VHDL Assignment #2: Design and Simulation of Digital Circuits

1 Instructions

- TA in charge: Muhammad Bilal Babar (muhammad.babar@mail.mcgill.ca) - please utilize discussion boards on myCourses for questions as much as possible.
- Due date: Friday, September 23, 2022 by 11:59 pm EDT.
- Submission is in teams using myCourses (only one team member submits). In the report, provide the names and McGill IDs of the team members.
- Late submissions will incur penalties as described in the course syllabus.

2 Introduction

In this assignment, you will learn how to create a circuit using a schematic diagram in Quartus. You will then simulate the circuit in ModelSim. You will also learn how to use the test bench writer tool in Quartus. Finally, you will learn and practice the use of concurrent statements.

3 Learning Outcomes

After completing this assignment you should know how to:

- Create a schematic gate diagram of a logic circuit
- Use CAD tools and VHDL to implement logic functions
- Synthesize logic functions
- Perform functional simulation

In this assignment you will learn how to use the Altera Quartus II FPGA design and Modelsim software.

4 Creating a Schematic Diagram Design File

To get some practice with Quartus, you will design a simple 4-bit comparator circuit, which you should name `firstname_lastname_comp`. This circuit has two 4-bit inputs, $A(0), A(1), A(2), A(3)$ and $B(0), B(1), B(2), B(3)$, and a single 1-bit output, $AeqB$. The output is to be 1 when the two inputs have exactly the same values, and be 0 otherwise. The boolean equation for the output in terms of the inputs is derived as:

$$AeqB = (A(3) \text{ XNOR } B(3)) \text{ AND } (A(2) \text{ XNOR } B(2)) \text{ AND } (A(1) \text{ XNOR } B(1)) \text{ AND } (A(0) \text{ XNOR } B(0)).$$

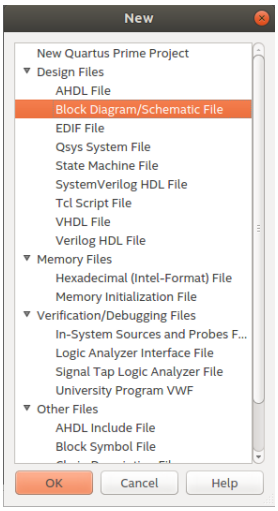
In the above, XNOR means the complement of the XOR operation.

In this course, you will learn two methods for describing circuits in Quartus:

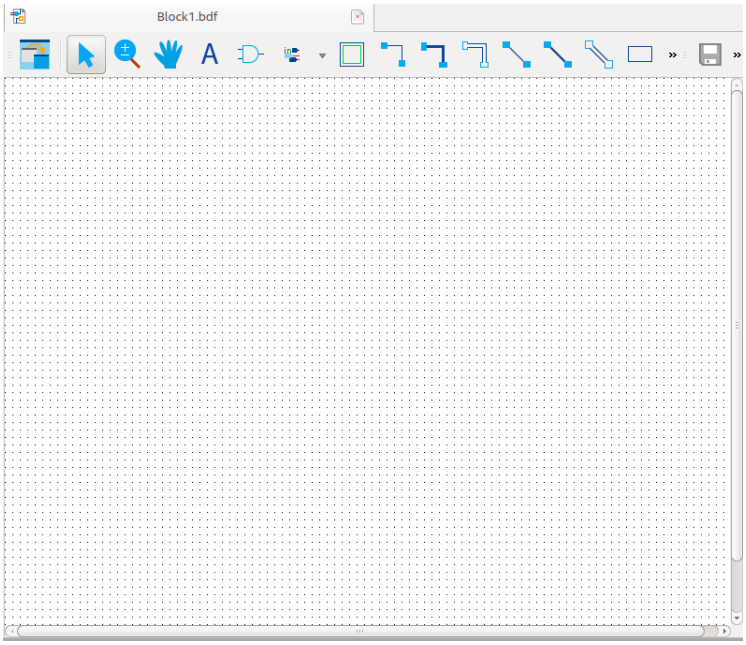
- Gate-level schematic diagrams
- VHDL descriptions

Schematic diagrams are graphical descriptions of the circuit, while VHDL uses textual descriptions. For most of the designs in the course you will use VHDL, but schematic entry is a good practice to get started with Quartus. To describe a circuit via a schematic diagram, follow the steps below.

The first step is to open the project that was created in Assignment #1. Once you have opened the project, click on the “File” menu of the main Quartus window and select the “New” menu item. The dialog box shown below will appear.

