ECSE 325 – Digital Systems
Winter 2020

Lab 2: Fixed-point Representation and Modelsim Simulation

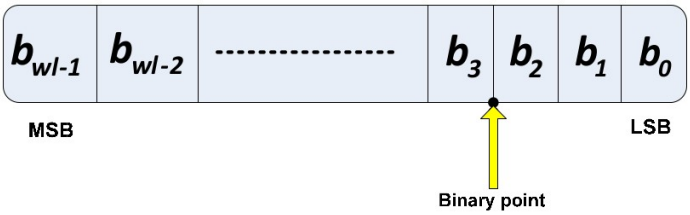
Overview

In this lab, you will learn the basics of *fixed-point representations*, VHDL *testbench* creation and *functional verification* using ModelSim. You will implement a multiplication-accumulation (MAC) unit using a fixed-point representation in VHDL, write a testbench code in VHDL and validate your design in ModelSim through following a step-by-step tutorial.

Fixed-point Representation Basics

- Given as Fixed-point:(W,F), where W: total wordlength; F: fractional length
    - Larger W,F: more precise computation
    - Smaller W, F: less area, reduced power consumption

The example below uses W=*wl+3,* F=3.



Given the two operands *a* and *b* represented using (W1,F1) and (W2,F2), respectively, the output precision required to guarantee no overflow occurrence for *a + b* is obtained by

(W1,F1)+(W2,F2)→(max(W1,W2)+1,max(F1,F2))

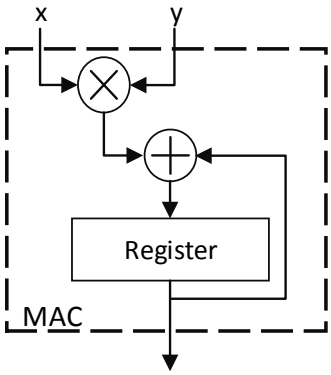
and for *a × b* is computed by

(W1,F1)+(W2,F2)→(W1 +W2,F1 +F2)

Fixed-point representation can employ either signed or unsigned formats.

Multiply-accumulate (MAC) Unit

In this lab, you will use a fixed-point MAC unit described in VHDL. The MAC unit consists of three main sub-units: multiplier, adder and register (as in figure below). The MAC unit takes two inputs at each clock cycle, multiplies and accumulates them over a given time period N.



The following pseudo-code summarizes the functionality of the MAC unit:

```
mac = 0; ready = 0;
for i = 1:N
        mac = mac + x(i) ∗ y(i);
end
ready = 1;
```

VHDL Description of MAC Unit

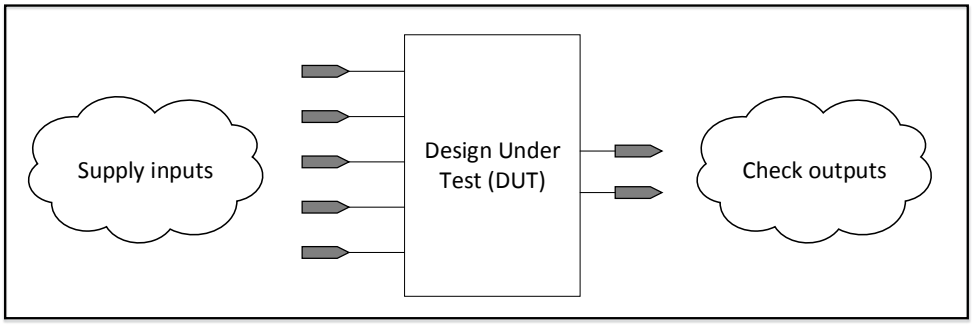
Before describing the MAC unit in VHDL, you need to find the required precision of each sub-block. You are provided with input data in files lab2-x.txt and lab2-y.txt: each file contains 1000 real values represented in floating-point format. First, using a programming language of your choice, find the precision required for each sub-unit to guarantee no overflow (i.e. no quantization error) for given data. Then, convert the integer values into 2's complement format and store them into the files named such as lab2-x- fixed-point.txt and lab2-y-fixed-point.txt. Finally, describe the MAC unit in VHDL using the obtained precision for each sub-unit and the following entity declaration (note that you have to obtain the right vector size), where NN is your group number:

```
entity gNN_MAC is
port (  x     : in std_logic_vector (?? downto 0); -- first  input, you have to first calculate length
        y     : in std_logic_vector (?? downto 0); -- second input, replace ?? with your length
        N     : in std_logic_vector (9  downto 0); -- total number of inputs
        clk   : in std_logic; -- clock
        rst   : in std_logic; -- asynchronous reset (active-high)
        mac   : out std_logic_vector (?? downto 0); -- output of MAC unit, replace the length
        ready : out std_logic); -- denotes the validity of the mac signal
end gNN_MAC;
```

