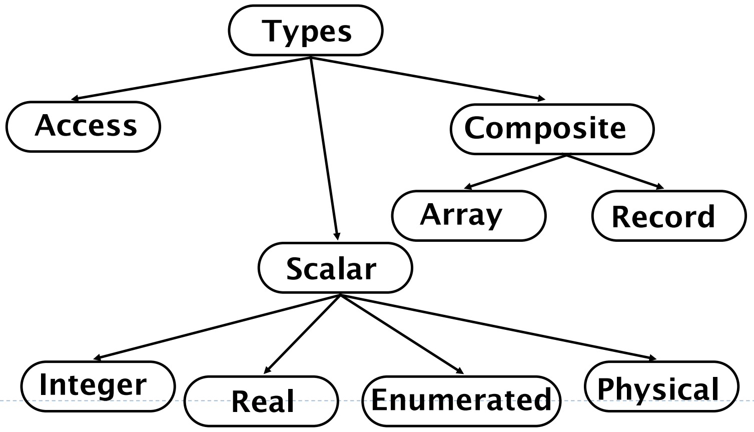
Data Types

- All declarations of VHDL ports, signals, and variables must specify their corresponding type or subtype



Scalar DataTypes and Operations

- Type of a data object defines the set of
  - values it can assume
  - operations that can be performed

- A scalar type consists of single individual.

Constant in VHDL

- Constants need to be declared in the following format:

  constant identifier : subtype := expression ;
  - **identifier** is the name of the constant
  - **subtype** is the of the constant
  - **expression** is the value of the constant
- Example:

  constant n_of_bytes : integer := 3;

  constant n_of_bits: integer 8*n_of_bytes;

  constant p: real := 3.418281828;

  constant actdelay : time := 5 ns;

  constant s_limit, max_index: integer := 512;

Variables in VHDL

- Variables need to be declared in the following format:

  variable identifier : subtype := expression ;
  - **identifier** is the name of the variable
  - **subtype** is the of the variable
  - **expression** is the optional initial value of the variable
- Example:

  variable sum, avg, large: real;

  variable index: integer := 0;

  variable start, end : time := 0 ns;

VHDL Data Types
Scalar Types

- Integer
  - Minimum range for any implementation as defined by standard: - 2,147,483,647 to 2,147,483,647
  - Example assignments to a variable of type integer :

```
ARCHITECTURE test_int OF test IS
BEGIN
PROCESS (X)
       VARIABLE a: INTEGER;
BEGIN
       a := 1;  -- OK
       a := -1;  -- OK
       a := 1.0;  -- illegal
END PROCESS;
END test_int;
```

VHDL Example

- A VHDL model that shows the constant and variable declarations.

```
architecture exmpl of ent is
        constant pi : real := 3.14159;
begin
        process is
                variable cnt: integer;
                variable f: real;
        begin
                cnt:= integer(pi); -- rounding pi to the nearest integer
                                -- statements using pi, f, and cnt
                cnt:=cnt+1; -- immediately change the cnt to a new value
                f := real(cnt); -- converting cnt to a real representation
                -- statements using pi, f, and cnt
        end process;
end architecture sample;
```

VHDL Data Types
Scalar Types (Cont.)

- Real
  - Minimum range for any implementation as defined by standard: -1.0E38 to 1.0E38
  - Example assignments to a variable of type real :

```
ARCHITECTURE test_real OF test IS
BEGIN
PROCESS (X)
       VARIABLE a: REAL;
BEGIN
       a := 1.3;  -- OK
       a := -7.5;  -- OK
       a := 1;  -- illegal
       a := 1.7E13;  -- OK
       a := 5.3 ns;  -- illegal
END PROCESS;
END test_real;
```

Floating-Point Types

- We can define new floating type as before
  - Definition example:

    type voltagelevel is range 0.0 to +10.0;

    type voltageswing is range -10.0 to +10.0;

- Variable declaration:

  variable a: voltagelevel;

Scalar Types

- Type in VHDL is very important:
  - In VHDL every object may only assume values of its nominated type.
  - In VHDL every operation include the type in which the operation can be applied.
- Early error detection.??
- Types are declared as follow: type identifier is type definition