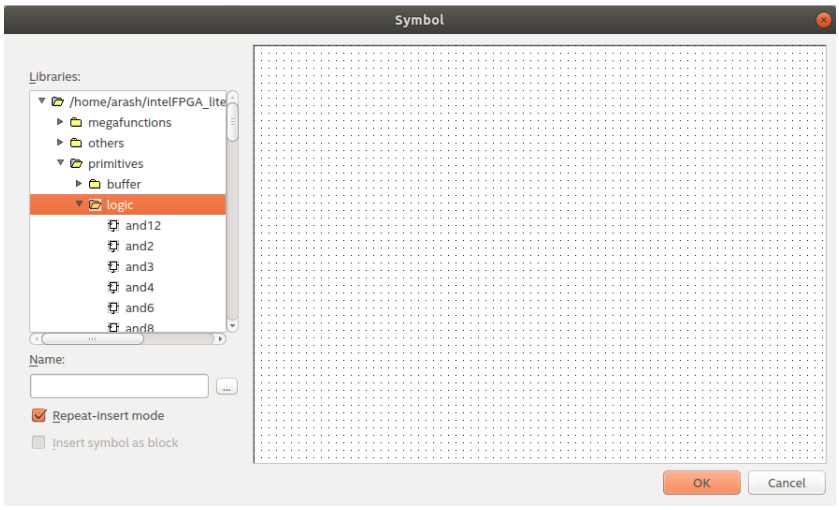


In the dialog box, select "Block Diagram/Schematic File" and click on "OK". A new window will appear in the main area of the Quartus window. You will draw your circuit schematic in this new window.

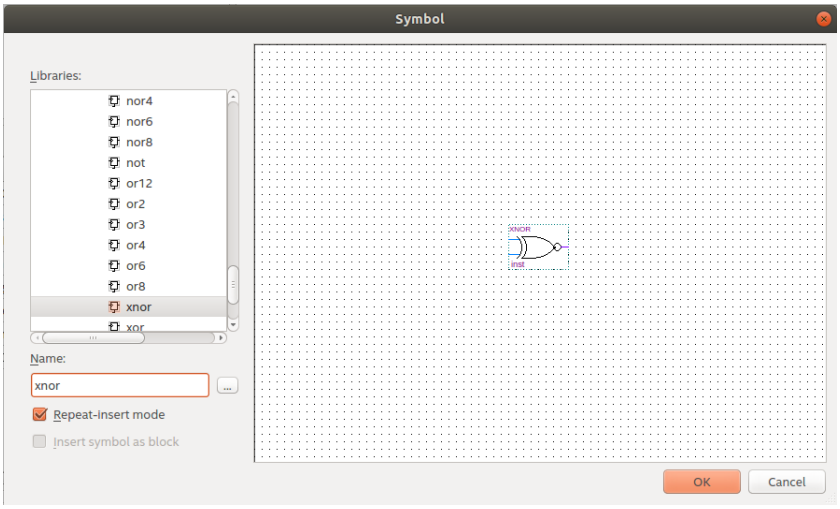


A toolbar will also appear, containing shortcuts for commands relevant to drawing schematics.

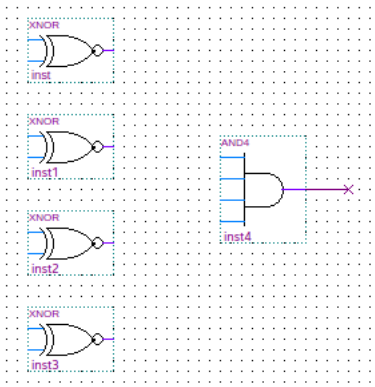
Click on the toolbar item that is shaped like an AND gate. This will bring up the "symbol" window that will allow you to select which symbol to add to your schematic drawing. Note that the symbol window can also be opened by double-clicking anywhere within the schematic window. You will want to add an XNOR gate symbol to your schematic. This can be done in two ways. The first way is to expand the library directory to the primitives/logic directory and then scroll down to the XNOR item and select it.



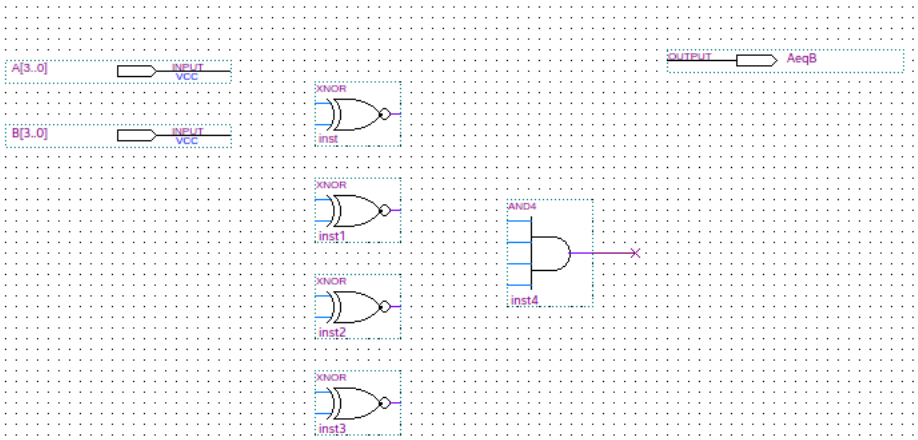
The second way is to type the symbol name directly into the Name box. You do not have to select the directory; the program will search the directory tree for the symbol.



After clicking on "OK" in the Symbol window, a floating image of an XNOR gate will appear in the schematic window. As you move the mouse, the symbol will follow. Position the symbol where you would like it to be and left-click the mouse. The symbol will now be placed onto the schematic. You need to AND together the outputs of the 4 XNOR gates. This can be done with one 4-input AND gate.



