Part II

Signals Circuits and Systems

Workshop 5: Two-Port Networks

Objective: To understand and verify concepts related to two-port networks.

Outcome: After successful completion of this session, the student will be able to

1. Understand concepts related to ABCD parameters
2. Understand how to model cascaded networks

Equipment Required:

1. Signal generator
2. Oscilloscope
3. Digital multimeter
4. Breadboard and wires

Components Required:

1. Capacitors - 100 nF (4 Nos.), 10 nF (1 No.)
2. Inductors - 10 mH (2 Nos.)
3. Resistors - 10 $k\Omega$ (2 Nos.), 1 $k\Omega$ (1 No.)

5.1 Introduction

A two-port network is an electrical circuit which is considered to be a black box with two pairs of terminals, one for input and the other for output (see Figure 5.1). The characteristics of the network is then modeled using a matrix containing parameters which define the relationship between the input and the output. Once this matrix is known, the output for any known input (or vice-versa) can be easily calculated. In this experiment, we will be focussing on calculating and practically verifying ABCD parameters for different reactive two-port networks.

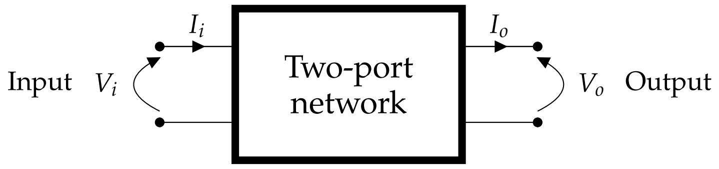


Figure 5.1: A general two-port network

ABCD parameters for the general two-port network illustrated in Figure 5.1 can be defined as

$$\begin{bmatrix} V_i \\ I_i \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} V_o \\ I_o \end{bmatrix} \tag{5.1}$$

which simplifies giving

$$\begin{aligned} V_i &= a_{11}V_o + a_{12}I_o \\ I_i &= a_{21}V_o + a_{22}I_o. \end{aligned} \tag{5.2}$$

The parameters can be evaluated by providing any known input and measuring the output. However, applying the input under the following output port configurations allow us to directly determine each parameter;

$$\begin{aligned}
\text{with output port open-circuited: } a_{11} &= \left.\frac{V_i}{V_o}\right|_{I_o=0} \\
\text{with output port short-circuited: } a_{12} &= \left.\frac{V_i}{I_o}\right|_{V_o=0} \\
\text{with output port open-circuited: } a_{21} &= \left.\frac{I_i}{V_o}\right|_{I_o=0} \\
\text{with output port short-circuited: } a_{22} &= \left.\frac{I_i}{I_o}\right|_{V_o=0} \qquad (5.3)
\end{aligned}$$

If we assume that a reactive load Z_L has been connected to the output port of the two-port network shown in Figure 5.1, the voltage gain $A_V = V_o/V_i$ of the network can be calculated in terms of ABCD parameters as

$$A_V = \frac{Z_L}{a_{11}Z_L + a_{12}}. \qquad (5.4)$$

5.2 Pre-Lab

Task 1. *Consider the two-port network illustrated in Figure 5.2. Following (5.3), calculate the ABCD parameters for the network in terms of C_1, C_2, R_1 and the frequency f.*

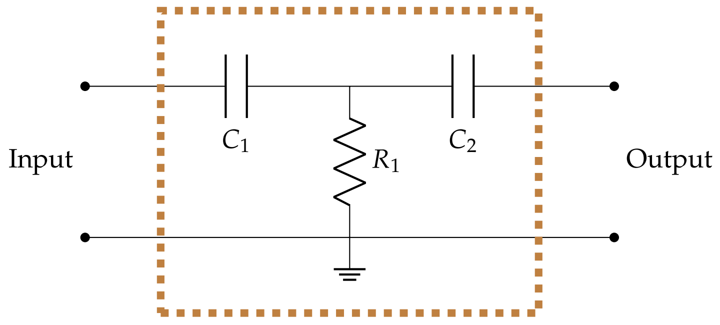


Figure 5.2: Two-port network I

Task 2. *Assume that a load of $Z_L = 1\ k\Omega$ is attached to the output port of the above two-port network. Following (5.4), calculate the voltage gain for the following values of input frequencies when $C_1 = C_2 = 100\ nF$ and $R_1 = 10\ k\Omega$; $f = 1\ kHz$, $f = 10\ kHz$*

Task 3. *Consider the two-port network illustrated in Figure 5.3. Calculate the ABCD parameters for the network in terms of L_1, L_2, C_3 and the frequency f.*

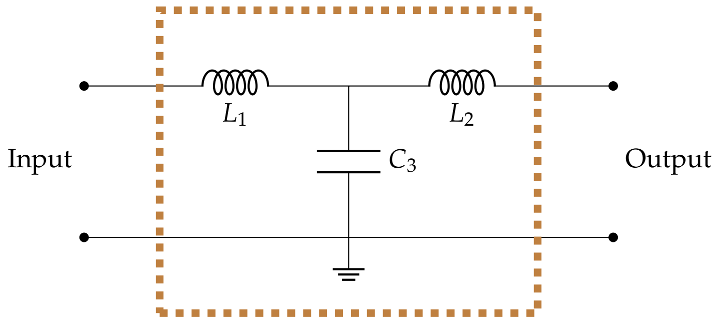


Figure 5.3: Two-port network II

Task 4. *Assume that a load of $Z_L = 1\ k\Omega$ is attached to the output port of the above two-port network. Following (5.4), calculate the voltage gain for the following values of input frequencies when $L_1 = L_2 = 10\ mH$ and $C_3 = 10\ nF$; $f = 10\ kHz$, $f = 100\ kHz$*

Task 5. *Determine the combined ABCD parameters for the cascaded two-port network illustrated in Figure 5.4 in terms of C_1, C_2, R_1 and the frequency f.*

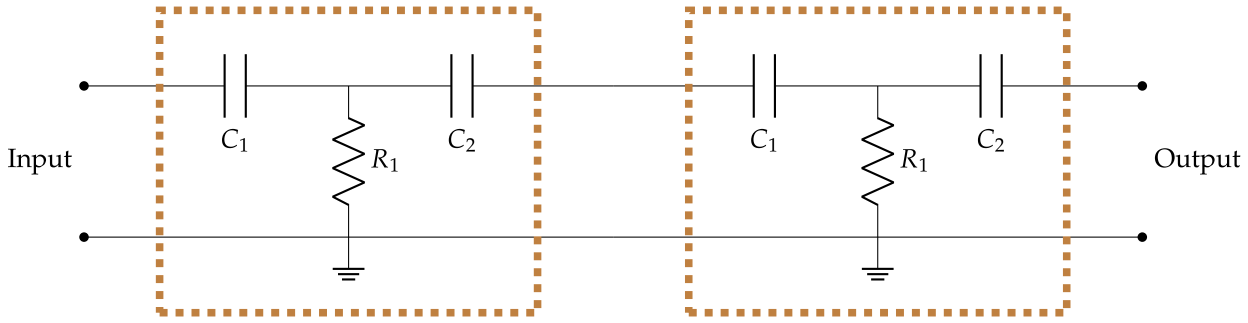


Figure 5.4: Cascaded two-port network

Task 6. *Assuming that a $Z_L = 1\ k\Omega$ load is connected to the output of the cascaded network, calculate the voltage gain of the network if $C_1 = C_2 = 100\ nF$ and $R_1 = 10\ k\Omega$ for the following frequencies;*
$f = 1\ kHz$, $f = 10\ kHz$

5.3 RC Network

Task 7. *Implement the circuit given in Figure 5.2 on the breadboard with $C_1 = C_2 = 100\ nF$ and $R_1 = 10\ k\Omega$. Connect a load $Z_L = 1\ k\Omega$ to the output port. Using the signal generator, provide a 1 V peak-to-peak sinusoidal signal to the input with the frequencies mentioned below. Observe the input and output waveforms using the oscilloscope and calculate the voltage gain corresponding to each frequency, $f = 1\ kHz$ and $f = 10\ kHz$.*

Task 8. *State any reasons for the differences (if there are any) observed between the theoretical gains calculated under Task 2 and the values obtained in the previous task.*

Task 9. *Comment on the nature of the dependency of the voltage gain on the frequency.*

5.4 LC Network

Task 10. *Implement the circuit given in Figure 5.3 on the breadboard with $L_1 = L_2 = 10\ mH$ and $C_3 = 10\ nF$. Connect a load $Z_L = 1\ k\Omega$ to the output port. Using the signal generator, provide a 1 V peak-to-peak sinusoidal signal* *to the input with the frequencies mentioned below. Observe the input and output waveforms using the oscilloscope and calculate the voltage gain for each frequency, $f = 10\ kHz$ and $f = 100\ kHz$.*

Task 11. *Comment on the nature of the dependency of the voltage gain on the frequency.*

5.5 Cascaded Network

Task 12. *Combine two RC networks (implemented under Task 7) to construct the cascaded two-port network illustrated in Figure 5.4. Connect a load $Z_L = 1\ k\Omega$ to the output port of the cascaded network. Provide a $1\ V$ peak-to-peak sinusoidal input with frequencies mentioned below, to the network using the signal generator. Observe the input and the output waveforms using the oscilloscope and calculate the voltage gains.*
$f = 1\ kHz$, $f = 10\ kHz$

Task 13. *Comment on the agreement of practically observed gains with the theoretical values calculated under Task 6.*

Task 14. *Comment on the nature of the dependency of the voltage gain of the cascaded network on the frequency.*

♣ The End ♣

Part III

Electronics

Workshop 5: Design and Implementation of a Full Adder

Objective: To design, implement and verify a full adder as an example of a combinational logic circuit

Outcome: After successful completion of this session, the student will be able to

1. Design combinational logic circuits using truth tables
2. Implement combinational logic circuits using digital ICs

Equipment Required:

1. A personal computer.
2. Intel Quartus Prime Lite Edition Design Software Version 20.1.1
3. ModelSim Intel FPGA Edition Version 20.1.1
4. Intel Cyclone IV Device Support file
5. DC power supply
6. Digital multimeter
7. Logic probe
8. Breadboard and wires

Components Required:

1. Logic gate ICs - 7486(XOR), 7408(AND), 7432(OR), 7483(Full adder) (1 No. each)
2. Resistors - 330 Ω (4 Nos.)
3. LEDs - Yellow (4 Nos.), Red (4 Nos.)

5.1 Introduction

This experiment focuses on designing and implementing combinational logic circuits. We consider a full adder as an example of a combinational logic circuit. The full adder will be designed using hierarchical design approach, where basic building blocks (half adders in this case) are used to design a complex circuit. The initial design done using the truth tables will be simulated using *Quartus Prime* and *ModelSim* software tools. Later, the designed full adder will be implemented using digital ICs. Finally, the functionality of a commercially available full adder IC will be observed.

