Identifiers, Data Types, and Operators

• Identifiers
  – Basic identifiers: start with a letter, do not end with "_"
  – Case insensitive

• Data Objects
  – Signals
  – Constants
  – Variables
  – Files
### VHDL Standard Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Range of values</th>
<th>Example declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>implementation defined</td>
<td>signal index: integer := 0;</td>
</tr>
<tr>
<td>real</td>
<td>implementation defined</td>
<td>variable val: real := 1.0;</td>
</tr>
<tr>
<td>boolean</td>
<td>(TRUE, FALSE)</td>
<td>variable test: boolean := TRUE;</td>
</tr>
<tr>
<td>character</td>
<td>defined in package STANDARD</td>
<td>variable term: character := '@';</td>
</tr>
<tr>
<td>bit</td>
<td>0, 1</td>
<td>signal ln1: bit := '0';</td>
</tr>
<tr>
<td>bit_vector</td>
<td>array with each element of type bit</td>
<td>variable PC: bit_vector(31 downto 0)</td>
</tr>
<tr>
<td>time</td>
<td>implementation defined</td>
<td>variable delay: time := 25 ns;</td>
</tr>
<tr>
<td>string</td>
<td>array with each element of type character</td>
<td>variable name : string(1 to 10) := &quot;model name&quot;;</td>
</tr>
<tr>
<td>natural</td>
<td>0 to the maximum integer value in the implementation</td>
<td>variable index: natural := 0;</td>
</tr>
<tr>
<td>positive</td>
<td>1 to the maximum integer value in the implementation</td>
<td>variable index: positive := 1;</td>
</tr>
</tbody>
</table>

#### Data Types (cont.)

- Enumerated data types are particularly useful for constructing models of computing systems
  
  - **Examples**
    
    ```vhdl
type instr_opcode is ('add', 'sub', 'xor', 'nor', 'beq', 'lw', 'sw');
type state is ('empty', 'half_full', 'half_empty', 'empty');
```

- Array types
  
  ```vhdl
type byte is array (7 downto 0) of std_logic;
type word is array (31 downto 0) of std_logic;
type memory is array (0 to 4095) of word;
```
Physical Types

Physical Types

Physical Types: Example

Physical Types: Example (cont.)

type capacitance is range 0 to 1E16 units
   ffr; -- femtofarads
   pfr = 1000 ffr;
   nfr = 1000 pfr;
   ufr = 1000 nfr
   mfr = 1000 ufr
   far = 1000 mfr;
   kfr = 1000 far;
end units;

• Rather than mapping the values to the real numbers,
  create new physical types


Physical Types: Example (cont.)

entity inv_rc is
generic (c_load: capacitance := 66 ffr); -- farads
port (i1 : in std_logic;
     o1: out : std_logic);
constant rpu: resistance:= 25000 ohms;
constant rpd : resistance := 15000 ohms;
end inv_rc;
architecture delay of inv_rc is
constant tplh: time := (rpu/ 1 l_o)* (c_load/1 ffr) *3 fs/1000;
constant tpll: time := (rpu/ 1 l_o)* (c_load/1 ffr) *3 fs/1000;
begin
  o1 <= '1' after tplh when i1 = '0' else
        '0' after tpll when i1 = '1' or i1 = 'Z' else
        'X' after tplh;
end delay;

Define a new overloaded multiplication operator

This expression now becomes

rpu * c_load * 3

Modeling with Physical Types

- Use packages to encapsulate type definitions, type conversions functions and arithmetic functions for new types

- Examples
  - Modeling power
  - Modeling silicon area
  - Modeling physical resources that are "cumulative"

Operators

- VHDL ‘93 vs. VHDL ‘87 operators

<table>
<thead>
<tr>
<th>Logical operators</th>
<th>and</th>
<th>or</th>
<th>nand</th>
<th>nor</th>
<th>xor</th>
<th>xnor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational operators</td>
<td>=</td>
<td>/=</td>
<td>&lt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&gt;=</td>
</tr>
<tr>
<td>Shift operators</td>
<td>sll</td>
<td>srl</td>
<td>sla</td>
<td>sra</td>
<td>rol</td>
<td>ror</td>
</tr>
<tr>
<td>Addition operators</td>
<td>+</td>
<td>-</td>
<td>&amp;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unary operators</td>
<td>+</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplying operators</td>
<td>*</td>
<td>/</td>
<td>mod</td>
<td>rem</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miscellaneous operators</td>
<td>**</td>
<td>abs</td>
<td>not</td>
<td>&amp;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- VHDL text or language reference manual for less commonly used operators and types