Before connecting the gates, you should enter the input and output ports by instantiating symbols named "input" and "output". This is done using the pin tool (to the right of the toolbar item shaped like an AND gate). We need 2 input ports named and 1 output port.

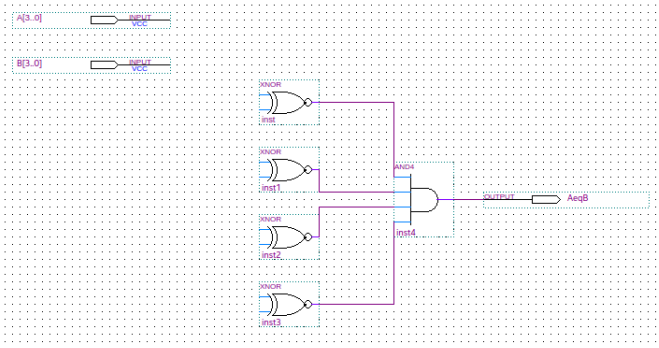


You now need to connect all of the gates together and hook them up to the input and output ports. There are two ways that you can connect nodes in the schematic:

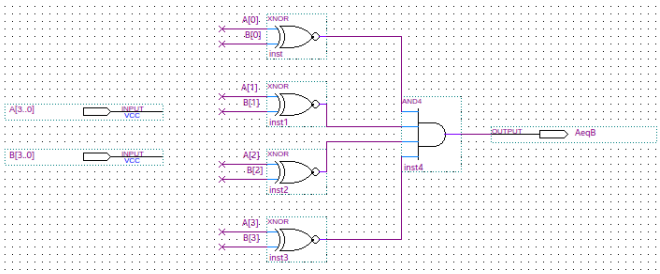
- Drawing wires between two nodes using the orthogonal node tool
- Giving two different nodes the same name or label

The first approach is best when making short connections between neighboring symbols, or when you want to make it immediately obvious from looking at the schematic that two nodes are connected. The second approach is the most convenient way of connecting busses (which are collections of individual nodes or wires). The software assumes that *two nodes with the same name are electrically connected.*

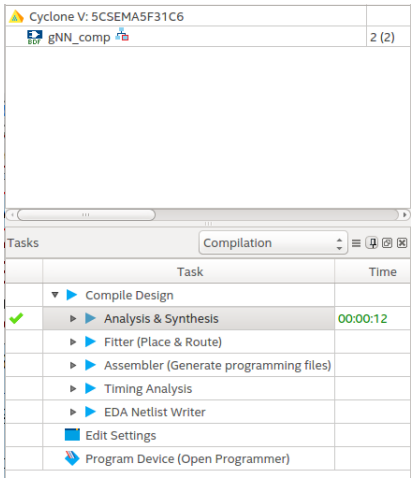
To draw wires between two nodes using the orthogonal node tool first select the tool from the toolbar to the top of the window. This tool allows to draw wires which run horizontally or vertically. When this tool is selected, the cursor changes into a crosshair. To draw a wire, position the cursor over the place where you want the wire to begin, and left-click the mouse. Then, while holding the mouse button down, drag the cursor to the place where you want the wire to end, then release. Using this approach, wire up the outputs of the XNOR gates to the inputs of the AND gate and similarly the output of the AND gate to the output pin $AeqB$. Note that if two wires cross each other, a connection between them will be made. In this case, a dot will appear indicating a connection between the two wires. If a mistake is made, you can simply select the wire by clicking on it and remove it by pressing the Delete key of your keyboard.



You can connect nodes without explicitly drawing in wires to connect them, by giving each node the same name. In order to use node naming as a connection technique, you must give names, or labels, to the nodes to be connected. Begin by naming the input busses. To do this, double-left-click on the left-most part of one of the input symbol. This should highlight the input name. Name the inputs as $A[3..0]$ and $B[3..0]$. Also name the output as $AeqB$. The name $A[3..0]$ means that A is a bus with 4 wires, having indices of $3, 2, \ldots, 0$. The node named $A[3]$ corresponds to the Most-Significant-Bit (MSB) of A, and $A[0]$ to the Least-Significant-Bit (LSB) of A. Add wires to the inputs of the XNOR gates. Use node naming (right click on the wire, and then on the "General" tab) to connect the input symbols to the XNOR gates.



You should now save the design, using the "Save As" item in the "File" menu. Call your file `firstname_lastname_comp.bdf`. Make sure that the "Add file to current project" box is selected. Once you have saved your file, you should compile it. Before compiling your file, you should first set your file as the top-level entity. In the Project Navigator window, change the Hierarchies tab to the Files tab. You should be able to see your file listed under the Files tab. Right click on your file and set it as the top-level entity by clicking on Set as Top Level-Entity. Now you can compile your file by double-clicking on the "Analysis & Synthesis" located on the Tasks window. Note that both the Tasks window and the Navigator window are on left side of the Quartus window. The analysis and synthesis process will check for syntax errors in your code. If the compilation is successful, a check mark will appear next to the "Analysis & Synthesis". In case of getting errors, you should go back to your design and fix the errors.