A single-bit half adder simply performs the addition of two bits and outputs the sum and the carry-out. Examples: $1 + 0 \implies \text{sum} = 1, \text{carry-out} = 0$; $1 + 1 \implies \text{sum} = 0, \text{carry-out} = 1$. However, we cannot extend half adders to add multi-bit numbers. Hence, we construct a full adder, which has an additional input named carry-in, using 2 half adders. Unlike half adders, full adder blocks can be repeatedly connected together to achieve addition of multi-bit numbers.
Example:

$$\begin{array}{cccc} & & \text{carry-in=1} & \\ & 1 & 0 & 1 \\ + & 1 & 0 & 1 \\ \hline & 0 & 1 & 0 \\ \hline\hline & \text{carry-out=1} & & \end{array} \tag{5.1}$$

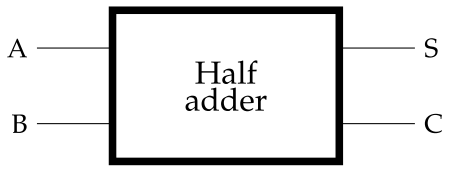


Figure 5.1: Block diagram of a half adder. A and B are input bits, S is the sum and C is the carry-out.

5.2 Pre-Lab

5.2.1 Truth Tables

Task 1. *Complete the truth table of the half adder (Table 5.1). Refer Figure 5.1.*

First bit (A)	Second bit (B)	Sum (S)	Carry-out (C)
0	0		
0	1		
1	0		
1	1		

Table 5.1: Truth table of the half adder

Task 2. *Derive and simplify boolean expressions for the outputs, S and C, in terms of the input bits, A and B.*

Task 3. *Complete the truth table of the full adder (Table 5.2).*

First bit (A)	Second bit (B)	Carry-in (C_{in})	Sum (S)	Carry-out (C_{out})
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Table 5.2: Truth table of the full adder

Task 4. *Derive boolean expressions for the outputs, S and C_{out}, in terms of the inputs A, B and C_{in}. Factorize the expressions so that the full adder can be implemented using 2 half adder blocks. Hint: Refer Task 2.*

Task 5. *Draw the block diagram of a full adder constructed using 2 half adders.*

5.2.2 Simulation Using Quartus and ModelSim

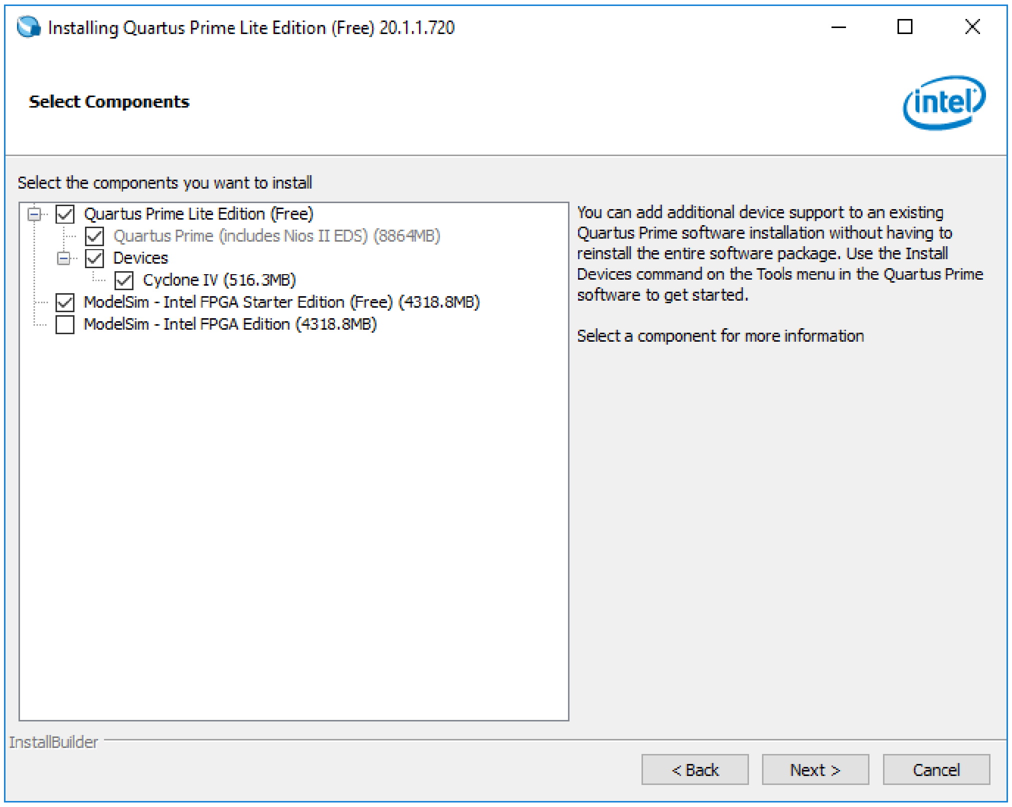
This section provides a step-by-step guide for installing Quartus and ModelSim, and simulating the full adder designed in the previous section.
Note: Required setup files have to be downloaded before starting the section.

Setting-up the Environment

Download setup files corresponding to

- Intel Quartus Prime Lite Edition Design Software Version 20.1.1
- ModelSim Intel FPGA Edition Version 20.1.1
- Intel Cyclone IV Device Support file

into a single directory and run the Quartus installer. Proceed until the installer prompts for selecting components to install. Select all the components in the list except ModelSim paid version (see Figure 5.2).


Installing Quartus Prime Lite Edition (Free) 20.1.1.720

Select Components

Select the components you want to install

- [✓] Quartus Prime Lite Edition (Free)
  - [✓] Quartus Prime (includes Nios II EDS) (8864MB)
  - [✓] Devices
    - [✓] Cyclone IV (516.3MB)
- [✓] ModelSim - Intel FPGA Starter Edition (Free) (4318.8MB)
- [ ] ModelSim - Intel FPGA Edition (4318.8MB)

You can add additional device support to an existing Quartus Prime software installation without having to reinstall the entire software package. Use the Install Devices command on the Tools menu in the Quartus Prime software to get started.

Select a component for more information

Figure 5.2: Quartus installer: selecting components to install

After setting-up the required software tools, open *Quartus Prime Lite 20.1.1* Design Software. It will bring you to the home screen of Quartus illustrated in Figure 5.3. To verify that the ModelSim has been integrated correctly, complete the steps mentioned below;

1. Click **Tools > Options** from the menu bar. This will open up the **Options** dialog box.
2. Select **EDA Tool Options** listed under **General** category in the left-hand side pane.
3. There should be file paths mentioned in-front of **ModelSim** and **ModelSim-Altera** (see Figure 5.4).
4. From your file manager/explorer, navigate to the mentioned location and check whether **modelsim.exe** file exists. If so, you are good to go. Press **Cancel** button to exit without modifying anything.
5. If not, find the location of the **modelsim.exe** file and replace the current paths by the correct path. Press **OK** to save and exit.

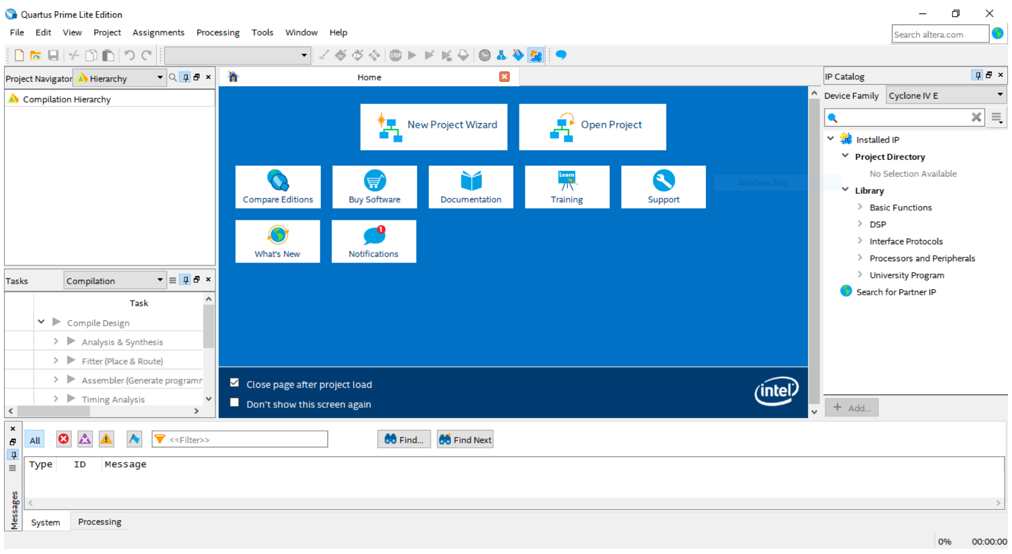
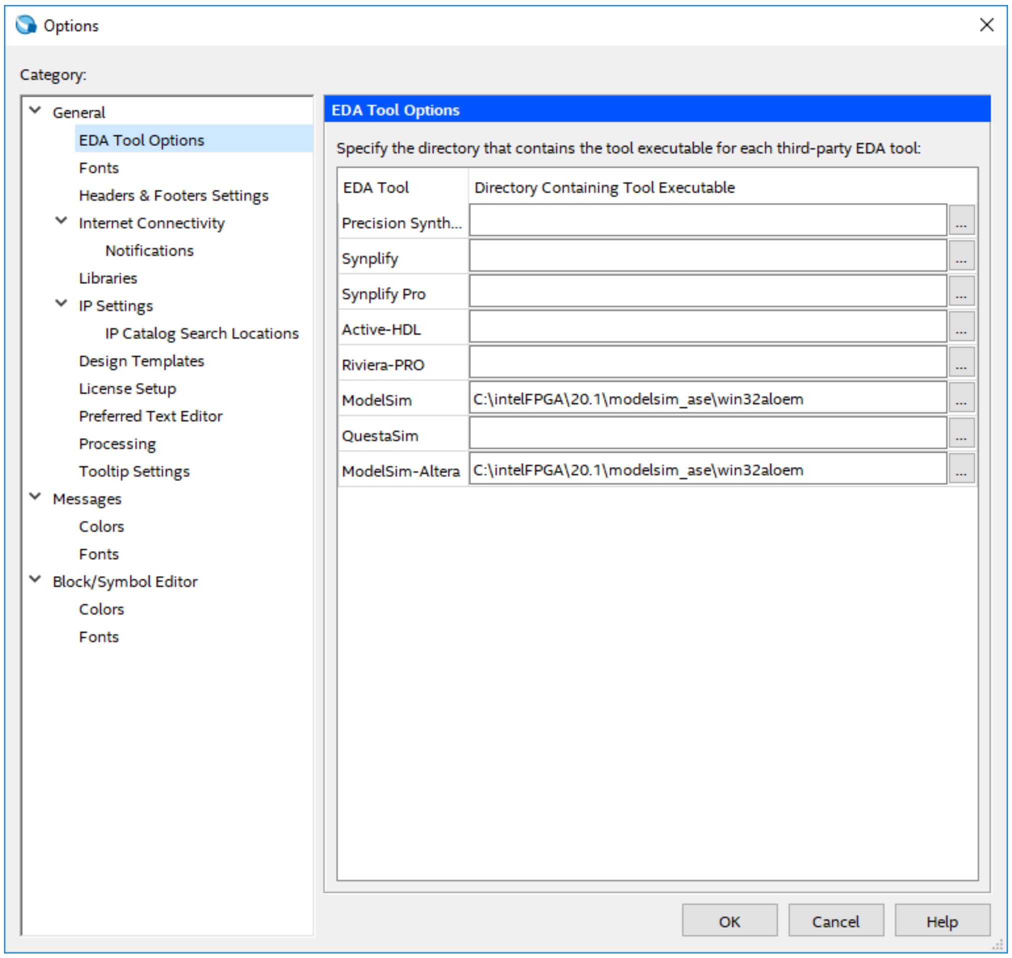