Testbench Creation – VHDL and ModelSim Setup

So far, you have implemented the MAC unit in VHDL. Now, you need to test the correctness of your implementation. To this end, a non-synthesizable VHDL code known as testbench is generally used. In fact, testbench code iteratively applies a sequence of controlled inputs to a circuit and compares its output against the expected output. If a mismatch is detected, an error

is displayed in the VHDL simulators log which can then be consulted to help direct a designer search for the problem in the circuits RTL description. Note that the expected outputs are usually generated by the programming language that was used for floating-point to fixed-point conversion (why?).

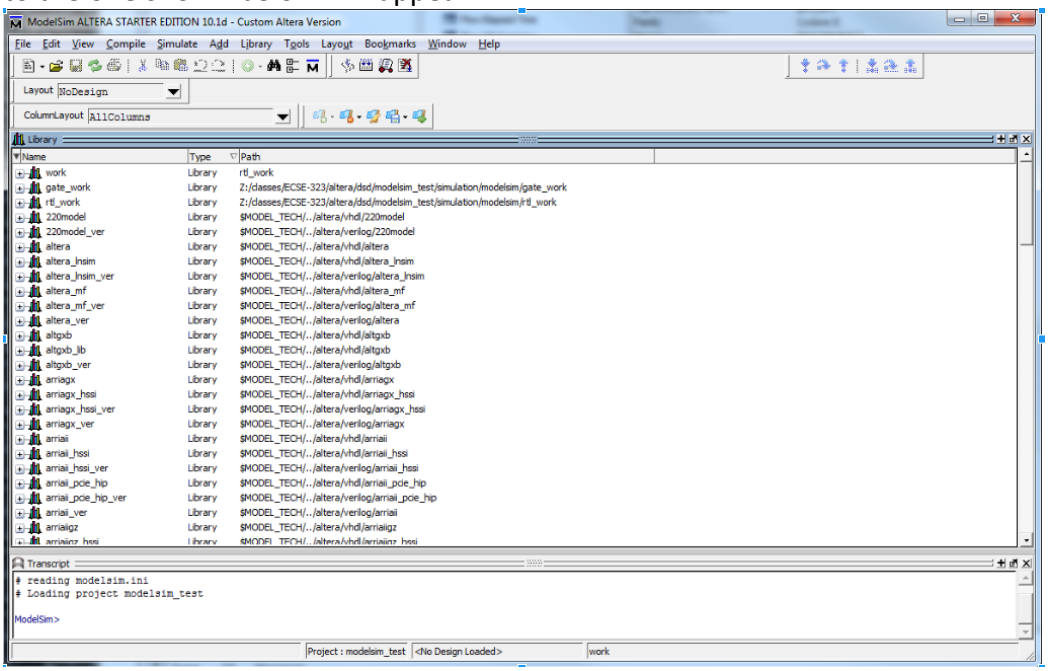


You will now write a testbench code from scratch for your implementation of MAC unit in ModelSim. Use the following steps to setup ModelSim:

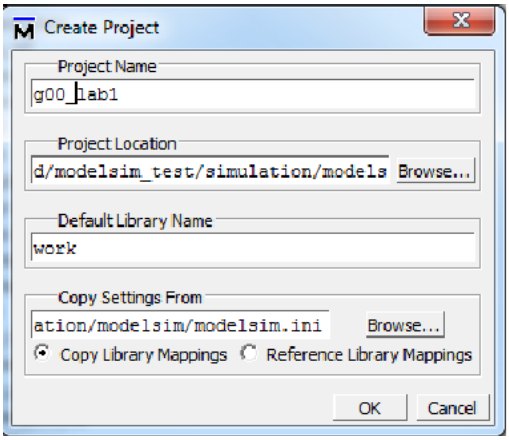
1. Launch ModelSim and Create a new project with **File > New > Project** . . .
2. Name the project and choose its path of your own choice
3. Click on the **Add Existing File** button and the VHDL code of your MAC circuit
4. Create a new VHDL file with **File > New > Source > VHDL** and name it as "gNN MAC tb.vhd"
5. The testbench entity is unique in that it has NO inputs or outputs (why?). Define an empty entity in the created VHDL file.
6. Compile the imported and created VHDL files.

