VHDL Introduction
Computer System Design Lab
CDA 4203L
TA: Andrew White
Fall 2003

VHDL - **V**HSIC (Very High Speed Integrated Circuit) **H**ardware **D**escription **L**anguage

VHDL is a programming language used to describe a digital circuit by behavior, function, and/or structure. If you think of a VHDL model as a program, then executing the program results in the simulation of the circuit it describes.

VHDL offers many advantages to the digital designer:
- Portability
- Reusability
- Technology Independent (Mostly?)
- Standardized
- Industry Support
- Documentation

Key points about VHDL
VHDL is **case insensitive**. Example: "SIGNAL", "signal", and "SiGnAL" specify the same thing. VHDL is a **strongly typed language**: data types for the most part are NOT interchangeable.

A basic VHDL design can be divided into two blocks or parts: the *entity* and the *architecture*. The entity statement defines the external interface (inputs and outputs) of the circuit. The architecture defines the internals of the circuit, which can be done in four ways:
- Behavioral, describes the behavior of a circuit using sequential statements
- Structural, describes the circuit in terms of the interconnection of components (Hierarchical)
- Data-Flow, describes the flow of information through the system
- Mixed, a combination of the above

Important Links to use in this course
University of Maryland, CSEE Dept VHDL guide
http://www.csee.umbc.edu/help/VHDL/summary_one.html

Accolade's Online VHDL Guide
http://www.acc-eda.com/vhdlref/

Peter Ashenden's VHDL Quick Start Lecture Slides
http://www.ashenden.com.au/designers-guide/DG-intro-lectures.html

```
-- Libraries to include, think of these as #include directives in C
-- The USE statements indicate what to use in that library
-- Libraries can contain data types, functions, other VHDL modules, and more
LIBRARY ieee;
  use ieee.std_logic_1164.ALL;
LIBRARY WORK;
  use WORK.ALL;



-- This ENTITY DECLARATION statement defines what the inputs and outputs are for
-- Our design
entity c_nand is
  port (input1 : in std_logic_vector(7  downto 0);
        input2 : in std_logic_vector(7  downto 0);
        output : out std_logic_vector(7 downto 0));
end  c_nand;



-- The Architecture specifies the actual internals of the circuit: behavior, structure, and/or data
-- flow
-- THIS ARCHITECTURE IS A BEHAVIORAL DESCRIPTION
architecture behavior of c_nand is

            -- Can put signal declarations, Component Declarations, Type
            -- Definitions, Procedure/Function Definitions, etc Here!

begin


             -- Process statement: Everything in this block happens sequentially
             -- The (input1,input2) next to the process keyword specifies the SENSITIVITY LIST,
             -- Every time input1 or input2 changes, the code in this process block gets re-
             -- evaluated which may result in the output changing
             P0 : process(input1,input2)

                        -- Variables are only used within process statements and must be defined
                        -- here before the begin keyword
                        variable  result : std_logic_vector(7 downto 0);

            begin
                        -- After begin  put all sequential statements for process here
                        outer : for n in 7 downto 0 loop
                                    result(n) := input1(n) NAND input2(n);
                        end loop outer;

                        output <= result;

            end process P0;

end behavior;
```

VHDL to know for first lab:

- **Comments**

"--" – single line comment indicator. Anything after the "--" will be ignored by a compiler or synthesis tool and is used for comments

- **Entity –** defines the outside interface to the circuit
  ex.

```
entity reg4 is
        port (do,d1,d2,d3,en,clk : in bit;
        qo,q1,q3,q4: out bit;);
end entity reg4;
```

- **Architecture** – construct that describes the internal functionality
  ex1.

```
architecture behav of reg4 is
        -- Signal Declarations, Component Declarations, Type
        -- Type Definitions, Procedure/Function Definitions, etc Here!
begin
        -- Put Concurrent Statements Here!
        ……
end architecture behav;
```

  Between the keywords *begin* and *end* there are concurrent statements, which as the name implies, execute concurrently. Processes, component instantiations, and signal assignments are all forms of concurrent statements that can go here.