Scalar Types Example

- Variables of atype can not be assigned to variables of btype

```
type atype is range 0 to 100;
type btype is range 0 to 100;
```

VHDL Data Types
Subtypes

- Subtype
  - Allows for user defined constraints on a data type
    - e.g. a subtype based on an unconstrained VHDL type
  - May include entire range of base type
  - Assignments that are out of the subtype range are illegal
    - Range violation detected at run time rather than compile time because only base type is checked at compile time
  - Subtype declaration syntax :

```
SUBTYPE name IS base_type RANGE <user range>;
```

  - Subtype example :

```
SUBTYPE first_ten IS INTEGER RANGE 0 TO 9;
```

Integer type

- Type declaration examples:

```
type day is range 0 to 31;
type year is range 0 to 2100;
type index_range is range 20 downto 10;
```

- Variable declaration examples:

```
variable today day := 3;
variable ayear year := 1999;
variable index: index_range; -- index is initially 20
```

Operation on Integer type

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

VHDL Data Types
Scalar Types (Cont.)

- Enumerated
  - User specifies list of possible values
  - Example declaration and usage of enumerated data type :

```
TYPE binary IS ( ON, OFF );
... some statements ...
ARCHITECTURE test_enum OF test IS
BEGIN
PROCESS (X)
        VARIABLE a: binary;
BEGIN
        a := ON;   -- OK
        ... more statements ...
        a := OFF;   -- OK
        ... more statements ...
END PROCESS;
END test_enum;
```

VHDL Physical Data Types

- Physical
  - Require associated units
  - Range must be specified
  - Example of physical type declaration :

```
TYPE resistance IS RANGE 0 TO 10000000

UNITS
        ohm;  -- ohm
        Kohm = 1000 ohm;  -- i.e. 1 KΩ
        Mohm = 1000 kohm;  -- i.e. 1 MΩ
END UNITS;
```

  - Time is the only physical type predefined in VHDL standard

Time Type

- Is a predefined physical type. Its definition is:

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

Enumeration Types

- Used to model of hardware at an abstract level.
  - A set of names is used to encode signal values.
- Enumeration type:
  - Literal values are enumerated in a list
  - The list must contain at least one value
- Example:

```
type alu operation is (pass, add, sub, xor, and);
type octallit is ('0','1','2','3','4','5','6','7');
```

Enumeration Types/Variables

- Variable declaration and usage:

```
type alu operation is (pass, add, sub, xor, and);
type octallit is ('0','1','2','3','4','5','6','7');

variable odigit: octallit := '0';
variable aluf : alu_operation := pass ;

aluf := add;
odigit := '7';
```

Enumeration

- Literal pass is overloaded

```
type alu1_operation is (pass, add8, sub8, xor8, and8);
type alu2_operation is (pass, add16, sub16, xor16, and16);
```

  To distinguish which one for the reader:

```
alu1_operation'(pass), alu2_operation'(pass)
```

- Predefined types in VHDL:

```
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);
type boolean is (false, true);
type bit is ('0', ,'1');
```

Subtype

- Define a restricted set of value from a base type:
- Examples:

```
subtype natural  is integer range 0 to highest integer;
subtype positive is integer range 1 to highest integer;
```

Attributes of Scalar Types

- attributes give information about the values included in the type.
- Predefined attributes for all scalar types:
  - T is the name of a scalar type
  - x is a value of that type
  - s is a string value

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

Attributes (Examples)

```
type resistance is range 0 to 1E9
units
        ohm;
        kohm = 1000 ohm;
        Mohm = 1000 kohm;
end units resistance;
type set index is range 21 downto 11;
type logicv is (unknown, low, undriven, high);
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'left = 21
set index'right = 11
set index'low = 11
set index'high = 21
set index'ascending = false
set index'image(14) = "14"
set index'value("20") = 20
logicv'left = unknown
logicv'right = high
logicv'low = unknown
logicv'high = high
logicv'ascending = true
logicv'image(undriven) = "undriven"
logicv'value("Low") = low
```

