Concepts in VLSI Design Laboratory

IT126IU

Instructor: Nguyen Toan Van

Lab 7:
Introduction to Verilog HDL and Quartus II

Objectives:

- To get familiar with Quartus II
- To get introduced to Hardware Description Language (HDL)
- To understand behavioral and structural Verilog descriptions

Verilog HDL

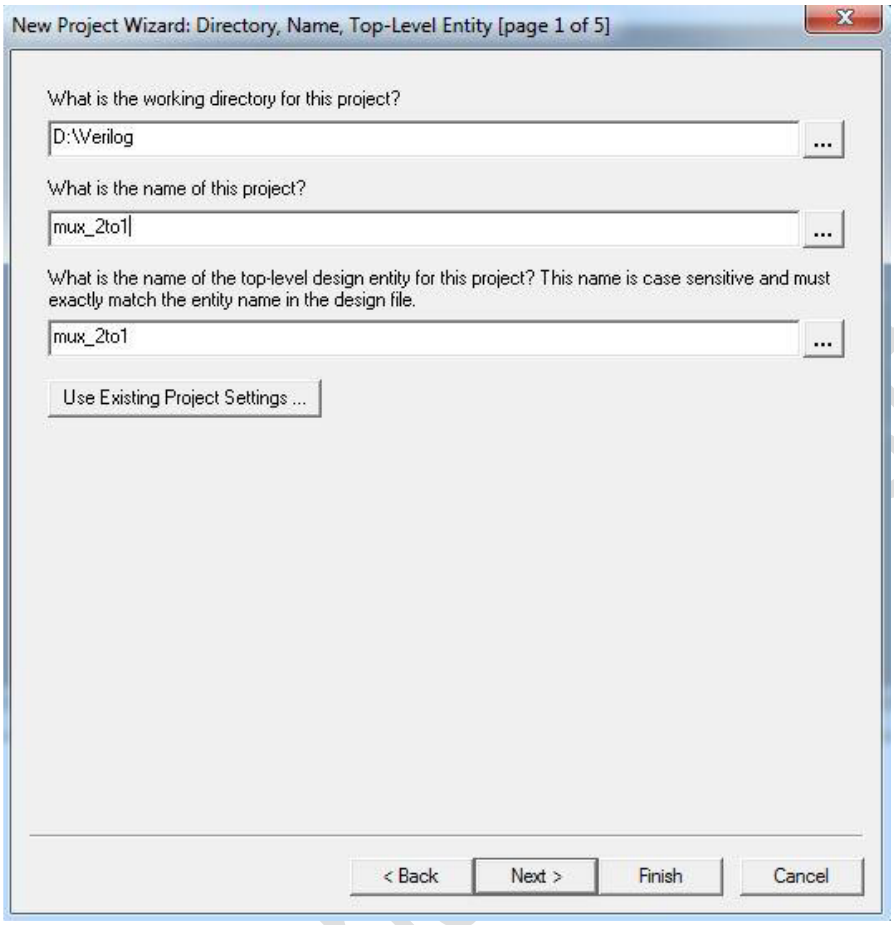
A hardware description language (HDL) is similar to a typical computer programming language except that an HDL is used to describe hardware rather than a program to be executed on a computer. Two HDLs are IEEE standards: Verilog HDL and VHDL (Very High Speed Integrated Circuit Hardware Description Language).

QUESTIONS:
What is hierarchical design? What design principles will make hierarchical layout design easier?

Download *Quartus II 13.1* at https://www.intel.com/content/www/us/en/software-kit/666221/intel-quartus-ii-web-edition-design-software-version-13-1-for-windows.html
Video: https://www.youtube.com/watch?v=rhD6_hx6DZk

Creating a New Project

1. Run Quartus II by clicking on the shortcut icon on desktop named *Quartus II Web Edition.* A getting started window may open up. You may put a tick on '**Don't show this screen again**', if you do not want this to appear again.
2. Each logic circuit or sub-circuit being designed with Quartus II software is called a *project*. The software works on one project at a time and keeps all the information for that project in s single directory (folder) in the file system. To begin a new logic circuit design, the first step is to create a directory to hold its files. Create a folder in *D:\, E:\ or F:* drive named after your ***student ID***. Execute ***File*** → ***New Project Wizard***. Click ***Next*** on the window that opens up.
3. In the following window, change the working directory of the project to your directory (e.g. D:\120105001) and give a name to the project as shown. 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***.