Figure 5.3: Quartus home screen



EDA Tool Options

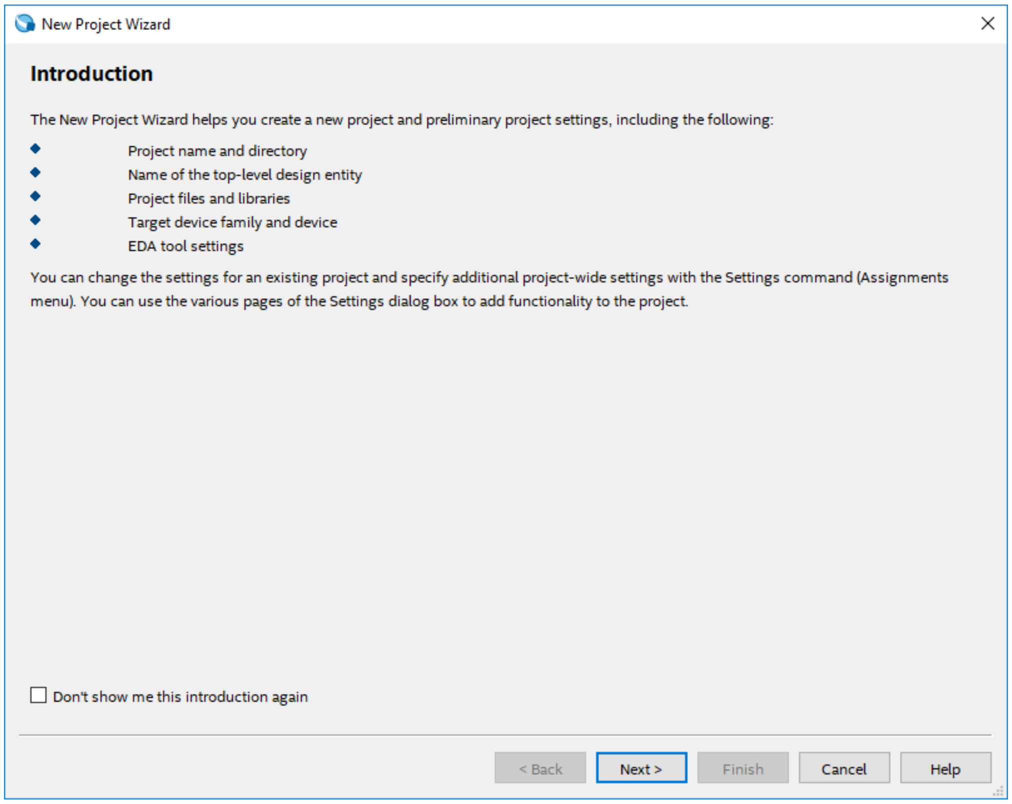
Specify the directory that contains the tool executable for each third-party EDA tool:

EDA Tool	Directory Containing Tool Executable
Precision Synth...	
Synplify	
Synplify Pro	
Active-HDL	
Riviera-PRO	
ModelSim	C:\intelFPGA\20.1\modelsim_ase\win32aloem
QuestaSim	
ModelSim-Altera	C:\intelFPGA\20.1\modelsim_ase\win32aloem

Figure 5.4: Setting ModelSim path

Creating a New Project

To start the **New Project Wizard**, either click on the "New Project Wizard" button on the home screen or goto **File > New Project Wizard**. This will open up the window illustrated in Figure 5.5.



Introduction

The New Project Wizard helps you create a new project and preliminary project settings, including the following:

- Project name and directory
- Name of the top-level design entity
- Project files and libraries
- Target device family and device
- EDA tool settings

You can change the settings for an existing project and specify additional project-wide settings with the Settings command (Assignments menu). You can use the various pages of the Settings dialog box to add functionality to the project.

☐ Don't show me this introduction again

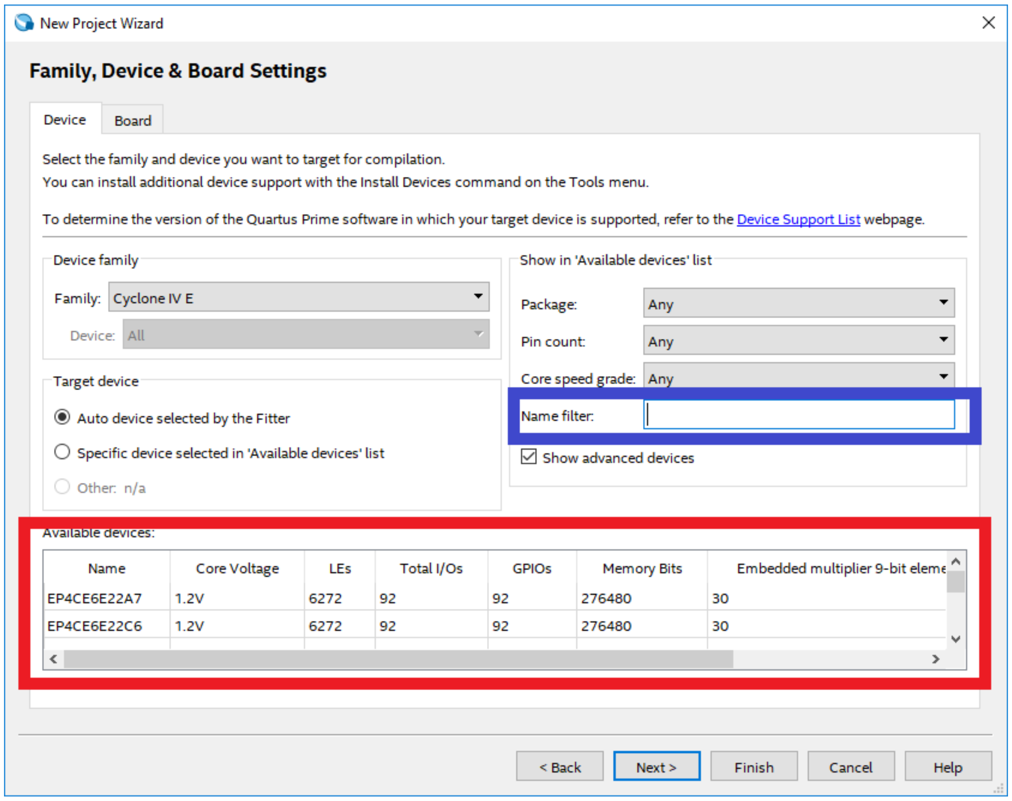
Figure 5.5: New Project Wizard

Follow the instructions listed below to create a new project.

1. Click **Next** to navigate to the project details form.
2. Specify a working directory and a suitable name for the project and click **Next**.
3. Specify the project type. Since we are creating this project from scratch, select **Empty project**. Click **Next**.
4. If we need to use any already-existing files, we can specify them here. Since there are no such files, click **Next** to continue.
5. Carefully select the device family and model. In order to use *DE0-Nano* development board (which will be used in a later experiment), select the following;

   - Family: Cyclone IV E
   - Device: EP4CE22F17C6N

   See Figure 5.6. Click **Next** to continue.
6. If we need to use any additional third part software tools along with Quartus, we can specify them here. Since we are not going to use any, click **Next** to move on.
7. Finally, a summary of all the attributes of the new project will be shown. If everything is correct, click **Finish**. This will create the new project and will bring you to the screen shown in Figure 5.7.



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
Device: 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 9-bit eleme
EP4CE6E22A7	1.2V	6272	92	92	276480	30
EP4CE6E22C6	1.2V	6272	92	92	276480	30

Figure 5.6: Family, device and board selection. Red square: select device type here. Blue square: type a part of the device name to filter the list

Constructing the Half Adder as a Schematic Design

As the first step of creating the schematic, we need to add a new file under the new project. We will construct our half adder module as a schematic design, where we can draw the block diagram of the module using basic logic gates. To add a schematic design file, follow the steps mentioned below;

1. Click **File > New**. This will open up the **New** window.

2. Select **Block Diagram/Schematic file** under **Design Files** category and click **OK**.

This will open the **Graphic Editor** window, where we can draw our schematic. In order to place logic components on the canvas, click on icon from the tool bar. It will open the **Symbol** window. Inside the **Libraries** pane, expand into **primitives > logic** and select a logic gate to place it on the canvas. While the **Repeat-insert mode** is checked, you can repeatedly click on the canvas to place several instances of the selected logic components.

Similarly, to place input/output ports of the module, select icon from the tool bar. To draw wires connecting the components, select icon. Any component or wire can be removed by first selecting the component/wire and then pressing **Delete** on keyboard. To change the properties of a component (name, default value etc.) double click on the component itself. This will open up a properties window.

