

# Week 1 Recitation

## Introduction & Verilog

### Introduction

The first part of this session will serve as an introduction. You will learn to use the main software for this class: Quartus Prime. With Quartus, you can code in the Verilog Hardware Description Language (HDL) in order to create different logic blocks and subsystems.

### Collaboration Policy

You will be working in groups of 2. Groups are allowed to collaborate.

### Equipment

- Computer with Quartus Prime software

### Tasks

To receive credit for this recitation, you must complete:

- ☐ Task 1: Learn about Verilog including the difference between Structural and Behavioral
- ☐ Task 2: Create a simple NAND gate in Quartus using structural verilog
- ☐ Task 3: Using a waveform to verify the operation of your circuit
- ☐ Task 4: Writing a testbench to verify the operation of your circuit

As you complete tasks, a TA must sign-off on the completion of each task. Ensure that the TA marks the completion of the tasks in Sakai. This recitation acts as a reference for writing and testing Verilog code. You do not need to finish all of the tasks by the end of recitation but you may not leave until either the end of the recitation period or you complete all the provided tasks. If you do not finish and would like to continue working on the tasks outside of recitation, you may go to office hours to receive assistance from the office hours TAs. Additionally, there is a list of common mistakes at the end of this document that may be helpful.

### Grading

- Completing Recitation Tasks: 1 point (pass/fail)

# Recitation

## What is Verilog

Verilog is a hardware description language (HDL). Unlike software programming languages (ex. Java, C), which dictate sequential commands for a computer to execute, HDLs model electronic circuits and systems. It is essential to remember that Verilog code is not a script of instructions: it defines electrical connections, simulating a circuit. These connections can be seen in the Netlist Viewer (described below).

Modeled after C, Verilog shares many similarities with the software language. The largest deviations between the two are **variable declaration** and the **order of code execution**. Verilog variables have defined sizes. Dynamically sized variables are not an option in hardware, so sizes must be fixed. Because circuit simulation does not rely solely on sequential logic, Verilog offers two types of operators: blocking (=) and non-blocking (<=) operators.

**Blocking operators** function like assignment operators in Java, executing **sequentially** (line by line) and assigning the value to the right to the variable on the left. **Non-blocking operators** execute in **parallel**. A series of non-blocking statements will execute at the same time, forming electrical connections practically simultaneously.

## Structural vs Behavioral Verilog

Structural Verilog is writing “code” that physically describes how components such as gates, decoders, multiplexers, and other digital logic devices are connected/wired together. It is similar to taking a breadboard circuit and writing up how each component is put together by labeling every IC and connecting wire. Since you are describing how a set of components is connected, the order does not matter.

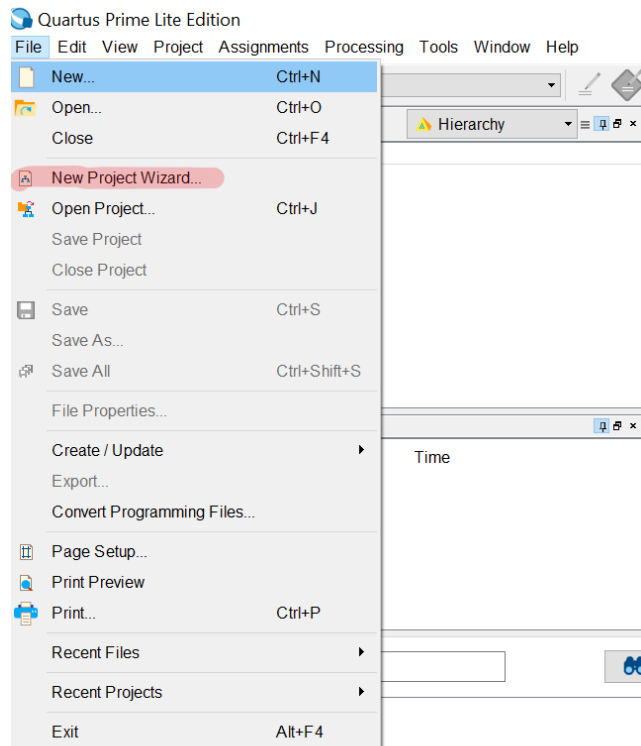
Behavioral Verilog describes the function of a circuit. This is usually done with `always` and/or `assign` statements. With behavioral Verilog, the compiler and synthesizer will determine and optimize the circuit layout of your code.

Structural Verilog is required for this class and project checkpoints since it is up to you to design the specific circuit layout and make your own decisions and trade offs on your implementation.

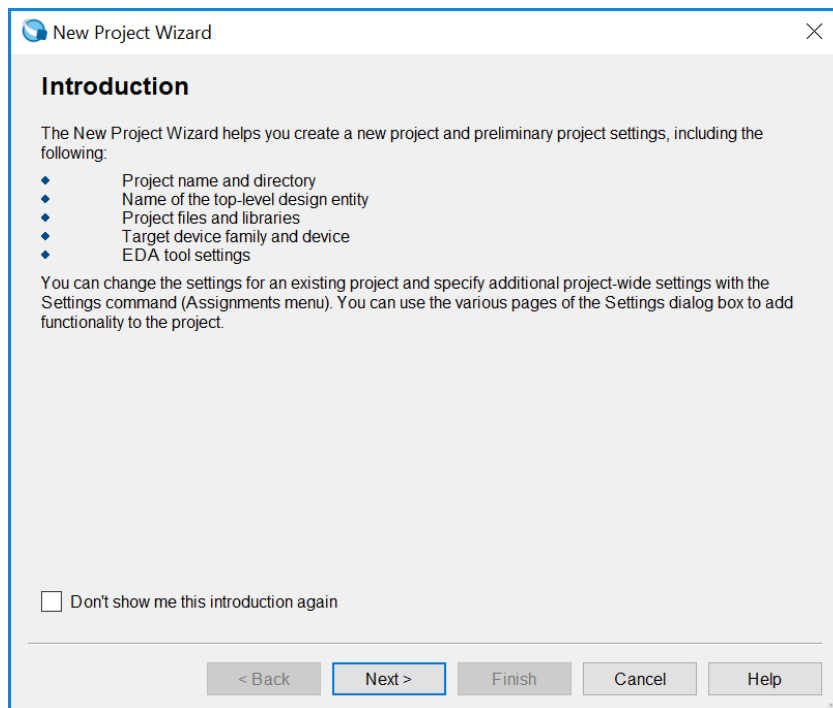
## Writing and Testing Verilog Code

### Quartus Environment Setup

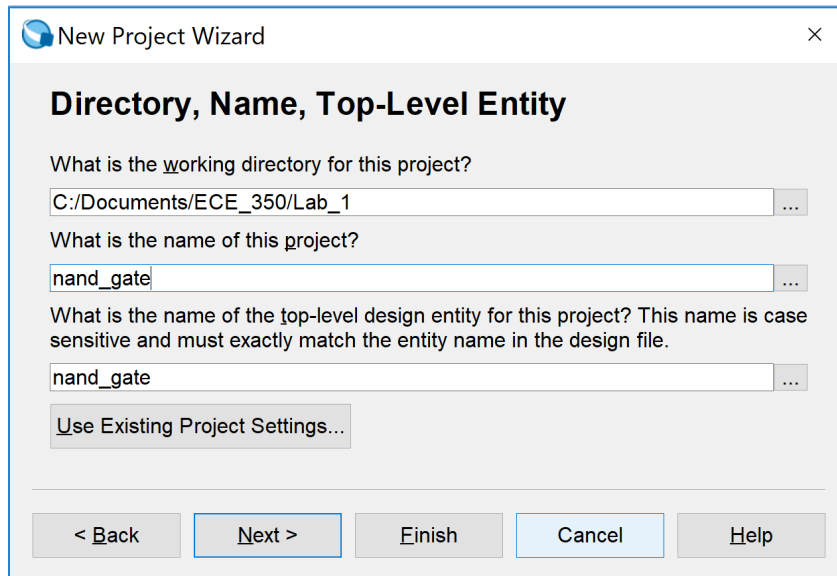
To begin environment setup, open Quartus and select New Project Wizard from File > New Project Wizard.