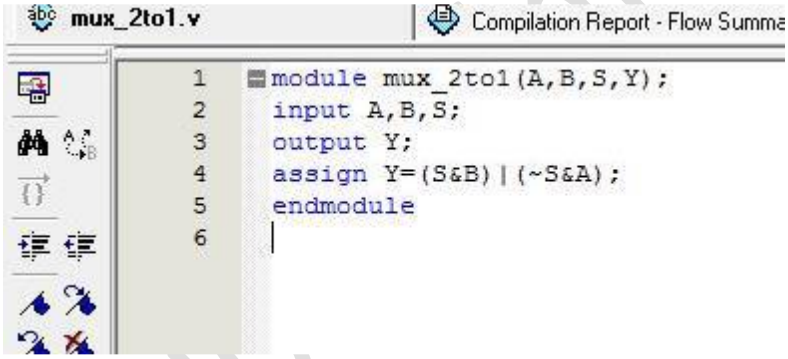


4. We can specify any existing file to be included in the project, if we want, in the next window. But we will skip it for now. Click ***Next*** on the next three windows that open up. Then in the last window, click ***Finish***.

Creating a new Verilog HDL file under a project (Design Entry), Compilation and Simulation

1. Execute ***File→New***. A window will open up. Select ***Verilog HDL File*** and click ***OK***.
2. The following Text Editor Window will open up and you can write your Verilog code in here and **save it using the same name as the project**.

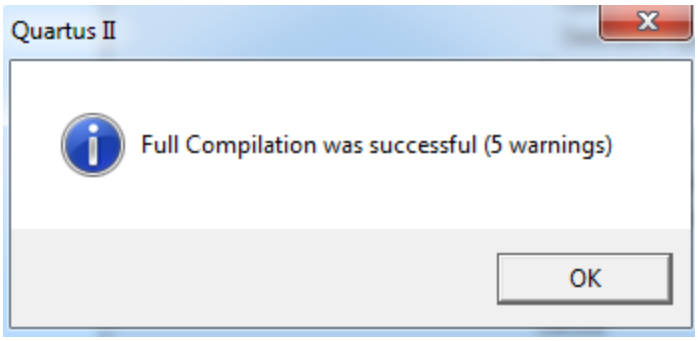
3. Write your Verilog code. This example shows Verilog code for a 2to1 MUX.



The syntax of Verilog code is sometimes difficult for a designer to remember. To help with the issue, the Text Editor provides a collection of Verilog templates. The templates provides examples of various types of Verilog statements, such as a module declaration, an always block, and assignment statements. It is worthwhile to browse through the templates by selecting ***Edit→Insert Template→Verilog HDL*** to become familiar with this resource.

