Data Types and Operators

Data types identify a set of values an object may assume and the operations that may be performed on it.

VHDL data type classifications:

- **Scalar: numeric, enumeration and physical objects**
- **Composite: Arrays and records**
- **Access: Value sets that point to dynamic variables**
- **File: Collection of data objects outside the model**

Certain scalar data types are predefined in a *package* called "*std*" (standard) and do not require a type declaration statement.

Examples:

- **boolean** *(true, false)*
- **bit** *('0', '1')*
- **integer** *(-2147483648 to 2147483647)*
- **real** *(-1.0E38 to 1.0E38)*
- **character** *(ascii character set)*
- **time** *(-2147483647 to 2147483647)*

Type declarations are used through constructs called *packages*.

We will use the package called *std_logic_1164* in our class. It contains the common types, procedures and functions we normally need.

A *package* is a group of related declarations and subprograms that serve a common purpose and can be reused in different parts of many models.

Using std_logic_1164

The package *std_logic_1164* is the package standardized by the IEEE that represents a nine-state logic value system known as *MVL9*.

To use the package we say:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

The *library* clause makes a selected library containing desired packages "visible" to a model.

The *use* clause makes the library packages visible to the model.

USE clause format:

```
USE symbolic_library.pkg_name.elements_to_use
```

The name *ieee* is a *symbolic* name. It is "*mapped*" to:

```
/usr/local/apps/mti/current/modeltech/ieee
```

using the MTI utility *vmap*.

You can see all the currently active mappings by typing: *vmap*

We do not have to declare a library work. Its existence and location "./work" is understood.

Using std_logic_1164

The nine states of std_logic_1164:
(/usr/local/apps/mti/current/modeltech/vhdl_src/ieee/stdlogic.vhd)

```
PACKAGE std_logic_1164 IS
-------------------------------------------
-- logic state system (unresolved)
-------------------------------------------
  TYPE std_ulogic IS (
'U', -- Uninitialized; the default value
'X', -- Forcing Unknown; bus contention
'0', -- Forcing 0; logic zero
'1', -- Forcing 1; logic one
'Z', -- High Impedance; 3-state buffer
'W', -- Weak Unknown; bus terminator
'L', -- Weak 0; pull down resistor
'H', -- Weak 1; pull up resistor
'-'  -- Don't care; used for synthesis);
```

Why would we want all these values for signals?

VHDL Operators

Object type also identifies the operations that may be performed on an object.

Operators defined for predefined data types in decreasing order of precedence:

- **Miscellaneous: **, ABS, NOT**
- **Multiplying Operators: *, /, MOD, REM**
- **Sign: +, -**
- **Adding Operators: +, -,&**
- **Shift Operators: ROL, ROR, SLA, SLL, SRA, SRL**
- **Relational Operators: =, /=, <, <=, >, >=**
- **Logical Operators: AND, OR, NAND, NOR, XOR, XNOR**

Not all these operators are synthesizable.

Overloading

Overloading allows standard operators to be applied to other user-defined data types.

An example of overloading is the function "AND", defined as:
(/usr/local/apps/mti/current/modeltech/vhdl_src/ieee/stdlogic.vhd)

```
FUNCTION "and" (l : std_logic; r : std_logic)
RETURN UX01;

FUNCTION "and" (l, r: std_logic_vector )
RETURN std_logic_vector;
```

For Examples

```
SIGNAL result0, signal1, signal2 : std_logic;
SIGNAL result1 : std_logic_vector(31 DOWNTO 0);
SIGNAL signal3 : std_logic_vector(31 DOWNTO 0);
SIGNAL signal4 : std_logic_vector(31 DOWNTO 0);

BEGIN
result0 <= signal1 AND signal2; -- simple AND
result1 <= signal3 AND signal4; -- many ANDs
END;
```

If we synthesize this code, what gate realization will we get?