Select “Next >”



Choose the directory in which you would like to save your project and enter the name of the project. Then select “Next >”



New Project Wizard

### Directory, Name, Top-Level Entity

What is the working directory for this project?

C:/Documents/ECE\_350/Lab\_1 ...

What is the name of this project?

nand\_gate ...

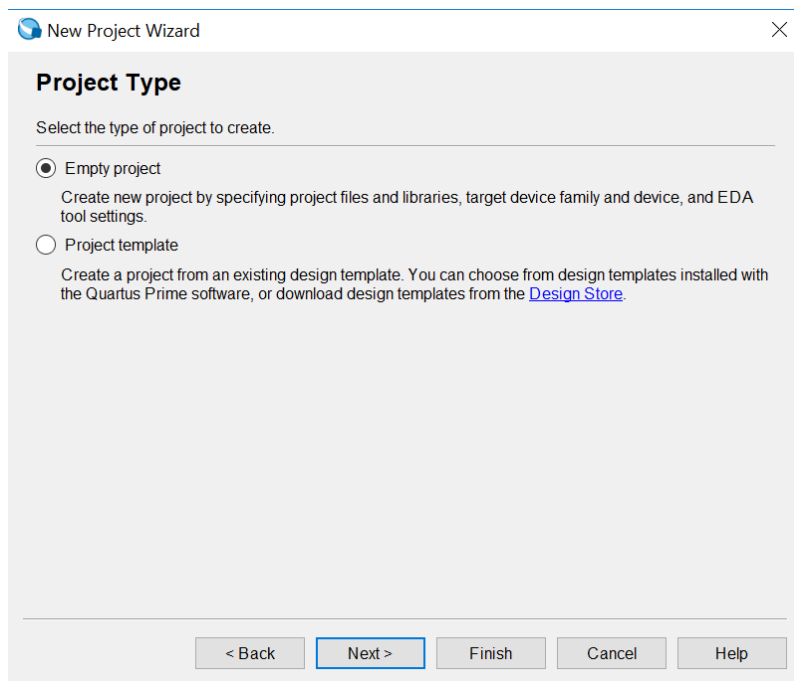
What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

nand\_gate ...

Use Existing Project Settings...

< Back   Next >   Finish   Cancel   Help

Select “Empty project” then “Next >”



New Project Wizard

### Project Type

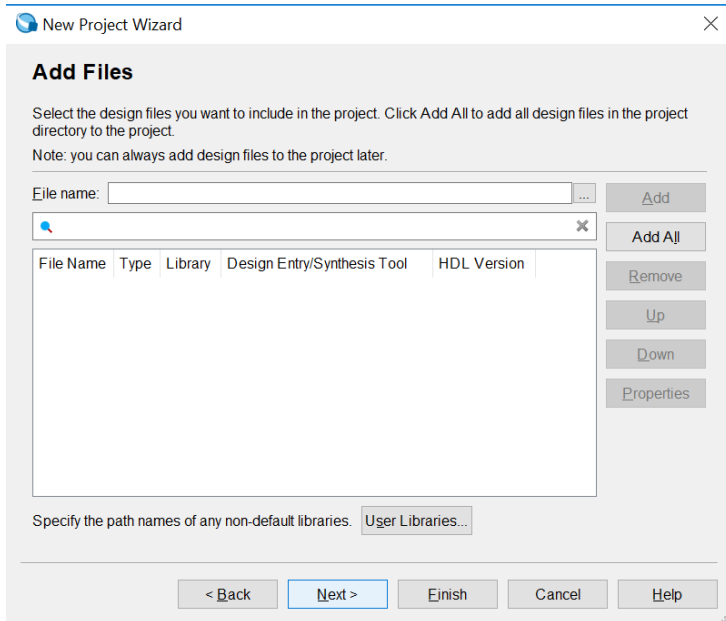
Select the type of project to create.

☒ Empty project  
Create new project by specifying project files and libraries, target device family and device, and EDA tool settings.

☐ Project template  
Create a project from an existing design template. You can choose from design templates installed with the Quartus Prime software, or download design templates from the [Design Store](#).

< Back   Next >   Finish   Cancel   Help

Select “Next >”



The "Add Files" screen of the New Project Wizard. It includes a text field for "File name:", an "Add" button, a search bar, an "Add All" button, a "Remove" button, "Up" and "Down" buttons, and a "Properties" button. A table with columns "File Name", "Type", "Library", "Design Entry/Synthesis Tool", and "HDL Version" is present. At the bottom, there is a "Specify the path names of any non-default libraries" field with a "User Libraries..." button. Navigation buttons at the bottom include "< Back", "Next >", "Finish", "Cancel", and "Help".

**Add Files**

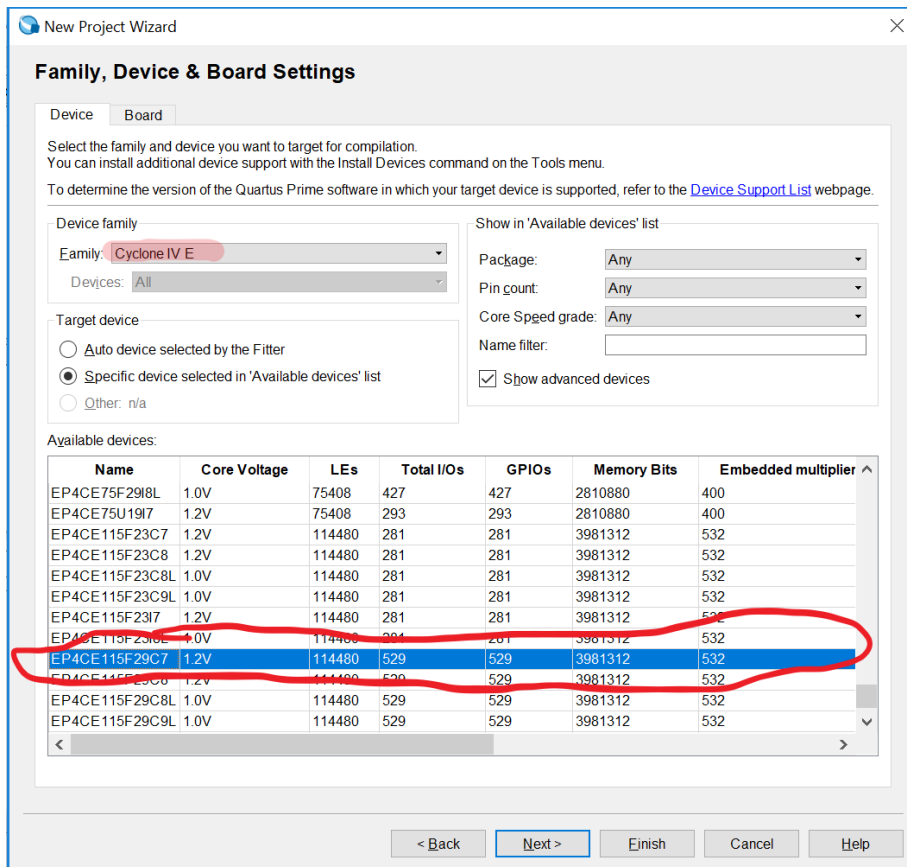
Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

Note: you can always add design files to the project later.

File name:  ...

Specify the path names of any non-default libraries.