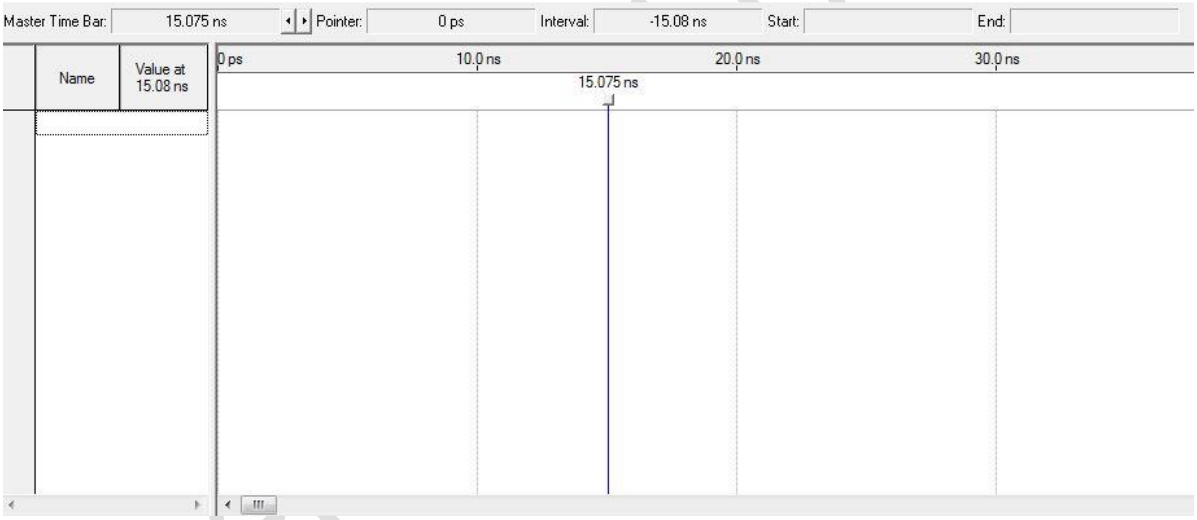
4. Click on the purple play button to start compilation of your Verilog code.

5. After successful compilation, you will get the following message. Ignore the Warnings for now. Click ***OK***.

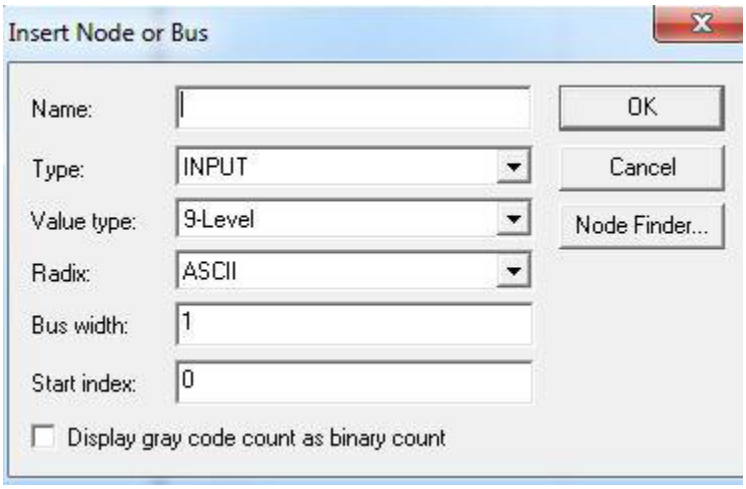


If the compiler does not report zero errors, then there is at least one mistake in Verilog code. In this case, a message corresponding to each error found will be displayed in the **Messages** window. Double-clicking on an error message will highlight the statement which is affected by the error, in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

