Lecture #1(a)
The VHDL entity-architecture Pair

ECE 37100: Computer Organization & Design Lab
Spring 2013
Hardware Description Languages (HDLs)

- Programming languages used to describe analog and/or digital circuits and subsystems
  - Our focus will be on describing digital hardware

- Three popular HDL’s
  - VHDL: VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
  - Verilog-HDL
  - SystemVerilog

- Other HDLs
  - SystemC
  - HandleC
  - BlueSpec

- Our focus will be on learning VHDL. If you learn VHDL, learning Verilog/System-Verilog will be easy!
VHDL

- Product of a US Government request for a new means of describing digital hardware
- Modeled after the Ada programming language
- Traditionally tougher to learn as compared to SystemVerilog/Verilog-HDL

Versions

- IEEE-1076 1987
- IEEE-1076 1993
- IEEE-1076 2000 (minor changes)
- IEEE-1076 2002 (minor changes)
- IEEE-1076 2008 **(major changes)**
VHDL Identifiers (i.e. Names and Labels)

- VHDL is case insensitive
  - Databus DATABUS
  - Databus DataBus

- General rules of thumb
  - All identifiers should start with alphabet characters (a-z or A-Z)
  - Use only alphabet characters (a-z, A-Z), digits (0-9), and underscores (_)
  - Do not use any punctuation or reserved characters within a name (!, ?, ., &,, +, -, etc.)
  - Do not use two or more consecutive underscore characters (___) within an identifier (e.g., Sel___A is invalid)
  - All identifiers (i.e. names/labels) in a module must be unique
VHDL Comments

- **Single line**: Start with `--` and end with carriage return
  - Can be placed anywhere on a line
  - Text after `--` treated as comment text

- **Multiline**: Start with `/*` and end with `*/`
  - Can start anywhere on the line
  - Text between `/* */` pair treated as comment text
  - **Only supported in VHDL-2008**

- **Example**

```vhdl
/*
   Obtain data from memory circuit (VHDL-2008)
*/
data_in <= data_bus; -- read memory data from bus
```
VHDL Example: 2-input NAND Gate

- Three parts to a VHDL model
- File extension for VHDL files is `.vhd` or `.vhdl`
- Filename should be the same as entity name

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity nand2 is
  port( a : in std_logic;
       b : in std_logic;
       f : out std_logic );
end entity nand2;

architecture model of nand2 is
begin
  f <= not(a and b);
end architecture model;
```

Library/Use Clauses
Entity Declaration
Architecture Declaration
Lecture #1 (a): VHDL Entity & Architecture

VHDL entity Declaration
VHDL Entity Declaration

- Defines component input/output interface ("black box" description)
- Must have unique name
  - entity name same as file name
- May have different implementations
- **Example**

```vhdl
entity nand2 is
  port( a : in std_logic;
       b : in std_logic;
       f : out std_logic );
end entity nand2;
```
VHDL Entity Declaration (cont’d)

- Made up of two **optional** parts: **generic** and **port** clauses
- **Port Clause**: Specifies **signal names**, **modes**, **types**, and **values**
  - **Signal Names**: Any valid VHDL identifier (i.e. any valid name)
  - **Port Modes**: Describes direction of signal flow on a port
  - **Types**: Any **built-in** or **user-defined** data type
  - **Value**: **Optional** Initialization for signals (for simulation only)
- **Generic Clause**: Specifies parameters for **reusable models**
  - More on this later!

```vhdl
entity entity_name is

  [generic( generic_parameter_list)]
  [port( signal_name : mode data_type [:= init_value];
         signal_name : mode data_type [:= init_value];
         ...
         signal_name : mode data_type [:= init_value] );]
end entity entity_name;
```
Port Modes: \textit{in}

- Data is driven into this port by an external driver
  - Can only be read within the design entity
- May only appear on right hand side (RHS) of signal assignment statements

![Diagram showing input port signal, external driver, and design entity]
Port Mode: *out*

- Can only be written (updated) within design entity
  - Cannot be read within the design entity
- Can only appear on **left hand side (LHS)** of signal assignment statements

![Diagram showing output port signals and internal driver relationships.](image-url)
Port Mode: *out* (cont’d)

- How to read an output port? Decouple output port from logic dependent on the output
  - Declare and utilize an intermediate signal
  - “Connect” intermediate signal to corresponding output port.

```plaintext
Z <= X
C <= X
```

**Diagram:**
- Design Entity
- Output Port Signal
- Intermediate signal “connects” to output port

**Diagram Image:**
- Internal Driver
- Output ports read within design entity with *intermediate signals*
Port Modes Summary

- Most common modes are in and out
- Read output ports using intermediate signals
- ECE 37100 lab assignment & project guidelines
  - Only use in and out port modes unless otherwise specified
Lecture #1 (a): VHDL Entity & Architecture

VHDL architecture Declaration
VHDL Architecture

- Describes internals of design entity
- Multiple architecture definitions possible for one design entity

Design Module

Entity Declaration

Architecture 1

Architecture 2

Architecture 3

---

```vhdl
architecture model1 of nand2 is
begin
  f <= not(a and b);
end architecture model1;
```

```vhdl
architecture model2 of nand2 is
  signal x : std_logic;
begin
  x <= a and b;
  f <= not x;
end architecture model2;
```
VHDL Architecture (cont’d)

- Consists of two parts: **declarative** part and **statement** part
  - Must have a unique architecture name
  - Must associate the architecture with only one design entity
  - Should place an entity/architecture pair in one file

```vhdl
architecture arch_name of entity_name is
  [component declarations;]
  [signal or variable declarations;]
  [constant declarations;]
  [type declarations;]
  [subprogram declarations;]
begin
  [concurrent signal statements;]
  [concurrent process statements;]
  [instantiations;]
end architecture arch_name;
```
Classes of Architecture Models

- HDL’s (including VHDL) allow the designer to model digital systems using various levels of abstraction via three types of modeling styles

  Three Types of Architecture Models
  - **Dataflow**: Describes how data moves (flows) through the system and the various processing steps involved.
  - **Structural**: Describes logic by its sub-components and interconnections (similar to wiring a schematic/breadboard)
  - **Behavioral**: Describes logic algorithmically (i.e. describes how a “black-box” of logic responds to input changes, no matter what is inside and how it really works)

- The three modeling styles can be used exclusively or in any combination to model digital systems