Task 6. *Construct the half adder as a schematic design. Refer Task 2. Rename all the ports correctly (A, B, S and C) and save the file in the current working directory (make sure to select* ***Add file to current project*** *in the* ***Save As*** *dialog box).*

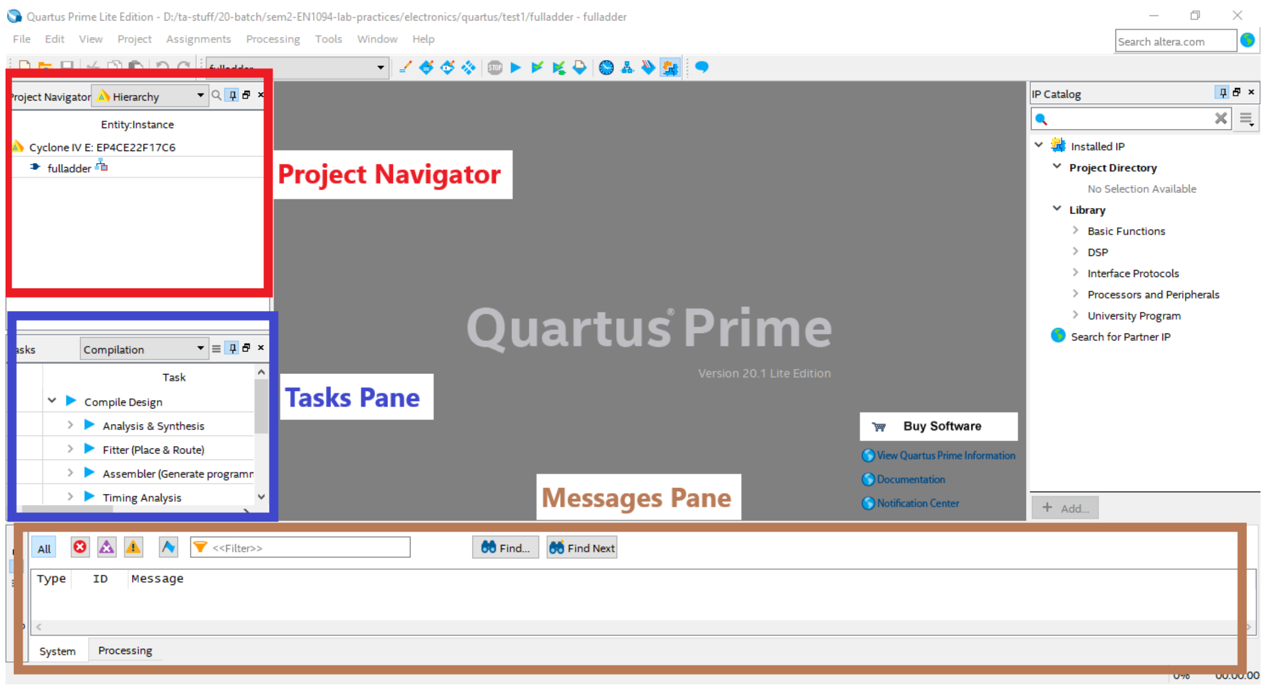


Figure 5.7: Project display. *Project Navigator* displays different files/modules of the project organized in different forms. *Tasks Pane* displays different tasks of the project flow and their status. *Messages Pane* displays messages corresponding to different tools and tasks.

Simulating the Half Adder

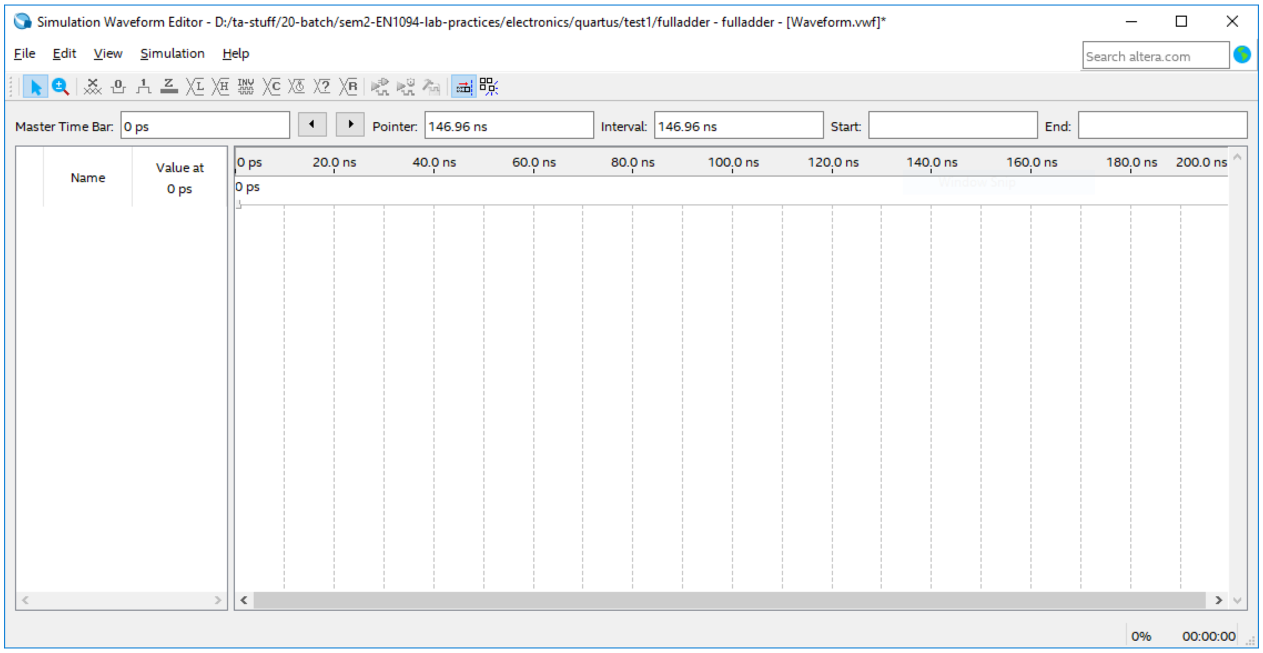
Prior to simulating or programming a design into a Field Programmable Gate Array (FPGA), the design must be compiled. In order to do so, we have to first specify the outer-most module of the design. Hence, before simulating the half adder created above, we have to first specify it as the top level entity of this project and then compile. To do so, follow the instructions given below;

1. In **Project Navigator** pane, select **Files** from the top-most drop down list. This will display all the files included in the project.
2. Right click on the half adder file and select **Set as Top-Level Entity**.
3. Compile the project by selecting **Processing > Start Compilation** from the menu bar.

Messages related to the compilation process will appear on the **Messages** pane. A **Compilation Report** will be displayed in the end. If the last message on the **Messages** pane informs that the compilation has been successful, you can proceed to simulation. Otherwise, scroll the pane to find out the error. Double clicking on an error will indicate its origin.

Simulation Waveform Editor will be used as the simulation tool. It can provide different waveforms specified by the user to the input ports of a compiled module and display the output waveforms. It will invoke the *ModelSim* simulation tool to generate the waveforms. To simulate the half adder, follow the instructions given below;

1. Click **File > New** from menu bar and add a new **University Program VWF** file listed under the **Verification/Debugging Files**. It will open up the **Simulation Waveform Editor** window (see Figure 5.8).
2. Set the simulation duration to 80ns by clicking **Edit > Set End Time...**.
3. Click **Edit > Insert > Insert Node or Bus...**.


Simulation Waveform Editor - D:/ta-stuff/20-batch/sem2-EN1094-lab-practices/electronics/quartus/test1/fulladder - fulladder - [Waveform.vwf]*

Master Time Bar: 0 ps Pointer: 146.96 ns Interval: 146.96 ns Start: End:

Figure 5.8: Simulation Waveform Editor

4. From the opened dialog box, click on **Node Finder...** button. It will open up the **Node Finder** dialog box.

5. In the **Node Finder**, set the **Filter:** to **Pins: all** from the drop down list and press **List**. A list of all the input/output nodes will be shown on the **Nodes Found:** pane (on the left hand side).

6. Add all the input/output nodes of the half adder to the **Selected Nodes:** pane (on the right hand side) by clicking on the **>** or **>>** buttons in-between the two panes.

7. Once all the nodes are included in the **Selected Nodes:** pane, click **OK**. This will insert a waveform for each node into the waveform editor.

Since we have made some changes, we can now save our waveform file by clicking **File > Save**. As the next step, we need to specify a waveform for each input node. To do this, follow the instructions given below;

1. On the waveform editor, click on an empty area within the time slot that you need to specify the value for. While pressing the mouse button, move the mouse so that the entire region where you need to specify the value is covered by the selection square (blue colour). See Figure 5.9.

2. After selecting the required part of the wave, press or button from the toolbar to make the selected part logic 0 or 1 respectively.

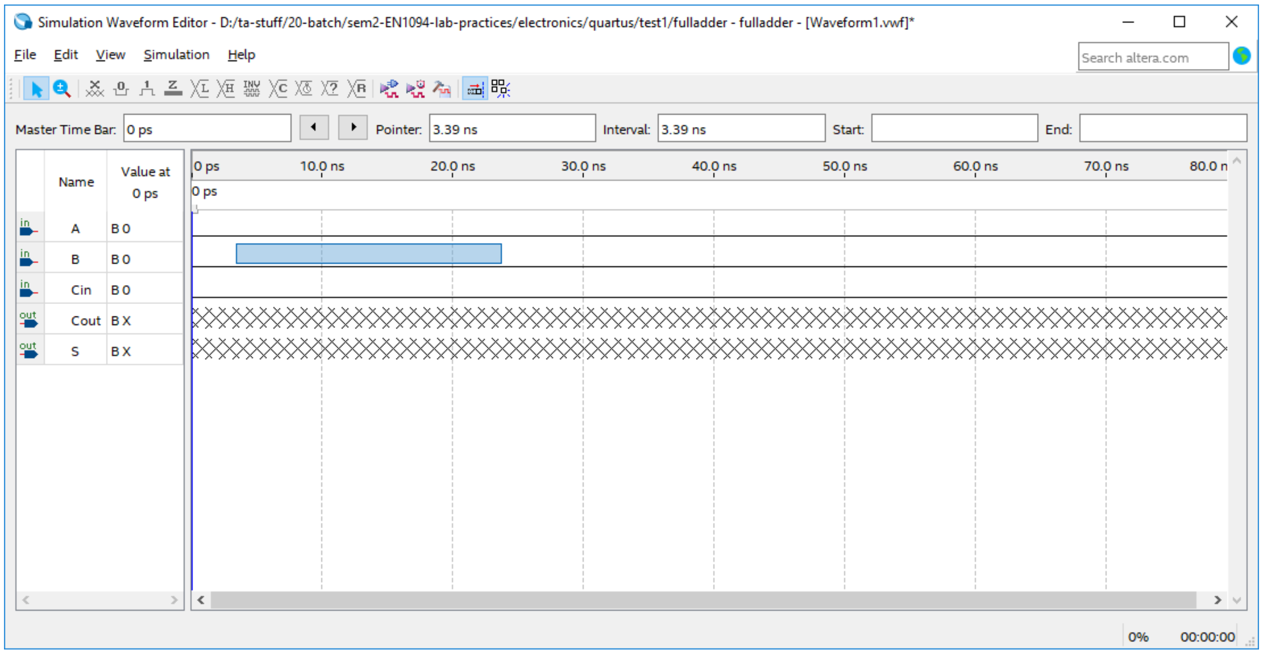
After all the input waveforms have been specified correctly, you can now run the simulation by following the instructions listed below;

1. Click **Simulation > Simulation Settings** on the menu bar. This will open up a settings window which includes scripts for running the simulation.