To get statistics on the resources required to implement your design, you can double click on "Fitter (Place & Route)". This process fits your design into the FPGA using its available resources and reports a summary of the resource utilization of your design on a window named "Compilation Report".

4.1 Simulation of the Circuit Using ModelSim

Once you have your circuit described using the schematic gate diagram, you should simulate it. The purpose of simulation is generally to determine:

1. If the circuit performs the desired function, and
2. if timing constraints are met.

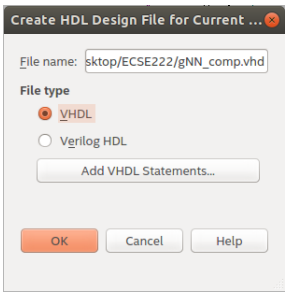
In the first case, we are only interested in the functionality of our implementation. We do not care about propagation delays and other timing issues. Because of this, we do not have to map our design to a target hardware. This type of simulation is called functional simulation.

The other form of simulation is called timing simulation. It requires that the design be mapped onto a target device, such as an FPGA. Based on the model of the device, the simulator can predict propagation delays, and provide a simulation that takes these into account. Thus, the timing simulation may produce results that are quite different from the purely functional simulation.

In this course, you will be using the ModelSim simulation software, created by the company Mentor Graphics

(actually you will use a version of it specific to Quartus, called Modelsim-Altera). The Modelsim software operates on an Hardware Description Language (HDL) description of the circuit to be simulated, written either in VHDL, Verilog, or System-Verilog. You will use VHDL.

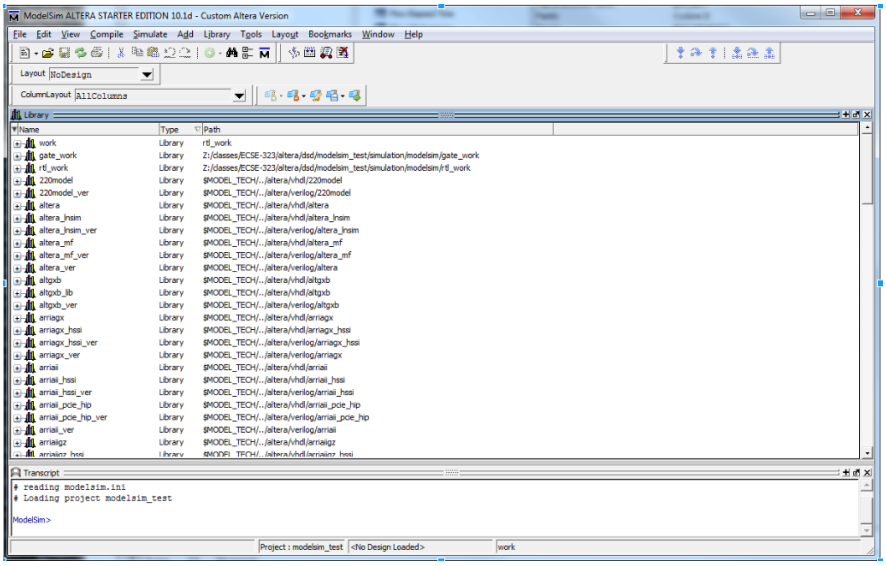
Modelsim does not understand schematic diagrams, so you need to convert your schematic diagram to an HDL description. You could just write the VHDL description directly, but you already have a schematic so it would be nice if you could use that. Fortunately, Quartus has the built-in capability of converting schematic diagrams to a VHDL description, so you can use that to convert your comparator schematic to VHDL. To convert this schematic to a VHDL description that the simulator can use, select "File">"Create/Update">"Create HDL Design File from Current File".



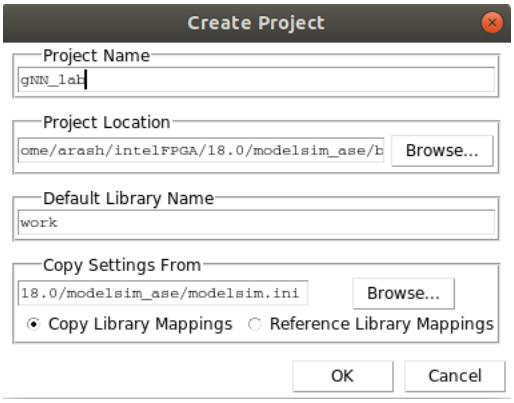
Upon clicking "OK," a VHDL description of your schematic file, named as `firstname_lastname_comp.vhd`, will be generated in your working directory. You can see the contents of this file using a text editor.

```
19  LIBRARY ieee;
20  USE ieee.std_logic_1164.all;
21
22  LIBRARY work;
23
24  ENTITY gNN_comp IS
25      PORT
26      (
27          A :  IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
28          B :  IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
29          AeqB :  OUT  STD_LOGIC
30      );
31  END gNN_comp;
32
33  ARCHITECTURE bdf_type OF gNN_comp IS
34
35  SIGNAL   SYNTHESIZED_WIRE_0 :  STD_LOGIC;
36  SIGNAL   SYNTHESIZED_WIRE_1 :  STD_LOGIC;
37  SIGNAL   SYNTHESIZED_WIRE_2 :  STD_LOGIC;
38  SIGNAL   SYNTHESIZED_WIRE_3 :  STD_LOGIC;
39
40
41  BEGIN
42
43
44
45  SYNTHESIZED_WIRE_0 <= NOT(A(0) XOR B(0));
46
47
48  SYNTHESIZED_WIRE_1 <= NOT(A(1) XOR B(1));
49
50
51  SYNTHESIZED_WIRE_2 <= NOT(A(2) XOR B(2));
52
53
54  SYNTHESIZED_WIRE_3 <= NOT(A(3) XOR B(3));
55
56
57  AeqB <= SYNTHESIZED_WIRE_0 AND SYNTHESIZED_WIRE_1 AND SYNTHESIZED_WIRE_2 AND SYNTHESIZED_WIRE_3;
58
59
60  END bdf_type;
```