- **Signals** – Signals are objects in VHDL. They are used to model carriers (wires or nets) in a circuit. The inputs and outputs specified in the port map are considered signals. Signals are also used to interconnect components in a structural statement. Also see the "<=" construct.

  Signal Declaration:

```
signal frame_bit : std_logic;
```

  signal *signal_name* **:** *signal_data_type***;**

- **Signal Assignment**

"<=" Operator – The signal assignment statement is a concurrent assignment of the right

to the left, unless within a process statement which occurs after a wait statement (see process). The left must be a signal and the right can be either a signal, variable, a valid value for the signal's data type, or combinational result.

ex.

```
a_xor_b <= inputA XOR inputB;
```

- **Process** – Sequential or Algorithmic execution of statements within the process is used for behavioral modeling of a circuit. Pure Behavioral Modeling only uses process statements. The process "waits" for an event (change of value) to occur on the signals it is "sensitive" to. These signals are specified in the *sensitivity list* or by *wait* statements. A process can have either *wait* statements or a *sensitivity list*. Once an event occurs on one of these signals, the process resumes execution. Any signal assignment statements within a process do not "take effect" until a *wait* statement has been reached in the process or, if a *sensitivity list* is used, until the process has reached the end. Unlike signal assignments, variable assignments occurring within a process statement occur immediately. Variables need to be declared before the begin keyword of a process.

  A process statement (or block) is considered a concurrent statement. This means that even though the contents of a process statement is executed sequentially, the entire process occurs concurrent to the other concurrent statements of an architecture. Architectures can have multiple processes within them.

  ex.1

```
-- Uses wait statements

-- Labels, like "label_a :", are optional
label_a: process is
        -- Put  any Variable Declarations here
begin

        -- After reserved word begin, put sequential statements to be
        -- executed within process
        and_a <= a1 and a2;

        -- this wait statement below waits for a1 and a2 to change before
        -- executing the process again
        wait on a1, a2;

end process  label_a;
```

ex.2

```
-- Same as example 1, except uses a sensitivity list

label_a: process(a1, a2)
        -- Put any Variable Decelerations here
begin

        -- After reserved word begin, put sequential statements to be
        -- executed within process
        and_a <= a1 and a2;

end process label_a;
```

- **Variable**s – Variables are used within processes and are "local" in scope to that process. These should be used sparingly as most FPGA/CPLD CAD tools will not synthesize these. Variable assignments within a process occur immediately, though support is growing.

- **Variable Assignments**

":=" Operator – Variable assignments statement are transactions assigning the value on the right to the variable on the left. Variables can be assigned the value of a signal, another variable, a valid value for the variable's data type, or a combinational result of a signal or variable (AND, OR, etc).

ex.

```
temp1 := input1 AND input2;
```

- **Loops/Conditional/Case Sequential Statements –** Go inside the process block

IF/ELSIF/ELSE – Conditional statement. It is good design practice to always have a full conditional branch – meaning always have an else part.

ex.

```
if clear = '1' then
        Q <= '0';
elsif clock = '1' then
        Q <= D;
else
        -- This last else is optional, as we are not going to
        -- do anything here!
end if;
```

FOR LOOP – Looping for a range. The index variable of the loop ('i' in this example) does not have to be declared.

ex.

```
-- Labels are optional
Label_optional : for i in 0 to 7 loop
        Output(i) <= input(i);
End loop Label_optional;
```

CASE – selects behavior based on the evaluation of a value of a choice
Case statement needs to have the "when others =>" condition to handle any values not specified by the case values. The "when others =>" condition needs to be the last in the case statement.

ex.

```
Label_optional : case  my_val  is
        when 0 =>
                a:=b;
        when 1 =>
                c:=d;
        when others =>
                null;
end case Label_optional;
```

- **Data Types** – There are many types to use in VHDL, but for the first labs these will be used:

  bit : a binary type with values (0 , 1). Type std_logic below should be used instead of this type when possible.

  std_logic : a binary type, equivalent to a bit, except that in order to model digital circuits correctly, there are additoinal values this type can hold.