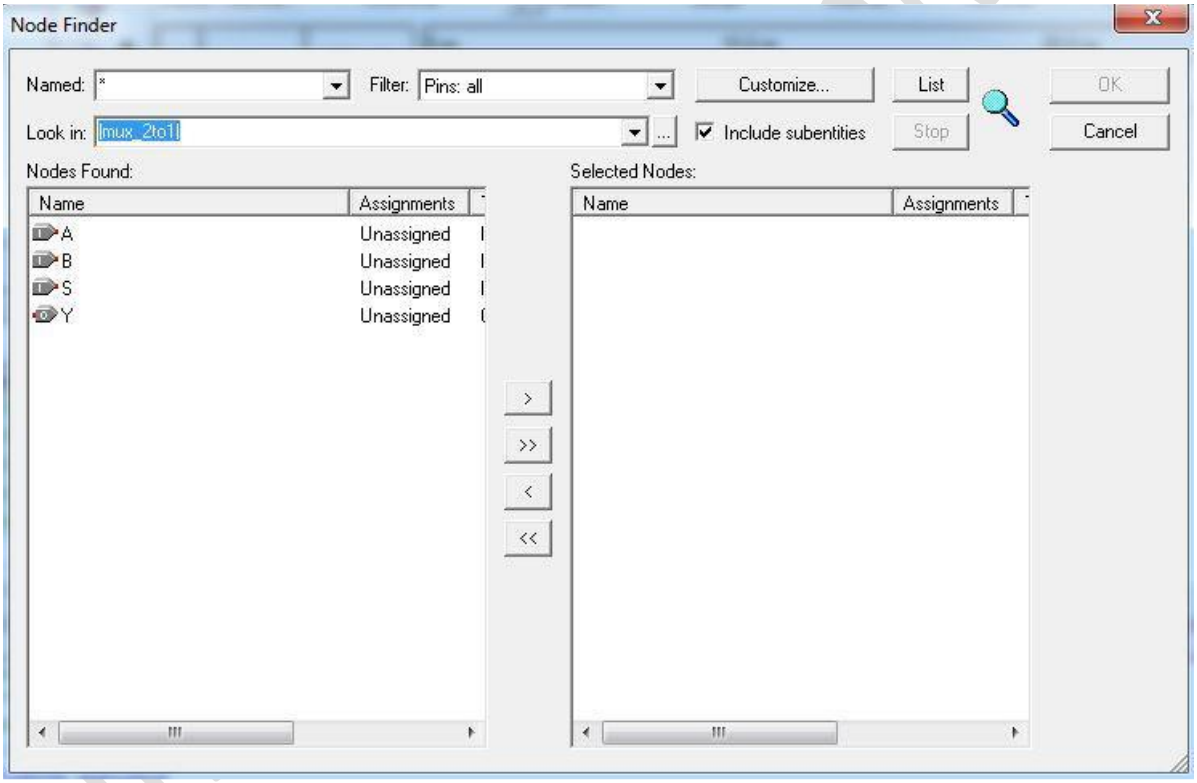
6. Now, execute ***File→New*** to create a vector waveform file which is required for simulating inputs and outputs. Select ***Vector Waveform File*** from the list and click ***OK***.
7. The following window will open up.



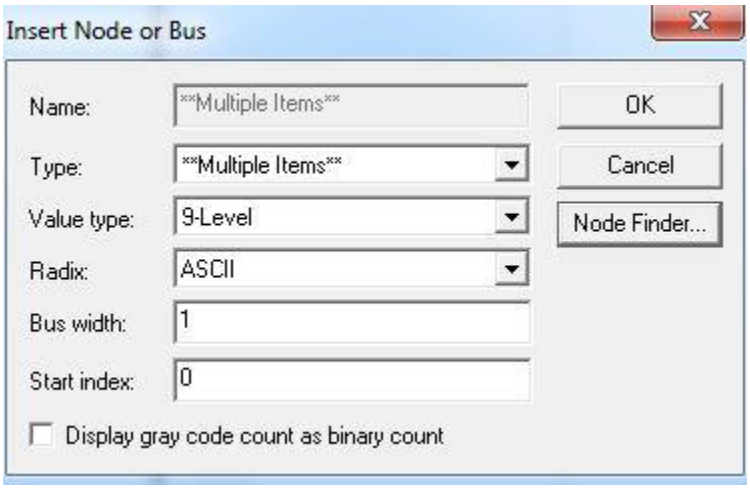
8. Double-click on the white space under '**Name|Value at 15.08 ns'.** Or Right-click on that space and select ***Insert→Insert Node or Bus***.
9. In the '**Insert Node or Bus**' window, click ***Node Finder***.