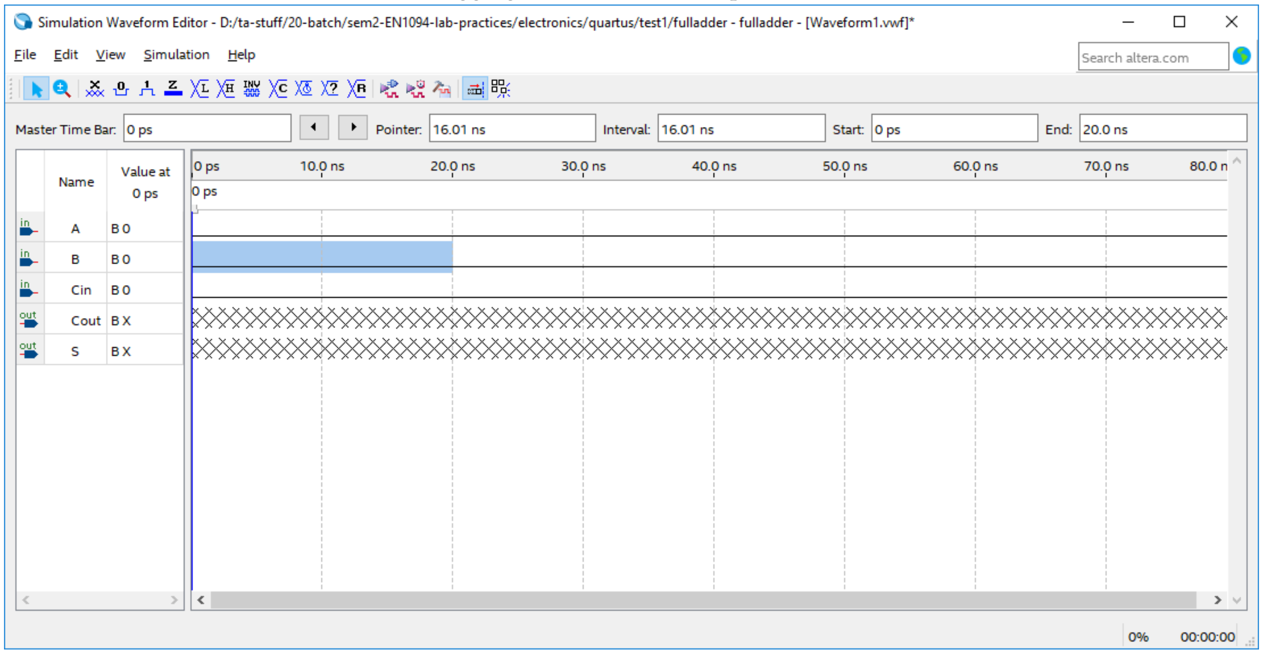
2. If not already selected, select **Verilog** as the **HDL Language**.

3. On the **Functional Simulation Settings** tab, locate the line **"vsim -novopt -c ....."** inside the **ModelSim Script (Functional Simulation)** pane. See Figure 5.10.

4. Carefully delete the **-novopt** command from the line. Keep everything else unmodified. If you made a mistake, press **Restore Defaults** button on the bottom of the window to reset.


Simulation Waveform Editor - D:/ta-stuff/20-batch/sem2-EN1094-lab-practices/electronics/quartus/test1/fulladder - fulladder - [Waveform1.vwf]*
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 3.39 ns
Interval: 3.39 ns
Start:
End:
Name
Value at 0 ps
A B 0
B B 0
Cin B 0
Cout B X
S B X
0 ps 10.0 ns 20.0 ns 30.0 ns 40.0 ns 50.0 ns 60.0 ns 70.0 ns 80.0 n
0%
00:00:00

(a) Dragging the mouse to select part of a wave


Simulation Waveform Editor - D:/ta-stuff/20-batch/sem2-EN1094-lab-practices/electronics/quartus/test1/fulladder - fulladder - [Waveform1.vwf]*
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 16.01 ns
Interval: 16.01 ns
Start: 0 ps
End: 20.0 ns
Name
Value at 0 ps
A B 0
B B 0
Cin B 0
Cout B X
S B X
0 ps 10.0 ns 20.0 ns 30.0 ns 40.0 ns 50.0 ns 60.0 ns 70.0 ns 80.0 n
0%
00:00:00

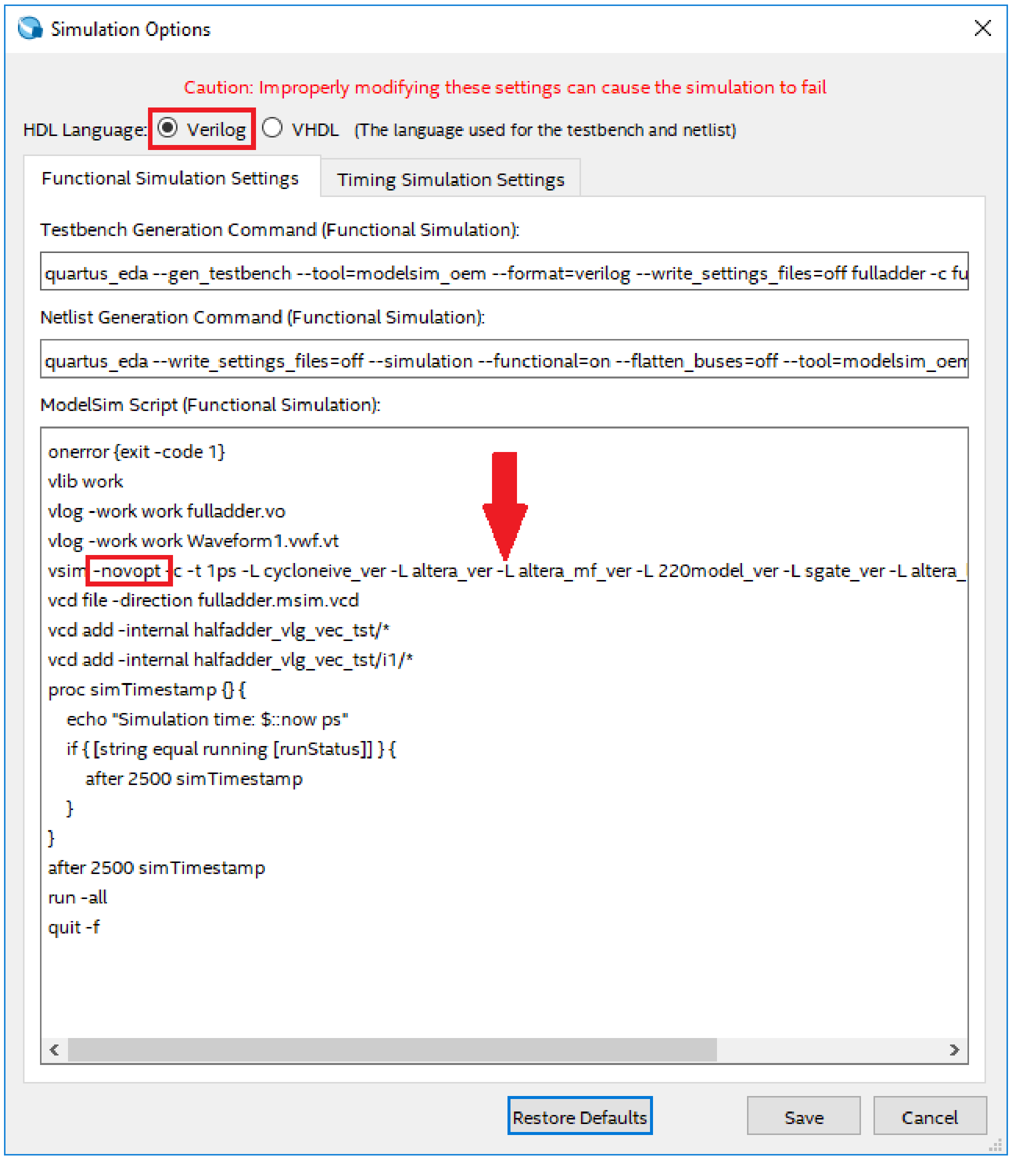
(b) Selected part of the wave after releasing the mouse button

Figure 5.9: Selecting part of a wave to specify the value

5. Press **Save** button to save settings and close the window.

6. To run the simulation, click **Simulation > Run Functional Simulation** from the menu bar. It will open up a new window displaying the status of different sub-tasks required for the simulation. In the end, if there are no errors, the output waveforms will be displayed on the **Simulation Waveform Editor**. An example is given in Figure 5.11.

Task 7. *Simulate the half adder constructed under Task 6. Verify the functionality against Table 5.1.*



Caution: Improperly modifying these settings can cause the simulation to fail

HDL Language: [✓ Verilog] VHDL (The language used for the testbench and netlist)

Testbench Generation Command (Functional Simulation):

```
quartus_eda --gen_testbench --tool=modelsim_oem --format=verilog --write_settings_files=off fulladder -c fu
```

Netlist Generation Command (Functional Simulation):

```
quartus_eda --write_settings_files=off --simulation --functional=on --flatten_buses=off --tool=modelsim_oem
```

ModelSim Script (Functional Simulation):

```
onerror {exit -code 1}
vlib work
vlog -work work fulladder.vo
vlog -work work Waveform1.vwf.vt
vsim -novopt -c -t 1ps -L cycloneive_ver -L altera_ver -L altera_mf_ver -L 220model_ver -L sgate_ver -L altera_
vcd file -direction fulladder.msim.vcd
vcd add -internal halfadder_vlg_vec_tst/*
vcd add -internal halfadder_vlg_vec_tst/i1/*
proc simTimestamp {} {
  echo "Simulation time: $::now ps"
  if { [string equal running [runStatus]] } {
    after 2500 simTimestamp
  }
}
after 2500 simTimestamp
run -all
quit -f
```

Figure 5.10: Simulation Settings

Constructing and Simulating the Full Adder

Quartus facilitates *hierarchical* designing. In other words, we can use previously designed modules as components of a new design. Consequently, we may use the already constructed half adder to create the full adder under this section. In order to use the half adder as a component of another design, we have to first create a symbol for it (note that this is only required in case of schematic designs, not for HDL designs). Follow the below steps to create a symbol for the half adder.

