Quick Points

- Midterm was a semi-success
  - Right time estimate, wrong planet (Pluto?)
  - Everyone did OK
- HW #3 extended to Thursday, 10/18 (12:00pm)

- Resources for the next couple of weeks
  - Will add some VHDL links to CprE 583 web page sometime this week

VHDL

- VHDL is a language for describing digital hardware used by industry worldwide
  - **VHDL** is an acronym for **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit **D**escription **L**anguage

  - Developed in the early ’80s
  - Three versions in common use: VHDL-87, VHDL-93, VHDL-01

VHDL v. Verilog

<table>
<thead>
<tr>
<th>VHDL</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Government Developed</td>
<td>Commercially Developed</td>
</tr>
<tr>
<td>Ada based</td>
<td>C based</td>
</tr>
<tr>
<td>Strongly Type Cast</td>
<td>Mildly Type Cast</td>
</tr>
<tr>
<td>Difficult to learn</td>
<td>Easier to Learn</td>
</tr>
<tr>
<td>More Powerful</td>
<td>Less Powerful</td>
</tr>
</tbody>
</table>

Outline

- Introduction
- VHDL Fundamentals
- Design Entities
- Libraries
- Logic, Wires, and Buses
- VHDL Design Styles
- Introductory Testbenches
**Naming and Labeling**

- VHDL is **not** case sensitive
  
  Example:
  
  Names or labels
  
  \[ \text{databus} \]
  
  \[ \text{Databus} \]
  
  \[ \text{DataBus} \]
  
  \[ \text{DATABUS} \]
  
  are all equivalent

**Naming and Labeling (cont.)**

General rules of thumb (according to VHDL-87)

1. All names should start with an alphabet character (a-z or A-Z)
2. Use only alphabet characters (a-z or A-Z) digits (0-9) and underscore (_)
3. Do not use any punctuation or reserved characters within a name (!, ?, ., &, +, -, etc.)
4. Do not use two or more consecutive underscore characters (___) within a name (e.g., Sel__A is invalid)
5. All names and labels in a given entity and architecture must be unique

**Free Format**

- VHDL is a “free format” language
  
  No formatting conventions, such as spacing or indentation imposed by VHDL compilers. Space and carriage return treated the same way.
  
  Example:
  
  ```
  if (a=b) then
  or
  if (a = b) then
  or
  if (a = b) then
  are all equivalent
  ```

**Comments**

- Comments in VHDL are indicated with a “double dash”, i.e., “--”
  
  - Comment indicator can be placed anywhere in the line
  
  - Any text that follows in the **same** line is treated as a comment
  
  - Carriage return terminates a comment
  
  - No method for commenting a block extending over a couple of lines
  
  Examples:
  
  ```vhd
  -- main subcircuit
  Data_in <= Data_bus;  -- reading data from the input FIFO
  ```

**Design Entity**

- **Design Entity** - most basic building block of a design
  
  One entity can have many different architectures

**Entity Declaration**

- **Entity Declaration** describes the interface of the component, i.e. the **input** and **output** ports

```vhd
ENTITY nand_gate IS
  PORT (a : IN STD_LOGIC;
        b : IN STD_LOGIC;
        z : OUT STD_LOGIC);
END nand_gate;
```
Entity Declaration (cont.)

ENTITY entity_name IS
PORT (
  port_name : signal_mode signal_type;
  port_name : signal_mode signal_type;
  ............
  port_name : signal_mode signal_type);
END entity_name;

Entity Declaration and Architecture

nand_gate.vhd

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
PORT ( 
  a : IN STD_LOGIC;
  b : IN STD_LOGIC;
  z : OUT STD_LOGIC);
END nand_gate;
ARCHITECTURE model OF nand_gate IS
BEGIN
  z <= a NAND b;
END model;

Port Modes

- The Port Mode of the interface describes the direction in which data travels with respect to the component
  - In: Data comes in this port and can only be read within the entity. It can appear only on the right side of a signal or variable assignment
  - Out: The value of an output port can only be updated within the entity. It cannot be read. It can only appear on the left side of a signal assignment
  - Inout: The value of a bi-directional port can be read and updated within the entity model. It can appear on both sides of a signal assignment
  - Buffer: Used for a signal that is an output from an entity. The value of the signal can be used inside the entity, which means that in an assignment statement the signal can appear on the left and right sides of the <= operator