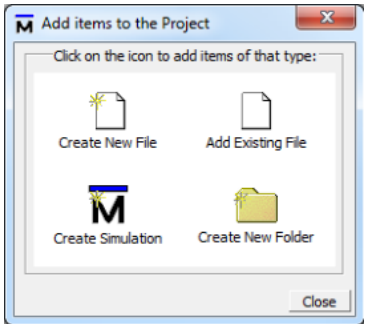
ModelSim should now successfully compile all the files.

Double-click on the ModelSim desktop icon to startup the ModelSim program. A window similar to the one shown below will appear.

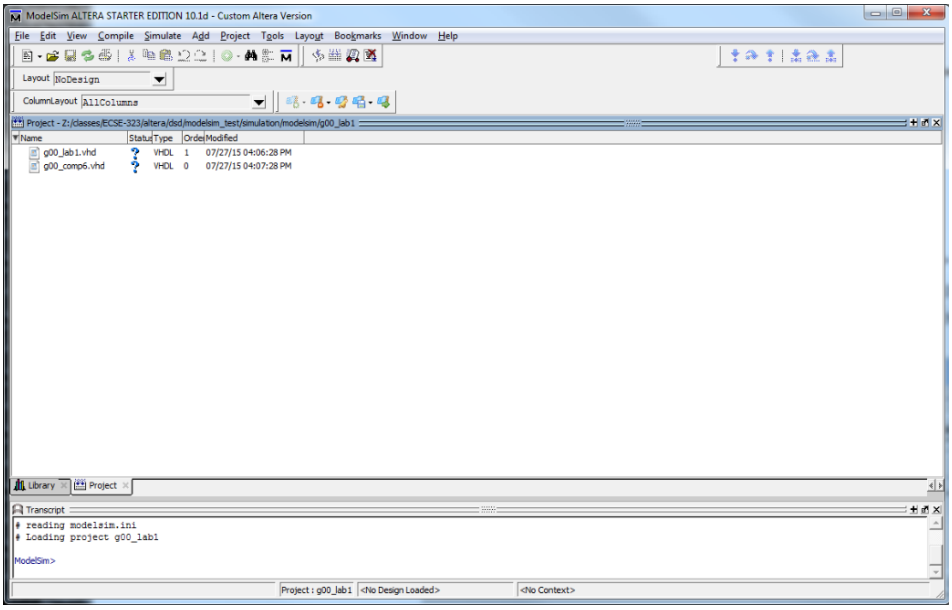


Select **File → New → Project** and, in the window that pops up, give the new project the name ”gNN lab2”. Click OK. Another dialog box will appear, allowing you to add files to the project. Click on ”**Add Existing File**” and select VHDL file that was generated earlier (gNN_MAC.vhd). You can also add files later.

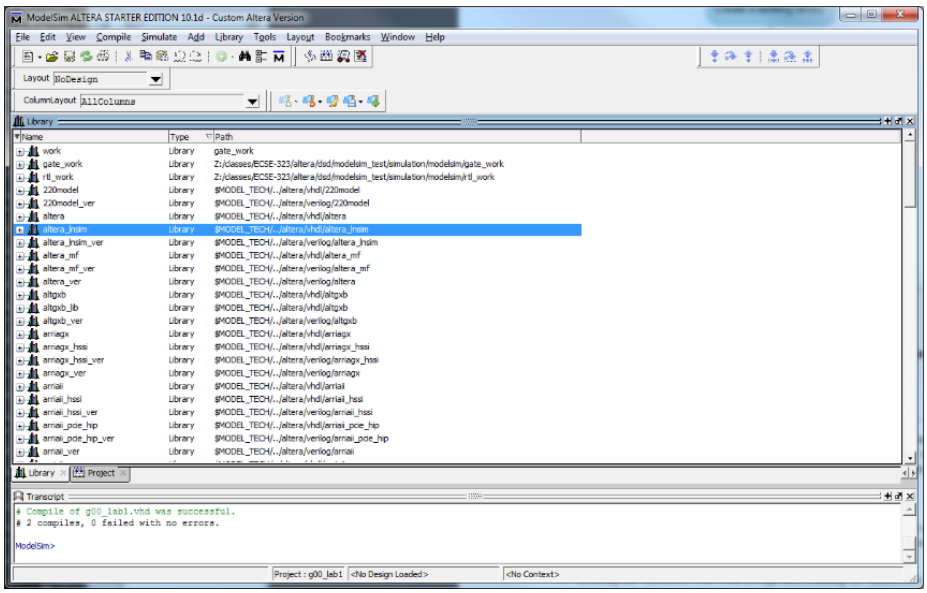




The ModelSim window will now show your VHDL file in the Project pane. An example is depicted below.