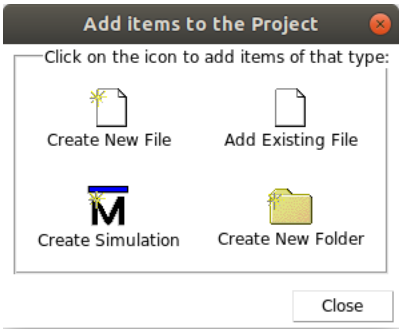
Now, we can start the simulation process. Double-click on the ModelSim desktop icon to run the ModelSim program. A window similar to the one shown below will appear.



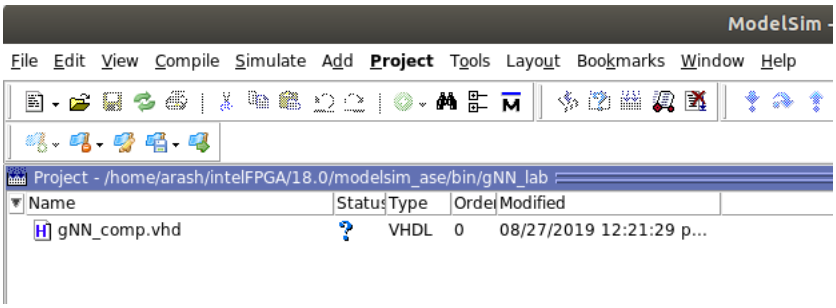
Select "FILE">"New">"Project" and, in the window that appears, give the project the name `firstname_lastname_lab2`.



Once you click OK, another dialog box will appear allowing you to add files to the project. Click on "Add Existing File" and select the VHDL file that was generated earlier (`firstname_lastname_comp.vhd`). You can also add files later.

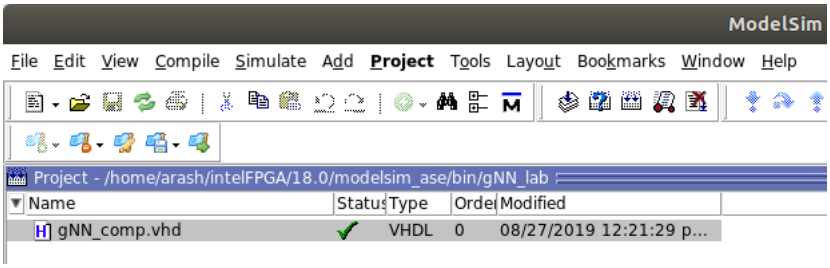


The ModelSim window will now show your VHDL file in the Project pane.

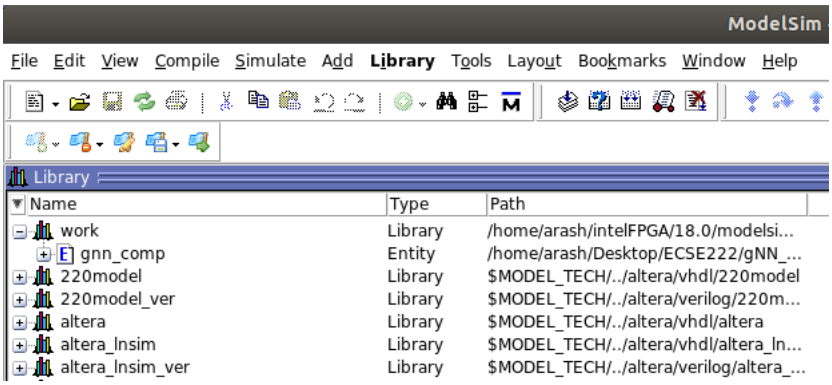


To simulate the design, ModelSim must analyze the VHDL files, a process known as compilation. The compiled files are stored in a library. By default, this is named "work". You can see this library in the "library" pane of the ModelSim window.

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 check marks, and a success message will appear in the **Transcript** pane (see figure below).



The compiled VHDL files will now appear in the library "work".