1. Open the half adder file by double clicking on it in the **Project Navigator**.

2. Click **File > Create / Update > Create Symbol Files for Current File**. This will open up **Create Symbol File** dialog box.

3. Give an appropriate name and save the symbol file inside the current working directory. Make sure that the file type is **Symbol File (*.bsf)**.

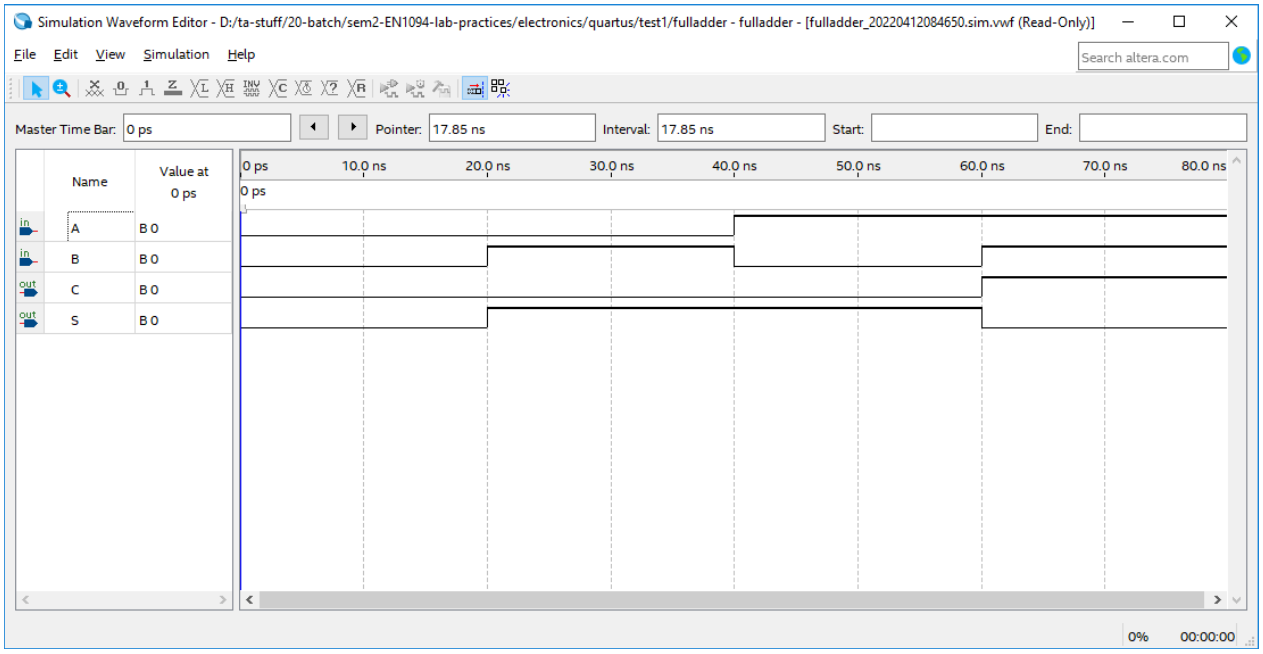

Simulation Waveform Editor - D:/ta-stuff/20-batch/sem2-EN1094-lab-practices/electronics/quartus/test1/fulladder - fulladder - [fulladder_20220412084650.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 17.85 ns
Interval: 17.85 ns
Start:
End:
Name
Value at 0 ps
0 ps
10.0 ns
20.0 ns
30.0 ns
40.0 ns
50.0 ns
60.0 ns
70.0 ns
80.0 ns
A B 0
B B 0
C B 0
S B 0
0%
00:00:00

Figure 5.11: Simulation result

Create a **Block Diagram/Schematic File** for the new top-level design. Make sure to add the file to the project. To insert instances of the half adder, follow the steps listed below;

1. Click on icon to open the **Symbol** window.
2. Click on the button with dots, located next to the text box under the title **Name:** to browse for new symbol files. See Figure 5.12.

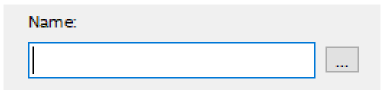


Figure 5.12: Browse button to locate new symbols

3. Locate the **.bsf** file corresponding to the half adder.
4. Now you can place it on the canvas and use it in a similar manner to an ordinary component.

Task 8. *Based on the results of Task 5, create a full adder using half adder blocks created under Task 6. Compile and simulate the design and verify the functionality with respect to Table 5.2. When compiling, make sure that the full adder file has been selected as the* ***Top-Level Entity****.*

5.3 Constructing a Full Adder Using Logic ICs

Task 9. *Identify the types and pinouts of the logic ICs using the datasheets provided.*

Task 10. *Construct two independent half adders on the breadboard using the logic ICs. Connect LEDs to the inputs (yellow) and the outputs (red) of each half adder as shown in Figure 5.13 to observe the logic levels easily. Verify the functionality with respect to Table 5.1.*

Task 11. *Connect the two half adders to construct a full adder according to the results of Task 5 and verify the functionality against Table 5.2.*

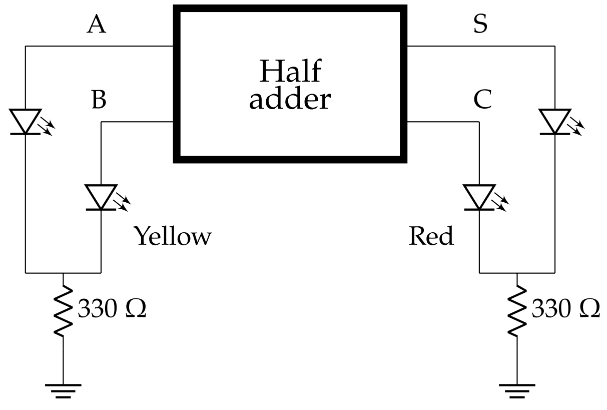


Figure 5.13: Inputs and outputs of half adder

5.4 Constructing a Full Adder Using a Full Adder IC

7483 is a commercially available full adder IC (see Figure 5.14). It can be used to add two 4-bit numbers and a carry-in bit. The result is provided as a 4-bit number, along with the carry-out. Therefore, the IC can be repeatedly connected to construct adders with a higher word length.

Task 12. *Identify the pinout of the 7483 full adder IC using the datasheet.*

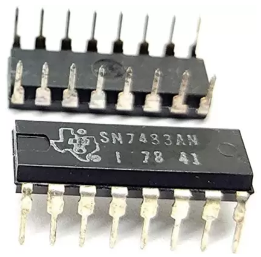


Figure 5.14: 7483 full adder IC

Task 13. *Connect LEDs along with current limiting resistors (330 Ω) to the outputs of the full adder IC (including the carry-out). Make sure that the carry-in is at logic zero. Perform the operations mentioned in Table 5.3 and record your observations.*

Operation	First input (A)				Second input (B)				Output (darken lit LEDs)
	A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	
6+2									○ ○ ○ ○ ○ C_o S_3 S_2 S_1 S_0
9+8									○ ○ ○ ○ ○ C_o S_3 S_2 S_1 S_0

Table 5.3: Observations: the full adder

Task 14. *An adder-subtractor can perform both addition and subtraction by representing signed numbers in their two's complement notation. Modify the inputs of the full adder IC as shown in Figure 5.15 to construct an adder-subtractor.*

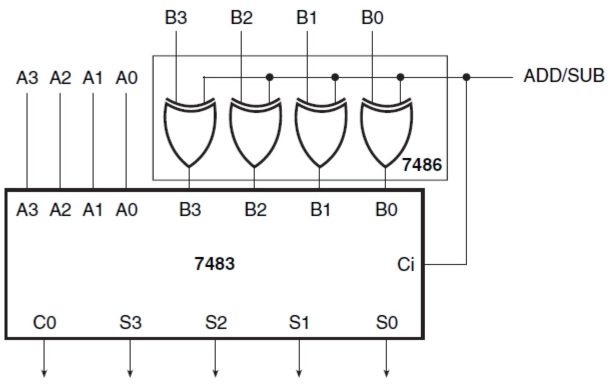


Figure 5.15: Adder-subtractor schematic

Task 15. *Perform the operations mentioned in Table 5.4 and record your observations.*

Operation	First input (A)				Second input (B)				Add/Sub	Output
	A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	C_i	(darken lit LEDs)
15-3										○ ○ ○ ○ ○ C_o S_3 S_2 S_1 S_0
8-13										○ ○ ○ ○ ○ C_o S_3 S_2 S_1 S_0

Table 5.4: Observations: the adder-subtractor

♣ The End ♣

Part IV

Telecommunication

Workshop 5: Point-to-Point Communication

Objective: To build and test a point-to-point communication system.

Outcome: After successfully completion of this session, the student would be able to

1. Identify the basic elements of a point-to-point communication system
2. Implementing a point-to-point communication system using 315/433 transmitter-receiver modules
3. Analyzing the packet-error rate of a point-to-point communication link

Equipment Required:

1. A Personal Computer Installed with Arduino software and RadioHead library
2. Two Arduino UNO Boards

Components Required:

315/433MHz transmitter-receiver modules

Two copper cables of length 17cm

Jumper wires

5.1 Point-to-Point Communication System

A point-to-point communication link connects a transmitter to a single receiver. The communication between an aircraft and a control tower is an example of a point-to-point communication link. In contrast, in point-to-multi-point (or a broadcast) communication, the transmitter can be heard by multiple receivers. A radio or TV broadcast system is an example. In this lab, we will be implementing a point-to-point communication link using 315/430 MHz transmitter-receiver modules. The link will transmit data in the form of packets.

5.2 Pre-Lab

In the pre-lab you will understand the setup for the implementation. You will also install and configure the necessary software.

5.2.1 Hardware Setup

We will be using 315/433MHz transmitter-receiver modules for communications. The module has a transmitter and a receiver. Both the modules are controlled by two separate Arduino UNO boards. We use the RadioHead Library to handle the communication. Figure 5.1 illustrates a block diagram of the communication system.

315/433 MHz transmitter module

The transmitter module implements OOK (On-Off Keying). It mainly consists of three sections.