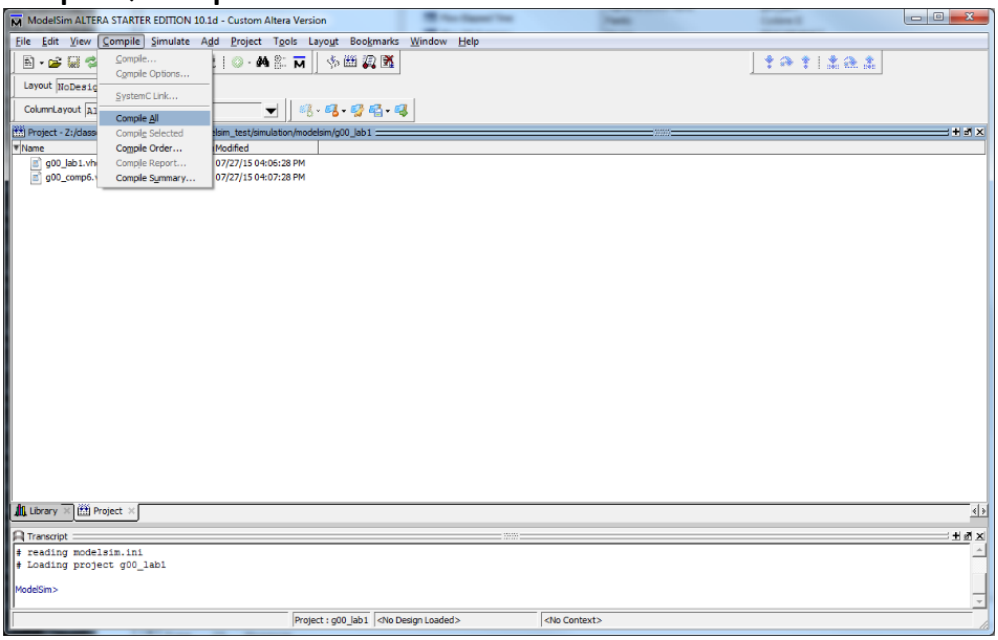


To simulate the design, ModelSim must analyze the VHDL files, a process known as *elaboration*. The compiled files are stored in a library. By default, it is named ”work”. You can see this library in the ”library” pane of the ModelSim window.