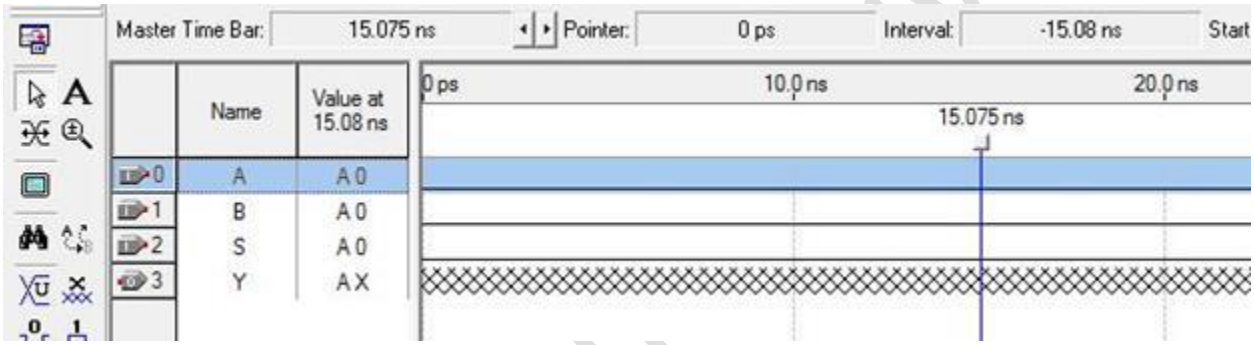
10. In the '**Node Finder**' window, Click ***List***. Make sure '**Pins:all**' is selected under '**Filter**'.
11. Now the window will look like the following one. Click on the '**>>**' Button.



12. Now, it should appear like the following. Click ***OK***.

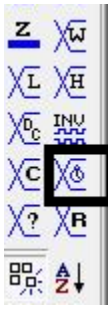


14. The vector waveform file will now look like the following:

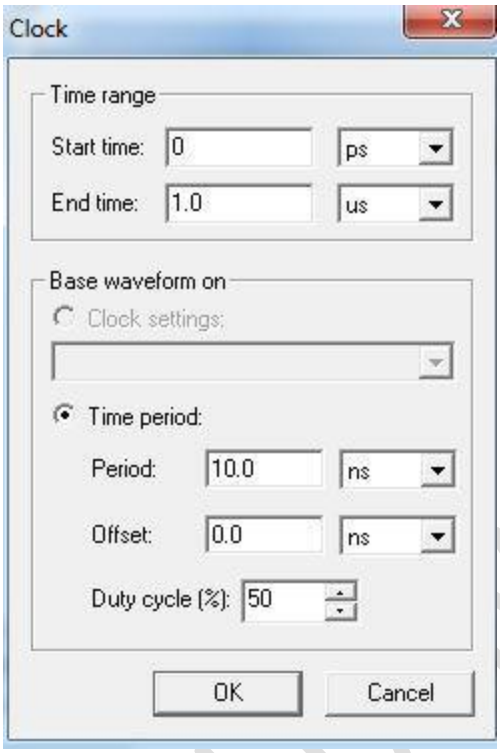


Now, you can clearly see the inputs and outputs.