Select “Cyclone IV E” under Device Family and then Device Name “EP4CE115F29C7”. Then select “Next >”



The "Family, Device & Board Settings" screen of the New Project Wizard. It has tabs for "Device" and "Board". The "Device" tab is active. It includes sections for "Device family" (Family: Cyclone IV E, Devices: All), "Target device" (Auto device selected by the Fitter, Specific device selected in 'Available devices' list, Other: n/a), and "Show in 'Available devices' list" (Package: Any, Pin count: Any, Core Speed grade: Any, Name filter: , Show advanced devices: ☒). A table of "Available devices" is shown with columns: Name, Core Voltage, LEs, Total I/Os, GPIOs, Memory Bits, and Embedded multiplier. The device EP4CE115F29C7 is highlighted with a red circle. Navigation buttons at the bottom include "< Back", "Next >", "Finish", "Cancel", and "Help".

**Family, Device & Board Settings**

Device Board

Select the family and device you want to target for compilation.  
You can install additional device support with the Install Devices command on the Tools menu.  
To determine the version of the Quartus Prime software in which your target device is supported, refer to the [Device Support List](#) webpage.

Device family

Family: **Cyclone IV E**

Devices: All

Target device

☐ Auto device selected by the Fitter

☒ Specific device selected in 'Available devices' list

☐ Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Core Speed grade: Any

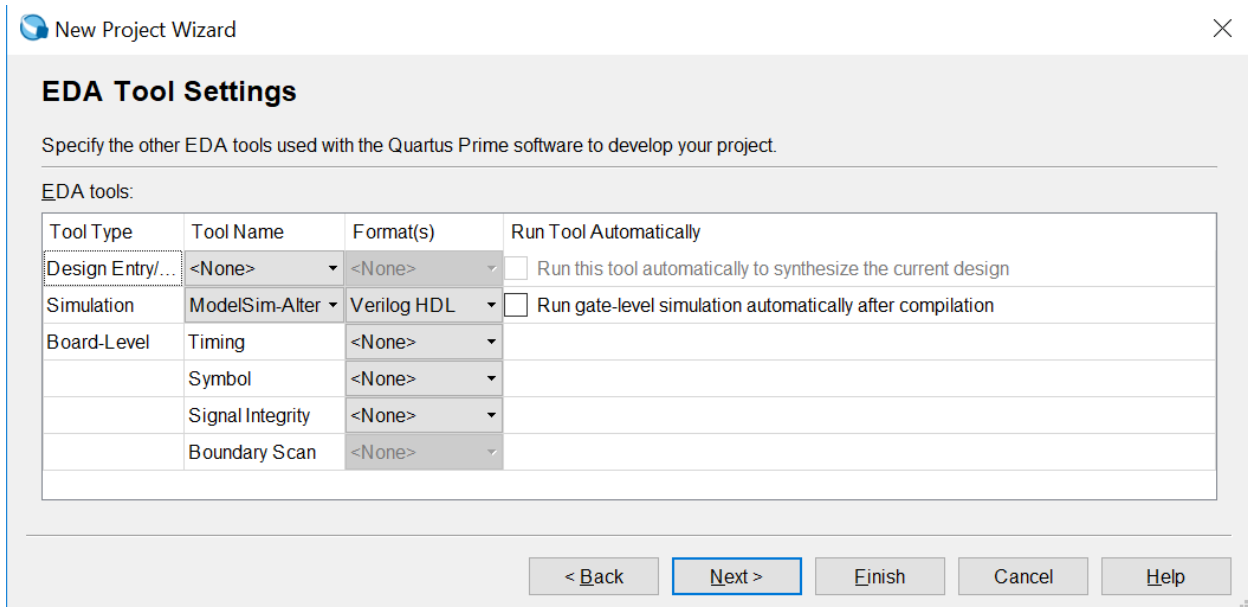
Name filter:

☒ Show advanced devices

Available devices:

Name	Core Voltage	LEs	Total I/Os	GPIOs	Memory Bits	Embedded multiplier
EP4CE75F29B8L	1.0V	75408	427	427	2810880	400
EP4CE75U19I7	1.2V	75408	293	293	2810880	400
EP4CE115F23C7	1.2V	114480	281	281	3981312	532
EP4CE115F23C8	1.2V	114480	281	281	3981312	532
EP4CE115F23C8L	1.0V	114480	281	281	3981312	532
EP4CE115F23C9L	1.0V	114480	281	281	3981312	532
EP4CE115F23I7	1.2V	114480	281	281	3981312	532
EP4CE115F23I7E	1.0V	114480	281	281	3981312	532
EP4CE115F29C7	1.2V	114480	529	529	3981312	532
EP4CE115F29C7E	1.2V	114480	529	529	3981312	532
EP4CE115F29C8L	1.0V	114480	529	529	3981312	532
EP4CE115F29C9L	1.0V	114480	529	529	3981312	532

Select "ModelSim-Altera" under Simulation "Tool Name", "Verilog HDL" under "Format(s)" and then Select "Next >"



New Project Wizard

### EDA Tool Settings

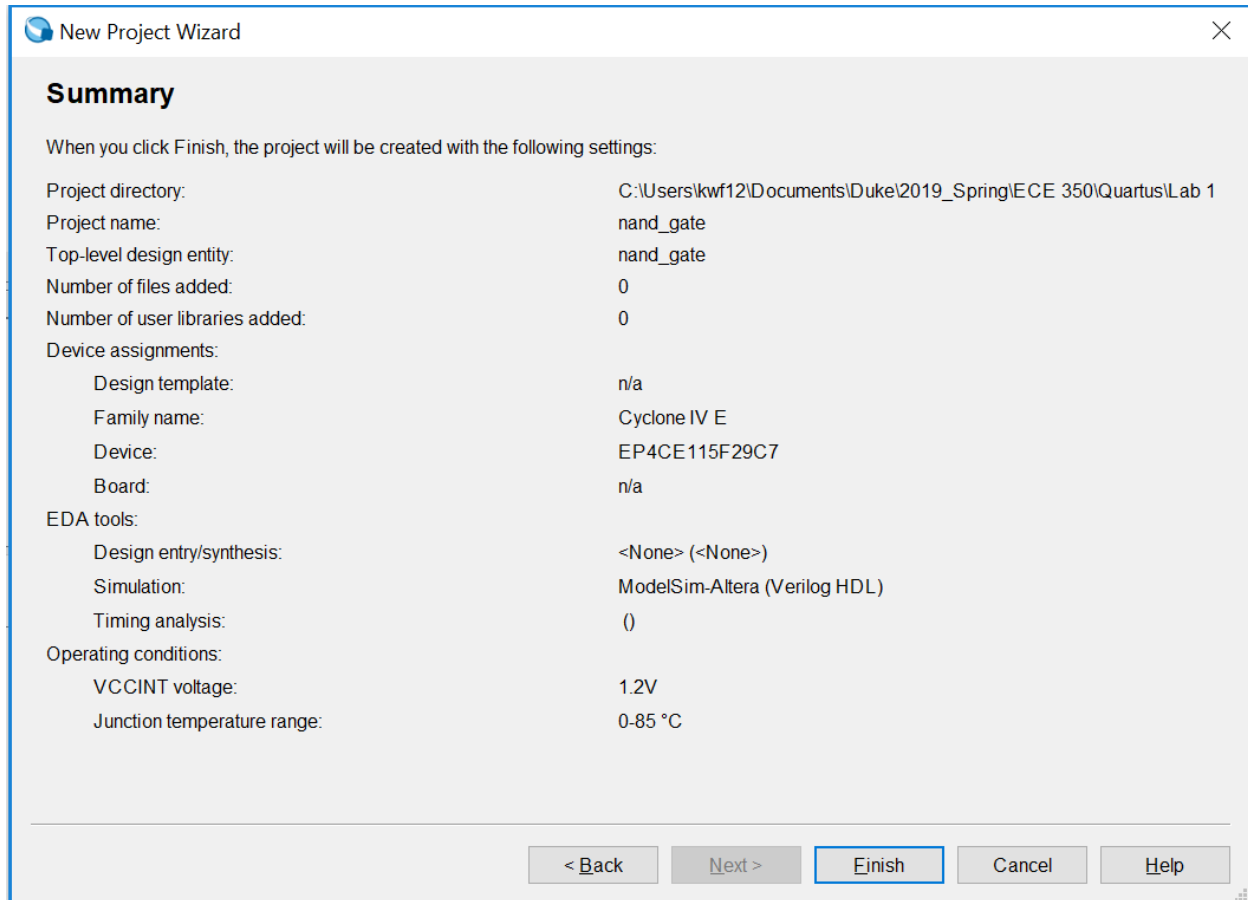
Specify the other EDA tools used with the Quartus Prime software to develop your project.