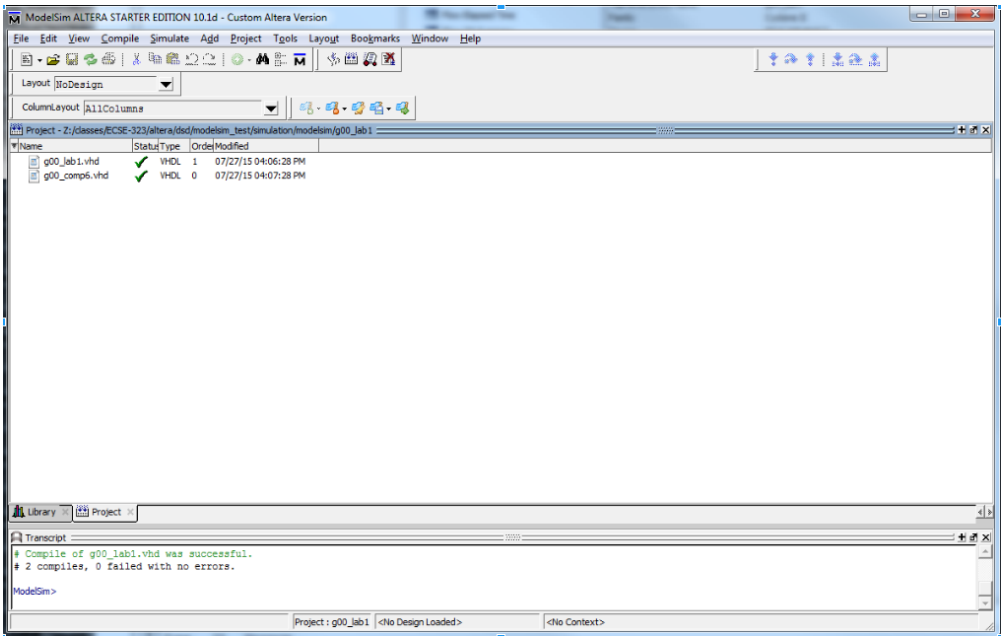


Setting ModelSim

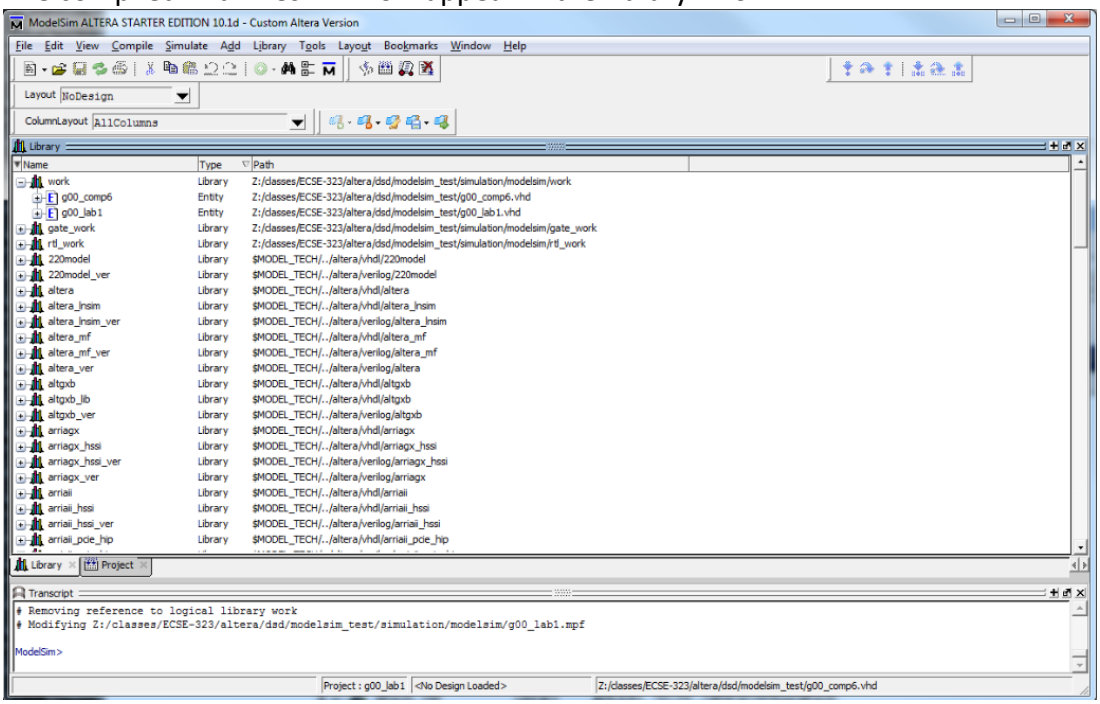
The question marks in the Status column in the Project tab indicate that either the files have not been compiled into the project or the source file has changed since the last compilation. To compile the files, select **Compile → Compile All** or right click in the Project window and select **Compile → Compile All**.



If the compilation is successful, the question marks in the Status column will turn to green check marks, and a success message will appear in the Transcript pane.