  'U' – uninitialized
  'X' – unknown
  '0' – low
  '1' – high
  'Z' – high impedance
  'W' – weak unknown
  'L' – weak low
  'H' – weak high
  '-' – don't care

  For XST Synthesis, the '0' and 'L' values are treated the same, as are the '1' and 'H' values. '-' values are treated as don't care and 'Z' as high impedance. 'U' and 'W' are not accepted by XST.

  std_logic_vector : A one dimensional array of std_logic values. The array has to given a valid range. Example, an 8-bit bus named output_bus would be declared as so:

  signal output_bus : **std_logic_vector**(7 downto 0);

  Or, if it is an output within a port statement of an entity block:

  output_bus : **out std_logic_vector**(7 downto 0);

  The direction of the range (the "(7 downto 0)" is the range) specifies the bit order used (what is the MSB).

  bit_vector : A one dimensional array of bit types. Properties of bit_vector array are the same as specified above for std_logic_vector. As with type bit, std_logic_vector should be used instead of this type.

- **Operators –** VHDL has built in support for many operators. Operators can also be overloaded, as is the case with many of the IEEE libraries. For this introduction, we will look at the following operators:

  AND,OR,XOR,XNOR,NOR,NAND
  Combinational operators take an operand to the left and an operand to the right. These operators only work on types bit and std_logic. They do not work on arrays

without being overloaded. They can take a single “bit” from the array as operands:

ex.

input1(4) **AND** input2(4)

NOT
This operator uses the operand to the right. The same restriction on types for the above operators apply to this one.

Lab 1: VHDL Combinational Circuit (1 of 2)
CDA 4203L
TA: Andrew White
Aug 27

Overview
Implement a simple circuit in VHDL using the combinational logic operations of XOR and OR. This lab is part of a 2 part lab. DO NOT DELETE YOUR PROJECT OR VHDL SOURCE FILE. Keep everything, as we will be using it with the next part of the lab. The lab report will be a combination of this part's material and the next part's material.

Prelab
Read through the VHDL Handout and explore the links provided on first page. This will also serve as the prelab for the second part of this lab.

Procedure
1. **Mount CSEE account**

Open the CAD tool...
2. **Start->Applications->Xilinx ISE 5->Project Navigator**

Open new project
3. **File->New Project**
4. Enter a name for the project in the *Project Name* box.
5. Enter the "H:\" in the *Project Location* dialog box.
6. Select "XST VHDL" as the *Design Flow* and click OK

Add New VHDL Source
7. **Project->New Source**
8. Choose VHDL Module in and enter a name for the VHDL file. This will be stored as "*file_name*.vhd" in the project directory on your grad account (H drive). Select "Add to Project" at bottom if it is not selected already and click **next**.
9. **In the Define VHDL Source window:**
   Port Name: input and output names of entity (cannot be reserved words)
   Add two inputs and two outputs, each must have a different name.
   Direction: Specify it as *in* for input, and *out* for output
   MSB*: leave blank
   LSB*: leave blank
   *MSB and LSB are used to specify the range of arrays like type *std_logic_vector*. So, for an 8 bit array, *MSB=7* and *LSB=0*

Behavioral Modeling with a PROCESS
10. Create a process in the architecture between the reserved words *begin* and *end*. Make the process sensitive to the two inputs using a sensitivity list for the process.
11. Inside the process, implement the following: Make one output the OR of the two inputs and make the other output the XOR of the inputs. Use signal assignment statements (2) and the XOR and OR operators. Do not use variables.

Check Syntax

12. In the "Processes for Current Source" window, right click on **Synthesize** and select **run**.
13. Once this has completed, check the bottom log window for any errors (use the scroll bar). Correct any errors that exist.
14. Expand **Synthesize** in the "Processes for Current Source" window, right click **View Synthesis Report** and select run. Print a copy of this.

Lab Report

For the Lab report, create a cover sheet with your Name, Course Name, and what day and time you attend lab (ex. MW 3:30 – 4:45). This format for cover sheet will be used for all following labs. Print a copy of your VHDL source file and your synthesis report from step *14* (use printer layout option of 4 per page for synthesis report). The lab report will be due the next lab with the lab material for the second part of the lab (there are two parts).