Attributes

- For a discrete scalar and physical type T, a value x of that type and an integer n the attributes are:

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:

```
type logicv is (unknown, low, undriven, high);

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

VHDL Objects

- There are four types of objects in VHDL
  - Constants
  - Variables
  - Signals
  - Files
- The scope of an object is as follows :
  - Objects declared in a package are available to all VHDL descriptions that *use* that package
  - Objects declared in an entity are available to all architectures associated with that entity
  - Objects declared in an architecture are available to all statements in that architecture
  - Objects declared in a process are available only within that process

VHDL Objects
Constants

- Name assigned to a specific value of a type
- Allow for easy update and readability
- Declaration of constant may omit value so that the value assignment may be deferred
  - Facilitates reconfiguration
- Declaration syntax :

```
CONSTANT constant_name : type_name [:= value];
```

- Declaration examples :

```
CONSTANT PI : REAL := 3.14;
CONSTANT SPEED : INTEGER;
```

VHDL Objects
Variables

- Provide convenient mechanism for local storage
  - E.g. loop counters, intermediate values
- Scope is process in which they are declared
  - VHDL '93 provides for global variables, to be discussed in the *Advanced Concepts in VHDL* module
- All variable assignments take place immediately
  - No delta or user specified delay is incurred
- Declaration syntax:

```
VARIABLE variable_name : type_name [:= value];
```

- Declaration examples :

```
VARIABLE opcode : BIT_VECTOR(3 DOWNTO 0) := "0000";
VARIABLE freq : INTEGER;
```

VHDL Objects
Signals

- Used for communication between VHDL components
- Real, physical signals in system often mapped to VHDL signals
- ALL VHDL signal assignments require either delta cycle or user-specified delay before new value is assumed
- Declaration syntax :

```
SIGNAL signal_name : type_name [:= value];
```

- Declaration and assignment examples :

```
SIGNAL brdy : BIT;
brdy <= '0' AFTER 5ns, '1' AFTER 10ns;
```

Signals and Variables

- This example highlights the difference between signals and variables

```
ARCHITECTURE test1 OF mux IS
SIGNAL x : BIT := '1';
SIGNAL y : BIT := '0';
BEGIN
PROCESS (in_sig, x, y)
      BEGIN
            x<=in_sig XOR y;
            y<=in_sig XOR x;
      END PROCESS;
END test1;
```

```
ARCHITECTURE test2 OF mux IS
SIGNAL y : BIT := '0';
BEGIN
PROCESS (in_sig, y)
      VARIABLE x :BIT:= '1';
BEGIN
      x := in_sig XOR y;
      y <= in_sig XOR x;
END PROCESS;
END test2;
```

- **Assuming a 1 to 0 transition on *in_sig*, what are the resulting values for *y* in the both cases?**

VHDL Objects
Signals vs Variables

- A key difference between variables and signals is the assignment delay

```
ARCHITECTURE sig_ex OF test IS
PROCESS (a, b, c, out_1)
BEGIN
        out_1 <= a NAND b;
        out_2 <= out_1 XOR c;
END PROCESS;
END sig_ex;
```

Time	a	b	c	out_1	out_2
0	0	1	1	1	0
1	1	1	1	1	0
1+d	1	1	1	0	0
1+2d	1	1	1	0	1

VHDL Objects
Signals vs Variables (Cont.)

