DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

CHITTAGONG UNIVERSITY OF ENGINEERING AND TECHNOLOGY

CHATTOGRAM – 4349, BANGLADESH

Experiment No. 9

Introduction to Verilog HDL and Quartus II

PRECAUTIONS:

- Students must carefully read the lab manual before coming to lab to avoid any inconveniences.
- Students must carry a flash drive to transfer files and lab manuals.
- Use of mobile phone in lab is strictly prohibited and punishable offence.
- Experiment files must be uploaded to GitHub including home tasks.

OBJECTIVES:

- To familiarize with Quartus II.
- To familiarize with Hardware Description Language (HDL).
- To understand behavioral and structural Verilog descriptions.

INTRODUCTION TO HDL:

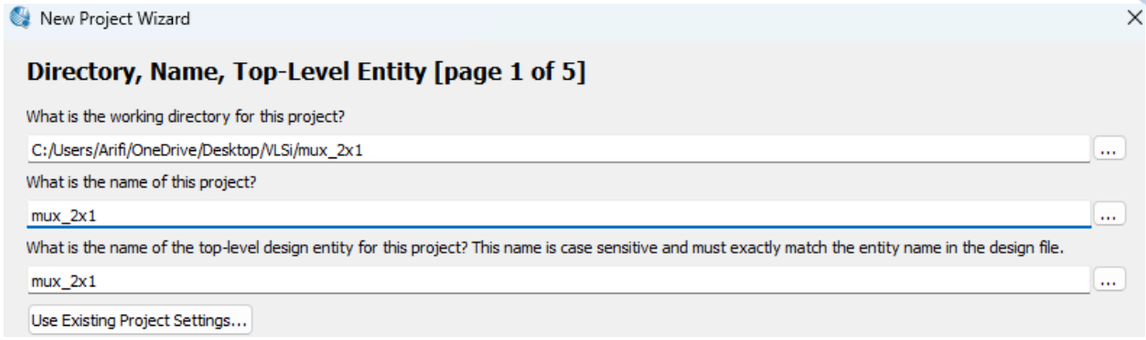
A hardware description language (HDL) may look similar to a typical computer programming language except, it describes hardware rather than code that can be executed by hardware. HDL is a specialized computer language used to describe the structure and behavior of electronic circuits, most commonly, digital logic circuits.

A hardware description language enables a precise, formal description of an electronic circuit that allows for the automated analysis and simulation of an electronics circuit. It also allows for the synthesis of an HDL description into a netlist (a specification of physical electronics components and how they are connected together), which can then be placed and routed to produce the set of masks used to create an integrated circuit. Two IEEE standard HDLs are: Verilog HDL and VHDL (Very High Speed Integrated Circuit Hardware Description Language).

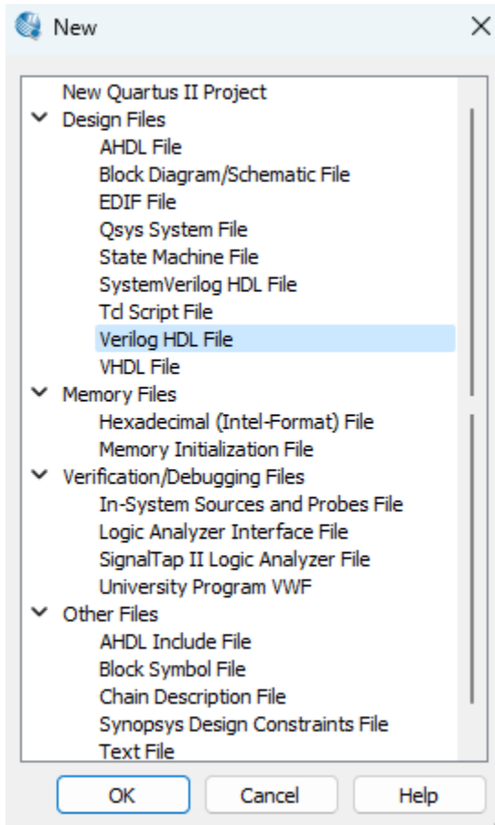
INTRODUCTION TO QUARTUS PRIME:

