Overview

- Part 1 - Basics and Constructs
  - VHDL basics
    - Notation
    - Types & constructs
    - Signals
    - Entities and architectures
    - Libraries and packages
  - Structural VHDL Example
  - VHDL Operators
  - Concurrent VHDL Examples
- Part 2 - Behavioral and Hierarchical Description
- Part 3 - Finite State Machines
- Part 4 - Registers and Counters
- Part 5 - Algorithmic State Machine Example: Binary Multiplier
VHDL Notation - 1

- VHDL is:
  - Case insensitive
  - Based on the programming language ADA
  - Strongly-typed language
- Comments
  -- [end of line]
- List separator: ,
- Statement terminator: ;

VHDL Notation - 2

- Types and values
  - Determined by use of packages (discussed later) that define various types and type conversions
  - IEEE 1076 predefined types:
    - type bit has two values 0 and 1
    - type bit_vector is an array of bits with integers as indices
    - type integer has values over a specified range of integers
    - type boolean is (TRUE, FALSE)
  - IEEE 1164 predefined types:
    - type std_ulogic has nine values U, X, 0, 1, Z, W, L, H, -
    - type std_ulogic_vector is an array of bits with natural (non-negative) numbers as the indices
    - subtype std_logic is std_ulogic with definitions for multiple signals applied to a single wire
    - subtype x01z is std_logic with the range X, 0, 1, Z
VHDL Notation - 3

- More on types
  - Most frequently used type: std_logic
    - Provides values needed for simulation, notably X and Z
  - Frequently used type: integer
    - Due to strong typing, essential for arithmetic operations
    - Requires additional packages to be used to perform type conversion between std_logic and integer

VHDL Notation - 4

- Constants
  - Binary
    - Single bit: '0', '1'
    - Multiple bit: B"11001", B"11_0001" (underline permitted for readability)
  - Other bases
    - Octal O"61", O"6_1"
    - Hex X"31", X"3_1"
    - Decimal 49
    - Real 49E+1
VHDL Notation - 5

- **Identifiers**
  - Examples: A, B1, abc, run, stop, c_in

- **Keywords**
  - Words reserved for special meanings
  - Cannot be used as identifiers
  - Examples: entity, architecture, and, if
  - Shown here in color
  - Shown in text in bold

VHDL Constructs

- **Structural:**
  - Describes interconnections of components (entities)
  - Analogous to logic diagrams or netlists

- **Concurrent VHDL or Dataflow:**
  - Consists of a collection of statements and processes that execute concurrently

- **Sequential VHDL:**
  - Consists of the sequences of statements within processes
  - Logic described may be combinational or sequential
Signal Declaration

- Signals can be viewed as "wires"
- Signals are concurrent and sequential objects
- A port declaration is a signal declaration with in or out added
- Examples:
  ```vhdl
  signal a, b: std_logic;
  signal widget: std_logic_vector(0 to 7);
      -- 0 is MSB and 7 is LSB
  signal c: std_logic_vector(2 downto 0);
      -- 2 is MSB and 0 is LSB
  port (DATA: in std_logic_vector(15 downto 0));
  signal product: std_logic_vector(0 to 31);
  port (NA: out std_logic);
  ```

Entities and Architectures

- **entity**
  - The primary hardware abstraction in VHDL
  - Provides: the entity name, the inputs and outputs
  - Analogous to a symbol in a block diagram
- **architecture**
  - Specifies the relationships between the inputs and outputs of a design entity
  - May be a mixture of structural, concurrent and sequential VHDL.
- A given entity may have multiple, different architectures.
- Examples of entities and architectures follow.
Libraries and Packages

- A library typically contains VHDL code or compiled VHDL code.
- A package consists of compiled VHDL code for multiple entities and associated architectures.
- A package is stored in a library.
- Example: package func_prims is stored in library lcdf_vhdl.
- func_prims provides compiled code for the following delay-free gates: and2, ..., and5, or2, ..., or5, nand2, ..., nand5, nor2, ..., nor5, not, xor2, and xnor2 in which integers 2 through 5 specify the number of gate inputs.
- Generation of the lcdf_vhdl library and the func_prims package:
  - Generate a new library named lcdf_vhdl.
  - Using the lcdf_vhdl library as the "work" library, compile the file func_prims.vhd (available from the VHDL web page) that contains the component, entity and architecture descriptions for the package.

First Example to Illustrate Entities, Architectures and Constructs

- IC7283 - a 1-bit adder from a commercial IC
A Structural VHDL Example

library IEEE, lcdf_vhdl;
use IEEE.std_logic_1164.all,
  lcdf_vhdl.func_prims.all;
entity IC7283 is
  port (A0,B0,C0: in std_logic;
        C1,S0: out std_logic);
end IC7283;
architecture structure of IC7283 is
  component NOT1
    port (in1: in std_logic;
          out1: out std_logic)
  end component;
  component NAND2
    port (in1,in2: in std_logic;
          out1: out std_logic)
  end component;
  component NOR2
    port (in1,in2: in std_logic;
          out1: out std_logic)
  end component;
  component AND2
    port (in1,in2: in std_logic;
          out1: out std_logic)
  end component;
  component XOR2
    port (in1,in2: in std_logic;
          out1: out std_logic)
  end component;
signal N1,N2,N3,N4,N5,N6,N7: std_logic;

Instantiation of two packages from two libraries. Applies only to the following entity.
Declaration of entity IC7283
Declaration of 3 inputs and 2 outputs of type std_logic.
End of entity declaration
Declaration of architecture named structure for entity IC7283
Declarations of the gate components to be used from package func_prims in library lcdf_vhdl