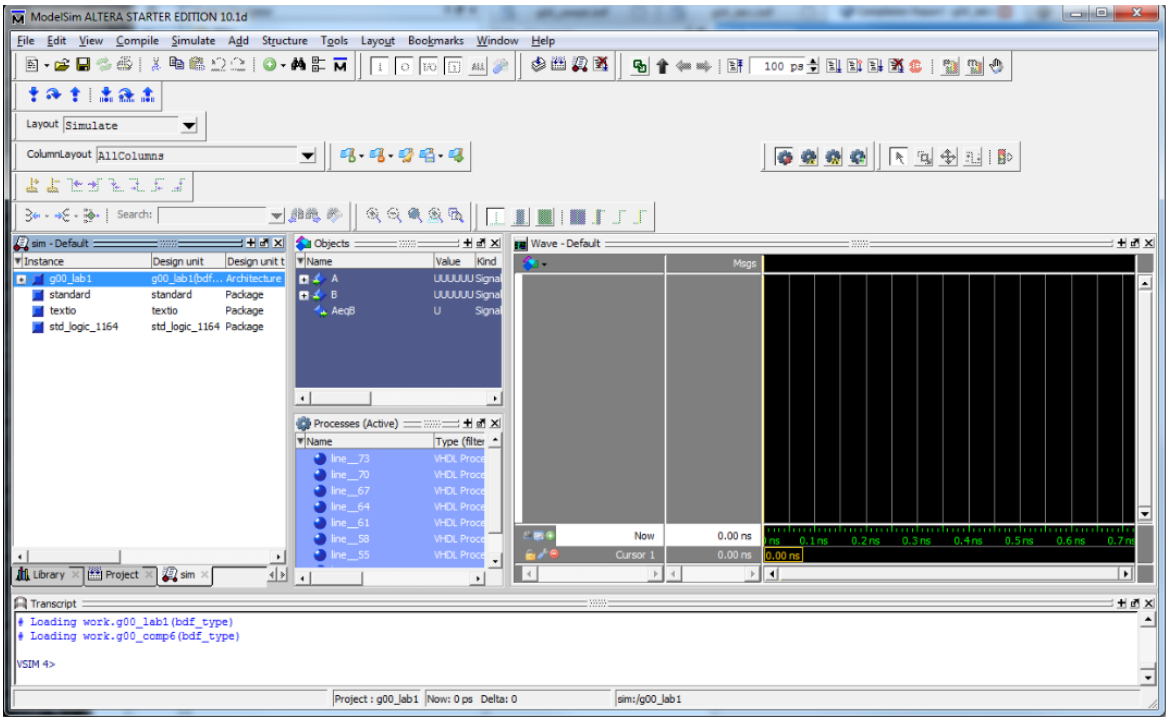


The compiled vhdl files will now appear in the library ”work”.



Starting Simulation

In the library window, double-click on gNN lab2. The system will start windows used in doing the simulation of the gNN lab2 module.



Observe that in the ”Objects” window, the signals have the value ”UUUUUU”, meaning that all inputs are undefined. If you ran the simulation now, the outputs would also be undefined.

You need to have a means of setting the inputs to certain patterns, and of observing the output responses to these inputs. In Modelsim, you use a special VHDL entity called a Testbench. A testbench is the VHDL code that generates inputs applied to your circuit, to automate the simulation of your circuit and see how its outputs respond to different inputs.

To validate the under test design (i.e., MAC), we need to add an architecture, as well as include the libraries containing various data types we are interested in manipulating (std logic, std logic vector, integer, . . .). Since the test vectors and expected outputs are usually generated in other programming languages such as C/C++, we also need **textio** package that allows the reading and writing of text files from VHDL.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use STD.textio.all;
use ieee.std_logic_textio.all;

entity gNN_MAC_tb is
end gNN_MAC_tb;

architecture test of gNN_MAC_tb is
begin
end architecture test;
```

To verify the component under test (i.e., the MAC unit), we must have access to interact with it. Therefore, the next step is to declare the design under test, instantiate from it, and wire it to the testbench. Wiring the MAC into the testbench requires information about its entity. More precisely, you need to create a signal for every port in the MAC entity as well as every I/O file. Note that it is necessary to guarantee that the testbench is in sync with what the MAC expects as inputs/outputs.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_textio.all;

entity gNN_MAC_tb is
end gNN_MAC_tb;

architecture test of gNN_MAC_tb is
--------------------------------------------------------------------------------
-- Declare the Component Under Test
--------------------------------------------------------------------------------
component gNN_MAC is
  port ( x     : in  std_logic_vector(?? downto 0);
         y     : in  std_logic_vector(?? downto 0);
         N     : out std_logic_vector(9  downto 0);
         clk   : in std_logic;
         rst   : in std_logic;
         mac   : out std_logic_vector (?? downto 0);
         ready : out std_logic);
end component gNN_MAC;
--------------------------------------------------------------------------------
-- Testbench Internal Signals
--------------------------------------------------------------------------------
file file_VECTORS_X : text;
file file_VECTORS_Y : text;
file file_RESULTS : text;

constant clk_PERIOD : time := 100 ns;

signal x_in      : in  std_logic_vector(?? downto 0);
signal y_in      : in  std_logic_vector(?? downto 0);
signal N_in      : out std_logic_vector(9  downto 0);
signal clk_in    : in std_logic;
signal rst_in    : in std_logic;
signal mac_out   : in std_logic_vector (?? downto 0);
signal ready_out : in std_logic;
```

To test a component, we must have access to it. Therefore, the next step is to instantiate the MAC unit, and wire it into the testbench. After this step, we would be able to interact with the MAC and test its functionality.

```
begin       -- Instantiate MAC
 gNN_MAC_INST : gNN_MAC
   port map (
          x => x_in,
          y => y_in,
          N => N_in,
          clk => clk,
          rst => rst,
          mac => mac_out,
          ready => ready_out
    );
```

In almost any testbench, a clock signal is usually required in order to synchronize stimulus signals within the testbench. One way to generate the clock signal is provided below.

```
--------------------------------------------------------------------------
-- Clock Generation
--------------------------------------------------------------------------
clk_generation : process
begin
   clk <= '1';
   wait for clk_PERIOD / 2;
   clk <= '0';
   wait for clk_PERIOD / 2;
end process clk_generation;
```