1. Start **Quartus II** from the start menu by searching **"quartus"**. **Quartus II** should start up. Each logic circuit or sub-circuit being designed with **Quartus** is called a project. The software works on one project at a time and keeps all the information for that project in a single directory own as its working directory. To begin a new logic circuit design, the first step is to create a directory to hold its files. Create a directory in any drive and give it a name. Then execute **File > New Project Wizard.**

2. In the new window, click **Next**. Then change the working directory to your preferred directory which you created. Also give your project a name. Note that the project must have a name, which is usually the same as the top-level design entity that will be included in the project. Click **Next.**



3. In this window we can add any existing files to our project, which we will skip this for now. Next window is the family, device and board settings. As we will not be using an FPGA at the moment, can select **"Cyclone IV GX"** from device family and click **Next**. Now in the EDA tool settings change simulation tool to **"ModelSim-Altera"** and format to **"Verilog HDL"**. Click **Next** then **Finish.**
4. Execute **File > New**. Then choose **"Verilog HDL File"** and click **OK**.



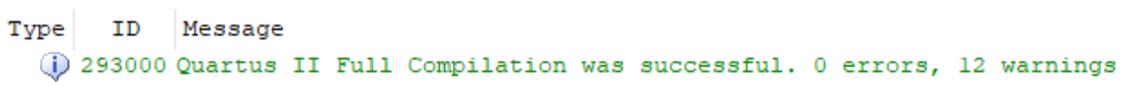
5. A text editor will open up where we can write our HDL code and save it as the same name as our project.
6. Write a Verilog code into the text editor. For example, HDL code for a 2x1 MUX is given below:

```
1  module mux_2x1 (A, B, S, Y);
2  input A, B, S;
3  output Y;
4
5  assign Y = (~S & A) | (S & B);
6  endmodule
```

Press Ctrl + S to save the file and name it the same as your project name. The syntax of Verilog could sometimes be difficult to remember. To help with this issue, you can use templates for various designs. You can browse through these or use them using **Edit > Insert Template > Verilog HDL**.

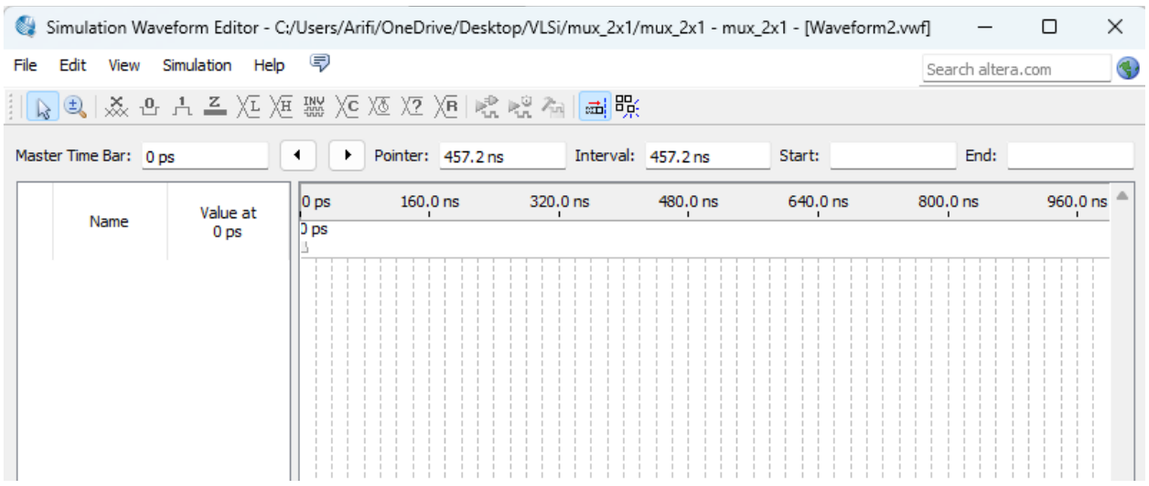
7. Click on the pruple play button to start the compilation of your Verilog code.

8. In the **Messages** window at the bottom of the screen, you should find compilation was successful with 0 errors and some warnings. You may neglect the warning for now but it must not contain any errors.