Since all of the inputs are undefined, if you ran the simulation now, the outputs would be undefined. So you need to have a means of setting the inputs to certain patterns, and of observing the outputs' responses to these inputs. In ModelSim, this is done by using a special VHDL entity called a Testbench. A testbench is special VHDL code that generates different inputs that will be applied to your circuit so that you can automate the simulation of your circuit and see how its outputs respond to different inputs. Note that the testbench *is only used in Modelsim for the purposes of simulating your circuit.* You will eventually synthesize your circuits into a real hardware chip called an FPGA. However, you will NOT synthesize the testbench into real hardware. Because of its special purpose (and that it will not be synthesized), the testbench entity is unique in that it has NO inputs or outputs, and uses some special statements that are only used in test benches. These special statements are not used when describing circuits that you will later synthesize to a FPGA. The testbench contains a single component instantiation statement that inserts the module to be tested (in this case the `firstname_lastname_comp` module), as well as some statements that describe how the test inputs are generated.

After you gain more experience you will be able to write VHDL testbenches from scratch. However, Quartus has a

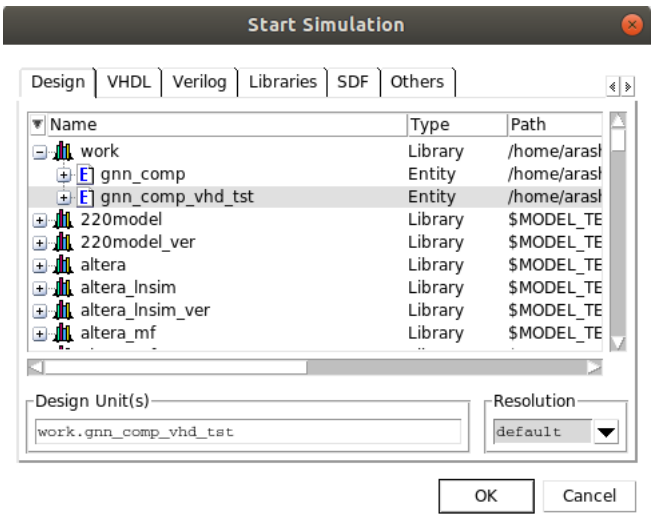
convenient built-in process, called the **Test Bench Writer**, which produces a VHDL template from your design that will get you started. To get the template, go back to the Quartus program, making sure that you have the `firstname_lastname_comp` project loaded. Then, in the Processing toolbar item, select "Start>Start Test Bench Template Writer". This will generate a VHDL file named `firstname_lastname_comp.vht` and place it in the simulation/modelsim directory.

Open the template in Quartus. Note that the template already includes the instantiation of the under test circuit (*i.e.*, `firstname_lastname_comp` component). It also includes the skeletons of two "process" blocks, one labeled "init" and the other labeled "always". The init process block can be deleted. You should edit the "always" process block to suit your needs, so in this case it will be used to generate the code signal waveform. You will notice that inside the process block, signal code are assigned multiple times! This should not make sense right now. If a signal was assigned multiple times using concurrent signal statements, this would be an error! However, the rules for statements inside a process block are different. Thankfully, it is not important to understand process blocks at this point. We will learn about them later on. The "wait for x ns" statement is a special VHDL statement that is only used in VHDL testbenches, and not in VHDL descriptions of synthesizable circuits that are intended to be implemented in real hardware. We never indicate time this way in synthesizable VHDL.

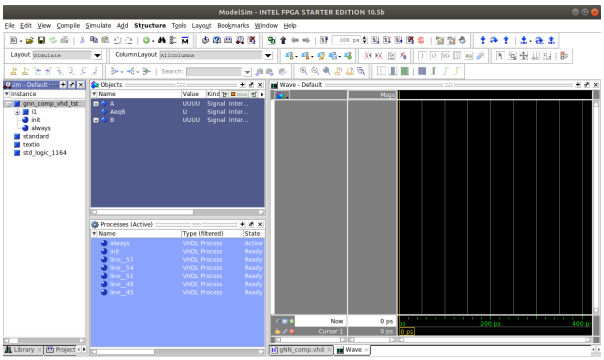
For a simple introductory example, replace the "always" process block with the following text:

```
always : PROCESS
BEGIN
    A <= "0000";
    B <= "0000";
    wait for 10 ns;
    A <= "1010";
    wait for 15 ns;
    B <= "1010";
    wait for 5 ns;
    A <= "1111";
    wait for 20 ns;
    B <= "1111";
    wait; -- this waits forever...
    END PROCESS always;
```

Once you have finished editing the testbench file, you need to add it to the project in ModelSim by selecting **Project** > **Add to Project** > **Existing File...**. Then you should select the testbench file in the **Project** pane, and click on **Compile Selected** from the **Compile** toolbar item. This will compile the testbench file. Now everything is ready for you to actually run a simulation! Select "Start Simulation" from the **Simulate** toolbar item in the ModelSim program. The window shown below will appear.



Select the `firstname_lastname_comp_vhd_tst` entity and click on OK. The ModelSim window should now look like the figure below. If you dont see the "Wave" window, you can enable it by clicking on "View > Wave".



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 the signals. Now, to actually run the simulation, click on the "Run all" icon in the toolbar. If you get an incorrect output waveform, you will have to go back and look at your design. If you make a correction to your VHDL code, you will have to re-run the compilation of the changed files in ModelSim. Finally, to rerun the simulation, first click on the "Restart" button, then click on the "Run all" button.