Library Declarations

IEEE Library declaration

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
PORT ( 
  a : IN STD_LOGIC;
  b : IN STD_LOGIC;
  z : OUT STD_LOGIC);
END nand_gate;
ARCHITECTURE model OF nand_gate IS
BEGIN
  z <= a NAND b;
END model;
Library Declarations (cont.)

```
LIBRARY  library_name;
USE library_name.pkg_name.pkg_parts;
```

Library Components

```
LIBRARY
PACKAGE 1
PACKAGE 2

TYPES
CONSTANTS
FUNCTIONS
PROCEDURES
COMPONENTS
```

Common Libraries

- **IEEE**
  - Specifies multi-level logic system, including STD_LOGIC, and STD_LOGIC_VECTOR data types
  - Needs to be explicitly declared
- **STD**
  - Specifies pre-defined data types (BIT, BOOLEAN, INTEGER, REAL, SIGNED, UNSIGNED, etc.),
    arithmetic operations, basic type conversion functions, basic text I/O functions, etc.
  - Visible by default
- **WORK**
  - Current designs after compilation
  - Visible by default

STD_LOGIC Demystified

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
  PORT (a : IN STD_LOGIC; b : IN STD_LOGIC; z : OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE model OF nand_gate IS
  BEGIN
    z <= a NAND b;
  END model;
```

STD_LOGIC Demystified (cont.)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>'X'</td>
<td>Forcing (Strong driven) Unknown</td>
</tr>
<tr>
<td>'0'</td>
<td>Forcing (Strong driven) 0</td>
</tr>
<tr>
<td>'1'</td>
<td>Forcing (Strong driven) 1</td>
</tr>
<tr>
<td>'Z'</td>
<td>High Impedance</td>
</tr>
<tr>
<td>'W'</td>
<td>Weak (Weakly driven) Unknown</td>
</tr>
<tr>
<td>'L'</td>
<td>Weak (Weakly driven) 0. Models a pull down.</td>
</tr>
<tr>
<td>'H'</td>
<td>Weak (Weakly driven) 1. Models a pull up.</td>
</tr>
<tr>
<td>'-'</td>
<td>Don't Care</td>
</tr>
</tbody>
</table>

Resolving Logic Levels

```
<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>0</th>
<th>1</th>
<th>Z</th>
<th>W</th>
<th>L</th>
<th>H</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>Z</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>Z</td>
<td>W</td>
<td>L</td>
<td>H</td>
<td>X</td>
</tr>
<tr>
<td>W</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>L</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>L</td>
<td>W</td>
<td>L</td>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>H</td>
<td>W</td>
<td>W</td>
<td>H</td>
<td>X</td>
</tr>
<tr>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Wires and Buses

- SIGNAL a : STD_LOGIC;
- SIGNAL b : STD_LOGIC_VECTOR(7 DOWNTO 0);

Standard Logic Vectors

- SIGNAL a: STD_LOGIC;
- SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
- SIGNAL c, d, e: STD_LOGIC_VECTOR(7 DOWNTO 0);

Vectors and Concatenation

- SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
- SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
- SIGNAL c, d, e: STD_LOGIC_VECTOR(7 DOWNTO 0);

VHDL Design Styles

- ENTITY xor3 IS
- PORT(
  A : IN STD_LOGIC;
  B : IN STD_LOGIC;
  C : IN STD_LOGIC;
  Result : OUT STD_LOGIC);
end xor3;

XOR3 Example

Dataflow Descriptions

- Describes how data moves through the system and the various processing steps
- Dataflow uses series of concurrent statements to realize logic
  - Concurrent statements are evaluated at the same time
  - Order of these statements doesn’t matter
- Dataflow is most useful style when series of Boolean equations can represent a logic
XOR3 Example (cont.)

ARCHITECTURE dataflow OF xor3 IS
SIGNAL U1_out: STD_LOGIC;
BEGIN
  U1_out <= A XOR B;
  Result <= U1_out XOR C;
END dataflow;

Structural Description

- Structural design is the simplest to understand
- Closest to schematic capture
- Utilizes simple building blocks to compose logic functions
- Components are interconnected in a hierarchical manner
- Structural descriptions may connect simple gates or complex, abstract components
- Structural style is useful when expressing a design that is naturally composed of sub-blocks

Component and Instantiation

- Named association connectivity (recommended)
- Positional association connectivity (not recommended)

Behavioral Description

- Accurately models what happens on the inputs and outputs of the black box
- Uses PROCESS statements in VHDL