- SAW resonator generating a 433MHz
- Switching transistor
- Antenna

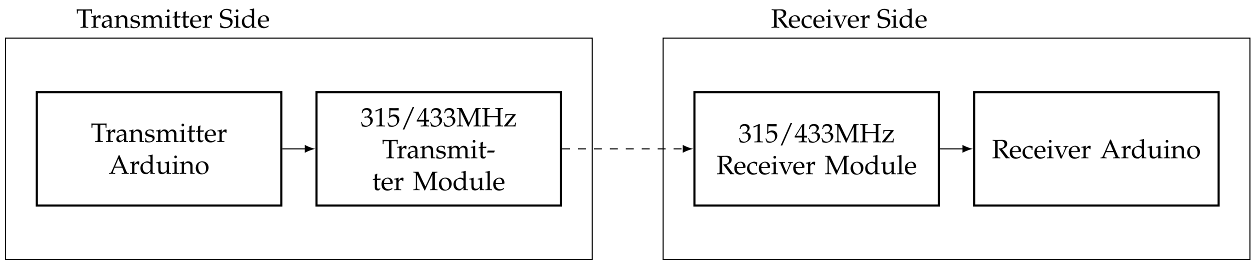


Figure 5.1: The block diagram of the point to point communication system we will be implementing

The working principle is simple. The resonator generates a continuous sinusoidal waveform. The switching transistor governs the connection between the antenna and the resonator. The binary data stream is connected to the the switching transistor. When binary 0 is given the switch is in off state and when binary 1 is given switch is in on state thereby providing a sinusoidal waveform to the antenna which is subsequently transmitted. The frequency of the waveform can be either 315 MHz or 433MHz depending on how the device is tuned. This technique is known as On-Off Keying (OOK), a basic digital modulation technique. The sinusoidal waveform is the carrier.

Figure 5.2 shows the transmitted waveform for the bit stream "111010011" using this technique. The Figure 5.3 shows the connections and the components of the transmitter module.

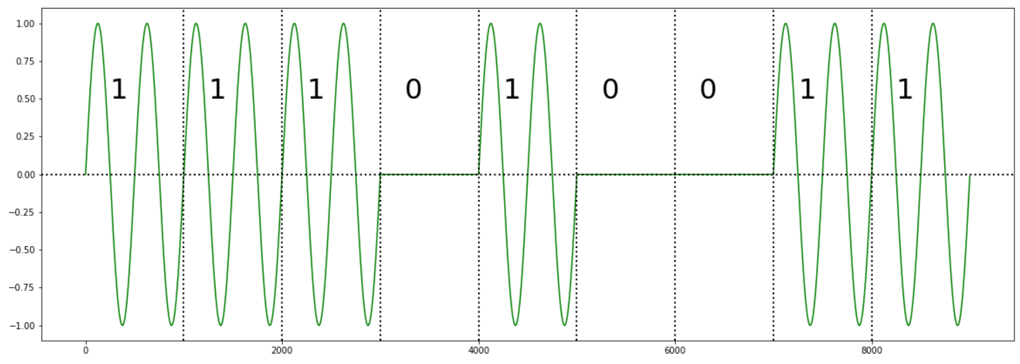


Figure 5.2: OOK waveform for the bit stream "111010011

5.2.2 315/433 MHz receiver module

The receiver module is a bit more complex than the transmitter but still has a simple implementation. The receiver contains four major parts

- Antenna
- RF Tuner Circuit
- Amplifier
- Phase-Locked Loop

The Antenna receives the RF signal transmitted by the transmitter. The RF Tuner Circuit is responsible for tuning the receiver to the frequency of the transmitted signal. The amplifier amplifies the transmitted signal. The Phase-Locked Loop is responsible for decoding the OOK signal and to generate the bit stream. Figure 5.4 shows the connections and the components of the receiver module.

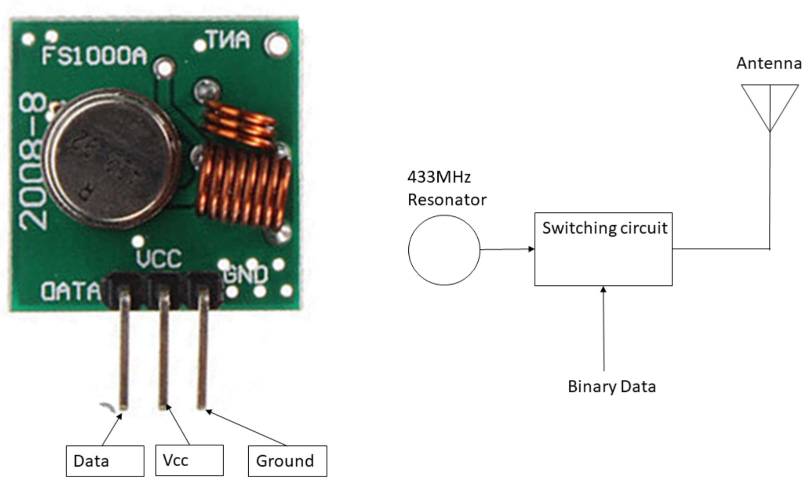


Figure 5.3: The transmitter module

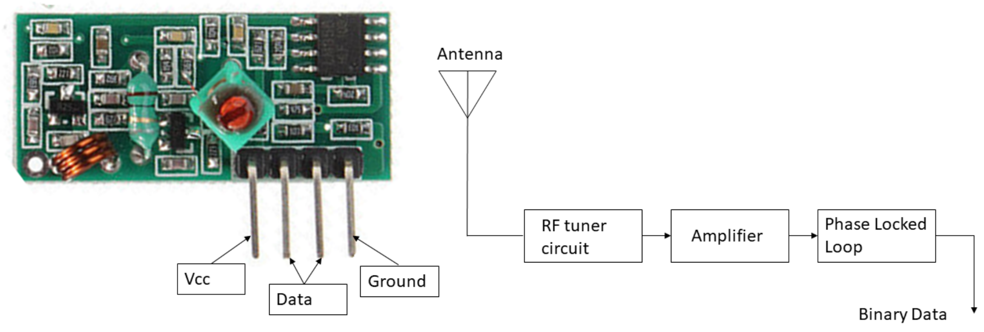


Figure 5.4: The receiver module

5.2.3 Setting Up the Arduino Software

Before moving onto the practical you will have to download the Arduino software. You can download the version compatible with your PC from `https://www.arduino.cc/en/software`. After downloading, install the software.

5.2.4 Setting Up the RadioHead Library

After downloading the Arduino IDE (Integrated Development Environment), you will have to add the RadioHead software. RadioHead provides many libraries which can be configured easily for different applications in wireless communications. If RadioHead is not already added to Arduino you can download the zip folder from `http://www.airspayce.com/mikem/arduino/RadioHead/RadioHead-1.121.zip`. After that you can add the zip library by **Sketch** → **Include library** → **Add .ZIP Library...**

In order to ensure reliable transmission of data, we transmit data in the form of packets. The RadioHead library is responsible for encapsulating data into packets. In simple terms it appends a training preamble, start symbol and a frame check sequence to the data. The composition of a RadioHead packet is depicted in Figure 5.5.

Training preamble: consists of 36 alternating 1's and 0's. Used by the receiver to adjust it's gain.

Start symbol: consists of 12 bits. These 12 bits indicate the receiver that a new data packet is arriving and it indicates when the actual data block will start.

Frame check sequence (FCS): These 16 bits are used by the receiver to check whether bit-errors have occurred.

Payload: Payload is the part which contains the actual data. In addition to the data it may contain the receiver address and the packet identification number. Receivers address is important when there are several transmitter-receiver pairs. In that case the receiver needs to know that the message is intended to it. Packet identification number is unique for each packet intended to a particular receiver. Depending on the size of the actual data block, number of bits in the payload may vary.

Training Preamble (36)	Start Symbol (12)	Payload (Variable)	FCS (16)

Figure 5.5: The RadioHead Packet

‘

5.3 Implementation of The Point to Point Communication System

Since the transmitter module implements OOK, we utilize the ASK (Amplitude Shift Keying) library of RadioHead. OOK is a special case of the more general digital modulation scheme ASK. Now we are ready to build the point to point communication link. First we will program the Arduino boards to send a message from the transmitter to the receiver.

5.3.1 The Transmitter Side

In our implementation we let the length of the payload section to be 6 bytes (48 bits). Since we are building a point-to-point communication link, the transmitted packets are intended only to a single receiver. First two bytes of the payload represents the receiver address. The instructor will provide you with the address for your group. The address is an integer between 0 and 99. For example if the address is 73, the first byte will represent the ASCII code for 7 and the second byte will represent the ASCII code for 3. The next four bytes will represent the packet ID which is a number from 0 to 1023 (Note that we are not transmitting actual data using these packets).

The following code will iteratively transmit packets with ID's ranging from 0 to 1023. Once it transmits a packet with ID 1023 it will start again from a packet with ID 0.

```
// Include RadioHead Amplitude Shift Keying Library
#include <RH_ASK.h>

// Include dependant SPI Library
#include <SPI.h>

// Create Amplitude Shift Keying Object
RH_ASK rf_driver;

// This array will store the six bytes representing the payload
char payload[6];

// Ask the instructor for the number of the receiver you
// are communicating with
int rec_num = 0;

// Stores the current id of the current transmitted packet

int current_packet = 0;

void calcRecNum(int receiver_number){
 // Calculates the two bytes representing the receiver_number
 int  p = receiver_number;
 int  i;
 for(i=0;i<2;i++){
   payload[1−i] = char((p%10)+int('0'));
   p = p/10;
 }
}

void calcId(int packet_ID){
 // Calculates the four bytes representing the packet ID
 int  p = packet_ID;
 int i;
 for(i=0;i<4;i++){
   payload[5−i] = char((p%10)+int('0'));
   p = p/10;
 }
}

void setup()
{
  // Initialize ASK Object
  rf_driver.init();
  // Set the receiver number
  calcRecNum(rec_num);
  Serial.begin(9600);
}



void loop()
{
  calcId(current_packet);
  rf_driver.send((uint8_t *)payload, strlen(payload));
  rf_driver.waitPacketSent();
  delay(100);

  // Incrementing the packet ID
  current_packet = current_packet + 1;
  if(current_packet == 1024){
   current_packet = 0;
  }

}
```

Connect the Arduino and the 315/433MHz transmitter module as shown in Figure 5.6. Enter the above code to a new Arduino file. Connect the Arduino to the PC. Using **Tools** → **Port** select the correct COM port to which the Arduino is connected. Click on the Upload button and wait until the uploading finishes. Now you can unplug the Arduino from the PC.