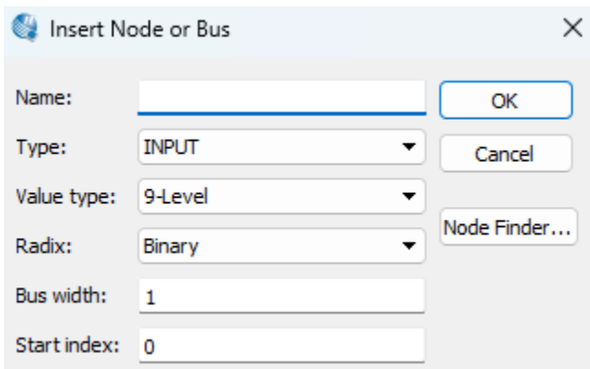


If the compiler does not report zero errors, then there is at least one mistake in the Verilog code. In this case, a message corresponding to each error found will be displayed in the **Messages** window. Double-clicking on an error message will highlight the statement which is affected by the error, in the Verilog code in the Text Editor window. The user can obtain more information about a specific error by selecting the error and pressing F1 function key. Correct the error and recompile the design.

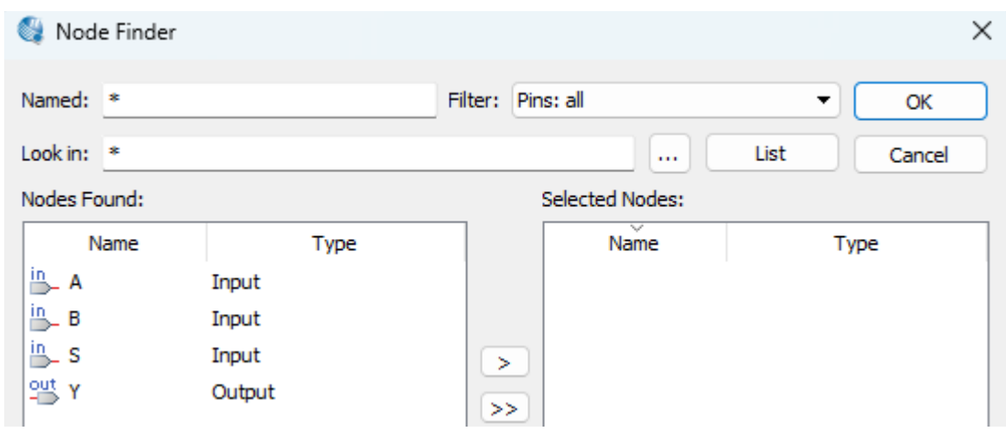
9. Now, execute **File > New** to create a vector waveform file which is required for simulating inputs and outputs. Select **"University Program VWF"** from the list and click **OK**. The following window should pop up:



10. Double-click on the white space under **"Name | Value At".** Or right-click on that space and select **Insert Node or BUS.**

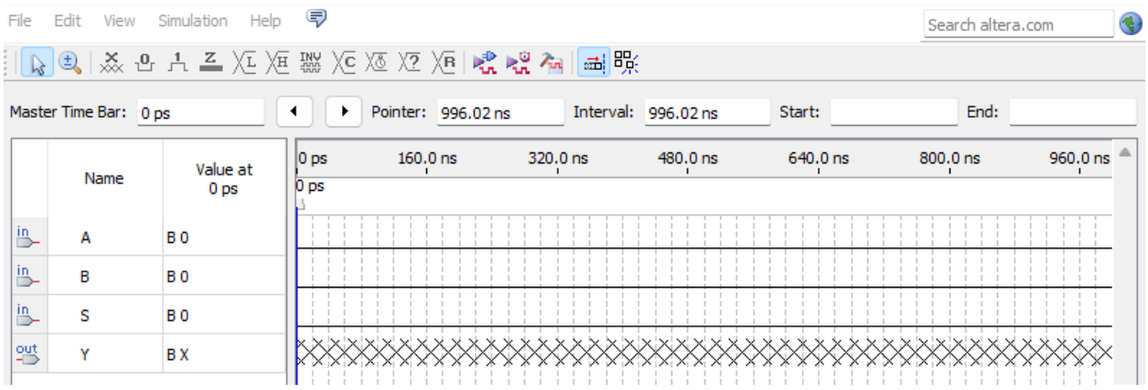


11. In the **"Insert Node or BUS"** window, click **Node Finder.** In the **Node Finder** window, select **Filter** to be **"Pins: all"**, then click the **"List"** button**.**



12. Now highlight all the nodes and click the >> button to select all of them. Now click **OK.** In the following window, click **OK** as well.

