1. Introduction

This lab. is an introduction to the VHDL language (scientific), The goal of this lab. is to present the two methods in design (behavioral - structural), also we will learn the language syntax and we will stop at each word of the program explaining what it is and things related to it since these are the first programs.

Also we will illustrate the differences between structural and behavioral modeling in VHDL. We will build a half adder as an example, this is the first program you will write it by your hand. As an exercise you will be asked to do the full adder in the lab.

2. Entity and Architecture

The NAND gate in figure (1) can be implemented as an AND gate followed by an inverter, see figure (2).
NAND Gate
AND-NOT Implementation.

- **The ENTITY construct**

VHDL files are basically divided into two parts, the entity and the architecture. The entity is basically where the circuits (in & out) ports are defined. There is a multitude of I/O ports available, but this lab will only deal with the two most usual ones, the INput and OUTput ports. (Other types of ports are for example the INOUT and BUFFER ports.)

The entity of the circuit in figure should look something like below. Please notice that comments in the code are made with a double-dash (--).

```vhdl
ENTITY nandgate IS
PORT(
    A: IN BIT;
    B: IN BIT;
    Q: OUT BIT -- Note! No ';' here!
);
END nandgate;
```

Now that we have defined the I/O interface to the rest of the world for the NAND gate, we should move on to the architecture of the circuit.

- **The ARCHITECTURE construct**
While the entity was comparable to figure (1), the architecture is comparable to figure (2). The entity told us nothing about how the circuit was implemented, this is taken care of by the architecture part of the VHDL code. The architecture of the NAND gate matching the entity above could then be written as something like this...

```
ARCHITECTURE dataflow OF nandgate IS
  SIGNAL Q_prim: BIT;
BEGIN
  Q_prim <= A AND B; -- The AND-operation.
  Q <= NOT Q_prim; -- The inverter.
END dataflow;
```

3. VHDL in more details.

VHDL is an acronym for Very high speed integrated circuit (VHSIC) Hardware Description Language which is a programming language that describes a logic circuit by function, behavior, and/or structure.

The general format of a VHDL program is built around the concept of BLOCKS which are the basic building units of a VHDL design. Within these design blocks a logic circuit of function can be easily described.

A VHDL design begins with an ENTITY block that describes the interface for the design. The interface defines the input and output logic signals of the circuit being designed. The ARCHITECTURE block describes the internal operation of the design. Within these blocks are numerous other functional blocks used to build the design elements of the logic circuit being created.

After the design is created, it can be simulated and synthesized to check its logical operation. SIMULATION is a bare bones type of test to see if the basic logic works according to design and concept.

Many software packages used for VHDL design also support schematic capture which takes a logic schematic or state diagram and translates it into VHDL code. This, in turn, makes the design process a lot easier. However, to fine tune any design, it helps to be familiar with the actual VHDL code.

REMEMBER: You are NOT writing software. You are DESCRIBING the functionality of the hardware you want.

When writing in C or other programming language you are allowed a lot of freedom by the compiler. But in this case you are physically creating blocks of digital circuits which are wired together and have to be implemented in a chip. A simple statement in C, like a division
of two numbers, causes great problems to a VHDL compiler, and the hardware implementation is very complicated. Have this in mind when coding. Think that the compiler and synthesizer have to be able to layout and wire your design, and download it to a chip.

**Coding style**

Make sure you include all the appropriate libraries needed for your design. In some cases you will need a certain library for compiling, and a different one for synthesizing. Also, if you have created any packages, include the USE statement so that the program can find your package file. This is an example of how your program header should look:

```vhdl
-- The name of your program
-- Your name
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE (your own user defined packages)
```

**Std_logic**

It is recommended to use the multi-valued logic system from the IEEE instead of the standard 'bit' data type. The new type is called 'std_logic' and is defined in the package 'std_logic_1164' which is placed in the library IEEE (i.e. it is included by the following statement: 'use IEEE.std_logic_1164.all'.

```vhdl
TYPE STD_ULOGIC IS (
    `U`, -- uninitialized (not connected)
    `X`, -- Forcing Unknown
    `0`, -- Forcing 0
    `1`, -- Forcing 1
    `Z`, -- High Impedance
    `W`, -- Weak Unknown
    `L`, -- Weak 0
    `H`, -- Weak 1
    `-', -- don't care);
```

**4. Behavior modeling, Processes**

We can write a behavior architecture body of an entity which describes the function in an abstract way. Such an architecture body includes only process statements.