A Structural VHDL Example (continued)

Declarations of 7 signals for use in interconnecting the gates
A Structural VHDL Example (continued)

begin
  g0: NOT1 port map (C0,N3);
  g1: NOT1 port map (N2,N5);
  g2: NOT1 port map (N3,N6);
  g3: NAND2 port map (A0,B0,N1);
  g4: NOR2 port map (A0,B0,N2);
  g5: NOR2 port map (N2,N4,C1);
  g6: AND2 port map (N1,N3,N4);
  g7: AND2 port map (N1,N5,N7);
  g8: XOR2 port map (N6,N7,S0);
end structure;

Beginning of the body of the architecture. There is an entry for each gate: gate_identifier: gate_name keywords port map signal list: (input, output) or (input1, input2, output) End of architecture and description

VHDL Operators

- Logical: and, or, nand, nor, xor, xnor, not
- Relational: =, /=, <, <=, >, >=
- Shift: sll, srl, sla, sra, rol, ror
  - Form is sdt – s is for shift, d is direction (d = l is for left, d = r is for right, and t is type (t = l is for logical, and t = r is for rotate).
- Adding +, −, &
  - & is concatenation which permits one-dimensional operands to be placed end-to-end to form a combined operand.
  - Example: For C_in and A(3:0), C_in & A is equivalent to a 5-bit register with C_in as the MSB and A(0) as the LSB.
- Sign +, −
- Multiplying: * (multiply), /(divide), mod (modulus), rem (remainder)
- Miscellaneous: abs (absolute value), ** (exponentiation)
Concurrent VHDL

- Signal assignment
  - Uses signal assignment operator \(<=\)
  - A signal is assigned its value after a delay, whether real or a delta time, an infinitesimal interval required in VHDL simulator implementations

- Examples:
  - \(z <= a \text{ or } b;\) -- \(z\) assigned after an infinitesimal delta time
  - \(z <= a \text{ nand } b \text{ after } 10 \text{ ns};\) -- \(z\) assigned after inertial delay of 10 ns
  - \(\text{widget} <= \text{transport}("00" \& a \& b) \text{ after } 10 \text{ ns};\)
  - \(\text{&}\) is the concatenation operator.

Concurrent VHDL Example Using Boolean Equations

- The entity is the same as for the structural VHDL example
  
  ```vhdl
  architecture dataflow_1 of IC7283 is
  signal N1, N2: std_logic;
  begin -- The assignment statements are Boolean equations.
    N1 <= not (A0 and B0);
    N2 <= not (A0 or B0);
    C1 <= not ((N1 and not C0) or N2);
    S0 <= ((not N2) and N1) xor (not (not C0));
  end dataflow_1;
  ```
Concurrent VHDL Example Using "with select"

```
library IEEE, lcdf_vhdl;
use IEEE.std_logic_1164.all;
entity IC7283_ws is
  port (Z: in std_logic_vector(2 downto 0);
        CS: out std_logic_vector(1 downto 0));
end IC7283_ws;
architecture dataflow_2 of IC7283_ws is
begin
  with Z select
  begin
    CS <= "00" when "000",
         "01" when "001",
         "01" when "010",
         "10" when "011",
         "01" when "100",
         "10" when "101",
         "10" when "110",
         "11" when "111",
         "XX" when others;
  end dataflow_2;
```

Defines Z as the conditioning signal.
Forms truth table with inputs on the right and outputs on the left.
Assigns XX to CS for the other std_logic triples on Z.
Second Example to Illustrate Entities, Architectures and Constructs

- Priority Encoder

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>D4 D3 D2 D1 D0</td>
<td>A2 A1 A0 V</td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>X X X 0</td>
</tr>
<tr>
<td>0 0 0 0 1</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 1 X</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>0 0 1 X X</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>0 1 X X X</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>1 X X X X</td>
<td>1 0 0 1</td>
</tr>
</tbody>
</table>

Concurrent VHDL Example Using "when else"

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
entity priority_encoder_we is
    port (D: in std_logic_vector (4 downto 0);
        A: out std_logic_vector (2 downto 0);
        V: out std_logic);
end priority_encoder_we;
architecture dataflow_3 of priority_encoder_we is
begin
A <= "100" when D(4) = '1' -- Can customize condition
   else "011" when D(4 downto 3) = "01" -- on each
   else "010" when D(4 downto 2) = "001" -- line.
   else "001" when D(4 downto 1) = "0001"
   else "000" when D = "00001"
   else "XXX";
V <= not(D = "00000");
end dataflow_3;
```
Concurrent "when else" vs. "with select"

- with select
  - Has simple form with
    - condition signal stated only once
    - only one word per line, otherwise
  - Ideal for implementing binary (0,1) truth tables

- when else
  - Has more complex form, but
  - Able to implement much more complex decision functions
    - condensed truth tables with 0, 1, X entries in rows
    - situations with limited cases of multiple condition signals

Terms of Use

- © 2004 by Pearson Education, Inc. All rights reserved.
- The following terms of use apply in addition to the standard Pearson Education Legal Notice.
- Permission is given to incorporate these materials into classroom presentations and handouts only to instructors adopting Logic and Computer Design Fundamentals as the course text.
- Permission is granted to the instructors adopting the book to post these materials on a protected website or protected ftp site in original or modified form. All other website or ftp postings, including those offering the materials for a fee, are prohibited.
- You may not remove or in any way alter this Terms of Use notice or any trademark, copyright, or other proprietary notice, including the copyright watermark on each slide.
- Return to Title Page