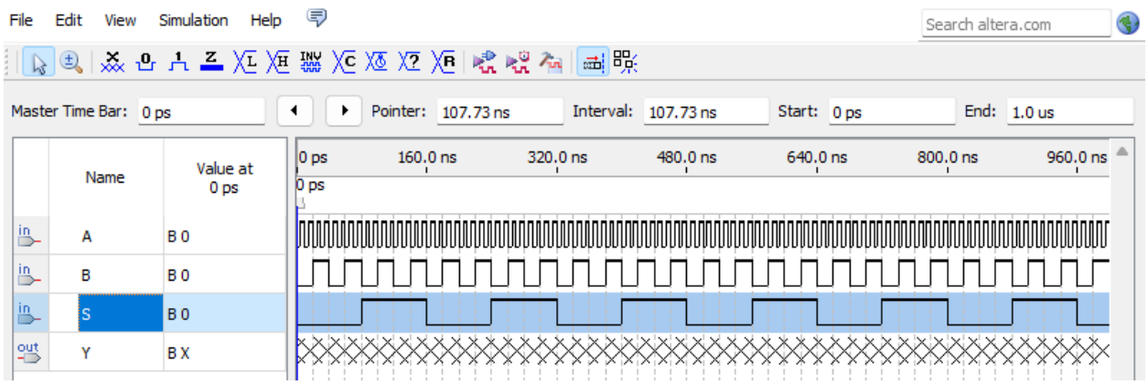
13. The vector waveform window should look like the following now:



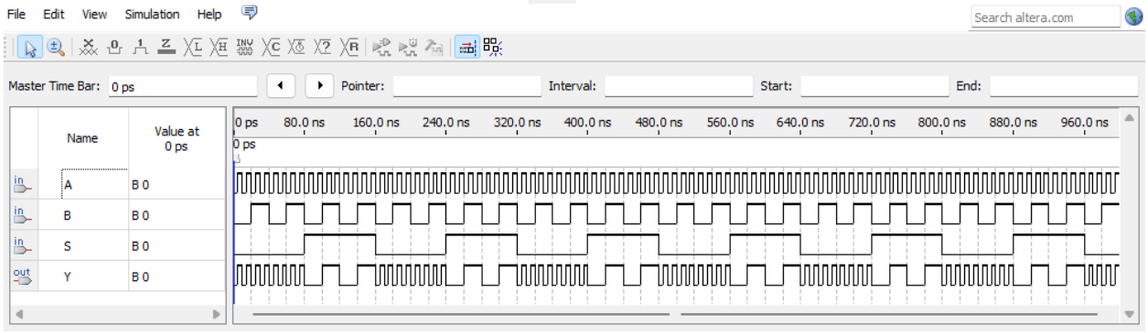
Now, we can clearly see all the inputs and outputs.

14. Select **"A"** from the left palette, click on the **"Overwrite Clock"** icon. In the **"Clock"** window, set the **Period** and **Duty Cycle** of the clock just as you did in previous experiments. Similarly set the timing for **"B"** and **"S"** as well.

15. After setting all the input clocks, it should look like the following figure. Note that, the output **"Y"** is displayed as having an unknown value at the time, which is indicate by a hashed pattern; its value will be determined during simulation.