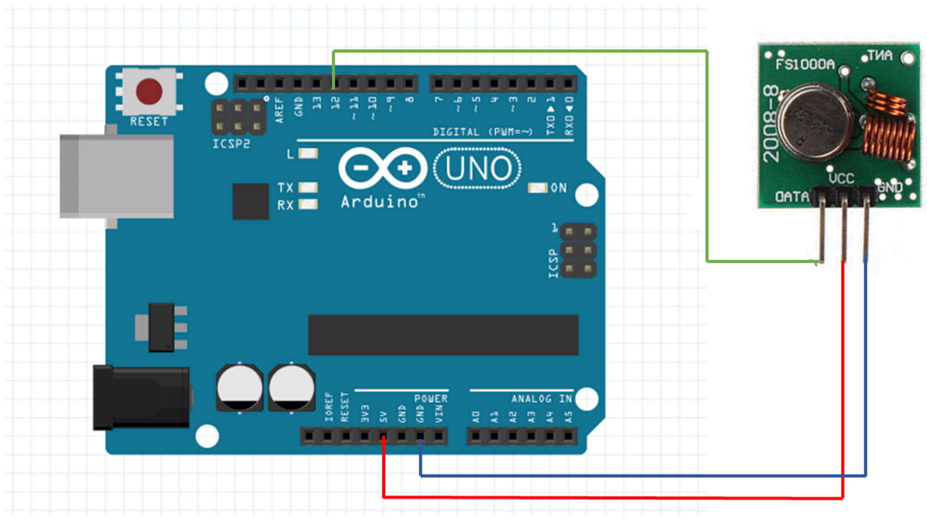


Figure 5.6: Configuration of the transmitter side

5.3.2 The Receiver Side

Next you will program the Arduino board for the receiver. When a packet arrives at the receiver, it checks whether the packet is valid by comparing the frame check sequence with the rest of the packet. If the packet in invalid (if bit errors have occurred during transmission) the packet is discarded. Then the receiver checks whether the packet is intended for it. If so the receiver calculates the ID of the packet which is subsequently printed to the serial monitor.

```
// Include RadioHead Amplitude Shift Keying Library
#include <RH_ASK.h>
// Include dependant SPI Library
#include <SPI.h>

// Create Amplitude Shift Keying Object
RH_ASK rf_driver;

const int sent_size = 1024;

//This array will store whether a packet with a particular ID is being received
bool rec[sent_size];

// Number of this receiver
int rec_num = 0;


// This array stores the received message
uint8_t buf[6];
uint8_t buflen;


bool checkRecNum(int receiver_number){
 //Checks whether the message is intended to the receiver
 int c = 0;
 int i;
 int pow10 = 1;
 for(i=0;i<2;i++){
     c = c + (pow10*(buf[1-i]-'0'));
     pow10 = pow10*10;
  }
  if(c == receiver_number){
    return true;
  }
  return false;

}



int calcPacID(){
//Calculates the packet ID
    int i;
    int pac_ID = 0;
    int pow10 = 1;
    for(i = 0;i<4;i++){
       pac_ID = pac_ID + (pow10*(buf[5-i]-'0'));
       pow10 = pow10*10;
    }
    return pac_ID;
}

void setup()
{
    // Initialize ASK Object
    rf_driver.init();
    // Setup Serial Monitor
    Serial.begin(9600);
}

void loop()
{

    buflen = sizeof(buf);
    // Check if received packet is correct size
    if (rf_driver.recv(buf, &buflen))
    { // Message received with valid checksum
      if(checkRecNum(rec_num)){
        // Message is intended to the receiver
        //Calculating the received packet ID
        int pac_ID = calcPacID();
        rec[pac_ID]=1;

        Serial.print("Packet Received: ");
        Serial.println(pac_ID);
      }
    }

}
```

Connect the Arduino and the 315/433MHz receiver module as shown in Figure 5.7. Similar to the transmitter side Arduino, upload the code to the Arduino board for the receiver. Do not disconnect the Arduino from the PC. Now connect the transmitter side Arduino to a power source. Navigate to **Tools** → **Serial Monitor**. The outputs should appear on the serial monitor (Make sure the correct COM port is selected).

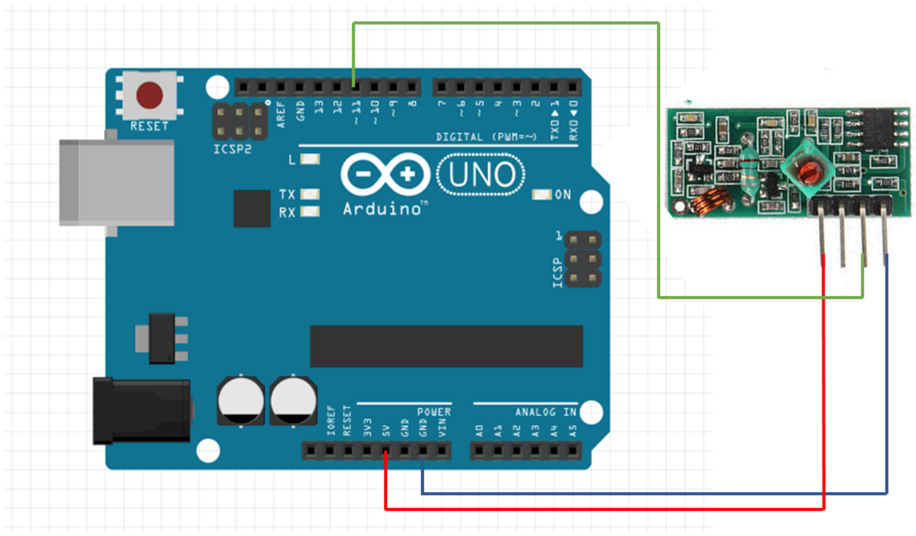


Figure 5.7: Configuration of the receiver side

Task 1. *Increase the distance between the transmitter and the receiver and identify the minimum distance beyond which packets are no longer received.*

5.4 Analyzing the Packet Error Rate of a Point to Point Communication System

Now we modify our code to send each packet only once (The packets with ID's from 0 to 1023 are sent only once). At the receiver we check how many of the transmitted packets are correctly received. Then the packet error rate is calculated as,

$$\text{Packet Error Rate} = 1 - \frac{\text{Number of Packets Correctly Received}}{1024} \tag{5.1}$$

Use the following code for the transmitter.

```
// Include RadioHead Amplitude Shift Keying Library
#include <RH_ASK.h>

// Include dependant SPI Library
#include <SPI.h>

// Create Amplitude Shift Keying Object
RH_ASK rf_driver;

// This array will store the six bytes representing the payload
char payload[6];

// Ask the instructor for the number of the receiver you
// are communicating with
int rec_num = 0;

// Stores the current id of the current transmitted packet

int current_packet = 0;
```

```
void calcRecNum(int receiver_number){
 // Calculates the two bytes representing the receiver_number
 int  p = receiver_number;
 int  i;
 for(i=0;i<2;i++){
    payload[1−i] = char((p%10)+int('0'));
    p = p/10;
 }
}

void calcId(int packet_ID){
 // Calculates the four bytes representing the packet ID
 int  p = packet_ID;
 int i;
 for(i=0;i<4;i++){
    payload[5−i] = char((p%10)+int('0'));
    p = p/10;
 }
}

void setup()
{
   // Initialize ASK Object
   rf_driver.init();
   // Set the receiver number
   calcRecNum(rec_num);
   Serial.begin(9600);
}



void loop()
{

   if(current_packet<1024){
     calcId(current_packet);
     rf_driver.send((uint8_t *)payload, strlen(payload));
     rf_driver.waitPacketSent();
     delay(100);
     // Incrementing the packet ID
     current_packet = current_packet + 1;
   }
}
```

Upload the following code to the receiver side Arduino.

```
// Include RadioHead Amplitude Shift Keying Library
#include <RH_ASK.h>
// Include dependant SPI Library
#include <SPI.h>

// Create Amplitude Shift Keying Object
RH_ASK rf_driver;

const int sent_size = 1024;

//This array will store whether a packet with a particular ID is being received
bool rec[sent_size];
```

```
// Number of this receiver
int rec_num = 0;


// This array stores the received message
uint8_t buf[6];


//Control paremeters to check whether the packet receiving has finished
long int last_time = 0;
long int cur_time = 0;
int printed = 0;
uint8_t buflen;
int cur_pac = 0;
long int time_per_pac = 225;

bool checkRecNum(int receiver_number){
 //Checks whether the message is intended to the receiver
 int c = 0;
 int i;
 int pow10 = 1;
 for(i=0;i<2;i++){
     c = c + (pow10*(buf[1-i]-'0'));
     pow10 = pow10*10;
 }
 if(c == receiver_number){
   return true;
 }
 return false;

}



int calcPacID(){
//Calculates the packet ID
   int i;
   int pac_ID = 0;
   int pow10 = 1;
   for(i = 0;i<4;i++){
      pac_ID = pac_ID + (pow10*(buf[5-i]-'0'));
      pow10 = pow10*10;
   }
   return pac_ID;
}

void setup()
{
   // Initialize ASK Object
   rf_driver.init();
   // Setup Serial Monitor
   Serial.begin(9600);
}

void loop()
{

   buflen = sizeof(buf);
```

```
  // Check if received packet is correct size
  if (rf_driver.recv(buf, &buflen))
  { // Message received with valid checksum
   if(checkRecNum(rec_num) && printed == 0){
     // Message is intended to the receiver

     //Calculating the received packet ID
     int pac_ID = calcPacID();
     rec[pac_ID]=1;
     cur_pac = pac_ID;
     Serial.print("Packet Received: ");
     Serial.println(pac_ID);
     last_time = millis();
   }
  }
  else{
   // If no valid packet is received for within the remaining
   // packet receiving has finished
   cur_time = millis();
   if(cur_time - last_time > time_per_pac*(sent_size-cur_pac)){
     if(printed ==0){
       int sum = 0;
       int i;
       for(i = 0;i<1024 ;i++){
        sum=sum+rec[i];
       }
      Serial.print("Packet Receiving Finished. Packet Error Rate: ");
      Serial.println(((float) (sent_size-sum)/(float) 1024));
      printed = 1;
     }
   }
  }
}
```

To ensure proper operation, make sure you power the transmitter side Arduino after powering the receiver side Arduino. Monitor the communication using the serial monitor as mentioned in section 5.3.2. Wait for the communication to complete. The packet error rate will appear on the serial monitor afterwards.

Task 2. *Calculate the PER for 5 different distances. Take 5 distances to cover the range between 0 and the distance obtained in task 1. For each distance repeat do the experiment for 5 iterations and obtain the average PER. Record the results in the table in the Task Sheet.*

Task 3. *Discuss the impact of the distance between the transmitter and the receiver antennas on reliable communication.*

Task 4. *Repeat task 2 with 17cm antennas fixed to the transmitter and the receiver and record the results in the table in the Task Sheet.*

Task 5. *Discuss the impact of adding an antenna.*

♣ The End ♣