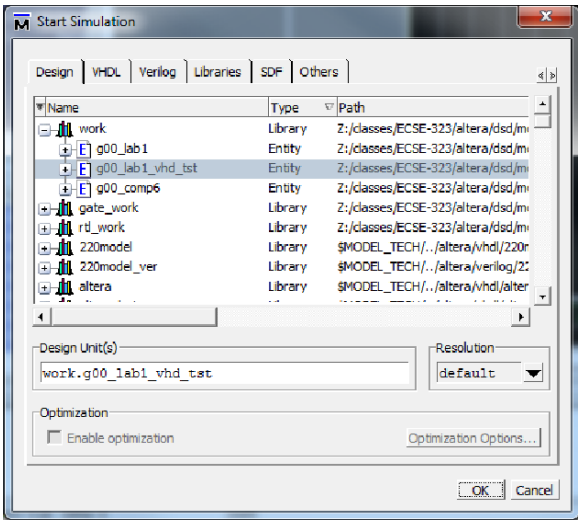
Now that the MAC unit is instantiated and wired into the testbench, we can start reading the test vectors into the circuit by the following steps:

- Read the input test vectors from the input data files (i.e., lab2-x-fixed-point.txt and lab2-y-fixed-point.txt)
- Supply the input signals of the MAC unit with the data obtained from the files
- Write the final output values into a file to be checked with the expected values
- Reset the circuit under test and repeat the above steps for new test vectors

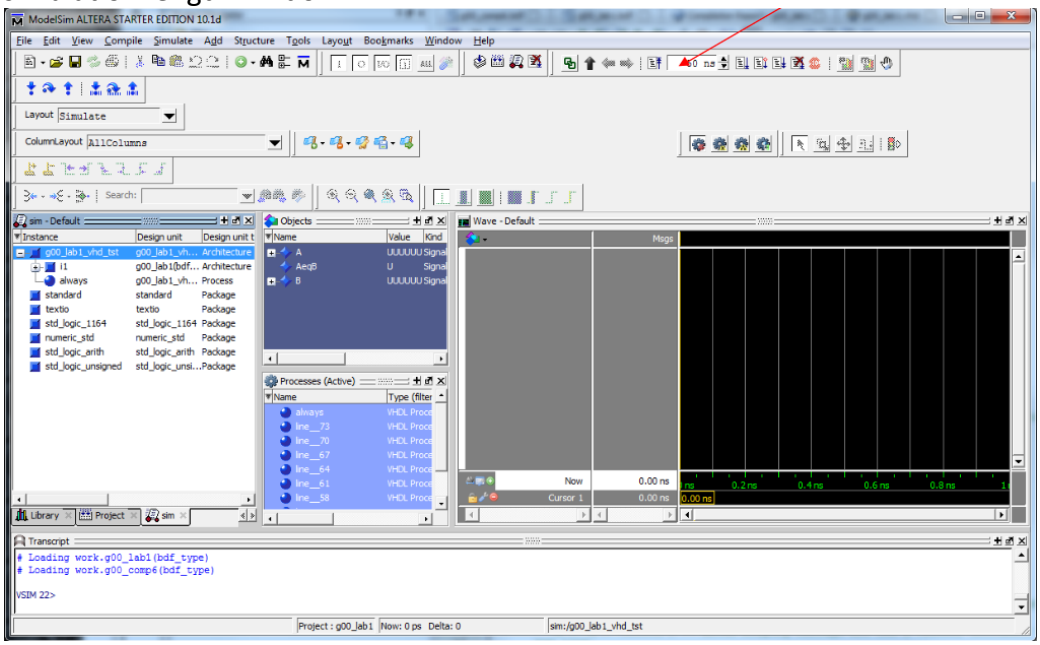
The following codes read each element of test vectors at each clock cycles and pass it to the circuit. Once the ready signal goes high, the final output is written into an output file.

```
-------------------------------------------------------------------------------
-- Providing Inputs
-------------------------------------------------------------------------------
feeding_instr : process is
   variable v_Iline1 : line;
   variable v_Iline2 : line;
   variable v_Oline : line;
   variable v_x_in     : in  std_logic_vector(?? downto 0);
   variable v_y_in     : in  std_logic_vector(?? downto 0);
  begin
   --reset the circuit
   N_in <= "1111101000"; -- N = 1000
   rst  <= '1';
   wait until rising_edge(clk);
   wait until rising_edge(clk);
   rst  <= '0';
   file_open(file_VECTORS_X, "lab2-x-fixed-point.txt", read_mode);
   file_open(file_VECTORS_Y, "lab2-y-fixed-point.txt", read_mode);
   file_open(file_RESULTS, "lab2-out.txt", write_mode);

   while not endfile(file_VECTORS_X) loop
     readline(file_VECTORS_X, v_Iline1);
     read(v_Iline1, v_x_in);
     readline(file_VECTORS_Y, v_Iline2);
     read(v_Iline2, v_y_in);

     x_in <= v_x_in;
     y_in <= v_y_in;

     wait until rising_edge(clk);
   end loop;

   if ready_out = '1' then
     write(v_Oline, mac_out);
     writeline(file_RESULTS, v_Oline);
     wait;
   end if;
  end process;
```