The simulation you ran in the previous part of the lab just had a couple of input signal transitions, and did not test all possible input patterns. There are 2^8 or 256 possible patterns in the `gNN_comp` circuit, so complete testing of the circuit will require you to simulate all of these patterns. In order to run through all possible 256 cases, we use a FOR LOOP that increments the value of code signal in the loop. This is done in the testbench shown below.

```
generate_test : PROCESS
BEGIN
    FOR i IN 0 to 16 LOOP --   loop over all A values
    A <= std_logic_vector(to_unsigned(i,4)); --   convert the loop variable i to std_logic_vector
    FOR j IN 0 to 16 LOOP --   loop over all B values
        B <= std_logic_vector(to_unsigned(j,4)); --   convert the loop variable i to
    std_logic_vector
        WAIT FOR 10 ns; -- suspend process for 10 nanoseconds at the start of each loop
    END LOOP; -- end the j loop
    END LOOP; -- end the i loop
    WAIT; -- we have gone through all possible input patterns, so suspend simulator forever
END PROCESS generate_test;
```

The process block generates all of the possible input patterns using a FOR loop. The RANGE attribute is equivalent to specifying the loop range as over the minimum value of the signal to its maximum value. Replace the "always" process block to perform the complete test. Note the convertor function of "to_unsigned" only works when its library (*i.e.*, ieee.numeric_std.all) is called in the testbench.

Modify your testbench so that it performs the exhaustive testing. In the ModelSim program recompile the testbench and then restart and re-run the simulation. Look at the simulated cases and check to see if they are correct.

5 Concurrent Statements in VHDL

Combinational circuits can be described in VHDL using "Concurrent statements". There are different types of concurrent statements in VHDL.

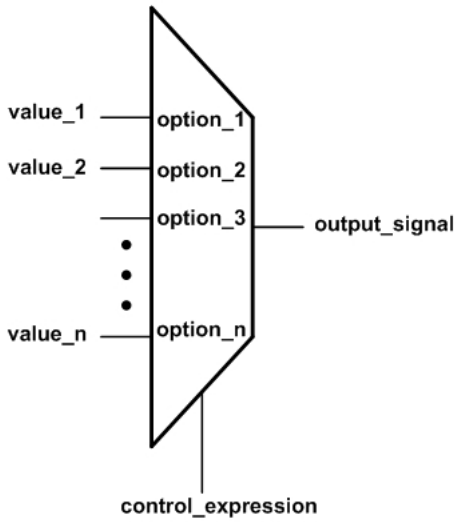
5.1 Direct Assignment

The simplest form of concurrent statements is the direct assignment. Below is an example of direct assignment which performs XOR operation between two variables:

```
c <= a XOR b;
```

5.2 Selected Signal Assignment or the "With/Select" Statement

Consider an n-to-1 multiplexer shown below. This block selects one of its n inputs and transfers the value of this input to the output terminal, *i.e.*, *output_signal*.



The selected signal assignment allows to implement the functionality of a multiplexer:

```
with control_expression select
    output_signal <=  value_1 when  option_1,
                      value_2 when  option_2,
                      ...
                      value_n when  option_n;
```

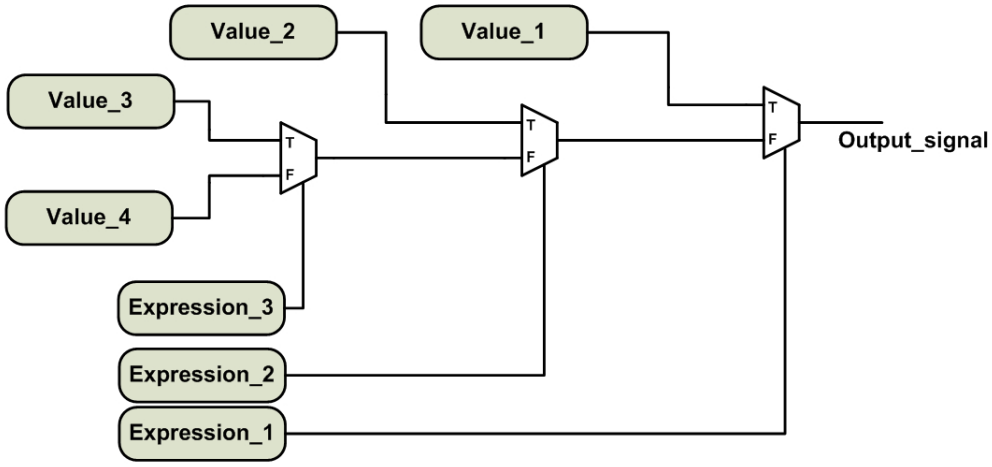
Here, the value of the *control_expression* will be compared with the n possible options, *i.e.*, *option_1*, *option_2*, ..., *option_n*. When a match is found, the value corresponding to that particular option will be assigned to the output signal, *i.e.*, *output_signal*. For example, if *control_expression* is the same as *option_2*, then *value_2* will be assigned to the *output_signal*. Note that the options of a "with/select" assignment must be mutually exclusive, *i.e.*, one option cannot be used more than once. Moreover, all possible values of the *control_expression* must be included in the set of options.