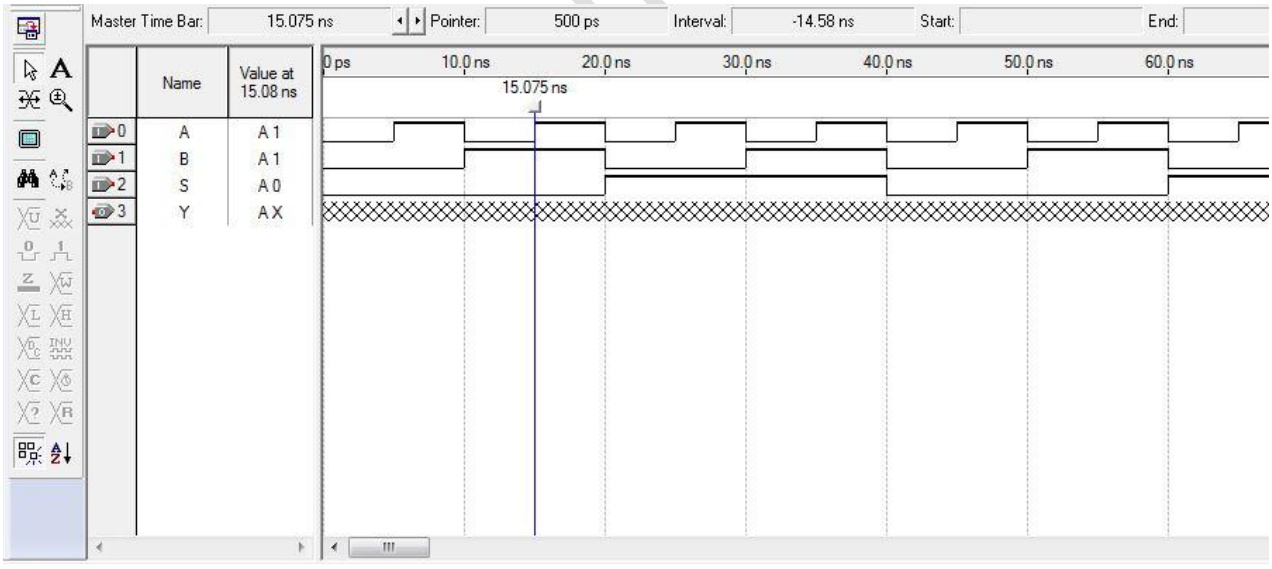
15. Select an input and from the left palette, click on the '**Overwrite clock**' icon.



16. In the '**Clock**' window, set parameters of the clock.



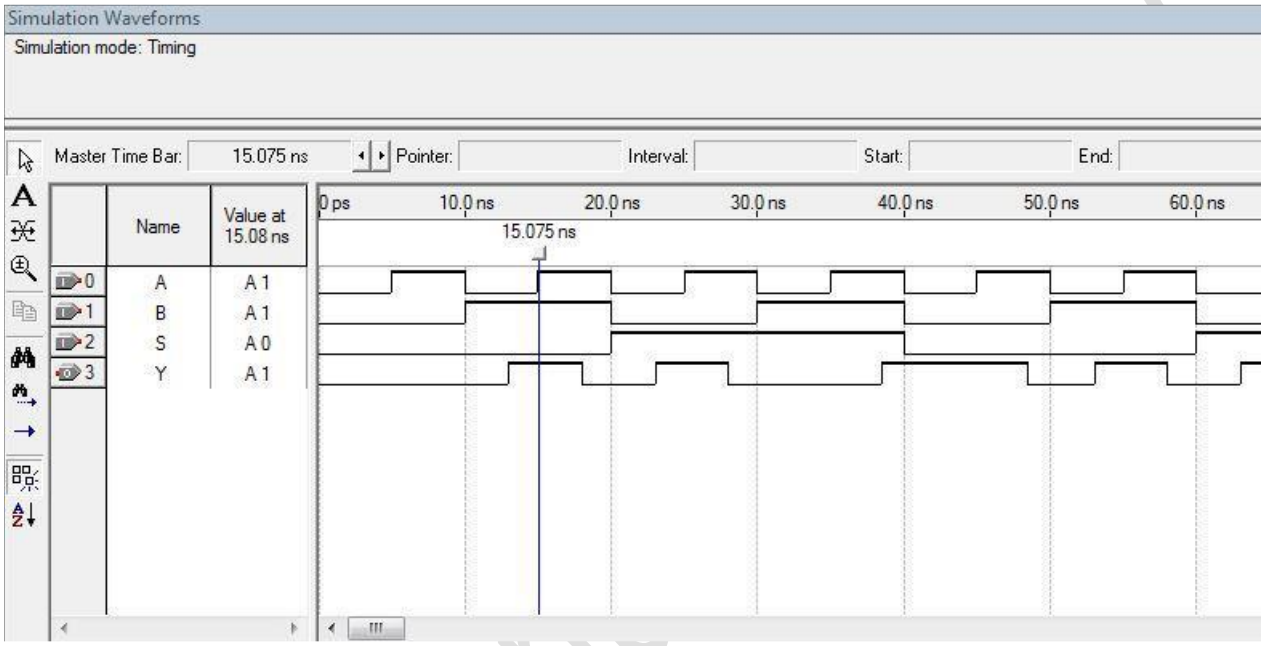
17. Now, after setting all the input clocks, **Vector Waveform File** will look like the following. Note that, the output Y is displayed as having an unknown value at that time, which is indicated by a hashed pattern; its value will be determined during simulation.



