Lecture 8: VHDL (2)

Computer Aided Digital Design
EE3109
Gopi K. Manne
Fall 2007

VHDL Objects
- 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 - Constant
- 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
- Declaration syntax:
  ```vhdl
  CONSTANT constant_name : type_name [:= value];
  ```
- Declaration examples:
  ```vhdl
  CONSTANT PI : REAL := 3.14;
  CONSTANT SPEED : INTEGER;
  ```

Variables
- Convenient for local storage
- Value can be changed & no time dimension associated
- Operations on variables are instantaneous
- Declaration Syntax:
  ```vhdl
  VARIABLE variable_name : type_name [:= value];
  ```
- Declaration Examples:
  ```vhdl
  VARIABLE code : BIT_VECTOR(3 DOWNTO 0) := "0000";
  VARIABLE freq : INTEGER;
  ```

Signals
- Communication between VHDL components
- Value can be changed
- Time dimension associated
- Signal is driven by one or more drivers
- Declaration Syntax:
  ```vhdl
  SIGNAL signal_name : type_name [:= value];
  ```
- Declaration Examples:
  ```vhdl
  SIGNAL brdy : BIT;
  brdy <= '0' AFTER 5ns, '1' AFTER 10ns;
  ```
- Difference between variables and signals is the assignment delay

Files
- Note that files are defined a fourth data type initially, but were reclassified as objects in VHDL in 1993
- Files provide a way for a VHDL design to communicate with the host environment
- File declarations make a file available for use to a design
- Files can be opened for reading and writing
- The package STANDARD defines basic file I/O routines for VHDL types
- The package TEXTIO defines more powerful routines handling I/O of text files
User defined data types

- User can define their own data type
- Declaration Syntax:
  ```vhdl
type data_type is type_name;
end type;
```
- Declaration Examples
  ```vhdl
type state_type is (S0, S1, S2, S3, S4);
signal state_name: state_type := S1;
```

VHDL Operators

- Logical - not, and, or, nand, nor, xor
- Relational - /=, =, <, <=, >, >=
- Shift - sll, srl, sla, sra, rol, ror
- Adding - +, -, &
- Multiplying - *, /, mod, rem
- Miscellaneous- **, abs
- Signing - +, -

VHDL Control Statements

- Similar to C or PASCAL.
- All are sequential
  - If...end if
  - case
  - loop
  - wait
  - exit
  - next
  - return

If ... End If

- evaluated from top to bottom
  ```vhdl
if conditional_expression then
  -- sequential statements
[elseif conditional_expression then]
  -- sequential statements
[end if];
```
- example
  ```vhdl
if a = 0 then
  level := 1;
else if a > 1000 then
  level := 3;
else
  level := 2;
end if;
```

Case

- Mutually exclusive sets of choices
- others is used to cover all remaining values
  ```vhdl
case expression is
  when choice1 => sequential statements;
  when choice2 => sequential statements;
  [when others => sequential statements;]
end case;
```
- example
  ```vhdl
case x (0 to 1) is
  when "00" => z := '0';
  when "01" => z := '1';
  when others => z := 'Z';
end case;
```

Modeling Styles

- Structural
  - Describe the circuit in terms of its interconnected components
  - Very low level (transistor) or a very high level description (block diagram)
- Behavioral
  - Define the operation of a circuit over time
  - Sequential statements inside a process
  - Examples include state diagrams, timing diagrams etc
- Dataflow
  - Define how data flows in the system
- Mixed
  - Combination of other styles.
**Structural**
- Focus is on how components are interconnected rather than the operation of each component
- Ability to define a list of components
- Defining signals to interconnect these components
- Distinguish between multiple copies of same component using labels

**Behavioral**
- Uses the process construct in VHDL

**Dataflow**
- Convenient for illustrating asynchronous and concurrent events, where delays represent actual hardware component delays
- Realistic way of modeling hardware dependencies and concurrencies

**Mixed**
- Combination of other styles
- Uses structural, concurrent and sequential statements

**Sequential Statements**
- Execute serially inside a process
  - Process statement
  - Wait statement
  - Variable assignment
  - Assertion statement
  - If statement
  - Case statement
  - Loop statement
  - Null statement
  - Next statement
  - Exit statement
  - Function call
  - Procedure call
  - Report statement

**Concurrent Statements**
- Execute in simultaneous fashion - parallel
  - Block
  - Concurrent signal assignment
  - Concurrent procedure call
  - Concurrent assertion
  - Generate
Structural Modeling – Example

- Ability to define a list of components
- Defining signals to interconnect these components
- Distinguish between multiple copies of same component using labels

2-bit adder from Full adder

- Full adder entity design –

```
entity FullAdder is
  port(a, b, cin: in bit; cout, sum: out bit);
end FullAdder;
```

```
entity Adder2 is
  port (A, B: in bit_vector(1 downto 0); Cin: in bit; S: out bit_vector(1 downto 0); Cout: out bit);
end Adder2;
```

For first Full adder
- A0 -> a
- B0 -> b
- Cin -> cin
- S0 -> sum
- C0 -> cout

For second Full adder
- A1 -> a
- B1 -> b
- Cin -> cin
- S1 -> sum
- Cout -> cout

2-bit Adder (Contd..)

architecture addertest of Adder2 is
  component FullAdder
    port (a, b, cin: in bit; cout, sum: out bit);
  end component;

  signal C0: bit;

  begin
    T1: FullAdder
      port map
        (A(0), B(0), Cin, S(0), C0);
    T2: FullAdder
      port map
        (A(1), B(1), Cin(0), S(1), Cout);
  end addertest;

Encoders / Decoders

<table>
<thead>
<tr>
<th>Input</th>
<th>O1</th>
<th>O2</th>
<th>O3</th>
<th>O4</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a' b'</th>
<th>a b</th>
<th>Enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>a' b'</td>
<td>a b</td>
<td></td>
</tr>
<tr>
<td>a' b'</td>
<td>a b</td>
<td></td>
</tr>
<tr>
<td>a' b'</td>
<td>a b</td>
<td></td>
</tr>
<tr>
<td>a' b'</td>
<td>a b</td>
<td></td>
</tr>
</tbody>
</table>

Encoders / Decoders

<table>
<thead>
<tr>
<th>B b B'</th>
<th>O1</th>
<th>O2</th>
<th>O3</th>
<th>O4</th>
<th>O5</th>
<th>O6</th>
<th>O7</th>
<th>O8</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Decoders / Encoders

Hierarchical Design

- Decoders
  - used in memories to select row to read or write
  - used to select one of 2N blocks to activate

- 3-8 Decoder using 2-4 Decoders
  - Design a 2-4 Decoder
  - Circuit design and Output waveform
  - Create a symbol
  - Design a 3-8 decoder
  - Circuit Design and Output Waveforms