VHDL Tutorial

Introduction
VHDL stands for VHSIC (Very High Speed Integrated Circuits) Hardware Description Language. It provides a way of programming a circuit using code rather than schematics. VHDL is one type of hardware description language (HDL). Verilog is another HDL, and is the language used in CIS371.

In later labs, we will explore the many features and capabilities of this powerful tool. For now, we will introduce you to the very basics of the language and how it can greatly simplify the representation of a circuit.

Structure of a VHDL File – The Entity
A digital system in VHDL consists of a design entity which can contain other entities. Each entity is modeled by an entity declaration and an architecture body. The declaration appears in the following form:

```
entity NAME_OF_ENTITY is
    Port ( signal_name1 : mode type;
           signal_name2 : mode type;
           ...
    signal_name2: mode type;
end NAME_OF_ENTITY;
```

As you will soon see, Xilinx helps generate the above code based on the input and output signals you provide.

Architecture
The architecture body specifies how the circuit operates and how it is implemented. For a simple circuit such as the half adder, all of your actual coding can go here. It appears as follows:

```
architecture architecture_name of NAME_OF_ENTITY is
begin
    your statements here
end architecture_name
```
Syntax

VHDL is similar to other programming languages in its syntax. For example, statements must end in a semi-colon (;). Also, with the assignment operation, the value of the argument to the right gets stored in the variable on the left as follows:

**Assignment:** signalout <= signalin;

For the above statement to be true, both signalin and signalout must be defined earlier.

Similar to binary operators in other languages (i.e. +, -, *), VHDL statements use particular keywords for logic gates to function as binary logic operators.

**Some Logic Operators:** and, or, not, xor

One point worth mentioning is the use of parentheses, which can be used to clarify an assignment as well as specify grouping. Incorrect (or lack of) use may change a statement’s meaning altogether.

**Parentheses:** C <= (not A) and B; *is not the same as* C <= not (A and B);

One last note to make is the use of comments. In VHDL, when a double dash (--) is used, any text to the right will be treated as a comment and will not be interpreted by the compiler. Comments are useful in explaining code.

**Comments:** C <= not A  -- this is a comment

With these tools in hand, you should be able to code the adders designed in this lab. The guide below will explain the process, while walking you through a trivial case.
Procedure

1. To begin, right-click on your project in the Sources window, and select “New Source.”

![Image of Sources window with New Source option highlighted]

2. Choose VHDL Module and give it a new name. Click “Next.”

![Image of project settings with VHDL Module selected and new name entered]

3. On the next screen, you define the modules. For “Port name” you will enter the name for an input or output signal, one on each line. Under “Direction” specify whether it is input or output. “Architecture Name” should be “Behavioral.” The rest of the other entries should remain empty. When you are done, click “Next.”

![Image of Entity Name and Architecture Name settings with myAND2 and Behavioral selected]

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>in</td>
</tr>
<tr>
<td>Y</td>
<td>in</td>
</tr>
<tr>
<td>Z</td>
<td>out</td>
</tr>
</tbody>
</table>

4. The next and last screen should provide a summary of your module. If you are satisfied, click “Finish.”

5. Xilinx should automatically generate most of the VHDL code necessary for your module to run. You will see many commented lines as well as use and library statements. Ignore these for now.
6. Now you must write lines of code to describe the architecture properly. Using the syntax described earlier and the definition for your outputs, you should be able to define the outputs in terms of the inputs.

```vhdl
30 entity myAND2 is
31 Port ( X : in STD_LOGIC;
32 Y : in STD_LOGIC;
33 Z : out STD_LOGIC);
34 end myAND2;
35
36 architecture Behavioral of myAND2 is
37 begin
38 Z <= X and Y;
39 end Behavioral;
```

7. Select the “Processes” window to the left (if you see the Hierarchy window, you will need to click the left tab). Double-click “Check syntax” to verify that you have coded correctly. If there are any errors, they will be shown in the bottom window.

8. To test the operation of your module, create a new source within your project and select “Test Bench WaveForm” (similar to testing schematics). This time, however, you must associate the VHDL module with the waveform. Repeat the process as before to assign input waveforms and generate the output waveforms.