18. Save the **Vector Waveform File**. **It must have the same name as the Verilog file**.

19. 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 blue play button and you should observe the simulated waveforms.

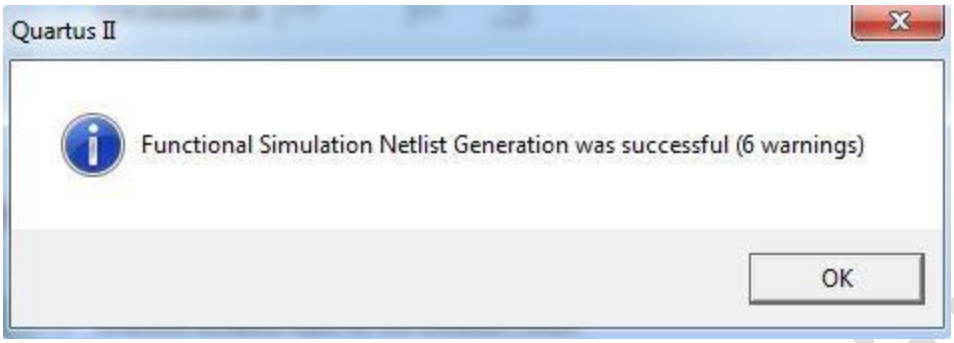


Note there is a delay between input and outputs which simulates the real effect of gate delays. If you are interested in only functional analysis rather than timing analysis, go through some more steps.

20. Execute ***Processing→Simulator Tool***.

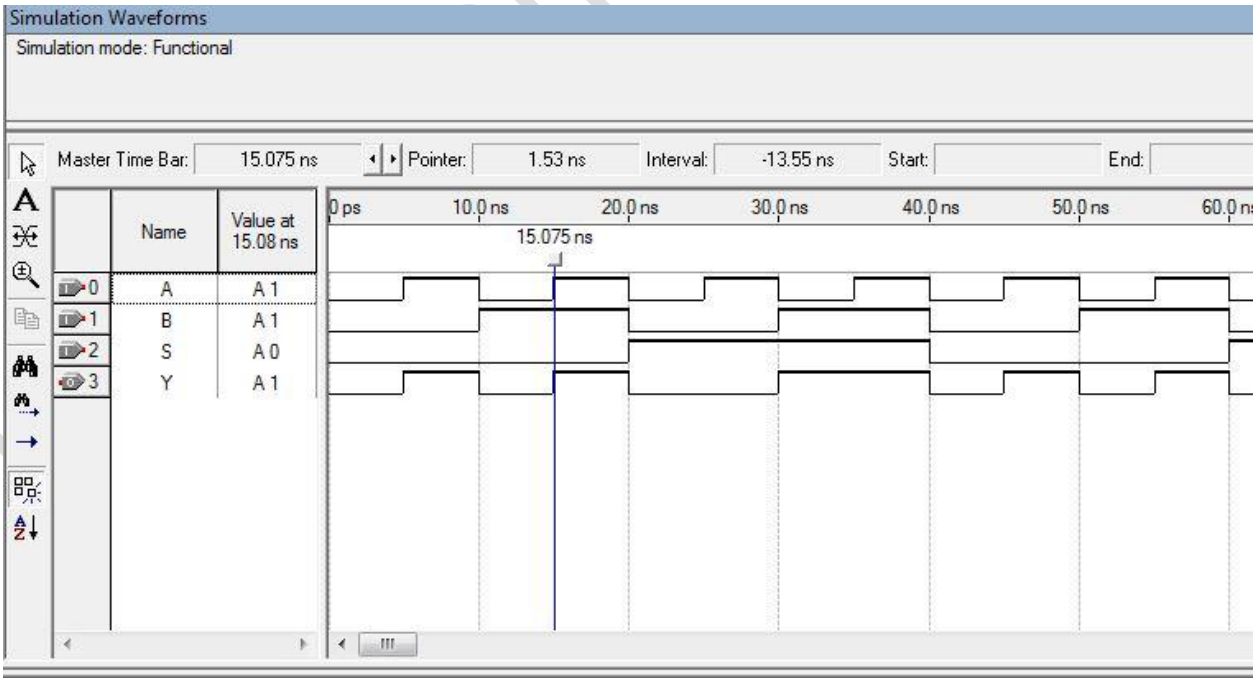
21. In the following window, select ***Functional*** as ***Simulation Mode*** and click on ***Generate Functional Simulation Netlist*.**