5.3 Conditional Signal Assignment or the "When/Else" Statement

The "when/else" statement is another way to describe concurrent signal assignments. In general, the syntax of the "when/else" statement is:

```
output_signal <=  value_1 when  expression_1 else
                  value_2 when  expression_2 else
                  ...
                  value_n;
```

In this case, the expressions after "when" are evaluated successively until a true expression is found. The assignment corresponding to this true expression will be performed. If none of these expressions are true, the last assignment will be executed. We should emphasize that the expressions after the "when" clauses are evaluated successively. As a result, expressions evaluated earlier have higher priority as compared to later expressions. Considering this, we can obtain the conceptual diagram of this assignment as shown below. This figure illustrates a conditional signal assignment with three "when" clauses.



5.4 Concurrent Statements at a Glance

Concurrent statements are executed at the same time and there is no significance to the order of these statements. This type of code is quite different from what we have learned in basic computer programming where the lines of code are executed one after the other.

The selected signal assignment or the "with/select" assignment allows us to implement the functionality of a multiplexer.

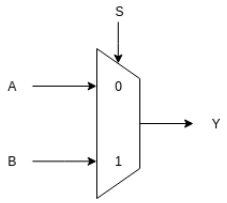
The options of a “with/select” assignment must be mutually exclusive, *i.e.*, one option cannot be used more than once. Moreover, all possible values of control_expression must be included in the set of options.

For the "when/else" statement, the expressions following the “when” clauses are evaluated successively. As a result, the expressions evaluated earlier have a higher priority as compared to the following expressions.

One important difference between the “with/select” and “when/else” assignment can be seen by comparing the conceptual implementation of these two statements. The "when/else" statement has a priority network; however, the “with/select” assignment avoids this chain structure and has a balanced structure.

6 VHDL Implementation of a 2-to-1 MUX

A multiplexer is a circuit that selects between several inputs and forwards it to a single output. In general, a 2^n-to-1 MUX has 2^n inputs with n selectors. A schematic diagram of a 2-to-1 multiplexer is given below.



According to the above schematic, the 2-to-1 multiplexer outputs the input signal A when the selector signal S is equal to 0 otherwise it outputs the input signal B. In this lab, you will implement this 2-to-1 multiplexer in VHDL using structural and behavioral styles. In the structural modeling of the 2-to-1 multiplexer in VHDL, the multiplexer is implemented using AND, OR or NOT gates only. More specifically, the structural description of the multiplexer literally realizes its boolean function. Describe the boolean function in VHDL using AND, OR or NOT gates only. Use the following entity declaration for your VHDL description of the 2-to-1 multiplexer:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity firstname_lastname_MUX_structural is
    Port (A          : in  std_logic;
```

```
          B          : in  std_logic;
          S          : in  std_logic;
          Y          : out std_logic);
end firstname_lastname_MUX_structural;
```

Make sure to replace `firstname_lastname` with your full name. Once completed, you will describe the architecture of the 2-to-1 multiplexer using behavioral style. In the behavioral description, you desrcribe the behavior of your target circuit and the synthesis tool creates a gate-level layout of your design. Use a single VHDL select assignment only and the entity declaration below to implement a behavioral description of the 2-to-1 multiplexer.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity firstname_lastname_MUX_behavioral is
    Port (A          : in  std_logic;
          B          : in  std_logic;
          S          : in  std_logic;
          Y          : out std_logic);
end firstname_lastname_MUX_behavioral;
```

Once described both behavioral and structural styles of the 2-to-1 multiplexer in VHDL, you are required to test your circuits. Write a testbench code and perform an exhaustive test for your VHDL descriptions of the 2-to-1 multiplexer.

7 Questions

1. Explain your VHDL code.
2. Report the number of pins and logic modules used to fit your designs on the FPGA board. These results can be obtained from the flow summary tab in the table of contents menu in Quartus.

	AeqB	2-to-1 MUX	
	Schematic	Structural	Behavioral
Logic Utilization (in ALMs)			
Total pins			

3. Show a representative simulation plot for the introductory testing example. You can simply include a snapshot from the waveform that you obtained from ModelSim. In order to fully capture all the signals from the waveform, you can adjust the display range using the magnifier icons.
4. Show representative simulation plots for the exhaustive test.
5. Show representative simulation plots of the 2-to-1 MUX circuits for all the possible input values.

8 Deliverables

You are required to submit the following deliverables on MyCourses. Please note that a single submission is required per group (by one of the group members).

- Lab report. The report should include the following parts: (1) Names and McGill IDs of group members, (2) an executive summary (short description of what you have done in this VHDL assignment), (3) answers to all questions in previous section (if applicable), (4) legible figures (screenshots) of schematics and simulation results, where all inputs, outputs, signals, and axes are marked and visible, (5) an explanation of the results obtained in the assignments (mark important points on the simulation plots), and (6) conclusions. Note - students are encouraged to take the reports seriously, points will be deducted for sloppy submissions.
- Project files. Create a single .zip file named `vhdl#_firstname_lastname` (replace `#` with the number of the current VHDL assignment and `firstname_lastname` with the name of the submitting group member). The .zip file should include the working directory of the project.