EDA tools:

Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/...	<None>	<None>	<input type="checkbox"/> Run this tool automatically to synthesize the current design
Simulation	ModelSim-Altera	Verilog HDL	<input type="checkbox"/> Run gate-level simulation automatically after compilation
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

< Back   **Next >**   Finish   Cancel   Help

Select "Finish"



New Project Wizard

### Summary

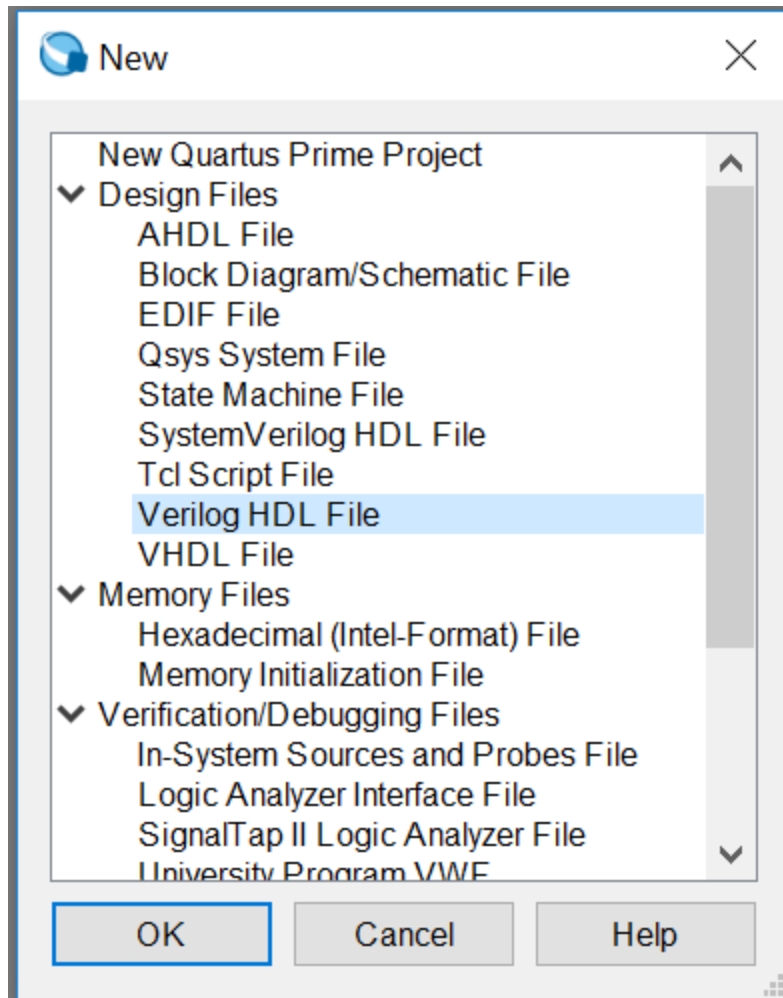
When you click Finish, the project will be created with the following settings:

Project directory:	C:\Users\kwf12\Documents\Duke\2019_Spring\ECE 350\Quartus\Lab 1
Project name:	nand_gate
Top-level design entity:	nand_gate
Number of files added:	0
Number of user libraries added:	0
Device assignments:	
Design template:	n/a
Family name:	Cyclone IV E
Device:	EP4CE115F29C7
Board:	n/a
EDA tools:	
Design entry/synthesis:	<None> (<None>)
Simulation:	ModelSim-Altera (Verilog HDL)
Timing analysis:	()
Operating conditions:	
VCCINT voltage:	1.2V
Junction temperature range:	0-85 °C

< Back   Next >   **Finish**   Cancel   Help

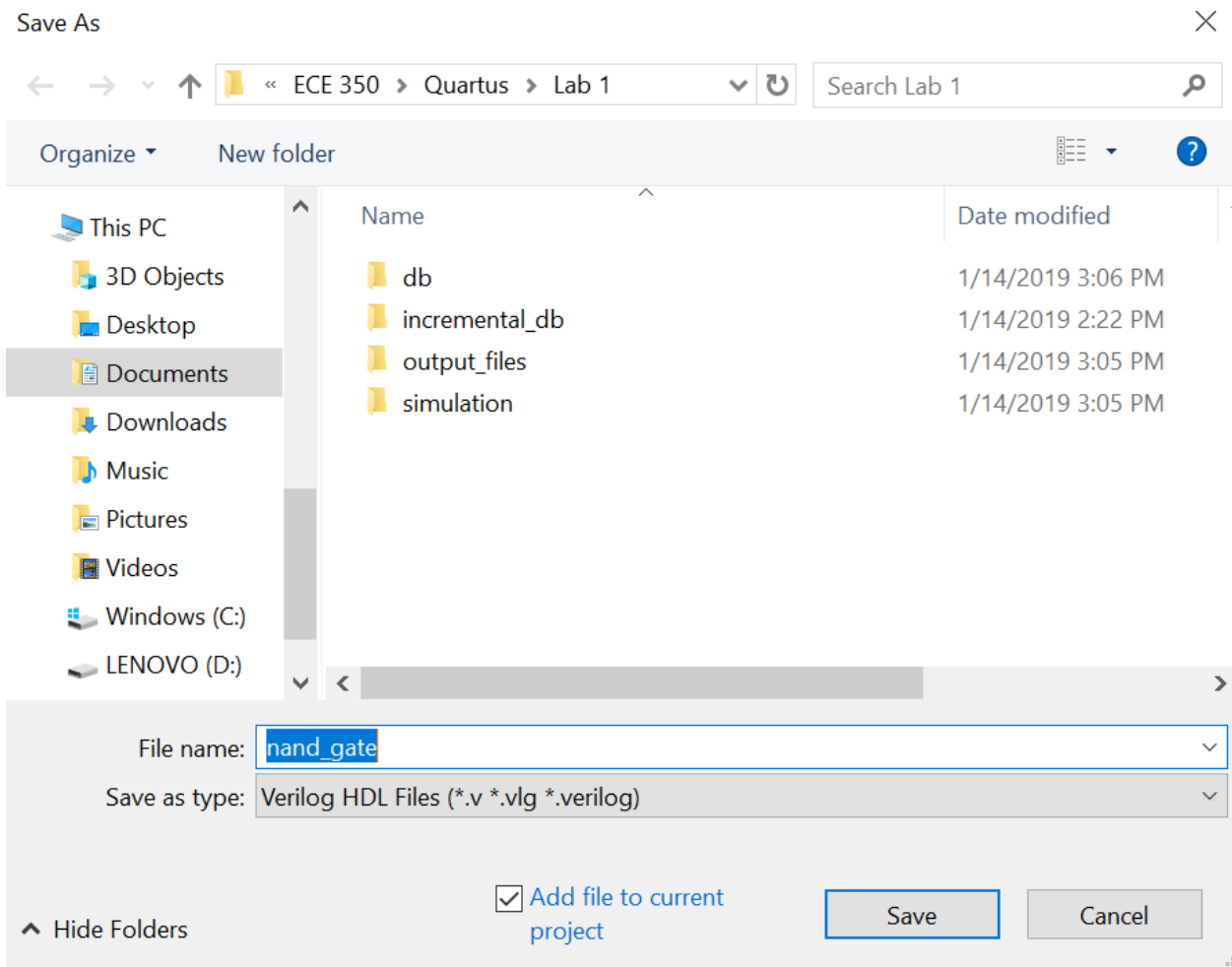
## Creating and Compiling a New Module

File > New, then choose Verilog HDL File:



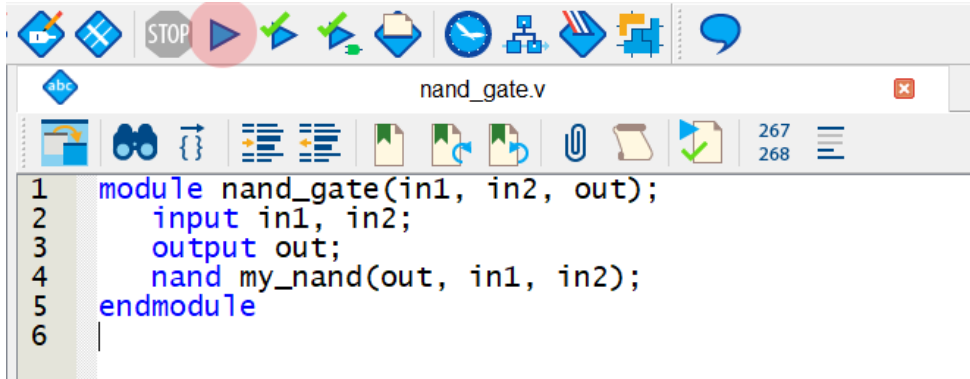
Add the code below to the new file you created. The module name (nand\_gate) must match the project name selected at the beginning in order to compile without modifications

Save the file with the same file name as the module name. Otherwise, there may be errors during testing and grading



Compile the project using the button highlighted below. This button does a full compile. Ask your TA about the different types of compilation. You will need to know these for future assignments





## Testing with Waveforms

Waveforms allow you to vary all of your inputs and see exactly how your outputs behave as a result of combinations of the inputs. You do this by designing a waveform for each of your inputs and then running the simulation; Quartus will then show you a waveform for each of your outputs. Waveforms give a more visual approach to debugging, whereas testbenches are written in Verilog or SystemVerilog (testbenches are covered in the following section).

1. Write your circuit that you wish to test. For this example, we will be testing a simple NAND gate, in a file called nand\_gate.v (you may use the nand gate you wrote in the previous section):

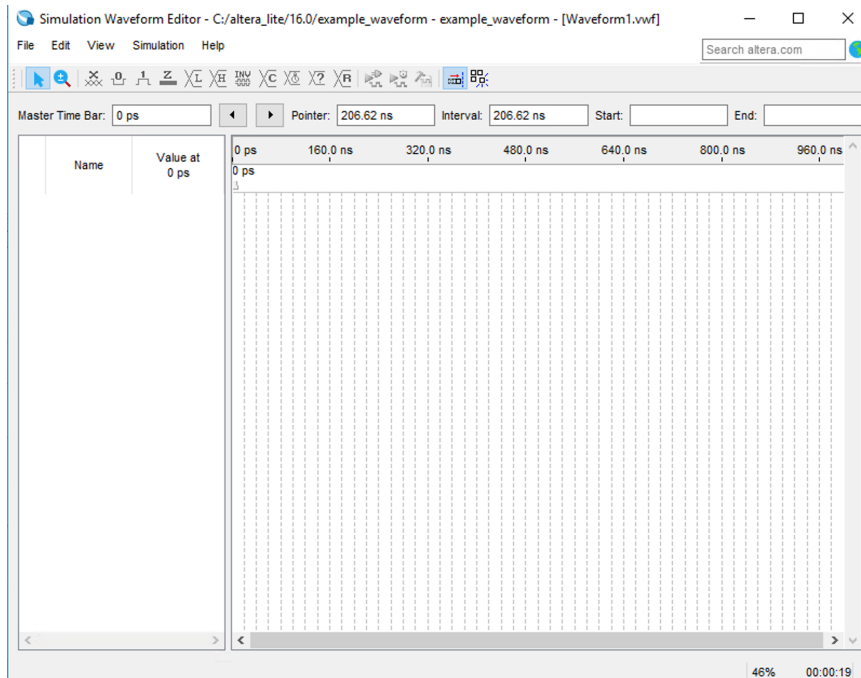
```
module nand_gate(a, b, f);

    input a, b;
    output f;

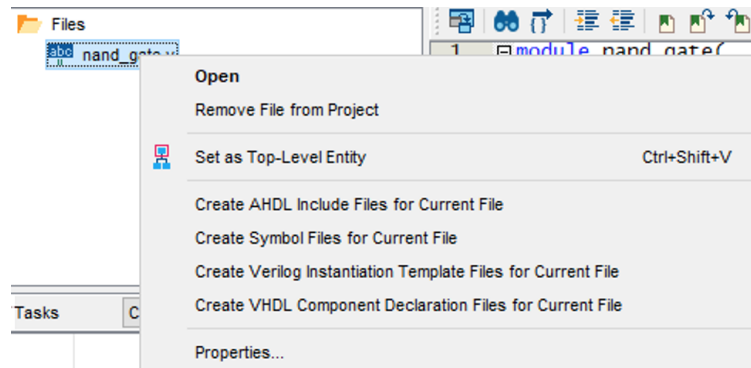
    nand first_nand(f, a, b);

endmodule
```

2. Go to "File -> New..." Under the "Verification/Debugging Files" section, select "University Program VWF". This should bring up this window



- a. **IMPORTANT NOTE**: for the next section, the module which you wish to test must have been compiled as the **top-level entity**. To do this, right-click on the file in the Files viewer and select “Set as Top-Level Entity”. This tells Quartus which file to run simulations on. After that, run the “Analysis and Synthesis” part of compilation by pressing **Ctrl+K**.



3. In the far left column (under “name”) of the Waveform Editor window, right-click anywhere and then select “Insert Node or Bus...” to bring up this window

**Insert Node or Bus**

Name:

Type:

Value type:

Radix:

Bus width:

Start index:

☐ Display gray code count as binary count

OK Cancel Node Finder...

4. Select "Node Finder..." and then select "List". When your nodes appear in the left-hand column (under "Nodes Found"), select the ">>" button. Now, the newly-opened Node Finder window should look as follows:

**Node Finder**

Named:  Filter:

Look in:  ... List Cancel

Nodes Found:

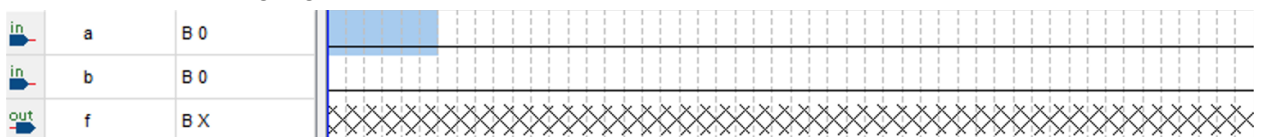
Name	Type
in a	Input
in b	Input
out f	Output

> >> < <<

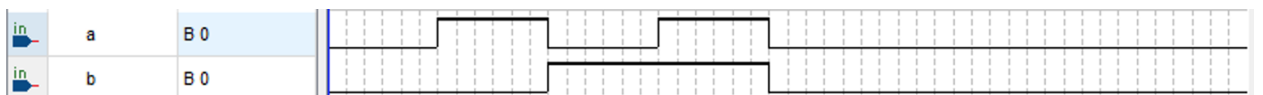
Selected Nodes:



Name	Type
in a	Input
in b	Input
out f	Output

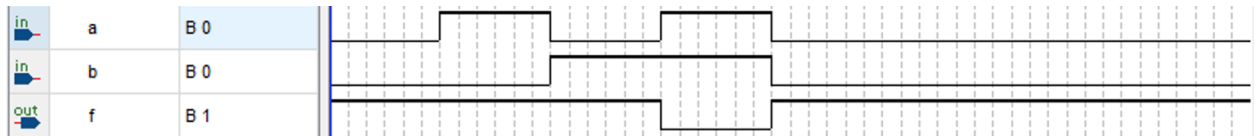
5. Select "OK" and then "OK" on the next window.
6. To begin to vary the inputs, click and drag on a section of the waveform for a given input as such (the blue highlighted portion):



7. To set an input to "high" for a period of time, select the image in the toolbar. To make it behave like a clock (periodic high-low), select the image in the toolbar and set the desired period. In this example, try the waveform shown in the diagram below to test every possible input combination (00, 01, 10, 11):



8. When you have completed the waveforms of each input to your desire, select the  button in the toolbar for a functional simulation (does not take into account timing) or the  button to run a timing simulation. You will only be able to run a timing simulation if you did a full compilation.
- a. **IMPORTANT NOTE:** Quartus will ask you to save the waveform before running the simulation. **Do not alter the name of the waveform** as this may cause issues with the simulation.
9. Quartus will show you the output at each combination of the inputs:

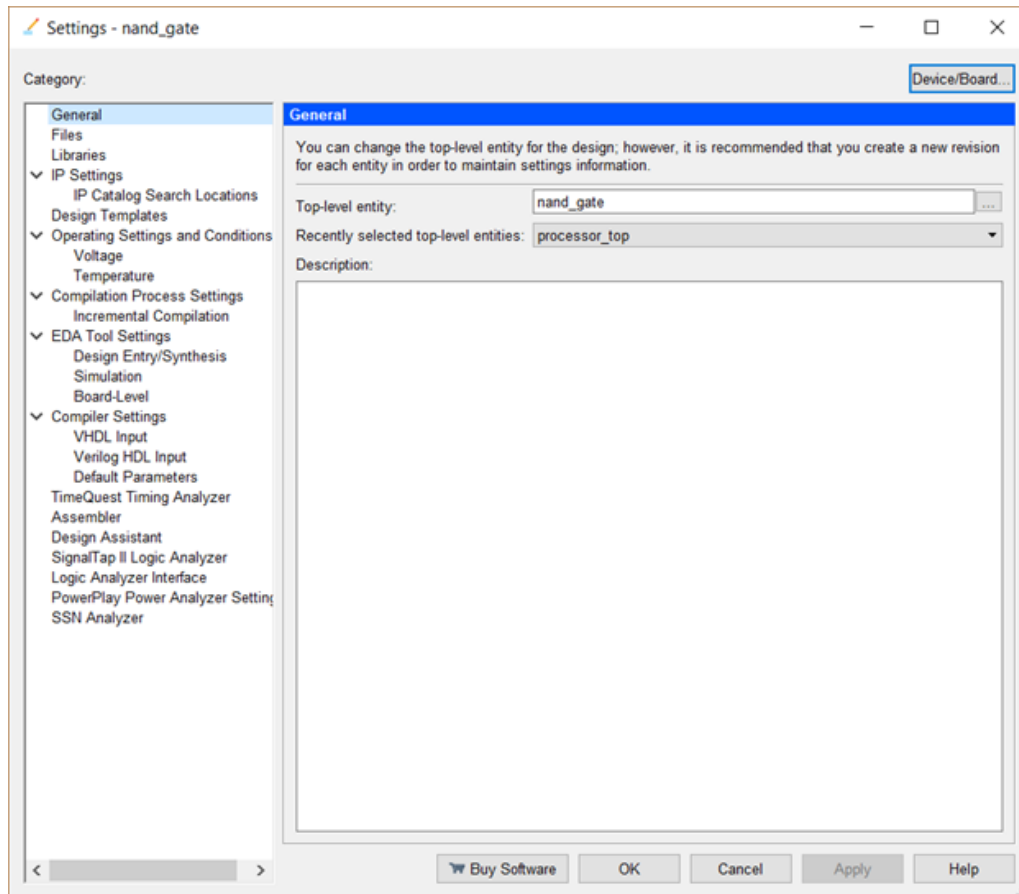


## Testing with Testbenches

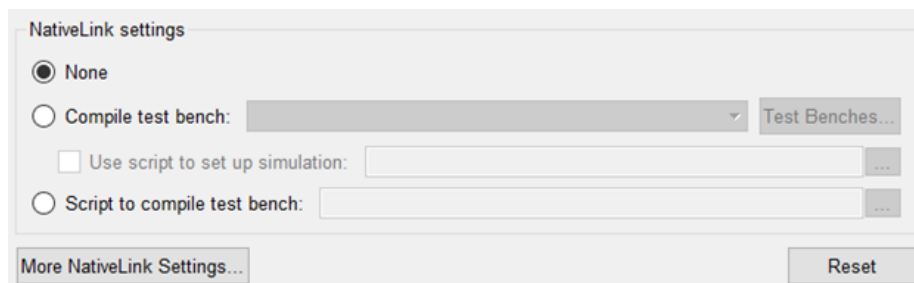
Testbenches are one of the two main ways to test your projects in Quartus. A testbench is akin to a unit test, in that it is a separate piece of code designed to test and validate parts of the actual software. Testbenches can be used to programmatically test any part of your design to as much rigor as you desire. Create a testbench the same way you create a new Verilog module: click File → New → Verilog HDL File.

Note: Always name your testbenches with “\_tb.v” at the end so that graders are aware these Verilog files are testbenches, since otherwise they are difficult to differentiate from regular Verilog files. This is important because testbenches may contain any kind of Verilog code you want, including behavioral Verilog, which would be banned in non-testbench files.

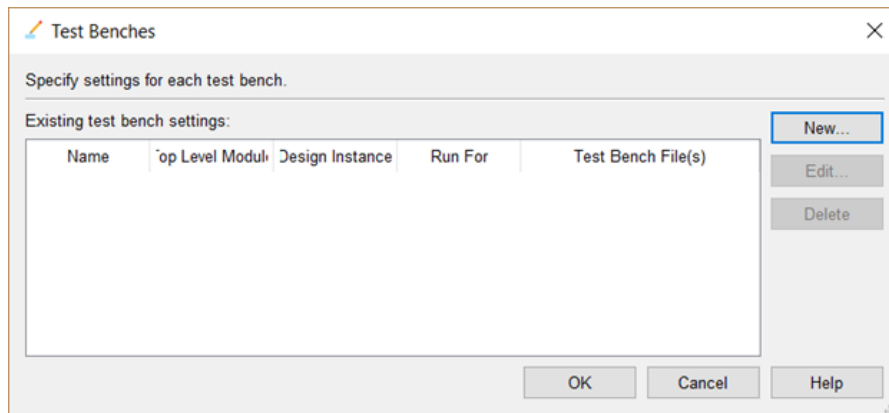
For this example, we will use the same piece of code as above, “nand\_gate.v”. Create an empty testbench called “nand\_gate\_tb.v”. Once your empty testbench is created, you will need to ensure Quartus understands that it is a testbench and not a normal piece of code. This allows Quartus to compile and run the testbench separately after the rest of the code is compiled. To do this, click Assignments → Settings. This should open up a screen that looks like this.