22. After generating a functional simulation netlist, the following window will appear. Click ***OK***.



23. Click on the blue play button and you should observe the following message.

24. Now observe the simulation waveforms from ***Processing→Simulation Report***, and note that there is no time delay between inputs and output and the function of this code is indeed that of a 2to1 MUX.



QUESTIONS:

1. Which ones of the following identifier names are incorrect according to Verilog syntax?

2to1MUX, MUX$2to1, mux_2to1, reg, reg4bit, 130105_counter

2. What are the differences between 'structural' and 'behavioral' Verilog codes? Show an example of each for a particular logic circuit.

Combinational Logic circuit design in Verilog HDL using Quartus II

Verilog codes for combinational logic circuits

Half-Adder:

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

2-to-1 MUX:

```
module mux21_fn(I0,I1S,f);
input I0,I1,S;
output reg f;
always@(I0,I1, S)
begin
if (S == 0)
f = I0;
else
f = I1;
end
endmodule
```

Priority Encoder:

```
module priority(W, Y, z);
input [3:0]W;
output reg [1:0]Y;
output reg z;
always @(W)
begin
z = 1;
casex (W)
4'b1xxx: Y = 3;
4'b01xx: Y = 2;
4'b001x: Y = 1;
4'b0001: Y = 0;
default: begin
z = 0;
Y = 2'bx;
```

```
end
endcase
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 C = 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
```

2-to-4 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
```


RTL Synthesis in Quartus II

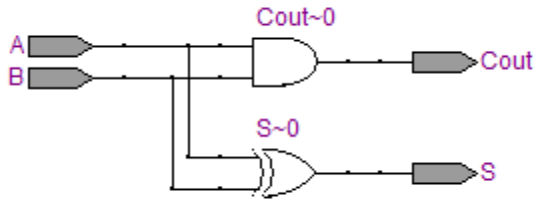
Logic synthesis is a process by which an abstract form of desired circuit behavior), is turned into a design implementation in terms of logic gates, by a synthesis tool. Common examples of this process include synthesis of HDLs, including VHDL and Verilog. Some synthesis tools generate bit-streams for programmable logic devices such as PALs or FPGAs, while others target the creation of ASICs. Logic synthesis is one aspect of electronic design automation.

In digital circuit design, **register-transfer level** (**RTL**) is a design abstraction which models a synchronous digital circuit in terms of the flow of digital signals (data) between hardware registers, and the logical operations performed on those signals. Register-transfer-level abstraction is used in hardware description languages (HDLs) to create high-level representations of a circuit, from which lower-level representations and ultimately actual wiring can be derived.

1. Write Verilog code for a logic circuit, compile it and verify its functionality through simulation. Suppose, you have completed all these steps for Verilog code of a half-adder.

```
1  module halfadder (A,B,Cout,S);
2  input A,B;
3  output S, Cout;
4  assign S = A^B;
5  assign Cout = A&B;
6  endmodule
7
```

2. Execute ***Tools→Netlist Viewers→RTL Viewer*.** A window will appear and you can see the RTL view.



As you can see, this is the logic circuit for a half-adder.