**Processes are used :**

- For describing component behavior when they cannot be simply modeled as delay elements.
- To model systems at high levels of abstraction.

**Process contains :**
Conventional programming language constructs. A process is a sequentially executed block of code, which contains:

- Evaluating expressions.
- Conditional execution.
- Repeated execution.
- Subprogram calls.
- Variable assignments, e.g., \( x := y \), which, unlike signal assignment, take effect immediately.
- if-then-else and loop statements to control flow, and
- Signal assignments to external signals.

**Notes:**

1. Signal assignment statements specify the new value and the time at which the signal is to acquire this value. The textual order of the concurrent signal assignment statements (CSAs) do NOT effect the results.
2. Processes contain sensitivity lists in which signals are listed, which determine when the process executes.
3. In reality, CSAs are also processes without the process, begin and end keywords.

**5. Structure modeling**

Structural model: A description of a system in terms of the interconnection of its components, rather than a description of what each component does. A structural model does NOT describe how output events are computed in response to input events.

**A VHDL structural description must possess:**

- The ability to define the list of components.
- The definition of a set of signals to be used to interconnect them.
- The ability to uniquely label (distinguish between) multiple copies of the same component.
6. Example: half adder

A half adder is a logic circuit that performs one-digit addition. It has two inputs (the bits to be summed) and two outputs (the sum bit and the carry bit). An example of a Boolean half adder is this circuit in figure (1):

![Half Adder Diagram](image)

The Entity for the half adder:

```vhdl
entity halfadder is
    port(
        a : in std_logic ;
        b : in std_logic ;
        c : out std_logic ;
        s : out std_logic
    );
end halfadder;
```

The Behavior Model for the half adder:

```vhdl
architecture halfadder_behavioral of halfadder is
begin
  firstproc: process (a,b)
  begin
    if(a='0' and b='0') then
      s<='0';
      c<='0';
    elsif(a='0' and b='1') then
      s<='1';
      c<='0';
    end if;
  end process;
end halfadder_behavioral;
```
elsif(a='1' and b='0') then
  s<='1';
  c<='0';
else
  s<='1';
  c<='1';
end if;

end process;

end;

The Structural Model for the half adder:

library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity myand is
  port(
    a : in std_logic ;
    b : in std_logic ;
    o : out std_logic
  );
end;

architecture andarch of myand is
begin
  o<=a and b;
end;

library IEEE;
use IEEE.STD_LOGIC_1164.all; (add it before EVERY entity declaration )
entity myxor is
  port(
    a : in std_logic ;
    b : in std_logic ;
    o : out std_logic
  );
end;
architecture xorarch of myxor is
begin
  o<=a xor b;
end;
architecture halfadder_struct of halfadder is
begin
  and1: entity work.myand(andarch) port map (a,b,c);
  xor1: entity work.myxor(xorarch) port map (a,b,s);
end;

The Test bench

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity HA_tb is
end HA_tb;

architecture HA_tb of HA_tb is

signal aa,bb,ss,cc : std_logic ;

begin
  ha1: entity work.halfadder(halfadder_behavioral) port map (aa,bb,cc,ss);
  or : ha1: entity work.halfadder(halfadder_halfadder_struct) port map (aa,bb,cc,ss);
  sim : process is
    begin
      -- manipulate input here
      aa<='0'; bb<='0'; wait for 10ns;
      aa<='0'; bb<='1'; wait for 10ns;
      aa<='1'; bb<='0'; wait for 10ns;
      aa<='1'; bb<='1'; wait for 10ns;
    end process;
end HA_tb;
7. Lab. exercise: full adder

Write a complete VHDL Behavior and structural description of a full adder, test the design using a suitable testbench (apply the truth table), and show the waveform.

--code will be posted later,,, now let's try it ourselves
8. Homework Exercises

1. Attach your lab work for entity declaration, structural architecture and behavioral architecture of Full adder. (with its test bench and diagrams)

2. Construct a 4-bit binary adder using structural way and show some random input combinations in your test bench.

3. Can this circuit be done using behavioral way? Why or why not?

4. (Bonus) write the entity and arch for the following diagram –you don’t have to simulate it–

Note: this is the circuit of 4-bit BCD adder 😊