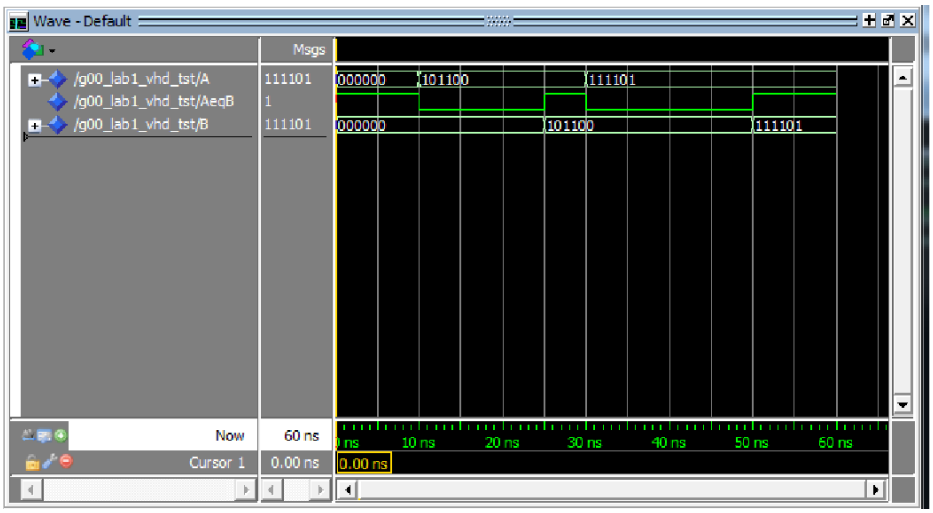
Now, one can actually run a simulation. Select ”**Start Simulation**” from the Simulate toolbar item in the ModelSim program. The following window will pop up:



The ModelSim window should now look like the figure below. Enter a value of 60ns into the simulation length window.



At first, the ”Wave” window will not have any signals in it. You can drag signals from the ”Objects” window by click on a signal, holding down the mouse button, and dragging the signal over to the Wave window. Do this for all three signals. Now, to actually run the simulation, click on the ”Run” icon in the toolbar (or press the F9 key). Below is an example output (you can right-click in the right-hand pane and select ”Zoom Full” to see the entire time range).



Exercise

So far, we have optimized the MAC unit and tested it for 1000 input instances. Now, we want to optimize the circuit for each 200-input set of the given test vectors by splitting it into 5 batches (each set contains 200 instances). Follow the steps below to customize the circuit for the new test vectors:

1. Split the 1000 input data into 5 batches, each containing 200 values
2. Convert the values of each batch into the fixed-point representation with the minimum number of bits
3. Create new test bench files for each batch (e.g. lab2-x-fixed-point-batch1.txt, lab2-y-fixed-point-batch1.txt)
4. Customize the MAC unit for the new test vectors if it is necessary
5. Write a testbench and verify the MAC unit with the new test vectors

Lab Demonstration

Each lab group will need to prepare a demonstration to TAs. The demonstration checkup sheet is attached at the end of this lab handout.

Lab Report

At the end of the lab, you should know how to prepare and compile a VHDL code and understand how it maps to FPGAs. You are required to submit a single PDF file that:

- is written in the standard technical report format
- documents every design choice clearly
- is organized for the grader to easily reproduce your results by running your code
- contains the code that is well-documented and easy to read

- should not cause the struggle for a grader to understand

Necessary parts

- An introduction in which the objective of the lab is discussed
- The VHDL code you wrote for the MAC unit, with explanation and comments
- Testbench VHDL code for the MAC unit, with explanation and comments
- Discussion on the resource utilization of your design (i.e. how many registers are used and why? What changes would you respect in the resource utilization if you increased the bit size of the counter?)
- Using provided vector set, screenshots of the testbench for verification of the MAC unit

Grading Sheet

Group Number:
Name 1:
Name 2:

Task	Grade	/Total	Comments
VHDL for MAC		/30	
Testbench VHDL		/25	
Resource Utilization		/10	
Design Verification		/35	