```
ARCHITECTURE var_ex OF test IS
BEGIN
PROCESS (a, b, c)
VARIABLE out_3 : BIT;
BEGIN
        out_3 := a NAND b;
        out_4 <= out_3 XOR c;
END PROCESS;
END var_ex;
```

Time	a	b	c	out_3	out_4
0	0	1	1	1	0
1	1	1	1	0	0
1+d	1	1	1	0	1

Simulation Cycle Revisited
Sequential vs Concurrent Statements

- VHDL is inherently a concurrent language
  - All VHDL processes execute concurrently
  - Concurrent signal assignment statements are actually one-line processes
- VHDL statements execute sequentially *within a process*
- Concurrent processes with sequential execution within a process offers maximum flexibility
  - Supports various levels of abstraction
  - Supports modeling of concurrent and sequential events as observed in real systems

Concurrent Statements

- Basic granularity of concurrency is the *process*
  - Processes are executed concurrently
  - Concurrent signal assignment statements are one-line processes
- Mechanism for achieving concurrency :
  - Processes communicate with each other via signals
  - Signal assignments require delay before new value is assumed
  - Simulation time advances when all active processes complete
  - Effect is concurrent processing
    - I.e. order in which processes are actually executed by simulator does not affect behavior
- Concurrent VHDL statements include :
  - Block, process, assert, signal assignment, procedure call, component instantiation

Sequential Statements

- Statements inside a *process* execute sequentially

```
ARCHITECTURE sequential OF test_mux
IS
BEGIN
select_proc : PROCESS (x,y)
BEGIN
IF (select_sig = '0') THEN
            z <= x;
ELSIF (select_sig = '1') THEN
            z <= y;
ELSE
            z <= "XXXX";
END IF;
END PROCESS select_proc;
END sequential;
```

```
TYPE level IS ('X', '0', '1', 'Z'); -- enumerated type

TYPE level_vector IS ARRAY (NATURAL RANGE <>) OF level;
-- type for vectors (buses)

SUBTYPE delay IS TIME; -- subtype for gate delays
```

```
ENTITY and2 IS

  GENERIC(trise : delay := 10 ns;
          tfall : delay := 8 ns);

  PORT(a, b : IN level;
       c : OUT level);

END and2;
```

```
ARCHITECTURE behav OF and2 IS

  BEGIN

    one : PROCESS (a,b)

      BEGIN
        IF (a = '1' AND b = '1') THEN
          c <= '1' AFTER trise;
        ELSIF (a = '0' OR b = '0') THEN
          c <= '0' AFTER tfall;
        ELSE
          c<= 'X' AFTER (trise+tfall)/2;
        END IF;

    END PROCESS one;

END behav;
```

Tri-State Buffer Example

```
ENTITY tri_state IS

  GENERIC(trise : delay := 6 ns;
          tfall : delay := 5 ns;
          thiz  : delay := 8 ns);

  PORT(a : IN level;
       e : IN level;
       b : OUT level);

END tri_state;
```

```
ARCHITECTURE behav OF tri_state IS

  BEGIN

    one : PROCESS (a,e)

      BEGIN
        IF (e = '1' AND a = '1') THEN
                -- enabled and valid data
          b <= '1' AFTER trise;
        ELSIF (e = '1' AND a = '0') THEN
          b <= '0' AFTER tfall;
        ELSIF (e = '0') THEN -- disabled
          b <= 'Z' AFTER thiz;
        ELSE -- invalid data or enable
          b <= 'X' AFTER (trise+tfall)/2;
        END IF;

    END PROCESS one;

END behav;
```

D Flip Flop Example

```
USE work.resources.all;

ENTITY dff IS

  GENERIC(tprop : delay := 8 ns;
          tsu   : delay := 2 ns);

  PORT(d      : IN level;
       clk    : IN level;
       enable : IN level;
       q      : OUT level;
       qn     : OUT level);

END dff;
```

```
ARCHITECTURE behav  OF dff IS
  BEGIN
   one : PROCESS (clk)
     BEGIN
       -- check for rising clock edge
       IF ((clk = '1' AND clk'LAST_VALUE = '0')
            AND enable = '1') THEN -- ff enabled
         -- first, check setup time requirement
         IF (d'STABLE(tsu)) THEN
           -- check valid input data
           IF (d = '0') THEN
             q <= '0' AFTER tprop;
             qn <= '1' AFTER tprop;
           ELSIF (d = '1') THEN
             q <= '1' AFTER tprop;
             qn <= '0' AFTER tprop;
           ELSE -- else invalid data
             q <= 'X';
             qn <= 'X';
           END IF;
         ELSE -- else violated setup time requirement
           q <= 'X';
           qn <= 'X';
         END IF;
       END IF;
   END PROCESS one;
END behav;
```