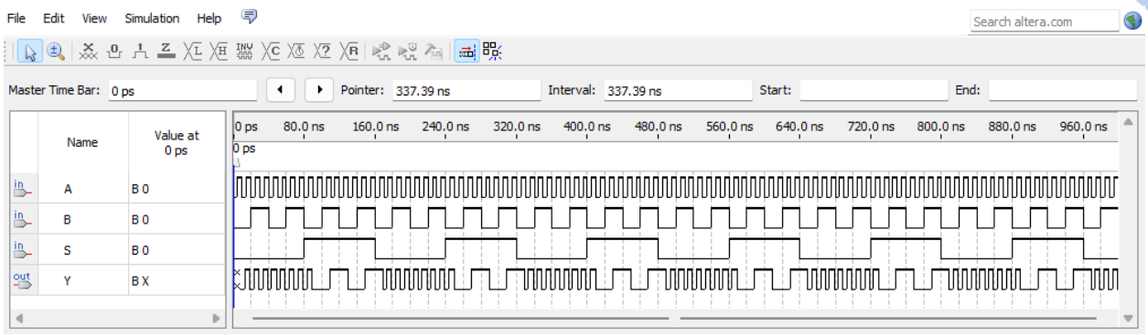


16. Save the **VWF** file by pressing Ctrl + S, and keep the default name.
17. A designed circuit can be simulated in two ways. The simplest way is to assume that there is no delay in propagation of signals through the circuit. This is called functional simulation. A more complex way is to take all propagation delays into account, which leads to timing simulation. Typically, functional simulation is used to verify the functional correctness of a circuit as it is being designed. This takes much less time, because the simulation can be performed simply by using the logic expressions that define the circuit.
Click on the **"Run Functional Simulation"** button to verify the results.



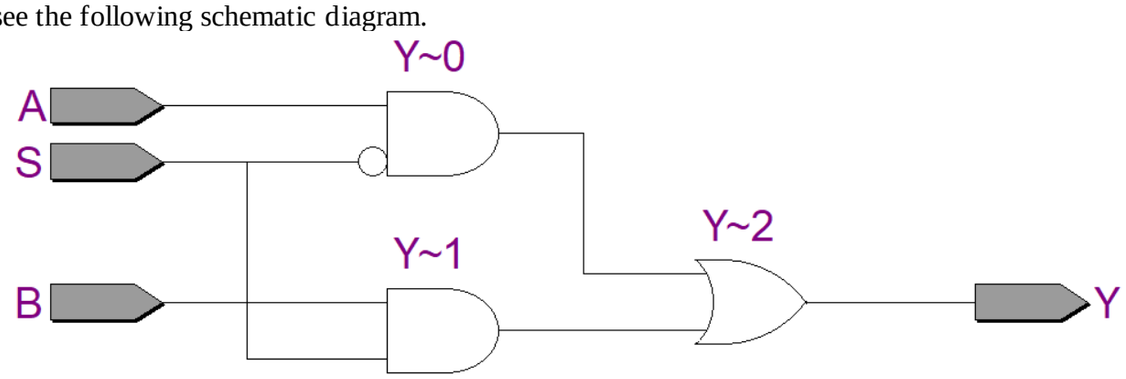
Notice that there is no gate switching delay as it is only a functional simulation.

18. Now click on the **"Run Timing Simulation"** button to verify the results.



You can check that there are obvious delays present in the output signal.

19. Now go back to your code, and execute **Tools > Netlist Viewers > RTL Viewer.** You should see the following schematic diagram.



Generate the following circuits in a similar manner:

Half-Adder:

```
module half_adder(A,B,Cout,S);
input A,B;
output S, Cout;
assign S = A^B;
assign Cout = A&B;
endmodule
```

2X1 MUX:

```
module mux21_fn(I0,I1,S,f);
input I0,I1,S;
output reg f;
always@(I0,I1,S)
begin
if (S == 0)
f = I0;
else
f = I1;
end
endmodule
```

Full Adder Using Half Adders:

```
module FA_001(A,B,C,Cout,S);
input A,B,C;
output Cout,S;
wire C1,C2,S1;
HA_001 f1(A, B, C1, S1);
HA_001 f2(S1, C, C2, S);
assign Cout = C1|C2;
endmodule
```

```
module HA_001(a,b,c,s);
input a,b;
output c,s;
assign c = a&b;
assign s = a^b;
endmodule
```

2X4 Decoder:

```
module dec2to4(W, En, Y);
input [1:0]W;
input En;
output reg [0:3]Y;
integer k;
always@(W, En)
for(k = 0; k < = 3; k = k+1)
if ((W == k) && (En == 1))
Y[k] = 1;
else
Y[k] = 0;
endmodule
```

Rubrics:

Criteria	Below Average (1)	Average (2)	Good (3)
Formatting	Report is not properly formatted, missing objectives, discussions and references.	Report is somewhat formatted but missing proper references.	Report is properly formatted.
Code	HDL codes are properly written.	HDL codes are properly written and simulation results are accurate.	
Writing	Writing is poor and not informative. It does not address the design decisions and contains high plagiarism.	Writing is average and original. Design decisions are somewhat explored.	Writing is excellent with every design decision adequately explored.
Diagram	Diagrams are of bad quality and unreadable.	Diagrams are clear and of high quality.	