Testbenches

- Processes generating stimuli
- Design Under Test (DUT)
- Observed Outputs
Testbench Definition

- **Testbench** applies stimuli (drives the inputs) to the Design Under Test (DUT) and (optionally) verifies expected outputs
- The results can be viewed in a waveform window or written to a file
- Since **Testbench** is written in VHDL, it is not restricted to a single simulation tool (portability)
- The same **Testbench** can be easily adapted to test different implementations (i.e. different architectures) of the same design

Testbench Anatomy

ENTITY tb IS
END tb;
ARCHITECTURE arch_tb OF tb IS
-- Local signals and constants
COMPONENT TestComp -- All DUT component declarations
PORT ( );
END COMPONENT;
BEGIN
  testSequence: PROCESS
    -- Input stimuli
  END PROCESS;
  DUT:TestComp PORT MAP( );
END arch_tb;

Testbench for XOR3

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY xor3_tb IS
END xor3_tb;
ARCHITECTURE xor3_tb_architecture OF xor3_tb IS
COMPONENT xor3
  PORT:
    A : IN STD_LOGIC;
    B : IN STD_LOGIC;
    C : IN STD_LOGIC;
    Result : OUT STD_LOGIC );
END COMPONENT;
-- Stimulus signals - mapped to the input and inout ports of tested entity
SIGNAL test_vector: STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL test_result : STD_LOGIC;
BEGIN
  UUT : xor3 PORT MAP( A => test_vector(0),
                  B => test_vector(1),
                  C => test_vector(2),
                  Result => test_result);
  Testing: PROCESS
    test_vector <= "000";
    WAIT FOR 10 ns;
    test_vector <= "001";
    WAIT FOR 10 ns;
    test_vector <= "010";
    WAIT FOR 10 ns;
    test_vector <= "011";
    WAIT FOR 10 ns;
    test_vector <= "100";
    WAIT FOR 10 ns;
    test_vector <= "101";
    WAIT FOR 10 ns;
    test_vector <= "110";
    WAIT FOR 10 ns;
    test_vector <= "111";
    WAIT FOR 10 ns;
  END PROCESS;
END xor3_tb_architecture;

What is a Process?

A process is a sequence of instructions referred to as sequential statements

- A process can be given a unique name using an optional LABEL
- This is followed by the keyword PROCESS
- The keyword BEGIN is used to indicate the start of the process
- All statements within the process are executed SEQUENTIALLY. Hence, order of statements is important
- A process must end with the keywords END PROCESS

Process Execution

- The execution of statements continues sequentially till the last statement in the process
- After execution of the last statement, the control is again passed to the beginning of the process

Program control is passed to the first statement after BEGIN
### WAIT Statements

- The last statement in the PROCESS is a **WAIT** instead of **WAIT FOR 10 ns**.
- This will cause the PROCESS to **suspend indefinitely** when the **WAIT** statement is executed.
- This form of **WAIT** can be used in a process included in a testbench when all possible combinations of inputs have been tested or a non-periodical signal has to be generated.

#### Testing: PROCESS

```
BEGIN
  test_vector<="00";
  WAIT FOR 10 ns;
  test_vector<="01";
  WAIT FOR 10 ns;
  test_vector<="10";
  WAIT FOR 10 ns;
  test_vector<="11";
  WAIT;
END PROCESS;
```

### WAIT FOR vs. WAIT

**WAIT FOR**: waveform will keep repeating itself forever

```
0 1 2 3 0 1 2 3 ...
```

**WAIT**: waveform will keep its state after the last wait instruction.

```
0 1 2 3 ...
```

### Loop Statement

- Loop Statement
  - Repeats a Section of VHDL Code
  - Example: process every element in an array in the same way

#### Loop Statement Example

```
BEGIN
  test_vector<="00";
  FOR i IN 0 TO 7 LOOP
    WAIT FOR 10 ns;
    test_vector<=test_vector+"001";
  END LOOP;
END PROCESS;
```

### Loop Statement Example (cont.)

```
BEGIN
  test_ab<="00";
  test_sel<="00";
  FOR i IN 0 TO 3 LOOP
    FOR j IN 0 TO 3 LOOP
      WAIT FOR 10 ns;
      test_ab<=test_ab+"01";
    END LOOP;
    test_sel<=test_sel+"01";
  END LOOP;
END PROCESS;
```