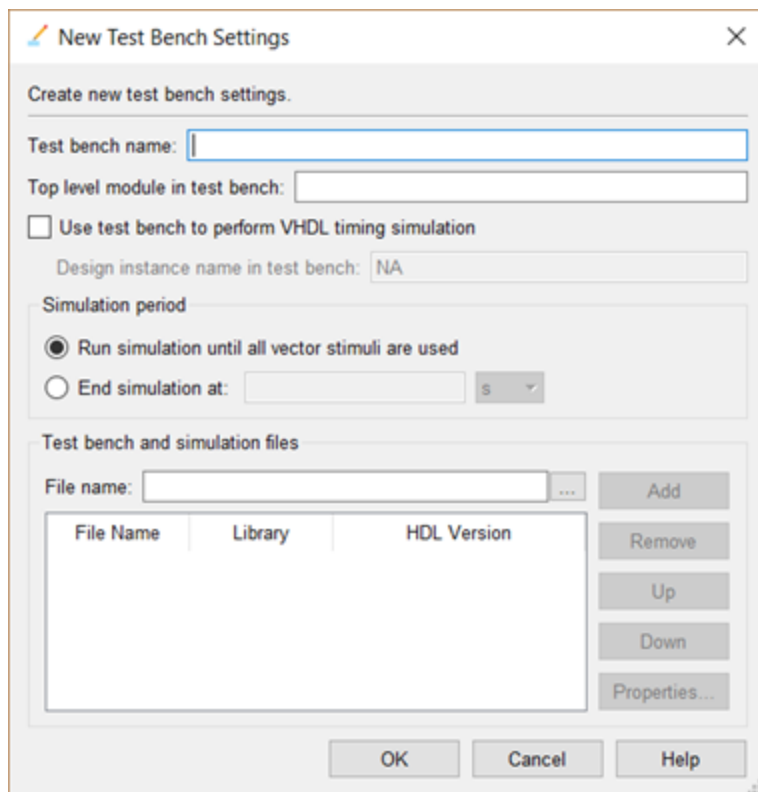
From this screen, click Simulation (under EDA Tool Settings) and look at the NativeLink Settings frame. You will need to select “Compile test bench” and then click the “Test Benches” button.



This will bring up the following page.



From here, click “New” and you will be brought to this page.



Now, just type in the name of the testbench at the top and click the small button with three dots (to the left of the Add button) to find and select your testbench file.

New Test Bench Settings

Create new test bench settings.

Test bench name:

Top level module in test bench:

☐ Use test bench to perform VHDL timing simulation

Design instance name in test bench:

Simulation period

☒ Run simulation until all vector stimuli are used

☐ End simulation at:  s

Test bench and simulation files

File name:  ... **Add**

File Name	Library	HDL Version
-----------	---------	-------------

Remove  
Up  
Down  
Properties...

OK Cancel Help

Finally, click Add. Your testbench will be automatically compiled and run every time you run an RTL simulation or a gate level simulation. For your nand\_gate\_tb.v file, you will need something like the following:

```
// set the timescale
`timescale 1 ns / 100 ps
module nand_gate_tb(); // testbenches take no arguments
    // set up inputs of NAND gate as registers so they can be manipulated
    at will
    reg in1;
    reg in2;

    // clocks are useful for testbenches because they allow you to check
    your
    // circuit at regular intervals large enough for signals to propagate
    reg clock;

    // set up output of NAND gate as wire
    wire out;
```

```

// prepare any other variables you want - nothing is off-limits in a
TB
integer num_errors;

// instantiate the NAND gate
nand_gate test_nand_gate (in1, in2, out);

// begin simulation
initial begin
    $display($time, " simulation start");

    clock = 1'b0;
    num_errors = 0;

    @(negedge clock); // wait for the clock to go negative
    in1 = 1'b0;
    in2 = 1'b0;

    @(negedge clock);
    if (out != 1'b1) begin
        $display("in1 %b in2 %b Error: expected 1 got %b", in1,
in2, out);
        num_errors = num_errors + 1;
    end

    @(negedge clock);
    in1 = 1'b0;
    in2 = 1'b1;

    @(negedge clock);
    if (out != 1'b1) begin
        $display("in1 %b in2 %b Error: expected 0 got %b", in1,
in2, out);
        num_errors = num_errors + 1;
    end

    @(negedge clock);
    in1 = 1'b1;
    in2 = 1'b0;

    @(negedge clock);

```



```

        if (out != 1'b1) begin
            $display("in1 %b in2 %b Error: expected 0 got %b", in1,
in2, out);
            num_errors = num_errors + 1;
        end

        @(negedge clock);
        in1 = 1'b1;
        in2 = 1'b1;

        @(negedge clock);
        if (out != 1'b0) begin
            $display("in1 %b in2 %b Error: expected 0 got %b", in1,
in2, out);
            num_errors = num_errors + 1;
        end

        if (num_errors == 0) begin
            $display("Simulation succeeded with no errors.");
        end else begin
            $display("Simulation failed with %d error(s).",
num_errors);
        end
    end

    always
        #10 clock = ~clock; // toggle clock every 10 timescale units

endmodule

```

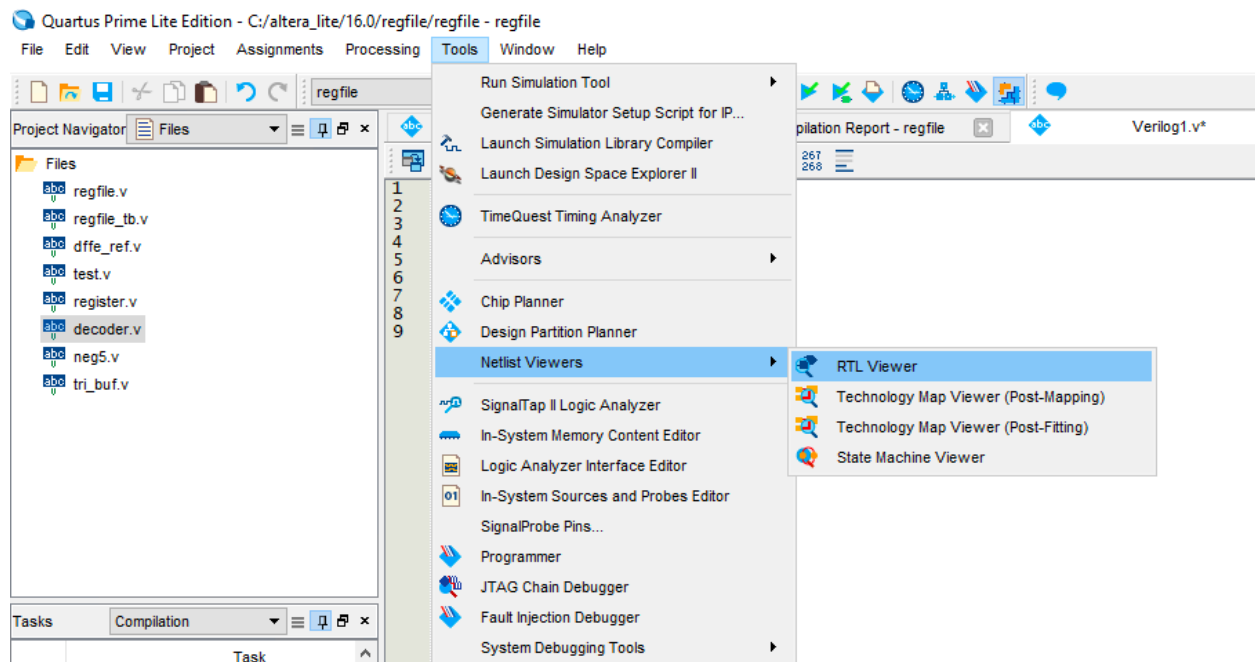
You do not necessarily need a clock for this example, but it can be helpful even when your circuit is not clocked.

Tip: If you make changes to your testbench without making any changes to your code, you do not have to recompile the entire project to run a test simulation again. You can simply save your testbench and run another RTL or gate level simulation and it will work properly.

To run your testbench, click Tools → Run Simulation Tool → RTL Simulation or Gate Level Simulation. The former has no timing checks, the latter does. Run RTL simulations first until your testbenches pass, then work on gate level simulations to make sure the timing is correct. Note: You need to run a full compilation before running gate-level simulations.

# Netlist Viewer

Quartus' Netlist Viewer is a great way to visualize the code you are writing in terms of the hardware it represents. Running the RTL Viewer under Tools -> Netlist Viewers -> RTL Viewer (shown below) after successful compilation will give you a logic gate circuit diagram of the top-level module of your project.



As an example, we will look at the circuit diagram of a simple 2 to 4 bit decoder (look at the Wikipedia page for “Binary Decoder” if you want to know more about them). The code used for this example is provided below, as well as the resulting RTL Viewer window.

```
module dec2to4(out, in);

    //declare inputs and outputs
    input [1:0] in;
    output [4:0] out;

    //declare nIn (the NOT of the input)
    wire [1:0] nIn;

    //wire in through not gates to nIn
```

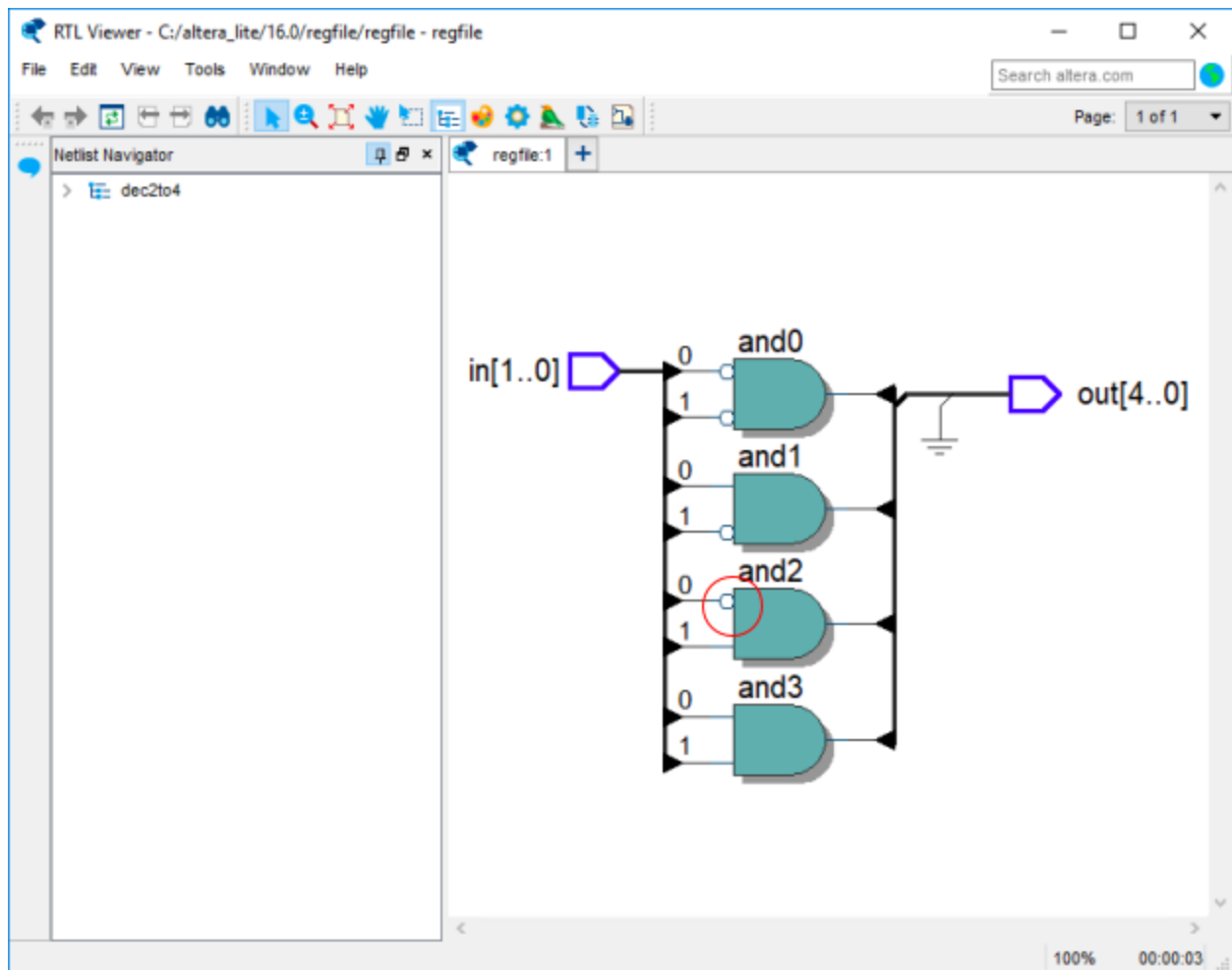
```

not not0(nIn[0], in[0]);
not not1(nIn[1], in[1]);

//decode with AND gates
and and0(out[0], nIn[1], nIn[0]);
and and1(out[1], nIn[1], in[0]);
and and2(out[2], in[1], nIn[0]);
and and3(out[3], in[1], in[0]);

```

```
endmodule
```



As can be seen from this screen, the input and output wires and the four AND gates we declared are obvious. We can see that the input and output wires use splitters to separate each individual bit. To see the NOT gates, we can look at the inputs of the AND gates. We see a little circle (one of them outlined in red above). This means the NOT of the wire is used as the input

to the gate. Netlist Viewer is useful for visually debugging circuits. Sometimes it is very hard to catch something that is miswired in your code, but it's possible that the same error is obvious in the Netlist Viewer. It may be helpful to think of Netlist Viewer as a tool that displays the circuit like you would see it in Logisim.

## Common Errors

- Warning: Verilog HDL Implicit Net warning at filename.v(line#): created implicit net for "w2"

This warning occurs when you use a wire that you didn't declare. Very often, this is because you made a typo when using it. Note that this isn't an error! It will show up under warnings.

- Error: Verilog HDL Declaration error at filename.v(line#): identifier "ex" is already declared in the present scope

This error occurs if you declare two modules with the same name, e.g.:

```
and and0(w2, w1, w0);  
and and0(w3, w2, w0);
```

- Error: "Illegal reference to net [x]"

This error generally occurs when you're using a wire in structural Verilog, where you need to be using a reg

- Critical Warning: Verilog HDL Instantiation warning at filename.v(line#): instance has no name

You've declared a module, but didn't give it a name

- Error: Verilog HDL syntax error at filename.v(line#) near text: "="; expecting ".", or "(".

You need to use the "assign" keyword when you set a wire equal to something, e.g.:

```
assign w0 = 1'b1;
```

- Warning: Verilog HDL assignment warning at filename.v(line#): truncated value with size 10 to match size of target (8)"

You're trying to pass the value of a wire with length x to a wire with length y. This is most likely to happen when you're passing a value into a function, and the function declares that the input should be a different length than that of the wire you're passing in. If the target is too small the value will be truncated; if the target is too large the value will be padded with zeros.

- Error: Can't resolve multiple constant drivers for net "x" at filename.v(line#)

You're trying to assign to a wire in two different places

- Error: Verilog HDL error at filename.v(line#): value cannot be assigned to input "x"
- Error: Net "x" at filename.v(line#) is already driven by input port "drivenInput", and cannot be driven by another signal

You're trying to assign a value to a wire that is an input into the function

- Error: Verilog HDL Module Declaration error at filename.v(line#): top module port "x" is not found in the port list

```
module example();  
input w;  
  
endmodule;
```

The input w is not declared in the port list (parentheses after module name "example")

- Error: Verilog HDL error at filename.v(line#): object "x" is not declared. Verify the object name is correct. If the name is correct, declare the object.

```
module example(w, y);  
input y;  
c  
endmodule;
```

The input w is not declared as an input even though it is in the port list